ArQTiC: A Full-stack Software Package for Simulating Materials on Quantum Computers

LINDSAY BASSMAN, Lawrence Berkeley National Lab
CONNOR POWERS, University of Southern California
WIBE A. DE JONG, Lawrence Berkeley National Lab

ArQTiC is an open-source, full-stack software package built for the simulations of materials on quantum computers. It currently can simulate materials that can be modeled by any Hamiltonian derived from a generic, one-dimensional, time-dependent Heisenberg Hamiltonian. ArQTiC includes modules for generating quantum programs for real- and imaginary-time evolution, quantum circuit optimization, connection to various quantum backends via the cloud, and post-processing of quantum results. By enabling users to seamlessly design, execute, and analyze materials simulations on quantum computers, ArQTiC opens this field to a broader community of scientists from a wider range of scientific domains.

CCS Concepts: • **Software and its engineering** → *Software creation and management*; • **Applied computing** → *Physical sciences and engineering;*

Additional Key Words and Phrases: Quantum materials, quantum simulations, full-stack software, high-level programming

ACM Reference format:
Lindsay Bassman, Connor Powers, and Wibe A. de Jong. 2022. ArQTiC: A Full-stack Software Package for Simulating Materials on Quantum Computers. *ACM Trans. Quantum Comput.* 3, 3, Article 17 (June 2022), 17 pages.
https://doi.org/10.1145/3511715

1 INTRODUCTION

Quantum computers are an emerging technology, which are poised to revolutionize the computational sciences [3, 13, 36, 43]. Using quantum bits, or qubits, as the units of information processing, quantum computers can capitalize on purely quantum phenomena such as superposition and entanglement to achieve exponential speed-ups and memory reductions compared to their classical counterparts for some applications. Originally conceived of for the simulation of quantum systems [20], quantum computers were later rigorously proven to offer a computational advantage in this area [2, 35, 65]. Indeed, the simulation of quantum materials is seen as one of the most promising applications for quantum computers in the near term [9]. Quantum materials are materials in which quantum effects at the microscopic level give rise to exotic phases or other emergent behaviors at the macroscopic level [29]. An explosion of research into quantum materials over the past decade suggests that such materials will be crucial for the development of next-generation

Authors' addresses: L. Bassman and W. A. de Jong, Lawrence Berkeley National Lab, 1 Cyclotron Road, Berkeley, CA, 94720; emails: {lbassman, wadejong}@lbl.gov; C. Powers, University of Southern California, 3551 Trousdale Parkway, Los Angeles, CA, 90007; email: cdpowers@usc.edu.

2643-6817/2022/06-ART17 $15.00
https://doi.org/10.1145/3511715

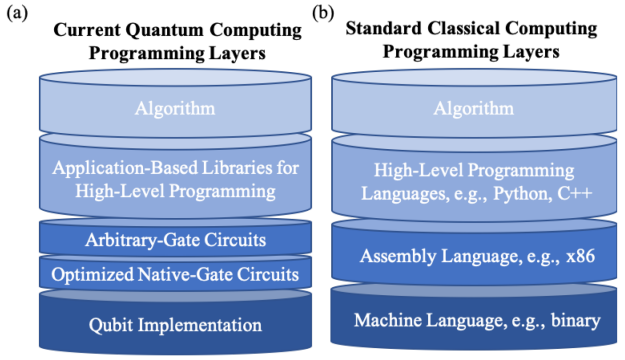

Fig. 1. Programming layers for quantum (a) and classical (b) computers.

technologies [6, 23, 57]. Thus, elucidating the properties and dynamics of quantum materials through simulation is a much-anticipated milestone for near-term quantum computers.

At present, the software available for designing and executing simulations of quantum materials on quantum computers is in a nascent stage. At the most fundamental level, a program run on a quantum computer is a sequence of physical operations performed on the qubits (e.g., electromagnetic pulses). However, much like writing code for classical computers in binary, writing code for quantum computers in terms of pulses can be cumbersome and difficult. To alleviate this burden, layers of abstraction can be sequentially added atop the pulse-level programming layer to facilitate writing quantum programs.

The current hierarchy of abstracted programming layers for quantum computing is depicted in Figure 1(a). At the bottom, the qubit implementation dictates which physical operations can be applied to the qubits. Abstracting one layer above this involves representing the simulation in terms of an optimized native-gate circuit, which is a serial list of quantum logic gates acting on the qubits, with native gates having a one-to-one correspondence with implementable operations on the qubits. Sitting a level above native-gate circuits are arbitrary-gate (i.e., any unitary matrix) circuits. Note that while native-gates are backend-dependent, arbitrary-gates are backend-agnostic, allowing greater flexibility. Abstracting one layer above gates, simulations can be designed using high-level programming models based on application-focused libraries. Finally, at the top resides the algorithm that abstracts away all implementation details and solely describes the general process by which the system of qubits should be manipulated. Figure 1(b) shows analogous programming levels in standard classical computers for comparison.

In the **noisy intermediate-scale quantum (NISQ)** era [46], developing code to run on quantum computers generally involves programming at the gate level, requiring a great deal of domain knowledge in quantum computation. Furthermore, different quantum backends (either the real quantum processors or quantum simulators, which simulate quantum computers with classical computers) use their own hardware-specific language, making it difficult to port simulation code written for one quantum machine to another. Together, this presents a large barrier to entry for researchers from other relevant areas of science, such as materials science and condensed matter physics, which can provide a wealth of new perspectives and insights for materials simulations on quantum computers. To lower this barrier to entry, we have developed an open-source, backend-agnostic, high-level programming library called **Architecture for Time-dependent Circuits (ArQTiC)**, to facilitate research into materials simulations on quantum computers for interested researchers from a broad range of backgrounds.

As a full-stack software package, ArQTiC provides access to each of the programming layers presented in Figure 1(a). At the top layer, ArQTiC implements two major algorithms for material simulation: (i) Hamiltonian evolution based the Trotter decomposition [35, 58] and (ii) imaginary-time evolution via the **quantum imaginary time evolution (QITE)** algorithm [42]. The former is useful for studying the dynamic behavior of materials and their properties, while the latter can be used to find ground- and excited-state energies as well as for generating thermal states, which can be used to compute properties of materials at finite temperatures.

To aid the user in implementing these algorithms, ArQTiC provides a programming library specific to the application of simulating materials on quantum computers. By narrowing its scope, ArQTiC is able to provide high-level functions for the automatic design of quantum circuits, enabling users to remain ignorant of the individual gates that comprise the circuits. Indeed, it broadens access to performing materials simulations on quantum computers to users without extensive knowledge about the construction of evolution operators from the material's Hamiltonian and the subsequent translation of such operators into gates, both of which are automated by ArQTiC.

This high-level programming layer characterizes ArQTiC's main contribution to the landscape of available quantum programming software, which includes Qiskit [1], PyQuil [50], Cirq [26], **Quantum Development Kit (QDK)** [40], Qibo [18], Tequila [31], and XACC [37]. While these general-purpose software platforms offer the ability to design quantum circuits for any application (as opposed ArQTiC's specialization to materials simulations), this comes at the cost of circuits generally needing to be built gate-by-gate, which requires extensive domain knowledge in quantum information and computation. In recent years, some high-level functionality has been added to these platforms to ease programming for specific applications, such as quantum chemistry, finance, and optimization. Other domain-specific packages like ArQTiC, such as OpenFermion [39] and PennyLane [12], provide higher-level programming above the gate level for quantum chemistry and machine learning, respectively. However, it should be emphasized that while simulations for materials and quantum chemistry applications are similar, the Hamiltonians used in each domain differ in their comprising operators. Whereas materials Hamiltonians tend to contain very local operators (usually operators acting only on nearest neighbor qubits), quantum chemistry Hamiltonians can often contain more non-local operators. Furthermore, most high-level programming libraries currently available for quantum chemistry rely on variational algorithms that are hybrid quantum-classical in nature, such as the variational quantum eigensolver [38], which rely on feedback loop of information between a classical and quantum processor. ArQTiC, by contrast, currently focuses on simulations of materials run purely on quantum devices. Therefore, fundamentally different functions are needed to generate circuits for simulations in these different domains. While high-level functions for designing quantum circuits for quantum chemistry applications are largely available, the development of high-level programming for applications in materials simulations had been lacking until the development of ArQTiC.

At the gate-level, ArQTiC provides automatic optimization of the quantum circuit using state-of-the-art circuit compilation tools and can translate circuits into several different languages targeting different quantum backends. Furthermore, ArQTiC can connect with the IBM and Rigetti quantum computers via the cloud to execute the circuits. Finally, ArQTiC provides post-processing and analysis of the raw data returned via the cloud from the quantum backend. This represents the secondary main contribution of ArQTiC to the quantum software landscape. The majority of currently available software platforms for executing quantum circuits will simply return the results from the quantum computer as a list of 0's and 1's. However, deciphering how these 0's and 1's correspond to the observable of interest again can require a great deal of domain knowledge. ArQTiC aids in this post-processing by automatically computing the value of a user-defined

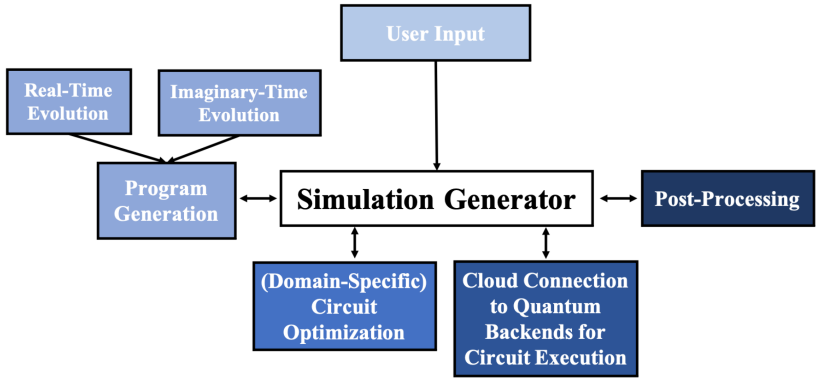

Fig. 2. Blueprint diagram of the ArQTiC modules. The Simulation Generator is the central data structure, containing all necessary information about the simulation. The Simulator Generator interacts with all other modules of ArQTiC, including program generation, circuit optimization, connection to quantum backends, and post-processing.

observable from the raw results of the quantum computer for a variety of dynamic observables, such as energy and magnetization.

While many materials simulations that can be run with ArQTiC can be carried out using most of the existing quantum software platforms (software for implementing real-time Hamiltonian evolution is largely available, though implementations for imaginary-time evolution is still not widely available), they would require a large amount of domain knowledge in quantum computing to perform separately all the individual steps of (i) Hamiltonian construction (possibly term-by-term), (ii) circuit design (possibly gate-by-gate), (iii) optimization, (iv) execution, and, finally, (v) post-processing of raw results. In contrast, ArQTiC fully automates the entire process, solely requiring the user to input the system's Hamiltonian parameters and a few other essential simulation parameters such as timestep and total simulation time. Furthermore, ArQTiC's focused functionality and minimal requirements of the user result in a quick learning time to fluency with the software. While the larger-scale, general-purpose software platforms offer more varied functionality, this comes at the price of a steeper and longer learning curve to get the materials simulations up and running. ArQTiC's full code, as well as an array of python notebooks demonstrating various simulation uses cases (including the illustrative examples given in Section 3), are available on GitHub [11]. By giving users the ability to easily generate, optimize, execute, and post-process quantum circuits simulating materials on quantum computers, ArQTiC, in essence, brings this important class of simulations to the masses.

2 SOFTWARE DESCRIPTION

ArQTiC offers a full-stack solution for the simulation of materials on quantum computers. Once the user defines all simulation parameters, ArQTiC can automatically generate, optimize, and execute quantum circuits, as well as post-process the experimental results with only a few high-level function calls and without any consideration by the user of the quantum gates that comprise the circuit. This enables researchers from a broad range of physical sciences to easily perform their own materials simulations on quantum computers without needing to understand the low-level mechanics and intricacies of quantum computation.

A blueprint of the ArQTiC programming library is shown in Figure 2, depicting the various modules. The central data structure is the Simulation Generator, which contains all the

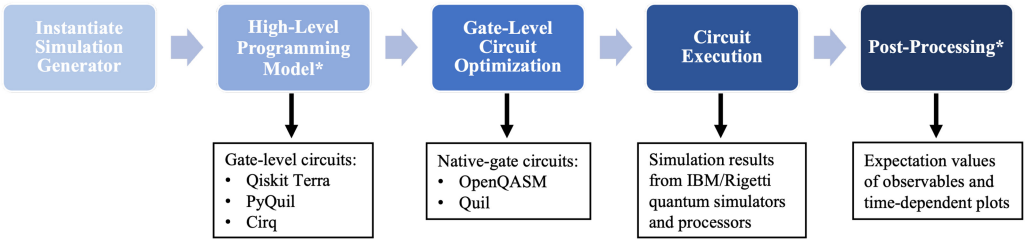

Fig. 3. ArQTiC workflow diagram. The top row of boxes summarizes the flow of information from the instantiation of the Simulation Generator, through the high-level programming layer, to the gate programming layer, down to pulse scheduling for circuit execution on the qubits. The bottom rows show optional outputs at each stage of the workflow. The starred components denote the main contributions of ArQTiC.

information relevant for the simulation. It can be instantiated in two ways: experienced programmers can instantiate the Simulation Generator and assign its attributes within a Python script, while those less familiar with programming may find it easier to pass an input text file containing all parameter definitions as an argument to the Simulation Generator instantiation. The Simulation Generator is what interfaces with all other modules of ArQTiC to perform a simulation, including modules for program generation (ArQTiC's intermediate representation for quantum circuits), circuit optimization, connection to quantum backends via the cloud, and post-processing.

Figure 3 shows a workflow diagram, which illustrates how all the modules of ArQTiC come together to seamlessly provide simulation results from a quantum backend. Note that the boxes in Figures 2 and 3 are color coordinated to demonstrate the correspondence between modules in ArQTiC and subsections of the workflow. Performing a simulation with ArQTiC begins with instantiating the Simulation Generator, whose attributes are provided by the user either in a Python script or via a simple input text file. Examples of instantiation of the Simulation Generator via input text file or Python scripting are provided in the illustrative examples in Section 3.

Currently, ArQTiC can generate simulations for materials that can be modeled with a time-dependent Heisenberg Hamiltonian in one-dimension of the form

$$H(t) = \sum_i J_i^x(t)\sigma_i^x\sigma_{i+1}^x + J_i^y(t)\sigma_i^y\sigma_{i+1}^y + J_i^z(t)\sigma_i^z\sigma_{i+1}^z + h_i^x(t)\sigma_i^x + h_i^y(t)\sigma_i^y + h_i^z(t)\sigma_i^z, \quad (1)$$

where J_i^α is the time-dependent strength of the exchange interaction between nearest neighbor spins i and $i+1$ in the α-direction, h_i^α is the time-dependent strength of the external magnetic field in the α-direction acting on spin i, and σ_i^α is the α-Pauli matrix acting on qubit i. The large amount of freedom in defining the parameters in Equation (1) makes this Hamiltonian quite versatile in its ability to model a wide range of systems including ubiquitous models such as the **transverse field Ising model (TFIM)**, the XY model, and the XXZ chain. The parity of the J_i^α parameters can be chosen to simulate ferromagnetic or antiferromagnetic systems. Setting parameter values to be uniform across all spin pairs versus randomly varied allows one to simulate ordered or disordered systems (such as spin glasses [34]), respectively. Finally, specifying a time-dependent function for the external magnetic field amplitude allows one to simulate laser pulses on material of interest [8, 47].

While Hamiltonian 1 encapsulates a large variety of materials simulations, it is nonetheless narrowed in scope; however, it is precisely this narrowing that enables ArQTiC to achieve such high levels of automation in circuit design, lifting the user out of the tedium of gate-level circuit generation. Furthermore, ArQTiC's current limitation to Hamiltonians that adhere to the form of Hamiltonian 1 allows the user to define their desired system Hamiltonian in an extremely concise

manner: The user need only define the parameters J^x, J^y, J^z, h^x, h^y, and h^z. Each of these parameters can either be defined as (i) a single real number if the parameter is constant across all qubits and timesteps, (ii) an array of real numbers (either defined explicitly or randomly selected from a user-defined range) if the parameter is constant for all timesteps but varies between different qubits, or (iii) a time-dependent function that is the same for qubits, but varies across timesteps. To construct the same Hamiltonian in most other quantum software platforms requires every single individual term to be defined explicitly. As the number of terms will grow with system size, Hamiltonian definition on other quantum software packages can become extremely tedious as system sizes increase.

Once the Simulation Generator object is instantiated, it can be used to generate "programs," which are ArQTiC's native intermediate representation of the quantum circuit that performs the simulation. A program is essentially a backend-agnostic, sequential list of arbitrary gates acting on the qubits representing the system. The advantage of working with this intermediate representation is that the gate-level circuit can be designed once and simply translated into any other formats required by a specific backend. Currently, ArQTiC supports converting its programs into Qiskit [1], PyQuil [50], and Cirq [15]. In particular, this makes it easy to run the same simulation on multiple different quantum backends for comparison.

The creation of a new program relies on algorithms for either real-time or imaginary-time evolution under a given system Hamiltonian, and separate modules exist for each. Generating a program for real-time evolution uses a first-order Trotter decomposition [35, 58] to approximate the time-evolution operator. Since the form of the Hamiltonian is known *a priori* (given by Equation (1)), ArQTiC can automatically decompose the Hamiltonian into groups of mutually commuting terms, which in turn allows the time-evolution operator to be decomposed into a product of operators (via the Trotter decomposition), each of which is straightforward to translate into a series of quantum gates. The user is responsible for providing the timestep size for the Trotter decomposition.

Real-time evolution can be simulated under a time-independent or time-dependent Hamiltonian with ArQTiC. Time-independent Hamiltonians are generally used for simulating quantum quenches, whereby the material system is initialized in the ground state of one Hamiltonian, but is made to evolve under a different Hamiltonian [19, 41, 49, 52]. Quenching from one Hamiltonian to the other can be viewed as instantaneously changing the environment of the material, thereby altering its Hamiltonian. These types of simulations aim to answer fundamental questions about many-body localization, the mechanisms and timescales of thermalization, the changes to or development of collective order (e.g., ferromagnetism, superconductivity, topological order) under a quench, the universality of the dynamics in quenches near critical points, and more [41].

Simulations under time-dependent Hamiltonians [44] can be useful within a few different paradigms. First, they can be used to simulate dynamic processes, such as scattering [16]. Second, they can be used to simulate materials in dynamic environments, such a time-dependent external magnetic field [7, 8]. A third use-case is for finding the ground state of a material through adiabatic quantum evolution [4]. Here, the material is initialized in the ground state of an initial Hamiltonian H_I, which is presumed to be easy to prepare on the quantum computer. The material is then evolved under a parameter-dependent Hamiltonian $H(s) = (1 - s)H_I + sH_P$, which slowly (adiabatically) transforms from the initial Hamiltonian H_I to the problem Hamiltonian H_P as the parameter s is varied from 0 to 1. The adiabatic theorem states that if the system is initialized in the ground state of H_I and s is varied from 0 to 1 slowly enough, then the system will remain in the instantaneous ground state of the Hamiltonian $H(s)$. Thus, at the end of the protocol, the system will be in the ground state of the problem Hamiltonian H_P, which is, in general, difficult to prepare. In this way, simulation with a time-dependent Hamiltonian within ArQTiC can be used to generate the ground state and measure the ground-state energy of various Hamiltonians.

Generating a program for imaginary-time evolution uses the QITE algorithm recently proposed by Motta et al. [42]. The difficulty with evolving a system through imaginary-time on a quantum computer is this operation is not unitary (quantum computers can only perform unitary operators on qubits). The QITE algorithm is able to generate a unitary approximation to this operator by sequentially building up a quantum circuit with a set of sub-circuits. Each sub-circuit carries out a unitary approximation of evolution through an imaginary timestep of size $\Delta\beta$. The sub-circuit for each subsequent timestep $\Delta\beta$ depends on measured expectation values from the total circuit up to the previous timestep. Each sub-circuit further decomposes the total number of qubits into smaller subsets of qubits of a given domain size. For example, a three-qubit system could have one subset with a domain size of three (i.e., the whole system) or two subsets with domain size of two (one subsystem qubits 1 and 2, and one subsystem with qubits 2 and 3). Given a domain size d, the QITE algorithm will build up the sub-circuit for a given imaginary timestep from a set of smaller circuits each acting on a particular subsystem of d qubits. For this algorithm to be feasibly executed, this domain-size must be kept small. The user is responsible for providing the parameters required for the QITE algorithm such as domain-size and imaginary timestep size.

Simulations of imaginary-time evolution are useful for two main applications. The first is for computing the ground-state energy of a material. As a system is evolved in imaginary time, the lowest energy states begin to dominate the system's wavefunction. Therefore, simulating the evolution of the material through imaginary time will cause measurements of the system's energy to result in the ground-state energy with higher and higher probability. The second application for imaginary time evolution is for generating thermal states, which can be used to compute properties of materials at finite temperatures. In particular, two methods for thermal state preparation involving imaginary-time evolution are (i) the **minimally entangled typical thermal states (METTS)** protocol [62] and canonical thermal pure quantum states algorithm [53]. While most quantum software platforms, such as Qiskit, PyQuil, and QDK, offer high-level functions to aid with generating circuits for real-time evolution, very few offer such functions for imaginary-time evolution. To the best of our knowledge, ArQTiC and XACC are unique in offering high-level functions for automatic generation of circuits using the QITE algorithm.

Whether ArQTiC generates programs for real- or imaginary-time evolution depends on a Boolean attribute of the Simulation Generator set by the user. Once a program has been created by the Simulator Generator, it must be translated into an optimized, native-gate quantum circuit targeting the quantum backend selected by the user. In the NISQ era, circuit optimization is equivalent to circuit minimization. This is because currently available quantum computers suffer from high gate-error rates and short qubit decoherence times, causing simulation results to lose fidelity as the quantum circuit gets larger.

ArQTiC offers several choices for circuit optimization. The first option uses the native circuit compiler of the chosen target quantum backend. For example, if the user wishes to run the simulation on the Rigetti quantum computer, then ArQTiC will translate the program into a PyQuil circuit and call PyQuil's native compilation function on the circuit. The second option is to use a popular, state-of-the-art circuit optimizer called tket [48]. The final option is a domain-specific option that can produce optimal constant-depth circuits for real-time evolution. Here, domain-specific refers to the fact that this circuit optimization technique can only be implemented for special subset of Hamiltonians, which are outlined in Reference [10]. For generic Hamiltonians, circuits for real-time evolution are expected to grow at least linearly in size with simulation time. Due to NISQ-era constraints on circuit size, this in turn limits the length of time that can feasibly be simulated on quantum computers. The domain-specific constant-depth circuits, however, enable simulations out to arbitrarily long times. In general, other quantum software platforms only offer

their own native compilers for circuit optimization. ArQTiC enables the user to choose between available state-of-the-art compilers or even compare results among the various compilers.

Once an optimized quantum circuit has been generated, the Simulation Generator can connect via the cloud to either the IBM or Rigetti quantum computers and send the circuits for execution. Upon job completion, results are sent back via the cloud and stored by the Simulator Generator for post-processing. Raw results from the quantum backend are returned in the form of counts of the number of times each qubit was measured to be 0 or 1. Thus, post-processing of the raw data is required to deduce the observable of interest, such as the value of some time-dependent material property. Post-processing of raw results in other quantum software platforms usually requires the user to write an individual script for each desired observable, which necessitates considerable programming fluency and domain knowledge in quantum computation. ArQTiC can automatically perform post-processing of raw results for a number of commonly used observables, as defined by the user. Furthermore, if requested by the user, then ArQTiC can also automatically plot the results and save the figures to file.

3 ILLUSTRATIVE EXAMPLES

3.1 Dynamic Simulation

In this example, we demonstrate how ArQTiC can facilitate the simulation of Anderson localization in a five-spin transverse field XY model when a transverse field is applied randomly across all spins. The Hamiltonian for the system is given by:

$$H = \sum_{i=1}^{n-1} \left(J_x \sigma_i^x \sigma_{i+1}^x + J_y \sigma_i^y \sigma_{i+1}^y \right) + \sum_{i=1}^{n} h_{z_i} \sigma_i^z, \tag{2}$$

where n is the number of spins in the chain, J_x and J_y are the coupling strengths in the x- and y−directions, respectively, σ_i^α is the αth Pauli matrix acting on spin i, and h_{z_i} is the strength of the external magnetic field acting in the transverse direction on spin i and is randomly selected for each spin from a uniform distribution centered around zero. The system is initialized with an excitation in the spin-chain, modeled by flipping the first spin to a spin-down while keeping the remaining spins in the spin-up state. The system is then evolved through time according to Hamiltonian 2. To track the displacement of the excitation through time, the excitation displacement [30] is measured at each timestep, given by the observable

$$N = \sum_{i=1}^{n} (i-1) \frac{1 - \sigma_i^z}{2}. \tag{3}$$

Figure 4 depicts how the Simulation Generator can be instantiated and used to easily run this simulation within ArQTiC. A tutorial for performing this simulation end-to-end can be found on GitHub [11]. To simulate the dynamics of the excitation displacement in this system, a separate circuit must be generated for each timestep. Each circuit must prepare the initial state, simulate the real-time evolution of the system from the initial state up through the given timestep, and prepare to measure the observable of interest. ArQTiC begins by preparing sub-programs for each of these steps, which will be combined into a final program. The sub-program for preparing the initial state of the system can be automatically generated in ArQTiC by setting the **initial_spins** attribute to "1 0 0 0 0," which will flip the first qubit to simulate an initial state with one excitation. Currently, ArQTiC only accepts product states as initial states, so this attribute can only be assigned with a string of 0's and 1's, indicating whether each qubit should be initialized in a spin-up or spin-down state, respectively. Due to ArQTiC's modularity, future extensions can expand the types in initial states that may be defined. Next, ArQTiC generates the sub-program

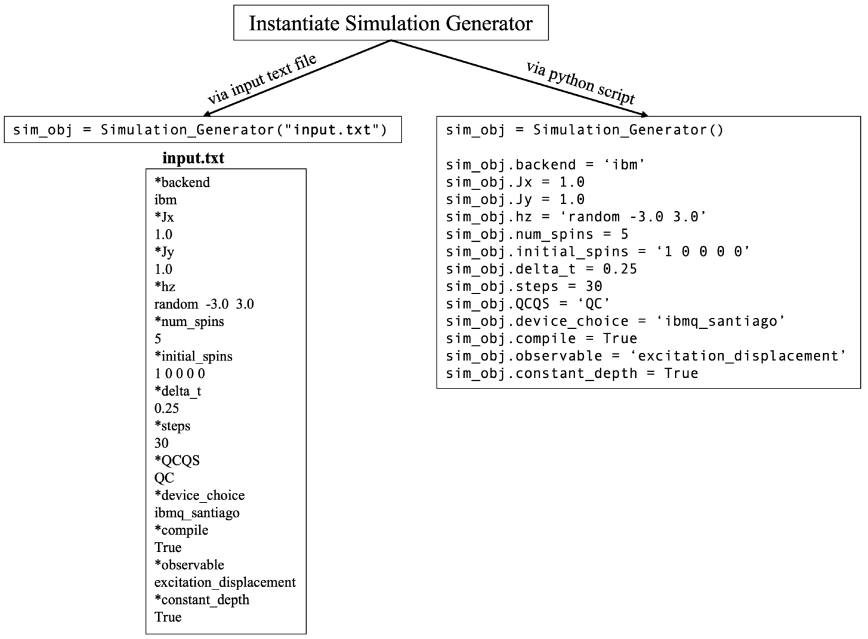

Fig. 4. Instantiation of the Simulation Generator to perform a noiseless dynamics simulation of excitation displacement within an XY spin chain with a randomized Z-direction magnetic field with strengths between −3 and 3 applied to each spin. The first spin is flipped to create the single-spin initial excitation of this example.

for the real-time evolution based on the definition of the parameters of the system Hamiltonian (**num_spins**, **Jx**, **Jy**, and **hz**), the timestep size **delta_t**, and the total simulation time for the given circuit (the first circuit simulates a total time **delta_t**, the second simulates a total time 2∗**delta_t**, etc.). Here, a first-order Trotter decomposition is used to approximate the time-evolution operator by a product of operators that are each straightforward to translate into a set of gates. Note that ArQTiC defaults to preparing programs for real-time evolution; if imaginary-time is desired, a Boolean flag must be set in the Simulation Generator (see the illustrative example in Section 3.2). Finally, ArQTiC generates the sub-program to prepares the system for measurement of the observable of interest, as defined by the **observable** attribute of the Simulation Generator, which in this case is set to "excitation_displacement." The three sub-programs are concatenated to create a complete set of programs for this dynamic simulation. The total number of programs required is given by the attribute **steps**, which defines a total simulation time of **steps** ∗ **delta _t**.

Once ArQTiC has automatically generated a complete set of programs, it will translate these into circuits to run on the desired quantum backend, as defined by the user with the **backend**, **QCQS**, and **device_choice** attributes. Currently, the user can select "ibm" or "rigetti" as the backend of choice and dictate whether the simulation is run on the platform's quantum simulator or real quantum processor by setting **QCQS** to "QS" or "QC," respectively. If "QC" is selected, then the name of the desired processor must be supplied by defining the attribute **device_choice**. Finally, if the user sets the **compile** attribute to "True," then ArQTiC will automatically compile the circuits according to the user-defined method. The default method is to use the native compiler of the target backend. In this case, the Hamiltonian of interest falls into the special subset of Hamiltonians for which a domain-specific compiler [10] can be used, which can generate constant-depth circuits. The user may elect this compilation method by setting the **constant_depth** attribute to "True."

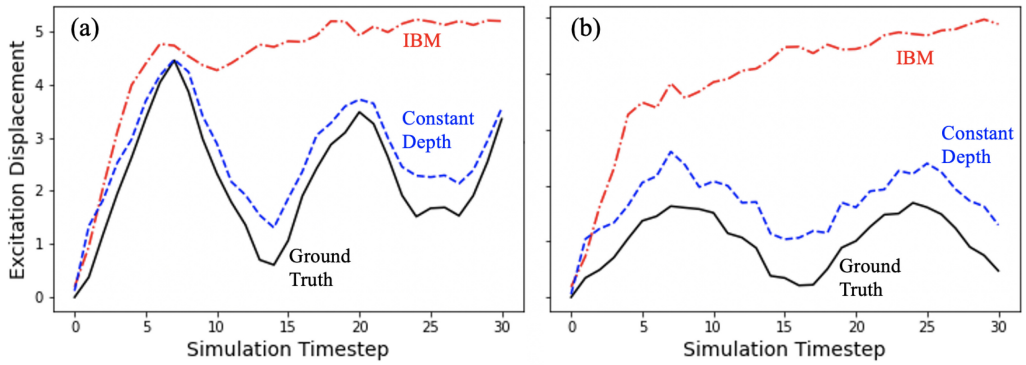

Fig. 5. Results of simulating a five-site transverse field XY spin chain with an initial single-spin excitation with (a) no external magnetic field and (b) randomized transverse external magnetic fields applied to each spin, resulting in localization of the initial excitation. Results from simulations performed on a noiseless quantum simulator are plotted in solid black. Results from simulations run on IBM's "ibmq_santiago" quantum processor are shown for circuits compiled with IBM's native compiler (red dashed-dot lines) and for circuits compiled with the constant-depth compiler (blue dashed lines.)

Results from dynamic simulations on IBM's "ibmq_santiago" device are shown in Figure 5 for a system with zero external magnetic field (Figure 5(a)) and for a system with a randomized external magnetic field drawn from a uniform distribution between −3 and 3 for each spin (Figure 5(b)). We show simulation results for circuits compiled with IBM's native circuit compiler (red dot-dashed lines) versus simulation results for the constant-depth circuits compiled with ArQTiC's domain-specific circuit optimizer (blue dashed lines). For reference, the ground-truth is depicted with the solid black lines.

As seen in Figure 5(a), when no external magnetic field is applied, the excitation is displaced nearly the length of the spin chain before gradually settling towards the center of the chain. However, when the h_{z_i} coefficients are randomly selected from a uniform distribution between −3 and 3, the excitation is confined to oscillating near the beginning of the chain, as seen in Figure 5(b), demonstrating the Anderson localization mechanism [30]. Comparing the results from the IBM-compiled circuits (red dashed-dot lines) to the results from the constant-depth circuits (blue dashed lines) demonstrates the improvement in simulation fidelity achieved with constant-depth circuits. Importantly, while results from the IBM-compiled quantum circuits do not show significantly different behavior for zero versus random external magnetic fields, the results from the constant-depth circuits do. Thus, while the constant-depth results may not be exactly quantitatively accurate, they do demonstrate the trend of Anderson localization, while the IBM-compiled results do not.

Figure 6 plots the total number of gates in the quantum circuit for each timestep for circuits compiled with IBM's native compiler (red dashed-dot line) versus using the domain-specific, constant-depth circuit compilation technique [10] integrated into ArQTiC. While the circuits to obtain similar results to the red dashed-dot lines in Figure 5 can be achieved with other quantum software platforms, much greater efforts and domain-knowledge are required. For circuit generation, many platforms do not offer the ability to construct the Hamiltonian in such a concise manner, rather the user needs to define the Hamiltonian term-by-term, which becomes extremely tedious as the system size grows. After Hamiltonian construction, in the worst-case scenario, a user may be required to generate the time-evolution operator from the Hamiltonian and convert this operator into gates manually, requiring substantial domain knowledge. While a

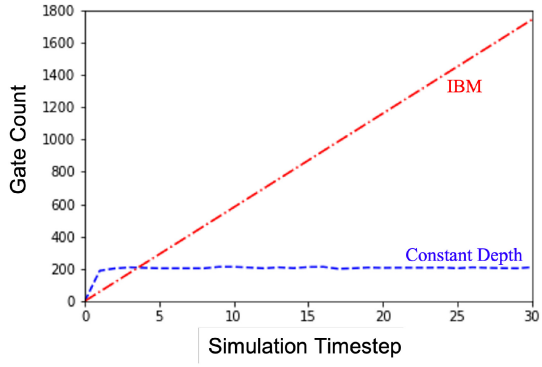

Fig. 6. Comparison of total gate counts in quantum circuits for each timestep when using IBM's native compiler (red dashed-dot line) versus the domain-specific, constant-depth compiler included with ArQTiC (blue dashed line).

handful quantum software platforms now offer some high-level functions for generating circuits for real-time evolution operators, they are usually general-purpose and not optimized in terms of how long they take to generate the output circuit or in terms of the size of the output circuit. Furthermore, a steep learning curve may be required for the user to get these high-level functions running for their particular problem. With ArQTiC, the user only needs to provide a set of realnumbers corresponding to the Hamiltonian parameters, a few other essential simulation parameters, and call a few high-level functions from ArQTiC's programming library. Obtaining the blue dashed lines in Figure 5 is non-trivial with other quantum software packages, as the domain-specific compiler ArQTiC uses to achieve these results is not yet integrated into other packages. Users would need to integrate this special-purpose compiler with other software platforms by hand.

Even when a user succeeds in generating analogous circuits with other quantum software packages, a secondary hurdle is the post-processing of the raw results generated from executing the circuits on a quantum backend. ArQTiC can automatically post-process raw data from the quantum computer or simulator to provide the time-dependence of a user-defined observable, and even automatically generate plots of the dynamics. Other quantum software packages, however, will generally only output the raw data. Users must then write their own post-processing scripts, which can require significant domain knowledge.

Finally, this illustrative example demonstrates how easy it is to compare simulation results across a variety of different parameters with ArQTiC. Comparing results from different compilers, as demonstrated by the red dashed-dot and blue dashed lines in Figure 5, only requires the user to change the attribute defining the compiler choice in the Simulation Generator between different runs of the simulation. It is also easy to compare results across different quantum backends by simply changing the attributes defining the backend (IBM or Rigetti) and defining whether the simulation runs on a real quantum processor or a quantum simulator (with the **QCQS** attribute). Perhaps most significant is the ability to easily re-run the simulations with altered Hamiltonian parameters. The different panels in Figure 5 are an example of the same simulation with two slightly different Hamiltonians. Using ArQTiC, all that needed to be changed between the simulations used to generate the results plotted in Figures 5(a) and 5(b) was adjusting the **hz** attribute of the Simulation Generator. Changing code for simulating these two systems with other quantum software packages is often much less trivial, as a whole new time evolution operator needs to be constructed.

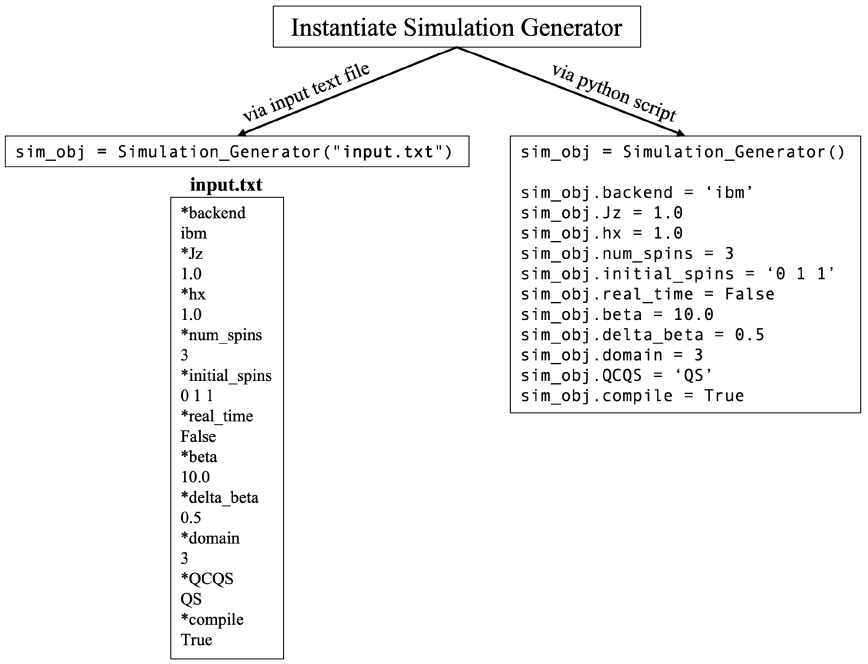

Fig. 7. Instantiation of the Simulation Generator required to perform a ground state energy calculation with QITE. These parameters can be used to generate the green curve shown in Figure 8.

3.2 QITE Simulation

In this example, we demonstrate how to use ArQTiC to find the ground state energy of a material with imaginary-time evolution. For any initial state, evolving the system through an imaginary time $it \equiv \beta$ (by applying the evolution operator $U = e^{-\beta H}$) will eventually drive the system to its ground state. Our system of interest is a three-spin TFIM with open boundary conditions. The Hamiltonian for this system can be written as:

$$H = J_z \sum_{i=1}^{n-1} \sigma_i^z \sigma_{i+1}^z + h_x \sum_{i=1}^{n} \sigma_i^x, \tag{4}$$

where σ_i^α is the αth Pauli operator acting on spin i, J_z gives the strength of the exchange coupling between nearest neighbor spins, h_x gives the strength of the external magnetic field acting uniformly on all the spins, and n gives the number of spins in the system. Figure 7 depicts how the Simulation Generator can be instantiated to run this simulation within ArQTiC. A tutorial for performing this simulation end-to-end can be found on GitHub [11]. To generate circuits for imaginary-time evolution, the **real _time** attribute of the Simulation Generator must be set to "False," as ArQTiC defaults to real-time evolution. The Hamiltonian parameters are assigned with the **Jz**, **hx**, and **num _spins** attributes. The **initial _spins** attribute can be set to any desired product state, as the imaginary-time evolution will drive all systems to their ground states. In the example, we demonstrate this for three different initial spin configurations. The total imaginary-time through which to evolve the system is set by the attribute **beta**, and the imaginary timestep is set by the attribute **delta _beta**. The domain size is set by the attribute **domain**, which can be equal to or smaller than the system size. We choose to run this simulation on the IBM quantum simulator by setting the **backend** attribute to "ibm" and the **QCQS** attribute to "QS." We set the

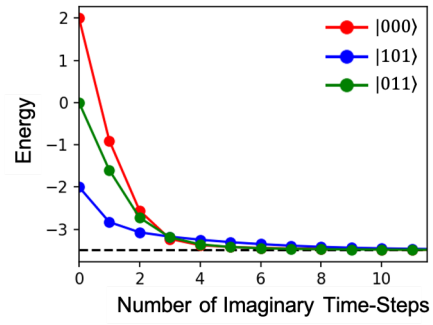

Fig. 8. Convergence to the ground state energy (black dashed line) of the three-spin TFIM spin-chain via QITE. Different colored lines correspond to different initial product states of the systems.

compile attribute to "True," which will automatically use IBM's native circuit compiler, since this was our selected **backend**.

Figure 8 shows how the measured final energy of the system (colored, solid lines) converges to its ground state (black, dashed line) as the number of imaginary timesteps is increased. Different colored lines correspond to starting the system in different initial product states. As shown, all initial states converge to the expected ground state in about eight imaginary timesteps.

ArQTiC is one of a small number of quantum software packages to offer an implementation of the QITE algorithm. Notably, XACC has an implementation, but none of the other major quantum software packages yet offer high-level functions for imaginary-time evolution. Generating the quantum circuits to perform this simulation with most other quantum software platforms (except XACC) would involve a user writing their own circuit implementation, gate-by-gate, for the QITE algorithm, which requires significant domain knowledge. At this point, ArQTiC's (and XACC's) QITE implementation generates circuits that are too large to execute with high-fidelity on current hardware; generating QITE circuits are on the order of hundreds of gates, even for very small systems. However, future releases of ArQTiC might take advantage of techniques for optimizing QITE circuit depths [24, 59]. In the meantime, ArQTiC can at least provide QITE circuits for proof-of-concept simulations run on quantum simulators. Indeed, ArQTiC could be used for the generation of the QITE circuits used for thermal state preparation in several recent works [7, 45, 55].

4 CONCLUSION

We have presented ArQTiC, an open-source, full-stack programming library for performing simulations of materials on quantum computers. By simply providing the material's Hamiltonian parameters and a few other simulation parameters, the user can rely on ArQTiC to seamlessly generate, optimize, and execute materials simulations on various quantum backends, as well as post-process the raw simulation results. The full code, as well as tutorial-style demonstrations of a number of various example simulation use cases, can be found on GitHub [11]. The current release is focused on the real- and imaginary-time evolution of the one-dimensional, time-dependent Heisenberg model, which, by constraining certain parameters, can represent various paradigmatic materials Hamiltonians including the TFIM, the (transverse) XY model, the XXZ chain, and more. This encapsulates most of the materials simulations that have been demonstrated on quantum hardware to date, including simulations of quantum criticality [17], scattering [25], non-equilibrium dynamics [8, 19, 32, 49, 66], and confinement and entanglement dynamics [60, 61], mesonic masses [60], thermal properties [7, 55], and magnon spectra [21].

In its current form, ArQTiC could be used to program most of these simulations, demonstrating its versatility, despite its narrowed scope. This scope, however, could be expanded in future releases to accommodate more systems and methods. Indeed, the modularity of ArQTiC facilitates adding a variety of new functionalities to the code without a massive reorganization of the software. Modules for handling new models, such as the Hubbard model, could be added, extending the kinds of materials that can be simulated. Modules for implementing circuits for real-time evolution with alternative methods, such as higher-order Trotter decompositions or variational approaches [5, 14], could be added. Modules for implementing imaginary-time evolution with alternative methods, such as those using block encoding [22] could be added. Modules generating hybrid quantum-classical execution flows, such as those using novel embedding theory techniques [28, 54], could also be integrated. Additional modules incorporating more advanced circuit compilation engines, such as QFast [64], could be added. It would also be possible to integrate modules to automatically perform various quantum error mitigation techniques, such as zero-noise extrapolation [33, 56]. Finally, for future releases, it would also be useful to capitalize on insights from the design automation community. Indeed, researchers are beginning to look at how to apply knowledge from the rather mature field of design automation for classical computation to quantum computation [27, 51, 63]. By allowing a broader community of scientists to easily perform simulations of materials on quantum computers, ArQTiC paves the way towards accelerated progress in both learning more about quantum materials as well as designing new quantum algorithms for materials simulations.

REFERENCES

[1] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, Francisco Jose Cabrera-Hernández, Jorge Carballo-Franquis, Adrian Chen, Chun-Fu Chen, Jerry M. Chow, Antonio D. Córcoles-Gonzales, Abigail J. Cross, Andrew W. Cross, Juan Cruz-Benito, Chris Culver, Salvador De La Puente González, Enrique De La Torre, Delton Ding, Eugene Dumitrescu, Ivan Duran, Pieter Eendebak, Mark Everitt, Ismael Faro Sertage, Albert Frisch, Andreas Fuhrer, Jay Gambetta, Borja Godoy Gago, Juan Gomez-Mosquera, Donny Greenberg, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Hiroshi Horii, Shaohan Hu, Takashi Imamichi, Toshinari Itoko, Ali Javadi-Abhari, Naoki Kanazawa, Anton Karazeev, Kevin Krsulich, Peng Liu, Yang Luh, Yunho Maeng, Manoel Marques, Francisco Jose Martín-Fernández, Douglas T. McClure, David McKay, Srujan Meesala, Antonio Mezzacapo, Nikolaj Moll, Diego Moreda Rodríguez, Giacomo Nannicini, Paul Nation, Pauline Ollitrault, Lee James O'Riordan, Hanhee Paik, Jesús Pérez, Anna Phan, Marco Pistoia, Viktor Prutyanov, Max Reuter, Julia Rice, Abdón Rodríguez Davila, Raymond Harry Putra Rudy, Mingi Ryu, Ninad Sathaye, Chris Schnabel, Eddie Schoute, Kanav Setia, Yunong Shi, Adenilton Silva, Yukio Siraichi, Seyon Sivarajah, John A. Smolin, Mathias Soeken, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor, Pete Taylour, Kenso Trabing, Matthew Treinish, Wes Turner, Desiree Vogt-Lee, Christophe Vuillot, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, Christopher Wood, Stephen Wood, Stefan Wörner, Ismail Yunus Akhalwaya, and Christa Zoufal. 2019. Qiskit: An open-source framework for quantum computing. Zenodo. DOI: https://doi.org/10.5281/zenodo.2562110

[2] Daniel S. Abrams and Seth Lloyd. 1997. Simulation of many-body Fermi systems on a universal quantum computer. *Phys. Rev. Lett.* 79, 13 (1997), 2586.

[3] Yuri Alexeev, Dave Bacon, Kenneth R. Brown, Robert Calderbank, Lincoln D. Carr, Frederic T. Chong, Brian DeMarco, Dirk Englund, Edward Farhi, Bill Fefferman, et al. 2021. Quantum computer systems for scientific discovery. *PRX Quant.* 2, 1 (2021), 017001.

[4] Rami Barends, Alireza Shabani, Lucas Lamata, Julian Kelly, Antonio Mezzacapo, Urtzi Las Heras, Ryan Babbush, Austin G. Fowler, Brooks Campbell, Yu Chen, et al. 2016. Digitized adiabatic quantum computing with a superconducting circuit. *Nature* 534, 7606 (2016), 222–226.

[5] Stefano Barison, Filippo Vicentini, and Giuseppe Carleo. 2021. An efficient quantum algorithm for the time evolution of parameterized circuits. *Quantum* 5 (2021), 512.

[6] D. N. Basov, R. D. Averitt, and D. Hsieh. 2017. Towards properties on demand in quantum materials. *Nat. Mater.* 16, 11 (2017), 1077–1088.

[7] Lindsay Bassman, Katherine Klymko, Norman M. Tubman, and Wibe A. de Jong. 2021. Computing free energies with fluctuation relations on quantum computers. *arXiv preprint arXiv:2103.09846* (2021).

[8] Lindsay Bassman, Kuang Liu, Aravind Krishnamoorthy, Thomas Linker, Yifan Geng, Daniel Shebib, Shogo Fukushima, Fuyuki Shimojo, Rajiv K. Kalia, Aiichiro Nakano, et al. 2020. Towards simulation of the dynamics of materials on quantum computers. *Phys. Rev. B* 101, 18 (2020), 184305.

[9] Lindsay Bassman, Miroslav Urbanek, Mekena Metcalf, Jonathan Carter, Alexander F. Kemper, and Wibe de Jong. 2021. Simulating quantum materials with digital quantum computers. *arXiv preprint arXiv:2101.08836* (2021).

[10] Lindsay Bassman, Roel Van Beeumen, Ed Younis, Ethan Smith, Costin Iancu, and Wibe A. de Jong. 2021. Constant-depth circuits for dynamic simulations of materials on quantum computers. *arXiv preprint arXiv:2103.07429* (2021).

[11] L. Bassman and C. Powers. 2021. Architecture for Quantum Time-dependent Circuits (ArQTiC). Retrieved from https://github.com/lebassman/ArQTiC.

[12] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, M. Sohaib Alam, Shahnawaz Ahmed, Juan Miguel Arrazola, Carsten Blank, Alain Delgado, Soran Jahangiri, et al. 2018. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968* (2018).

[13] Yudong Cao, Jonathan Romero, Jonathan P. Olson, Matthias Degroote, Peter D. Johnson, Mária Kieferová, Ian D. Kivlichan, Tim Menke, Borja Peropadre, Nicolas P. D. Sawaya, et al. 2019. Quantum chemistry in the age of quantum computing. *Chem. Rev.* 119, 19 (2019), 10856–10915.

[14] Cristina Cirstoiu, Zoe Holmes, Joseph Iosue, Lukasz Cincio, Patrick J. Coles, and Andrew Sornborger. 2020. Variational fast forwarding for quantum simulation beyond the coherence time. *npj Quant. Inf.* 6, 1 (2020), 1–10.

[15] Cirq Developers. 2021. Cirq. arXiv:10.5281/zenodo.4062499.

[16] Weijie Du, James P. Vary, Xingbo Zhao, and Wei Zuo. 2020. Quantum simulation of nuclear inelastic scattering. *arXiv preprint arXiv:2006.01369* (2020).

[17] Maxime Dupont and Joel E. Moore. 2021. Quantum criticality using a superconducting quantum processor. *arXiv preprint arXiv:2109.10909* (2021).

[18] Stavros Efthymiou, Sergi Ramos-Calderer, Carlos Bravo-Prieto, Adrián Pérez-Salinas, Diego García-Martín, Artur Garcia-Saez, José Ignacio Latorre, and Stefano Carrazza. 2020. Qibo: A framework for quantum simulation with hardware acceleration. *arXiv preprint arXiv:2009.01845* (2020).

[19] Benedikt Fauseweh and Jian-Xin Zhu. 2021. Digital quantum simulation of non-equilibrium quantum many-body systems. *Quant. Inf. Process.* 20, 4 (2021), 1–16.

[20] Richard P. Feynman. 1982. Simulating physics with computers. *Int. J. Theor. Phys.* 21, 6 (1982), 467–488. DOI: https://doi.org/10.1007/BF02650179

[21] Akhil Francis, J. K. Freericks, and A. F. Kemper. 2020. Quantum computation of magnon spectra. *Phys. Rev. B* 101, 1 (2020), 014411.

[22] Robert M. Gingrich and Colin P. Williams. 2004. Non-unitary probabilistic quantum computing. In *Proceedings of the Winter International Synposium on Information and Communication Technologies (WISICT'04)*. Trinity College Dublin, 1–6.

[23] Feliciano Giustino, Jin Hong Lee, Felix Trier, Manuel Bibes, Stephen M. Winter, Roser Valentí, Young-Woo Son, Louis Taillefer, Christoph Heil, Adriana I. Figueroa, et al. 2021. The 2021 quantum materials roadmap. *J. Phys.: Mater.* 3, 4 (2021), 042006.

[24] Niladri Gomes, Feng Zhang, Noah F. Berthusen, Cai-Zhuang Wang, Kai-Ming Ho, Peter P. Orth, and Yongxin Yao. 2020. Efficient step-merged quantum imaginary time evolution algorithm for quantum chemistry. *J. Chem. Theor. Computat.* 16, 10 (2020), 6256–6266.

[25] Erik Gustafson, Yannick Meurice, and Judah Unmuth-Yockey. 2019. Quantum simulation of scattering in the quantum Ising model. *Phys. Rev. D* 99, 9 (2019), 094503.

[26] Andrew Hancock, Austin Garcia, Jacob Shedenhelm, Jordan Cowen, and Calista Carey. 2019. Cirq: A Python framework for creating, editing, and invoking quantum circuits. Retrieved from https://github.com/quantumlib/Cirq.

[27] Robin Harper. 2021. Introducing design automation for quantum computing, Alwin Zulehner and Robert Wille. ISBN 978-3-030-41753-6, 2020, Springer International Publishing. 222 Pages, 51 b/w illustrations, 14 illustrations in colour. *Genet. Program. Evolv. Mach.* 22, 3 (2021), 387–389.

[28] Leighton O. Jones, Martín A. Mosquera, George C. Schatz, and Mark A. Ratner. 2020. Embedding methods for quantum chemistry: Applications from materials to life sciences. *J. Amer. Chem. Societ.* 142, 7 (2020), 3281–3295.

[29] B. Keimer and J. E. Moore. 2017. The physics of quantum materials. *Nat. Phys.* 13, 11 (2017), 1045–1055.

[30] E. Kökcü, T. Steckmann, J. K. Freericks, E. F. Dumetriscu, and A. F. Kemper. 2021. Fixed Depth Hamiltonian Simulation via Cartan Decomposition. arXiv:2104.00728 [quant-ph].

[31] Jakob S. Kottmann, Sumner Alperin-Lea, Teresa Tamayo-Mendoza, Alba Cervera-Lierta, Cyrille Lavigne, Tzu-Ching Yen, Vladyslav Verteletskyi, Philipp Schleich, Abhinav Anand, Matthias Degroote, et al. 2021. Tequila: A platform for rapid development of quantum algorithms. *Quant. Sci. Technol.* 6, 2 (2021), 024009.

[32] H. Lamm and S. Lawrence. 2018. Simulation of nonequilibrium dynamics on a quantum computer. *Phys. Rev. Lett.* 121 (2018), 170501. DOI: https://doi.org/10.1103/PhysRevLett.121.170501

[33] Ying Li and Simon C. Benjamin. 2017. Efficient variational quantum simulator incorporating active error minimization. *Phys. Rev. X* 7, 2 (2017), 021050.
[34] Daniel A. Lidar and Ofer Biham. 1997. Simulating Ising spin glasses on a quantum computer. *Phys. Rev. E* 56, 3 (1997), 3661.
[35] Seth Lloyd. 1996. Universal quantum simulators. *Science* 273, 5278 (1996), 1073–1078. DOI: https://doi.org/10.1126/science.273.5278.1073
[36] Sam McArdle, Suguru Endo, Alan Aspuru-Guzik, Simon C. Benjamin, and Xiao Yuan. 2020. Quantum computational chemistry. *Rev. Mod. Phys.* 92, 1 (2020), 015003.
[37] Alexander J. McCaskey, Dmitry I. Lyakh, Eugene F. Dumitrescu, Sarah S. Powers, and Travis S. Humble. 2020. XACC: A system-level software infrastructure for heterogeneous quantum–classical computing. *Quant. Sci. Technol.* 5, 2 (Feb. 2020), 024002. DOI: https://doi.org/10.1088/2058-9565/ab6bf6
[38] Jarrod R. McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. 2016. The theory of variational hybrid quantum-classical algorithms. *New J. Phys.* 18, 2 (2016), 023023.
[39] Jarrod R. McClean, Nicholas C. Rubin, Kevin J. Sung, Ian D. Kivlichan, Xavier Bonet-Monroig, Yudong Cao, Chengyu Dai, E. Schuyler Fried, Craig Gidney, Brendan Gimby, et al. 2020. OpenFermion: The electronic structure package for quantum computers. *Quant. Sci. Technol.* 5, 3 (2020), 034014.
[40] Microsoft. 2021. Quantum Development Kit. Retrieved from https://azure.microsoft.com/en-us/resources/development-kit/quantum-computing.
[41] Aditi Mitra. 2018. Quantum quench dynamics. *Ann. Rev. Condens. Matter Phys.* 9 (2018), 245–259.
[42] Mario Motta, Chong Sun, Adrian T. K. Tan, Matthew J. O'Rourke, Erika Ye, Austin J. Minnich, Fernando G. S. L. Brandão, and Garnet Kin-Lic Chan. 2020. Determining eigenstates and thermal states on a quantum computer using quantum imaginary time evolution. *Nat. Phys.* 16, 2 (2020), 205–210.
[43] Carlos Outeiral, Martin Strahm, Jiye Shi, Garrett M. Morris, Simon C. Benjamin, and Charlotte M. Deane. 2021. The prospects of quantum computing in computational molecular biology. *Wiley Interdisc. Rev.: Computat. Molec. Sci.* 11, 1 (2021), e1481.
[44] David Poulin, Angie Qarry, Rolando Somma, and Frank Verstraete. 2011. Quantum simulation of time-dependent Hamiltonians and the convenient illusion of Hilbert space. *Phys. Rev. Lett.* 106, 17 (2011), 170501.
[45] Connor Powers, Lindsay Bassman, and Wibe A. de Jong. 2021. Exploring finite temperature properties of materials with quantum computers. *arXiv preprint arXiv:2109.01619* (2021).
[46] John Preskill. 2018. Quantum computing in the NISQ era and beyond. *Quantum* 2 (2018), 79. DOI: https://doi.org/10.22331/q-2018-08-06-79
[47] Dongbin Shin, Hannes Hübener, Umberto De Giovannini, Hosub Jin, Angel Rubio, and Noejung Park. 2018. Phonon-driven spin-floquet magneto-valleytronics in MoS 2. *Nat. Commun.* 9, 1 (2018), 638.
[48] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. 2020. t| ket>: A retargetable compiler for NISQ devices. *Quant. Sci. Technol.* 6, 1 (2020), 014003.
[49] Adam Smith, M. S. Kim, Frank Pollmann, and Johannes Knolle. 2019. Simulating quantum many-body dynamics on a current digital quantum computer. *npj Quant. Inf.* 5, 1 (2019), 1–13.
[50] Robert S. Smith, Michael J. Curtis, and William J. Zeng. 2016. A Practical Quantum Instruction Set Architecture. arXiv:1608.03355 [quant-ph].
[51] Mathias Soeken, Thomas Haener, and Martin Roetteler. 2018. Programming quantum computers using design automation. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 137–146.
[52] Alejandro Sopena, Max Hunter Gordon, Germán Sierra, and Esperanza López. 2021. Simulating quench dynamics on a digital quantum computer with data-driven error mitigation. *arXiv preprint arXiv:2103.12680* (2021).
[53] Sho Sugiura and Akira Shimizu. 2013. Canonical thermal pure quantum state. *Phys. Rev. Lett.* 111, 1 (July 2013). DOI: https://doi.org/10.1103/physrevlett.111.010401
[54] Qiming Sun and Garnet Kin-Lic Chan. 2016. Quantum embedding theories. *Acc. Chem. Res.* 8 (2016), 2705–2712.
[55] Shi-Ning Sun, Mario Motta, Ruslan N. Tazhigulov, Adrian T. K. Tan, Garnet Kin-Lic Chan, and Austin J. Minnich. 2021. Quantum computation of finite-temperature static and dynamical properties of spin systems using quantum imaginary time evolution. *PRX Quant.* 2, 1 (2021), 010317.
[56] Kristan Temme, Sergey Bravyi, and Jay M. Gambetta. 2017. Error mitigation for short-depth quantum circuits. *Phys. Rev. Lett.* 119, 18 (2017), 180509.
[57] Yoshinori Tokura, Masashi Kawasaki, and Naoto Nagaosa. 2017. Emergent functions of quantum materials. *Nat. Phys.* 13, 11 (2017), 1056–1068.
[58] Hale F. Trotter. 1959. On the product of semi-groups of operators. *Proc. Amer. Math. Soc.* 10, 4 (1959), 545–551.
[59] Jean-Loup Ville, Alexis Morvan, Akel Hashim, Ravi K. Naik, Marie Lu, Bradley Mitchell, John-Mark Kreikebaum, Kevin P. O'Brien, Joel J. Wallman, Ian Hincks, et al. 2021. Leveraging randomized compiling for the QITE algorithm. *arXiv preprint arXiv:2104.08785* (2021).

[60] Joseph Vovrosh, Kiran E. Khosla, Sean Greenaway, Christopher Self, Myungshik Kim, and Johannes Knolle. 2021. Efficient mitigation of depolarizing errors in quantum simulations. *arXiv e-prints* (2021), arXiv–2101.
[61] Joseph Vovrosh and Johannes Knolle. 2021. Confinement and entanglement dynamics on a digital quantum computer. *Sci. Rep.* 11, 1 (2021), 1–8.
[62] Steven R. White. 2009. Minimally entangled typical quantum states at finite temperature. *Phys. Rev. Lett.* 102, 19 (2009), 190601.
[63] Robert Wille, Stefan Hillmich, and Lukas Burgholzer. 2020. JKQ: JKU tools for quantum computing. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–5.
[64] Ed Younis, Koushik Sen, Katherine Yelick, and Costin Iancu. 2021. QFAST: Conflating search and numerical optimization for scalable quantum circuit synthesis. *arXiv preprint arXiv:2103.07093* (2021).
[65] Christof Zalka. 1998. Simulating quantum systems on a quantum computer. *Proc. R. Soc. A* 454, 1969 (1998), 313–322. DOI: https://doi.org/10.1098/rspa.1998.0162
[66] A. A. Zhukov, S. V. Remizov, W. V. Pogosov, and Yu E. Lozovik. 2018. Algorithmic simulation of far-from-equilibrium dynamics using quantum computer. *Quant. Inf. Process.* 17, 9 (2018), 1–26.

Received May 2021; revised December 2021; accepted January 2022