Application-Motivated, Holistic Benchmarking of a Full Quantum Computing Stack

Daniel Mills∗ 1,2, Seyon Sivarajah2, Travis L. Scholten3, and Ross Duncan2,4

1University of Edinburgh, 10 Crichton Street, Edinburgh EH8 9AB, UK
2Cambridge Quantum Computing Ltd, 9a Bridge Street, Cambridge, CB2 1UB, UK
3IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA
4University of Strathclyde, 26 Richmond Street, Glasgow, G1 1XH, UK

June 3, 2020

Abstract

Quantum computing systems need to be benchmarked in terms of practical tasks they would be expected to do. Here, we propose 3 “application-motivated” circuit classes for benchmarking: deep (relevant for state preparation in the variational quantum eigensolver algorithm), shallow (inspired by IQP-type circuits that might be useful for near-term quantum machine learning), and square (inspired by the quantum volume benchmark). We quantify the performance of a quantum computing system in running circuits from these classes using several figures of merit, all of which require exponential classical computing resources and a polynomial number of classical samples (bitstrings) from the system. We study how performance varies with the compilation strategy used and the device on which the circuit is run. Using systems made available by IBM Quantum, we examine their performance, showing that noise-aware compilation strategies may be beneficial, and that device connectivity and noise levels play a crucial role in the performance of the system according to our benchmarks.

∗Corresponding author: daniel.mills@cambridgequantum.com

Contents

1 Introduction 2
2 Circuit Classes 4
  2.1 Shallow Circuits: IQP . . . . . . . . 4
  2.2 Square Circuits: Random Circuit Sampling . . . . . . . . . . . . . . 5
  2.3 Deep Circuits: Pauli Gadgets . . . . . . . . 7
3 Figures of Merit 7
  3.1 Heavy Output Generation Benchmarking . . . . . . . . . . . . . . . 8
  3.2 Cross-Entropy Difference . . . . . . . . 10
  3.3 ℓ1-Norm Distance . . . . . . . . . . 11
  3.4 Metric Comparison . . . . . . . . . . 12
4 Quantum Computing Stack 12
  4.1 Software Development Kits . . . . . . 12
  4.2 Compilers . . . . . . . . . . . . . . 13
  4.3 Devices . . . . . . . . . . . . . . . 13
5 Experimental Results 14
  5.1 Full Stack Benchmarking . . . . . . 16
  5.2 Application Motivated Benchmarks . 22
  5.3 Insights from Classical Simulation . 24
6 Conclusion 25

A Exponential Distribution 35
  A.1 Square Circuits . . . . . . . . . . . 35
  A.2 Deep Circuits . . . . . . . . . . . 35
1 Introduction

As quantum computers evolve from bespoke laboratory experiments comprising a handful of qubits, to more general-purpose, programmable, commercial-grade systems [1–5], new techniques for characterizing them are needed. Quantum characterization, validation, and verification (QCVV) protocols to detect, diagnose, and quantify errors in quantum computers, originally focused on properties of one or several qubits (e.g., $T_1$ and $T_2$ times, gate error rates, state preparation fidelity, etc). As multi-qubit quantum computing systems develop, the scope of QCVV must expand. In particular, a need has arisen for “holistic” benchmarks - ones which stress test a quantum computing system in its entirety, not just individual components. Holistic benchmarks are desirable for two reasons: they enable comparison across different systems, and allow for tracking the performance of a fixed system over time.

“Holistic benchmarking” of a quantum computing system could refer to benchmarking the physical implementation of a collection of qubits, without referring to the computational task these qubits would perform. This idea is most useful when testing physical properties of a collection of qubits\(^2\). The complementary view (taken in this work) is that holistic benchmarks test the quantum computational capabilities of the complete system. Under this view, the entire compute stack – qubits, compilation strategy, classical control hardware, etc. – should be benchmarked collectively. Such “full-stack” benchmarking provides information that benchmarking individual stack components cannot, as it captures the performance of the system as an integrated unit.

At the same time, running a full-stack benchmark on a fixed computational system, while useful for tracking the performance of that system over time, provides little information on how different combinations of the stack’s components could change system performance. For this reason, full-stack benchmarking should, as much as possible, make explicit the variable components of the stack, and systematically vary those components to see how the inclusion of a particular component affects system-level performance.

Here, we will focus on benchmarking systems made available by IBM Quantum, and investigate two components of the stack: the compilation strategy used to map an abstract circuit onto one that is executable on a quantum computer and the device used to run the compiled circuit and return the results. While the particular systems used here have other components (such as pulse synthesizers), we do not look at the impact of those pieces on full-stack performance.

The design of new compilers for quantum circuits is an active area of research, especially “noise-aware” compilation strategies which use knowledge of the physical properties of the system’s qubits to improve results [1, 9–11]. The proliferation of compilers necessitates understanding how the inclusion of particular compilation strategies in the stack affects performance. Problem instances requiring compilation, which are often more representative of real world problems, typically show differing performance from those that do not [12]. In particular, noise-aware compilation strategies make assumptions about the influence of noise processes on overall system performance, so full-stack benchmarking is necessary to verify those assumptions.

The benchmarks defined here have two parts: a circuit class and a figure of merit. The circuit class describes the type of circuit to be run by the system, and the figure of merit quantifies how well the system did when running circuits from that class. This approach is inspired by volumetric benchmarking [13].

Because quantum computing systems are used for particular applications, the circuits classes

\(^1\)This should be compared to the benchmarking of classical computers, with the LINPACK benchmarks [6, 7] being used to build the TOP500 ranking of supercomputers [8].

\(^2\)A simple example is crosstalk detection, where the output of the benchmarking could be a table of coupling values between all connected qubits.
should, in some way, test the performance of a system in those arenas [14]. At least two notions have been put forth as to how to define such classes. One proposes benchmarks based on often-used quantum algorithmic primitives [13], the examples given being primitives of Grover iterations and Trotterized Hamiltonian simulation.

An alternative is to pick a particular instance of an application and check for the accuracy of the results returned by the system when running that instance. Naturally, to measure non-negligible accuracy on noisy near-term systems the applications and instances must also be near-term by design. Such benchmarks have been defined in the context of quantum simulation [15–19], quantum machine learning [14, 20–23], discrete optimisation [12, 24–26], and quantum computational supremacy [3, 27–29]. This approach has the advantage that the definition of success is fairly straightforward. The downside is that performance as measured by one instance of an application may not be predictive of performance for the application generically.

The “application-motivated” circuit classes defined here draw inspiration from [13] (looking at computational primitives) but also draw inspiration from the literature above, by focusing on computational primitives of near-term quantum computing applications (chemistry and machine learning, in particular). A system which does well on an application-motivated benchmark should do well in running the application the benchmark was derived from. Three such “application-motivated” circuit classes are introduced here. Drawing inspiration from the volumetric benchmarking approach, the classes cover varying depth regimes and are (somewhat) controllable in depth. In brief, the classes – as labelled by their depth regimes – are:

**Deep:** Inspired by product formula circuits, including state preparation circuits used in the variational quantum eigensolver (VQE) algorithm for quantum chemistry [30–32].

**Shallow:** Inspired by hardware-efficient ansatze [33, 34] which may be useful for near-term quantum machine learning and chemistry applications [35–37].

**Square:** Inspired by the circuits used to calculate a system’s quantum volume [38].

Section 2 provides details of these circuit classes, and presents algorithms for generating them.

How well a stack executes a circuit is assessed here via continuous figures of merit, rather than binary ones which may only verify correctness. This is because the outcomes from noisy devices will likely not be correct, while information about closeness to the correct answer is still highly valuable. Further, techniques for the verification of universal quantum computation requires many qubits or qubit communication or both, none of which are accessible using present-day noisy devices [39, 40]. Indeed, to reflect the current state-of-the-art, where there exist few devices with limited networking between them [4, 41], we will focus on examples of how classical computers can be used to perform benchmarks, as opposed to using small quantum computers to benchmark each other [42, 43].

We use three figures of merit, calculated using classical computers. These are: heavy output generation probability [44], cross-entropy difference [29], and $\ell_1$-norm distance. Estimating each of these figures of merit requires knowledge about the ideal (noise-free) outcome probabilities of bit-strings the system could produce.

In practice, calculating the ideal outcome probabilities requires direct simulation of the circuit under consideration. Consequently, scaling to tens or hundreds of qubits will be challenging in general, particularly if the $\ell_1$-norm distance is used as the figure of merit. However, by considering circuits with few qubits we allow ourselves the ability to simulate the circuits classically, and to gain an insight into the behaviour of larger devices [3, 45].

We refer to a set of benchmarks as a *benchmarking suite*, each benchmark being defined by unique combinations of each circuit class and figure of merit. Using a benchmarking suite enables the derivation of broad insights about the behaviour and performance of a quantum computing system across a wide variety of possible applications. Their varying demands on the quantum computing resources (qubits, depth) allows for the exploration of the best routes to extract the most utility from near-term quantum computers. In sum, our benchmarking approach is both application-motivated and holistic.

The remainder of this paper is comprised as follows: Section 2 details the circuit classes, including algorithms for generating the circuits; Section 3 ex-
explains the figures of merit we use; Section 4 introduces the software stack, as well as hardware made available by IBM Quantum, that comprise the systems we’ll be benchmarking; and Section 5 shows the results of our benchmarking. We conclude in Section 6.

2 Circuit Classes

This section presents the formal definitions of the circuits used in this work, while also identifying the motivations for their use in benchmarking. These motivations include both the class of applications they represent and the properties of the quantum computing stacks that they will probe. Collectively, this selection of circuit classes encompass an array of potential applications of quantum computing, covering circuits of varied depth, connectivity, and gate types.

2.1 Shallow Circuits: IQP

Instantaneous Quantum Polytime (IQP) circuits [46] can be implemented using commuting gates. As well as being simpler to implement than universal quantum circuits, there are strong theoretical reasons to believe that, even in the presence of noise, IQP circuits cannot be simulated using classical computers [47–49]. This has allowed for the application of noisy quantum technology in areas such as machine learning [35, 36] and interactive two-player games [43, 46]. The connection between IQP and a demonstration of quantum computational supremacy on near-term hardware makes their implementation a pertinent benchmark of the performance of these devices.

The shallow class of circuits, whose depth increases slowly with width, is a subclass of IQP circuits. These circuits probe the performance of a quantum computing stack in fine-grained detail by measuring the impact of including more qubits (quasi-) independently of increasing circuit depth. This is useful when for understanding the performance of a device being utilised for applications whose qubit requirement grows more quickly than the circuit depth.

Definitions and Related Results An n-qubit IQP circuit consists of gates that are diagonal in the Pauli-X basis, acting on the $|0\rangle^n$ state, with measurement taking place in the computational basis. For this class of circuits, Theorem 1 applies.

**Theorem 1 (Informal [48])** Assuming either one of two conjectures, relating to the hardness of approximating the Ising partition function and the gap of degree 3 polynomials, and the stability of the Polynomial Hierarchy\(^3\), it is impossible to classically sample from the output probability distribution of any IQP circuit in polynomial time, up to an $\ell_1$-norm distance of $1/192$.

This class is called “instantaneous” because these gates commute with one another, which in turn reduces the amount of time that the quantum state will need to be stored. In addition, the impossibility of simulating IQP circuits is shown to hold when restricted by physically motivated constraints such as limited connectivity and constant error rates on each qubit [49].

An equivalent, commonly-considered definition is that IQP circuits consist of gates diagonal in the Pauli-Z basis, sandwiched between two layers of Hadamard gates acting on all qubits. Algorithm 1 is used to generate IQP circuits of this form. Note that Algorithm 1 limits the connectivity allowed between the qubits, so it does not generate all circuits in the IQP class.

The depth of this circuit may be arrived at by observing that finding an optimal order of application of $CZ$ is equivalent to finding a edge colouring of the graph $G_n$. In this case a 4-colouring can be found in polynomial time [51]. Algorithm 1 includes discrete randomness over the graphs, $G_n$, and continuous randomness over the rotation angles, $\alpha_i$.

The design of circuits in Algorithm 1 may be compared to other sparse IQP circuits [49], IQP circuits on 2D lattices [49, 52], and random 3-regular graphs used for benchmarking [12]. For our purposes these require too high-connectivity, are too architecture-specific, and are too application-specific, respectively. There are sparse IQP circuits for which verification schemes exist [52, 53] although the connectivity is too architecture-specific for our purposes, with the verification scheme requiring limits to the measurement noise which we cannot guarantee.

\(^3\)The non-collapse of the Polynomial Hierarchy is widely conjectured to be true [50].
Algorithm 1 The pattern for building shallow circuits.

**Input:** Number of qubits, \( n \in \mathbb{Z} \\
Worst \ case \ depth: \ 7 \\
Output: \ Circuit, \ C_n

1: Initialise \( n \) qubits, labelled \( q_1, \ldots, q_n \), in the state \( |0\rangle \).
2:  
3: for all \( i \in \{1, \ldots, n\} \) do
4:     Act \( H \) on \( q_i \)
5: end for
6:  
7: Generate a random binomial graph, \( G_n \), with \( n \) vertices and edge probability 0.5, post selecting on those that are connected and have degree less than 4.
8:  
9: for all edges \( \{i, j\} \) in \( G_n \) do
10:    Act \( CZ \) between \( q_i \) and \( q_j \)
11: end for
12: 
13: for all \( i \in \{1, \ldots, n\} \) do
14:    Generate \( \alpha_i \in [0, 2\pi] \) uniformly at random.
15:    Act \( RZ(\alpha_i) \) on \( q_i \)
16: end for
17: 
18: for all \( i \in \{1, \ldots, n\} \) do
19:    Act \( H \) on \( q_i \)
20: end for
21:  
22: Measure \( q_1, \ldots, q_n \) in the computational basis

**Discussion** The close connection, through Theorem 1, of quantum computational supremacy and shallow circuits

\(^4\) explicitly measured in \( \ell_1 \)-norm distance, provides a measure of a quantum computing stack’s quality; namely, by analysing the closeness of the distributions it produces to the ideal ones, as measured by the \( \ell_1 \)-norm distance, and comparing this value to 1/192.

However, as the output probabilities of shallow circuits are not exponentially distributed, we cannot use Cross-Entropy Benchmarking. Similarly the theoretical value of heavy output probability for circuits with exponentially distributed output probabilities, as discussed in Section 3.1, cannot be used here.

Instead, we use the empirical value of the ideal heavy output probability, in the place of a theoretically derived one, as a point of comparison with the behaviour of the quantum computing stack being benchmarked. This approach requires calculation of all output probabilities and summation of the probabilities of those that are heavy. This can be done for the small circuits investigated here, but allows for the benchmarking of fewer qubits than would be accessible if a theoretical value was known.

Before compilation shallow circuits have constant depth, allowing us to measure the impact of increasing circuit width independently of increasing circuit depth. Further, because Algorithm 1 limits the connectivity allowed between the qubits, the increase in circuit depth due to compilation onto limited-connectivity architectures is also minimised, while avoiding a choice of connectivity favouring one device in particular. By bounding connectivity, but allowing all connections in principle, we avoid biasing against architectures that allow all-to-all connectivity, which would still perform well.

### 2.2 Square Circuits: Random Circuit Sampling

While circuits required for applications are typically not random, sampling from the output
tributions of random circuits built from two-qubit gates has been suggested as a means to demonstrate quantum computational supremacy [29, 44, 54, 55]. Further, by utilising uniformly random two-qubit unitaries, the class we define here, which we refer to as square circuits, provides a benchmark at all layers of the quantum computing stack. In particular it tests the ability of the device to implement a universal gate set, the diversity and quality of the gates available, and the compilation strategy’s ability to decompose these gates to the native architecture. Further, as quantum circuits can always be approximated up to arbitrary precision using two-qubit unitary gates [56], square circuits can help us understand the performance of quantum computing stacks when implementing computations requiring a universal gate set.

Definitions and Related Results A random circuit, for a fixed number of qubits $n$ and coupling map $G_n$, is generated by applying $m = \text{poly}(n)$ uniformly random two-qubit SU(4) gates between qubits connected by edges of $G_n$. Here, “uniformly random” means according to the Haar measure.

Random Circuit Sampling (RCS) is the task of producing samples from the output distribution of random circuits. To perform RCS approximately is to sample from a distribution close to that produced by the random circuit. This task has been shown to be hard even in the average case [54, 55], as outlined in Theorem 2, which improves upon the worst case result for IQP circuits as seen in Theorem 1.

Theorem 2 (Informal [54]) There exists a collection of coupling maps $G_n$, with one for each $n$, and procedure for generating random circuits respecting each $G_n$, for which there is no classical randomised algorithm that performs approximate RCS, to within inverse polynomial $\ell_1$-norm distance error, for a constant fraction of the random circuits.

The conditions imposed on which coupling maps and circuit generation procedures are covered by this theorem are quite mild, but in particular this can be done using circuits with depth $O(n)$ acting on a 2D square lattice [44, 54]. While this is relevant for devices built using superconducting technology [3], we wish to avoid biasing in favour of this technology in particular.

The circuits used here – which are almost identical to those used for the quantum volume benchmark [38] – are generated according to Algorithm 2. We refer to this class of circuits as square circuits, and note that they consist of $n$ layers of two-qubit gates acting between a bipartition of the qubits. There is discrete randomness over the possible bipartition of the qubits, and continuous randomness over the random two-qubit SU(4) gates.

Algorithm 2 The pattern for building square circuits.

**Input:** Number of qubits, $n \in \mathbb{Z}$  
**Worst case depth:** $n$  
**Output:** Circuit, $C_n$

1: Initialise $n$ qubits, labelled $q_1, ..., q_n$, in the state $|\rangle_0$
2: for each layer $t$ up to depth $n$ do
3: \>$\triangleright$ The contents of this loop constitutes a layer. The choice of the number of layers used here is discussed in Appendix A.1.
4: \>$\triangleright$ The contents of this loop constitutes a layer. The choice of the number of layers used here is discussed in Appendix A.1.
5: \>$\triangleright$ The contents of this loop constitutes a layer. The choice of the number of layers used here is discussed in Appendix A.1.
6: Divide the qubits into $\lfloor \frac{n}{2} \rfloor$ pairs $\{q_{i,1}, q_{i,2}\}$ at random.
7: for all $i \in \mathbb{Z}$, $0 \leq i \leq \lfloor \frac{n}{2} \rfloor$ do
8: \>$\triangleright$ The contents of this loop constitutes a layer. The choice of the number of layers used here is discussed in Appendix A.1.
9: Act $U_{i,t} \in \text{SU}(4)$ uniformly at random according to the Haar measure.
10: Act $U_{i,t} \in \text{SU}(4)$ uniformly at random according to the Haar measure.
11: end for
12: end for
13: Measure all qubits in the computational basis.

Discussion By allowing two-qubit gates to act between any pair of qubits in the uncompiled circuit, square circuits avoid favouring any device in particular [3, 29, 44]. This choice adheres closely to our motivations of being hardware-agnostic. In addition, assuming all-to-all connectivity passes the burden of mapping the circuit onto the device to the compilation strategy, which is in line with our wish to benchmark the full quantum computing stack. That said, any architecture whose coupling map closely mirrors the uncompiled circuit will be advantaged, as even a naive compilation strategy will perform well in that case.
In [38] similar circuits are used but with all-to-all connectivity restricted to nearest neighbour connectivity on a line, and the addition of permutation layers. As this disadvantages devices with a completely connected coupling map\(^5\) [5], a property which would typically be an advantage, we choose not to make this restriction here. Notice, however, that naively compiling square circuits onto an architecture with nearest neighbour connectivity on a line would result in the circuits of [38]. This similarity makes a comparison between experiments involving these circuits relevant. As a result, compiling square circuits to superconducting devices (where connectivity is low) will generally result in a circuit similar to those used in the quantum volume benchmark, as many SWAP operations are required regardless.

In addition, square circuits fulfil the necessary conditions to apply HOG, as defined in Problem 1. Namely, the distribution \(p_C\) is sufficiently far from uniform in the required sense, as introduced in Section 3.1, which we demonstrate in Appendix A.1.

### 2.3 Deep Circuits: Pauli Gadgets

Pauli gadgets [57] are quantum circuits implementing an operation corresponding to exponentiating a Pauli tensor. Sequences of Pauli gadgets acting on qubits form product formula circuits, most commonly used in Hamiltonian simulation [30]. Many algorithms employing these circuits require fault-tolerant devices, but they are also the basis of trial state preparation circuits in many variational algorithms, which are the most promising applications of noisy quantum computers. A notable example of this in quantum chemistry is the physically-motivated UCC family of trial states used in the variational quantum eigensolver (VQE) [31, 58]. As near-term quantum computers hold promise as useful tools for studying quantum chemistry, we propose that the quality of an implementation of these gadgets is a useful benchmark, and use them to define the deep circuit class.

Note that the circuits in this class differ from running the VQE end-to-end. Focusing on the state preparation portion of a VQE circuit, we might deduce performance of the quantum computing stack when running the VQE on a number of molecules\(^6\). The intuition being that if the state preparation sub-component is accurate, then the error in the expectation values of measured observables will be due to errors in implementing those observables, or the readout process itself.

### Definitions and Related Results

These circuits are built as in Algorithm 3. They are constructed from several layers of Pauli Gadgets, each acting on a random subset of \(n\) qubits. In the worst case each Pauli gadget will demand \(4n + 1\) gates: \(2n\) Pauli gates, \(2(n - 1)\) CX gates, and one RZ gate.

In the construction of deep circuits there is discrete randomness over the choice of Pauli string, \(s\), and continuous randomness over a rotation angle \(\alpha\).

### Discussion

By establishing the exponential distribution of the output probabilities from deep circuits, as we do in Appendix A.2, we allow ourselves the capacity to use Heavy Output Generation Benchmarking and Cross-Entropy Benchmarking as introduced in Section 3. This constitutes a novel extension of those approaches to application-motivated benchmarking, and the unique ability for us to benchmark application-motivated circuits, using polynomially many samples from a device. This provides a novel insight into the capacity of near-term hardware to implement quantum chemistry circuits.

### 3 Figures of Merit

Suppose a quantum computer is programmed to run a circuit \(C\) or a unitary \(U\). Figures of merit compare \(p_U (p_C)\), the ideal output probabilities for \(U\) \((C)\), and \(D_U (D_C)\), the distributions produced by an implementation, which may be noisy. The remainder of this section outlines three figures of merit and details: their definition, the continuous range of values they can take, their dependence on noise, and the procedure for calculating their value.

---

5Note that some compilation strategies may identify that the SWAP gates in the permutation layer may be removed for devices with all-to-all connectivity. We avoid this dependence on the compilation strategy by fixing the connectivity in the uncompiled circuit to be all-to-all.

6Here we do not explore the relationship between the performance of a quantum computing stack when implementing deep circuits and when implementing VQE but regard it as important for future work.
Algorithm 3 The pattern for building deep circuits.

| Line | Description |
|------|-------------|
| 1: | function PHASEGADGET($\alpha$, $\{\hat{q}_1, ..., \hat{q}_p\}$) |
| 2: | if $p = 1$ then |
| 3: | Act RZ($\alpha$) on $\hat{q}_1$ |
| 4: | else |
| 5: | Act CX between $\hat{q}_1$ and $\hat{q}_2$ |
| 6: | PHASEGADGET($\alpha$, $\{\hat{q}_2, ..., \hat{q}_p\}$) |
| 7: | Act CX between $\hat{q}_1$ and $\hat{q}_2$ |
| 8: | end if |
| 9: | end function |
| 10: | function PAULI($\{\hat{q}_1, ..., \hat{q}_p\}$, s) |
| 11: | if $s_1 = X$ then |
| 12: | Act RX($\frac{\pi}{2}$) on $\hat{q}_1$ |
| 13: | else if $s_1 = Y$ then |
| 14: | Act H on the $\hat{q}_p$ |
| 15: | end if |
| 16: | PAULI($\{\hat{q}_2, ..., \hat{q}_p\}$, s) |
| 17: | end function |
| 18: | function PAULIGADGET($\alpha$, qubits, s) |
| 19: | PAULI(qubits, s) |
| 20: | PHASEGADGET($\alpha$, qubits) |
| 21: | Act the inverse of PAULI(qubits, s) |
| 22: | end function |
| 23: | Initialise $n$ qubits, labelled $q_1, ..., q_n$, in the state $|0\rangle$. |
| 24: | for each layer $t$ up to depth $3n + 1$ do |
| 25: | The contents of this for loop constitutes a layer. The choice of the number of layers used here is discussed in Appendix A.2. |
| 26: | Select a random string $s \in \{1, X, Y, Z\}^n$ |
| 27: | Generate random angle $\alpha \in [0, 2\pi]$ |
| 28: | PAULIGADGET($\alpha$, $\{q_i : s_i \neq 1\}$, s) |
| 29: | end for |
| 30: | Measure all qubits in the computational basis |

from samples produced by an implementation. As noted in the introduction, we use continuous figures of merit which require classical resources to compute.

3.1 Heavy Output Generation Benchmarking

Heavy Output Generation [44] (HOG) is the problem which demands that, given a quantum circuit $C$ as input, strings $x_1, ..., x_k$ be generated which are predominantly those that are the most likely in the output distribution of $C$. That is to say, outputs with the highest probability in the ideal distribution should be produced most regularly.

If the ideal distribution is sufficiently far from uniform, this problem provides a means to distinguish between samples from the ideal distribution and a trivial attempt to mimic such a sampling procedure, namely producing uniformly random strings. Although a simple problem, this task is also conjectured to be hard for a classical computer to perform in general [44].

Importantly, a solution to HOG can be verified by a classical device using polynomial samples from the real distribution. In combination, these properties make the study of the likely output of a distribution a useful tool in benchmarking near-term quantum devices.

Definitions and Related Results Let $p_C(x) = |\langle x | C | 0^n \rangle|^2$ be the probability of measuring the output $x$ in the output probability distribution of an ideal implementation of a circuit $C$. An output $z \in \{0, 1\}^n$ is heavy for a quantum circuit $C$, if $|\langle z | C | 0^n \rangle|^2$ is greater than the median of the set $\{p_C(x) : x \in \{0, 1\}^n\}$.

We can define the probability that samples drawn from a distribution $D_C$ will be heavy outputs in the distribution $p_C$, called the heavy output generation probability of $D_C$, as follows. Here $\delta_C(x) = 1$ if $x$ is heavy for $C$, and 0 otherwise.

$$\text{HOG} (D_C, p_C) = \sum_{x \in \{0, 1\}^n} D_C(x) \delta_C(x)$$ (1)

For HOG $(D_C, p_C)$ to help us distinguish between an ideal implementation of $C$ and a trivial attempt to mimic it by generating random bit strings, HOG $(p_C, p_C)$ should be greater than 0.5. In fact,
HOG \((p_C, p_C)\) is expected to be \(1 + \log 2) / 2 \approx 0.846574\) [44] for circuit classes whose distribution of measurement probabilities, \(p\), is of the exponential form\(^7\) \(\Pr (p) = Ne^{-Np}\), where \(N = 2^n\). This is discussed at length in Appendix A. When the output distributions of a class of circuits is shown to take this form it is meaningful to define the Heavy Output Generation problem.

**Problem 1 (Heavy Output Generation [44])**

Given a measure \(\mu\) over a class of circuits, the family of distributions \(\{D_C\}\) is said to satisfy HOG if the following is true.

\[
\mathbb{E}_{C \sim \mu}[HOG(D_C; p_C)] \geq \frac{2}{3}
\]  

(2)

Indeed, the exponential distribution of the output probabilities of the random circuits defined in [38] allowed for the definition of the quantum volume of a device. This is the largest \(n\) for which distributions \(\{D_{C_n}\}\) which solve the HOG problem introduced in Problem 1, where \(C_n\) are random circuits defined in [38], can be sampled from.

The motivation for the introduction of quantum volume is the classical hardness of solving the HOG problem of Problem 1 for random circuits, under the QUATH assumption of Assumption 1.

**Assumption 1** The QUantum THreshold assumption (QUATH) [44] is that there is no polynomial time classical algorithm that takes as input the description of a random circuit \(C \leftarrow \mu\) and which guesses whether \(|\langle 0^n | C | 0^n \rangle|^2\) is greater or less than the median value in \(\{p_C(x) : x \in \{0, 1\}^n\}\) with success probability at least \(1/2 + \Omega(1/2)\) over the choices of \(C\).

As opposed to the statement that HOG is hard, QUATH does not reference sampling, and concerns only the difficulty of approximating amplitudes. QUATH can be evidenced by observing the difficulties of calculating output probability amplitudes [44].

**Ideal and Noisy Implementations** HOG is solved efficiently by a quantum computer, simply by implementing the circuit \(C\). In the case of extreme noise, and the convergence of the real distribution \(D_C\) to the uniform distribution \(U\), HOG \((D_C, p_C) = 1/2\). This is compared to the case where the output probabilities are exponentially distributed, where \(D_U = p_U\), when we would expect to have HOG \((D_C, p_C) = (1 + \log 2) / 2\). The continuum of values in between provides a valuable figure of merit, which we call Heavy Output Generation Benchmarking, for a quantum computing stack.

**Calculation From Samples** We approximate HOG \((D_C, p_C)\) in a number of operations which grows exponentially with the number of qubits, but using only a polynomial number of samples from the real distribution \(D_C\), by calculating the ideal probabilities \(p_C(x)\). To do so we simply calculate the following expression, where \(x_1, ..., x_k\) are samples drawn from \(D_U\).

\[
\frac{1}{k} \sum_{i=1}^{k} \delta_C (x_i)
\]  

(3)

By the law of large numbers, this converges to HOG \((D_U, p_U)\) in the limit of increasing sample size.

**Discussion** The connections between HOG and quantum computational supremacy allow us to extract valuable insights into the ability of a quantum computing stack to demonstrate quantum computational supremacy. It provides a minimal, single value with which to compare quantum computing stacks, with an intuitive interpretation. The HOG problem of Problem 1, in particular, is easy to solve on a fault tolerant quantum computer with overwhelming success probability.

As with quantum volume, we too will consider the largest \(n\) for which solving the HOG problem of Problem 1 is possible for the circuit classes in Section 2 which have exponentially distributed output probabilities. This is not the case for all circuit classes used here, and for those for which it is not we will explicitly calculate the ideal heavy output probability as a point of comparison. Intuitively, the largest \(n\) solving this problem verifies the largest Hilbert space accessible to a quantum computing stack.
3.2 Cross-Entropy Difference

Cross-entropy benchmarking [29] relates to the average probability, in the ideal distribution, $p_U$, of the outputs which are sampled from the real distribution, $D_U$. For distributions which are far from uniform, and with a spread of probabilities of outcomes, this measure can be used to distinguish an ideal from a real implementation. Ideal implementations will regularly produce the higher probability outputs, obtaining a high benchmark value, while even a small shift in the distribution will lower the value.

The value of the cross entropy difference can be calculated using exponential classical resources, from a polynomial number of samples from a quantum computer, which allows for its utilisation in benchmarking smaller quantum devices [3, 29, 60, 61]. There are also well developed means by which this quantity can be used as a means of extrapolating from the behaviour of smaller devices to that of larger devices, which might demonstrate quantum computational supremacy [3].

Definitions and Related Results  Intuitively, the entropy, $H(D)$, of a distribution, $D$, as defined in equation (4), measures the expectation of one's ‘surprise’ at observing samples from $D$. In this case, this is measured by $f_D(x) = -\log(D(x))$, which accordingly decreases with increasing probability of the outcome occurring.

$$H(D) = \sum_{x \in \{0,1\}^n} D(x) \log \left( \frac{1}{D(x)} \right) \quad (4)$$

By extension, the cross-entropy measures one's surprise when sampling from $D$ when expecting $D'$. This may be restated as the additional information required to describe $D$ given a description of $D'$. Formally, cross-entropy is defined as in Definition 1.

Definition 1 (Cross-Entropy) The cross-entropy between two probability distributions $D$ and $D'$ is

$$\text{CE}(D, D') = \sum_{x \in \{0,1\}^n} D(x) \log \left( \frac{1}{D(x)} \right) \quad (5)$$

Then the cross-entropy difference is simply $\text{CE}(U, D') - \text{CE}(D, D')$, where $U$ is the uniform distribution.

Definition 2 (Cross-Entropy Difference) The cross-entropy difference between two probability distributions $D$ and $D'$ is

$$\text{CED}(D, D') = \sum_{x \in \{0,1\}^n} \left( 1 - D(x) \right) \log \left( \frac{1}{D'(x)} \right). \quad (6)$$

Therefore, the cross-entropy difference can be thought of intuitively as answering “is the distribution $D'$ best predicted by $D$ or by the uniform distribution?”. A different but related definition sets $f_D(x) = 2^{-n} - D(x)^8$, in which case the related quantity is referred to as linear cross entropy [3]. In this case the connection to the average probability of the outputs sampled is clearer.

Ideal and Noisy Implementations  The cross-entropy, $\text{CE}(D_U, p_U)$, between the output distribution, $p_U$, of a unitary, $U$, and the output distribution of an ideal implementation of $U$, $D_U$, reduces to the entropy of $p_U$. In the case where the probabilities $p_U(x)$ are approximately independent and identically distributed according to the exponential distribution, we have that $H(p_U) = \log 2^n + \gamma - 1$ [29], where $\gamma$ is Euler’s constant.

In the case where the probabilities $D(x)$ are uncorrelated with those of $p_U(x)$ we arrive at the following prediction of the cross-entropy [29].

$$E_U [\text{CE}(D_U, p_U)] = \log 2^n + \gamma \quad (7)$$

$D(x)$ and $p_U(x)$ are uncorrelated if, for example, $D$ is the uniform distribution, or, in the case of demonstrations of quantum computational supremacy, if $D$ is the output of a polynomial cost classical algorithm [29].

These results allow us to identify the extreme values taken by the cross-entropy difference.

$D_U = p_U$: When the unitary is implemented perfectly $\text{CED}(D_U, p_U) = 1$.

$D_U = U$: When samples are generated uniformly at random $\text{CED}(D_U, p_U) = 0$.

---

---
As such, the cross entropy difference gives a value between 0 and 1 which measures the accuracy of the implementation of a unitary, the calculation of which is called Cross-Entropy Benchmarking.

**Calculation From Samples** By the law of large numbers, the following expression converges to \( \text{CE}(D_U, p_U) \), where \( x_1, \ldots, x_k \) are samples drawn from \( D_U \).

\[
\frac{1}{k} \sum_{i=1}^{k} \log \left( \frac{1}{p_U(x_i)} \right) \tag{8}
\]

This can be used by a classical computer to approximate the value for \( \text{CE}(D_U, p_U) \). While only a polynomial number of samples \( x_i \) are required, the calculation of \( p_U(x_i) \) takes exponential time.

In our case, to avoid requiring the inverse of 0 in this approximation, we chose to use an approximation to \( p_U \). Namely we approximate it by the larger of \( p_U \) and an inverse exponential in the number of qubits, as is inspired by the average case supremacy results related to random circuits [54, 55].

**Discussion** The comparison to the uniform distribution which the cross-entropy difference provides is valuable as, if an honest attempt is being made to recreate a distribution, at worst \( U \) could be produced. In addition, the cross-entropy gives an estimate for the average circuit fidelity [29], when the conditions of the above discussion are met, facilitating the characterisation of noise levels in implementations of quantum circuits. While Cross-Entropy Benchmarking on its own cannot be used to distinguish error channels, in combination with the techniques introduced here, it can provide insight into this information.

By approximating the fidelity of smaller circuits, Cross-Entropy Benchmarking allows us to characterise larger ones. This is achieved by combining the fidelities of the smaller circuits which themselves combine to give a larger one. This method has been introduced and employed to benchmark demonstrations of quantum computational supremacy [3]. In that domain, calculating the cross-entropy difference of the larger circuit would otherwise be too computationally costly.

The average circuit fidelity may be calculated by decoupling two halves of the device\(^9\), performing Cross-Entropy Benchmarking of the circuit built from gates in the larger circuit which act only on each half respectively, and multiplying together the results of both. This approach is feasible when it can be justified, through numerical simulations and experimental implementations, that the average circuit fidelities do combine in this fashion. This is so when the errors on each output are uncorrelated with the amplitude of that output in the ideal probability distribution.

### 3.3 \( \ell_1 \)-Norm Distance

The \( \ell_1 \)-norm distance between two probability distributions measures the total difference between the probabilities the distributions assign to elements of their sample space. Such a metric is sufficiently strong that for several classes of quantum circuits it is known that classical simulation of all circuits in the class to within some \( \ell_1 \)-norm distance of the ideal distribution would contradict commonly held computational complexity theoretic conjectures [48, 54, 62].

Unlike the previous two figures of merit, approximating the \( \ell_1 \)-norm distance requires a full characterisation of the ideal output distribution. In the cases where few qubits are considered, as is so here, it is possible to perform such characterisations. For larger qubit counts, the cross-entropy benchmarking and heavy output generation are the preferred benchmarking schemes.

**Definitions and Related Results** In the case of distributions over the sample space \( \{0,1\}^n \), the \( \ell_1 \)-norm distance is defined as follows.

**Definition 3 (\( \ell_1 \)-norm distance)** For distributions \( D \) and \( D' \) over the sample space \( \{0,1\}^n \) the \( \ell_1 \)-norm distance between them is defined as

\[
\ell_1(D, D') = \sum_{x \in \{0,1\}^n} |D(x) - D'(x)|. \tag{9}
\]

**Ideal and Noisy Implementations** An ideal implementation of a unitary would result in a \( \ell_1 \)-norm distance of \( \ell_1(D_U, p_U) = 0 \). However, noise will likely make it incredibly difficult for even fault tolerant quantum computers to achieve a \( \ell_1 \)-norm fully coupled circuits are investigated to ensure the accuracy of this method of combining fidelities.

---

\(^9\)In the work of [3] both decoupled, partially coupled, and
distance of 0 and so bounds, such as that discussed in Theorem 1, are often put on the value instead. Indeed in that case it is sufficient for \( \ell_1(D_U, p_U) \) to be bounded for a demonstration of quantum computational supremacy to occur.

Once again, the \( \ell_1 \)-norm distance takes a continuous range of values allowing for comparison between implementations of circuits.

**Calculation From Samples** In this work we will approximate the \( \ell_1 \)-norm distance between the ideal and real distributions using samples from the real distribution. Given samples \( s = \{x_1, \ldots, x_m\} \) from \( D_U \), let \( s^x \) be the number of times \( x \) appears in \( s \). Define \( D_U \) by \( D_U(x) = (s^x)/m \). Then the approximation we will use for \( \ell_1(D_U, p_U) \) is \( \ell_1(D_U, p_U) \).

**Discussion** Because of its independence from probability values themselves, the \( \ell_1 \)-norm distance is regarded as a fair measure on the closeness of distributions. That is to say, it is reasonable to require quantum computers to produce samples from distributions. This might not be true for measures of distance, such as multiplicative error, which require zero probability outcomes are preserved in the presence of noise, but for which very strong connections to quantum computational supremacy also exist [47].

### 3.4 Metric Comparison

Unfortunately, Cross-Entropy Benchmarking and Heavy Output Generation Benchmarking cannot be used to bound the \( \ell_1 \)-norm distance [54], which, as noted in Section 3.3, provides strong guarantees of demonstrations of quantum computational supremacy\(^1\). That is, the \( \ell_1 \)-norm distance provides uniquely (amongst the metrics studied here) strong assurances about quantum computational supremacy. This comes at the cost of requiring full state vector simulation to calculate it, consuming memory which grows exponentially in the number of qubits. As the circuit widths approach those large enough to demonstrate quantum computational supremacy, memory requirements become the bottleneck [63].

In the case of Heavy Output Generation Benchmarking and Cross-Entropy Benchmarking only polynomially many single output probabilities are required, allowing the utilisation of *Feynman simulators* [44]. These compute output bit string amplitudes by adding all Feynman path contributions. This extends the domain of classical simulation by overcoming the memory storage problem, establishing the frontier of what’s possible on classical computers [3, 64, 65]. However, this method still requires exponential time to perform and so reaches its own limit for large numbers of qubits.

Since \( \text{HOG}(D_U, p_U) \) and \( \text{CE}(D_U, p_U) \) are expectations of different functions of ideal output probabilities, \( \delta(p_U) \) and \( -\log(p_U) \) respectively, over the experimental output distribution, they capture different features of the outputs [54]. In fact \( \text{HOG}(D_U, p_U) \) can also be used to approximate circuit fidelity, however the standard deviation of the estimator is larger than that for \( \text{CE}(D_U, p_U) \) [3].

## 4 Quantum Computing Stack

Each component of a quantum computing stack exerts an influence on overall performance, and identifying the distinct impact of a particular component is often hard. To disentangle these factors, we must clearly identify the components used during benchmarking. Here we detail the components used to build the quantum computing stacks explored in Section 5. The diverse selection of components allows us to investigate a variety of ways of building a quantum computing stack.

### 4.1 Software Development Kits

We use a combination of tools available via *pytket* [10, 66] and *Qiskit* [1, 67]. *pytket* is a Python module which provides an environment for constructing and implementing quantum circuits, as well as for interfacing with CQC’s t\(|\text{ket}\rangle\), a retargetable compiler for near term quantum devices featuring hardware-agnostic optimisation. *Qiskit* is an open-source quantum computing software development framework for programming, simulating, and inter-

\(^1\)Interestingly, empirical result show a slight negative correlation between \( \ell_1 \)-norm distance and experimental heavy output probability (normalized to the heavy output probability of an ideal device). This relationship is discussed in Appendix D, which provides empirical details of this relationship.
acting with quantum processors, which also provides a compiler. Details of the versions of the software used are seen in Table 2 of Appendix B.

We use three parts of Qiskit in this work. First is the transpiler architecture, which enables users to define a custom compilation strategies by executing a series of passes on the input circuit, as discussed in Section 4.2. The second part of Qiskit we use is the library of predefined passes. Finally, a provider is used to access hardware made available over the cloud by IBM Quantum. The provider enables users to send circuits to hardware, retrieve results, and query the hardware for its properties.

Similarly, we use pytket to generate and manipulate circuits in several ways. Firstly we use the tket compiler to construct compilation strategies which optimise the input circuit for the target hardware, utilising predefined passes available in tket. Secondly we use pytket to define abstract circuits and to convert tket’s native representation of the circuit into a Qiskit QuantumCircuit object which is then dispatched to IBM Quantum’s systems for execution.

4.2 Compilers

Compilers provide tools to construct executable quantum circuits from abstract circuit models. This is done by defining passes which may manipulate a representation of a quantum circuit, often by taking account of limited connectivity architectures, or minimising quantities such as gate depth, but need not perform any manipulation. These passes are composed to form compilation strategies which should output executable quantum circuits. Quantum compiling is an active area of research, and there are many pieces of software available for quantum compiling. As noted above, in this work we use two: tket and the compiler available in Qiskit.

For the purposes of this work, the problem of quantum compilation is divided into three tasks.

Placement: Determine onto which physical qubits of a given device the virtual qubits in the circuit’s representation should be initially mapped.

Routing: Modify a circuit to conform to the qubit layout of a specific architecture, for example, by inserting SWAP gates to allow non-adjacent qubits to interact. Circuits are rarely designed with the device’s coupling map in mind, so this step is important.

Optimisation: Work to minimise some property of a circuit. This may be gate count or depth, which is done to improve implementation accuracy by reducing the impact of noise.

Each of these tasks could consider such things as the trade-offs between the connectivity of a particular subgraph of the device and the amount of crosstalk present in that subgraph.

Both pytket and Qiskit have multiple placement, optimisation, and routing passes. We compare the performance of 5 compilation strategies built from these passes. Two of them, noise-unaware pytket and noise-unaware Qiskit, compile the circuit without knowledge of the device’s noise properties. Another two, noise-aware pytket and noise-aware Qiskit, do take noise properties into account. As a baseline, we consider a simple compilation strategy from pytket using only routing, without optimisation or noise-awareness; we refer to this pass as only pytket routing. We detail these schemes in Appendix B. The main difference between the noise-aware schemes is that noise-aware pytket prioritises the minimisation of gate errors during placement, whereas noise-aware Qiskit prioritises readout and CX errors.

4.3 Devices

We benchmark some of the devices made available over the cloud by IBM Quantum. The devices we use are referred to by the unique names ibmqx2, ibmq_16_melbourne, ibmq_singapore and ibmq_ourense. Each device has a set of native gates which all gates in a given circuit must be decomposed to. For all the devices considered here, the native gates are: an identity operation, I; 3 “u-gates” [78], as defined in equation (10); and a

---

11These properties include the graph connectivity, single- and two-qubit error rates, and qubit T1 and T2 times. Some of the noise-aware compilation strategies we use require knowledge of these properties.

12An example of this is a pass which counts the gates in the circuit.

13This is true of pytket 0.3.0; as a result of this work, later versions of pytket take into account readout error.
controlled-NOT (CX) gate.

\[
U_3 (\theta, \phi, \lambda) = \begin{pmatrix}
\cos \left( \frac{\theta}{2} \right) & -e^{i\lambda} \sin \left( \frac{\theta}{2} \right) \\
e^{i\phi} \sin \left( \frac{\theta}{2} \right) & e^{i(\lambda+\phi)} \cos \left( \frac{\theta}{2} \right)
\end{pmatrix}
\]

\[
U_2 (\phi, \lambda) = U_3 \left( \frac{\pi}{2}, \phi, \lambda \right)
\]

\[
U_1 (\lambda) = U_3 (0, 0, \lambda)
\]

(10)

Two of the device properties used by the noise-aware compilation strategies are their connectivity and calibration data. The connectivity of a device refers to the connectivity of the graph representing how the qubits are coupled to one another. This information is contained in a device’s coupling map, which, in the cases of the devices studied here, are shown in Appendix C.1 and summarised in Table 1.

Device calibration data includes information about single- and two-qubit error rates, readout error, and qubit frequency, \( T_1 \), and \( T_2 \) times. The noise-aware compilation strategies we investigate use the gate error rates and readout error. Full details of noise levels can be found in Appendix C.2 with average values given in Figure 1. This information is updated twice daily, with the data in Figure 1 averaged over the period 2020-01-29 to 2020-02-10 during which the experiments were conducted.

The results of Section 5 depend heavily on the noise levels of the device at the time at which the computation is implemented. This is doubly true in the case of the noise-aware optimisation schemes as a circuit optimised at one time may not perform as well over time as the noise levels of the devices change. To reduce this effect we endeavoured to compile and run circuits within as short a time interval as possible.

5 Experimental Results

In this section we identify, using the benchmark suite defined in Section 2 and Section 3, which properties of different levels of a quantum computing stack of Section 4 result in the best performance. This allows us to suggest means to extract as much computing power as is possible from the devices available now, and in the near future. The benchmark suite is used here to probe the performance of quantum computing stacks in three ways:

**Full Stack Benchmarking:** In Section 5.1 we perform benchmarks of the full quantum computing stack. Incorporating and thoroughly investigating the compilation strategy, in particular, helps develop an understanding of how circuit compilation influences the performance of the quantum computing stack. In the case of noise-aware compilation strategies, this also highlights how the assumptions made by the strategy about the importance of different kinds of noise impacts performance.

**Application Motivated Benchmarks:** In Section 5.2, by including three quite different circuit classes in our benchmark suite, we explore how a quantum computing stack may perform when implementing a wide array of applications.

**Insights from Classical Simulation:** In Section 5.3 we explore how benchmarks themselves can assist in the task of developing new noise models. By identifying when benchmark values for real implementations and those we expect from simulations using noise models differ, noise channels which should be added to the noise models to achieve greater agreement with real devices can be identified. This is of particular importance as noise-aware compilation strategies often utilise noise properties.

In the following subsections we present results on each of these topics. For each circuit class and fixed number of qubits, 200 circuits were generated according to the circuit generation algorithms of Section 2. Each circuit is compiled by a given compilation strategy onto a particular device. The compiled circuits were then run on the device, using 8192 repetitions (samples) from each compiled circuit, which generates 8192 bitstrings. The compiled circuits are also classically simulated using a noise model built from the device calibration information at the time of the device run. See Data Availability for access to the full experimental data set.

The resulting bitstrings are then processed according to the figures of merit given in Section 3. The distribution of the figures of merit are com-
| Device             | Vertices | Average Degree | Radius | Minimum Cycle Length |
|--------------------|----------|----------------|--------|----------------------|
| ibmqx2             | 5        | 2.4            | 1      | 3                    |
| ibmq_16_melbourne  | 15       | 2.3            | 4      | 4                    |
| ibmq_ourense       | 5        | 1.6            | 2      | N/A                  |
| ibmq_singapore     | 20       | 2.3            | 4      | 6                    |

Table 1: **Selected graph properties of the coupling maps of devices studied in this work.** This table displays: the number of vertices in the graph (corresponding to the number of qubits on the device); the average degree, which is the mean number of edges incident on each vertex; the radius, which is the minimax distance over all pairs of vertices; and the minimum cycle length, which is the smallest number of edges per cycle over all cycles of the graph. See Appendix C.1 for full details of the coupling maps of the devices explored here.

![Error Rate](image)

**Figure 1:** **Average error rates across devices used in this work.** Bars show the mean error rates across the whole device, while error bars give the standard deviation. Devices shown here are: ibmqx2, ibmq_ourense, ibmq_singapore, ibmq_16_melbourne. Further details can be found in Appendix C.2
pared by their mean, and their shape, which is aggregated into a box-and-whisker plot. Uncompiled circuits were also perfectly simulated without noise in order to calculate the ideal heavy output probability. These points are referred to as Noise-Free in the figures below.

5.1 Full Stack Benchmarking

Impact of the Compilation Strategy

The two layers of the quantum computing stack we study are the compilation strategy and the device on which the compiled circuit is run. Using a fixed device and comparing multiple compilation strategies allows us to determine which strategy tends to perform well. Further, we aggregate performance over all compilation strategies as a way of estimating the performance of a “generic” strategy. Similarly, using a fixed strategy and comparing its performance on multiple devices enables a study of how the assumptions made by the strategy about the devices impact performance when those assumptions don’t always hold.

Figure 2 displays experimental results making this comparison when implementing square circuits on ibmq_16_melbourne, using heavy output generation probability as the figure of merit. The noise-aware pytket compilation strategy performs somewhat better, on average, than a generic strategy. Because the aggregated information (“All Strategies” in Figure 2) includes aggregation over noise-aware pytket, these results indicate that other compilation strategies perform a bit worse, since the performance of the aggregate is generally lower than that of noise-aware pytket. This reveals both the potential for compilation strategy driven improvements in performance, and the insights into such improvements which full quantum computing stack benchmarking brings.

Aggregation over compilation strategies is not only useful for identifying strategies which are better in general. Doing so also provides a way of identifying devices which perform well, by “washing out” the effect of the compilation strategy on performance. That is, the strong performance (using a systems-level benchmark) of a given device might be caused by the compilation strategy; to reduce the effect of the strategy, aggregation over several can be done.

For example, Figure 3 shows that by considering performance with a fixed compilation strategy (in this case, noise-aware pytket), ibmq_singapore would be considered to perform similarly, if not slightly better than ibmq_ourense, as measured by $\ell_1$-norm distance. However, aggregating over all strategies, as is done in Figure 4, shows ibmq_ourense to perform better. This suggests that ibmq_ourense might be a better device for a “generic” compilation strategy to compile to.

An instance-by-instance comparison of different compilation strategies also helps us understand their limitations. For example, Figure 5 reveals noise-aware pytket works best at reproducing the ideal distribution of heavy output probabilities of square circuits on ibmq_16_melbourne. This is likely in part due to the routing scheme, as is revealed by the strong performance of only pytket routing.

Similarly, Figure 6 shows that noise-aware pytket is amongst the worst-performing compilation strategies for lower numbers of qubits, while it is amongst the best-performing for higher numbers. This could be a result of the way in which noise-aware pytket prioritises noise in its routing scheme, with gate errors taking precedence.

These results highlight the fact that full-stack benchmarking can help provide a more detailed understanding of how the components of a system affect performance.

Noise Level, Connectivity Trade Off

Another particularly important example of this is the examination of the connectedness of the device and its noise levels. More highly-connected architectures typically allow for shallower implementations of a given circuit as compared to less-connected ones, but the noise levels in a more highly-connected architecture may be higher due to crosstalk. This creates a trade-off between

---

14Note that due to the fact we aggregate over all 5 compilation strategies, the distribution of heavy output probabilities amongst the “All Strategies” category contains five times as many points as compared to those for noise-aware pytket.

15For larger numbers of qubits and deeper circuits, gate errors becomes more impactful on the total noise, and materialises as giving noise-aware pytket an advantage for larger numbers of qubits.
Figure 2: Comparison of fixed compilation strategy to average of all strategies, using the heavy outputs probability metric, when running square circuits using the real `ibmq_16_melbourne` device. Boxes show quartiles of the dataset while the whiskers extend to 1.5 times the IQR past the low and high quartiles. White circles give the mean.

Figure 3: Comparison of devices, using the $\ell_1$-norm distance metric, when running shallow circuits compiled using `noise-aware pytket`. Both simulations using Qiskit noise models, and implementations on real devices, are included. Boxes show quartiles of the dataset while the whiskers extend to 1.5 times the IQR past the upper and lower quartiles. White circles give the mean.
Figure 4: Comparison of real devices, using the \(\ell_1\)-norm distance metric, when running shallow circuits compiled using all compilation strategies. Here we compile onto each device using all compilation strategies, including all compiled circuits in this plot. Boxes show quartiles of the dataset while the whiskers extend to 1.5 times the IQR past the upper and lower quartiles. White circles give the mean.

Figure 5: Comparison of compilation strategies, using the heavy outputs probability metric, when square circuits are ran on the real ibmq_16_melbourne device. Boxes show quartiles of the dataset while the whiskers extend to 1.5 times the IQR past the upper and lower quartiles. White circles give the mean.
connectivity and the total amount of noise incurred when running a computation.

As noise affects the accuracy of the computation, this trade-off has practical implications for the performance of a device. Indeed, reducing the connectivity between superconducting qubits is used as a tool to reduce noise levels [79]. This can also be counteracted by decoupling qubits [3] but this is not utilised in the devices studied here. Figure 7 shows that devices with lower noise levels (ibmq_singapore and ibmq_ourense) typically outperform devices with higher noise levels (ibmqx2 and ibmq_16_melbourne) despite the latter’s higher connectivity. An interesting exception to this is for 4 qubits, where ibmq_16_melbourne performs best, likely because of the 4-qubit cycles in its connectivity graph. This reduces the SWAP operations necessary for implementing the circuit, reducing the overall circuit depth. This reveals the increase in performance that can be expected when the connectivity of the device and the problem instance are similar [12]. Similar results hold for Cross-Entropy Benchmarking, as shown in Figure 8.

In general, we expect that circuits whose structure can naturally be mapped to the connectivity of the device will generally perform well, whereas those which cannot, will not. In general though, lower-noise devices will tend to perform best.

Comparison with Previous Results
As discussed in Section 2.2, our definition of square circuits differs from previous experiments [38], by allowing for all-to-all connectivity before compilation, as opposed to utilising permutation layers. However, as a naive compilation of square circuits onto a one-dimensional, nearest-neighbour connectivity would recreate the same circuits as used in [38], we might expect the results from our experiments to be similar, and a comparison of the results from these experiments is warranted. Further, in the case of superconducting devices, as are explored here, we would expect the circuit after compilation to be similar, as many SWAP operations will need to take place in both cases.

Figure 7 shows that all quantum computing stacks explored here produce heavy outputs with probability greater than $\frac{2}{3}$ on average for circuits acting on at most 3 qubits, with ibmq_ourense typically performing best. Previous results reported that ibmq_singapore could demonstrate a quantum volume, as defined in [38], of $2^4$ [80]. That our experiments produce different results is surprising, given their aforementioned similarity. One uncontrollable variable is changes of the device over the time between previous experiments and this one, which may influence this discrepancy. Indeed, studying the change in the value of volumet-
Figure 7: Comparison of devices, using the heavy outputs probability metric, when running square circuits compiled using noise-aware pytket. Both simulations using Qiskit noise models, and implementations on real devices, are included. Boxes show quartiles of the dataset while the whiskers extend to 1.5 times the IQR past the upper and lower quartiles. White circles give the mean.
Figure 8: Comparison of devices, using the cross entropy difference metric, when running square circuits compiled using **noise-aware pytket**. Both simulations using Qiskit noise models, and implementations on real devices, are included. Boxes show quartiles of the dataset while the whiskers extend to 1.5 times the IQR past the upper and lower quartiles. White circles give the mean.
ric benchmarks over time would be interesting and useful, although no such effort is known to us.

5.2 Application Motivated Benchmarks

The same quantum computing stack will perform differently when running different applications, as the structure of the circuits they require will generally be different. Differences in performance are seen in the context of our application-motivated benchmarks. For example, consider Figure 9, which shows performance when implementing sparsely connected circuits, and Figure 10, which shows performance when implementing chemistry-motivated circuits. In the case of Figure 9, the \textit{ibmqx2} device outperforms \textit{ibmq\_singapore}, while in the case of Figure 10 the reverse is true.

Quantum Chemistry

Figure 10 suggests \textit{ibmq\_ourense} is best for quantum chemistry applications, because it performs well when running deep circuits\footnote{This comes with the caveat, as mentioned in Section 2.3, that the connection between the quality of an implementation of these computational primitives, as measured by this benchmark, and accurate ground state energy calculations in VQE has not been demonstrated experimentally.}. In particular Figure 10 indicates that the average circuit fidelity is highest for implementations on \textit{ibmq\_ourense}.

In Figure 10, all devices converge to the minimum value of cross-entropy difference at 4 qubits. To extend an investigation of this sort to more qubits would require lower noise levels or chemistry motivated circuits which generate exponentially distributed output probabilities at lower depth.

Shallow Circuits as a Benchmark

Figure 11 demonstrates that shallow circuits allow us to benchmark the behaviour of a quantum computing stack for applications involving circuits with many qubits but low circuit depth [35, 43, 46]. In this case we are able to continue our analysis, beyond that of Figure 7, of those devices which perform sufficiently well for a smaller number of qubits, and which have architectures including more qubits to investigate.

The results show \textit{ibmq\_singapore} outperforms the comparably sized \textit{ibmq\_16\_melbourne} and has comparable performance to \textit{ibmq\_ourense} for smaller numbers of qubits. \textit{ibmq\_singapore} outperforms \textit{ibmq\_ourense} by having more qubits available. This superior performance of \textit{ibmq\_singapore} is in comparison to the results of Figure 7, where \textit{ibmq\_ourense} was shown to perform well. This justifies our suggestion that shallow circuits should be included in benchmarking suites. Doing so allows for the exploration of higher qubit requirement computations. In this setting devices that perform poorly when implementing square circuits or deep circuits may perform well.

Shallow Circuits and $\ell_1$-Norm Distance

Theorem 1 provides a convenient criterion for success in implementing shallow circuits; namely an $\ell_1$-norm distance of not more than $1/192$ from the ideal distribution. Figure 4 explores the closeness to a successful implementation and reveals that, on average over all compilation strategies, the best performing device is \textit{ibmq\_ourense}.

Figure 6 explores the best performing compilation strategies for \textit{ibmq\_ourense}. It shows that, compared to the other strategies, the mean $\ell_1$-norm distance is marginally smaller for noise-aware pytket when the number of qubits is larger, while noise-unaware Qiskit and noise-aware Qiskit perform well for fewer qubits. We explore the performance of noise-aware pytket further, as the instances with higher qubit counts are relevant for use cases of quantum computers. Indeed, Figure 3 shows \textit{ibmq\_ourense} and \textit{ibmq\_singapore} perform similarly when the noise-aware pytket optimiser is used, despite \textit{ibmq\_ourense} performing better on average. This is likely because \textit{ibmq\_singapore} has a sub lattice with comparable noise levels to \textit{ibmq\_ourense}, which noise-aware pytket is able to isolate, while on average the levels are higher.

No device consistently brings the $\ell_1$-norm distance to within $1/192$ of the ideal. However, \textit{ibmq\_ourense} seems to slightly outperform the other devices, showing the benefit of lower noise levels over high connectivity or high numbers of qubits. While this methodology would be impossible to extend to the demonstrations of quantum computational supremacy, we hope that exploring it for these quantum computational supremacy related circuits will provide insights into the best quantum computing stack for such demonstrations.
Figure 9: **Comparison of real devices, using the cross entropy difference metric, when running shallow circuits compiled using noise-aware Qiskit.** Boxes show quartiles of the dataset while the whiskers extend to 1.5 times the IQR past the upper and lower quartiles. White circles give the mean.

Figure 10: **Comparison of devices, using the cross entropy difference metric, when running deep circuits compiled using noise-aware Qiskit.** Both simulations using Qiskit noise models, and implementations on real devices, are included. Boxes show quartiles of the dataset while the whiskers extend to 1.5 times the IQR past the upper and lower quartiles. White circles give the mean.
Figure 11: Comparison of real devices, using the heavy outputs probability metric, when running shallow circuits compiled using noise-aware pytket. Boxes show quartiles of the dataset while the whiskers extend to 1.5 times the IQR past the upper and lower quartiles. White circles give the mean.

5.3 Insights from Classical Simulation

The noise present in a non-fault-tolerant quantum computer results in discrepancies between results obtained from running on real hardware and those that would be obtained from an ideal quantum computer. Often, noise models are utilised during classical simulation to investigate the effects of noise and help identify why these discrepancies occur [81]. However, a perfect model of the noise, which could reproduce the results of real hardware (up to statistical error) could require many parameters to completely specify it. Therefore, most noise models consider only a small handful of physical effects. Consequently discrepancies between the results of noisy simulation and running experiments on real hardware always remain.

Historically, closing the gap between noisy simulation and real hardware required developing noise models of increasing sophistication. Developing them typically requires a great deal of physics expertise to identify new noise channels. Further, new experiments would have to be designed in order to estimate their parameters in the noise model.

Here, we suggest some of the benchmarks conducted in this work could be helpful in identifying whether new noise channels should be incorporated into a noise model. In particular, by isolating the circuit types and coupling maps for which the discrepancies are greatest, it is possible to speculate about the possible causes of the mismatch. This investigation could also influence the performance of noise-aware compilation strategies, which use properties of the noise. Verifying the accuracy of these noise properties, through verification of the accuracy of the resulting noise models, could improve the performance of noise-aware strategies.

For the devices explored here, the noise models are built using Qiskit. They are derived from a device’s properties and include one- and two-qubit gate errors and single-qubit readout errors. We find these noise models are inadequate to explain some of the discrepancies observed in the data.

Noise Does Not Just Flatten Distributions

One discrepancy between experiments and noisy simulations is the spread of the data. For example, Figure 7 shows that only in the experimental case do the whiskers of the plot fall below the value 0.5, indicating the heavy outputs are less likely than they would be in the uniform distribution. Some noise type, in particular one which shifts the probability density, rather than uniformly flattening it, is not considered, or is under appreciated, by the noise models used. Identifying that noise channel is left to future work, though we speculate it may be related to a kind of thermal relaxation error.

18These are modelled to consist of a depolarising errors followed by a thermal relaxation errors.
Noise Models Under Represent Some Noise Channels

The classical simulations in Figure 7 suggest ibmq2 should perform similarly to ibmq_ourense in most cases. In fact, it quite consistently performs worse. This is isolated in Figure 8, with the same phenomenon being observed in Figure 3 and Figure 10, showing the behavior is consistent across all circuit types and figures of merit.

This difference between simulated and experimental results is pronounced in the case of Figure 10, where deep circuits are used. This suggests the noise models may be underestimating the error from time-dependent noises such as depolarising and dephasing, or from two-qubit gates which are more prevalent in deep circuits.

Another such example of a two-qubit noise channel, which is explicitly not accounted for in the noise models, is crosstalk. The results in Figure 8 are consistent with the expectation that crosstalk should have the greatest impact on more highly connected devices [79]. As such crosstalk may be the origin of the discrepancy. Of note is the fact this benchmark wasn’t explicitly designed to capture the effects of crosstalk, and yet those effects manifest themselves in its results. We anticipate that including crosstalk-aware passes in compilation strategies [77] would reduce the discrepancy.

6 Conclusion

The performance of quantum computing devices is highly dependent on several factors. Amongst them are the noise levels of the device, the software used to construct and manipulate the circuits implemented, and the applications for which the device is used. The impact of these factors on the performance of a quantum computing stack are intertwined, making the task of predicting its holistic performance from knowledge of the performance of each component impossible. In order to understand and measure the performance of quantum computing stacks, benchmarks must take this into consideration.

In this work we have addressed this problem by introducing a methodology for performing application-motivated, holistic benchmarking of the full quantum computing stack. To do so we provide a benchmark suite utilising differing circuit classes and figures of merit to access a variety of properties of the device. This includes the use of three circuit classes: deep circuits and shallow circuits, which are novel to this paper; and square circuits, which resemble random circuits used in other benchmarking experiments [38]. In addition we make use of a diverse selection of figures of merit to measure the performance of the quantum computing stacks considered, namely: Heavy Output Generation Benchmarking, Cross-Entropy Benchmarking, and the $\ell_1$-norm distance.

In particular, in the form of deep circuits we present an alternative to previous approaches to application-motivated benchmarking. This is by considering circuits inspired by one of the primitives utilised in VQE, namely Pauli gadgets employed for state preparation, rather than VQE itself. Further, while we have found that the performances of quantum computing stacks are indistinguishable when using square circuits and Heavy Output Generation Benchmarking for a large number of qubits, shallow circuits extend the number of qubits for which detail can be observed, while also being consistent with philosophy of volumetric benchmarking.

We demonstrate this benchmark suite by employing it on ibmq2, ibmq_16_melbourne, ibmq_ourense, and ibmq_singapore. In doing so we justified our thesis that the accuracy of a computation depends on several levels of the quantum computing stack, and that each layer should not be considered in isolation. For example, identifying that the increased connectivity of a device does not compensate for the increased noise, as we do in Section 5.1, shows the impact of this layer of the stack, and justifies investigating devices with a variety of coupling maps and noise levels. By showing the differing performance between five compilation strategies, we are able to identify, in Section 5.1, the dependence of the best compilation strategy to use on the device and the dimension of the circuit. This illustrates the dependence of the performance of the quantum computing stack on the compilation layer, and the interdependence between the compilation strategy, device and application on the overall performance of the quantum computing stack. In particular, noise-aware compilation strategies often perform well, when the noise model used by the strategy is accurate, as discussed in Section 5.3.
In Section 5.2, the wide selection of circuits within the proposed benchmark suite reveals that the same device, evaluated according to a fixed figure of merit, will perform differently when running different applications, whose circuits are compiled by the same compilation strategy. Indeed the comparative performance of (compilation strategy, device) pairs is shown to vary between our circuit classes. This justifies our inclusion of circuit classes which collectively cover a wide selection of applications in the benchmark suite proposed here, and our full quantum computing stack approach.

We foresee the benchmarks conducted in this work providing a means to select the best quantum computing stack, of those explored here, for a particular task, and vice versa. As such we also anticipate that a variety of new quantum computing stacks could be benchmarked in the way described in this work, empowering the user with knowledge about the performance of current quantum technologies for particular tasks.

These benchmarks may, in time, come to complement noise models and calibration information as a means to disseminate information about a device’s performance. This parallels the use of the LINPACK benchmarks [6] alongside FLOPS to compare diverse classical computers. Recently, quantum volume, as defined in [38], has started to be adopted as one such metric [82], and we hope the benchmark suite developed here will be incorporated similarly. Further, our benchmarks may facilitate an understanding of how new, or hard-to-characterize, noise affects the practical performance of quantum computers, as implied by the classical simulations of Section 5.3.

The work presented here could be extended in several directions. The first is to examine the impact of incorporating these benchmarks into a compilation strategy. While noise-aware compilation strategies currently use properties of qubits to decide how to compile a circuit, it would be interesting to explore if instead optimising for these benchmarks would change the compilation. The trade-off between the benefits of doing so against the increased compilation time resulting from the time taken to perform the benchmarks should then also be assessed. This information would help in the understanding of the interplay between the amount of classical circuit optimisation performed and the amount by which the performance of a quantum system can be increased.

Second, the philosophy of application-motivated benchmarking could be extended to circuits which are more easily classically simulable. Because of their reliance on classical simulation, the benchmarks introduced here may be used up to, but not after, the point of demonstrating quantum computational supremacy. Hence new circuit classes will need to be introduced which can be classically simulated in this regime. Alternatively, application-motivated benchmarks that are derived from combining benchmarks of smaller devices [3] could be developed.

Third, we envision a need to systematically study how properties of hardware, such as noise levels or connectivity, influence a given device’s performance. In this work, we were limited to the particular devices made available by IBM Quantum, which limits our ability to perform such a systematic inquiry. It is nevertheless vital to do so, as the results of Section 5.1 show that changing the hardware can dramatically influence performance. Indeed, this would allow us understand if the observations made in Section 5.1 are typical, and to explore the existence of other relationships. This could be achieved by implementing this benchmark suite on more devices, or synthetic devices with tunable coupling maps and noise information.

Finally, there is a need study the correlation between the results of an application-motivated benchmark and the performance of a quantum computing stack at running the application which motivated it. This would show that benchmarking application subroutines provides reliable predictors of performance when running the application itself. While similar work has explored the correlation between the classification accuracy and circuit properties of parametrised quantum circuits [37], comparing the performance of the benchmarks defined here with their applications is a subject for future work. For example, comparing the performance of a stack at implementing deep circuits and running the VQE algorithm would show the extent to which quantum computing stacks that perform well at a particular kind of state preparation circuit also perform well in estimating properties of a wide range of molecules.
Data Availability

The data gathered during the experiments conducted for this work, as presented in Section 5, are available at http://doi.org/10.5281/zenodo.3832121. QASM files representing the circuits executed, the per sample bitstring outputs from device and simulator executions, and device calibration data gathered throughout the course of the experiments, are provided.

Acknowledgements

DM acknowledges support from the Engineering and Physical Sciences Research Council (grant EP/L01503X/1). This work was made possible, in part, by systems built by IBM as part of the IBM Quantum program, and made accessible via membership in the IBM Q Network.

IBM, IBM Q, and Qiskit are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product or service names may be trademarks or service marks of IBM or other companies.

References

[1] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, et al. Qiskit: An Open-source Framework for Quantum Computing, January 2019. URL https://doi.org/10.5281/zenodo.2562111.

[2] 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, April 2020. doi:10.1088/2058-9565/ab7559. URL https://doi.org/10.1088%2F2058-9565%2Fab7559.

[3] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, et al. Quantum supremacy using a programmable superconducting processor. Nature, 574(7779):505–510, 2019. ISSN 1476-4687. doi:10.1038/s41586-019-1666-5. URL https://doi.org/10.1038/s41586-019-1666-5.

[4] Simon J. Devitt. Performing quantum computing experiments in the cloud. Phys. Rev. A, 94:032329, September 2016. doi:10.1103/PhysRevA.94.032329. URL https://link.aps.org/doi/10.1103/PhysRevA.94.032329.

[5] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe. Demonstration of a small programmable quantum computer with atomic qubits. Nature, 536(7614):63–66, August 2016. ISSN 1476-4687. doi:10.1038/nature18648. URL https://doi.org/10.1038/nature18648.

[6] A Petitet, R C Whaley, J Dongarra, and A Cleary. Hpl - a portable implementation of the high-performance linpack benchmark for distributed-memory computers. URL http://www.netlib.org/benchmark/hpl/.

[7] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. The linpack benchmark: past, present and future. Concurrency and Computation: Practice and Experience, 15(9):803–820, 2003. doi:10.1002/cpe.728. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.728.

[8] Jack Dongarra and Piotr Luszczek. TOP500, pages 2055–2057. Springer US, Boston, MA, 2011. ISBN 978-0-387-09766-4. doi:10.1007/978-0-387-09766-4_157. URL https://doi.org/10.1007/978-0-387-09766-4_157.

[9] Ryan LaRose. Overview and Comparison of Gate Level Quantum Software Platforms. Quantum, 3:130, March 2019. ISSN 2521-327X. doi:10.22331/q-2019-03-25-130. URL https://doi.org/10.22331/q-2019-03-25-130.

[10] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. \(|\psi\rangle\): A retargetable compiler for nisq devices. Quantum Science and Technology, 2020. doi:10.1088/2058-9565/ab8e92. URL https://iopscience.iop.org/article/10.1088/2058-9565/ab8e92.
[11] Robert S. Smith, Michael J. Curtis, and William J. Zeng. A practical quantum instruction set architecture. 2016. URL https://arxiv.org/abs/1608.03355.

[12] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, et al. Quantum approximate optimization of non-planar graph problems on a planar superconducting processor, 2020. URL https://arxiv.org/abs/2004.04197.

[13] Robin Blume-Kohout and Kevin C. Young. A volumetric framework for quantum computer benchmarks, 2019. URL https://arxiv.org/abs/1904.05546.

[14] 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, 2017. ISSN 0027-8424. doi:10.1073/pnas.1618020114. URL https://www.pnas.org/content/114/13/3305.

[15] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C. Benjamin, and Xiao Yuan. Quantum computational chemistry. Rev. Mod. Phys., 92:015003, March 2020. doi:10.1103/RevModPhys.92.015003. URL https://link.aps.org/doi/10.1103/RevModPhys.92.015003.

[16] Alexander J. McCaskey, Zachary P. Parks, Jacek Jakowski, Shirley V. Moore, Titus D. Morris, Travis S. Humble, and Raphael C. Pooser. Quantum chemistry as a benchmark for near-term quantum computers. npj Quantum Information, 5(1):99, 2019. ISSN 2056-6387. doi:10.1038/s41534-019-0209-0. URL https://doi.org/10.1038/s41534-019-0209-0.

[17] Pierre-Luc Dallaire-Demers, Michal Stęchly, Jerome F. Gonthier, Ntwali Toussaint Bashige, Jonathan Romero, and Yudong Cao. An application benchmark for fermionic quantum simulations. 2020. URL https://arxiv.org/abs/2003.01862.

[18] E. F. Dumitrescu, A. J. McCaskey, G. Hagen, G. R. Jansen, T. D. Morris, T. Papenbrock, R. C. Pooser, D. J. Dean, and P. Lougovski. Cloud quantum computing of an atomic nucleus. Phys. Rev. Lett., 120:210501, May 2018. doi:10.1103/PhysRevLett.120.210501. URL https://link.aps.org/doi/10.1103/PhysRevLett.120.210501.

[19] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, et al. Hartree-fock on a superconducting qubit quantum computer, 2020. URL https://arxiv.org/abs/2004.04174.

[20] Vedran Dunjko and Hans J Briegel. Machine learning & artificial intelligence in the quantum domain: a review of recent progress. Reports on Progress in Physics, 81(7):074001, June 2018. doi:10.1088/1361-6633/aab406. URL https://doi.org/10.1088%2F1361-6633%2Faab406.

[21] Marcello Benedetti, Delfina Garcia-Pintos, Oscar Perdomo, Vicente Leyton-Ortega, Yunseong Nam, and Alejandro Perdomo-Ortiz. A generative modeling approach for benchmarking and training shallow quantum circuits. npj Quantum Information, 5(1):45, 2019. ISSN 2056-6387. doi:10.1038/s41534-019-0157-8. URL https://doi.org/10.1038/s41534-019-0157-8.

[22] Kathleen E. Hamilton and Raphael C. Pooser. Error-mitigated data-driven circuit learning on noisy quantum hardware. 2019. URL https://arxiv.org/abs/1911.13289.

[23] Kathleen E. Hamilton, Eugene F. Dumitrescu, and Raphael C. Pooser. Generative model benchmarks for superconducting qubits. Phys. Rev. A, 99:062323, June 2019. doi:10.1103/PhysRevA.99.062323. URL https://link.aps.org/doi/10.1103/PhysRevA.99.062323.

[24] Madita Willsch, Dennis Willsch, Fengping Jin, Hans De Raedt, and Kristel Michielsen. Benchmarking the quantum approximate optimization algorithm, 2019. URL https://arxiv.org/abs/1907.02359.
[25] Andreas Bengtsson, Pontus Vikstål, Christopher Warren, Marika Svensson, et al. Quantum approximate optimization of the exact-cover problem on a superconducting quantum processor, 2019. URL https://arxiv.org/abs/1912.10495.

[26] G. Pagano, A. Bapat, P. Becker, K. S. Collins, A. De, P. W. Hess, H. B. Kaplan, A. Kyprianidis, W. L. Tan, C. Baldwin, L. T. Brady, A. Deshpande, F. Liu, S. Jordan, A. V. Gorshkov, and C. Monroe. Quantum approximate optimization of the long-range Ising model with a trapped-ion quantum simulator, 2019. URL https://arxiv.org/abs/1906.02700.

[27] John Preskill. Quantum computing and the entanglement frontier, 2012. URL https://arxiv.org/abs/1203.5813.

[28] Aram W. Harrow and Ashley Montanaro. Quantum computational supremacy. Nature, 549(7671):203–209, September 2017. ISSN 1476-4687. doi:10.1038/nature23458. URL http://dx.doi.org/10.1038/nature23458.

[29] 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, June 2018. ISSN 1745-2481. doi:10.1038/s41567-018-0124-x. URL https://doi.org/10.1038/s41567-018-0124-x.

[30] Dominic W. Berry, Graeme Ahokas, Richard Cleve, and Barry C. Sanders. Efficient quantum algorithms for simulating sparse Hamiltonians. Communications in Mathematical Physics, 270(2):359–371, Mar 2007. ISSN 1432-0916. doi:10.1007/s00220-006-0150-x. URL https://doi.org/10.1007/s00220-006-0150-x.

[31] 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, July 2014. ISSN 2041-1723. doi:10.1038/ncomms5213. URL https://doi.org/10.1038/ncomms5213.

[32] Jonathan Romero, Ryan Babbush, Jarrod R McClean, Cornelius Hempel, Peter J Love, and Alán Aspuru-Guzik. Strategies for quantum computing molecular energies using the unitary coupled cluster ansatz. Quantum Science and Technology, 4(1):014008, October 2018. doi:10.1088/2058-9565/aad3e4. URL https://iopscience.iop.org/article/10.1088/2058-9565/aad3e4.

[33] Sukin Sim, Peter D. Johnson, and Alán Aspuru-Guzik. Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. Advanced Quantum Technologies, 2(12):1900070, 2019. doi:10.1002/qute.201900070. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/qute.201900070.

[34] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M. Chow, and Jay M. Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. Nature, 549(7671):242–246, September 2017. ISSN 1476-4687. doi:10.1038/nature23879. URL https://doi.org/10.1038/nature23879.

[35] Brian Coyle, Daniel Mills, Vincent Danos, and Elham Kashefi. The born supremacy: Quantum advantage and training of an Ising Born machine. 2019. URL https://arxiv.org/abs/1904.02214.

[36] Yuxuan Du, Min-Hsiu Hsieh, Tongliang Liu, and Dacheng Tao. The expressive power of parameterized quantum circuits, 2018. URL https://arxiv.org/abs/1810.11922.

[37] Thomas Hubregtsen, Josef Pichlmeier, and Koen Bertels. Evaluation of parameterized quantum circuits: on the design, and the relation between classification accuracy, expressibility and entangling capability. 2020. URL https://arxiv.org/abs/2003.09887.

[38] Andrew W. Cross, Lev S. Bishop, Sarah Sheldon, Paul D. Nation, and Jay M. Gambetta. Validating quantum computers using randomized model circuits. Phys. Rev. A, 100:032328, September 2019.
doi:10.1103/PhysRevA.100.032328. URL https://link.aps.org/doi/10.1103/PhysRevA.100.032328.

[39] Alexandru Gheorghiu, Theodoros Kapourniotis, and Elham Kashefi. Verification of quantum computation: An overview of existing approaches. *Theor. Comp. Sys.*, 63 (4):715–808, May 2019. ISSN 1432-4350. doi:10.1007/s00224-018-9872-3. URL https://doi.org/10.1007/s00224-018-9872-3.

[40] Urmila Mahadev. Classical verification of quantum computations. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 259–267. IEEE, 2018. URL http://ieeefocs.org/FOCS-2018-Papers/pdfs/59f259.pdf.

[41] Wojciech Kozlowski and Stephanie Wehner. Towards large-scale quantum networks. In *Proceedings of the Sixth Annual ACM International Conference on Nanoscale Computing and Communication*, NANOCOM ’19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368971. doi:10.1145/3345312.3345497. URL https://doi.org/10.1145/3345312.3345497.

[42] C. Greganti, T. F. Demarie, M. Ringbauer, J. A. Jones, V. Saggio, I. A. Calafell, L. A. Rozema, A. Erhard, M. Meth, L. Postler, R. Stricker, P. Schindler, R. Blatt, T. Monz, P. Walther, and J. F. Fitzsimons. Cross-verification of independent quantum devices. 2019. URL https://arxiv.org/abs/1905.09790.

[43] Daniel Mills, Anna Pappa, Theodoros Kapourniotis, and Elham Kashefi. Information theoretically secure hypothesis test for temporally unstructured quantum computation (extended abstract). *Electronic Proceedings in Theoretical Computer Science*, 266:209–221, February 2018. ISSN 2075-2180. doi:10.4204/eptcs.266.14. URL http://dx.doi.org/10.4204/EPTCS.266.14.

[44] Scott Aaronson and Lijie Chen. Complexity-theoretic foundations of quantum supremacy experiments. 2016. URL https://arxiv.org/abs/1612.05903.

[45] Nathan Wiebe, Christopher Granade, and D G Cory. Quantum bootstrapping via compressed quantum hamiltonian learning. *New Journal of Physics*, 17(2):022005, February 2015. doi:10.1088/1367-2630/17/2/022005. URL https://doi.org/10.1088%2F1367-2630%2F17%2F2%2F022005.

[46] Dan Shepherd and Michael J. Bremner. Temporally unstructured quantum computation. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 465(2105):1413–1439, 2009. doi:10.1098/rspa.2008.0443. URL https://royalsocietypublishing.org/doi/abs/10.1098/rspa.2008.0443.

[47] Michael J. Bremner, Richard Jozsa, and Dan J. Shepherd. Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 467(2126):459–472, 2011. doi:10.1098/rspa.2010.0301. URL https://royalsocietypublishing.org/doi/abs/10.1098/rspa.2010.0301.

[48] Michael J. Bremner, Ashley Montanaro, and Dan J. Shepherd. Average-case complexity versus approximate simulation of commuting quantum computations. *Phys. Rev. Lett.*, 117:080501, August 2016. doi:10.1103/PhysRevLett.117.080501. URL https://link.aps.org/doi/10.1103/PhysRevLett.117.080501.

[49] Michael J. Bremner, Ashley Montanaro, and Dan J. Shepherd. Achieving quantum supremacy with sparse and noisy commuting quantum computations. *Quantum*, 1:8, April 2017. ISSN 2521-327X. doi:10.22331/q-2017-04-25-8. URL https://doi.org/10.22331/q-2017-04-25-8.

[50] Richard M. Karp and Richard J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, STOC ’80, page 302–309, New York, NY, USA, 1980. Association for Computing Machinery. ISBN 0897910176.
doi:10.1145/800141.804678. URL https://doi.org/10.1145/800141.804678.

[51] J. Misra and David Gries. A constructive proof of vizing’s theorem. Information Processing Letters, 41(3):131 – 133, 1992. ISSN 0020-0190(92)90041-S. URL http://www.sciencedirect.com/science/article/pii/0020019092900415.

[52] Juan Bermejo-Vega, Dominik Hangleiter, Martin Schwarz, Robert Raussendorf, and Jens Eisert. Architectures for quantum simulation showing a quantum speedup. Phys. Rev. X, 8:021010, April 2018. doi:10.1103/PhysRevX.8.021010. URL https://link.aps.org/doi/10.1103/PhysRevX.8.021010.

[53] D Hangleiter, M Kliesch, M Schwarz, and J Eisert. Direct certification of a class of quantum simulations. Quantum Science and Technology, 2(1):015004, February 2017. doi:10.1088/2058-9565/2/1/015004. URL https://doi.org/10.1088%2F2058-9565%2F2%2F1%2F015004.

[54] Adam Bouland, Bill Fefferman, Chinmay Nirkhe, and Umesh Vazirani. Quantum supremacy and the complexity of random circuit sampling. 2018. URL https://arxiv.org/abs/1803.04402.

[55] Ramis Movassagh. Efficient unitary paths and quantum computational supremacy: A proof of average-case hardness of random circuit sampling. 2018. URL https://arxiv.org/abs/1810.04681.

[56] Michael A. Nielsen and Isaac L. Chuang. Quantum computation and quantum information: 10th anniversary edition, 2010.

[57] Alexander Cowtan, Silas Dilkes, Ross Duncan, Will Simmons, and Seyon Sivarajah. Phase gadget synthesis for shallow circuits. Electronic Proceedings in Theoretical Computer Science, 318:214–229, April 2020. ISSN 2075-2180. doi:10.4204/eptcs.318.13. URL http://dx.doi.org/10.4204/EPTCS.318.13.

[58] Panagiotis Kl. Barkoutsos, Jerome F. Gonthier, Igor Sokolov, Nikolaj Moll, Gian Salis, Andreas Fuhrer, Marc Ganzhorn, Daniel J. Egger, Matthias Troyer, Antonio Mezzacapo, Stefan Filipp, and Ivano Tavernelli. Quantum algorithms for electronic structure calculations: Particle-hole hamiltonian and optimized wave-function expansions. Phys. Rev. A, 98:022322, August 2018. doi:10.1103/PhysRevA.98.022322. URL https://link.aps.org/doi/10.1103/PhysRevA.98.022322.

[59] C. E. Porter and R. G. Thomas. Fluctuations of nuclear reaction widths. Phys. Rev., 104:483–491, October 1956. doi:10.1103/PhysRev.104.483. URL https://link.aps.org/doi/10.1103/PhysRev.104.483.

[60] C. Neill, P. Roushan, K. Kechedzhi, S. Boixo, et al. A blueprint for demonstrating quantum supremacy with superconducting qubits. Science, 360(6385):195–199, 2018. ISSN 0036-8075. doi:10.1126/science.aao4309. URL https://science.sciencemag.org/content/360/6385/195.

[61] Sergio Boixo, Vadim N. Smelyanskiy, and Hartmut Neven. Fourier analysis of sampling from noisy chaotic quantum circuits. 2017. URL https://arxiv.org/abs/1708.01875.

[62] Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics. In Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing, STOC ’11, page 333–342, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306911. doi:10.1145/1993636.1993682. URL https://doi.org/10.1145/1993636.1993682.

[63] Edwin Pednault, John A. Gunnels, Giacomo Nannicini, Lior Horesh, and Robert Wisnieff. Leveraging secondary storage to simulate deep 54-qubit sycamore circuits. 2019. URL https://arxiv.org/abs/1910.09534.

[64] Benjamin Villalonga, Sergio Boixo, Bron Nelson, Christopher Hene, Eleanor Rieffel, Rupak Biswas, and Salvatore Mandrà. A flexible high-performance simulator for verifying
and benchmarking quantum circuits implemented on real hardware. npj Quantum Information, 5(1):86, 2019. ISSN 2056-6387. doi:10.1038/s41534-019-0196-1. URL https://doi.org/10.1038/s41534-019-0196-1.

[65] Benjamin Villalonga, Dmitry Lyakh, Sergio Boixo, Hartmut Neven, Travis S Humble, Rupak Biswas, Eleanor G Rieffel, Alan Ho, and Salvatore Mandrà. Establishing the quantum supremacy frontier with a 281 piflop/s simulation. Quantum Science and Technology, 5(3):034003, apr 2020. doi:10.1088/2058-9565/ab7eeb. URL https://doi.org/10.1088%2F2058-9565%2Fab7eeb.

[66] pytket documentation. URL https://cqcl.github.io/pytket/build/html/index.html.

[67] qiskit documentation. URL https://qiskit.org/documentation/.

[68] Sumeet Khatri, Ryan LaRose, Alexander Poremba, Lukasz Cincio, Andrew T. Sornborger, and Patrick J. Coles. Quantum-assisted quantum compiling. Quantum, 3:140, May 2019. ISSN 2521-327X. doi:10.22331/q-2019-05-13-140. URL https://doi.org/10.22331/q-2019-05-13-140.

[69] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T. Chong, and Margaret Martonosi. Scaffold: A framework for compilation and analysis of quantum computing programs. In Proceedings of the 11th ACM Conference on Computing Frontiers, CF '14, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450326833. doi:10.1145/2578917.2578939. URL https://doi.org/10.1145/2578917.2578939.

[70] Aram W. Harrow, Benjamin Recht, and Isaac L. Chuang. Efficient discrete approximations of quantum gates. Journal of Mathematical Physics, 43(9):4445–4451, 2002. doi:10.1063/1.1495899. URL https://doi.org/10.1063/1.1495899.

[71] Joel J. Wallman and Joseph Emerson. Noise tailoring for scalable quantum computation via randomized compiling. Phys. Rev. A, 94:052325, November 2016. doi:10.1103/PhysRevA.94.052325. URL https://link.aps.org/doi/10.1103/PhysRevA.94.052325.

[72] Jeff Heckey, Shruti Patil, Ali JavadiAbhari, Adam Holmes, Daniel Kudrow, Kenneth R. Brown, Diana Franklin, Frederic T. Chong, and Margaret Martonosi. Compiler management of communication and parallelism for quantum computation. In Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS ’15, page 445–456, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450328357. doi:10.1145/2694344.2694357. URL https://doi.org/10.1145/2694344.2694357.

[73] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS ’19, page 1015–1029, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362405. doi:10.1145/3297858.3304075. URL https://doi.org/10.1145/3297858.3304075.

[74] Ali J Abhari, Arvin Faruque, Mohammad J Dousti, Lukas Svec, Oana Catu, Amlan Chakrabati, Che-Fu Chiang, Seth Vanderwilt, John Black, and Fred Chong. Scaffold: Quantum programming language. Technical report, PRINCETON UNIV NJ DEPT OF COMPUTER SCIENCE, 2012. URL https://www.cs.princeton.edu/research/techreps/TR–934-12.

[75] Alexander J McCaskey, Dmitry I Lyakh, Eugene F Dumitrescu, Sarah S Powers, and Travis S Humble. XACC: a system-level software infrastructure for heterogeneous quantum–classical computing. Quantum Science and Technology, 5(2):024002, February
[76] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. On the Qubit Routing Problem. In Wim van Dam and Laura Mancinska, editors, 14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019), volume 135 of Leibniz International Proceedings in Informatics (LIPIcs), pages 5:1–5:32, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-112-2. doi:10.4230/LIPIcs.TQC.2019.5. URL http://drops.dagstuhl.de/opus/volltexte/2019/10397.

[77] Prakash Murali, David C. Mckay, Margaret Martonosi, and Ali Javadi-Abhari. Software mitigation of crosstalk on noisy intermediate-scale quantum computers. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS ’20, page 1001–1016, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371025. doi:10.1145/3373376.3378477. URL https://doi.org/10.1145/3373376.3378477.

[78] IBM Quantum Experience User Guide. Advanced single-qubit gates, 2019. URL https://quantum-computing.ibm.com/support/guides/user-guide.

[79] Christopher Chamberland, Guanyu Zhu, Theodore J. Yoder, Jared B. Hertzberg, and Andrew W. Cross. Topological and subsystem codes on low-degree graphs with flag qubits. Phys. Rev. X, 10:011022, January 2020. doi:10.1103/PhysRevX.10.011022. URL https://link.aps.org/doi/10.1103/PhysRevX.10.011022.

[80] Jerry Chow and Jay Gambetta. Quantum takes flight: Moving from laboratory demonstrations to building systems, 2020. URL https://www.ibm.com/blogs/research/2020/01/quantum-volume-32/.

[81] Iskren Vankov, Daniel Mills, Petros Wallden, and Elham Kashefi. Methods for classically simulating noisy networked quantum architectures. Quantum Science and Technology, 5 (1):014001, November 2019. doi:10.1088/2058-9565/ab54a4. URL https://doi.org/10.1088%2F2058-9565%2Fab54a4.

[82] 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, C. Ryan-Anderson, and B. Neyenhuis. Demonstration of the qccd trapped-ion quantum computer architecture, 2020. URL https://arxiv.org/abs/2003.01293.

[83] Joseph Emerson, Yaakov S. Weinstein, Marcos Saraceno, Seth Lloyd, and David G. Cory. Pseudo-random unitary operators for quantum information processing. Science, 302(5653):2098–2100, 2003. ISSN 0036-8075. doi:10.1126/science.1090790. URL https://science.sciencemag.org/content/302/5653/2098.

[84] Joseph Emerson, Etera Livine, and Seth Lloyd. Convergence conditions for random quantum circuits. Phys. Rev. A, 72:060302, December 2005. doi:10.1103/PhysRevA.72.060302. URL https://link.aps.org/doi/10.1103/PhysRevA.72.060302.

[85] Fernando G. S. L. Brandão, Aram W. Harrow, and Michal Horodecki. Local random quantum circuits are approximate polynomial-designs. Communications in Mathematical Physics, 346(2):397–434, September 2016. ISSN 1432-0916. doi:10.1007/s00220-016-2706-8. URL https://doi.org/10.1007/s00220-016-2706-8.

[86] Andrew Fagan and Ross Duncan. Optimising clifford circuits with quantomatic. Electronic Proceedings in Theoretical Computer Science, 287:85–105, January 2019. ISSN 2075-2180. doi:10.4204/eptcs.287.5. URL http://dx.doi.org/10.4204/EPTCS.287.5.

[87] M Blaauboer and R L de Visser. An analytical decomposition protocol for optimal implementation of two-qubit entangling
gates. *Journal of Physics A: Mathematical and Theoretical*, 41(39):395307, sep 2008. doi:10.1088/1751-8113/41/39/395307. URL https://doi.org/10.1088%2F1751-8113%2F41%2F39%2F395307.

[88] Easwar Magesan and Jay M. Gambetta. Effective hamiltonian models of the cross-resonance gate. *Phys. Rev. A*, 101:052308, May 2020. doi:10.1103/PhysRevA.101.052308. URL https://link.aps.org/doi/10.1103/PhysRevA.101.052308.

[89] Timothy Proctor, Kenneth Rudinger, Kevin Young, Mohan Sarovar, and Robin Blume-Kohout. What randomized benchmarking actually measures. *Phys. Rev. Lett.*, 119:130502, September 2017. doi:10.1103/PhysRevLett.119.130502. URL https://link.aps.org/doi/10.1103/PhysRevLett.119.130502.

[90] 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, January 2008. doi:10.1103/PhysRevA.77.012307. URL https://link.aps.org/doi/10.1103/PhysRevA.77.012307.

[91] Arnaud Carignan-Dugas, Kristine Boone, Joel J. Wallman, and Joseph Emerson. From randomized benchmarking experiments to gate-set circuit fidelity: how to interpret randomized benchmarking decay parameters. *New Journal of Physics*, 20(9):092001, September 2018. doi:10.1088/1367-2630/aadcc7. URL https://doi.org/10.1088%2F1367-2630%2Faadcc7.

[92] Timothy J. Proctor, Arnaud Carignan-Dugas, Kenneth Rudinger, Erik Nielsen, Robin Blume-Kohout, and Kevin Young. Direct randomized benchmarking for multiqubit devices. *Phys. Rev. Lett.*, 123:030503, July 2019. doi:10.1103/PhysRevLett.123.030503. URL https://link.aps.org/doi/10.1103/PhysRevLett.123.030503.

[93] Jay M. Gambetta, A. D. Córcoles, S. T. Merkel, B. R. Johnson, John A. Smolin, Jerry M. Chow, Colm A. Ryan, Chad Rigetti, S. Poletto, Thomas A. Ohki, Mark B. Ketchen, and M. Steffen. Characterization of addressability by simultaneous randomized benchmarking. *Phys. Rev. Lett.*, 109:240504, December 2012. doi:10.1103/PhysRevLett.109.240504. URL https://link.aps.org/doi/10.1103/PhysRevLett.109.240504.

[94] David C. McKay, Andrew W. Cross, Christopher J. Wood, and Jay M. Gambetta. Correlated randomized benchmarking, 2020. URL https://arxiv.org/abs/2003.02354.

[95] 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.

[96] Easwar Magesan, Jay M. Gambetta, and Joseph Emerson. Characterizing quantum gates via randomized benchmarking. *Phys. Rev. A*, 85:042311, April 2012. doi:10.1103/PhysRevA.85.042311. URL https://link.aps.org/doi/10.1103/PhysRevA.85.042311.

[97] A. D. Córcoles, Jay M. Gambetta, Jerry M. Chow, John A. Smolin, Matthew Ware, Joel Strand, B. L. T. Plourde, and M. Steffen. Process verification of two-qubit quantum gates by randomized benchmarking. *Phys. Rev. A*, 87:030301, March 2013. doi:10.1103/PhysRevA.87.030301. URL https://link.aps.org/doi/10.1103/PhysRevA.87.030301.

[98] David C. McKay, Sarah Sheldon, John A. Smolin, Jerry M. Chow, and Jay M. Gambetta. Three-qubit randomized benchmarking. *Phys. Rev. Lett.*, 122:200502, May 2019. doi:10.1103/PhysRevLett.122.200502. URL https://link.aps.org/doi/10.1103/PhysRevLett.122.200502.
The exponential distribution, with rate $\lambda$, is a probability distribution with the probability density function

$$\Pr(x) = \lambda e^{-\lambda x}.$$  

This is the distribution of waiting times between events in a Poisson process. We are concerned with showing that output probabilities of the circuits classes considered here are exponentially distributed. Such a property is a signature of quantum chaos, and that a class of circuits is approximately Haar random [29, 83, 84]. It also allows for the calculation of both the ideal value of the cross-entropy discussed in Section 3.2, and the ideal heavy output probability as discussed in Section 3.1. This in turn allows us to fully exploit Cross-Entropy Benchmarking and Heavy Output Generation Benchmarking. Here we will argue numerically which of the circuits we introduce in Section 2 generate output probabilities of this form, and discuss the implications when they do not.

We also demonstrate why the circuit depths used in Section 2 are necessary to generate output probabilities of this form. To do this we generate 100 circuits of each type and number of layers, where a layer is as defined in the respective Algorithms of Section 2. We then calculate the ideal output probabilities using classical simulation and compare this distribution of output probabilities to the exponential distribution. In the case of square circuits and deep circuits, we notice a better approximation of the exponential distribution by the distribution of output probabilities, measured by the $\ell_1$-norm distance between the two, as the number of layers increases. We can use this to isolate the number of layers at which the difference approaches its minimum.

A.1 Square Circuits

The exponential form of the distribution of the output probabilities from random circuits similar to square circuits has been established [29, 44]. As the procedure we use to generate square circuits, seen in Algorithm 2, differs slightly from that used for other similar random circuits [29, 38, 44], we explore the distribution of its output probabilities here.

The relevant results are seen in Figure 12. In particular, it can be seen from Figure 12b that the minimum value of $\ell_1$-norm distance between the distribution of output probabilities and the exponential distribution is approached at a number of layers equal to the number of qubits, justifying our choice of layer numbers in Algorithm 2. It may be that asymptotically the number of layers required is sub-linear [29], although for the circuit sizes used here a linear growth in depth is appropriate. Figure 12a illustrates the closeness of fit of the two distributions.

A.2 Deep Circuits

Unlike with square circuits, there is no precedent for utilising deep circuits to generate exponentially distributed output probabilities, as we do here. This allows us to use deep circuits as a uniquely insightful benchmark of the performance of quantum computing stacks, grounded both in the theoretical results of Section 3, and in pertinent applications.

The relevant results are seen in Figure 13. In particular, it can be seen from Figure 13b that the minimum value of $\ell_1$-norm distance between the distribution of output probabilities and the exponential distribution is approached at a number of layers equal to three times the number of qubits, plus one, justifying our choice of layer numbers in Algorithm 3. Figure 13a illustrates the closeness of fit of the two distributions.

The depth required to achieve an exponential distribution of outcome probabilities with deep circuits is greater than is the case for square circuits. Indeed, random circuits were initially introduced as the shallowest circuits required to generate such output probabilities [29]. This sacrifice in depth is made to achieve a benchmark which is uniquely application motivated, as discussed in Section 2.

A.3 Shallow Circuits

Unlike in the case of square circuits and deep circuits, the output probabilities of shallow circuits are not exponentially distributed. This is unsurprising since random circuits with this limited connectivity are thought to require at least depth
(a) The distribution of output probabilities from a circuit $C$, where $C$ is a 5 qubit circuit, from the square circuits class as defined in Algorithm 2.

(b) The $\ell_1$-norm distance between the distribution of output probabilities of square circuits and the exponential distribution $2^n e^{-2^n x}$, where $n$ is the number of qubits. A layer is defined as in Algorithm 2. Colours correspond to numbers of qubits in the following way: 2 [ ], 3 [ ], 4 [ ], 5 [ ].

Figure 12: Exponential distribution fitting data for square circuits.

(a) The distribution of output probabilities from a circuit $C$, where $C$ is a 5 qubit circuit, from the deep circuits class as defined in Algorithm 3.

(b) The $\ell_1$-norm distance between the distribution of output probabilities of deep circuits and the exponential distribution $2^n e^{-2^n x}$, where $n$ is the number of qubits. A layer is defined as in Algorithm 3. Colours correspond to numbers of qubits in the following way: 2 [ ], 3 [ ], 4 [ ], 5 [ ].

Figure 13: Exponential distribution fitting data for deep circuits.
\(O(\sqrt{n})\) to create such a feature [52, 54, 85]. This has the unfortunate side effect that the results of Section 3.2 do not apply, and so Cross-Entropy Benchmarking cannot be used.

While it is also true that the predictions made about the ideal heavy output probability, as discussed in Section 3.1, also do not apply, a study of the heavy output probability is still of interest. In particular, while we cannot connect the benchmark to the HOG problem of Problem 1, we can compare the probability of generating heavy outputs to the ideal probability of producing heavy outputs, as calculated by classical simulation.

B Compilation Strategies

This section details the compilation strategies explored in each of our experiments. For the circuit families and figures of merit investigated here, the compilation strategies we used were designed and empirically confirmed to perform well at the compilation tasks at hand. The version of each package used are listed in Table 2.

**noise-unaware pytket and noise-aware pytket**
The noise-unaware pytket and noise-aware pytket compilation strategies are generated using Algorithm 4. noise-unaware pytket is generated by passing False as input to Algorithm 4, and noise-aware pytket by passing True.

Of particular interest are the following functions:

- **OptimiseCliffords**: Simplifies Clifford gate sequences [86].
- **KAKDecomposition**: Identifies two-qubit subcircuits with more than 3 CXs and reduces them via the KAK/Cartan decomposition [87].
- **route**: Modifies the circuit to satisfy the architectural constraints [76]. This will introduce SWAP gates.
- **noise_aware_placement**: Selects initial qubit placement taking in to account reported device gate error rates [10].
- **line_placement**: Attempts to place qubits next to those they interact with in the first few time slices. This does not take device error rates into account.

**Algorithm 4 pytket compilation strategies.** The passes listed here are named as in the documentation for pytket [66], where additional detail on their actions can be found.

```
Input: noise_aware ∈ \{True, False\}

1: OptimiseCliffords
2: KAKDecomposition
3: RebaseToRzRx \(\triangleright\) Convert to IBM gate set
4: CommuteRzRxThroughCX
5: if noise_aware then
6:     noise_aware_placement
7:     else
8:         line_placement
9:     end if
10: route
11: decompose_SWAP_to_CX \(\triangleright\) Orientate CX to coupling map
12: redirect_CX_gates
13: OptimisePostRouting \(\triangleleft\) Optimisation preserving placement and orientation
```

**noise-unaware Qiskit and noise-aware Qiskit**
The noise-unaware Qiskit and noise-aware Qiskit compilation strategies, as defined in Algorithm 5, are heavily inspired by level_3_passmanager, a preconfigured compilation strategy made available in Qiskit. noise-unaware Qiskit is generated by passing noise_aware as False in Algorithm 4, and noise-aware Qiskit by passing True.

Where possible we passed stochastic as True in order to use StochasticSwap instead of BasicSwap during the swap mapping pass. In general, StochasticSwap generates circuits with lower depth; however, for the versions listed in Table 2, it proved faulty for some circuit sizes and device coupling maps used in this work. StochasticSwap may also result in repeated measurement of the same qubit, which cannot be implemented. Repeated compilation attempts may therefore be necessary,
| Package       | Version |
|--------------|---------|
| Qiskit [1, 67] | 0.12.0  |
| pytket [10, 66] | 0.3.0   |

Table 2: Packages used in this work, and their corresponding versions.

and if this fails the circuit is not included in the plots of Section 5.

Of particular note are the following functions:

**NoiseAdaptiveLayout:** Selects initial qubit placement based on minimising readout error rates [73].

**DenseLayout:** Chooses placement by finding the most connected subset of qubits.

**Unroller:** Decomposes unitary operation to desired gate set.

**StochasticSwap:** Adds SWAP gates to adhere to coupling map using a randomised algorithm.

**BasicSwap:** Produces a circuit adhering to coupling map using a simple rule: CX gates in the circuit which are not supported by the hardware are preceded with necessary SWAP gates.

**only pytket routing** In this case we perform, in the order as listed, the pytket operations: `route`, `decompose_SWAP_to_CX`, and `redirect_CX_gates`. We then account for the architecture gate set, without any further optimisation.

### C Device Data

Two device properties leveraged by our compilation strategies are the coupling maps, describing the connectivity of the qubits and in which directions CX gates can be performed, and the calibration information, describing the noise levels of the device. These properties, and devices noise levels in particular, are considered valuable benchmarks of the performance of the device in their own right.

These properties are collectively influential in noise-aware compiling, as detailed in Appendix B.

**Algorithm 5** Qiskit compilation strategies. The passes listed here are named as in the documentation for Qiskit [67], where additional detail on their actions can be found.

```plaintext
Input:
  - noise_aware ∈ {True, False}
  - stochastic ∈ {True, False}

1: Unroller
2: if noise_aware then
3: NoiseAdaptiveLayout
4: else
5: DenseLayout
6: end if
7: AncillaAllocation ▷ Assign idle qubits as ancillas
8: if stochastic then
9: StochasticSwap
10: else
11: BasicSwap
12: end if
13: Decompose(SwapGate) ▷ Decompose SWAP to CX
14: CXDirection ▷ Orientate CX to coupling map
15: Collect2qBlocks ▷ Gather 2 qubit blocks
16: ConsolidateBlocks
17: Unroller ▷ Unroll two-qubit blocks
18: Optimize1qGates ▷ Combine chains of one-qubit gates
19: CXDirection
20: Collect2qBlocks
21: ConsolidateBlocks
22: Unroller ▷ Unroll two-qubit blocks
23: Optimize1qGates ▷ Combine chains of one-qubit gates
24: CXDirection
25: CXDirection
```
There circuits are compiled to adhere to the device’s coupling map, while also aiming to minimise some function of the calibration information. Because full quantum computing stack holistic benchmarking encompasses the circuit compilation strategies, it provides a novel way of using device information to benchmark an entire system, instead of simply the physical qubits which comprise it.

C.1 Device Coupling Maps

A coupling map of a device is a graphical representation of how two-qubit gates can be applied across the device. In this representation, each qubit is represented by a vertex, with directed edges joining qubits between which a two-qubit gate can be applied. For the devices considered here, this two-qubit gate is a CX gate, implemented using the cross-resonance interaction of transmon qubits [88]. The direction of the edge is from the control to the target qubit of the CX gate, with bi-directional edges indicating that both qubits can be used as either the control or target. The coupling maps of the devices investigated in this work are shown in Figure 14. For those devices all edges are bi-directional, although this is not typical when the asymmetric CX is employed.

As discussed in Section 4, a trade-off exists between the connectivity of the device and the number of two-qubit gates necessary to implement a given circuit. More highly connected coupling maps typically require fewer two-qubit gates to implement a fixed unitary than less connected ones, owing to the reduced need for SWAP gates to account for discrepancies between the coupling maps of the uncompiled circuit and the device. While this reduced depth can reduce the impact of time based noise channels, this is counterbalanced by the higher levels of cross-talk experienced by qubits corresponding to vertices with high degree in the device’s coupling map [79].

C.2 Device Calibration Information

The noise-aware tools employed by the compilation strategies explored in this work consider three kinds of errors which can occur, namely: readout error, single-qubit gate error, and two-qubit gate error. For the devices provided through IBM Quantum, this information is contained in calibration data which is accessible using tools in the Qiskit library, and is updated twice daily. The experiments in this paper were conducted between 2020-01-29 and 2020-02-10 with the calibration data in Figure 15 and Figure 16 aggregated over this time period.

An assignment or readout error corresponds to an incorrect reading of the state of the qubit; for example, returning “0” when the proper label is “1”, or vice-versa. The probability of incorrectly labelling the qubit is called the readout error, denoted, $\epsilon_a$, and is calculated as

$$\epsilon_a = \frac{\Pr(\text{"0"} | \text{[1]}) + \Pr(\text{"1"} | \text{[0]})}{2}$$

$\epsilon_a$ is estimated by repeatedly preparing a qubit in a known state, immediately measuring it, and then counting the number of times the measurement returns the wrong label. This value, for the devices explored in this paper, is reported in Figure 15a.

Errors affecting the gates of the device correspond to an incorrect operation applied by the device. There are many ways to quantify the effect of this error, with IBM Quantum’s devices reporting randomized benchmarking (RB) numbers [89, 90]. The RB number, $\epsilon^C$, is estimated by running many self-inverting Clifford circuits, consisting of $m$ layers of gates drawn from the $n$-qubit Clifford group, inverted at layer $m + 1$. The survival probability, which is the probability the input state is unchanged, can then be estimated. Under a broad set of noise models and assumptions [89, 91], this survival probability can be shown to decay exponentially with $m$. Consequently, it can be estimated by fitting a decay curve of the form $A p^m + B$. The RB number is related to $p \in [0,1]$, called the depolarisation/decay rate, by

$$\epsilon^C = (1 - p) (1 - 1/D),$$

where $D = 2^n$, and $n$ is the number of qubits acted on by the Clifford gates. $\epsilon^C$, which is also referred to as the error per Clifford of the device, is minimised at $p = 1$, in which case the survival probability is constant and set by the state preparation and measurement errors.

The Clifford gates necessary for RB must be compiled to the native gate set of the device. Using an estimate of $\epsilon^C$, an estimate of the error per gate, $\tilde{\epsilon}_G$, for a gate $G$, can be obtained by multiplying $\epsilon^C$ by
Figure 14: Coupling maps of the devices studied in this work. Vertices, represented by blue circles, correspond to qubits, while edges are directed from the control to the target qubits of permitted two-qubit gates.

a factor related to the average number of uses of $G$ when implementing a random Clifford operation:

$$
\epsilon_G^C \sim \epsilon_G \times \# \text{ uses of } G \text{ per Clifford.}
$$

Values for $\epsilon_U^2$, the error per gate for $U_2$ gates, can be found in Figure 15b, and $\epsilon_X^C$, that for $C_X$ gates, in Figure 16. The commonly reported average fidelity for $U_3$ gates is $1 - (1 - \epsilon_U^2)^2$.

There are many variants of randomized benchmarking, such as direct RB [92], simultaneous RB [93], and correlated RB [94]. For details on the randomized benchmarking protocol used by IBM Quantum, see [93, 95–98].

The experiments necessary for cross-entropy benchmarking may themselves also be used to estimate a depolarization rate in a similar way to RB [3]. Instead of using random Clifford circuits, however, the random circuits are run. Under the assumption that the action of a random circuit can be described using a depolarizing error model (with equal-probability Pauli errors), then the Pauli error, $\epsilon^p$, can be estimated as

$$
\epsilon^p = (1 - p)(1 - 1/D^2).
$$

Here, $p$ is the depolarization rate of the survival probability under the action of random circuits, estimated as above. Interestingly, $\epsilon^p$ can be estimated using single and two-qubit RB information.

Several important noise channels, most notably cross-talk, are not included in the device calibration data. As shown in Section 5, the effects of this noise can be inferred through the application-motivated benchmarks we introduce in this work, by showing the trade-off between connectivity of the device and cross-talk [79].

D Empirical Relationship Between Heavy Output Generation Probability and L1 Distance

As discussed in Section 3.4, the theoretical foundations for believing that implementing shallow circuits to within a fixed $\ell_1$-norm distance constitutes a demonstration of quantum computational supremacy are stronger than for implementations with high heavy output generation probability. That being said, Figure 3 and Figure 11 contain similar features. For example, ibmq_16_melbourne consistently performs the worst, with ibmq_singapore and ibmq_ourense performing the best in both figures of merit. An interesting question, then, is how these two figures of merit generally relate to one another.

If the $\ell_1$-norm distance was 0, the experimental outcome frequencies would equal the ideal outcome probabilities. Consequently, the heavy output probabilities would be the same between the device and an ideal quantum computer. Because the heavy output probability depends on the circuit in question, when examining the empirical relationship between $\ell_1$-norm distance and heavy output probability, it is useful to normalize the latter by the heavy output probability of an ideally-implemented circuit. We define the normalised heavy output generation probability as the ratio of the heavy output probability of the device and the heavy output probability from an ideally implemented circuit. Therefore if the $\ell_1$-norm distance was 0, the normalised heavy output generation probability would be 1.
(a) **Average readout error.** The readout error is the probability the state of a given qubit is incorrectly labelled.

(b) **Average error per $U_2$ gate.** The error per gate is a measure of how accurately the $U_2$ gate is applied.

Figure 15: **Error per single qubit operations on the devices used in this work.** Bars indicate the average error rates; error bars are one standard deviation. Data aggregated based on calibration data collected over the course of our experiments. Devices shown here are: ibmqx2 [■], ibmq_ourense [■], ibmq_singapore [■], ibmq_16_melbourne [■]. A logarithmic scale is used.
Figure 16: Average error per CX operation on the devices used in this work. The error per CX gate is a measure of how accurately the CX gate is applied. Bars indicate the average error rates; error bars are one standard deviation. Data aggregated based on calibration data collected over the course of our experiments. Devices shown here are: ibmqx2, ibmq_ourense, ibmq_singapore, ibmq_16_melbourne. A logarithmic scale is used.
As the $\ell_1$-norm distance increases, the experimental frequencies increasingly differ from the ideal outcome probabilities. Two things then may happen: heavy outputs are produced more regularly, in which case the normalised heavy output generation probability will grow above $1$; or less regularly, in which case the normalised heavy output generation probability will fall below $1$. In practice, we expect the distribution produced by the device to converge to the uniform one over all bit strings as the noise increases, so we expect the normalised heavy output generation probability to fall with increasing $\ell_1$-norm distance.

The empirical relationship between the normalised heavy output generation probability and $\ell_1$-norm distance is shown in Figure 17. For each circuit, Figure 17 plots the $\ell_1$-norm distance of the distribution produced by a real device against the normalised heavy output generation probability. As expected, a negative correlation exists between these two figures of merit. For the deepest circuits, and in particular the widest circuits from the deep circuits class, the cluster of points can be seen to indicate that the the normalised heavy output generation probability falls more slowly as the $\ell_1$-norm distance becomes larger. This is because the minimum value of heavy output generation probability is being reached, which is to say that the output distribution from the real device has converged to the uniform one, while more detail can be extracted by considering the $\ell_1$-norm distance.

This correlation is encouraging as, in the regime where it becomes impossible to calculate the $\ell_1$-norm distance, we can be justified in believing that the correlation between the features present in the plot throughout this section persist. This line of reasoning is similar to that used when Cross-Entropy Benchmarking is used to predict demonstrations of quantum computational supremacy in the regime when it too becomes impossible to calculate [3]. Note also that this correlation contrasts with the knowledge that, in general, the probability of producing heavy outputs does not provide an upper bound on the $\ell_1$-norm distance [54], and reveals that in practice it may be relied upon to do so.
Figure 17: Scatter plot and linear regression line comparing the normalised heavy output generation probability and $\ell_1$-norm distance. Each point corresponds to one circuit of the class and width as labelled. Colours correspond to numbers of qubits in the following way: 2 [ ], 3 [ ], 4 [ ], 5 [ ], 6 [ ], 7 [ ].