2QAN: A quantum compiler for 2-local qubit Hamiltonian simulation algorithms

Lingling Lao
Department of Physics and Astronomy, University College London
London, United Kingdom

Dan E. Browne
Department of Physics and Astronomy, University College London
London, United Kingdom

ABSTRACT
Simulating quantum systems is one of the most important potential applications of quantum computers. The high-level circuit defining the simulation needs to be compiled into one that complies with hardware limitations such as qubit architecture (connectivity) and instruction (gate) set. General-purpose quantum compilers work at the gate level and have little knowledge of the mathematical properties of quantum applications, missing further optimization opportunities. Existing application-specific compilers only apply advanced optimizations in the scheduling procedure and are restricted to the CNOT or CZ gate set. In this work, we develop a compiler, named 2QAN, to optimize quantum circuits for 2-local qubit Hamiltonian simulation problems, a framework which includes the important quantum approximate optimization algorithm (QAOA). In particular, we exploit the flexibility of permuting different operators in the Hamiltonian (no matter whether they commute) and propose permutation-aware techniques for qubit routing, gate optimization and scheduling to minimize compilation overhead. 2QAN can target different architectures and different instruction sets. Compilation results on four applications (up to 50 qubits) and three quantum computers (namely, Google Sycamore, IBMQ Montreal and Rigetti Aspen) show that 2QAN outperforms state-of-the-art general-purpose compilers and application-specific compilers. Specifically, 2QAN can reduce the number of inserted SWAP gates by 11.5X, reduce overhead in hardware gate count by 68.5X, and reduce overhead in circuit depth by 21X. Experimental results on the Montreal device demonstrate that benchmarks compiled by 2QAN achieve the highest fidelity.

CCS CONCEPTS
- Hardware → Quantum computation; • Software and its engineering → Compilers.

KEYWORDS
Quantum computing, quantum simulation, quantum compilation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:
Lingling Lao and Dan E. Browne. 2022. 2QAN: A quantum compiler for 2-local qubit Hamiltonian simulation algorithms. In The 49th Annual International Symposium on Computer Architecture (ISCA ’22), June 18–22, 2022, New York, NY, USA. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3470496.3527394

1 INTRODUCTION
Near-term quantum computers have a small number of qubits (tens to thousands) and non-negligible gate errors, making it impractical to implement conventional quantum algorithms such as Shor’s factoring algorithm [1] which requires millions of qubits for a fault-tolerant implementation [2]. Quantum simulation (or Hamiltonian simulation) as first proposed by Richard Feynman [3] may require fewer qubits and be one of the first practical applications of quantum computers. Quantum simulation has broad applications in understanding the behaviour of physical systems in areas such as condensed-matter physics [4], high-energy physics [5], cosmology [6], quantum chemistry [7], and quantum field theory [8] and has been demonstrated in different quantum technologies [9, 10] (see [11] for a detailed review). Additionally, quantum simulation has been used to design new algorithms for solving linear systems [12], semidefinite programs [13], quantum walks [14], etc. Its main challenge is to find an efficient circuit that asymptotically approximates the time evolution of a Hamiltonian. The product formula approach has a straightforward implementation [15–17] and good performance in practice [18, 19]. Given a Hamiltonian that is decomposed as a sum of Hermitian terms (\(H = \sum_{j=1}^{L} h_j H_j\)), the product formula approximates the exponential of this Hamiltonian as a product of exponentials of individual terms and each exponential can be efficiently realized by a quantum circuit.

This high-level circuit representation is typically hardware agnostic and needs to be decomposed into the instruction set supported by the underlying quantum hardware. In noisy intermediate-scale quantum (NISQ) computers [20], the universal instruction set is normally composed of arbitrary single-qubit rotations plus one or a few two-qubit gates (e.g., the SYC, CNOT, iSWAP gates from Google, IBM, Rigetti respectively in Figure 1). Furthermore, these quantum computers only allow two-qubit gates between specific qubit pairs, i.e., there is limited qubit connectivity.Compilation techniques are required to map circuit qubits onto hardware qubits and insert SWAP gates to move qubits into neighbouring positions, increasing circuit size in terms of gate count and circuit depth. Two-qubit gates have much higher error rates than single-qubit rotations and qubits have short coherence time [21]. Therefore, it is critical to minimize compilation overhead for high-fidelity circuit implementation.
Many approaches have been proposed for compiling quantum circuits onto NISQ computers [24–31]. These compilers operate at the gate level and are designed for general circuits with little knowledge of the mathematical properties of the target applications. Exploiting the synergy between applications and compilation techniques can provide more optimizations of the implementation circuit, which will further improve application performance. For quantum simulation problems, the ordering of different terms in the Hamiltonian is arbitrary and one could permute the exponents of these terms without losing computational accuracy. This is, however, difficult or even impossible for a compiler to recognize at gate level, especially when many of these operators do not commute with each other (changing the order of non-commuting gates violates program semantics). Several application-specific compilers have also been developed [32–40], but their optimization techniques are performed on one particular compilation pass and are limited to the CNOT or CZ gate set.

In this work, we identify the flexibility in the Hamiltonian operator permutation and exploit it in the compilation procedure. In particular, we propose permutation-aware qubit routing, gate scheduling, and gate optimization techniques to efficiently compile circuits for 2-local qubit Hamiltonian simulation problems. The quantum approximate optimization algorithm (QAOA) [41] also has Hamiltonians in this form. The developed compiler, named 2QAN, can target different qubit architecture (topologies) and different gate sets. Evaluation results on three quantum computers show that 2QAN can significantly reduce gate count and circuit depth compared to state-of-the-art quantum compilers. Furthermore, we experimentally demonstrate the advantages of 2QAN on the IBMQ Montreal device. The evaluation toolflow is presented in Figure 2.

The main contributions of this work are:

- We discover an optimization opportunity for compiling quantum simulation problems on NISQ computers, i.e., the order of Hamiltonian terms is flexible and a quantum compiler can permute their exponentials (operators in the product formula) to minimize compilation overhead.
- We exploit this flexibility and develop 2QAN, a quantum compiler for efficiently compiling 2-local qubit Hamiltonian simulation problems on NISQ computers. We first propose a permutation-aware qubit routing heuristic to minimize the number of inserted SWAP gates. Then we implement a unitary unifying pass that combines a SWAP gate with a circuit gate to further reduce gate overhead. Moreover, we design a permutation-aware gate scheduling technique to minimize circuit depth. The routing and scheduling algorithms have quadratic time complexity in the number of gates and are scalable for large systems.
- We perform all the proposed permutation-aware compilation passes prior to the gate decomposition pass as shown in Figure 2. That is, these optimization algorithms are independent of the underlying hardware two-qubit gates, allowing 2QAN to target different instruction sets.
- We evaluate the proposed compiler by compiling the Heisenberg model, XY model, Ising model, and the QAOA circuits onto three industrial quantum computers, Google Sycamore [21], IBMQ Montreal [22], Rigetti Aspen [23]. Across all

1 Pronounced "toucan". We propose the name 2QAN since this compiler targets 2-local Qubit Hamiltonian simulation, is hardware-adapted and designed for NISQ quantum devices. https://github.com/lllingoo/2QAN
2 BACKGROUND

2.1 Quantum simulation

Circuit construction: Consider a system Hamiltonian that is decomposed into a sum of polynomially many Hermitian terms, 

\[ H = \sum_{j=1}^{L} h_j H_j, \]

its time evolution can be described by the unitary \( U(t) = \exp(itH) \). The goal is to find an efficient circuit construction for \( U(t) \). It is hard to decompose this unitary directly when some terms anti-commute (which is typically the case in physical systems of interest), i.e., if \( H_j H_k \neq H_k H_j \), then \( \exp(itH) \neq \prod_{j=1}^{J} \exp(ith_j H_j) \). Alternatively, one can use the product formula approach \([V(\Delta t)]^{\frac{1}{\Delta t}} \) to asymptotically approximate \( \exp(itH) \) [15–17], where

\[ V(\Delta t) = \prod_{j=1}^{L} \exp(i\Delta t h_j H_j). \] (1)

\( V(\Delta t) \) is called one Trotterization step in which each exponential operator \( \exp(i\Delta t h_j H_j) \) can be more easily decomposed into a hardware gate set. \( V(\Delta t) \) is repeated \( \frac{1}{\Delta t} \) times to construct the entire circuit and choosing small \( \Delta t \) will suppress approximation errors.

Operator permutation: We note that the order of the \( L \) individual operators in each Trotter step (Equation 1) is arbitrary and can be chosen from any of the \( L! \) permutations at the application level [42, 43]. This permutation flexibility allows one to perform optimized compilation for the Hamiltonian simulation circuit. In contrast, general-purpose quantum compilers assume a specific order and will not allow such rearrangement when operators do not commute.

Our work focuses on developing a permutation-aware compiler for optimizing each Trotter step, which will improve the overall circuit construction of quantum simulation since a large number of such repetitions need to be performed. This compiler can also be applied to the QAOA benchmarks [41] and the problem-inspired variational quantum eigensolver (VQE) ansatz [44] which have similar circuit constructions to Equation 1 except that \( \Delta t \) is replaced by adjustable parameters. Other versatile VQE ansatzes such as the hardware-efficient ansatz [45] need to be compiled by generic techniques.

2-local qubit Hamiltonian: In this work we consider 2-local qubit Hamiltonians that originate in many physical systems,

\[ H = \sum_{(u,v) \in E} H_{uv} + \sum_{k \in V} H_k. \] (2)

\( H_{uv} \) are two-qubit Hamiltonian terms and \( H_k \) are single-qubit Hamiltonians. The interaction graph of this Hamiltonian is represented by \( G(V, E) \), \( V \) is the set of qubits and \( E \) is the set of edges. The transverse Ising model, XY model, and Heisenberg model all have Hamiltonians in this form,

\[ H_{\text{Ising}} = \sum_{(u,v) \in E} \gamma_{uv} Z_u Z_v + \sum_{k \in V} \beta_k X_k, \] (3)

\[ H_{\text{XY}} = \sum_{(u,v) \in E} (\alpha_{uv} X_u X_v + \beta_{uv} Y_u Y_v), \] (4)

\[ H_{\text{Heisenberg}} = \sum_{(u,v) \in E} (\alpha_{uv} X_u X_v + \beta_{uv} Y_u Y_v + \gamma_{uv} Z_u Z_v), \] (5)

where \( X,Y,Z \) are the Pauli operators. These are important models in the study of many-body physics and have been used for solving various problems. For example, the Heisenberg model is used to study magnetic systems in which the interacting particles have opposing spins [46]. The Ising model can be exploited for studying spontaneous magnetization in ferromagnetic film [47], neuroscience [48], and spin glasses [49]. The famous QAOA for solving combinatorial optimization problems also has the Ising model Hamiltonian [41].

2.2 Circuit compilation

NISQ computers have hardware limitations such as their native gate set and qubit connectivity, making general quantum circuits not directly executable. A hardware gate set is typically composed of arbitrary single-qubit rotations and a few two-qubit gates which vary across QC vendors. For example, IBM devices currently have the CNOT as the native two-qubit gate [22], Rigetti implement both CZ and iSWAP gates [23, 50], and Google support CZ, SYC, \( \sqrt{\text{SWAP}} \) gates [21, 51]. High-level quantum circuits need to be decomposed into the given gate set by using analytical algorithms [52, 53] or numerical approaches [54, 55]. Furthermore, two-qubit gates can only be executed on the connected (nearest neighbouring, NN) qubits. Movement operations such as SWAP gates need to be inserted for performing non-NN gates, increasing the number of gates and circuit depth. NISQ computers have limited qubit coherence time and high gate error rates. It is crucial to minimize circuit sizes for the reliable implementation of quantum applications.

To this purpose, quantum compilation techniques, including qubit mapping and routing, gate decomposition and scheduling, have been developed to efficiently transform high-level quantum circuits into hardware-compatible ones. For general quantum circuits, the routing and scheduling algorithms assign dependencies between gates to maintain the correctness of the program semantics [24–31]. These dependencies are typically generated based on the gate order appearing in an input circuit. These general-purpose quantum compilers work at the gate level and do not consider application-level
Figure 3: Examples of compiling a 6-qubit 2-local Hamiltonian (Equation 2) to a $2 \times 3$ grid architecture. (a) The circuit for one Trotter step, where the two(single)-qubit operators implement the evolution of two(single)-qubit Hamiltonians and two-qubit operators do not commute. (b-c) Compilation procedure. Top figures show the compiled circuits and bottom figures show the qubit mappings (nodes are qubits and edges represent their connectivity). Gates between dashed lines can be performed in parallel. Qubits in (b-c) always correspond to the circuit qubits in (a) for better readability. (b) A generic compiler respects the gate dependencies in (a). (c) The 2QAN compiler exploits the operator permutation flexibility and reduces the two-qubit gate count from 12 to 9 and the circuit depth from 7 to 5 (Each of inserted SWAPs is merged with a circuit unitary in gray area).

Compilation opportunities for Hamiltonian simulation: As mentioned previously, when constructing a circuit for each Trotter step, one is free to use any permutation of the operators in the product formula (even when they do not commute). This application-level property allows additional optimizations for Hamiltonian simulation problems but is not exploited by general-purpose quantum compilers. Figure 3 shows an example of compiling a 6-qubit Hamiltonian circuit to a $2 \times 3$ grid architecture. The compiler, considering the flexibility in operator permutation, places all the nearest-neighboring (NN) gates in the initial qubit map even though they are placed in later timesteps in the input circuit, avoiding unnecessary SWAP gates (e.g., the orange gate on q1 and q2 is moved to an earlier cycle in (c)). A generic compiler respects the gate dependency in the input circuit and cannot reorder non-commuting gates (e.g., the orange gate is scheduled in the last cycle in (b) and requires a SWAP gate for NN implementation). Furthermore, this permutation flexibility allows one to reschedule a circuit gate to the timestep of a SWAP gate acting on the same qubits (e.g., the blue gates are moved to the gray area in (c)) and then merge/unify these two gates into one gate, further decreasing gate count. In this work, we design such an application-specific compiler for 2-local qubit Hamiltonian simulation problems to improve application fidelity.

3 COMPILATION TECHNIQUES

In this section, we introduce the proposed compilation techniques underlying 2QAN. Figure 2 shows the overview of our 2QAN compiler.

3.1 Qubit mapping

The goal of qubit mapping is to find an optimal qubit initial placement such that the number of qubit-moving operations required for implementing all two-qubit gates is minimized. Similar to the approaches in [56–58], the qubit mapping problem is formulated as a quadratic assignment problem (QAP). That is, the problem of allocating each qubit in the circuit (facility) to one qubit in the device (location) with the cost defined by a function of the distance and interaction times (flow) between circuit qubits. Let $n$ be the number of circuit qubits or physical qubits and $N = \{1, 2, \cdots, n\}$ denotes the qubit set. The objective function is

$$\min_{\phi \in S_n} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{\phi(i)\phi(j)}$$

(6)

where $S_n$ is the set of all permutations $\phi: N \rightarrow N$, $f_{ij}$ is the number of two-qubit gates between circuit qubits $i$ and $j$, $d_{\phi(i)\phi(j)}$ is the distance between hardware qubits $\phi(i)$ and $\phi(j)$ and is calculated by using the Floyd-Warshall algorithm.

QAP is an NP-hard problem [59] and we use the Tabu search heuristic algorithm [60, 61] to efficiently find good qubit mappings in this work. Other heuristics such as simulated annealing [62] and a greedy randomized adaptive search [63] can also be used for solving this problem. Prior works observed that the QAP formulation of qubit mapping may not work well for general quantum circuits [56–58]. This is because qubits need to interact in a specific order (gate dependency) and the initial mapping benefits diminish after insertion of SWAPs, e.g., some NN qubits may be moved further apart but will interact later (see the orange gate in Figure 3(b)). This is not the case for 2-local Hamiltonian simulation problems as any operator that is NN in a qubit map can be scheduled directly regardless of their order in the circuit (see the orange gate in Figure 3(c)). Thus, at this stage, we are already exploiting the permutation flexibility of this application.
3.2 Qubit routing

Generally, not all two-qubit gates are nearest neighbouring in an initial qubit map. Operations such as SWAP gates need to be inserted to move non-NN qubits, causing overheads in gate count and circuit depth. Efficient qubit routing techniques are required to minimize the compilation overhead. Different from existing qubit routing algorithms which respect the gate order in the input circuit [25, 28, 64, 65], the routing in 2QAN exploits the operator permutation flexibility in a Hamiltonian. The pseudocode is in Algorithm 1 (permutation-aware steps are highlighted with *) and an example is shown in Figure 3c.

Given an input circuit that implements one Trotter step of a Hamiltonian, our routing algorithm starts by searching all the two-qubit gates that are NN in an initial qubit layout (e.g., there are 7 NN two-qubit gates in Figure 3c). These gates are directly mapped no matter which timesteps they are scheduled in in the initial circuit (permutation flexibility). SWAP gates are needed to perform the remaining two-qubit gates (Lines 2-3), e.g., red gates on (q0, q3) and (q4, q5) in Figure 3c. For these non-NN gates, 2QAN compares their qubit distances in the hardware (Equation 6) and selects the shortest-distance one to route first (Line 5). If there are multiple shortest ones, it selects the first one in this set. Then the routing algorithm finds all possible SWAP gates that act on one of the qubits of the chosen gate g (Line 6). It evaluates these SWAPs based on a SWAP selection criteria (will be explained shortly) and selects the best one and adds it to the circuit (Line 7). This new SWAP gate updates the qubit map and NN gates are found for this map and are removed from the un-routed gate set (Lines 8-10). This procedure (Lines 5-10) is repeated until all two-qubit gates have been performed. The time complexity of the routing algorithm is \( O(m^2n) \), \( m \) is the number of two-qubit gates and \( n \) is the number of qubits.

Algorithm 1 Permutation-aware routing

Input: Un-routed circuit, initial map \( \phi_0 \), device topology

Output: Routed circuit, a set of qubit maps \( \{ \phi_i \} \) and a set of NN gates corresponding to each map \( \{ G_{\phi_i} \} \)

1. Initialize the set of qubit maps \( \Phi = \{ \phi_0 \} \)
2. *Initialize the set of NN gates for each map, \( G = \{ G_{\phi_i} \} \), \( G_{\phi_0} \) ← all NN two-qubit gates for map \( \phi_0 \)
3. Initialize \( G_{\text{ur}} \) ← all un-routed (non-NN) two-qubit gates
4. while \( G_{\text{ur}} \neq \emptyset \) do
5. *Select the gate \( g \in G_{\text{ur}} \) that has shortest distance in \( \phi_i \)
6. \( S_g \) ← Find all SWAP gates on qubits in \( g \)
7. *Select the best SWAP from \( S_g \) and add it to \( G_{\phi_i} \)
8. Update qubit map from \( \phi_i \) to \( \phi_{i+1} \)
9. *\( G_{\phi_{i+1}} \) ← Find NN gates in \( G_{\text{ur}} \) for map \( \phi_{i+1} \)
10. *Remove all gates in \( G_{\phi_{i+1}} \) from \( G_{\text{ur}} \), add \( G_{\phi_{i+1}} \) to \( G \), add \( \phi_i \) to \( \Phi \)
11. end while

SWAP selection criteria: The best SWAP gate is chosen based on three criteria:

1. Least SWAP count: It will lead to the minimal cost in Equation 6 (i.e., the minimal number of SWAP gates) for the remaining non-NN gates.
2. Shortest circuit depth: It can be most interleaved with previously mapped gates, introducing the least depth overhead.
3. Best gate optimization: It can be merged with a circuit gate, i.e., if there is a circuit gate applied on the same qubits as this SWAP, the compiler will replace these two gates by a single unitary representing their product (more details will be introduced in the next section).

For our compiler configuration, we will evaluate the best SWAP based on the three criteria in the above priority order. Evaluating these criteria in a different order may further improve the compiler but will not be explored here. When there are multiple best options, a random one will be chosen. For example, both SWAP (0,3) and SWAP (2,3) in map \( \phi_0 \) of Figure 3c can be added for performing the two-qubit gate on pair (0,2) and they have the same cost regarding the first two criteria. SWAP (2,3) is selected because it can be combined with a circuit gate that operate on the same qubits. For implementing the circuit gate on (4,5), both SWAP (1,5) and SWAP (1,4) are the best, and the compiler randomly selects one.

3.3 Unitary unifying

SWAP unitary unifying: For each added SWAP gate, the compiler searches over all circuit gates. If there is a circuit gate operating on the same qubit pair as this SWAP, the compiler reschedules the circuit gate to the SWAP gate cycle and unifies them into a single unitary (see the example in Figure 4). Such rescheduling is allowed because of the operator permutation flexibility in Hamiltonian simulation. The unified unitary (referred as a dressed SWAP) is the product of the circuit gate unitary and the SWAP unitary. This unitary unifying helps further reduce the compilation overhead. Figure 5 shows the decomposition of a SWAP gate, a circuit gate \( \exp(i\theta ZZ) \), and their unified unitary into CNOT gates and single-qubit rotations. The unified unitary requires 3 CNOTs while the implementation of these two gates with individual decompositions (in the left circuit in Figure 4) would require 5 CNOTs in total.

Circuit unitary unifying: Similarly, we merge all circuit gates that act on the same qubit pair into one single unitary. For example, there are three two-qubit Pauli terms on one qubit pair in the Heisenberg model (Equation 5). The exponential of a two-qubit Pauli operator normally requires 2 CNOTs. Implementing each of these three exponentials individually would use 6 CNOTs in total and compilation based on this implementation (e.g., as used in the Paulihedral compiler [40]) may be sub-optimal. In contrast, the unified unitary only requires 3 CNOTs since any two-qubit gate can be implemented by at most 3 CNOTs [66, 67]. To reduce gate count, we preprocess all 2-local Hamiltonian simulation circuits...
using circuit unitary unifying prior to other compilation passes (not shown in Figure 2).

3.4 Gate scheduling

Scheduling without dependency: When connectivity limitations are not considered, the individual operators in one Trotter step (Equation 1) can be performed in any order. Graph coloring algorithms [68] can be used to schedule such circuits. In the graph construction, nodes represent operators (gates), and two nodes are connected by an edge if they have common qubits and cannot be scheduled in parallel. We use the default greedy algorithm in NetworkX version 2.5 for our scheduling pass. The circuits without taking into account topology constraints are scheduled by this method and are baseline circuits used for calculating compilation overhead.

Scheduling with dependency: For connectivity-constrained quantum computers, the routing pass in previous section will be applied. The router outputs a list of qubit maps and a set of NN gates corresponding to each map, including both circuit gates and (dressed) SWAP gates. The order of a (dressed) SWAP gate and a circuit gate cannot be exchanged if the SWAP is inserted to make the circuit gate NN. One can generate a gate dependency graph based on the gate order after qubit routing and apply conventional scheduling algorithms [25, 26] to minimize circuit depth. This approach may perform sufficiently well for some circuits such as the one in Figure 3c, but more optimizations can be achieved for other circuits by considering the operator permutation in Hamiltonian simulation. In this work, we apply such application-specific optimizations and the pseudocode of the proposed gate scheduling algorithm is presented in Algorithm 2 (permutation-aware steps are highlighted with *) and an example is shown in Figure 6.

Hybrid scheduling: Algorithm 2 is a hybrid of the above two scheduling techniques. The flexible operator permutation property implies that circuit gates can be scheduled in any qubit map where they are NN. The NN circuit gates in the initial qubit map do not have dependencies and are first scheduled using the graph coloring algorithm (Line 1). For other qubit maps, dependencies between SWAP and circuit gates need to be respected. The algorithm initializes at the circuit cycle \( t = 0 \). We implement an as-late-as-possible (ALAP) schedule so the algorithm assigns the final qubit map to cycle 0 (\( M_0 \), Line 2). For cycle \( t \), it first finds all circuit gates that can be scheduled (Lines 6-8). A circuit gate can be scheduled at \( t \) only if its qubits are NN in map \( M_t \) and these qubits are not currently occupied by any other gates. For example, the gate on (1,3) in Figure 6b is scheduled at cycle 0 while it needs to be scheduled at a different cycle in Figure 6a because of the dependency constraint when using a generic scheduler (which cannot schedule it until its predecessor gate on (1,5) is performed). Afterwards, the scheduler finds all SWAP gates that can be scheduled at this cycle (Lines 9-12). Except the qubit NN and availability requirements, a SWAP gate can be scheduled at \( t \) only if the circuit gates that depends on it have

![Figure 5: Examples of decomposing the SWAP, the unitary \( \exp(i\theta ZZ) \), and their product into CNOTs and single-qubit rotations.](image)

![Algorithm 2 Permutation-aware scheduling](image)

![Figure 6: ALAP scheduling examples. (a) A generic scheduler respects the gate order provided by the routing pass (takes 4 cycles). (b) The hybrid scheduler considers the flexibility of permuting circuit gates and respects the dependency between a SWAP gate and its corresponding circuit gates (takes 3 cycles). For example, it schedules gate on pair (1,3) at cycle 0 because it is a NN circuit gate in map \( \phi_1 \) while a generic scheduler will not schedule it until its predecessor (gate on (1,5)) is performed.](image)
been scheduled (e.g., red gates depend on the two SWAPs in Figure 6b). Once a SWAP is inserted, the qubit map for next cycle will be updated. The procedure in Lines 6-13 is repeated until all gates are scheduled. Finally, the algorithm reverses the scheduled gate sequence (Line 15). The time complexity of this hybrid scheduling algorithm scales quadratically with the number of gates.

All the permutation-aware compiler passes do not depend on the native hardware gates and are performed prior to the gate decomposition procedure (Figure 2), providing the flexibility that 2QAN can target different instruction sets. Other application-specific compilers such as the QAOA compiler [36–38] and the Paulihedral compiler [40] are restricted to the CNOT/CZ gate set and lack optimizations in the routing and unitary unifying procedure.

4 EXPERIMENTAL SETUP

Benchmarks: We consider Hamiltonians of a linear array of qubits with nearest neighbouring (NN) and next nearest neighbouring (NNN) interactions for the transverse Ising model, XY model, and Heisenberg model. They are denoted NNN Ising, NNN XY, and NNN Heisenberg. The time evolution of a Hamiltonian is implemented using the product formula \( U(t) = e^{-iHt} = \prod_{j=1}^{L} e^{-iH_jt/r} \), where \( r \) is the number of Trotter steps. The coefficients of \( H_j \) are randomly sampled from \((0, \pi)\). The number of two-qubit operators for NNN Ising, NNN XY, and NNN Heisenberg models in each Trotter step is \( 2n - 3 \), where \( n \) is the number of qubits ranging from 6 to 50 in our evaluation.

In addition, we also use QAOA [41] for solving the MAX-CUT problems on 3-regular graphs (QAOA-REG-3) as our benchmark. QAOA is a popular benchmark for testing the performance of quantum computers and has been demonstrated in different quantum processors [69, 70]. In particular, the connectivity differs for each problem graph, which is well suited for evaluating compilation techniques. QAOA has the same form as the Ising model, its problem Hamiltonian is \( C = \sum_{(u,v) \in E} Z_u Z_v \) and drive Hamiltonian is \( B = \sum_{k \in V} X_k \). The circuit implementation of one-layer QAOA is

\[
U(\gamma, \beta) = \prod_{(u,v) \in E} \exp(i\gamma Z_u Z_v) \prod_{k \in V} \exp(i\beta X_k),
\]

Different from Hamiltonian simulation, these parameters \((\gamma, \beta)\) differ in every layer. We randomly sample 10 graph instances for each problem size. The operator parameters for each instance are chosen at their theoretically optimal values and are calculated using the tools provided in Cirq [71]. The number of two-qubit operators in one-layer of QAOA-REG-3 is \( 3n/2 \), we consider qubit numbers \( n \) from 4 to 22.

Quantum computers: We compile these benchmarks onto three quantum computers, Google Sycamore [21], IBMQ Montreal [22], Rigetti Aspen [23]. As shown in Figure 1, Sycamore has a grid architecture with SYC as hardware two-qubit gate, Montreal has a dodecagon lattice with CNOT as native gate, Aspen has connected octagons and iSWAP as native gate. All three devices support arbitrary single-qubit rotations. Experiments on Montreal were performed on 29th October, 2021, the average CNOT error rate was 1.241%, average read-out error rate was 1.832%, and average T1=87.75 us and T2=72.65 us.

Quantum compilers: We first compare our 2QAN compiler with two state-of-the-art general-purpose compilers, the tket compiler version 0.11.0 with the recommended ‘FullPass’ [30] and the Qiskit compiler version 0.26.2 with optimization level 3 [31]. We then compare 2QAN with two application-specific compilers, the QAOA compiler (IC-QAOA) with default settings [36–38] and the Paulihedral compiler for quantum simulation [40]. For circuits with larger number of qubits, the default mapping in tket may fail to find a qubit initial placement and we use their ‘LinePlacement’ pass instead. tket and Qiskit have advanced circuit optimizations for the CNOT or CZ gates. For devices that have different hardware two-qubit gates, we disable the gate decomposition pass in Qiskit and tket. The 2QAN compiler always performs permutation-aware passes prior to gate decomposition. The mapped circuits that have application-level units will need to be decomposed into hardware gate sets. We apply the ‘SynthesiseIBM’ decomposition pass in tket to decompose the mapped circuits by 2QAN for the Montreal device. We use the analytical method in Cirq [72] to decompose QAOA and Ising units into SYC gates. For other application units and hardware gates, we use the numerical approach developed in [55] for finding more efficient decomposition. Both Qiskit and 2QAN involve randomization in the mapping procedure, we run their mapping passes 5 times and choose the best results. We also pre-process the input circuits for tket and Qiskit by applying the circuit unitary unifying in Section 3.3 to reduce compilation overhead.

Metrics: Similar to prior works, we use the number of inserted SWAP gates (smaller is better), the number of hardware two-qubit gates (smaller is better), the depth of two-qubit gates (shorter is better), and the depth of all gates (shorter is better) as metrics to compare the performance of different compilers. We also calculate the increase in gate count and circuit depth (i.e., compilation overhead, less is better) compared to the circuits without considering connectivity constraints (i.e., the baseline implementation). Furthermore, we experimentally evaluate the application performance of QAOA benchmarks on the Montreal device. The performance is measured by the normalized cost function \( C / C_{\text{min}} \) (larger is better) [70]. 1 means the perfect result and 0 corresponds to the random guessing result.

Implementation: We implement 2QAN in Python 3.8. All compilation in our evaluation was performed on a laptop with an Intel Core i7 processor (2.30GHz and 32GB RAM).

5 COMPILER EVALUATION

5.1 Reducing compilation overhead compared to general-purpose compilers

In this section, we compare the compilation overhead of our 2QAN compiler with tket and Qiskit. Figures 7, 8, 9 show the compilation results on Google Sycamore, Rigetti Aspen, and IBM Montreal, respectively. Compared to tket and Qiskit, 2QAN has least compilation overhead in terms of the number of inserted SWAP gates, the number of hardware two-qubit gates, and the circuit depth. We summarize the overhead reduction of 2QAN versus tket and Qiskit in Tables 1 and 2 respectively. We define the overhead reduction as the ratio of the overhead of tket or Qiskit and the overhead of 2QAN.

Across all benchmarks and quantum computers, 2QAN inserts on average 3.6x fewer SWAPs than tket and 9.1x fewer SWAPs
We further compare 2QAN with the QAOA compiler in [36–38] and the Paulihedral compiler in [40]. We will only evaluate the compilations results on IBM quantum computers because these two compilers are restricted to the CNOT or CZ gate set. Compared to the QAOA compiler, 2QAN reduces SWAP count, CNOT count overhead, and CNOT depth overhead by on average 2.6x, 4x, and 2.8x, respectively (Figure 9j-9l). The Paulihedral compiler is not open-sourced yet and we directly use the compilation results from [40] as shown in Table 3. The Heisenberg models were compiled by assuming all-to-all connectivity and the QAOA-REG-m were compiled to the IBMQ Manhattan device that has a dodecagon lattice and uses the CNOT as its native two-qubit gate. We generate 10 random graphs for each QAOA problem and present the average compilation results of 2QAN in Table 3. The CNOT count and circuit depth achieved by Paulihedral are on average 1.59x and 1.64x as high as the circuits compiled by 2QAN. Therefore, by employing the permutation-aware optimizations in the routing and unitary synthesizing passes, 2QAN outperforms both application-specific compilers in terms of compilation overhead.

### 5.3 Improving application performance

To study how the compilation overhead reduction impacts application performance, we experimentally implemented the QAOA benchmarks with different numbers of layers on the IBMQ Montreal device. For the multiple-layer QAOA circuits, the 2QAN compiler only performs compilation for the first layer and obtains a circuit structure in \( n \) times of the overhead c1. For odd number layers, it directly uses the compiled circuit structure in c1 and assigns corresponding parameters for each gate. For even number layers, it simply reverses the two-qubit gate order in c1. In contrast, \(|\psi\rangle\) and Qiskit compile a multiple-layer QAOA circuit as a whole. For all four compilers, the compilation overhead of a \( n \)-layer QAOA circuit is approximately \( n \) times of the overhead of a single-layer circuit. In NISQ computers, the implementation of a quantum application that has fewer hardware gates (reducing gate errors) and shorter circuit depth (reducing decoherence

### 5.2 Reducing compilation overhead compared to application-specific compilers

We further compare 2QAN with the QAOA compiler in [36–38] and the Paulihedral compiler in [40]. We will only evaluate the

### Table 1: The average (avg) and maximum (max) compilation overhead reduction when comparing 2QAN with t\(|\psi\rangle\)[30]. For the cases with blank values ‘–’, the 2QAN compiler has negligible overhead.

| Benchmark | Sycamore | Aspen | Montreal |
|-----------|----------|-------|----------|
|           | SWAPs avg | SYCs max | SYC Depth max | SWAPs avg | SWAPs iSWAPs iSWAP Depth avg | SWAPs avg | SYCs avg | CNOTs max | CNOT Depth max |
| NNN Heisenberg | 1.7x | 3.9x | - | 1.2x | 2.7x | 1.1x | 1.8x | 3.2x | 5x | 1.5x | 3.2x | 2.2x | 4.6x | 6x | 12.8x | 2.4x | 5.2x |
| NNN XY | 1.7x | 3.4x | 6.2x | 21.1x | 1.2x | 2.1x | 1.1x | 1.7x | 1.9x | 3.8x | 1.3x | 2.5x | 2.8x | 4.5x | 5.3x | 8.3x | 2.7x | 3.9x |
| NNN Ising | 1.9x | 3.9x | 5.6x | 10.7x | 1.3x | 2.9x | 1.1x | 1.8x | 2x | 3.2x | 1.5x | 3.4x | 2.7x | 4.1x | 4.9x | 7.8x | 2.7x | 4x |
| QAOA-REG-3 | 1.8x | 2.5x | 3.2x | 4.3x | 9.5x | 1.8x | 2.4x | 3x | 4.5x | 2.2x | 3.8x | 2x | 2.4x | 3x | 4.3x | 3x | 4x |

| Benchmark | Sycamore | Aspen | Montreal |
|-----------|----------|-------|----------|
|           | SWAPs avg | SYCs max | SYC Depth max | SWAPs avg | SYCs avg | CNOTs max | CNOT Depth max |
| NNN Heisenberg | 6x | 9.7x | - | 1.8x | 2.6x | 3.3x | 4.3x | 9.3x | 13x | 2.7x | 3.4x | 5.1x | 10.1x | 14x | 27.8x | 3.8x | 7.6x |
| NNN XY | 6.5x | 11.9x | 24.1x | 68.5x | 2.1x | 3.5x | 3.2x | 4.1x | 5.4x | 8.4x | 2.5x | 3.4x | 5.8x | 8.3x | 10.7x | 16.1x | 4x | 6x |
| NNN Ising | 6.7x | 11.5x | 19x | 30.7x | 2.4x | 4.1x | 3.3x | 4.1x | 5.6x | 7.4x | 3x | 3.9x | 5.3x | 8.2x | 9.7x | 15x | 4x | 7.4x |
| QAOA-REG-3 | 4.7x | 6.4x | 8x | 10.6x | 6.9x | 21x | 3.1x | 4x | 5.2x | 6.9x | 3.5x | 5x | 3x | 3.9x | 4.4x | 5.1x | 3.7x | 4.8x |

### Table 2: The average (avg) and maximum (max) compilation overhead reduction when comparing 2QAN with Qiskit [31]. For the cases with blank values ‘–’, the 2QAN compiler almost has negligible overhead.

| Benchmark | Paulihedral | 2QAN |
|-----------|------------|------|
|           | CNOTs Depth | CNOTs Depth |
| Heisenberg-1D (30 qubits) | 87 | 13 |
| Heisenberg-2D (30 qubits) | 216 | 43 |
| Heisenberg-3D (30 qubits) | 395 | 65 |
| QAOA-REG-4 (20 qubits) | 366 | 147 |
| QAOA-REG-8 (20 qubits) | 539 | 246 |
| QAOA-REG-12 (20 qubits) | 678 | 319 |

| Benchmark | Paulihedral | 2QAN |
|-----------|------------|------|
|           | CNOTs Depth | CNOTs Depth |
| Heisenberg-1D (30 qubits) | 87 | 13 |
| Heisenberg-2D (30 qubits) | 216 | 43 |
| Heisenberg-3D (30 qubits) | 395 | 65 |
| QAOA-REG-4 (20 qubits) | 366 | 147 |
| QAOA-REG-8 (20 qubits) | 539 | 246 |
| QAOA-REG-12 (20 qubits) | 678 | 319 |

| Benchmark | Paulihedral | 2QAN |
|-----------|------------|------|
|           | CNOTs Depth | CNOTs Depth |
| Heisenberg-1D (30 qubits) | 87 | 13 |
| Heisenberg-2D (30 qubits) | 216 | 43 |
| Heisenberg-3D (30 qubits) | 395 | 65 |
| QAOA-REG-4 (20 qubits) | 366 | 147 |
| QAOA-REG-8 (20 qubits) | 539 | 246 |
| QAOA-REG-12 (20 qubits) | 678 | 319 |

| Benchmark | Paulihedral | 2QAN |
|-----------|------------|------|
|           | CNOTs Depth | CNOTs Depth |
| Heisenberg-1D (30 qubits) | 87 | 13 |
| Heisenberg-2D (30 qubits) | 216 | 43 |
| Heisenberg-3D (30 qubits) | 395 | 65 |
| QAOA-REG-4 (20 qubits) | 366 | 147 |
| QAOA-REG-8 (20 qubits) | 539 | 246 |
| QAOA-REG-12 (20 qubits) | 678 | 319 |

Table 3: Comparison with Paulihedral [40]. The numbers in brackets show the standard deviation over 10 instances.
errors) should have better performance (higher fidelity). This is experimentally demonstrated in Figure 10, the circuits compiled by 2QAN have the best application performance compared to the results from using t|ket>, Qiskit, and IC-QAOA for all problem sizes and all numbers of QAOA layers.

For example, as noise accumulates in QAOA circuits they will often converge to the value 0 corresponding to a random guess. Here we see that the 3-layer QAOA benchmarks compiled by t|ket> or Qiskit or IC-QAOA already approach zero for problems with 8 qubits. In comparison, 2QAN only comes close to this value for much larger problems (around 20 qubits). Ideally (without hardware noise), the application performance should improve with the number of QAOA layers. However, in practice there is a trade off, with additional layers also increasing the overall error probabilities, and hence decreasing application performance. Figure 10 shows that the QAOA performance for all problem sizes decreases when the number of layers increases for all compilers except the 4-qubit QAOA compiled by 2QAN of which performance improves when increasing the layer number from 1 to 2. This implies that efficient compilation techniques can enhance device capacity and potentially pave the way towards practical applications of quantum computing.

### 5.4 Scalability and runtime

We use the Tabu search algorithm for solving the qubit placement problem. This algorithm is fast for solving small problems, e.g., it takes around 1.6 seconds for the 10-qubit Ising model and 12.2 seconds for the 20-qubit QAOA. However, it becomes slow for larger
problems and takes 330.2 (976.3) seconds for the 40(50)-qubit Heisenberg model. Qubit placement is not our main optimization goal in this work. More efficient algorithms exist for solving QAP problems [73] and other qubit mapping techniques [30, 31] could also be applied in future work. The proposed routing and scheduling heuristics (Algorithm 1 and Algorithm 2) scale at most quadratically with the number of gates in one Trotter step. For example, the compilation time including routing, unitary unifying, and scheduling for the 20(40)-qubit XY model in one Trotter step is around 0.007 (0.015) seconds. Moreover, we only perform the compilation for the first Trotter step and simply use this circuit for odd-number steps and reverse the two-qubit gate order for even-number steps. Such an implementation saves compilation time and is similar to the second-order Trotterization in [17]. This runtime evaluation and scalability analysis give us confidence that the 2QAN compiler will scale up to near-term quantum applications with a considerable number of qubits and gates.

### 6 RELATED WORK

The product formula is the most established approach to simulate the dynamics of quantum systems. Approximation errors arise when there are anti-commuting terms in the Hamiltonian. To achieve a desired precision, the time evolution is divided into many small time steps. Assume a simulation circuit has $r$ Trotter steps and the operator count for implementing each step is $G$. Minimizing the circuit size $Gr$ while maintaining computational accuracy is crucial for practical implementation. Besides the high-order approximation approach [17], many randomization approaches have been proposed to further reduce $Gr$ [42, 43, 74–76]. Afterwards, these high-level circuits still need to be decomposed into native hardware gates. One popular approach for optimizing low-level circuits for Hamiltonian simulation is to first group Pauli terms into commuting sets and then apply simultaneous diagonalization of Pauli exponentials in each set [77, 78] or exploit gate cancellation between consecutive Pauli exponentials [79, 80]. Other works present circuit
Application-specific compilation techniques will be advantageous for NISQ computing. For example, optimized compilers have been developed for variational quantum algorithms [32–39] and quantum simulation with product formulas [40, 84]. The optimization techniques in most specialized compilers (such as IC-QAOA [36–38], Paulihedral [40]) are restricted to CNOT or CZ gates and are not applicable to (or cause higher overhead in) quantum computers with other hardware gates such as the SYC gate in Google Sycamore [21] and the iSWAP gate in Rigetti Aspen [23]. Furthermore, most of them do not exploit the flexible operator permutation property in the product formula approach to quantum simulation (no matter whether these operators commute or not). For instance, the compilers in [33–38] only check for commuting gates in the circuit mapping procedure but many terms in the Hamiltonian may not commute (e.g., the XY model and Heisenberg model) and changing the order of anti-commuting gates will be prohibited. The Paulihedral compiler in [40] considers this flexibility in the term scheduling

Figure 9: Compilation results of the one-layer NNN Heisenberg model, NNN XY, NNN Ising model, and QAOA-REG-3 on the IBMQ Montreal device. The 2QAN compiler has least compilation overhead (# SWAPs, # CNOTs, and circuit depth) compared to t|ket⟩ [30], Qiskit [31], and the QAOA compiler (IC-QAOA) [36–38].
procedure but lacks optimizations in qubit routing and unitary unifying. Our 2QAN compiler can target different architectures and provide efficient compilation for different gate sets.

7 CONCLUSIONS

We have developed an application-specific compiler for 2-local qubit Hamiltonian simulation problems. The 2QAN compiler exploits the flexibility of permuting operators in a Hamiltonian and performs optimizations on the qubit routing, gate synthesis, and gate scheduling passes. Evaluation results show that 2QAN can significantly reduce compilation overhead compared to two general-purpose compilers and two application-specific compilers across three quantum computers with different topologies and gate sets. For some applications, 2QAN even has no gate overhead. Furthermore, experimental results demonstrate that 2QAN can achieve the highest application fidelity in practice on hardware. We believe application-specific compilation techniques will help enhance the performance of quantum applications on NISQ devices and allow them to explore their maximum capacities.

Future work will perform more optimizations and investigate other research directions. First, the objective of 2QAN is to minimize the number of SWAP gates and circuit depth. NISQ computers have inhomogeneities and noise-aware compilation techniques can be used to maximize application fidelity [28, 29, 64, 83]. Moreover, error mitigation techniques can be applied to further reduce errors [85–88]. In the experimental results reported here, we only compiled the first Trotter step in Hamiltonian simulation and simply applied a reverse scheduling for the two-qubit gates in the even-number steps. Prior works prove that randomizing the operator order in each step can reduce the simulation costs [42, 43]. Future work can adapt 2QAN to this randomization and analyze how randomized compiling affects approximation errors and the efficiency of quantum simulation algorithms. In addition, it is worthwhile to investigate how to generalize these compilation techniques for k-local Hamiltonians and other quantum simulation algorithms.

ACKNOWLEDGMENTS

We thank Prakash Murali for valuable feedback on the manuscript. We thank Silas Dilkes for help with the (ket) compiler. We thank César A. Rodríguez Rosario for help with the experimental setup on the IBMQ Montreal device via Strangeworks QC platform. We acknowledge the use of the IBM Q platform for this work. We acknowledge funding from the EPSRC Prosperity Partnership in Quantum Software for Modelling and Simulation (Grant No. EP/S005021/1) and support from the Unitary Fund to open source the 2QAN compiler.

REFERENCES

[1] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factorization. In Proceedings 35th Annual Symposium on Foundations of Computer Science, pages 124–134, 1994.
[2] Joe O’Gorman and Earl T. Campbell. Quantum computation with realistic magic-state factories. Phys. Rev. A, 95:032338, Mar 2017.
[3] Richard P Feynman. Simulating physics with computers. International Journal of Theoretical Physics, 21(6/7), 1982.
[4] D. Jaksch, C. Bruder, C. W. Gardiner, and P. Zoller. Cold bosonic atoms in optical lattices. Phys. Rev. Lett., 81:3108–3111, Oct 1998.
[5] Tim Byrnes and Yoshihisa Yamamoto. Simulating lattice gauge theories on a quantum computer. Phys. Rev. A, 73:022328, Feb 2006.
[6] P. D. Nation, M. P. Blencowe, A. J. Rimberg, and E. Buks. Analogue hawking radiation in a dc-squid array transmission line. Phys. Rev. Lett., 103:087004, Aug 2009.
[7] David Poulin, Matthew B Hastings, Dave Wecker, Nathan Wiebe, Andrew C Doherty, and Matthias Troyer. The Trotter step size required for accurate quantum simulation of quantum chemistry. arXiv preprint arXiv:1406.4920, 2014.
[8] Stephen P Jordan, Keith SM Lee, and John Preskill. Quantum algorithms for quantum field theories. Science, 336(6085):1130–1133, 2012.
[9] Cornelius Hempel, Christine Maier, Jonathan Romero, Jarrod McClean, Thomas Monz, Heng Shen, Petar Jurcevic, Ben P. Lanyon, Peter Love, Ryan Babbush, Alán Aspuru-Guzik, Rainer Blatt, and Christian F. Roos. Quantum chemistry calculations on a trapped-ion quantum simulator. Phys. Rev. X, 8:031022, Jul 2018.
[10] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Andreas Bengtsson, Sergio Boixo, Michael Broughton, Bob B Buckley, David A Buell, Ibran Burkett, Nicholas Budnellen, Yu Chen, Zijun Chen, Yu-An

Figure 10: Experimental results of running QAOA-REG-3 on the IBMQ Montreal device. The y axis shows the application performance, which is measured by the normalized cost function \( \langle C \rangle / C_{\text{min}} \) (larger is better). Each problem size is averaged over 10 different instances (error bars show the standard deviation). The operator parameters of QAOA circuits were chosen at their theoretically optimal values. For the same layer QAOA, 2QAN always achieves better application performance compared to other compilers. The ideal costs increase with the number of QAOA layers (noiseless results in (a-c)). However, in experimental implementation increasing QAOA layers (increasing circuit sizes) increases the probabilities of hardware errors, which may decrease QAOA performance. The application performance of the 4-qubit QAOA compiled by 2QAN improves when the number of layers is increased from 1 to 2 while using the other compilers cannot.
[79] Matthew B Hastings, Dave Wecker, Bela Bauer, and Matthias Troyer. Improving quantum algorithms for quantum chemistry. arXiv preprint arXiv:1403.1539, 2014.
[80] Kaiwen Gui, Teagse Tomesh, Pranav Gokhale, Yunong Shi, Frederic T Chong, Margaret Martonosi, and Martin Suchara. Term grouping and travelling salesperson for digital quantum simulation. arXiv preprint arXiv:2002.05983, 2020.
[81] Alexander Cowtan, Silas Dilkes, Ross Duncan, Will Simmons, and Seyon Sivarajah. Phase gadget synthesis for shallow circuits. arXiv preprint arXiv:1906.01734, 2019.
[82] Arianne Meijer-van de Griend and Ross Duncan. Architecture-aware synthesis of phase polynomials for NISQ devices. arXiv preprint arXiv:2004.06052, 2020.
[83] 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, pages 1001–1016, 2020.
[84] Ian D. Kivlichan, Jarrod McClean, Nathan Wiebe, Craig Gidney, Alán Aspuru-Guzik, Garnet Kin-Lic Chan, and Ryan Babbush. Quantum simulation of electronic structure with linear depth and connectivity. Phys. Rev. Lett., 120:110501, Mar 2018.
[85] Kristan Temme, Sergey Bravyi, and Jay M. Gambetta. Error mitigation for short-depth quantum circuits. Phys. Rev. Lett., 119:180509, Nov 2017.
[86] Ying Li and Simon C. Benjamin. Efficient variational quantum simulator incorporating active error minimization. Phys. Rev. X, 7:021050, Jun 2017.
[87] Sergey Bravyi, Sarah Sheldon, Abhinav Kandala, David C. Mckay, and Jay M. Gambetta. Mitigating measurement errors in multiqubit experiments. Phys. Rev. A, 103:042605, Apr 2021.
[88] Swamit S Tannu and Moinuddin K Qureshi. Mitigating measurement errors in quantum computers by exploiting state-dependent bias. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, pages 279–290, 2019.