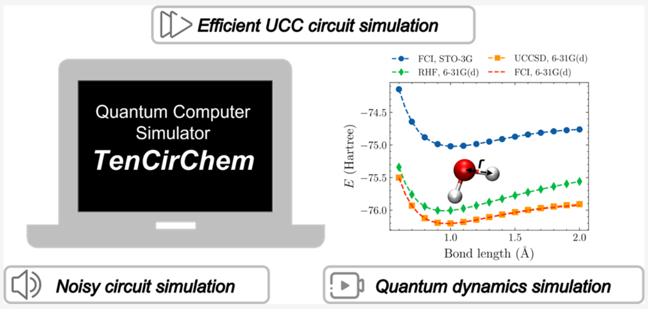
TenCirChem: An Efficient Quantum Computational Chemistry Package for the NISQ Era

Weitang Li,* Jonathan Allcock, Lixue Cheng, Shi-Xin Zhang, Yu-Qin Chen, Jonathan P. Mailoa, Zhigang Shuai, and Shengyu Zhang*

Cite This: J. Chem. Theory Comput. 2023, 19, 3966−3981 Read Online

ACCESS | Metrics & More | Article Recommendations | Supporting Information

ABSTRACT: TenCirChem is an open-source Python library for simulating variational quantum algorithms for quantum computational chemistry. TenCirChem shows high-performance in the simulation of unitary coupled-cluster circuits, using compact representations of quantum states and excitation operators. Additionally, TenCirChem supports noisy circuit simulation and provides algorithms for variational quantum dynamics. TenCirChem's capabilities are demonstrated through various examples, such as the calculation of the potential energy curve of H_2O with a 6-31G(d) basis set using a 34-qubit quantum circuit, the examination of the impact of quantum gate errors on the variational energy of the H_2 molecule, and the exploration of the Marcus inverted region for charge transfer rate based on variational quantum dynamics. Furthermore, TenCirChem is capable of running real quantum hardware experiments, making it a versatile tool for both simulation and experimentation in the field of quantum computational chemistry.



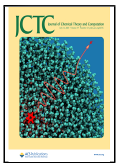
1. INTRODUCTION

Quantum computation leverages quantum effects to store and process data, which could lead to a revolution in computational chemistry.[1−5] With the advent of noisy intermediate scale quantum (NISQ) computing,[6] devices with tens of error-prone qubits are increasingly becoming available for use to researchers and the public.[7−10] Since such devices are not capable of running large-depth structured quantum algorithms, the question of how to best utilize them to solve chemistry problems is of deep scientific and commercial significance. In recent years, following pioneering proposals such as the variational quantum eigensolver (VQE),[11] research has focused on variational algorithms based on parametrized quantum circuits (PQC).[12−14] In the long term, the quantum phase estimation (QPE) algorithm offers a promising route to evaluating the energy of molecular systems, as it may potentially provide an exponential advantage over classical full configuration interaction (FCI) treatment.[15,16] However, compared with VQE, the circuit depths required to run QPE are typically far too deep to run on near-term quantum devices, making VQE the approach of choice in the NISQ era.

VQE uses a parametrized quantum circuit to represent a system's wave function and works by tuning the circuit parameters to minimize the energy variationally. In this setting, the quantum circuit plays the same role as the ansatz in classical quantum chemistry algorithms, and similarly, a variety of different ansätze have been proposed. The first such proposal was the disentangled unitary coupled cluster (UCC) ansatz,[11] which is a variant of the classical coupled cluster method. While disentangled UCC differs from the traditional UCC ansatz developed for classical computers,[17,18] for brevity, in the following, we will omit the word "disentangled". UCC is known to be difficult to implement in classical computers, yet the ansatz can be transformed into a PQC straightforwardly[19] and implemented on a quantum computer. While excitations are commonly restricted to single and double excitations—a case referred to as UCCSD—a number of other variants of UCC have also been proposed.[20−22] Another popular family of ansätze is the hardware-efficient ansatz (HEA),[23] which is designed to be first and foremost easily implementable on a target quantum hardware platform, and includes fewer quantum chemistry heuristics compared to UCC. In addition to ground state properties, such as molecular energy, chemists are also interested in quantum dynamics.[24] Under the PQC framework, the time evolution of the circuit parameters can be determined using time-dependent variational principles.[25,26]

While the quality and availability of quantum computing hardware continue to improve, the majority of real experiments are currently still constrained to small numbers of qubits and

Received: March 19, 2023
Published: June 15, 2023



very shallow circuits.[23,27−31] Furthermore, optimizing circuit parameters on real devices is challenging: gradient-based methods are inefficient, as they rely on computing finite differences or parameter-shifts,[32,33] and both gradient and gradient-free methods are susceptible to noise. These factors mean that for the foreseeable future the classical simulation of variational quantum algorithms will play an important role in studying and analyzing their performance and viability. Although a variety of simulation software is currently available, opportunities remain to significantly improve efficiency, functionality, and flexibility.

Here we introduce TenCirChem, an efficient open-source Python library for simulating variational quantum chemistry algorithms designed to be easy to use as a black box while allowing for a high degree of flexibility and customizability. TenCirChem is powered by TensorCircuit,[34] a recently released quantum circuit simulation package featuring an advanced tensor network contraction simulation engine. A recently developed algorithm to efficiently simulate the UCC ansatz on classical computers[35−37] is also implemented in TenCirChem with notable performance optimizations. We demonstrate how to use TenCirChem by providing concrete code snippets as well as links to online tutorials for features, such as efficient UCC calculation, noisy circuit simulation, and quantum dynamics simulation. In the examples, we include an accurate UCCSD potential energy curve of the water molecule corresponding to 34 qubits, the influence of quantum gate error rate and measurement shots on VQE energy, the spin relaxation dynamics of the spin-boson model, and much more.

The paper is structured as follows. In the rest of the section, we briefly review variational quantum algorithms for quantum chemistry and existing simulation packages. In section 2, we present the overall architecture and workflow of TenCirChem with an emphasis on the related theoretical background. Sections 3, 4, and 5 illustrate the features contained in the TenCirChem modules through code snippets and detailed simulation examples.

1.1. Variational Quantum Algorithms for Quantum Chemistry.

Variational quantum algorithms for quantum chemistry have been an active area of research since the original VQE proposal.[11] For both ground state properties and quantum dynamics, while details vary, proposed protocols typically adhere to the following workflow:

Input. The input to the problem is the system Hamiltonian in a second-quantized form. For electronic structure problems, this can be expressed as

$$H = \sum_{pq} h_{pq} a_p^\dagger a_q + \frac{1}{2}\sum_{pqrs} h_{pqrs} a_p^\dagger a_q^\dagger a_r a_s + E_{\text{nuc}} \tag{1}$$

where h_{pq} and $h_{pqrs} = [ps|qr]$ are one-electron and two-electron integrals, and $a_p^\dagger$ and a_p are fermionic creation and annihilation operators, respectively, acting on the p-th spin-orbital. E_{nuc} is the nuclear repulsion energy. For quantum dynamics simulations, the Hamiltonian varies from system to system, and may contain only electronic terms[38,39] or both electronic and vibrational terms.[40,41] The latter is termed vibronic-coupling Hamiltonians.

Conversion to Qubits. The Hamiltonian, either electronic or vibronic-coupling, is converted into an N-qubit Hamiltonian of the form

$$H = \sum_j^M \alpha_j P_j \tag{2}$$

where the α_j are real coefficients, M is the total number of terms, and the P_j are Pauli strings of the form $P_j = \sigma_{i_1} \otimes \sigma_{i_2} \otimes ... \sigma_{i_N}$ where the σ_{i_k} are single qubit Pauli operators or the identity. For electronic Hamiltonians, the Jordan-Wigner transformation,[42] parity transformation[43,44] and Bravyi-Kitaev transformation[44] are popular choices to perform this conversion. The Jordan-Wigner and Bravyi-Kitaev transformations require one qubit per spin-orbital, while the parity transformation can save 2 qubits, relative to these other methods, due to electron number conservation in each spin sector. For vibronic-coupling Hamiltonians, the nuclear states also need to be encoded into qubits via unary or binary encodings.[45]

Choice of Ansatz. A parametrized ansatz state $|\psi(\theta)\rangle = U(\theta)|\psi_0\rangle$ is selected, where $U(\theta)$ is the unitary corresponding to a quantum circuit characterized by a vector θ of tunable parameters and $|\psi_0\rangle$ is a chosen reference state or initial state. The core role of quantum computers or quantum processing units (QPUs) in variational algorithms is to use quantum circuits to execute $U(\theta)$ and prepare $|\psi(\theta)\rangle$. Thus, the ability of the variational ansatz in expressing the system wave function is a key factor impacting the accuracy of variational quantum algorithms for quantum chemistry. For electronic Hamiltonians, the proposed ansätze generally fall into unitary coupled cluster (UCC) based approaches,[11,21,46−48] and hardware-efficient ansätze.[23,27−31] For quantum dynamics simulation, the appropriate family of ansatz varies from system to system. The variational Hamiltonian ansatz[49,50] and adaptive ansatz[51] automate the task of constructing problem-specific circuit ansätze.

Updating Parameters. In the case of VQE, because the true ground state energy E_0 of the system gives a lower bound on the energy expectation value $\langle H\rangle_\theta := \langle\psi(\theta)|H|\psi(\theta)\rangle$, $\langle H\rangle_\theta$ is minimized with respect to the parameters θ, which in turn gives an upper bound on E_0, i.e.,

$$E_0 \le \min_\theta \langle H\rangle_\theta$$

The quality of this bound depends on the chosen ansatz. In general, the function is nonconvex and one can only expect to find a local minimum, which depends on the parameter initializations and the optimization protocol used. For dynamics problems, the parameters are propagated according to time-dependent variational principles, of which McLachlan's variational principle is among the most popular choices for quantum computing.[26,52] McLachlan's variational principle minimizes the distance between the ideal time derivative, $-iH|\psi(\theta)\rangle$, and actual time derivative $\frac{\partial}{\partial t}|\psi(\theta)\rangle$

$$\min_{\dot\theta} \left\| \left(\frac{\partial}{\partial t} + iH\right)|\psi(\theta)\rangle \right\| \tag{3}$$

At each time step, $\dot\theta = \partial\theta/\partial t$ is solved, and then θ is evolved numerically according to $\dot\theta$. The equation of motion for θ_k, the kth element in vector θ, is

$$\sum_k M_{jk}\dot\theta_k = V_j \tag{4}$$

where

$$M_{jk} = \text{Re}\left\{\frac{\partial\langle\psi|}{\partial\theta_j}\frac{\partial|\psi\rangle}{\partial\theta_k}\right\}, \qquad V_j = \text{Im}\left\{\frac{\partial\langle\psi|}{\partial\theta_j}\text{H}|\psi\rangle\right\} \tag{5}$$

Both M_{jk} and V_j can be evaluated on a quantum computer with one additional ancilla qubit via the Hadamard test.[25] $\dot{\theta}_k$ can then be deduced using a standard linear equation solver on a classical computer. After that, θ_k is propagated in time with time step τ either by the simple forward Euler method $\theta_k(t+\tau) = \theta_k(t) + \tau\dot{\theta}_k(t)$ or more sophisticated initial value problem solvers, such as Runge−Kutta solvers.

1.2. Existing Packages: Challenges and Opportunities.

A number of useful software packages are currently available that facilitate the research, design, and validation of quantum computational chemistry algorithms. A variety of general quantum computing packages, such as QisKit,[53] PennyLane,[54] and MindQuantum,[55] have modules or subpackages dedicated to chemistry applications, and there are also stand-alone quantum simulation packages that are mostly or completely designed for quantum computational chemistry, such as Tequilla,[56] Q2 Chemistry,[57] and QForte.[58]

The efficiency and scalability of the packages depend crucially on the underlying circuit simulation algorithm. The most commonly encountered quantum circuit simulator is a statevector simulator, which is implemented in all simulation packages mentioned above. By using a statevector simulator, all (exponentially many with respect to the number of qubits) amplitudes of the wave function are computed and stored. For such simulators, the largest simulated UCC circuit to date contains fewer than 30 qubits.[59] In contrast, matrix product state (MPS) simulators are able to tackle larger circuits, with around 100 qubits[60,61] possible. Among the packages mentioned above, QisKit, PennyLane, and Q2 Chemistry have implemented the MPS simulator. However, MPS simulation involves approximations, which could be problematic for the validation of quantum algorithms. Another challenge for software development is noisy circuit simulation and interfacing with hardware. These features can help researchers better understand the behavior of quantum algorithms in real-world conditions, facilitating the development of noise-resilient quantum algorithms and better error mitigation techniques,[47,50,62] which are essential for building practical quantum computers. QisKit is currently the only package capable of noisy circuit simulation as well asinterfacing with quantum hardware. Furthermore, to the best of the authors' knowledge, quantum dynamics algorithms of molecular systems are not implemented in any packages currently available.

TensorCircuit is a recent open-source quantum circuit simulator based on tensor network contraction which, in certain cases, enables the simulation of up to 600 qubits without any approximation.[34] Differences between tensor network contraction simulators and MPS simulators are described in section 2.4. TensorCircuit is built upon industry-leading machine learning frameworks, such as JAX[63] and employs automatic differentiation (AD), just-in-time (JIT) compilation, and vectorized parallelism to accelerate simulation. TensorCircuit also possesses a fully fledged noise model that enables efficient noisy circuit simulation. With its TensorCircuit backend, TenCirChem provides a competitive option for developing and analyzing quantum computational chemistry algorithms. The role of TensorCircuit in TenCirChem is described in more detail in section 2.1.

2. TENCIRCHEM OVERVIEW AND THEORETICAL BACKGROUNDS

In this section, we provide a high-level picture of TenCirChem by introducing its architecture and typical workflow. In addition, we present the theoretical background relevant to the code examples found in sections 3, 4, and 5.

2.1. Architecture and Workflow.

Architecture. TenCirChem consists of two primary modules (Figure 1),

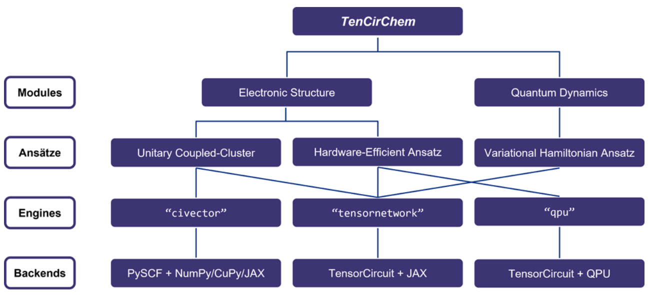


Figure 1. Architecture of TenCirChem. The electronic structure and quantum dynamics modules are described in depth in sections 3, 4, and 5. See section 2.3 for more information about the supported ansätze. For more details on engines and backends, see section 2.4.

corresponding to the (i) electronic structure and (ii) quantum dynamics. Both modules contain a set of prebuilt ansätze, and user-specified custom ansätze are also supported. Due to the popularity and unique mathematical structure of the UCC ansatz, an efficient simulation engine "`civector`" is implemented specifically for its efficient simulation. The details of the engine are included in the Supporting Information. In short, the engine exploits particle number conservation to store the system wave function in *configuration interaction vector* (`civector`) form and uses *UCC factor expansion* to accelerate simulation. This engine relies heavily on the FCI module of PySCF[64] for the construction and manipulation of the configuration interaction vector. In addition, the gradient is evaluated by autodifferentiation with reversible computing, which is more memory-efficient than the traditional reverse-mode autodifferentiation implemented in standard machine-learning packages.[65] The "`tensornetwork`" engine uses the TensorCircuit tensor network contraction kernels to perform circuit simulation. A variant of the `tensornetwork` engine is the `tensornetwork-noise` engine, which further includes circuit gate noise and measurement uncertainty in the simulation using the relevant utilities implemented in TensorCircuit. The "`qpu`" engine delegates circuit execution to real QPUs through TensorCircuit.

Workflow. As mentioned in the introduction, the starting point for variational quantum chemistry algorithms is normally taken to be the Hamiltonian in the second-quantized form. For electronic structure problems, this means specifying the one- and two-electron integrals h_{pq} and h_{pqrs} in eq 1. In TenCirChem this can be done either directly, by inputting these numbers explicitly, or indirectly, by specifying the 3D coordinates, atom types, and basis sets of each of the atoms in a molecule and having PySCF compute the integrals automatically. The Hamiltonian in quantum dynamics is defined using the Renormalizer[66] package by specifying each term in the Hamiltonian.

Next, the Jordan-Wigner, parity, or Bravyi-Kitaev conversion from electronic Hamiltonian (eq 1) to qubit Hamiltonian (eq 2)

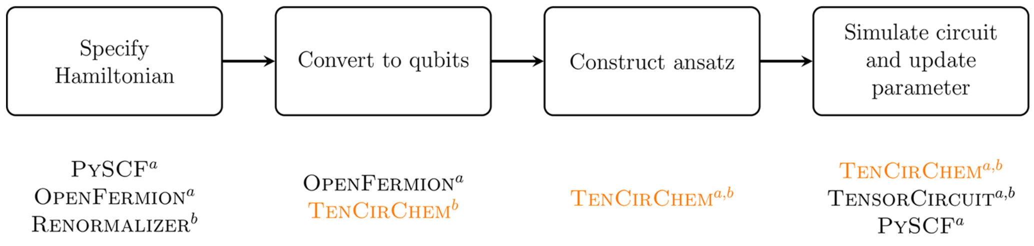


Figure 2. Typical workflow of variational quantum algorithms and the underlying packages used for each task in TenCirChem. [a]For electronic structure. [b]For quantum dynamics.

is performed by OpenFermion.[67] The encoding of nuclear states to qubits and the construction of the ansätze are performed within the TenCirChem package. The workflow and the underlying packages are summarized in Figure 2, in accordance with section 1.1, and will be elaborated on in the following sections. Standard infrastructure packages, such as NumPy and SciPy, are omitted for brevity. The specific tasks of each underlying package are summarized below:

- PySCF: specifying molecules; calculating integrals; configuration interaction vector construction and manipulation
- OpenFermion: specifying operators in fermion operator or qubit operator format; conversion from fermion operator to qubit operator
- Renormalizer: specifying general quantum dynamics Hamiltonian; conversion from symbolic operators to numeric operators
- TensorCircuit: constructing custom PQC as ansatz; quantum circuit simulation via tensor network contraction; interfacing with QPU

2.2. Conversion to Qubits. Quantum simulation of physical systems requires converting fermions (for electronic structure calculations) and bosons (for encoding nuclear wave functions and quantum dynamics) to qubits.

2.2.1. Electronic Structure. Fermionic creation and annihilation operators obey the anticommutation properties

$$\{a_i, a_j^\dagger\} = a_i a_j^\dagger + a_j^\dagger a_i = \delta_{ij}$$
$$\{a_i^\dagger, a_j^\dagger\} = \{a_i, a_j\} = 0 \tag{6}$$

while the qubit creation operator $c^\dagger = \frac{1}{2}(X - iY)$ and annihilation operator $c = \frac{1}{2}(X + iY)$ obey the commutation properties

$$\{c_i, c_j^\dagger\} = \delta_{ij}, \quad [c_i^\dagger, c_j^\dagger] = [c_i, c_j] = 0 \tag{7}$$

To comply with the required fermionic anticommutation properties, the Jordan-Wigner transformation[42] can be used to map fermionic ladder operators into products of Pauli Z and qubit ladder operators

$$a_j = \bigotimes_{l=0}^{j-1} Z_l \otimes c_j \tag{8}$$

Alternatively, the parity transformation[43,44] achieves the same goal by letting each qubit record the parity of all preceding orbitals and itself. The mapping is then

$$a_j = (c_j \otimes |0\rangle\langle 0|_{j-1} - c_j^\dagger \otimes |1\rangle\langle 1|_{j-1}) \otimes \bigotimes_{l=j+1}^{N-1} X_l \tag{9}$$

where N is the total number of qubits. If the ansatz conserves the total particle number, then the last qubit will not change during computation and can be removed without a loss of accuracy. If S_z is also conserved, another qubit can be saved. Thus, the parity transformation uses 2 fewer qubits than the Jordan-Wigner transformation. In TenCirChem, both transformations are carried out by calling OpenFermion.

2.2.2. Quantum Dynamics. One of the key differences between quantum dynamics simulation and electronic structure calculations is that quantum dynamics requires incorporation of nuclear information in time evolution. Similar to the case of electrons, a proper basis set must first be chosen to represent the nuclear wave function and operators in second-quantized form.[40,62,68] One popular choice is the harmonic eigenbasis, in which the nuclear position operator x and momentum operator p are expressed as

$$x = \sqrt{\frac{1}{2m\omega}}(b^\dagger + b),$$
$$p = i\sqrt{\frac{m\omega}{2}}(b^\dagger - b) \tag{10}$$

where m is the oscillator mass, ω is the oscillator frequency, and $b^\dagger$ (b) are bosonic creation (annihilation) operators, respectively. The infinite boson levels are then truncated to the lowest N levels for further encoding. Other choices for the nuclear basis set include the real-space grid basis and a variety of discrete value representations.[69] In any case, each nuclear degree of freedom is represented by a finite number of basis states $|\psi\rangle = \sum_i^N c_i |i\rangle$.

In quantum simulation, each bosonic basis vector $|i\rangle$ must be encoded into qubits, so that the system wave function can be described by a quantum circuit and operators can be represented by Pauli strings. The principle for the encoding is equivalent to that of encoding integers into bit strings, and there are two common encoding strategies, based on unary and binary encodings, requiring $O(N)$ and $O(\log N)$ qubits, respectively.[45] Binary encoding with Gray codes is a variant of the standard binary encoding that involves changing only one bit between consecutive integers, which can be used to facilitate the efficient encoding of operators $b^\dagger$ and b. TenCirChem supports all of the encoding schemes and, by default, uses binary encoding with Gray codes because of its compact representation. We note that an efficient variational encoding scheme requiring only $O(1)$ qubits was proposed recently.[70]

2.3. Supported Ansätze. TenCirChem supports three classes of variational ansätze: (i) Unitary Coupled Cluster (UCC) (of which UCCSD, k-UpCCGSD, and paired UCCD (pUCCD) are variants), (ii) Hardware-Efficient, and (iii) Variational Hamiltonian.

2.3.1. Unitary Coupled Cluster Ansatz. UCC ansätze have the form

$$e^{T(\theta)}|\phi\rangle \tag{11}$$

where

$$T(\theta) = \sum_{pq} \theta_{pq}(a_p^\dagger a_q - \text{h.c.}) + \sum_{pqrs} \theta_{pqrs}(a_p^\dagger a_q^\dagger a_r a_s - \text{h.c.})... \tag{12}$$

θ denotes the vector of tunable parameters θ_{pq}, θ_{pqrs}, ..., and $|\phi\rangle = |\text{HF}\rangle$ is the Hartree−Fock state. In general, excitations of any order are permitted, although it is common to only consider single and double excitations, in which case the ansatz is referred to as UCCSD. The indices are usually confined to excitations from occupied orbitals to virtual (unoccupied) orbitals. "Generalized" excitations do not have to satisfy this requirement. The generalized counterpart of UCCSD is UCCGSD.[20]

As the terms in eq 12 do not necessarily commute, the exact implementation of eq 11 on a gate model quantum computer is difficult. In fact, to the best of our knowledge, the ansatz defined by eq 11 and eq 12 is never used in any practical VQE algorithm. Instead, one considers a Trotterized approximation of the ansatz, also referred to as disentangled UCC.[71] Although disentangled UCC is traditionally viewed as an approximation to the original UCC, it can in fact exactly represent general fermionic states given appropriate operator ordering and should probably be considered as an alternative ansatz to the original UCC.[71] In general, the disentangled UCC ansatz can be written as

$$|\Psi(\theta)\rangle_{\text{UCC}} := \prod_{k=N_{\text{ex}}}^{1} e^{\theta_k G_k}|\phi\rangle \tag{13}$$

where each G_k is the anti-Hermitian excitation operator for each term in eq 12, and N_{ex} is the total number of excitations. In the following, the terms $e^{\theta_k G_k}$ are referred to as UCC factors. The UCC ansätze implemented in TenCirChem can all be viewed as special cases, as shown in eq 13. The specific formulations of UCCSD, k-UpCCGSD, and pUCCD are reviewed in the Supporting Information.

2.3.2. Hardware-Efficient Ansatz. Hardware-efficient ansätze are designed to be easily implementable on NISQ devices but do not necessarily preserve symmetries in molecular systems. Finding the global minimum for such circuits can be challenging due to the vanishing gradient phenomenon and the presence of local minima.[72,73] However, because the required circuit depths can be much smaller than for UCC, hardware-efficient ansätze are widely used in quantum computational chemistry experiments performed on real quantum processors.[23,27−31] TenCirChem implements one of the most popular hardware-efficient ansätze, the R_y ansatz,[30,73−75] whose circuit consists of interleaved layers of R_y and CNOT gates

$$|\Psi(\theta)\rangle_{R_y} := \prod_{l=k}^{1} [L_{R_y}^{(l)}(\theta) L_{\text{CNOT}}^{(l)}] L_{R_y}^{(0)}(\theta)|\phi\rangle \tag{14}$$

where k is the total number of layers, and the layers are defined as

$$L_{\text{CNOT}}^{(l)} = \prod_{j=N-1}^{1} \text{CNOT}_{j,j+1},$$
$$L_{R_y}^{(l)}(\theta) = \prod_{j=N}^{1} \text{RY}_j(\theta_{lj}) \tag{15}$$

The gate subscripts refer to the qubit index on which the gate acts, and N is the total number of qubits. An example of the ansatz is depicted in Figure 6. One reason for using the R_y gate is that it enforces the wave function coefficients to be real, which is a desired property for electronic structure problems.

As a huge number of possible hardware-efficient ansätze can be defined, instead of providing templates for all of them, TenCirChem allows the hardware-efficient ansätze from the Qiskit library to be directly imported and other customized ansätze can be defined by the user using TensorCircuit (see section 4.1 for examples).

2.3.3. Variational Hamiltonian Ansatz. For quantum dynamics simulations, TenCirChem provides utilities to construct the variational Hamiltonian ansatz[49,50] based on user-specified Hamiltonians. This type of ansatz depends on the Pauli strings P_j contained in the Hamiltonian and thus encodes Hamiltonian information. Suppose the Hamiltonian is composed of M Pauli strings P_j, as defined in eq 2. The corresponding variational Hamiltonian ansatz has the form

$$|\Psi\rangle = \prod_{l}^{k} \prod_{j}^{M} e^{-i\theta_{lj}P_j}|\phi\rangle \tag{16}$$

where $|\phi\rangle$ is the initial state of the system, θ_{kj} is the circuit parameter, and k is a parameter defining the number of layers in the ansatz. This heuristic construction is based on the fact that, in the short time limit, the system wave function $e^{-iHt}|\phi\rangle$ can be exactly described by eq 16.

2.4. Engines and Backends. We use the term "engine" to refer to the methods, such as `tensornetwork` and `civector`, used to simulate the quantum circuits in TenCirChem. The term "backend" refers to the underlying numerical package used to perform the simulation, such as NumPy and JAX. The most common engine in TenCirChem is the `tensornetwork` engine powered by TensorCircuit, which views the quantum circuit as a network of low-rank tensors and performs circuit simulation via tensor network contraction.[76] This approach can offer significant advantages over full statevector simulators which may encounter memory bottlenecks. As a result, tensor network contraction is the most frequently used method for performing large-scale quantum circuit simulations, such as the simulation of random circuits used in quantum supremacy experiments.[77,78] When contracting a tensor network, it is crucial to find an efficient contraction path. The default contraction path finder in TenCirChem is based on a greedy algorithm, which has minimal overhead but may not be satisfactory for large quantum circuits. The customization of the contraction paths is possible via TensorCircuit.

We note that the tensor network contraction simulator should not be confused with an MPS simulator,[60] although the two do share a number of things in common. In particular, the tensor network contraction method does not assume the circuit wave function to adopt MPS form and does not perform any truncations to the wave function.

The tensor network simulator can also be extended to the simulation of noisy circuits using density matrix ρ. The simulation of noisy quantum circuits is thus twice as memory intensive as pure state simulation. We stress that the engine does not perform any truncation to the target density matrix, which is different from a number of preceding works.[79,80] In the presence of noise, unfortunately, symmetries encoded in the ansatz are broken and cannot be exploited. In TensorCircuit, noise channels $\mathcal{E}$ are defined by their corresponding Kraus operators K_i, i.e.,

$$\mathcal{E}(\rho) = \sum_i K_i \rho K_i^\dagger \tag{17}$$

The action of $\{K_i\}$ on the circuit is achieved by transforming them into a superoperator. More specifically, if the channel acts on N qubits, it is converted to a matrix with size $4^N \times 4^N$, as a node in the density matrix tensor network. We note that TensorCircuit supports noisy circuit simulation via Monte Carlo sampling in addition to using density matrices, although this algorithm is not currently implemented in TenCirChem.

Of the various backends supported by TenCirChem, the JAX backend is perhaps the most versatile and can generally be used in most cases. This backend also supports GPU and TPU calculations once JAX is properly configured. If the `civector` engine is used, the NumPy and CuPy backends are preferred for CPU and GPU calculations, respectively. The reason for this is that the code in the `civector` engine is highly optimized, and just-in-time (JIT) compilation overheads become a bottleneck.

2.5. Installing and Contributing to TenCirChem. TenCirChem is written in Python and can be installed via `pip install tencirchem` using the command line. The source code is hosted on the GitHub repository https://github.com/tencent-quantum-lab/TenCirChem. We welcome all members of the quantum chemistry community to contribute to the continued development of TenCirChem.

3. THE ELECTRONIC STRUCTURE MODULE I. UNITARY-COUPLED CLUSTER ANSATZ

3.1. Basic Usage. The basic usage of TenCirChem can be illustrated in five lines of codes, which we demonstrate below using the UCCSD ansatz to find the minimum energy of an H_2 molecule:

```
from tencirchem import M, UCCSD

h2 = M(atom=[["H", 0, 0, 0], ["H", 0, 0, 0.741]])

uccsd = UCCSD(h2)
uccsd.kernel()
uccsd.print_summary(include_circuit=True)
```

Code Snippet 1: Simple UCCSD calculation for the hydrogen molecule.

The remainder of this section will explain Code Snippet 1 in more detail.

3.1.1. Defining the Molecule and Basis Set. TenCirChem uses the PySCF `Mole` object to define the molecule of interest. In other words, `from tencirchem import M` is equivalent to `from pyscf import M`. The syntax for defining molecules is thus the same as that for PySCF. For example, using 3D coordinates, atom types and coordinates (in Angstroms) are specified in a list as shown in Code Snippet 1. By default, the STO-3G minimal basis set is used, but PySCF allows other basis sets to be specified via the basis argument, e.g.,

```
h2 = M(atom=[["H", 0, 0, 0], ["H", 0, 0, 0.741]], basis="cc-pvdz")
```

Built-in Molecules. In addition to specifying molecules by hand, the user can import a number of prespecified molecules in the STO-3G basis set from the `tencirchem.molecule` module for debugging and fast prototyping, e.g.,

```
from tencirchem.molecule import h2, h4, h8
```

Once imported, the molecules can be used exactly as if they had been defined manually. The coordinates and the basis sets of these built-in molecules can be obtained by inspecting the `Mole` object with `m.atom` and `m.basis`, respectively. The `tencirchem.molecule.h2` molecule, with a bond distance of 0.741 Å, is frequently used in the following code examples.

3.1.2. Specifying the Ansatz and Obtaining Reference Energies. Once the H_2 molecule has been specified, we initialize the UCCSD ansatz via the line

```
uccsd = UCCSD(h2)
```

In addition to UCCSD, TenCirChem supports k-UpCCGSD and pUCCD ansätze. These are based on the UCC base class, and can be implemented in a similar way to UCCSD once imported via

```
from tencirchem import KUPCCGSD, PUCCD
```

Ansatz information is accessible through class attributes (see Code Snippet 2). The conventions for the orbital indices are described in the Supporting Information.

```
>>> # two single excitations and one double excitation
>>> uccsd.ex_ops
[(3, 2), (1, 0), (1, 3, 2, 0)]
>>> # 'param_ids' maps excitation operators to parameters
>>> # some excitation operators share the same parameter due to symmetry
>>> uccsd.param_ids
[0, 0, 1]
>>> uccsd.init_guess  # generated by MP2
[0.0, -0.07260814651571333]
>>> uccsd.params # the optimized parameters
array([ 3.40632986e-17, -1.12995353e-01])
```

Code Snippet 2: Several useful attributes of the UCCSD class. Example corresponds to the H_2 molecule.

UCCSD Class as a Toolbox. The `UCCSD` class also allows other useful attributes to be obtained (see Code Snippet 3). The purpose is to make information in TenCirChem easy to inspect, facilitating its use as a handy toolbox. For example, one may use the `UCCSD` class to inspect for molecular attributes, electron integrals, reference energies, and the Jordan-Wigner transformed Hamiltonian, all of which are available without needing to first call the `uccsd.kernel()` method.

```
>>> uccsd.n_qubits, uccsd.n_elec
(4, 2)
>>> # PySCF objects
>>> uccsd.mol, uccsd.hf
(<pyscf.gto.mole.Mole at 0x7f9daec92b90>, <pyscf.scf.hf.RHF at 0x7f9d92d1ec10>)
>>> # reference energies
>>> uccsd.e_hf, uccsd.e_mp2, uccsd.e_ccsd, uccsd.e_fci, uccsd.e_nuc
(-1.116706137236105,
 -1.1298675557838804,
 -1.1372745709766439,
 -1.1372744055294395,
 0.7141392859919029)
>>> # one and two-body integrals in molecular orbital basis, chemists' notation
>>> uccsd.int1e.shape, uccsd.int2e.shape
((2, 2), (2, 2, 2, 2))
>>> uccsd.h_fermion_op  # Hamiltonian as openfermion FermionOperator
0.7141392859919029 [] +
-1.2527052599711868 [0^ 0] +
-0.48227117798977825 [0^ 1^ 0 1] +
-0.6745650967143663 [0^ 2^ 0 2] +
[...... The rest of the output omitted ......]
>>> uccsd.h_qubit_op  # Hamiltonian as openfermion QubitOperator
(-0.09835117053027564+0j) [] +
(0.04531660419443148+0j) [X0 X1 X2 X3] +
(0.04531660419443148+0j) [X0 X1 Y2 Y3] +
(0.04531660419443148+0j) [Y0 Y1 X2 X3] +
[...... The rest of the output omitted ......]
```

Code Snippet 3: Several other useful attributes of the UCCSD class are also directly available without first needing to call the `uccsd.kernel()` method.

Specifying Integrals Directly. Rather than using a molecule as the input to UCCSD, one may define a UCCSD object by specifying the one- and two-electron integrals, without needing to explicitly refer to a molecule. That is, applying UCCSD to the second-quantized *ab initio* Hamiltonian (eq 1) can be done via

```
uccsd_from_integral = UCCSD.from_integral(int1e, int2e, n_elec, e_core)
```

with

- `int1e`: an array of shape (N, N) with p, q-th element storing the one-electron integral h_{pq}. Here N is the number of spatial orbitals.
- `int2e`: an array of shape (N, N, N, N) with p, q, r, s-th element storing the two-electron integral $(pq|rs)$, in spatial orbital and chemists' notation.

- n_elec: an integer specifying the total number of electrons in the system. Currently, TenCirChem only supports closed-shell molecules.
- e_core: a floating-point number for the nuclear repulsion energy E_{nuc} or the core energy if frozen occupied orbitals are involved.

3.1.3. Optimizing the Energy. The command uccsd.kernel() runs the optimization procedure to minimize the ansatz energy with respect to the variational parameters and returns the minimum energy found. By default, TenCirChem uses the L-BFGS-B optimizer implemented in SciPy.[81] Once uccsd.kernel() has been run, the minimum ansatz energy can be accessed by uccsd.energy() or uccsd.e_ucc. The system statevector and configuration interaction vector are available by uccsd.statevector() and uccsd.civector(), e.g.,

```
>>> uccsd.civector() # configuration interaction vector
array([ 9.93623806e-01,  1.08284918e-16,  1.08284918e-16, -1.12746318e-01])
```

The optimized parameters can be obtained by uccsd.params, and the one- and two-body reduced density matrices are available by uccsd.make_rdm1() and uccsd.make_rdm2(), respectively. Functions such as uccsd.energy, uccsd.statevector, and uccsd.civector also accept custom circuit variational parameters. For example,

```
>>> # if all parameters are zero, then the UCC energy is identical to HF energy
>>> uccsd.energy(np.zeros(uccsd.n_params)), uccsd.e_hf
(-1.1167061372361045, -1.116706137236105)
```

3.1.4. Outputting Quantum Circuits. The quantum circuit corresponding to the ansatz state can be obtained via

```
c = uccsd.get_circuit()
```

which returns a TensorCircuit Circuit object, which can then be inspected, manipulated, and executed. For example, a summary of the gates in the circuit can be obtained by

```
c.gate_summary()
```

Also, conversion of the circuit from the TensorCircuit to the Qiskit format is straightforward:

```
c_qiskit = c.to_qiskit()
```

By default, TenCirChem compiles circuits using the efficient method of Yordanov, Arvidsson-Shukur, and Barnes (YAB),[82,83] which is based on multiqubit controlled rotations (see Figure 3). Currently triple or higher order excitations are not implemented in TenCirChem. uccsd.get_circuit (decompose_multicontrol = True) generates the circuit with the multiqubit controlled gate decomposed into elementary rotation gates and CNOT gates. This decomposition is useful for noisy circuit simulation or execution on hardware. See Code Snippet 10 for an example. Compilation via a more traditional Trotterized method is also possible, using

```
c = uccsd.get_circuit(trotter=True)
```

although this can lead to circuits significantly deeper than those compiled by the YAB method.[82]

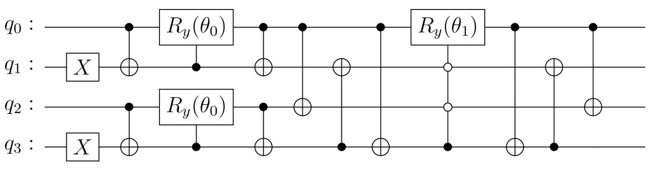


Figure 3. Quantum circuit corresponding to the UCCSD ansatz for the H_2 molecule with STO-3G basis, compiled by the method in ref 82. The two X gates at the left of the circuit are used to create the initial Hartree−Fock state $|0101\rangle$. The multicontrol gate applies a parametrized R_y rotation to one of the four qubits, conditioned on the state of the other three.

3.2. Applications and Benchmarking. In this section, we use concrete examples to demonstrate the efficiency and flexibility of TenCirChem. All calculations are carried out on a single computational node with the civector engine. Throughout the paper, the wall time reported only accounts for the VQE optimization time. Classical preprocessings, such as obtaining integrals and Hartree−Fock calculations, are not included in the reported time.

3.2.1. Hydrogen Chains and the Hydrogen Molecule. Table 1 gives the time required by TenCirChem to compute the minimum UCCSD energy for chains of increasing numbers of hydrogen atoms using an STO-3G basis set. Atoms in each chain are uniformly distributed with an interatomic distance set to 0.8 Å. The largest system simulated is H_{16}, corresponding to 32 qubits. For such a system, storing the wave function in Fock space alone is already an arduous task, while the running time for TenCirChem is still acceptable. Whereas other common quantum simulation packages usually struggle to simulate H_{10} (which requires 20 qubits), TenCirChem is able to obtain the UCCSD ground state energy of this chain in 14 s on a single GPU node. As a concrete example, Qiskit-Nature version 0.5.2 spends 34 s simulating H_4 and 943 s simulating H_6, and reports an out-of-memory error when simulating larger systems. Other packages, such as PennyLane and Tequilla, show a similar performance. We note that the high efficiency of TenCirChem is primarily due to the civector engine, and hardware acceleration plays a relatively minor role.

The hydrogen chain actually represents a class of molecular systems that are particularly difficult to simulate in TenCirChem. In these systems, the number of spatial orbitals N is the same as the number of electrons M. In the following, we consider a case at the other extreme, where $M \ll N$, and simulate the potential energy curve of the H_2 molecule using the cc-pVTZ basis set.[84] This corresponds to $M = 2$, $N = 28$, and a VQE circuit on 56 qubits. Running on a single laptop CPU, TenCirChem is able to produce all UCCSD energy values in Table 2 in 20 s. Because of the small value of M, both the dimension of the configuration interaction space (784) and the number of excitation operators (155) are small. Since the civector engine is exact, the small error in Table 2 is likely due to the inexactness of the disentangled UCC ansatz.

3.2.2. Potential Energy Curve of H_2O. We now compute the potential energy curve of a realistic molecule, H_2O. This molecule has been used to benchmark many quantum algorithms,[85,86] yet previous works usually use a minimal basis set or small active space. Here, we use TenCirChem to calculate the UCCSD energy of H_2O with the 6-31G(d) basis set[87,88] with only the 1s orbital of the O atom frozen. This corresponds to 8 electrons in 17 orbitals, a quantum circuit on 34 qubits, and an ansatz involving 565 independent parameters and 1078 excitations.

We study the symmetric stretching of the O−H bond with the H−O−H angle fixed at the experimental value of 104.45°.[89] Results are summarized in Figure 4. The UCCSD energy by TenCirChem is very close to that of the FCI solution, with a slight deviation observed at long bond lengths. The equilibrium bond length is determined to be 0.97 Å, in good agreement with

Table 1. UCCSD Ground State Energy Optimization Time for Hydrogen Chains Using TenCirChem[a]

molecule	qubits	excitations	parameters	device	time (s)	energy (mH)	error (mH)
H_4	8	18	11	CPU	0.05	−2,167.55	0.01
H_6	12	69	39	CPU	0.4	−3,204.14	0.27
H_8	16	200	108	CPU	1.9	−4,242.52	0.88
H_{10}	20	467	246	GPU	14	−5,282.21	1.37
H_{12}	24	954	495	GPU	51	−6,322.29	2.13
H_{14}	28	1,749	899	GPU	2,093	−7,362.81	2.92
H_{16}	32	2,976	1,520	GPU	50,909	−8,403.63	3.72

[a]CPU calculations are based on an Intel(R) Xeon(R) Platinum 8255C CPU @ 2.50 GHz, while GPU calculations use the NVIDIA(R) Tesla(R) V100-PCIE-32GB. Energy errors are with respect to FCI energy, and could be reduced by adding more excitation operators in the ansatz.[71].

Table 2. UCCSD/cc-pVTZ Calculation of H_2 in TenCirChem, Using a VQE Circuit on 56 Qubits. Energy Errors Are with Respect to FCI

atom distance (Å)	energy (mH)	error (mH)
0.3	−692.85983	0.00000
0.6	−1153.51794	0.00000
0.9	−1160.41280	0.00001
1.2	−1112.06753	0.00005
1.5	−1066.16823	0.00014
1.8	−1034.07993	0.00023
2.1	−1015.56097	0.00028
2.4	−1006.42068	0.00029
2.7	−1002.39048	0.00028

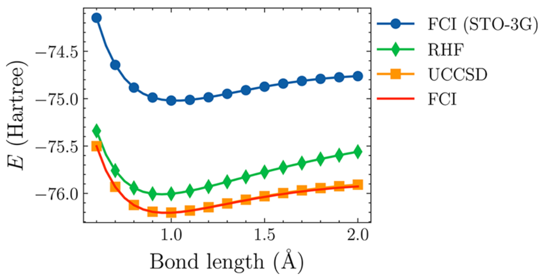


Figure 4. Potential energy curve for the symmetric stretching of the O−H bond in H_2O with the 6-31G(d) basis set. The 1s orbital of the O atom is frozen, and the whole system is described by an (8e, 17o) active space and a quantum circuit on 34 qubits. The FCI energy with STO-3G basis set is included for comparison.

the experimental value (0.9584 Å). To highlight the necessity of using an appropriate basis set, in Figure 4 the FCI energy using the minimal STO-3G basis set is also given. As expected, the energy corresponding to the STO-3G basis set is significantly higher than that corresponding to the 6-31G(d) basis set. Furthermore, the equilibrium bond distance predicted using the STO-3G basis set is 1.02 Å, which is significantly larger than the experimental value. On average, TenCirChem takes approximately 15 min to produce each UCCSD energy value in Figure 4, using the same GPU platform as in Table 1.

3.2.3. Hubbard Model. As a final example, we use TenCirChem and UCCSD to calculate the ground state energy of the 1D Hubbard model at half-filling. The Hamiltonian is

$$H = -t\sum_{j,\sigma}(c^{\dagger}_{j+1,\sigma}c_{j,\sigma} + c^{\dagger}_{j,\sigma}c_{j+1,\sigma}) + U\sum_{j} n_{j\uparrow}n_{j\downarrow} \tag{18}$$

where $c^{\dagger}_{j,\sigma}$ ($c_{j,\sigma}$) creates (annihilates) an electron with spin σ at site j, $n_{j\uparrow}$ and $n_{j\downarrow}$ are electron occupation number operators at site j, t is the hopping integral and U characterizes the Coulomb repulsion interaction strength. We assumed periodic boundary conditions.

Although the Hubbard model is not directly related to any molecular systems, the Hamiltonian is a special case of the *ab initio* Hamiltonian of eq 1, and thus, TenCirChem allows the construction of the corresponding UCCSD objects by specifying the one-body and two-body integrals.

```python
import numpy as np
from tencirchem import UCCSD

n = 6  # the number of sites
n_elec = n  # half filled
t = U = 1  # model parameters

# setup the integrals
int1e = np.zeros((n,n))
for i in range(n - 1):
    int1e[i, i + 1] = int1e[i + 1, i] = -t
int1e[n - 1, 0] = int1e[0, n - 1] = -t
int2e = np.zeros((n, n, n, n))
for i in range(n):
    int2e[i, i, i, i] = U

# do the calculation
uccsd = UCCSD.from_integral(int1e, int2e, n_elec)
uccsd.kernel()
```

Code Snippet 4: Calculation of the UCCSD energy of half-filled Hubbard model.

Results are shown in Figure 5. When U/t is small, both the CCSD and UCCSD coincide well with the FCI solution. As U/t increases, CCSD deviates from the FCI solution, whereas the UCCSD solution tracks the FCI value much more closely, indicating that UCCSD can be better than CCSD at capturing strong correlations, a finding consistent with a previous report.[90]

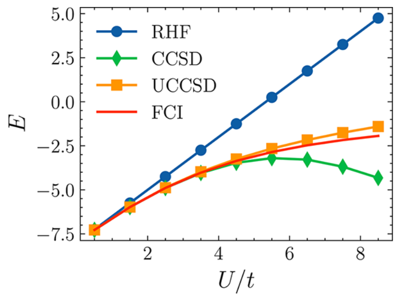


Figure 5. Ground state energy of the one-dimensional half-filled Hubbard model by UCCSD with increasing coulomb repulsion strength U/t. The UCCSD energy computed by TenCirChem tracks the FCI value significantly better than the CCSD energy.

To better recover static correlation using UCCSD, more terms can be added to the UCC ansatz or one may consider combining UCC with other algorithms, such as density matrix embedding theory.[47,91−94] On average, TenCirChem requires 0.1 s to calculate each UCCSD energy value in Figure 5 on a regular CPU without GPU acceleration.

4. THE ELECTRONIC STRUCTURE MODULE II. HARDWARE-EFFICIENT ANSATZ AND NOISY CIRCUIT SIMULATION

In real-world quantum computing systems, noise and errors are inevitable due to factors such as imperfect hardware, decoherence, and finite-shot measurement uncertainty. Noisy circuit simulators allow researchers to simulate the effects of noise on quantum circuits and gain a better understanding of the behavior of quantum algorithms in realistic conditions. In this section, we use the hydrogen molecule H_2 to illustrate how to perform noisy circuit simulation in TenCirChem. After using the parity transformation, the corresponding Hamiltonian acts on 2 qubits.

4.1. Basis Usage. TenCirChem uses the HEA (Hardware-Efficient Ansatz) class for noisy circuit simulation, which has an interface different from that of the UCC classes encountered in section 3. A primary motivation for using HEA ansätze is that almost all UCC circuits (with perhaps pUCCD circuits being the exception) are too large for both noisy circuit simulation as well as execution on currently available quantum devices. In contrast, HEA circuits are, by design, constructed to be implementable on whatever device is available.

Another reason is that from a user's perspective, the HEA class ought to provide a greater degree of low-level customization compared to the UCC classes. For example, for noiseless UCC simulation, the ansatz is defined by excitation operators, and compilation to native gates is of less interest. Conversely, in the context of noisy HEA circuit simulation, it is imperative to give users complete control over the ansatz circuit. Nevertheless, noisy UCC circuit simulation can be carried out within TenCirChem without much additional effort, which will be demonstrated in section 4.1.3.

4.1.1. Specifying the Hamiltonian and the Ansatz. Unlike UCC, the HEA class is not initialized by specifying the molecule. Rather, HEA takes as inputs (i) the Hamiltonian in `openfermion.QubitOperator` form and (ii) the circuit ansatz.

Within OpenFermion, the Hamiltonian can be constructed by interfacing with quantum chemistry packages such as PySCF using `openfermion.MolecularData`.

```python
from openfermion import MolecularData
from openfermionpyscf import run_pyscf

geometry = [["H", [0, 0, 0]], ['H', [0, 0, 0.741]]]
basis = "STO-3G"
multiplicity = 1
molecule = MolecularData(geometry, basis, multiplicity)
molecule = run_pyscf(molecule)
h_fermion_op = molecule.get_molecular_hamiltonian()
```

Code Snippet 5: Construction of the system Hamiltonian in fermion operator form using `OpenFermion`.

An alternative approach is to use the `UCC` class attributes such as `uccsd.h_fermion_op`.

```python
from tencirchem import UCCSD, parity
from tencirchem.molecule import h2

uccsd = UCCSD(h2)
# Hamiltonian as openfermion.FermionOperator
h_fermion_op = uccsd.h_fermion_op
# use parity transformation for qubit Hamiltonian and remove 2 qubits
h_qubit_op = parity(h_fermion_op, uccsd.n_qubits, uccsd.n_elec)
```

Code Snippet 6: Construction of the system Hamiltonian in qubit operator form using UCCSD class attributes.

The ansatz is defined using TensorCircuit. In the example below, we construct a 1-layer R_y ansatz comprising 4 parameters, and the circuit diagram is depicted in Figure 6. The initial parameters create the HF state $|01\rangle$ under parity transformation. In addition to defining the ansatz from scratch, it is also possible to use a predefined ansatz, which will be discussed more thoroughly in section 4.1.3.

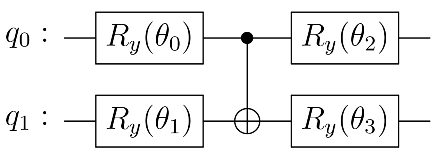


Figure 6. Circuit diagram for a 2-qubit 1-layered R_y ansatz. More layers can be added by repeating the CNOT and R_y rotation unit.

```python
import numpy as np
from tensorcircuit import Circuit

n_qubits = 2
n_params = 4
init_guess = [0, np.pi, 0, 0]
def get_circuit(params):
    c = Circuit(n_qubits)
    # the ansatz body
    c.ry(0, theta=params[0])
    c.ry(1, theta=params[1])
    c.cnot(0, 1)
    c.ry(0, theta=params[2])
    c.ry(1, theta=params[3])
    return c
```

Code Snippet 7: Defining the 1-layer R_y ansatz using TensorCircuit

4.1.2. Noisy Circuit Simulation and Optimization. The remaining workflow is similar to the `UCCSD` case: (i) create an HEA instance, (ii) run the kernel, and (iii) print the summary:

```python
1 from tencirchem import HEA
2 hea = HEA(h_qubit_op, get_circuit, init_guess, engine="tensornetwork-noise")
3 hea.kernel()
4 hea.print_summary()
```

The default optimizer used is again L-BFGS-B from SciPy.

The gradients are evaluated by the parameter-shift rule.[32,33] The parameter-shift rule uses two separate circuits to evaluate the gradient of one parameter. Its essence can be summarized using a single-qubit, single-gate illustrative example. Suppose the circuit wave function is $|\psi\rangle = e^{\theta R}|\phi\rangle$ with $R^2 = -I$. This leads to $|\psi\rangle = (\cos\theta + \sin\theta R)|\phi\rangle$, and the energy expectation is then

$$\langle E\rangle(\theta) = \cos^2\theta\langle\varphi|H|\varphi\rangle + \sin\theta\cos\theta\langle\varphi|RH + HR|\varphi\rangle + \sin^2\theta\langle\varphi|RHR|\varphi\rangle \tag{19}$$

Thus

$$\frac{\partial\langle E\rangle}{\partial\theta} = \langle E\rangle\left(\theta + \frac{\pi}{4}\right) - \langle E\rangle\left(\theta - \frac{\pi}{4}\right) \tag{20}$$

The expression is slightly different from the literature where $e^{(\theta/2)R}$ is usually assumed.

By default, using the `tensornetwork-noise` engine (line 2) adds isotropic depolarizing noise[95] with error probability $p = 0.02$

$$\mathcal{E}(\rho) = (1 - p)\rho + \frac{p}{15}\sum_{i=1}^{15}\mathcal{P}_j\rho\mathcal{P}_j \tag{21}$$

to each CNOT gate in the circuit. Here ρ is the density matrix, and the $\mathcal{P}_j$ are the two-qubit Pauli matrices (excluding the identity operator I). Hereafter we will also refer to p as the CNOT error probability. The value $p = 0.02$ corresponds to approximately 98% CNOT average gate fidelity, defined as

$$F = \int\langle\psi|\mathcal{E}(|\psi\rangle\langle\psi|)|\psi\rangle d\psi = 1 - \frac{4}{5}p \tag{22}$$

Single qubit gate noise is not included. The energy obtained in this case is −1.1203.

Other noise models can be specified using the `NoiseConf` class from TensorCircuit, as illustrated below:

```
from tensorcircuit.noisemodel import NoiseConf
from tensorcircuit.channels import isotropicdepolarizingchannel

engine_conf = NoiseConf()
# larger noise, corresponding to 80% fidelity
channel = isotropicdepolarizingchannel(p=0.25, num_qubits=2)
engine_conf.add_noise("cnot", channel)

hea = HEA(h_qubit_op, get_circuit, init_guess, \
          engine="tensornetwork-noise", engine_conf=engine_conf)
hea.kernel()
hea.print_summary()
```

Code Snippet 8: A simple customized noise model with depolarizing error probability p = 0.25, corresponding to average gate fidelity of 80%. The energy obtained is −0.9245, higher than the value of −1.1203 obtained from the p = 0.02 case.

In addition to the depolarizing channel, TensorCircuit supports the amplitude damping channel, phase damping channel, and thermal relaxation channel.[95]

The `tensornetwork-noise` engine does not consider measurement uncertainty. Measurement uncertainty can be accounted for using the `tensornetwork-noise&shot` engine, as shown below:

```
>>> # number of shots: 4096
>>> for i in range(5):
>>>     print(hea.energy(engine="tensornetwork-noise&shot"))
-0.9209869918508058
-0.9306141224500795
-0.9176478740443613
-0.929275708950061
-0.9241459510180587
>>> # number of shots: 4096 * 128"
>>> hea.shots = 4096 * 128
>>> for i in range(5):
>>>     print(hea.energy(engine="tensornetwork-noise&shot"))
-0.9246797253478469
-0.9251027083245665
-0.9243841547125008
-0.9236478492038139
-0.9248303187894168
```

Code Snippet 9: Accounting for measurement unceraínty in noisy circuit simulation using the `tensornetwork-noise&shot` engine.

By default, the number of measurement shots made for each term in the Hamiltonian is 4096. Increasing the number of measurement shots decreases the energy uncertainty. Here the engine is switched temporarily at runtime, and specifying the engine while initializing the `HEA` class is also viable, by `hea=HEA(*args, engine=engine)`.

If desired, a real quantum hardware engine can be specified by setting the engine to `qpu`. The circuit is then executed on a 9-qubit superconducting QPU hosted by the Tencent Quantum Lab quantum cloud service. A private token is required for successful execution. The platform is currently under closed beta, and we are developing more features. The access token and configuration documents can be requested by sending an e-mail to the authors.

If noiseless results are desired for the HEA ansatz, this can be achieved by using the `tensornetwork` engine, i.e., `hea.energy(engine="tensornetwork")`. In this case, the energy is in exact agreement with the FCI energy for the hydrogen molecule system.

4.1.3. Using Predefined Ansätze.

At the end of the section, we show how to use predefined ansatze for noisy circuit simulation. TenCirChem has implemented the R_y ansatz. In the following code, the `HEA` instance is rebuilt using the `HEA.ry` function, which constructs the qubit Hamiltonian and the R_y ansatz automatically. The result after running the kernel is exactly the same as what we've illustrated step by step above.

```
hea = HEA.ry(uccsd.int1e, uccsd.int2e, uccsd.n_elec, uccsd.e_core,  \
             n_layers=1, engine="tensornetwork-noise")
hea.kernel()
```

If desired, the UCC ansatz defined in the UCC class can be fed into the `HEA` class for a noisy circuit simulation. The following code snippet shows how to simulate the UCCSD circuit of the H_2 molecule in the presence of noise. As described in section 3.1.4, TenCirChem by default uses a multiqubit controlled R_y gate for UCC circuit simulation, which has to be decomposed into elementary gates before actual execution on quantum computers. The gradient is turned off because the parameter-shift rule is not directly applicable to the circuit in which multiple gates share the same parameter. The COBYLA optimizer implemented in SciPy[81] is used for optimization. The energy obtained is −0.8358, which is significantly higher than the energy by the HEA ansatz because there are 18 noisy CNOT gates in the circuit. We comment that simulating UCC circuits with the HEA class is nonstandard. Usually, it is necessary to simplify the ansatz before execution under realistic hardware conditions.[11,12]

```
from functools import partial
get_circuit = partial(uccsd.get_circuit, decompose_multicontrol=True)
hea = HEA(uccsd.h_qubit_op, get_circuit, uccsd.init_guess, \
          engine="tensornetwork-noise")
hea.grad = "free"
hea.kernel()
```

Code Snippet 10: Noisy simulation of UCC circuits.

4.2. Applications.

4.2.1. Effect of Gate Noise on VQE Energy.

As a first application, we show how the CNOT depolarizing error affects the optimized VQE energy. Results are listed in Figure 7. The R_y hardware-efficient ansatz and the H_2 molecule with parity transformation are used, as in section 4.1.

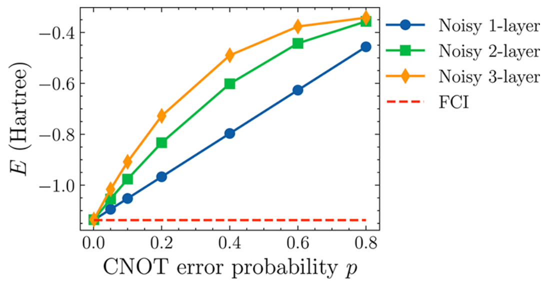


Figure 7. VQE ground state energy of H_2 with parity transformation in the presence of depolarizing error, characterized by the error probability p. The R_y hardware-efficient ansatz is used, with between 1 and 3 layers.

We additionally tested how the number of ansatz layers affects the obtained energy. If a one-layer ansatz is adopted, the energy increases linearly with the error probability up to $p = 0.8$. This is a consequence of the fact that this ansatz, when compiled into a quantum circuit, contains only a single CNOT gate. Increasing the number of layers does not lead to a more accurate energy estimation. Rather, the noise caused by more CNOT gates in the ansatz further degrades the results. TenCirChem requires approximately 1 s to produce each data point in Figure 7, which involves iterating the VQE parameter optimization procedure until convergence.

4.2.2. Effect of Measurement Shots on VQE Energy Uncertainty.

Here we show how the number of measurement shots N_{shots} affects the standard deviation of the optimized VQE energy. Results are listed in Figure 8. The H_2 molecule of section 4.2.1 is used with a single-layer R_y ansatz. We tested N_{shots} values from 2^8 to 2^{13}. For each value of N_{shots} investigated, the system energy E is evaluated 64 times, and the standard deviation is calculated. Note that the circuit parameters are kept constant at the optimal value and the VQE iteration is not run.

As expected, the standard deviation of E is proportional to $\sqrt{\frac{1}{N_{\text{shots}}}}$.[1] Increasing the error probability p causes a larger standard deviation of the energy, although the effect is far less significant than that caused by adjusting N_{shots}. TenCirChem

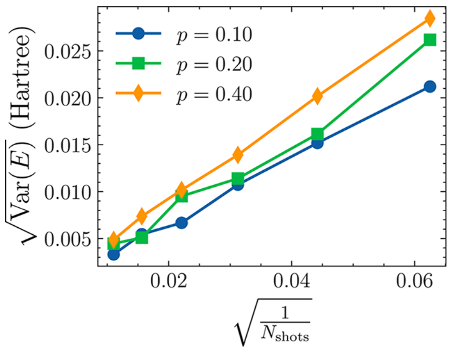


Figure 8. VQE energy in the presence of depolarizing error, characterized by the error probability p. The R_y circuit with 1 layer is used as the ansatz.

requires approximately 1 s to produce a single data point in Figure 8, each of which involves 64 separate energy evaluations.

4.2.3. Running Quantum Hardware Experiments. In this section, we use QPUs to compute the VQE energy of the H_2 molecule, and the results are summarized in Figure 9. We use the same one-layered R_y ansatz described in previous sections. To reduce the computational cost, the parameter vector is reduced to $[\theta, \pi, 0, 0]$ with only one variable θ. Noiseless simulation shows that using this ansatz is sufficient to reach the FCI energy. On QPU, 8192 shots are taken to determine the expectation value of each term in the Hamiltonian. The error is mitigated by standard readout error mitigation. For each set of circuit parameters, the energy is evaluated 8 times to produce the final energy expectation and its standard deviation. We first compute the potential energy curve using the classically optimized parameter, and then run the full VQE optimization starting from $\theta = 0$. In both cases, the QPU engine can correctly describe the dissociation curve of the H_2 molecule. Moreover, we observe that the energy standard deviation increases as the bond length decreases. This is because at short bond lengths the Pauli strings have large coefficients due to strong interactions between the particles.

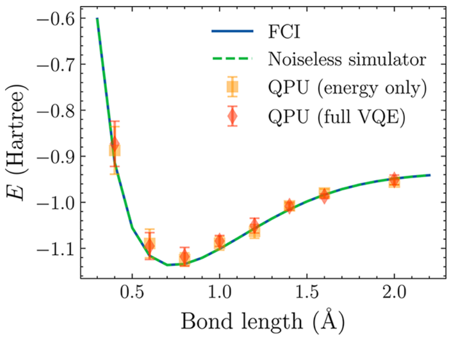


Figure 9. VQE potential energy curve of H_2 computed on a QPU backend. QPU energies obtained from the classically optimized parameter and the full VQE optimization process are both shown.

5. THE QUANTUM DYNAMICS MODULE

5.1. Basic Usage.

5.1.1. Defining the Hamiltonian and Basis Set. The first step in quantum dynamics simulations is specifying the Hamiltonian. In this section, we focus on the spin-boson model with 1 bosonic mode:

$$H = \frac{\epsilon}{2}\sigma_z + \Delta\sigma_x + \omega b^\dagger b + g\sigma_z(b^\dagger + b) \tag{23}$$

where $\sigma_z = Z$ and $\sigma_x = X$ are Pauli matrices, ϵ is the eigenfrequency, and Δ is the tunnelling rate. The spin-boson model is a classical model for a variety of chemical processes, such as electron transfer and photochemistry.[96] For ease of demonstration, we truncate the allowed boson states to two levels. More sophisticated examples can be found in section 5.3.

Defining the system Hamiltonian and basis sets is performed as follows:

```python
 1 from tencirchem import Op, BasisHalfSpin, BasisSHO
 2
 3 epsilon = 0
 4 delta = 1
 5 omega = 1
 6 g = 0.5
 7
 8 ham_terms = [
 9     Op("sigma_z", "spin", epsilon),
10     Op("sigma_x", "spin", delta),
11     Op(r"b^\dagger b", "boson", omega),
12     Op("sigma_z", "spin", g) * Op(r"b^\dagger+b", "boson")
13 ]
14 basis = [BasisHalfSpin("spin"), BasisSHO("boson", omega=omega, nbas=2)]
```

Code Snippet 11: Defining the system Hamiltonian and basis set for 1-mode spin-boson model.

The `Op` and the basis set classes are directly imported from the RENORMALIZER package. Each term in the Hamiltonian takes three arguments: (i) the operator symbol; (ii) the name of the associated degree of freedom; and (iii) the (optional) coefficient.

In setting the basis sets (line 14), `BasisSHO` refers to the simple harmonic oscillator basis, and `nbas=2` restricts the phonon states to the lowest two levels.

Conversion of the Hamiltonian and basis sets into qubits is then performed in TENCIRCHEM using the `qubit_encode_op` and `qubit_encode_basis` commands, as illustrated below:

```python
from tencirchem.dynamic import qubit_encode_op, qubit_encode_basis
boson_encoding = "gray"
ham_terms_spin, constant = qubit_encode_op(ham_terms, basis, boson_encoding)
basis_spin = qubit_encode_basis(basis, boson_encoding)
```

Code Snippet 12: Transformation from physical basis set into qubit spin basis. Here we use the binary encoding with Gray code approach (see section 2.2.2) to encode the bosonic states.

After transformation, the Hamiltonian and basis sets can be inspected as follows:

```python
>>> ham_terms_spin
[Op('X', ['spin'], 1.0),
 Op('Z', [('boson', 'TCCQUBIT-0')], -0.5),
 Op('Z X', ['spin', ('boson', 'TCCQUBIT-0')], 0.5)]
>>> basis_spin
[BasisHalfSpin(dof: spin, nbas: 2),
 BasisHalfSpin(dof: ('boson', 'TCCQUBIT-0'), nbas: 2)]
```

Code Snippet 13: Inspection of the transformed Hamiltonian.

As expected, the Hamiltonian is now expressed in terms of Pauli strings and the basis sets are all transformed to the spin-$\frac{1}{2}$ basis set. Here, the term in square brackets in each operator of the above Code Snippet represents the label(s) of the spin-$\frac{1}{2}$ basis on which the Pauli operators are defined. As we restrict to two phonon levels, only one spin-$\frac{1}{2}$ basis set with label (`boson`, `TCCQUBIT-0`) is generated, i.e., the phonon mode is represented by a single qubit.

5.1.2. Construct the Ansatz. The variational Hamiltonian ansatz (see Figure 10) defined in eq 16 can now be constructed using the `get_ansatz` function, and the Jacobian is then obtained by `get_jacobian_func`:

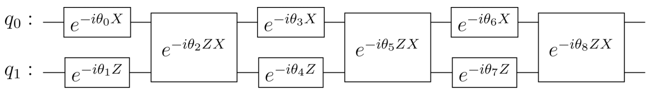


Figure 10. Quantum circuit corresponding to the 3-layered variational Hamiltonian ansatz for the 1-mode spin-boson model using binary encoding with Gray code. q_0 represents the spin, and q_1 represents the boson.

```python
import tensorcircuit as tc
from tencirchem import set_backend
from tencirchem.dynamic import get_ansatz, get_jacobian_func

# dynamics simulation requires auto-differentiation from JAX.
set_backend("jax")

# the initial state
init_circuit = tc.Circuit(len(basis_spin))
# number of layers
n_layers = 3
# get the ansatz. Note that the spin basis is fed in
ansatz = get_ansatz(ham_terms_spin, basis_spin, n_layers, init_circuit)

# ansatz accepts parameters and outputs wavefunction
import numpy as np
print(ansatz(np.zeros(n_layers * len(ham_terms_spin))))
# the function to evaluate the jacobian of the wavefunction
jacobian_func = get_jacobian_func(ansatz)
```

Code Snippet 14: Construction of the ansatz and the function to evaluate the Jacobian.

Here, `basis_spin` defines the ordering of the qubits in the circuit. More specifically, the first and second elements in `basis_spin` refer to the spin and boson, respectively, and thus in the quantum circuit the first qubit represents the spin and the second qubit represents the boson. Note that the backend needs to be set to JAX explicitly because the default NumPy backend does not support automatic differentiation.

5.1.3. Time Evolution. With the ansatz function to calculate the wave function and its Jacobian, the equation of motion eq 5 can now be readily solved to determine $\dot{\theta}_k$. The following code wraps the derivative function in the SciPy format and solves the initial value problem. The wave function is evolved from time $t = 0$ to $t = 10$ using a time interval of 0.1.

```python
from tencirchem import get_dense_operator
from tencirchem.dynamic import get_deriv
# the Hamiltonian in dense matrix format
h = get_dense_operator(basis_spin, ham_terms_spin)
# time derivative for circuit parameters in the scipy solve_ivp format
def scipy_deriv(t, _theta):
    return get_deriv(ansatz, jacobian_func, _theta, h)
from scipy.integrate import solve_ivp
# time step
tau = 0.1
# initial value
theta = np.zeros(n_layers * len(ham_terms_spin))
for n in range(100):
    # time evolution
    scipy_sol = solve_ivp(scipy_deriv, [n * tau, (n + 1) * tau], theta)
    # time evolved parameter
    theta = scipy_sol.y[:, -1]
```

Code Snippet 15: Time evolution of the spin-boson system.

Note that TenCirChem also provides a high-level interface for simulating dynamics, e.g.,

```python
from tencirchem import TimeEvolution

te = TimeEvolution(ham_terms, basis)
for i in range(100):
    te.kernel(0.1)
```

Code Snippet 16: Time evolution of the spin-boson system using the `TimeEvolution` class.

Using this interface, the user need only specify the Hamiltonian and the basis sets, and encoding to qubits is carried out automatically using binary encoding with Gray code. Configuration of the encoding strategy, the circuit initial condition, and the number of layers in the ansatz are also supported.

5.2. Implementation. TenCirChem uses classical matrix manipulations to calculate the M matrix and V vector in the equation of motion (eq 4) instead of faithfully simulating the quantum circuit. The first step is to obtain the Jacobian $\frac{\partial|\Psi\rangle}{\partial\theta_k}$ by forward-mode autodifferentiation via the JAX backend. Forward-mode autodifferentiation is preferred over reverse-mode autodifferentiation in this case, because usually there are more amplitudes in $|\Psi\rangle$ than parameters θ_k. The whole matrix M is then calculated by a single matrix−matrix multiplication, according to eq 5. Similarly, the whole vector V is calculated by computing $H|\psi\rangle$ via a single matrix-vector multiplication and then another matrix-vector multiplication between the Jacobian and $H|\psi\rangle$. This implementation minimizes the just-in-time compile time in JAX, because evaluating the Jacobian is the only bottleneck that needs to be just-in-time compiled during the workflow. The M matrix is usually not invertible due to linear dependencies in $\frac{\partial|\Psi\rangle}{\partial\theta_k}$. Its eigenvalues λ_i are modified with $\lambda_i \to \lambda_i + \epsilon e^{-\lambda_i/\epsilon}$ for regularization, where ϵ is a predefined small number, typically 1×10^{-5}.

5.3. Applications. *5.3.1. Spin Relaxation of Spin-Boson Model.* Here we show the simulated dynamics of the one-mode spin-boson model defined in eq 23, with parameters $\epsilon = 0$, $\Delta = 1$, $\omega = 1$, and $g = 0.5$. Unlike the previous example which truncated the boson to the lowest two states, here we allow 8 bosonic states, and the corresponding circuit acts on 4 qubits. The simulated results are shown in Figure 11, and they are in good agreement with the solution with exact diagonalization.

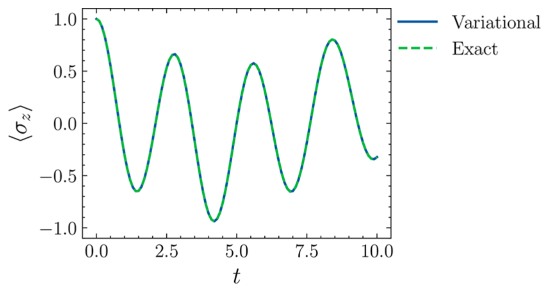


Figure 11. Spin relaxation of the one-mode spin-boson model simulated by variational quantum dynamics.

5.3.2. Marcus Inverted Region of the Charge Transfer Model. In this section, we simulate the charge transfer or oxidation−reduction reaction between two molecules using the Marcus model[97]

$$H = -V(a_0^\dagger a_1 + a_1^\dagger a_0) + \Delta G a_1^\dagger a_1 + \omega \sum_{i=0,1} b_i^\dagger b_i + g\omega \sum_{i=0,1} a_i^\dagger a_i (b_i^\dagger + b_i) \tag{24}$$

Here a_0 and a_1 are annihilation operators for the charge on the first and the second molecule, respectively. We assume a transfer integral $V = -0.1$, a dimensionless coupling constant $g = 1$, and a vibration frequency $\omega = 0.5$. The Marcus charge transfer theory predicts that, by decreasing the reaction Gibbs free energy change ΔG, the reaction rate k will first increase and then decrease, according to

$$k = \frac{V^2}{\hbar}\sqrt{\frac{\pi\beta}{\lambda}} \exp\left\{-\frac{\beta(\lambda + \Delta G)^2}{4\lambda}\right\} \tag{25}$$

where $\lambda = 2g^2\omega$ is the reorganization energy and β is the inverse temperature. Due to the quadratic term on the exponential, the maximum rate is reached when $\Delta G = -\lambda = -1$.

In the simulation, we again truncate the number of boson states to 8, and thus, the Hamiltonian is encoded into 7 qubits using the Gray code method. The charge is initially set to be

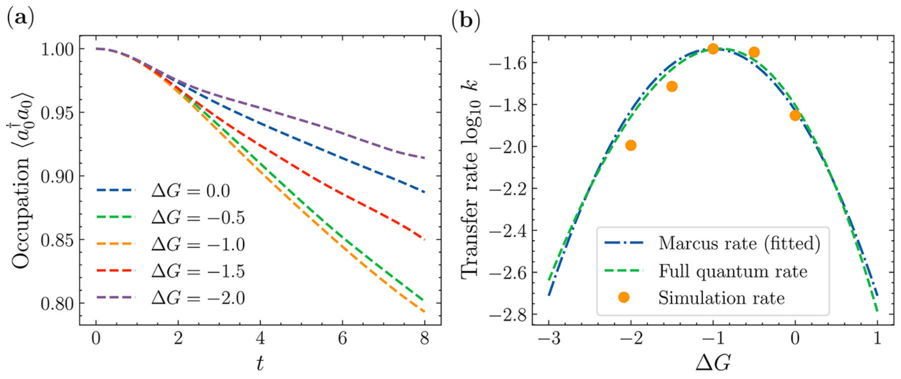


Figure 12. (a) Charge occupation on the first molecule vs time at several values of ΔG. (b) Charge transfer rate k as a function of ΔG, for $2 < t < 8$. The dots are from a linear fit based on $\langle a_0^\dagger a_0 \rangle(t) = -kt + c$ to the linear region in (a), and the dashed lines are the predictions of the Marcus theory (eq 25) and full quantum charge transfer theory.

located at the first molecule, and the nuclear coordinates are relaxed according to the localized charge.[98] The system is then evolved to $t = 8$ using TenCirChem and we compute $\langle a_0^\dagger a_0 \rangle$ (the charge occupation on the first molecule) as a function of time, with ΔG ranging from 0 to -2.

Results are shown in Figure 12(a). During an initial period ($t < 2$), the reaction rate increases with t. For $2 < t < 8$, the reaction reaches a steady state and the reaction rate is approximately constant.

In Figure 12(b) we plot the reaction rate k from $t = 2$ to $t = 8$. Since the effect of temperature is not modeled in this simulation, β is fit to a value of 2.71 by matching the simulated $k(\Delta G)$ data with the least-square method. The simulated rate reaches the maximum at $\Delta G = -1$, in agreement with the theoretical prediction. We also include the charge transfer rate from full quantum nuclear tunneling enabled charge transfer theory,[99] which predicts a very similar parabolic curve without fitting β.

6. SUMMARY AND OUTLOOK

In this paper, we introduce TenCirChem, a Python-based open-source library for quantum computational chemistry, hosted on the GitHub repository https://github.com/tencent-quantum-lab/TenCirChem. TenCirChem aims to provide both black-box calculations of existing quantum algorithms and flexible interfaces for the rapid prototyping of novel computational methods. TenCirChem features high-performance UCC calculations, noisy circuit simulations with both quantum gate error and measurement noise, and variational quantum dynamics simulations. TenCirChem is designed to expose its internal data structure for inspection and modification, enabling in-depth customization. Although written in Python, TenCirChem reaches state-of-the-art performance via the powerful TensorCircuit backend and various chemistry-focused optimizations such as UCC factor expansion. Its efficiency is demonstrated by the exact UCCSD simulation of H_{16} in the STO-3G basis (32 qubits), H_2 in the cc-pVTZ basis (56 qubits), and H_2O in the 6-31G(d) basis (34 qubits), requiring only moderate computational time and hardware.

The development of TenCirChem is an ongoing process. In the future, additional QPU engine configuration options will be provided. Features such as support for higher order excitations, the treatment of open-shell systems, algorithms for excited states and periodic systems, algorithms for fault-tolerant quantum computers, further performance optimization, and more code examples reproducing published algorithms are also under active consideration.

■ ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge at https://pubs.acs.org/doi/10.1021/acs.jctc.3c00319.

Brief review of UCC ansätze implemented in TenCirChem; conventions for orbital indices and qubit indices in TenCirChem; output of the print_summary command; advanced features of the UCC classes; algorithm for efficient simulation of UCC circuits (PDF)

■ AUTHOR INFORMATION

Corresponding Authors

Weitang Li − *Tencent Quantum Lab, Shenzhen 518057, China*; orcid.org/0000-0002-8739-641X; Email: liw31@gmail.com

Shengyu Zhang − *Tencent Quantum Lab, Hongkong 999077, China*; Email: shengyzhang@tencent.com

Authors

Jonathan Allcock − *Tencent Quantum Lab, Hongkong 999077, China*

Lixue Cheng − *Tencent Quantum Lab, Shenzhen 518057, China*

Shi-Xin Zhang − *Tencent Quantum Lab, Shenzhen 518057, China*

Yu-Qin Chen − *Tencent Quantum Lab, Shenzhen 518057, China*

Jonathan P. Mailoa − *Tencent Quantum Lab, Shenzhen 518057, China*

Zhigang Shuai − *Department of Chemistry, Tsinghua University, Beijing 100084, China; School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen 518172, China*; orcid.org/0000-0003-3867-2331

Complete contact information is available at: https://pubs.acs.org/10.1021/acs.jctc.3c00319

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

The authors thank Zhaofeng Ye for designing the TenCirChem logo. Weitang Li would like to thank Sainan Huai and Tianqi Cai for their helpful discussions. This work is supported by the National Natural Science Foundation of China through grant numbers 22273005 and 21788102. This work is also supported by Shenzhen Science and Technology Program.

REFERENCES

(1) McClean, J. R.; Romero, J.; Babbush, R.; Aspuru-Guzik, A. The theory of variational hybrid quantum-classical algorithms. *New J. Phys.* **2016**, *18*, 023023.

(2) Cao, Y.; Romero, J.; Olson, J. P.; Degroote, M.; Johnson, P. D.; Kieferová, M.; Kivlichan, I. D.; Menke, T.; Peropadre, B.; Sawaya, N. P.; Sim, S.; Veis, L.; Aspuru-Guzik, A. Quantum chemistry in the age of quantum computing. *Chem. Rev.* **2019**, *119*, 10856−10915.

(3) Bauer, B.; Bravyi, S.; Motta, M.; Chan, G. K.-L. Quantum algorithms for quantum chemistry and quantum materials science. *Chem. Rev.* **2020**, *120*, 12685−12717.

(4) McArdle, S.; Endo, S.; Aspuru-Guzik, A.; Benjamin, S. C.; Yuan, X. Quantum computational chemistry. *Rev. Mod. Phys.* **2020**, *92*, 015003.

(5) Liu, J.; Fan, Y.; Li, Z.; Yang, J. Quantum algorithms for electronic structures: basis sets and boundary conditions. *Chem. Soc. Rev.* **2022**, *51*, 3263.

(6) Preskill, J. Quantum Computing in the NISQ era and beyond. *Quantum* **2018**, *2*, 79.

(7) Arute, F.; Arya, K.; Babbush, R.; Bacon, D.; Bardin, J. C.; Barends, R.; Biswas, R.; Boixo, S.; Brandao, F. G. S. L.; Buell, D. A.; Burkett, B.; Chen, Y.; Chen, Z.; Chiaro, B.; Collins, R.; Courtney, W.; Dunsworth, A.; Farhi, E.; Foxen, B.; Fowler, A.; Gidney, C.; Giustina, M.; Graff, R.; Guerin, K.; Habegger, S.; Harrigan, M. P.; Hartmann, M. J.; Ho, A.; Hoffmann, M.; Huang, T.; Humble, T. S.; Isakov, S. V.; Jeffrey, E.; Jiang, Z.; Kafri, D.; Kechedzhi, K.; Kelly, J.; Klimov, P. V.; Knysh, S.; Korotkov, A.; Kostritsa, F.; Landhuis, D.; Lindmark, M.; Lucero, E.; Lyakh, D.; Mandrà, S.; McClean, J. R.; McEwen, M.; Megrant, A.; Mi, X.; Michielsen, K.; Mohseni, M.; Mutus, J.; Naaman, O.; Neeley, M.; Neill, C.; Niu, M. Y.; Ostby, E.; Petukhov, A.; Platt, J. C.; Quintana, C.; Rieffel, E. G.; Roushan, P.; Rubin, N. C.; Sank, D.; Satzinger, K. J.; Smelyanskiy, V.; Sung, K. J.; Trevithick, M. D.; Vainsencher, A.; Villalonga, B.; White, T.; Yao, Z. J.; Yeh, P.; Zalcman, A.; Neven, H.; Martinis, J. M. Quantum supremacy using a programmable superconducting processor. *Nature* **2019**, *574*, 505−510.

(8) Gong, M.; Wang, S.; Zha, C.; Chen, M.-C.; Huang, H.-L.; Wu, Y.; Zhu, Q.; Zhao, Y.; Li, S.; Guo, S.; Qian, H.; Ye, Y.; Chen, F.; Ying, C.; Yu, J.; Fan, D.; Wu, D.; Su, H.; Deng, H.; Rong, H.; Zhang, K.; Cao, S.; Lin, J.; Xu, Y.; Sun, L.; Guo, C.; Li, N.; Liang, F.; Bastidas, V. M.; Nemoto, K.; Munro, W. J.; Huo, Y.-H.; Lu, C.-Y.; Peng, C.-Z.; Zhu, X.; Pan, J.-W. Quantum walks on a programmable two-dimensional 62-qubit superconducting processor. *Science* **2021**, *372*, 948−952.

(9) Xu, S.; Sun, Z.-Z.; Wang, K.; Xiang, L.; Bao, Z.; Zhu, Z.; Shen, F.; Song, Z.; Zhang, P.; Ren, W.; Zhang, X.; Dong, H.; Deng, J.; Chen, J.; Wu, Y.; Tan, Z.; Gao, Y.; Jin, F.; Zhu, X.; Zhang, C.; Wang, N.; Zou, Y.; Zhong, J.; Zhang, A.; Li, W.; Jiang, W.; Yu, L.-W.; Yao, Y.; Wang, Z.; Li, H.; Guo, Q.; Song, C.; Wang, H.; Deng, D.-L. Digital Simulation of Projective Non-Abelian Anyons with 68 Superconducting Qubits. *Chin. Phys. Lett.* **2023**, *40*, 60301.

(10) Google Quantum, A. I. Suppressing quantum errors by scaling a surface code logical qubit. *Nature* **2023**, *614*, 676−681.

(11) Peruzzo, A.; McClean, J.; Shadbolt, P.; Yung, M.-H.; Zhou, X.-Q.; Love, P. J.; Aspuru-Guzik, A.; O'brien, J. L. A variational eigenvalue solver on a photonic quantum processor. *Nat. Commun.* **2014**, *5*, 4213.

(12) O'Malley, P. J. J.; Babbush, R.; Kivlichan, I. D.; Romero, J.; McClean, J. R.; Barends, R.; Kelly, J.; Roushan, P.; Tranter, A.; Ding, N.; Campbell, B.; Chen, Y.; Chen, Z.; Chiaro, B.; Dunsworth, A.; Fowler, A. G.; Jeffrey, E.; Lucero, E.; Megrant, A.; Mutus, J. Y.; Neeley, M.; Neill, C.; Quintana, C.; Sank, D.; Vainsencher, A.; Wenner, J.; White, T. C.; Coveney, P. V.; Love, P. J.; Neven, H.; Aspuru-Guzik, A.; Martinis, J. M. Scalable quantum simulation of molecular energies. *Phys. Rev. X.* **2016**, *6*, 031007.

(13) Cerezo, M.; Arrasmith, A.; Babbush, R.; Benjamin, S. C.; Endo, S.; Fujii, K.; McClean, J. R.; Mitarai, K.; Yuan, X.; Cincio, L.; Coles, P. J. Variational quantum algorithms. *Nat. Rev. Phys.* **2021**, *3*, 625−644.

(14) Tilly, J.; Chen, H.; Cao, S.; Picozzi, D.; Setia, K.; Li, Y.; Grant, E.; Wossnig, L.; Rungger, I.; Booth, G. H.; Tennyson, J. The variational quantum eigensolver: a review of methods and best practices. *Phys. Rep.* **2022**, *986*, 1−128.

(15) Aspuru-Guzik, A.; Dutoi, A. D.; Love, P. J.; Head-Gordon, M. Simulated quantum computation of molecular energies. *Science* **2005**, *309*, 1704−1707.

(16) Lee, S.; Lee, J.; Zhai, H.; Tong, Y.; Dalzell, A. M.; Kumar, A.; Helms, P.; Gray, J.; Cui, Z.-H.; Liu, W.; Kastoryano, M.; Babbush, R.; Preskill, J.; Reichman, D. R.; Campbell, E. T.; Valeev, E. F.; Lin, L.; Chan, G. K.-L. Evaluating the evidence for exponential quantum advantage in ground-state quantum chemistry. *Nature Commun.* **2023**, *14*, 1952.

(17) Bartlett, R. J.; Kucharski, S. A.; Noga, J. Alternative coupled-cluster ansätze II. The unitary coupled-cluster method. *Chem. Phys. Lett.* **1989**, *155*, 133−140.

(18) Kutzelnigg, W. Error analysis and improvements of coupled-cluster theory. *Theor. Chim. Acta* **1991**, *80*, 349−386.

(19) Anand, A.; Schleich, P.; Alperin-Lea, S.; Jensen, P. W.; Sim, S.; Díaz-Tinoco, M.; Kottmann, J. S.; Degroote, M.; Izmaylov, A. F.; Aspuru-Guzik, A. A quantum computing view on unitary coupled cluster theory. *Chem. Soc. Rev.* **2022**, *51*, 1659.

(20) Lee, J.; Huggins, W. J.; Head-Gordon, M.; Whaley, K. B. Generalized unitary coupled cluster wave functions for quantum computation. *J. Chem. Theory and Comput.* **2019**, *15*, 311−324.

(21) Grimsley, H. R.; Economou, S. E.; Barnes, E.; Mayhall, N. J. An adaptive variational algorithm for exact molecular simulations on a quantum computer. *Nat. Commun.* **2019**, *10*, 1−9.

(22) Elfving, V. E.; Millaruelo, M.; Gámez, J. A.; Gogolin, C. Simulating quantum chemistry in the seniority-zero space on qubit-based quantum computers. *Phys. Rev. A* **2021**, *103*, 032605.

(23) Kandala, A.; Mezzacapo, A.; Temme, K.; Takita, M.; Brink, M.; Chow, J. M.; Gambetta, J. M. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature* **2017**, *549*, 242−246.

(24) Ollitrault, P. J.; Miessen, A.; Tavernelli, I. Molecular quantum dynamics: A quantum computing perspective. *Acc. Chem. Res.* **2021**, *54*, 4229−4238.

(25) Li, Y.; Benjamin, S. C. Efficient variational quantum simulator incorporating active error minimization. *Phys. Rev. X* **2017**, *7*, 021050.

(26) Yuan, X.; Endo, S.; Zhao, Q.; Li, Y.; Benjamin, S. C. Theory of variational quantum simulation. *Quantum* **2019**, *3*, 191.

(27) Colless, J. I.; Ramasesh, V. V.; Dahlen, D.; Blok, M. S.; Kimchi-Schwartz, M. E.; McClean, J. R.; Carter, J.; de Jong, W. A.; Siddiqi, I. Computation of molecular spectra on a quantum processor with an error-resilient algorithm. *Phys. Rev. X.* **2018**, *8*, 011021.

(28) Kandala, A.; Temme, K.; Córcoles, A. D.; Mezzacapo, A.; Chow, J. M.; Gambetta, J. M. Error mitigation extends the computational reach of a noisy quantum processor. *Nature* **2019**, *567*, 491−495.

(29) Rice, J. E.; Gujarati, T. P.; Motta, M.; Takeshita, T. Y.; Lee, E.; Latone, J. A.; Garcia, J. M. Quantum computation of dominant products in lithium−sulfur batteries. *J. Chem. Phys.* **2021**, *154*, 134115.

(30) Gao, Q.; Jones, G. O.; Motta, M.; Sugawara, M.; Watanabe, H. C.; Kobayashi, T.; Watanabe, E.; Ohnishi, Y.-y.; Nakamura, H.; Yamamoto, N. Applications of quantum computing for investigations of electronic transitions in phenylsulfonyl-carbazole TADF emitters. *npj Comput. Mater.* **2021**, *7*, 70.

(31) Kirsopp, J. J.; Di Paola, C.; Manrique, D. Z.; Krompiec, M.; Greene-Diniz, G.; Guba, W.; Meyder, A.; Wolf, D.; Strahm, M.; Muñoz Ramo, D. Quantum computational quantification of protein-ligand interactions. *Int. J. Quantum Chem.* **2022**, *122*, No. e26975.

(32) Mitarai, K.; Negoro, M.; Kitagawa, M.; Fujii, K. Quantum circuit learning. *Phys. Rev. A* **2018**, *98*, 032309.

(33) Schuld, M.; Bergholm, V.; Gogolin, C.; Izaac, J.; Killoran, N. Evaluating analytic gradients on quantum hardware. *Phys. Rev. A* **2019**, *99*, 032331.

(34) Zhang, S.-X.; Allcock, J.; Wan, Z.-Q.; Liu, S.; Sun, J.; Yu, H.; Yang, X.-H.; Qiu, J.; Ye, Z.; Chen, Y.-Q.; Lee, C.-K.; Zheng, Y.-C.; Jian, S.-K.; Yao, H.; Hsieh, C.-Y.; Zhang, S. Tensorcircuit: A quantum software framework for the NISQ era. *Quantum* **2023**, *7*, 912.

(35) Chen, J.; Cheng, H.-P.; Freericks, J. K. Quantum-inspired algorithm for the factorized form of unitary coupled cluster theory. *J. Chem. Theory and Comput.* **2021**, *17*, 841−847.
(36) Kottmann, J. S.; Anand, A.; Aspuru-Guzik, A. A feasible approach for automatically differentiable unitary coupled-cluster on quantum computers. *Chem. Sci.* **2021**, *12*, 3497−3508.
(37) Rubin, N. C.; Gunst, K.; White, A.; Freitag, L.; Throssell, K.; Chan, G. K.-L.; Babbush, R.; Shiozaki, T. The Fermionic Quantum Emulator. *Quantum* **2021**, *5*, 568.
(38) Lee, C.-K.; Hsieh, C.-Y.; Zhang, S.; Shi, L. Simulation of Condensed-Phase Spectroscopy with Near-Term Digital Quantum Computers. *J. Chem. Theory and Comput.* **2021**, *17*, 7178−7186.
(39) Lee, C.-K.; Zhong Lau, J. W.; Shi, L.; Kwek, L. C. Simulating energy transfer in molecular systems with digital quantum computers. *J. Chem. Theory and Comput.* **2022**, *18*, 1347−1358.
(40) Ollitrault, P. J.; Mazzola, G.; Tavernelli, I. Nonadiabatic molecular quantum dynamics with quantum computers. *Phys. Rev. Lett.* **2020**, *125*, 260511.
(41) Lee, C.-K.; Hsieh, C.-Y.; Zhang, S.; Shi, L. Variational Quantum Simulation of Chemical Dynamics with Quantum Computers. *J. Chem. Theory and Comput.* **2022**, *18*, 2105−2113.
(42) Jordan, P.; Wigner, E. Über das Paulische Äquivalenzverbot. *Zeitschrift für Physik* **1928**, *47*, 631−651.
(43) Bravyi, S. B.; Kitaev, A. Y. Fermionic quantum computation. *Ann. Phys.* **2002**, *298*, 210−226.
(44) Seeley, J. T.; Richard, M. J.; Love, P. J. The Bravyi-Kitaev transformation for quantum computation of electronic structure. *J. Chem. Phys.* **2012**, *137*, 224109.
(45) Sawaya, N. P.; Menke, T.; Kyaw, T. H.; Johri, S.; Aspuru-Guzik, A.; Guerreschi, G. G. Resource-efficient digital quantum simulation of *d*-level systems for photonic, vibrational, and spin-*s* Hamiltonians. *npj Quantum Inf* **2020**, *6*, 1−13.
(46) Nam, Y.; Chen, J.-S.; Pisenti, N. C.; Wright, K.; Delaney, C.; Maslov, D.; Brown, K. R.; Allen, S.; Amini, J. M.; Apisdorf, J.; Beck, K. M.; Blinov, A.; Chaplin, V.; Chmielewski, M.; Collins, C.; Debnath, S.; Hudek, K. M.; Ducore, A. M.; Keesan, M.; Kreikemeier, S. M.; Mizrahi, J.; Solomon, P.; Williams, M.; Wong-Campos, J. D.; Moehring, D.; Monroe, C.; Kim, J. Ground-state energy estimation of the water molecule on a trapped-ion quantum computer. *npj Quantum Inf* **2020**, *6*, 33.
(47) Li, W.; Huang, Z.; Cao, C.; Huang, Y.; Shuai, Z.; Sun, X.; Sun, J.; Yuan, X.; Lv, D. Toward practical quantum embedding simulation of realistic chemical systems on near-term quantum computers. *Chem. Sci.* **2022**, *13*, 8953−8962.
(48) Guo, S.; Sun, J.; Qian, H.; Gong, M.; Zhang, Y.; Chen, F.; Ye, Y.; Wu, Y.; Cao, S.; Liu, K.; Zha, C.; Ying, C.; Zhu, Q.; Huang, H.-L.; Zhao, Y.; Li, S.; Wang, S.; Yu, J.; Fan, D.; Wu, D.; Su, H.; Deng, H.; Rong, H.; Li, Y.; Zhang, K.; Chung, T.-H.; Liang, F.; Lin, J.; Xu, Y.; Sun, L.; Guo, C.; Li, N.; Huo, Y.-H.; Peng, C.-Z.; Lu, C.-Y.; Yuan, X.; Zhu, X.; Pan, J.-W. Experimental quantum computational chemistry with optimized unitary coupled cluster ansatz. *arXiv* **2022**, No. 2212.08006.
(49) Wecker, D.; Hastings, M. B.; Troyer, M. Progress towards practical quantum variational algorithms. *Phys. Rev. A* **2015**, *92*, 042303.
(50) Miessen, A.; Ollitrault, P. J.; Tavernelli, I. Quantum algorithms for quantum dynamics: a performance study on the spin-boson model. *Phys. Rev. Res.* **2021**, *3*, 043212.
(51) Yao, Y.-X.; Gomes, N.; Zhang, F.; Wang, C.-Z.; Ho, K.-M.; Iadecola, T.; Orth, P. P. Adaptive variational quantum dynamics simulations. *PRX Quantum* **2021**, *2*, 030307.
(52) Broeckhove, J.; Lathouwers, L.; Kesteloot, E.; Van Leuven, P. On the equivalence of time-dependent variational principles. *Chem. Phys. Lett.* **1988**, *149*, 547−550.
(53) Qiskit Contributors. *Qiskit: An Open-source Framework for Quantum Computing*; IBM, 2023.
(54) Bergholm, V.; Izaac, J.; Schuld, M.; Gogolin, C.; Ahmed, S.; Ajith, V.; Sohaib Alam, M.; Alonso-Linaje, G.; AkashNarayanan, B.; Asadi, A.; Arrazola, J. M.; Azad, U.; Banning, S.; Blank, C.; Bromley, T. R.; Cordier, B. A.; Ceroni, J.; Delgado, A.; Di Matteo, O.; Dusko, A.; Garg, T.; Guala, D.; Hayes, A.; Hill, R.; Ijaz, A.; Isacsson, T.; Ittah, D.; Jahangiri, S.; Jain, P.; Jiang, E.; Khandelwal, A.; Kottmann, K.; Lang, R. A.; Lee, C.; Loke, T.; Lowe, A.; McKiernan, K.; Meyer, J. J.; Montañez-Barrera, J. A.; Moyard, R.; Niu, Z.; O'Riordan, L. J.; Oud, S.; Panigrahi, A.; Park, C.-Y.; Polatajko, D.; Quesada, N.; Roberts, C.; Sá, N.; Schoch, I.; Shi, B.; Shu, S.; Sim, S.; Singh, A.; Strandberg, I.; Soni, J.; Száva, A.; Thabet, S.; Vargas-Hernández, R. A.; Vincent, T.; Vitucci, N.; Weber, M.; Wierichs, D.; Wiersema, R.; Willmann, M.; Wong, V.; Zhang, S.; Killoran, N. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *ArXiv* **2018**, No. 1811.04968.v4.
(55) *MindQuantum Developer, MindQuantum*, version 0.6.0. 2021; https://gitee.com/mindspore/mindquantum (accessed May 29, 2023).
(56) Kottmann, J. S.; Alperin-Lea, S.; Tamayo-Mendoza, T.; Cervera-Lierta, A.; Lavigne, C.; Yen, T.-C.; Verteletskyi, V.; Schleich, P.; Anand, A.; Degroote, M.; Chaney, S.; Kesibi, M.; Curnow, N. G.; Solo, B.; Tsilimigkounakis, G.; Zendejas-Morales, C.; Izmaylov, A. F.; Aspuru-Guzik, A. Tequila: A platform for rapid development of quantum algorithms. *Quantum Sci. Technol.* **2021**, *6*, 024009.
(57) Fan, Y.; Liu, J.; Zeng, X.; Xu, Z.; Shang, H.; Li, Z.; Yang, J. Q^2 Chemistry: A quantum computation platform for quantum chemistry. *arXiv* **2022**, No. 2208.10978.
(58) Stair, N. H.; Evangelista, F. A. QForte: An efficient state-vector emulator and quantum algorithms library for molecular electronic structure. *J. Chem. Theory and Comput.* **2022**, *18*, 1555−1568.
(59) Cao, C.; Hu, J.; Zhang, W.; Xu, X.; Chen, D.; Yu, F.; Li, J.; Hu, H.-S.; Lv, D.; Yung, M.-H. Progress toward larger molecular simulation on a quantum computer: Simulating a system with up to 28 qubits accelerated by point-group symmetry. *Phys. Rev. A* **2022**, *105*, 062452.
(60) Schollwöck, U. The density-matrix renormalization group in the age of matrix product states. *Ann. Phys.* **2011**, *326*, 96−192.
(61) Shang, H.; Fan, Y.; Shen, L.; Guo, C.; Liu, J.; Duan, X.; Li, F.; Li, Z. Towards practical and massively parallel quantum computing emulation for quantum chemistry. *npj Quantum Inf* **2023**, *9*, 33.
(62) Ollitrault, P. J.; Baiardi, A.; Reiher, M.; Tavernelli, I. Hardware efficient quantum algorithms for vibrational structure calculations. *Chem. Sci.* **2020**, *11*, 6842−6855.
(63) Bradbury, J.; Frostig, R.; Hawkins, P.; Johnson, M. J.; Leary, C.; Maclaurin, D.; Necula, G.; Paszke, A.; VanderPlas, J.; Wanderman-Milne, S.; Zhang, Q. *JAX: composable transformations of Python+NumPy programs*. 2018; http://github.com/google/jax (accessed May 29, 2023).
(64) Sun, Q.; Berkelbach, T. C.; Blunt, N. S.; Booth, G. H.; Guo, S.; Li, Z.; Liu, J.; McClain, J.; Sayfutyarova, E. R.; Sharma, S.; Wouters, S.; Chan, G. K.-L. PySCF: the Python-based simulations of chemistry framework. *Wiley Interdiscip. Rev. Comput. Mol. Sci.* **2018**, *8*, No. e1340.
(65) Luo, X.-Z.; Liu, J.-G.; Zhang, P.; Wang, L. Yao.jl: Extensible, efficient framework for quantum algorithm design. *Quantum* **2020**, *4*, 341.
(66) Ren, J.; Li, W.; Jiang, T.; Wang, Y.; Shuai, Z. *Renormalizer Package*. https://github.com/shuaigroup/Renormalizer (accessed May 29, 2023).
(67) McClean, J. R.; Rubin, N. C.; Sung, K. J.; Kivlichan, I. D.; Bonet-Monroig, X.; Cao, Y.; Dai, C.; Fried, E. S.; Gidney, C.; Gimby, B.; Gokhale, P.; Häner, T.; Hardikar, T.; Havlíček, V.; Higgott, O.; Huang, C.; Izaac, J.; Jiang, Z.; Liu, X.; McArdle, S.; Neeley, M.; O'Brien, T.; O'Gorman, B.; Ozfidan, I.; Radin, M. D.; Romero, J.; Sawaya, N. P. D.; Senjean, B.; Setia, K.; Sim, S.; Steiger, D. S.; Steudtner, M.; Sun, Q.; Sun, W.; Wang, D.; Zhang, F.; Babbush, R. OpenFermion: The electronic structure package for quantum computers. *Quantum Sci. Technol.* **2020**, *5*, 034014.
(68) McArdle, S.; Mayorov, A.; Shan, X.; Benjamin, S.; Yuan, X. Digital quantum simulation of molecular vibrations. *Chem. Sci.* **2019**, *10*, 5725−5735.
(69) Colbert, D. T.; Miller, W. H. A novel discrete variable representation for quantum mechanical reactive scattering via the *S*-matrix Kohn method. *J. Chem. Phys.* **1992**, *96*, 1982−1991.

(70) Li, W.; Ren, J.; Huai, S.; Cai, T.; Shuai, Z.; Zhang, S. Efficient quantum simulation of electron-phonon systems by variational basis state encoder. *Phys. Rev. Res.* **2023**, *5*, 023046.
(71) Evangelista, F. A.; Chan, G. K.-L.; Scuseria, G. E. Exact parameterization of fermionic wave functions via unitary coupled cluster theory. *J. Chem. Phys.* **2019**, *151*, 244112.
(72) McClean, J. R.; Boixo, S.; Smelyanskiy, V. N.; Babbush, R.; Neven, H. Barren plateaus in quantum neural network training landscapes. *Nat. Commun.* **2018**, *9*, 4812.
(73) Choy, B.; Wales, D. J. Molecular Energy Landscapes of Hardware-Efficient Ansatz in Quantum Computing. *J. Chem. Theory and Comput.* **2023**, *19*, 1197−1206.
(74) Gao, Q.; Nakamura, H.; Gujarati, T. P.; Jones, G. O.; Rice, J. E.; Wood, S. P.; Pistoia, M.; Garcia, J. M.; Yamamoto, N. Computational investigations of the lithium superoxide dimer rearrangement on noisy quantum devices. *J. Phys. Chem. A* **2021**, *125*, 1827−1836.
(75) Miháliková, I.; Pivoluska, M.; Plesch, M.; Friák, M.; Nagaj, D.; Šob, M. The Cost of Improving the Precision of the Variational Quantum Eigensolver for Quantum Chemistry. *Nanomaterials* **2022**, *12*, 243.
(76) Markov, I. L.; Shi, Y. Simulating quantum computation by contracting tensor networks. *SIAM J. Comput.* **2008**, *38*, 963−981.
(77) Liu, Y. A.; Liu, X. L.; Li, F. N.; Fu, H.; Yang, Y.; Song, J.; Zhao, P.; Wang, Z.; Peng, D.; Chen, H.; Guo, C.; Huang, H.; Wu, W.; Chen, D. Closing the "quantum supremacy" gap: achieving real-time simulation of a random quantum circuit using a new Sunway supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*; Association for Computing Machinery, 2021; pp 1−12.
(78) Pan, F.; Zhang, P. Simulation of quantum circuits using the big-batch tensor network method. *Phys. Rev. Lett.* **2022**, *128*, 030501.
(79) Noh, K.; Jiang, L.; Fefferman, B. Efficient classical simulation of noisy random quantum circuits in one dimension. *Quantum* **2020**, *4*, 318.
(80) Cheng, S.; Cao, C.; Zhang, C.; Liu, Y.; Hou, S.-Y.; Xu, P.; Zeng, B. Simulating noisy quantum circuits with matrix product density operators. *Phys. Rev. Res.* **2021**, 3, 023005.
(81) Virtanen, P.; Gommers, R.; Oliphant, T. E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; van der Walt, S. J.; Brett, M.; Wilson, J.; Millman, K. J.; Mayorov, N.; Nelson, A. R. J.; Jones, E.; Kern, R.; Larson, E.; Carey, C. J.; Polat, İ.; Feng, Y.; Moore, E. W.; VanderPlas, J.; Laxalde, D.; Perktold, J.; Cimrman, R.; Henriksen, I.; Quintero, E. A.; Harris, C. R.; Archibald, A. M.; Ribeiro, A. H.; Pedregosa, F.; van Mulbregt, P.; et al. SciPy 1.0 Contributors, SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nat. Methods* **2020**, *17*, 261−272.
(82) Yordanov, Y. S.; Arvidsson-Shukur, D. R.; Barnes, C. H. Efficient quantum circuits for quantum computational chemistry. *Phys. Rev. A* **2020**, *102*, 062612.
(83) Magoulas, I.; Evangelista, F. A. CNOT-Efficient Circuits for Arbitrary Rank Many-Body Fermionic and Qubit Excitations. *J. Chem. Theory Comput.* **2023**, *19*, 822−836.
(84) Dunning, T. H., Jr Gaussian basis sets for use in correlated molecular calculations. I. The atoms boron through neon and hydrogen. *J. Chem. Phys.* **1989**, *90*, 1007−1023.
(85) Li, Z.; Liu, X.; Wang, H.; Ashhab, S.; Cui, J.; Chen, H.; Peng, X.; Du, J. Quantum simulation of resonant transitions for solving the eigenproblem of an effective water Hamiltonian. *Phys. Rev. Lett.* **2019**, *122*, 090504.
(86) Ryabinkin, I. G.; Lang, R. A.; Genin, S. N.; Izmaylov, A. F. Iterative qubit coupled cluster approach with efficient screening of generators. *J. Chem. Theory and Comput.* **2020**, *16*, 1055−1063.
(87) Hehre, W. J.; Ditchfield, R.; Pople, J. A. Self-consistent molecular orbital methods. XII. Further extensions of Gaussian-type basis sets for use in molecular orbital studies of organic molecules. *J. Chem. Phys.* **1972**, *56*, 2257−2261.
(88) Hariharan, P. C.; Pople, J. A. The influence of polarization functions on molecular orbital hydrogenation energies. *Theor. Chim. Acta* **1973**, *28*, 213−222.
(89) Hoy, A.; Bunker, P. R. A precise solution of the rotation bending Schrödinger equation for a triatomic molecule with application to the water molecule. *J. Mol. Spectrosc.* **1979**, *74*, 1−8.
(90) Sokolov, I. O.; Barkoutsos, P. K.; Ollitrault, P. J.; Greenberg, D.; Rice, J.; Pistoia, M.; Tavernelli, I. Quantum orbital-optimized unitary coupled cluster methods in the strongly correlated regime: Can quantum algorithms outperform their classical equivalents? *J. Chem. Phys.* **2020**, *152*, 124107.
(91) Knizia, G.; Chan, G. K.-L. Density matrix embedding: A simple alternative to dynamical mean-field theory. *Phys. Rev. Lett.* **2012**, *109*, 186404.
(92) Knizia, G.; Chan, G. K.-L. Density matrix embedding: A strong-coupling quantum embedding theory. *J. Chem. Theory Comput.* **2013**, *9*, 1428−1432.
(93) Motta, M.; Ceperley, D. M.; Chan, G. K.-L.; Gomez, J. A.; Gull, E.; Guo, S.; Jimé nez-Hoyos, C. A.; Lan, T. N.; Li, J.; Ma, F.; Millis, A. J.; Prokof'ev, N. V.; Ray, U.; Scuseria, G. E.; Sorella, S.; Stoudenmire, E. M.; Sun, Q.; Tupitsyn, I. S.; White, S. R.; Zgid, D.; Zhang, S. Towards the solution of the many-electron problem in real materials: Equation of state of the hydrogen chain with state-of-the-art many-body methods. *Phys. Rev. X* **2017**, *7*, 031059.
(94) Mineh, L.; Montanaro, A. Solving the Hubbard model using density matrix embedding theory and the variational quantum eigensolver. *Phys. Rev. B* **2022**, *105*, 125117.
(95) Nielsen, M. A.; Chuang, I. *Quantum computation and quantum information*; Cambridge University Press, 2010.
(96) Leggett, A. J.; Chakravarty, S.; Dorsey, A. T.; Fisher, M. P.; Garg, A.; Zwerger, W. Dynamics of the dissipative two-state system. *Rev. Mod. Phys.* **1987**, *59*, 1.
(97) Shuai, Z.; Li, W.; Ren, J.; Jiang, Y.; Geng, H. Applying Marcus theory to describe the carrier transports in organic semiconductors: Limitations and beyond. *J. Chem. Phys.* **2020**, *153*, 080902.
(98) Kloss, B.; Reichman, D. R.; Tempelaar, R. Multiset matrix product state calculations reveal mobile Franck-Condon excitations under strong Holstein-type coupling. *Phys. Rev. Lett.* **2019**, *123*, 126601.
(99) Nan, G.; Yang, X.; Wang, L.; Shuai, Z.; Zhao, Y. Nuclear tunneling effects of charge transport in rubrene, tetracene, and pentacene. *Phys. Rev. B* **2009**, *79*, 115203.