QASMBench: A Low-Level Quantum Benchmark Suite for NISQ Evaluation and Simulation

ANG LI, SAMUEL STEIN, SRIRAM KRISHNAMOORTHY, and JAMES ANG, Pacific Northwest National Laboratory, USA

The rapid development of quantum computing (QC) in the NISQ era urgently demands a low-level benchmark suite and insightful evaluation metrics for characterizing the properties of prototype NISQ devices, the efficiency of QC programming compilers, schedulers and assemblers, and the capability of quantum simulators in a classical computer. In this work, we fill this gap by proposing a low-level, easy-to-use benchmark suite called QASMBench based on the OpenQASM assembly representation. It consolidates commonly used quantum routines and kernels from a variety of domains including chemistry, simulation, linear algebra, searching, optimization, arithmetic, machine learning, fault tolerance, cryptography, etc., trading-off between generality and usability. To analyze these kernels in terms of NISQ device execution, in addition to circuit width and depth, we propose four circuit metrics including gate density, retention lifespan, measurement density, and entanglement variance, to extract more insights about the execution efficiency, the susceptibility to NISQ error, and the potential gain from machine-specific optimizations. Most of the QASMBench application code can be launched and verified in IBM-Q directly. With the help from *q-convert*, QASMBench can be evaluated on various platforms and simulation environments. QASMBench is released at: http://github.com/pnnl/QASMBench.

CCS Concepts: • **Computer systems organization → Quantum computing**; • **Hardware → Quantum computation**; • **Software and its engineering → Software libraries and repositories**.

Additional Key Words and Phrases: Benchmark, OpenQASM, quantum metrics, NISQ

ACM Reference Format:
Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. 2021. QASMBench: A Low-Level Quantum Benchmark Suite for NISQ Evaluation and Simulation. *ACM Trans. Quantum Comput.* 1, 1 (June 2021), 22 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Quantum computing (QC) [9, 86] has been envisioned as one of the most promising computing paradigms beyond Moore's Law for tackling difficult computing challenges that are classically intractable arising from various domains like chemistry [46, 62], machine learning [13, 99], cryptography [47, 102], linear algebra [22, 52], finance [95, 116], recommendation systems [64], physics simulation [33, 40], networking [65, 82], and so on. QC relies on fundamental quantum mechanisms such as superposition, entanglement, interference, tunneling, etc. for performing computation, in hopes of significant or even exponential speedups over existing algorithms in classical computers [25, 102].

Despite holding great promise, QC in contemporary noisy-intermediate-scale-quantum (NISQ) [92] devices is still distant from outperforming classical computers regarding general problems. NISQ devices describe the near-term quantum platforms comprising 50 to hundreds of qubits,

Authors' address: Ang Li, ang.li@pnnl.gov; Samuel Stein, samuel.stein@pnnl.gov; Sriram Krishnamoorthy, sriram@pnnl.gov; James Ang, ang@pnnl.gov, Pacific Northwest National Laboratory, 902 Battelle Blvd, Richland, WA, USA, 99354.

2643-6817/2021/6-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

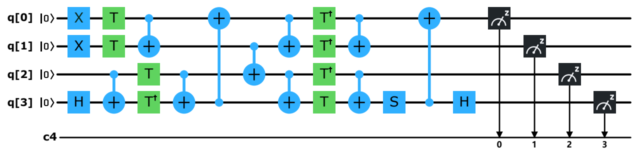

Fig. 1. A quantum adder circuit with width=4 (4 qubits) and depth=23 (23 gates) followed by measurement.

which is inadequate for full-scale error-correction but can already present promising results in a number of problems, and lay the foundations towards practical quantum demonstration [92].

The QC landscape is developing rapidly nowadays. From the software perspective, large number of QC software are developed in classical programming languages (e.g., Python [48, 105], C/C++ [59], JavaScript [55], ML [3]). A list of opensource QC projects, quantum algorithms, and quantum simulators can be found in [41], [61], and [94], respectively. From the hardware perspective, the development of QC testbeds have already proceeded to a stage where small working prototypes are available through cloud service, such as IBM-Q [54], Microsoft Azure [24], and Rigetti [96], showing encouraging results [20, 36]. Finally, from the application perspective, Google recently announced the demonstration of quantum supremacy on the task of sampling the output of a pseudo-random quantum circuit, using a 53-qubit NISQ system [5]. Several quantum machine learning frameworks (e.g., [16, 55]) and chemistry simulation packages have also been released (e.g., [55, 77]).

To close the gap between practical quantum applications and real quantum machines, contributions from the computer system and architecture community are undoubtedly critical in tacking several key challenges [19], including software and hardware verification [109], defining and perforating abstraction boundaries [84], managing parallelism and communication [70], mapping and scheduling operations [100], elevating control complexity [71], hardware-specific optimizations [83, 104, 110], investigating and mitigating noise [85, 108], etc. These massive effort, however, are currently evaluated and verified using randomly selected QC benchmarks [31, 70, 71, 83, 85, 100, 108–110], lacking the common ground for judicious analysis and fair comparison among each other. The communities thus urgently desire a low-level, easy-to-use QC benchmark suite to foster the software-hardware codesign effort for the purpose of validation and verification. This is particularly vital in the NISQ era, given the noise and technology limitation continuously to be the major challenges in the near feature. In this work, we propose a low-level benchmark suite called QASMBench based on the OpenQASM assembly-level intermediate representation (IR) [28]. It collects commonly used quantum algorithms and routines (e.g., the adder circuit in Figure 1) from a variety of distinct domains, including quantum chemistry, simulation, linear algebra, searching, optimization, arithmetic, machine learning, fault tolerance, cryptography, etc. The design of the benchmark suite trades-off between generality and usability, covering a wide spectrum regarding circuit depth (i.e., number of gates) and width (i.e., number of qubits). Additionally, to analyze and compare the benchmark applications, we propose four circuit evaluation metrics including gate density, retention lifespan, measurement density, and entanglement variance, to offer deeper insights about the execution efficiency, the susceptibility to system/readout error, and the potential impact from device-specific optimizations, when mapping to a NISQ device. Most of the QASMBench applications can be directly uploaded and executed in IBM-Q [54] or launched through Qiskit. The opensource tool *q-convert* enables the translation of QASMBench to other QC programming languages or representations.

This paper is organized as follows: In Section 1, we briefly describe NISQ and the landscape of OpenQASM. In Section 2, we introduce the QASMBench applications. In Section 3, we propose our

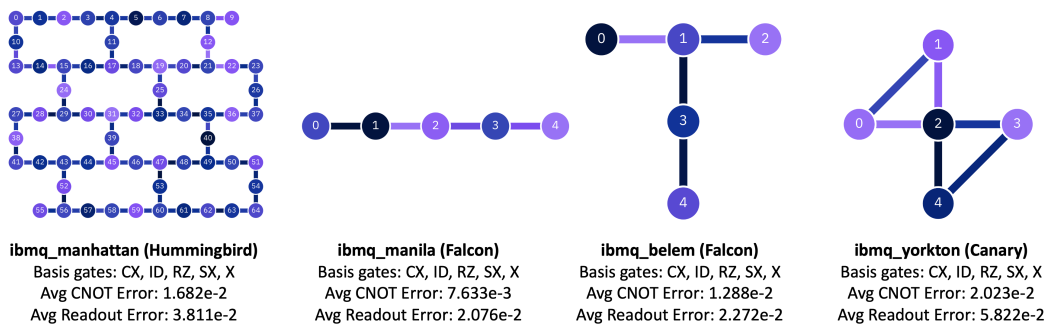
ibmq_manhattan (Hummingbird)
Basis gates: CX, ID, RZ, SX, X
Avg CNOT Error: 1.682e-2
Avg Readout Error: 3.811e-2
ibmq_manila (Falcon)
Basis gates: CX, ID, RZ, SX, X
Avg CNOT Error: 7.633e-3
Avg Readout Error: 2.076e-2
ibmq_belem (Falcon)
Basis gates: CX, ID, RZ, SX, X
Avg CNOT Error: 1.288e-2
Avg Readout Error: 2.272e-2
ibmq_yorkton (Canary)
Basis gates: CX, ID, RZ, SX, X
Avg CNOT Error: 2.023e-2
Avg Readout Error: 5.822e-2

Fig. 2. IBM-Q NISQ device topology. The color of nodes implies frequency (GHz) of the qubit or how fast a 1-qubit gate can be executed. The color of the connection implies the gate time in nanoseconds for 2-qubit gate such as CX.

evaluation metrics. We characterize the QASMBench applications using the metrics in Section 4. Finally, we summarize and draw the conclusion.

1 BACKGROUND: QUANTUM COMPUTING IN THE NISQ ERA

We describe the NISQ devices and the OpenQASM programming environment.

1.1 NISQ Devices

NISQ devices describe the near-term quantum platforms incorporating 50 to less than a thousand qubits [92]. The qubits are currently built using a variety of different technologies, including superconducting qubits [23, 97], trapped-ions qubits [21, 69], spin qubits [75, 91], photonic qubits [6, 87], etc. To correctly run a QC circuit, the physical qubits have to stay coherent long enough for allowing the accomplishment of all the gate operations, thus imposing rigid constraints on allowable circuit depth [26]. The **T1 coherence** time describes the time for a qubit to decay from the excited state $|1\rangle$ to the ground state $|0\rangle$. The **T2 coherence** time describes the time for a qubit to transit its state due to environment interaction [110]. Consequently, gate operations should be quickly and precisely performed on the qubits during the coherent window for maintaining the quantum states. The probability of involving error during the gate operation is known as **gate error**. The measurement operation, which reads-out classical output from a qubit by collapsing its final quantum state, is also critical for the correctness and efficiency of QC [44]. The **readout error** describes the probability of inaccurate measurement of a qubit state [117]. To precisely sample the output distribution with the existence of gate and read-out error, a quantum circuit is often executed many times (known as **shots** [54] or repetition [48]).

Each NISQ device has its hardware-defined basis gate set. All quantum circuits are translated into an object circuit that only contains the basis gates before they are executed by the NISQ machine. This process is called **quantum transpilation**. For IBM-Q devices, the basis gate set is typically `CX`, `ID`, `RZ`, `SX`, `X`, as shown in Figure 2. Additionally, due to the limitations of NISQ devices, only certain physical qubits are directly connected. In other words, physical qubits in NISQ devices are linked following certain **topology**, as illustrated in Figure 2. The topology limits the locations where two-qubit gates such as `CX` can be performed. If a `CX` is applied over two remote qubits, a series of `SWAP` gates are needed to relocate the two qubits to a connected tuple in the topology before the `CX` can be applied. For example, `CX(0,4)` in *ibmq_manila* in Figure 2 requires 4 extra `SWAP` gates. Additionally, more swaps may be needed to switch the qubit(s) back to their original

Table 1. OpenQASM gate definition (5 basic gates + 11 standard gates + 18 composition gates).

Gates	Meaning	Gates	Meaning	Gates	Meaning
U3	3 parameter 2 pulse 1-qubit	TDG	conjugate of sqrt(S)	CRZ	Controlled RZ rotation
U2	2 parameter 1 pulse 1-qubit	RX	X-axis rotation	CU1	Controlled phase rotation
U1	1 parameter 0 pulse 1-qubit	RY	Y-axis rotation	CU3	Controlled U3
CX	Controlled-NOT	RZ	Z-axis rotation	RXX	2-qubit XX rotation
ID	Idle gate or identity	CZ	Controlled phase	RZZ	2-qubit ZZ rotation
X	Pauli-X bit flip	CY	Controlled Y	RCCX	Relative-phase CXX
Y	Pauli-Y bit and phase flip	SWAP	Swap	RC3X	Relative-phase 3-controlled X
Z	Pauli-Z phase flip	CH	Controlled H	C3X	3-controlled X
H	Hadamard	CCX	Toffoli	C3XSQRTX	3-controlled sqrt(X)
S	sqrt(Z) phase	CSWAP	Fredkin	C4X	4-controlled X
SDG	conjugate of sqrt(Z)	CRX	Controlled RX rotation		
T	sqrt(S) phase	CRY	Controlled RY rotation		

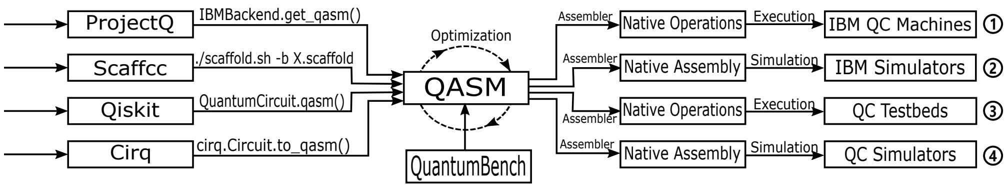

Fig. 3. The OpenQASM toolchain.

position(s), adding extra overhead and error. **Qubit mapping** describes the process of mapping of the logical qubits of a quantum circuit to the physical qubits of a NISQ device, which significantly impacts QC accuracy and execution efficiency.

1.2 OpenQASM Ecosystem

OpenQASM (*Open Quantum Assembly Language*) [28] is a low-level intermediate representation (IR) of quantum instructions, which is similar to traditional *Hardware Description Language* (HDL) like *Verilog* and *VHDL*. OpenQASM is a unified low-level assembly language for IBM-Q quantum machines. These NISQ devices, being accessible through IBM-Q cloud or IBM-Q network [54], have been explored by many existing works [26, 53, 67, 83–85, 110]. An OpenQASM code can be directly uploaded and verified in an IBM-Q machine or launched through Qiskit.

Table 1 lists the types of gate that are defined in OpenQASM specification (i.e. the "`qelib1.inc`" header file) [28]. Within these gates, the first five, i.e., `U3`, `U2`, `U1`, `CX`, and `ID`, are *basic gates* that are expected to be supported by the quantum backend. From `X` to `RZ` are *standard gates* defined atomically in OpenQASM. These standard gates will be converted into basic gates during machine-specific assembling & mapping phase before actual execution. The remaining gates from `CZ` to `C4X` are *composition gates* that are constructed by standard gates. These gates are defined in `qelib1.inc` for the convenience of usage.

OpenQASM is a low-level assembly language. It is executed "sequentially" without any loops, branches or jumps, making it very convenient for static analysis and simulating in a classical simulator. OpenQASM is widely supported. Several popular QC software frameworks such as Qiskit [55], Cirq [48], Scaffold [59], ProjectQ [105], etc. can dump the ready-to-execute circuit into an OpenQASM file, so it can be validated in the IBM-Q backends. Figure 3 shows the landscape of OpenQASM. As can be seen, OpenQASM stands at the conjunction between quantum software and hardware, being critical for both platform-agnostic or platform-specific optimization, mapping, scheduling, evaluation, profiling, and simulation. An OpenQASM based benchmark suite can be useful for all these activities. In the following, we briefly discuss each of the front-end frameworks.

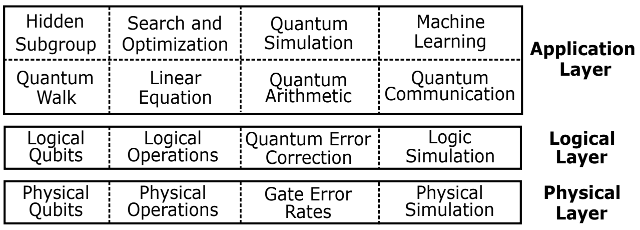

Fig. 4. Quantum computing stack for the categorization of QASMBench. It is built based on [81] by augmenting the categories of quantum arithmetic, quantum machine learning and quantum communication.

Qiskit: The *Quantum Information Software Kit* (Qiskit) [55] is a quantum software platform developed by IBM. Qiskit is mainly based on Python but is also available in JavaScript and Swift [115]. The package comprises several tools, such as *qiskit-aer* for simulation, *qiskit-ignis* for hardware verification and noise addressing, and *qiskit-aqua* for exemplary applications. OpenQASM can be easily dumped from a Qiskit program using "*QuantumCircuit.qasm()*".

Cirq: Cirq [48] is a QC software platform developed by Google. It is based on Python. Despite claimed to be the interface for connecting to their 72-qubit Bristlecone quantum computer, this is not yet available to the general users. Cirq incorporates a local simulator for generic gates, and a *Xmon-Simulator* for simulating the native gateset of Google's quantum computers. In addition, Cirq offers a function called "*to_qasm()*" in each circuit object so that a circuit can be dumped as an OpenQASM code that is runable on IBM-Q backends. Note, based on our experience, not all Cirq code can be translated to OpenQASM.

Scaffold: Scaffold [59] is a quantum programming language embedded in the C/C++ programming language based on the LLVM compiler toolchain [68]. The major goal of Scaffold is to assist in developing quantum algorithms and advanced optimization, leveraging the existing LLVM compiling flow. By supporting advanced language structures such as loops, Scaffold can generate very complex OpenQASM code, such as the VQE examples in QASMBench. A Scaffold program can be compiled by *Scaffcc* [59] to native OpenQASM code using the "*-b*" compiler option.

ProjectQ: ProjectQ is a quantum software platform developed by Steiger el al. from ETH Zürich [105]. Similar to Scaffold, ProjectQ does not have its own dedicated real quantum backend, but relies on classical simulation. However, it provides a way to generate OpenQASM code so that a ProjectQ program can be verified on an IBM testbed — through "`IBMBackend.get_qasm()`".

2 QASMBENCH

We introduce QASMBench in this section. Some of the benchmark routines are generated and reproduced from existing open-source QC software packages, including [28, 48, 55, 59, 77, 80, 98] through the approaches described in Figure 3 whereas others are locally developed.

Depending on the number of qubits used, QASMBench is partitioned into three categories: **Small-scale** with qubits ranging from 2 to 5, as most IBM-Q machines feature 5 qubits. The benchmarks in this category are listed in Table 2. **Medium-scale** with qubits ranging from 6 to 15, partially because the IBM-Q devices open to the public are having the maximum number of qubits equal to 15 (i.e., *ibmq_melbourne*) for now. The benchmarks in this category are listed in Table 3. **Large-scale** contains benchmarks with more than 15 qubits, listed in Table 4.

For each item in Table 2, 3 and 4, we list its name, brief description, and the algorithm category it belongs to in the quantum stack (see Figure 4). We show the number of qubits and gates utilized in the routines. The gate number here refers to *standard* OpenQASM gates (not basic gates or

Table 2. QASMBench Small-scale Benchmark.

Benchmark	Description	Algorithms	Qubits	Gates	CX	Ref
`adder`	Quantum ripple-carry adder	Quantum Arithmetic	4	23	10	[59]
`basis_change`	Transform the single-particle baseis of an linearly connected electronic structure	Quantum Simulation	3	53	10	[77]
`basis_trotter`	Implement Trotter steps for molecule LiH at equilibrium geometry	Quantum Simulation	4	1626	582	[77]
`deutsch`	Deutsch algorithm with 2 qubits for $f(x) = x$	Hidden Subgroup	2	5	1	[28]
`dnn`	Quantum Deep Neural Network	Quantum Machine Learning	2	268	84	[106]
`qec_dist3`	Error correction with distance 3 and 5 qubits	Error Correction	5	114	49	[80]
`grover`	Grover's algorithm	Search and Optimization	2	16	2	[4]
`inverseqft`	Performs an exact inversion of quantum Fourier tranform	Hidden Subgroup	4	8	0	[28]
`linearsolver`	Solver for a linear equation of one qubit	Linear Equation	3	19	4	[14]
`lpn`	Learning parity with noise	Machine Learning	5	11	2	[98]
`pea`	Phase estimation algorithm	Hidden Subgroup	5	98	42	[28]
`qaoa`	Quantum approximate optimization algorithm	Search and Optimization	3	15	6	[56]
`qec_sm`	Repetition code syndrome measurement	Error Correction	5	5	4	[28]
`qec_en`	Quantum repetition code encoder	Error Correction	5	25	10	[98]
`qft`	Quantum Fourier transform	Hidden Subgroupe	4	36	12	[28]
`quantumwalks`	Quantum walks on graphs with up to 4 nodes	Quantum Walk	2	11	3	[79]
`shor`	Shor's algorithm	Hidden Subgroup	5	64	30	[55]
`toffoli`	Toffoli gate	Logical Operation	3	18	6	[59]
`jellium`	Variational ansatz for a Jellium Hamiltonian with a linear-swap network	Quantum Simulation	4	54	16	[77]
`wstate`	W-state preparation and assessment	Logical Operation	3	30	9	[28]

Table 3. QASMBench Medium-scale Benchmarks.

Benchmark	Description	Domain	Qubits	Gates	CX	Ref
`dnn`	Quantum Deep Neural Network	Quantum Machine Learning	8	1200	384	[106]
`adder`	Quantum ripple-carry adder	Quantum Arithmetic	10	142	65	[28]
`bb84`	A quantum key distribution circuit	Quantum Communication	8	27	0	[48]
`bv`	Bernstein-Vazirani Algorithm	Hidden Subgroup	14	41	13	[28]
`ising`	Ising model simulation via QC	Quantum Simulation	10	480	90	[59]
`multipler`	Quantum multipler	Quantum Arithmetic	15	574	246	[48]
`multiply`	Performing 3×5 in a quantum circuit	Quantum Arithmetic	13	98	40	[4]
`qaoa`	Quantum approximate optimization algorithm	Search and Optimization	6	270	54	[48]
`qf21`	Using quantum phase estimation to factor the number 21	Hidden Subgroup	15	311	115	[4]
`qft`	Quantum Fourier transform	Hidden Subgroup	15	540	210	[28]
`qpe`	Quantum phase estimation algorithm	Hidden Subgroup	9	123	43	[4]
`sat`	Boolean satisfiability problem via QC	Searching and Optimization	11	679	252	[28]
`seca`	Shor's error correction algorithm for teleportation	Error Correction	11	216	84	[4]
`simons`	Simon's algorithm	Hidden Subgroup	6	44	14	[4]
`vqe_uccsd`	Variational quantum eigensolver with UCCSD	Linear Equation	8	10808	5488	[59]

Table 4. QASMBench Large-scale Benchmarks.

Benchmark	Description	Domain	Qubits	Gates	CX	Ref
`bigadder`	Quantum ripple-carry adder	Quantum Arithmetic	18	284	130	[28]
`bv`	Bernstein-Vazirani algorithm	Hidden Subgroup	19	56	18	[28]
`cc`	Counterfeit coin finding problem via QC	Hidden Subgroup	18	34	17	[28]
`ghz_state`	GHZ State preparation and assessment	Logical Operation	23	23	22	[49]
`ising`	Ising model simulation via QC	Quantum Simulation	26	280	50	[59]
`multipler`	Quantum multipler	Quantum Arithmetic	25	3723	750	[48]
`qft`	Quantum Fourier tranform	Hidden Subgroup	20	970	380	[28]
`square_root`	Computing the square root of an number via amplitude amplification	Quantum Arithmetic	480	16064	6274	[59]
`swap_test`	SWAP Test algorithm implementation	Logical Operation	25	446	96	[63]
`wstate`	W-state preparation and assessment	Logical Operation	27	446	96	[28]

composition gates) as discussed in Section 1.2. We also list the number of `CX` gates in the tables, which indicates a major source of delay and error. Each algorithm is briefly introduced under its respective category. However, certain routines appear in multiple categories, we only describe them in the first appearance. Last not the least, although QASMbench is developed based on OpenQASM, they can be converted to other representations such as Q#, PyQuil, Cirq, etc. through the Javascript based *q-convert* tool, which is also available online: http://quantum-circuit.com/qconvert.

2.1 Small-scale Benchmarks

Adder: The quantum adder uses each qubit as a classical bit, and builds a traditional adder through quantum gates. Here, we build the full-adder circuit using a basic component — a 4-qubit ripple-carry adder [30, 112] with a carry-in and a carry-out qubit. Given the granularity of 4-qubits, a m-qubit adder would require $2m + m/4 + 1$ qubits in total where $m/4 + 1$ qubits are used for the carrying bits between the basic 4-qubit adder components. We included a 4-qubit adder (see Figure 1) in small-scale, a 10-qubit adder in medium-scale, and a 18-qubit adder in large-scale. This benchmark is motivated by its scalability, easy-to-verify and usefulness in building quantum numeric units in the post-NISQ era.

Basis_change: The OpenQASM routine of this benchmark is generated from Google's Cirq based OpenFermion library [77] for manipulating fermionic systems towards chemistry simulation on quantum computers. It shows how to use a quantum circuit to transform the single-particle basis of an linearly connected electronic structure so as to realize exact evolution under a random Hermitian one-body fermionic operator. This is included as a quantum chemistry benchmark.

Basis_trotter: This routine is also generated from the Cirq based OpenFermion library [77] using the method described in Figure 3. It is designed to implement the Trotter [12, 72] steps for an actual molecule LiH at equilibrium geometry, so as to simulate basis molecular Hamiltonians taking the following form $H = \sum_{pq} T_{pq} a_p^\dagger a_q + \sum_{pqrs} V_{pqrs} a_p^\dagger a_q a_r^\dagger a_s$. The circuit uses 4 qubits but is quite deep, challenging the T1 and T2 decay of a NISQ device. It is included as another chemistry benchmark.

Deutsch: The Deutsch-Jozsa algorithm [32] is a generalization of Deutsch's algorithm. The problem is defined as – given an oracle function $f\{0,1\}^n \longrightarrow \{0,1\}$ which takes an n binary digit as input, and outputs either 0 or 1. The objective is to find whether the outputs are the same for all of the inputs (say constant), or are one value (i.e., 0 or 1) for half of the inputs and the other (i.e., 1 or 0) for the remaining half (say balanced). Here, if the 2^n possible inputs are encoded into an n-qubit register, and if the oracle function can be expressed as gate operations in a black-box, the quantum computer can get the answer with only one-time evaluation to the oracle function whereas a deterministic classical computer would require $2^{n-1} + 1$ times evaluations. This algorithm is among the first examples to demonstrate that a quantum algorithm can achieve exponentially speedup over any possible deterministic classical algorithms. It is included as a baseline circuit.

DNN: Quantum deep neural networks [106] is a quantum adaption of the classical neural network where variational quantum circuits with a layered-approach are parameterized by a bank of θ values. QNN consists of gates such as `RY(`θ`)`, `CX` and `CRY(`θ`)` for encoding classical data and weight information as well as introduces entanglement. Tasks such as classification are commonly used in evaluating these models where an attainable objective function is described alongside quantum gate differentiation, where the circuit is optimized.The domain of Quantum Machine Learning (QML) and QNN have drawn substantial attention recently [16] as the hope for a quantum advantage in machine learning, thereby motivating the inclusion of the quantum DNN benchmark.

QEC: Quantum error correction (QEC) is to introduce redundancy into the original circuit by using additional physical qubits so that the state of the original circuit is effectively protected against noise. The surface code is often considered to be among the most promising QEC approaches as it can cope with error rates around 10^{-2} [43]. The *qec_dist3* benchmark performing a 3-distance 5-qubit surface code which is developed based on [80]. It is the smallest code that can correct arbitrary single-qubit error. QASMBench includes other QEC benchmarks, such as **qec_en** and **qec_sm** for encoder and syndrome measurement of repetition code, and **seca** for Shor's QEC for teleportation. QEC is perhaps the most important algorithm for NISQ and the success of QC in the long run, hence we include four QEC benchmarks in QASMBench.

Grover: Grover's algorithm [50] is a quantum algorithm for searching in a database, offering quadratic speedup over classical searching methods. It takes a black-box oracle realizing the function: $f(x) = 1$ if $x = y$, $f(x) = 0$ if $x \neq y$, and find y within a randomly ordered sequence of N items using $O(\sqrt{N})$ operations and $O(N \log N)$ gates with probability $p \geq 2/3$. Grover's algorithm is appealing in that it determines with high probability the unique input given the output is pre-known. Grover's algorithm is one of the fundamental quantum algorithms considering the importance of searching in the big-data era.

QFT & InverseQFT: The Quantum Fourier Transform (QFT) [25] (and its inverse) applies (inverse) Fourier transformation to the wave function amplitudes. It is a linear transform over the states of qubits, which is the quantum analogue of the discrete (inverse) Fourier transform. QFT is a basic component of many well-known quantum algorithms, including Shor's algorithm, quantum phase estimation, hidden subgroup problem, etc. Besides, it has some interesting features such as zero entanglement variance, as well be discussed later.

LinearSolver: This benchmark realizes the HHL solver [52] for solving a linear equation of 1-qubit. The HHL algorithm estimates the outcome of a scalar measurement on the solution vector to a sparse linear equation system. We originally attempted to dump the multi-qubits HHL routine from Cirq [48], but encountered error in generating the OpenQASM code due to the mismatch between Google's native quantum IR and OpenQASM (e.g., missing the `CCCRy` representation). HHL is a critical quantum algorithm given the importance of linear algebra in scientific computation and machine learning.

LPN: The LPN (*learning parity with noise*) benchmark is a machine learning problem of learning a hidden parity function defined by the unknown binary string in the presence of noise. This application is very promising for NISQ devices because (i) it is computationally intractable for classical approaches [89]; (ii) it actually leverages the noise presented in the NISQ devices [29], which is usually seen as a hurdle in other applications.

Pea: Quantum *phase-estimation* (QPE) [86] is an algorithm to compute the eigenvalue $e^{2\pi i\theta}$ of a unitary operator operating on m qubits with the eigenvector $|\psi\rangle$ such that $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$, $0 \leq \theta < 1$. QPE is another fundamental quantum algorithm being the key component of Shor's method. This *Pea* benchmark circuit describes a 2-qubit system including a read-out ancilla bit, and a physical system bit [34], showcasing the property of QPE.

QAOA: The quantum approximate optimization algorithm (QAOA) [38, 118] is another variational quantum algorithm that is particularly interesting to NISQ devices, given its ability to tolerant certain degree of noise through the unique quantum-classic hybrid algorithm design. QAOA is designed to solve combinatorial optimization problems [117], such as in graph analytics. In QAOA, a quantum subroutine is embedded in a classical search loop. The initial quantum state is prepared according to a set of variational parameters. These parameters are adjusted based on the measurement in the previous iteration. The QAOA benchmark in QASMBench represents the quantum workload for a sample iteration.

QuantumWalks: Quantum walks [2, 113] is the quantum analogue of the classical random-walk algorithm, which has been adopted for many problems including graph isomorphism [45] and ranking nodes in a network [88]. Both discrete and continuous-time quantum-walk algorithms have been formulated by existing works [2, 39]. The quantum-walk benchmark here represents the workload for a single step of the walks [79].

Shor: Shor's algorithm [101] is a famous QC algorithm for integer factorization in polynomial-time. It demonstrates theoretical exponential speedups over the classical algorithms, posing substantial threat and concern over the widely-used public-key cryptography systems like RSA. Given its importance in quantum cryptography, we include a 5-qubit sample here in QASMBench. We also include another example — **qf21** for showcasing the process of factoring the integer of 21.

Toffoli: Similar to the Fredkin gate, the Toffoli gate [111] or `CCX`) gate is another universal gate. It refers to a circuit with three inputs and three outputs that inverts the third qubit if the first two qubits are both 1, otherwise all bits stay unchanged. We include the 3-qubit Toffoli gate as a baseline benchmark in QASMBench.

VQE: The variational quantum algorithm is to optimize a parameterized quantum circuit ansatz applied to some initial state for minimizing a cost function defined according to the output state [60, 78]. When applied in quantum simulation (i.e., simulating physical phenomena using QC rather than simulating QC in a classical computer), the goal of the algorithm is often to prepare ground states. The cost function is often the expectation value of a Hamiltonian. If the initial state is $|\psi\rangle$, the Hamiltonian is H, the ansatz is $U(\vec{\theta})$ where $\vec{\theta}$ is the variational parameter, then the cost function is $E(\vec{\theta}) = \langle\psi| U^{\dagger}(\vec{\theta})HU(\vec{\theta}) |\psi\rangle$. The *Jellium* benchmark creates a variational ansatz for a Jellium Hamiltonian via a linear-swap network. We also include two *Unitary Coupled Cluster Single-Double* VQE ansatz [114] for estimating the ground energy of molecules due to their ultra deep circuits.

W_state: *w_state* is one of the basic states exhibiting properties not observed in a classical system. The *w_state* [37] is a special type of entangled quantum state generalized by:

$$|W\rangle = \frac{1}{\sqrt{n}}(|10\dots0\rangle + |01\dots0\rangle + \dots + |00\dots1\rangle) \text{ for } n > 2$$

It describes the condition that for an n-qubit system, no matter which single qubit is measured or lost, the remaining (n-1) qubits still remain entangled. This property makes it useful for improving the robustness of ensemble-based quantum memories [42]. This routine implements the 3-qubit condition where the three qubits follow $|W\rangle = \frac{1}{\sqrt{3}}(|001\rangle + |010\rangle + |100\rangle)$.

2.2 Medium-scale Benchmarks

BB84: *BB84* [10] is a quantum key distribution (QKD) protocol which is the first quantum cryptographic protocol [15] based on the non-cloning quantum laws for showcasing provably secure key generation. It relies on the fact that it is not possible to obtain information distinguishing two non-orthogonal states without disturbing the signal. Since QKD is a critical component for quantum communication and quantum networks, we include BB84 as a sample benchmark in QASMBench.

BV: The Bernstein-Vazirani algorithm [11] is an extension to the Deutsch-Josza algorithm [32]. The problem is that, given a black-box function f, which takes in n bits as input, and outputs the bitwise product of a string s against the input. This is defined as $x, f(x) = s \times xmod(2)$ where the string s is hidden. Classically, to determine the hidden string, we need n evaluations (100..0, 0100..0, ..., 0000..1). This algorithm can decide s through a single query to the oracle on a quantum computer. Given it is another application that can showcase exponential quantum speedup, we include it in QASMBench.

Ising: The Quantum Ising Model [17, 66] consists of an array of quantum spins arranged in a certain lattice. The spins, which are expressed in quantum operators, can only interact with their neighbors. The phase transition is due to quantum fluctuation introduced by the transverse field. Given the quantum annealing machines (e.g., D-Wave) basically resolving optimization problems that can be expressed in an Ising model, we include it as a benchmark routine in QASMBench.

Multiplier: Similar to the adder, the quantum multiplier also uses a qubit as a classical bit to construct a multiplier unit. Our implementation is similar to the one from Cirq [48], which requires $5n$ qubits for an n-qubit multiplier and internally calls the n-qubit adder by n times for aggregating the partial results. The multiplier comprises the three phases: *preparing operand*, *multiply* and

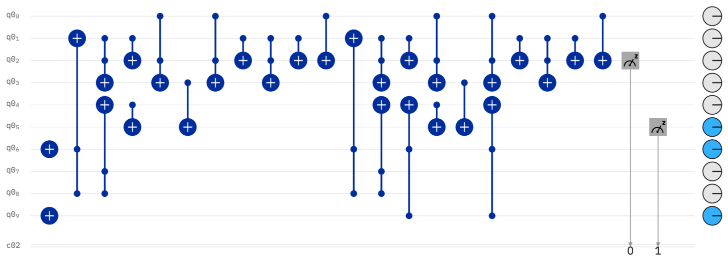

Fig. 5. A 2-qubit multiplier. The execution phases include: (1) *Preparation* (2) *Execution* (3) *Measurement.*

measure. Figure 5 shows an example of a 2-qubit multiplier using 10 qubits. Similarly to the adder, this is motivated by its simple scalability and easy verification.

SAT: The *Boolean Satisfiability* problem (SAT) determines whether there is an assignment of variables that satisfies a given Boolean function [7, 107] which is one of the first proven NP-complete problems. Since many optimization problem (e.g., in power-grid domain) can be attributed as a mixed-integer-linear-programming (MILP) problem, which can be formulated as a binary optimization problem [18] similar to SAT, we include SAT here as a benchmark in QASMBench.

Simon: Simon's algorithm [103] was among the first quantum algorithms to show exponential speedups over the best classical deterministic or probabilistic algorithm. It finds an unknown nonzero $s \in \{0,1\}^n$ that satisfies $f(x) = f(x \oplus s)$. We include it here as another benchmark showing quantum advantage. Besides, Simon's algorithm is more sensitive to noise in NISQ devices.

2.3 Large-scale Benchmarks

CC: The *counterfeit coin* problem is a mathematical puzzle that attempts to find all false coins from a given bunch of coins using a balance scale [57]. Let's say you have a set of N coins, and up to k of them are counterfeit and hence lighter. Using a scale, which returns the information of "balanced" or "tilted", we can deduce if any, or how many of the coins are counterfeit. The classical algorithm requires a time complexity of $O(k\log(k/N))$. The quantum algorithm offers an exponential speed-up, with a time complexity of $O(k^{\frac{1}{4}})$ [58].

GHZ-state: A *Greenberger–Horne–Zeilinger* (GHZ) state [49] is a particular type of entangled quantum state that involves at least three qubits: $|GHZ\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$. Since it describes the superposition of all qubits being 0 with all of them being 1, it represents the maximally entangled quantum state. Consequently, characterizing GHZ state can provide very useful information about the fidelity of multi-qubit interactions, which is critical for constructing large-scale quantum computers. For an n-qubit system, GHZ state is generalized as:

$$|GHZ\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes n} + |1\rangle^{\otimes n}) \text{ for } n > 2$$

To construct a GHZ-state for an n-qubit system, from an all-zero states $|\Psi\rangle = |0\rangle^{\otimes n}$, after an `H` gate is applied on the first qubit, $|\Psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle^{\otimes n} + |1\rangle \otimes |0\rangle^{\otimes n-1})$. We then apply a series of `CX` (or `CNOT`) gates from qubit 0 to qubit (n-1). Running on a full-fidelity quantum machine with 5 logic qubits, the measurement result should be 50% $0^{\times n}$ and 50% $1^{\times n}$. This application represents another basic quantum state which exploits across all the available qubits. It can be used to prove CNOT gate stability of a NISQ device.

Squarer_root: This routine relies on amplitude amplification [51] to find the square root of an n-bit number using Grover's search technique. This is a complex application in QASMBench.

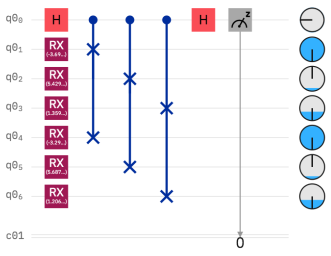

Fig. 6. 3-qubit swap_test.

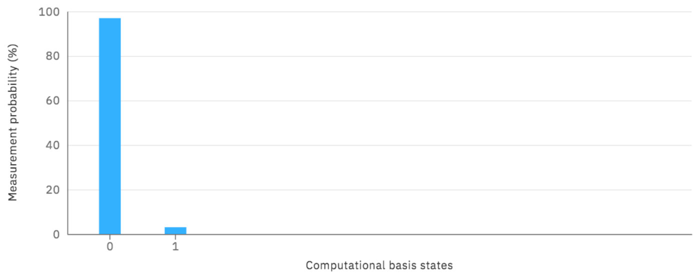

Fig. 7. Measurement result for a 3-qubit swap_test circuit using 7 qubits.

Swap-test: The swap-test is a very convenient and economic way to measure the difference between two quantum states, or how two quantum ($|\psi\rangle$ and $|\phi\rangle$) states are divergent from each other. The swap-test can be very useful in quantum machine learning for estimating the prediction accuracy (e.g., distance of prediction from the tag). It uses an ancilla qubit for example qubit 0. The initial state of the system is $|0, \psi, \phi\rangle$. The ancilla qubit is firstly put in superposition via an H gate, as shown in Figure 6, which converts the system state into $\frac{1}{\sqrt{2}}(|0, \psi, \phi\rangle + |1, \psi, \phi\rangle)$. By applying the series of controlled-swap gate (or CSWAP, which swaps the states of two qubits only if the control qubit is 1, see Table 1), the system state becomes $\frac{1}{\sqrt{2}}(|0, \psi, \phi\rangle + |1, \phi, \psi\rangle)$. Then, by applying another H gate on qubit-0, the system transits to $\frac{1}{2}(|0\rangle(|\psi, \phi\rangle + |\phi, \psi\rangle) + |1\rangle(|\psi, \phi\rangle - |\phi, \psi\rangle))$. When measuring qubit-0, the probability of obtaining 0 is $P(0) = \frac{1}{2} + \frac{1}{2}|\langle\psi|\phi\rangle|^2$. Therefore, if $|\psi\rangle$ and $|\phi\rangle$ are orthogonal, $|\langle\psi|\phi\rangle|^2 = 0$, the probability of obtaining 0 is 50%; if $|\psi\rangle$ and $|\phi\rangle$ are equal, $|\langle\psi|\phi\rangle|^2 = 1$, the probability of obtaining 0 is 100%. The more the two states are similar, the higher the probability of obtaining 0 approaches 1. Note that the swap-test needs sufficient number of repetitions to achieve an accurate sampling of the probability of the ancilla qubit. The measurement probability in Figure 7 shows that the two 3-qubit states in Figure 6 are similar to each other with $P(0) \approx 1$. The SWAP test is a crucial algorithm in many Quantum processes, and is easily scaled out to any application size with substantial entanglement, hence is an ideal NISQ evaluation benchmark.

3 QASMBENCH CHARACTERIZATION METRICS

We propose a set of quantum circuit based evaluation metrics representing various features of a quantum application. These metrics are designed such that through them certain estimation can be performed on executing a particular circuit over a particular NISQ device, practicing the software-hardware co-design paradigm. The metrics serve as useful indicators on how a quantum circuit can stress a NISQ hardware device. The scripts for profiling these metrics will be provided along with the QASMBench code.

3.1 Circuit Width

Circuit width is defined as the number of qubits that enter the superposition state at least once within an application's lifespan. Qubits that are measured in the interim of a circuit and re-enter superposition are only counted as one qubit towards the circuit width. Circuit width dictates the spatial capacity required for a quantum device in order to run the quantum circuit. Circuit Width is mathematically defined in Equation 1:

$$\text{Circuit Width} = n_{q_{\text{active}}} \tag{1}$$

3.2 Circuit Depth

Circuit depth is defined as the minimum time-evolution steps required to complete a quantum application. Time evolution is the process of completing all gates defined at time $t = t_j$, and once these are completed, the circuit moves onto time $t = t_{j+1}$, where the following gates are to be processed. To keep generality and avoid the impact from low-level optimization, circuit depth here is calculated solely using standard OpenQASM gates (see Section 1.2). That is to say, the OpenQASM code is decomposed to standard gates before counting the accurate time evolution steps required. As a concrete example, an `X` gate is a standard gate and hence requires 1 time-step, whereas a `CRZ` gate requires 4 standard gates to complete, thereby requiring 4 timesteps.

Circuit depth can be computed by decomposing OpenQASM code into a $n_q \times t$ matrix Q, where Q_{q_i,t_j} is the time-evolution steps to complete the gate on qubit i at time j. The sum of the maximum time in each column is then equal to the minimum time required for a quantum application This is described in Equation 2:

$$\text{Circuit Depth} = \sum_{j=1}^{t} \max_{0 \le i < \text{width}} (Q_{q_i,t_j}) \tag{2}$$

3.3 Gate Density

Gate density, or operation density, describes the occupancy of gate slots along the time-evolution steps of a quantum circuit. As certain qubits might need to wait for other qubits in the time evolution (i.e, gate dependency), they remain idle by executing the `ID` gate (see Table 1). Consequently, if a gate slot is empty due to dependency, it implies a lower occupancy for the quantum hardware. This is similar to a classical processor, where data dependency introduces pipeline bubbles and reduced occupancy. We propose Gate Density to measure the likely occupancy of a circuit when mapping to a quantum hardware, as formulated in Equation 3. Note, the gate density is also calculated after decomposing down to the OpenQASM standard gates.

$$\text{Gate Density} = \frac{G_{\text{1-qubit}} + 2 \times G_{\text{2-qubit}}}{\text{Circuit Depth} \times \text{Circuit Width}} \tag{3}$$

where $G_{1-qubit}$ refers to the number of 1-qubit standard gates and $G_{2-qubit}$ refers to the number of 2-qubit standard gates (see Table 1).

3.4 Retention Lifespan

Retention Lifespan describes the maximum lifespan of a qubit within a system, and is motivated by the T1 and T2 coherence time of a quantum device (see Section 1.1). A longer lifespan of a quantum system implies more decay to the ground state (T1) and state-transition due to environment noise (T2), thus is more susceptible to information loss. Therefore, we propose taking the qubit with the longest lifespan to determine the system's retention lifespan. Using this metric, one can estimate if a particular circuit can be executed in a NISQ device with high fidelity, given its T1/T2 coherence time. Note, all IBM-Q machines offer T1/T2 coherence time as status indicators for the hardware. As circuit depth can grow substantially, we introduce the log operator to shrink the scale. If D_i is used to describe the lifespan or depth of the qubit i, Retention Lifespan, which measures the circuit size and sensitivity to quantum system error, is defined in Equation 4:

$$\text{Retention Lifespan} = \max_{0 \le i < width} (\log(D_i)) \tag{4}$$

3.5 Measurement Density

Measurement density assesses the importance of measurements in a circuit. A higher measurement count implies the fact that each measurement might be of relatively less importance (e.g., periodic measurement in QEC, or measurement over ancilla qubits), whereas for application with less measurements, the measurement may be of utmost importance. The importance also increases when a measurement accounts for a wider and/or deeper circuit. A good example is the SWAP test (see Figure 7), where the circuit can be very large but only one measurement is taken to report the similarity. Consequently, this measurement is extremely important to the application. Measurement Density is defined in Equation 5, where $N_{\text{measurement}}$ is the number of measurements in an application. Since the circuit depth/width can be large and the importance of measurement decays when circuit depth/width keeps on increasing, we add a log to shrink the scale.

$$\text{Measurement Density} = \frac{\log\left(\text{Circuit Depth} \times \text{Circuit Width}\right)}{N_{\text{measurement}}} \tag{5}$$

3.6 Entanglement Variance

Entanglement Variance measures the balance of entanglement across the qubits of a circuit. Circuits with a higher Entanglement Variance indicate that certain qubits are more connected than other qubits (i.e., using more 2-qubit gates such as CX than others). This metric implies that when the circuit is mapped to a NISQ device: (i) less SWAP gates are needed if those hotspot qubits are mapped to the central vertices in the NISQ device topology, such as Qubit-1 in *ibmq_belem* and Qubit-2 in *ibmq_yorkton* in Figure 2). A higher entanglement variance implies a higher potential benefit from a good logic-physical qubit-mapping through quantum transpilation. If the entanglement variance is zero, little benefit should be expected from a better transpilation strategy; (ii) Given 2-qubit gate is one of the major sources introducing error, a higher Entanglement Variance implies uneven error introduction among qubits. We define Entanglement Variance in Equation 6, where G_{q_i}(2-qubits) is the number of 2-qubit gates operating on qubit-i, and $\overline{G_q}$ is the average number of 2-qubit gates operating over each qubit. The inclusion of plus 1 is to eliminate log errors with 0 variance.

$$\text{Entanglement Variance} = \frac{\log\left(\sum_{i=0}^{width}(G_{q_i}(\text{2-qubits}) - \overline{G_q}(\text{2-qubits}))^2 + 1\right)}{\text{Circuit Width}} \tag{6}$$

4 EVALUATION

4.1 Circuit Metrics

In this subsection, we evaluate QASMBench applications described in Section 2 using the circuit metrics proposed in Section 3.

4.1.1 Circuit Width. Figure 8 illustrates the width of the circuits in QASMBench. Since we partition QASMBench with respect to the number of qubits (1-5 for small-scale, 6-15 for medium-scale, and >15 for large-scale), this can be directly watched in Figure 8.

4.1.2 Circuit Depth. Figure 9 shows the depth of the benchmark circuits from QASMBench. Apparently, some circuits (e.g., *DNN*, *VQE*, *QFT*) are significantly deeper than other circuits. This figures shows that QASMBench covers a wide range of circuit depth from 11 in the *Linearsolver*_n-3 to 7252 in the *VQE*_UCCSD_n-8, offering a comprehensive circuit database for evaluating NISQ capacity and stability.

4.1.3 Gate Density. The plot of Gate Density for QASMBench is visualized in Figure 10. The general observation is that, with larger scale circuits, the gate density tends to be more sparse, implying reduced execution occupancy or efficiency. On the contrary, smaller circuits tend to be denser and

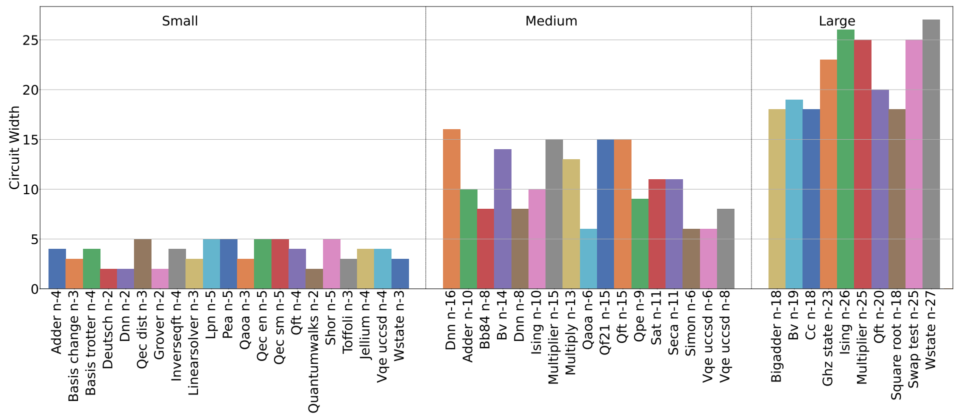

Fig. 8. Circuit Width of QASMBench circuits with respect to the small, medium and large categories.

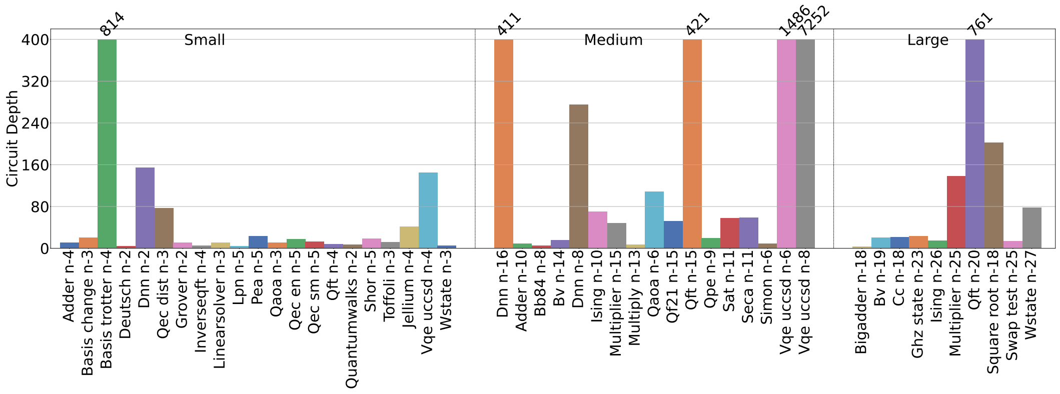

Fig. 9. Circuit Depth of QASMBench circuits.

more carefully designed, so that qubits are less likely to wait for other qubits before evolving onto the next time step. Nevertheless, QASMBench includes densely packed large and medium circuits with *Ising*_n-26 having a Gate Density factor of 0.8462 for large scale circuits, and *BB84*_n-8 with a Gate Density of 0.675 in the medium scale category. Note, a low gate density may also imply the amount of internal operation dependency and the potential gain with a better circuit reform.

4.1.4 Retention Lifespan. Retention Lifespan, visualized in Figure 11, illustrates the highly varying maximum qubit lifespan within the benchmark circuits of QASMBench. Short circuits such as *Deutsch*_n-2 are required to preserve a quantum state for a relatively short period of time with a retention lifespan of 1.3863, compared to the *Square-Root*_n-18 circuit with a retention life span of 5.6168. Note, retention lifespan is related to circuit depth, but taking dependency into consideration and is specific to a certain qubit rather than the overall system.

4.1.5 Measurement Density. The measurement density for the QASMBench circuits is illustrated in Figure 12, which measures the importance of measurement operations in a circuit. As can be seen, *SWAP-test*, *multiplier*, *DNN* and *QFT* show significantly higher measurement density than the other circuits. We have already discussed Swap-Test in Section 3.5. *QFT* has a very deep circuit

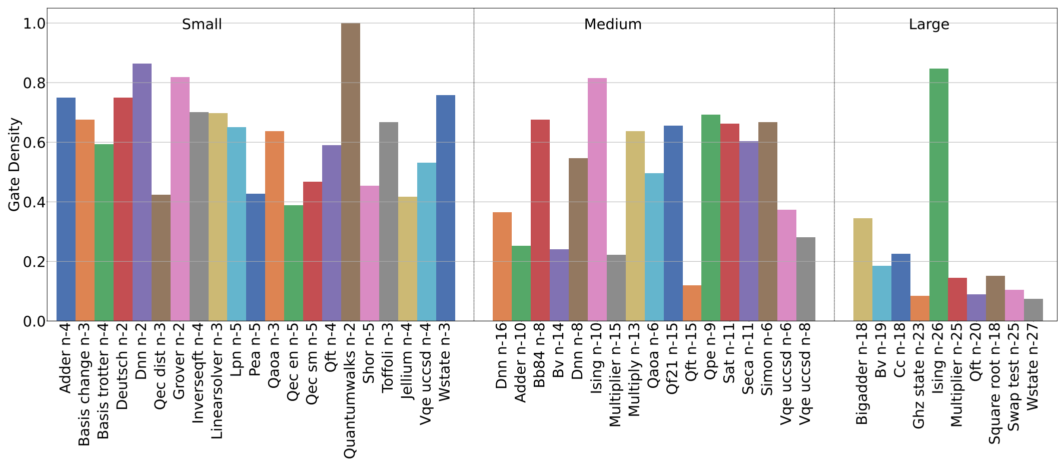

Fig. 10. Gate Density of QASMBench circuits.

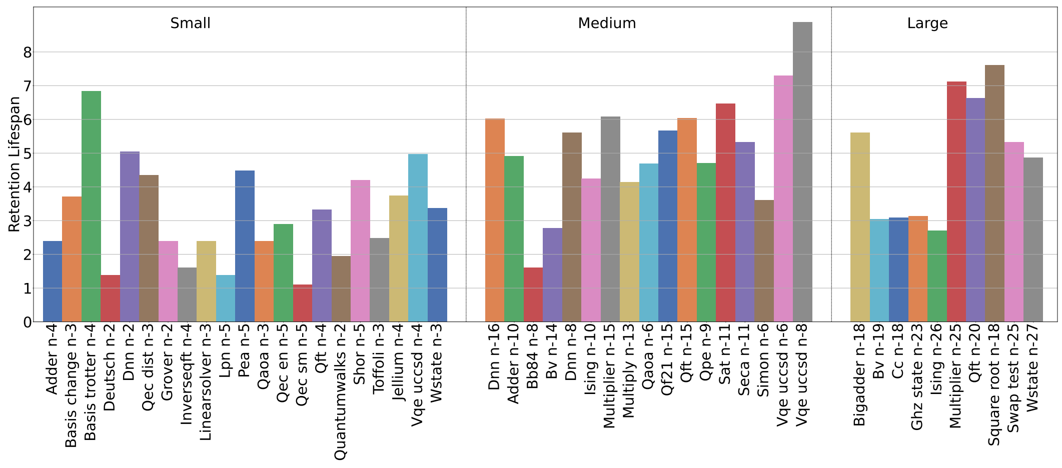

Fig. 11. Retention Lifespan of QASMBench Circuits

which explains why the only measurement at the end is of more importance. For *multiplier*, only qubits storing the product results are measured at the end. DNN essentially embeds the SWAP-test as a component for the estimation of the cost function.

4.1.6 Entanglement Variance. Figure 13 shows the entanglement variance for the QASMBench circuits. The figure demonstrates QASMBench's diverse entanglement arrangements, with densely entangled circuits such as Dnn n-2 and Grover n-2, where all entanglement operations entangle the only two qubits available. Therefore, the entanglement is evenly distributed across the circuit. However, compared to Basis trotter n-4, the primary bulk of entanglement is between qubits 1 and 2, whereas qubits 0 and 4 interact substantially less with other qubits. Consequently, when mapping *Basis-trotter* to a NISQ device like *ibmq_belem*, it involves much less SWAP gates if qubit-1 and 2 are mapped to Q1 and Q3 in Figure 2 than the reverse.

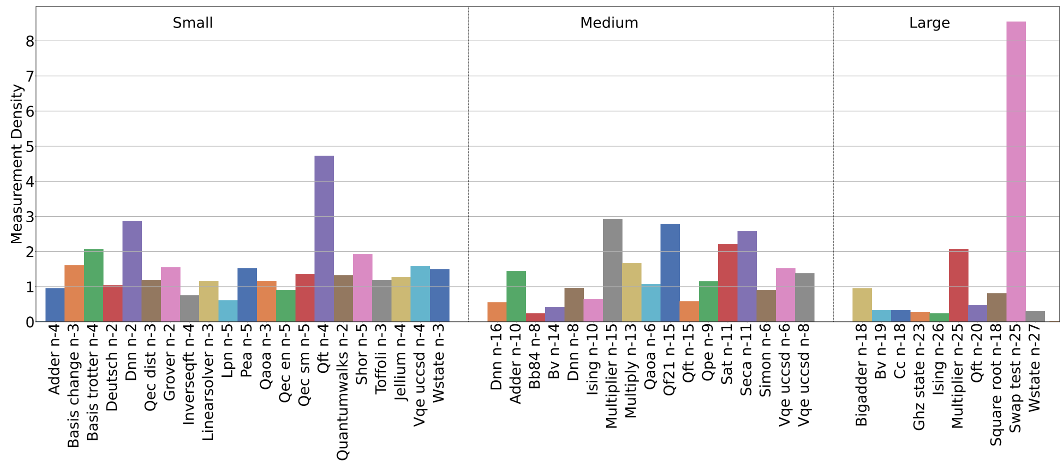

Fig. 12. Measurement Density of QASMBench Circuits

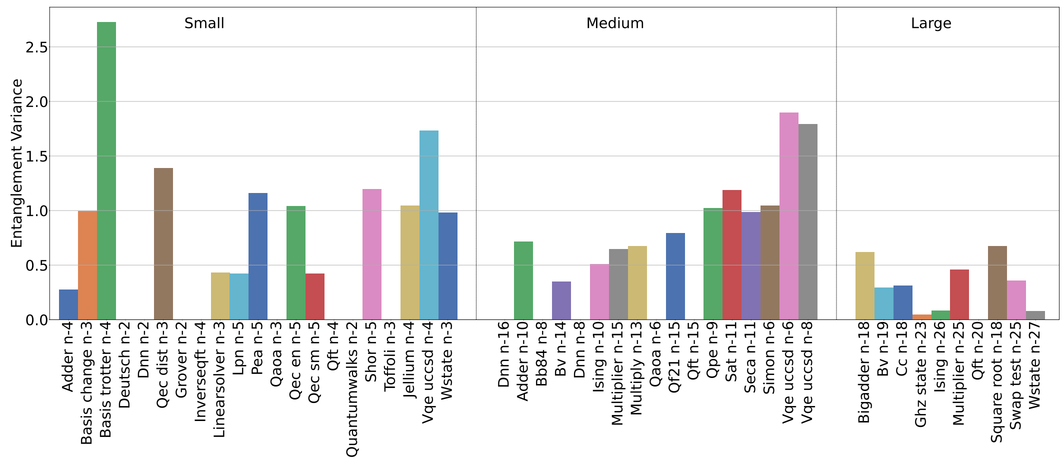

Fig. 13. Entanglement Variance of QASMBench Circuits

4.2 IBM-Q Evaluation

We have evaluated kernels in the small category in the *ibmq_burlington* NISQ machine, which features 5 qubits. We evaluated kernels in the medium category in *ibmq_melbourne* which features 15 qubits. We evaluated most of the kernels in the large category in *ibmq_paris* featuring 27 qubits. This is only for validation purposes and due to space consideration, we leave the result figures in the QASMBench GitHub repository.

5 RELATED WORK

To the best of our knowledge, QASMBench is the first benchmark suite aiming at evaluating NISQ devices using quantum applications from a broad range of domains. Existing works have already proposed to adopt randomly generated benchmarks for evaluating the error rates of different gate operations regarding a particular NISQ device [73, 74, 93]. Recently, Patel et al. [90] evaluated several IBM-Q machines using seven benchmarks and drew interesting observations regarding the error and execution time. Since the work mainly focused on extracting and validating new insights, only limited applications were included and no new metrics had been proposed.

Regarding QC metrics, Cross et al. defined the quantum volume protocol which quantified the largest random circuit of equal width and depth that a NISQ device can successfully implement [27], taking gate and measurement error into consideration. Quantum Linpack, proposed by Dong and Lin [35] recently, attempted to develop a random circuit block-encoded matrix (RACBEM) for measuring the QC capability in resolving a random system of linear equations. Benedetti et al. [8] proposed the qBAS score metric for benchmarking the shallow quantum-classical hybrid systems using synthetic data set. These metrics, despite being comprehensive, are not quite intuitive for understanding. Besides, these metrics all involve certain randomness due to synthetically generated inputs or circuits, which do not necessarily share the properties of real quantum workloads. Given the randomness, the validation requires offline simulation in a classic computer, ultimately limiting the scalability as the simulation cost in a classical computer scales exponentially with increased number of qubits.

Regarding the general evaluation of NISQ devices, in addition to [90], LaRose [67] compared the four widely used QC programming environments: Forest/pyQuil, Qiskit, Project and QDK/Q# in terms of library support, hardware, compiler and simulator performance. McCaskey et al. [76] proposed XACC, a programming model that defines a low-level intermediate representation and compiler front-end to provide a unified abstraction for evaluating different quantum hardware and programming environments. Finally, Abhijith et al. [1] developed a great hands-on tutorial in explaining the principles of QC using 20 different quantum algorithms, using the 5-qubit IBM-Q machine as the validation platform. However, this work is mainly for education purposes using the small-scale circuits (5 qubits) rather than serving as a QC benchmark suite.

6 CONCLUSION

In this paper, we propose a light-weighted, low-level and easy-to-use benchmark suite called QASMBench based on the OpenQASM quantum assembly language. It collects commonly seen quantum algorithms and routines from a variety of domains with distinct properties, serving the need for convenient quantum computing characterization and evaluation purposes for the computer system and architecture communities. Additionally, we propose four novel circuit metrics including gate density, retention lifespan, measurement density, and entanglement variance for assessing the execution efficiency, susceptibility to NISQ error, and potential gain from low-level optimizations.

ACKNOWLEDGMENTS

This material is based upon work supported by the U.S. Department of Energy, Office of Science, National Quantum Information Science Research Centers, Co-design Center for Quantum Advantage (C2QA) under contract number DE-SC0012704. The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830.

REFERENCES

[1] J Abhijith, Adetokunbo Adedoyin, John Ambrosiano, Petr Anisimov, Andreas Bärtschi, William Casper, Gopinath Chennupati, Carleton Coffrin, Hristo Djidjev, David Gunter, et al. 2018. Quantum algorithm implementations for beginners. *arXiv e-prints* (2018), arXiv–1804.

[2] Dorit Aharonov, Andris Ambainis, Julia Kempe, and Umesh Vazirani. 2001. Quantum walks on graphs. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*. 50–59.

[3] Thorsten Altenkirch and Jonathan Grattage. 2005. A functional quantum programming language. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS'05)*. IEEE, 249–258.

[4] ANAKIN. 2019. https://github.com/AgentANAKIN/.

[5] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510.

[6] Alán Aspuru-Guzik and Philip Walther. 2012. Photonic quantum simulators. *Nature physics* 8, 4 (2012), 285–291.

[7] Carlos Barrón-Romero. 2015. Classical and Quantum Algorithms for the Boolean Satisfiability Problem. *arXiv preprint arXiv:1510.02682* (2015).

[8] Marcello Benedetti, Delfina Garcia-Pintos, Oscar Perdomo, Vicente Leyton-Ortega, Yunseong Nam, and Alejandro Perdomo-Ortiz. 2019. A generative modeling approach for benchmarking and training shallow quantum circuits. *npj Quantum Information* 5, 1 (2019), 1–9.

[9] Paul Benioff. 1980. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of statistical physics* 22, 5 (1980), 563–591.

[10] Charles H Bennett and Gilles Brassard. 2020. Quantum cryptography: Public key distribution and coin tossing. *arXiv preprint arXiv:2003.06557* (2020).

[11] Ethan Bernstein and Umesh Vazirani. 1997. Quantum complexity theory. *SIAM Journal on computing* 26, 5 (1997), 1411–1473.

[12] Dominic W Berry, Graeme Ahokas, Richard Cleve, and Barry C Sanders. 2007. Efficient quantum algorithms for simulating sparse Hamiltonians. *Communications in Mathematical Physics* 270, 2 (2007), 359–371.

[13] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. 2017. Quantum machine learning. *Nature* 549, 7671 (2017), 195–202.

[14] BramDo. 2017. Quantum Examples QASM. https://github.com/BramDo/quantum_examples_qasm.

[15] Cyril Branciard, Nicolas Gisin, Barbara Kraus, and Valerio Scarani. 2005. Security of two quantum cryptography protocols using the same four qubit states. *Physical Review A* 72, 3 (2005), 032301.

[16] Michael Broughton, Guillaume Verdon, Trevor McCourt, Antonio J Martinez, Jae Hyeon Yoo, Sergei V Isakov, Philip Massey, Murphy Yuezhen Niu, Ramin Halavati, Evan Peters, et al. 2020. TensorFlow Quantum: A Software Framework for Quantum Machine Learning. *arXiv preprint arXiv:2003.02989* (2020).

[17] Bikas K Chakrabarti, Amit Dutta, and Parongama Sen. 2008. *Quantum Ising phases and transitions in transverse Ising models*. Vol. 41. Springer Science & Business Media.

[18] Bo Chen, Zhigang Ye, Chen Chen, and Jianhui Wang. 2018. Toward a MILP modeling framework for distribution system restoration. *IEEE Transactions on Power Systems* 34, 3 (2018), 1749–1760.

[19] Frederic T Chong. 2018. Quantum Computing is Getting Real: Architecture, PL, and OS Roles in Closing the Gap between Quantum Algorithms and Machines. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 285–285.

[20] Lukasz Cincio, Yiğit Subaşı, Andrew T Sornborger, and Patrick J Coles. 2018. Learning the quantum algorithm for state overlap. *New Journal of Physics* 20, 11 (2018), 113022.

[21] Juan I Cirac and Peter Zoller. 1995. Quantum computations with cold trapped ions. *Physical review letters* 74, 20 (1995), 4091.

[22] B David Clader, Bryan C Jacobs, and Chad R Sprouse. 2013. Preconditioned quantum linear system algorithm. *Physical review letters* 110, 25 (2013), 250504.

[23] John Clarke and Frank K Wilhelm. 2008. Superconducting quantum bits. *Nature* 453, 7198 (2008), 1031–1042.

[24] Marshall Copeland, Julian Soh, Anthony Puca, Mike Manning, and David Gollob. 2015. Microsoft Azure. *Apress: New York, NY, USA* (2015).

[25] Don Coppersmith. 2002. An approximate Fourier transform useful in quantum factoring. *arXiv preprint quant-ph/0201067* (2002).

[26] Antonio D Córcoles, Abhinav Kandala, Ali Javadi-Abhari, Douglas T McClure, Andrew W Cross, Kristan Temme, Paul D Nation, Matthias Steffen, and JM Gambetta. 2019. Challenges and Opportunities of Near-Term Quantum Computing Systems. *arXiv preprint arXiv:1910.02894* (2019).

[27] Andrew W Cross, Lev S Bishop, Sarah Sheldon, Paul D Nation, and Jay M Gambetta. 2019. Validating quantum computers using randomized model circuits. *Physical Review A* 100, 3 (2019), 032328.

[28] Andrew W Cross, Lev S Bishop, John A Smolin, and Jay M Gambetta. 2017. Open quantum assembly language. *arXiv preprint arXiv:1707.03429* (2017). Repo: https://github.com/Qiskit/openqasm.

[29] Andrew W Cross, Graeme Smith, and John A Smolin. 2015. Quantum learning robust against noise. *Physical Review A* 92, 1 (2015), 012327.

[30] Steven A Cuccaro, Thomas G Draper, Samuel A Kutin, and David Petrie Moulton. 2004. A new quantum ripple-carry addition circuit. *arXiv preprint quant-ph/0410184* (2004).

[31] Poulami Das, Swamit S Tannu, Prashant J Nair, and Moinuddin Qureshi. 2019. A Case for Multi-Programming Quantum Computers. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 291–303.

[32] David Deutsch and Richard Jozsa. 1992. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439, 1907 (1992), 553–558.
[33] David P DiVincenzo. 2000. The physical implementation of quantum computation. *Fortschritte der Physik: Progress of Physics* 48, 9-11 (2000), 771–783.
[34] Miroslav Dobšíček, Göran Johansson, Vitaly Shumeiko, and Göran Wendin. 2007. Arbitrary accuracy iterative quantum phase estimation algorithm using a single ancillary qubit: A two-qubit benchmark. *Physical Review A* 76, 3 (2007), 030306.
[35] Yulong Dong and Lin Lin. 2020. Random circuit block-encoded matrix and a proposal of quantum LINPACK benchmark. *arXiv preprint arXiv:2006.04010* (2020).
[36] Eugene F Dumitrescu, Alex J McCaskey, Gaute Hagen, Gustav R Jansen, Titus D Morris, T Papenbrock, Raphael C Pooser, David Jarvis Dean, and Pavel Lougovski. 2018. Cloud quantum computing of an atomic nucleus. *Physical review letters* 120, 21 (2018), 210501.
[37] Wolfgang Dür, Guifre Vidal, and J Ignacio Cirac. 2000. Three qubits can be entangled in two inequivalent ways. *Physical Review A* 62, 6 (2000), 062314.
[38] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028* (2014).
[39] Edward Farhi and Sam Gutmann. 1998. Quantum computation and decision trees. *Physical Review A* 58, 2 (1998), 915.
[40] Richard P Feynman. 1999. Simulating physics with computers. *Int. J. Theor. Phys* 21, 6/7 (1999).
[41] Mark Fingerhuth. 2020. Open-Source Quantum Software Projects. https://github.com/qosf/awesome-quantum-software.
[42] Michael Fleischhauer and Mikhail D Lukin. 2002. Quantum memory for photons: Dark-state polaritons. *Physical Review A* 65, 2 (2002), 022314.
[43] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. 2012. Surface codes: Towards practical large-scale quantum computation. *Physical Review A* 86, 3 (2012), 032324.
[44] Xiang Fu, Michiel Adriaan Rol, Cornelis Christiaan Bultink, J Van Someren, Nader Khammassi, Imran Ashraf, RFL Vermeulen, JC De Sterke, WJ Vlothuizen, RN Schouten, et al. 2017. An experimental microarchitecture for a superconducting quantum processor. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. 813–825.
[45] John King Gamble, Mark Friesen, Dong Zhou, Robert Joynt, and SN Coppersmith. 2010. Two-particle quantum walks applied to the graph isomorphism problem. *Physical Review A* 81, 5 (2010), 052313.
[46] Iulia M Georgescu, Sahel Ashhab, and Franco Nori. 2014. Quantum simulation. *Reviews of Modern Physics* 86, 1 (2014), 153.
[47] Nicolas Gisin, Grégoire Ribordy, Wolfgang Tittel, and Hugo Zbinden. 2002. Quantum cryptography. *Reviews of modern physics* 74, 1 (2002), 145.
[48] Google. [n.d.]. URL: https://github.com/quantumlib/Cirq.
[49] Daniel M Greenberger, Michael A Horne, and Anton Zeilinger. 1989. Going beyond Bell's theorem. In *Bell's theorem, quantum theory and conceptions of the universe*. Springer, 69–72.
[50] Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 212–219.
[51] Lov K Grover. 1998. Quantum computers can search rapidly by using almost any transformation. *Physical Review Letters* 80, 19 (1998), 4329.
[52] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. 2009. Quantum algorithm for linear systems of equations. *Physical review letters* 103, 15 (2009), 150502.
[53] Ni-Ni Huang, Wei-Hao Huang, and Che-Ming Li. 2020. Identification of networking quantum teleportation on 14-qubit IBM universal quantum computer. *Scientific reports* 10, 1 (2020), 1–12.
[54] IBM. [n.d.]. IBM Quantum Experience. URL: https://quantum-computing.ibm.com/.
[55] IBM. [n.d.]. Qiskit: Elements for building a quantum future. URL: https://github.com/Qiskit/qiskit.
[56] Joseph T. Iosue. [n.d.]. QAOAPython: The Quantum Approximate Optimization Algorithm implemented on Cirq, ProjectQ, and Qiskit. URL: https://github.com/jtiosue/QAOAPython.
[57] Kazuo Iwama, Harumichi Nishimura, Rudy Raymond, and Junichi Teruyama. 2010. Quantum counterfeit coin problems. In *International Symposium on Algorithms and Computation*. Springer, 85–96.
[58] Kazuo Iwama, Harumichi Nishimura, Rudy Raymond, and Junichi Teruyama. 2012. Quantum counterfeit coin problems. *Theoretical Computer Science* 456 (2012), 51–64.
[59] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T Chong, and Margaret Martonosi. 2014. ScaffCC: a framework for compilation and analysis of quantum computing programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers*. 1–10. Repo: https://github.com/epiqc/ScaffCC.

[60] Tyson Jones, Suguru Endo, Sam McArdle, Xiao Yuan, and Simon C Benjamin. 2019. Variational quantum algorithms for discovering Hamiltonian spectra. *Physical Review A* 99, 6 (2019), 062304.
[61] Stephen Jordan. [n.d.]. Quantum Algorithm Zoo. URL: https://quantumalgorithmzoo.org/.
[62] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M Chow, and Jay M Gambetta. 2017. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature* 549, 7671 (2017), 242–246.
[63] Min-Sung Kang, Jino Heo, Seong-Gon Choi, Sung Moon, and Sang-Wook Han. 2019. Implementation of SWAP test for two unknown states in photons via cross-Kerr nonlinearities under decoherence effect. *Scientific reports* 9, 1 (2019), 1–14.
[64] Iordanis Kerenidis and Anupam Prakash. 2016. Quantum recommendation systems. *arXiv preprint arXiv:1603.08675* (2016).
[65] H Jeff Kimble. 2008. The quantum internet. *Nature* 453, 7198 (2008), 1023–1030.
[66] Henning Labuhn, Daniel Barredo, Sylvain Ravets, Sylvain De Léséleuc, Tommaso Macrì, Thierry Lahaye, and Antoine Browaeys. 2016. Tunable two-dimensional arrays of single Rydberg atoms for realizing quantum Ising models. *Nature* 534, 7609 (2016), 667–670.
[67] Ryan LaRose. 2019. Overview and comparison of gate level quantum software platforms. *Quantum* 3 (2019), 130.
[68] Chris Lattner and Vikram Adve. 2004. LLVM: A compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.* IEEE, 75–86.
[69] Dietrich Leibfried, Rainer Blatt, Christopher Monroe, and David Wineland. 2003. Quantum dynamics of single trapped ions. *Reviews of Modern Physics* 75, 1 (2003), 281.
[70] Gushu Li, Yufei Ding, and Yuan Xie. 2019. Tackling the qubit mapping problem for NISQ-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 1001–1014.
[71] Gushu Li, Yufei Ding, and Yuan Xie. 2020. Towards Efficient Superconducting Quantum Processor Architecture Design. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 1031–1045.
[72] Seth Lloyd. 1996. Universal quantum simulators. *Science* (1996), 1073–1078.
[73] Easwar Magesan, Jay M Gambetta, and Joseph Emerson. 2011. Scalable and robust randomized benchmarking of quantum processes. *Physical review letters* 106, 18 (2011), 180504.
[74] Easwar Magesan, Jay M Gambetta, and Joseph Emerson. 2012. Characterizing quantum gates via randomized benchmarking. *Physical Review A* 85, 4 (2012), 042311.
[75] R Maurand, X Jehl, D Kotekar-Patil, A Corna, H Bohuslavskyi, R Laviéville, L Hutin, S Barraud, M Vinet, M Sanquer, et al. 2016. A CMOS silicon spin qubit. *Nature communications* 7, 1 (2016), 1–6.
[76] Alexander J McCaskey, Eugene F Dumitrescu, Dmitry Liakh, Mengsu Chen, Wu-chun Feng, and Travis S Humble. 2018. A language and hardware independent approach to quantum–classical computing. *SoftwareX* 7 (2018), 245–254.
[77] Jarrod McClean, Nicholas Rubin, Kevin Sung, Ian David Kivlichan, Xavier Bonet-Monroig, Yudong Cao, Chengyu Dai, Eric Schuyler Fried, Craig Gidney, Brendan Gimby, et al. 2020. OpenFermion: the electronic structure package for quantum computers. *Quantum Science and Technology* (2020). Repo: https://github.com/quantumlib/OpenFermion-Cirq.
[78] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. 2016. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics* 18, 2 (2016), 023023.
[79] Raffaele Miceli. [n.d.]. Quantum_Walks: Qiskit code to simulate quantum walks on graphs with up to 4 nodes. URL: https://github.com/raffmiceli/Quantum_Walks.
[80] Kristel Michielsen, Madita Nocon, Dennis Willsch, Fengping Jin, Thomas Lippert, and Hans De Raedt. 2017. Benchmarking gate-based quantum computers. *Computer Physics Communications* 220 (2017), 44–55.
[81] Ashley Montanaro. 2016. Quantum algorithms: an overview. *npj Quantum Information* 2, 1 (2016), 1–8.
[82] WJ Munro, KA Harrison, AM Stephens, SJ Devitt, and Kae Nemoto. 2010. From quantum multiplexing to high-performance quantum networking. *Nature Photonics* 4, 11 (2010), 792.
[83] Prakash Murali, Jonathan M Baker, Ali Javadi-Abhari, Frederic T Chong, and Margaret Martonosi. 2019. 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*. 1015–1029.
[84] Prakash Murali, Norbert Matthias Linke, Margaret Martonosi, Ali Javadi Abhari, Nhung Hong Nguyen, and Cinthia Huerta Alderete. 2019. Full-stack, real-system quantum computer studies: architectural comparisons and design insights. In *Proceedings of the 46th International Symposium on Computer Architecture*. 527–540.
[85] Prakash Murali, David C McKay, Margaret Martonosi, and Ali Javadi-Abhari. 2020. Software Mitigation of Crosstalk on Noisy Intermediate-Scale Quantum Computers. *arXiv preprint arXiv:2001.02826* (2020).
[86] Michael A Nielsen and Isaac Chuang. 2002. Quantum computation and quantum information.

[87] Jeremy L O'brien, Akira Furusawa, and Jelena Vučković. 2009. Photonic quantum technologies. *Nature Photonics* 3, 12 (2009), 687.

[88] Giuseppe Davide Paparo and MA Martin-Delgado. 2012. Google in a quantum network. *Scientific reports* 2 (2012), 444.

[89] Daniel K Park, June-Koo K Rhee, and Soonchil Lee. 2018. Noise-tolerant parity learning with one quantum bit. *Physical Review A* 97, 3 (2018), 032327.

[90] Tirthak Patel, Abhay Potharaju, Baolin Li, Rohan Roy, and Devesh Tiwari. 2020. Experimental evaluation of NISQ quantum computers: error measurement, characterization, and implications. In *2020 SC20: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, 636–650.

[91] Jarryd J Pla, Kuan Y Tan, Juan P Dehollain, Wee H Lim, John JL Morton, David N Jamieson, Andrew S Dzurak, and Andrea Morello. 2012. A single-atom electron spin qubit in silicon. *Nature* 489, 7417 (2012), 541–545.

[92] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (2018), 79.

[93] Timothy J Proctor, Arnaud Carignan-Dugas, Kenneth Rudinger, Erik Nielsen, Robin Blume-Kohout, and Kevin Young. 2019. Direct randomized benchmarking for multiqubit devices. *Physical review letters* 123, 3 (2019), 030503.

[94] Quantiki. 2020. List of QC simulators. https://www.quantiki.org/wiki/list-qc-simulators.

[95] Patrick Rebentrost, Brajesh Gupt, and Thomas R Bromley. 2018. Quantum computational finance: Monte Carlo pricing of financial derivatives. *Physical Review A* 98, 2 (2018), 022321.

[96] Rigetti. [n.d.]. Rigetti: Think Quantum. URL: https://rigetti.com/.

[97] Chad Rigetti, Jay M Gambetta, Stefano Poletto, BLT Plourde, Jerry M Chow, AD Córcoles, John A Smolin, Seth T Merkel, JR Rozen, George A Keefe, et al. 2012. Superconducting qubit in a waveguide cavity with a coherence time approaching 0.1 ms. *Physical Review B* 86, 10 (2012), 100506.

[98] Gonçalo Sampaio. 2017. Code in QASM for quantum circuits and algorithms. https://github.com/sampaio96/Quantum-Computing.

[99] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. 2015. An introduction to quantum machine learning. *Contemporary Physics* 56, 2 (2015), 172–185.

[100] Yunong Shi, Nelson Leung, Pranav Gokhale, Zane Rossi, David I Schuster, Henry Hoffmann, and Frederic T Chong. 2019. Optimized compilation of aggregated instructions for realistic quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 1031–1044.

[101] Peter W Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 124–134.

[102] Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review* 41, 2 (1999), 303–332.

[103] Daniel R Simon. 1997. On the power of quantum computation. *SIAM journal on computing* 26, 5 (1997), 1474–1483.

[104] Kaitlin N Smith and Mitchell A Thornton. 2019. A quantum computational compiler and design tool for technology-specific targets. In *Proceedings of the 46th International Symposium on Computer Architecture*. 579–588.

[105] Damian S Steiger, Thomas Häner, and Matthias Troyer. 2018. ProjectQ: an open source software framework for quantum computing. *Quantum* 2 (2018), 49.

[106] Samuel A Stein. 2021. Quantum Computing Aided Machine Learning Through Quantum State Fidelity. (2021).

[107] Juexiao Su, Tianheng Tu, and Lei He. 2016. A quantum annealing approach for boolean satisfiability problem. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[108] Swamit S Tannu and Moinuddin Qureshi. 2019. Ensemble of Diverse Mappings: Improving Reliability of Quantum Computers by Orchestrating Dissimilar Mistakes. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 253–265.

[109] Swamit S Tannu and Moinuddin K Qureshi. 2019. Mitigating Measurement Errors in Quantum Computers by Exploiting State-Dependent Bias. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 279–290.

[110] Swamit S Tannu and Moinuddin K Qureshi. 2019. Not all qubits are created equal: a case for variability-aware policies for NISQ-era quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 987–999.

[111] Tommaso Toffoli. 1980. Reversible computing. In *International Colloquium on Automata, Languages, and Programming*. Springer, 632–644.

[112] Vlatko Vedral, Adriano Barenco, and Artur Ekert. 1996. Quantum networks for elementary arithmetic operations. *Physical Review A* 54, 1 (1996), 147.

[113] Salvador Elías Venegas-Andraca. 2012. Quantum walks: a comprehensive review. *Quantum Information Processing* 11, 5 (2012), 1015–1106.

[114] James D Whitfield, Jacob Biamonte, and Alán Aspuru-Guzik. 2011. Simulation of electronic structure Hamiltonians using quantum computers. *Molecular Physics* 109, 5 (2011), 735–750.

[115] Michael Wilde, Mihael Hategan, Justin M Wozniak, Ben Clifford, Daniel S Katz, and Ian Foster. 2011. Swift: A language for distributed parallel scripting. *Parallel Comput.* 37, 9 (2011), 633–652.

[116] Stefan Woerner and Daniel J Egger. 2019. Quantum risk analysis. *npj Quantum Information* 5, 1 (2019), 1–8.

[117] Muqing Zheng, Ang Li, Tamás Terlaky, and Xiu Yang. 2020. A bayesian approach for characterizing and mitigating gate and measurement errors. *arXiv preprint arXiv:2010.09188* (2020).

[118] Leo Zhou, Sheng-Tao Wang, Soonwon Choi, Hannes Pichler, and Mikhail D Lukin. 2018. Quantum approximate optimization algorithm: performance, mechanism, and implementation on near-term devices. *arXiv preprint arXiv:1812.01041* (2018).