

QForte: an efficient state-vector emulator and quantum algorithms library for molecular electronic structure

Nicholas H. Stair and Francesco A. Evangelista*

Department of Chemistry and Cherry Emerson Center for Scientific Computation, Emory University, Atlanta, Georgia, 30322, U.S.A.

E-mail: francesco.evangelista@emory.edu

Abstract

We introduce a novel open-source software package QFORTE, a comprehensive development tool for new quantum simulation algorithms. QFORTE incorporates functionality for handling molecular Hamiltonians, fermionic encoding, ansatz construction, time evolution, and state-vector emulation, requiring only a classical electronic structure package as a dependency. QFORTE also contains black-box implementations of a wide variety of quantum algorithms, including variational and projective quantum eigensolvers, adaptive eigensolvers, quantum imaginary time evolution, and quantum Krylov methods. We highlight two features of QFORTE: i) how the Python class structure of QFORTE enables the facile implementation of new algorithms, and ii) how existing algorithms can be executed in just a few lines of code.

1. INTRODUCTION

The past decade has seen tremendous progress in the development of quantum computational hardware, enabling early demonstrations of quantum advantage,¹ and numerous non-trivial applications ranging from quantum simulation²⁻⁷ to constrained optimization.^{8,9} These advances

have concurrently inspired rapid development of numerous quantum algorithms for both noisy intermediate-scale quantum¹⁰ (NISQ) and fault-tolerant devices. In the field of quantum simulation of many-body systems, a variety of methods now exist^{11–16} which vary dramatically in quantum resource requirements, often with a tradeoff between circuit depth and measurement overhead. Unfortunately, new quantum algorithmic developments are rarely accompanied by detailed numerical comparison to existing methods. The infrequency of cross comparison is largely due to the lack of widely accessible reference implementations of quantum algorithms. To help ameliorate this issue, we introduce a new open-source software project, QFORTE, the first black-box quantum algorithms library for molecular electronic structure.

The software ecosystem for the development of new quantum algorithms is ever-expanding, ranging from full-stack industry-backed packages,^{17–21} to task specific open-source projects.^{22–29} This is particularly true in the context of quantum algorithms for molecular electronic structure, because there are many complex software challenges involved in the complete user workflow pipeline (as illustrated in Fig. 1), beginning with specification of a molecular geometry and ending with a numerical prediction of the molecular energy and properties. In order to accomplish this

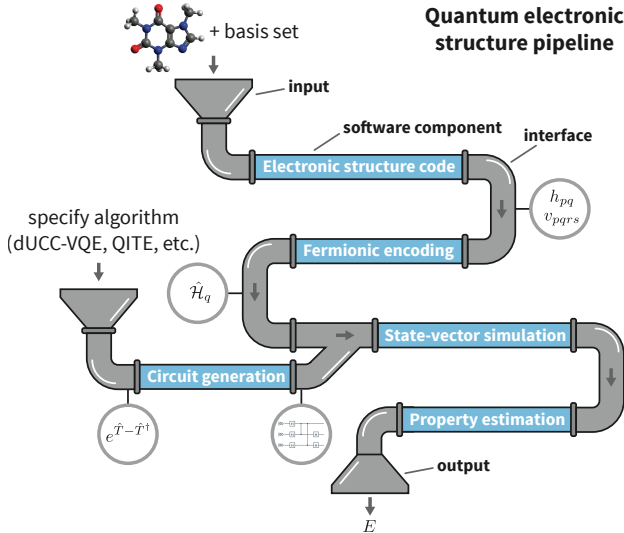


Figure 1: The quantum electronic structure pipeline.

task, one must use a classical electronic structure package (e.g., PYSCF,³⁰ Psi4,³¹ etc.) to obtain the matrix representation of one- and two-body operators (the integrals h_{pq} and v_{pqrs}) that enter in

the second-quantized molecular Hamiltonian

$$\hat{\mathcal{H}} = \sum_{pq} h_{pq} \hat{a}_p^\dagger \hat{a}_q + \frac{1}{4} \sum_{pqrs} v_{pqrs} \hat{a}_p^\dagger \hat{a}_q^\dagger \hat{a}_s \hat{a}_r \quad (1)$$

where \hat{a}^\dagger and \hat{a} are, respectively, fermionic creation and annihilation operators labeled by the index of the spin orbital on which they act. The second quantized Hamiltonian is then represented in the product algebra of Pauli operators ($X \equiv \sigma_1$, $Y \equiv \sigma_2$, $Z \equiv \sigma_3$) using an isospectral mapping often referred to as fermionic encoding. As such, one must utilize a package for fermionic encoding (e.g., OPENFERMION²³), as well as appropriate application programming interfaces (API) (e.g., OPENFERMION-PYSCF or OPENFERMION-PSI4), to map operators (such as the Hamiltonian, $\hat{\mathcal{H}}$) to the so-called qubit representation ($\hat{\mathcal{H}}_q$) given as a linear combination of a given number (N_{PS}) of Pauli strings (products of Pauli operators) \hat{P}_ℓ :

$$\hat{\mathcal{H}} \xrightarrow{\text{fermionic encoding}} \hat{\mathcal{H}}_q = \sum_{\ell}^{N_{\text{PS}}} h_{\ell} \hat{P}_{\ell}. \quad (2)$$

A generic Pauli string is a unique product of n_{ℓ} Pauli operators $\hat{P}_k^{(\ell)}$:

$$\hat{P}_{\ell} = \prod_k^{n_{\ell}} \hat{P}_k^{(\ell)}, \quad (3)$$

where the compound index $k = (p, [X, Y, \text{ or } Z])$ labels a combination of a specific Pauli operator and the qubit (p) on which it acts.

In order to apply quantum circuits associated with encoded operators (such as the time evolution unitary, or various unitary ansätze) one is required to install one of the numerous backend quantum-computer emulators such as those implemented in QISKIT¹⁷ (IBM), CIRQ¹⁸ (Google), QSIM³² (Google), FQE²² (Google/QST), Q#¹⁹ (Microsoft), PYQUILL²⁰ (Rigetti), ProjectQ,²⁹ STRAWBERRYFIELDS²¹ (Xanadu), QULACS,²⁶ QHIPSTER,²⁴ and YAO²⁵ each associated with a distinct API. We note that in addition to software implementations, there have also been numerous advances in algorithmic considerations for state vector simulation/emulation such as those reported in Refs.

33–39.

In the outermost software layer exist packages such as TEQUILA²⁷ that serve as sandbox implementation tools which solve many of the interoperability challenges associated with interfacing the aforementioned dependencies. While flexible packages for sand-box development are undoubtedly important, it is still usually left to the user to implement a desired algorithm, which is generally a non-trivial task given the diversity of quantum algorithms present in modern literature and the challenge of utilizing an inhomogeneous software ecosystem. A single-package black-box implementation of quantum algorithms incorporating all of the elements of the quantum electronic structure pipeline is highly desirable for researchers interested in generating comparative results rapidly.

An additional challenge to black-box implementations of quantum algorithms is their now significant level of diversity. Arguably the simplest and most well established algorithm is the variational quantum eigensolver^{11,12} (VQE). In VQE, the ground state is approximated by a trial state optimized in a procedure that combines a classical optimization algorithm with quantum measurement of the energy and gradients of the trial state. While implementation of a vanilla VQE code is straightforward for toy examples, automatic generation of the ansatz circuit is generally more complicated and obviously dependent on the specific form of the ansatz (i.e. disentangled unitary coupled cluster,^{12,40,41} hardware-efficient,³ Hamiltonian variational,⁴² qubit coupled cluster,⁴³ to name a few). Moreover, many hybrid approaches such as adaptive ansatz approaches^{44,45} and subspace expansion methods⁴⁶ incorporate the basic VQE schema as a subroutine and require additional implementation for determination of matrix elements to solve a generalized eigenvalue problem and/or an additional algorithmic layer to extend the ansatz. Similar implementation challenges exist for algorithms that rely on (often controlled) Hamiltonian time evolution such as quantum phase estimation^{47–49} (QPE), or time-evolved subspace methods.^{50–52} For algorithms that measure projected quantities such as quantum imaginary time evolution¹³ (QITE), the projective quantum eigensolver⁵³ (PQE), and the quantum quantum antihermitian contracted Schrödinger equation (QACSE) solver,^{16,54–56} one must additionally implement the (often iterative) parameter

update procedure.

Our new open-source package QFORTE is an end-to-end electronic structure package for quantum algorithms, and is still capable of facilitating sand-box implementations of new algorithms, only relying on a classical electronic structure package as a dependency. The remainder of this article is organized as follows. In Sec. 2 we will describe key component classes in QFORTE and its interface to Psi4. In Sec. 3 will discuss each of the quantum algorithms currently implemented in QFORTE as well as some of their implementation details in terms of the key components. In Sec. 4 we discuss representative timings for critical subroutines such as determining Hamiltonian expectation values. Finally, in Sec. 5 we demonstrate an example of how QFORTE can (i) be used to implement a new quantum algorithm and (ii) be used to compare new algorithms to the ones already implemented in its library.

2. OVERVIEW OF THE STRUCTURE OF QFORTE

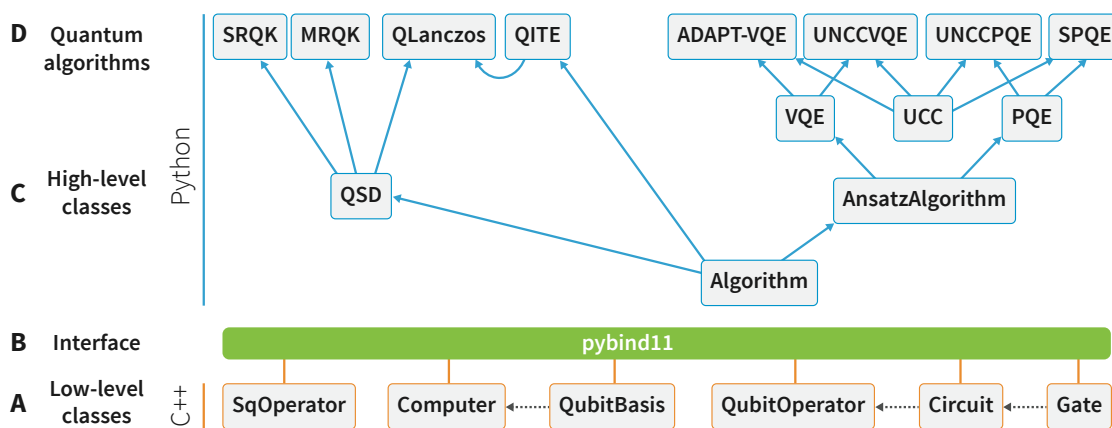


Figure 2: The structure of the QFORTE package. Solid lines indicate inheritance (pointing from the base to the derived class) while dashed lines indicate use of objects from another class (pointing from the class used to the one using it). A) Low-level C++ classes are used to store and manipulate a state vector. B) The C++ classes are exposed to Python via the PYBIND11 library. C) A high-level Python classes implement components of quantum algorithms. D) Quantum algorithms available to the user are implemented using classes from the high-level API.

The main software components of QFORTE are illustrated in Fig. 2. The lowest level contains components that require efficient execution, including the quantum computer (state-vector) emulator (Computer) and quantum circuits (Circuit). These are implemented as classes in C++ and exposed in Python via PYBIND11.⁵⁷ Higher-level components such as the SystemFactory

class which interfaces QFORTE to classical electronic structure packages, and the subclasses that implement algorithms are all written in Python and use the PYBIND11 interface to the lower-level components. Here we will give an overview of some of the most important components of QFORTE.

2.1 The quantum computer emulator. One core feature of QFORTE is a dedicated quantum computer emulator.

State-vector quantum computer emulators store and manipulate a classical representation of the full quantum state realized on quantum hardware.

The QubitBasis class. The state-vector emulator in QFORTE makes heavy use of an elementary C++ `QubitBasis` class used to represent an element of a n_{qb} qubit Fock space basis $\mathcal{F} = \{|q_0\rangle \otimes \cdots \otimes |q_{n_{\text{qb}}-1}\rangle\}$. Each instance of `QubitBasis` represents an element of \mathcal{F} , and is characterized by a 64-bit unsigned integer such that each bit represents the binary state of a qubit:

$$\underbrace{q_{63} \cdots q_1 q_0}_{\text{64-bit unsigned integer}} \equiv \underbrace{|q_0\rangle \otimes \cdots \otimes |q_{63}\rangle}_{\text{element of the Fock-space basis}}, \quad \text{with } q_i \in \{0, 1\} \quad (4)$$

Because application of quantum gates to any basis element results in new elements with target qubit(s) flipped, the unsigned integer representation is beneficial as it allows for very efficient bitwise operations. The binary number corresponding to an element of the Fock-space basis also offers a convenient way to map the qubit multi-index $q_0 q_1 \cdots$ to an integer (address).

The QuantumComputer class. The backbone of the state-vector emulator in QFORTE is the `Computer` class, which, for a given number of qubits (n_{qb}) stores a state vector of the form

$$|\Psi_q\rangle = \sum_{q_0 \cdots q_{n_{\text{qb}}-1}}^{\{0,1\}} C_{q_0, \dots, q_{n_{\text{qb}}-1}} |q_0\rangle \otimes \cdots \otimes |q_{n_{\text{qb}}-1}\rangle \quad (5)$$

By default, a `Computer` object is initialized in the state $|\Psi_q\rangle = |0\rangle \otimes \cdots \otimes |0\rangle$. The `Computer` class is comprised of a complex vector to store $C_{q_0, \dots, q_{n_{\text{qb}}-1}}$, as well as a vector of `QubitBasis` objects

(both of dimension $2^{n_{\text{qb}}}$ where n_{qb} is the number of qubits). An example of how to instantiate a `Computer` with four qubits is shown in Lst. 1.

```
1 import qforte as qf
2 nqb = 4
3 qcomp = qf.Computer(nqb)
```

Listing 1: Initializing a `Computer` object with four qubits.

The Gate class. Once a `Computer` is initialized, the state can be modified by applying gates, encoded in the class `Gate`. The `Gate` class (when used in conjunction with a `Computer`) is the most fundamental building block for all quantum algorithms in QFORTE. Some of the most pertinent gates used in quantum simulation are the Pauli gates (X , Y , and Z), the Hadamard gate H (not to be confused with the Hamiltonian $\hat{\mathcal{H}}$), the controlled NOT [CNOT] gate, and the parametric z rotation gate $R_z(\theta)$. A full list of gates implemented in QFORTE can be found in the documentation.

The `Gate` class has several important attributes including its type, the target (and optionally the control) qubit index, and a matrix of complex values that represents the operator. Instantiating a `Gate` is simply done via the `gate()` member function, as shown in Lst. 2.

```
1 target_idx = 4
2 X_4gate = qf.gate('X', target_idx)
```

Listing 2: Initializing a `Gate` object. Here we instantiate a Pauli X gate that will target the qubit q_4 .

Listing 3 shows a small example of using QFORTE's state-vector emulator to construct the two-qubit Bell state

$$|\Psi_{\text{Bell}}\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle \quad (6)$$

by applying H_0 followed by $\text{CNOT}_{0,1}$ to the state $|00\rangle$. Recall that the action of the Hadamard gate H is:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \quad (7)$$

Recall that the action of the controlled NOT gate [with target qubit q_0 , and control qubit q_1

(CNOT_{0,1})] is:

$$\begin{aligned} \text{CNOT}_{0,1} |00\rangle &= |00\rangle & \text{CNOT}_{0,1} |01\rangle &= |11\rangle \\ \text{CNOT}_{0,1} |10\rangle &= |10\rangle & \text{CNOT}_{0,1} |11\rangle &= |01\rangle \end{aligned} \tag{8}$$

```
1 # Initialize a two-qubit QuantumComputer.
2 nqb = 2
3 qbell = qf.Computer(nqb)
4
5 # Initialize the gates needed to build the Bell state.
6 H_0 = qf.gate('H', 0)
7 CNOT_0_1 = qf.gate('CNOT', 1, 0)
8
9 # Apply the Hadamard gate.
10 qbell.apply_gate(H_0)
11
12 # Apply the CNOT gate.
13 qbell.apply_gate(CNOT_0_1)
```

Listing 3: Creating a Bell state using a circuit that applies the unitary CNOT_{0,1}H₀ to the state |00⟩.

In QFORTE, the action of a gate on a quantum state (represented by a `Computer` object) is implemented with algorithms specialized for different gates. For example, in Fig. 3 we illustrate how the X_2 operator is applied to a 3-qubit state. Since X_2 modifies only the first bit (by flipping it), the operation $X_2 |\Psi\rangle \rightarrow |\Psi'\rangle$ can be implemented with a low-level copy operation that realizes the following action on the coefficients in a `Computer` object: $C'_{q_0q_1q_2} = C_{q_0q_1(1-q_2)}$. Since the operation is performed on continuous sections of the state vector, the effect of cache misses is minimized and the operation can be easily vectorized on multi-core architectures. The same principle illustrated here can be applied to more complex one-qubit gates such as parameterized rotations and two-qubit gates such as CNOT.

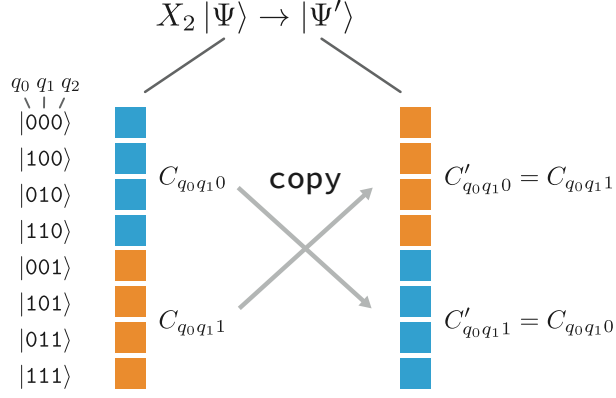


Figure 3: Example of how gate operations are implemented in QFORTE. In this example, the X_2 gate is applied to a general state Ψ producing a new state Ψ' . The action of X_2 is implemented with a low-level C++ copy operation to two separate portions of the state vector.

The Circuit class. In virtually any quantum algorithm it is necessary to apply many gates sequentially. A so-called quantum circuit, commonly referred to as a unitary (\hat{U}), is represented by a product of quantum gates, making the overall circuit itself a unitary operation. The `Circuit` class operates at one level above the `Gate` class and its primary attribute is a vector of `Gate` objects.

Although any product of elementary gates technically constitutes a circuit, one of the most important circuit structures in quantum simulation is that which represents unitaries obtained by exponentiating a Pauli string \hat{P}_ℓ [as defined in in Eq. (3)]:

$$e^{i\theta_\ell \hat{P}_\ell}. \quad (9)$$

The function `exp_pauli_string` in QFORTE is responsible for compiling Eq. (9) into a circuit containing one- and two-qubit gates. An example of how one would construct such a circuit in QFORTE for the operator $\exp(-i0.5X_2Z_1Z_0)$ is shown in Lst. 4, while in Fig. 4 we report the corresponding quantum circuit generated by the function `exp_pauli_string`. This algorithm follows a standard approach^{11,12,58} of using operator identities (e.g., like $X = HZH$) to express the

starting expression in terms of the exponential of products of Z operators only, namely

$$\begin{aligned} \exp(-i0.5 X_2 Z_1 Z_0) &= \exp(-i0.5 H_2 Z_2 H_2 Z_1 Z_0) \\ &= H_2 \exp(-i0.5 Z_2 Z_1 Z_0) H_2 \end{aligned} \quad (10)$$

where one uses the fact that H_2 is its own inverse ($H_2 H_2 = 1$) and that it commutes with Z_0 and Z_1 . Next, the term $\exp(-i0.5 Z_2 Z_1 Z_0)$, responsible for the fermionic sign, is implemented as a cascade of CNOT gates, a z-axis rotation of $2\theta_\ell$, and the inverse of the CNOT cascade:

$$e^{-i0.5 Z_2 Z_1 Z_0} = \text{CNOT}_{0,1} \text{CNOT}_{1,2} R_z(1.0) \text{CNOT}_{1,2} \text{CNOT}_{0,1} \quad (11)$$

```

1 # Construct the desired preliminary circuit (X2 Z1 Z0)
2 circ = qf.Circuit()
3 circ.add_gate(qf.gate('Z', 0))
4 circ.add_gate(qf.gate('Z', 1))
5 circ.add_gate(qf.gate('X', 2))
6
7 # Define the factor = -i theta
8 factor = -0.5j
9
10 # Construct the unitary for the exponential.
11 Uexp, phase = exp_pauli_string(factor, circ)

```

Listing 4: Constructing the circuit corresponding to the operator $\exp(-i0.5 X_2 Z_1 Z_0)$.

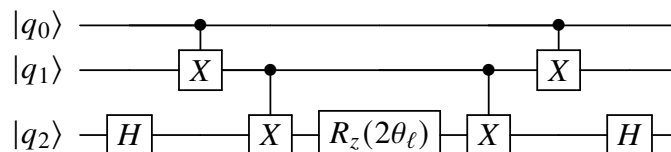


Figure 4: Circuit diagram for exponential unitary operators commonly used in many quantum simulation algorithms. This example considers operators of the form $\exp(-i\theta_\ell X_2 Z_1 Z_0)$.

2.2 The QubitOperator and SQOperator classes. The outer-most operations class in QForte is the QubitOperator class. This class represents operators \hat{O} that are linear combinations of N_ℓ unitaries (\hat{U}_ℓ) as

$$\hat{O} = \sum_{\ell} u_{\ell} \hat{U}_{\ell} \quad (12)$$

where u_{ℓ} is a complex coefficient. The key attribute of the QubitOperator class is a vector of pairs containing a complex coefficient and a Circuit object. In QFORTE's implementation, the QubitOperator class is used to represent objects such as the Hamiltonian $\hat{\mathcal{H}}_q$ or the cluster operator⁵⁹ \hat{T}_q (in their qubit representations). It is important to note that although it is always possible in QFORTE to apply a QubitOperator to a state, unless \hat{O} is unitary, this operation cannot be realized physically on a quantum device.

The ability to evaluate expectation values of the form $\langle \hat{O} \rangle = \langle \Psi_q | \hat{O} | \Psi_q \rangle$ is also paramount to many quantum algorithms. Examples include the energy and energy gradients for VQE, as well as projected quantities such as off-diagonal matrix elements in QSD techniques, or residuals in PQE. In QFORTE this is accomplished easily by passing the operator to an expectation value function. Listing 5 for example shows how one would measure the expectation value of the operator $\hat{O} = X_0 + \hat{X}_1$ with respect to the Bell state (constructed in Lst. 3). We note that it is also possible to determine approximate expectation values in QFORTE based on a user specified number of measurements.

```

1 # Initialize the operator
2 O = QubitOperator()
3 O.add_term(1.0, build_circuit('X_0'))
4 O.add_term(1.0, build_circuit('X_1'))
5
6 # Get the expectation value.
7 exp_val = qbell.expectation(O)

```

Listing 5: Measuring expectation values for QubitOperators.

QFORTE also supports operators in the form of second quantization, that is, operators comprised of fermionic annihilation and creation operators. The SQOperator class functions very similarly

to the `QubitOperator` class, but utilizes a slightly different syntax. We note that second quantized operators in `QFORTE` always assume that the individual fermionic operators are normal ordered (creation operators appear to the left of annihilation operators) within a term. The second quantized operators can then be transformed to the `QubitOperator` representation (given as a linear combination of products of Pauli operators) via the Jordan-Wigner (JW) transformation.⁶⁰ Other common fermionic encodings⁶¹⁻⁶⁴ are not implemented natively in `QFORTE` and will be added in future releases. Under the JW transformation, there is a one-to-one mapping between a spin orbital ϕ_p and qubit q_p such that the fermionic annihilation (\hat{a}_p) and creation (\hat{a}_p^\dagger) operators are represented by combinations of Pauli strings

$$\hat{a}_p = \frac{1}{2}(X_p + iY_p)Z_{p-1} \dots Z_0 \quad (13)$$

and,

$$\hat{a}_p^\dagger = \frac{1}{2}(X_p - iY_p)Z_{p-1} \dots Z_0 \quad (14)$$

Listing 6 shows how to instantiate a `SQOperator` with the operator $0.5 \hat{a}_1^\dagger \hat{a}_2 - 0.25 \hat{a}_4^\dagger \hat{a}_2^\dagger \hat{a}_3 \hat{a}_1$ and transform it into a `QubitOperator`.

```

1 # Initialize the second quantized operator.
2 sq_op = qf.SQOperator()
3
4 # Construct the terms and add them to the list.
5 h1 = 0.5
6 h2 = -0.25j
7 sq_op.add_term(h1, [1], [2])
8 sq_op.add_term(h2, [4, 2], [3, 1])
9
10 # Transform to the qubit operator representation.
11 pauli_op = sq_op.jw_transform()

```

Listing 6: Converting `SQOperators` into `QuantumOperators`. This example constructs the operator $0.5 \hat{a}_1^\dagger \hat{a}_2 - 0.25 \hat{a}_4^\dagger \hat{a}_2^\dagger \hat{a}_3 \hat{a}_1$.

2.3 The molecule class. As discussed in the introduction, the first step in a quantum electronic structure computation is to obtain the molecular Hamiltonian in the qubit operator representation, based on a specified molecular geometry. In QFORTE this is all accomplished using the `system_factory` and `molecule` classes. To begin, as shown in Lst. 7, one simply imports the appropriate modules and specifies a geometry (in this case as a list of element symbols and xyz coordinates). Once the molecule class has been populated, the user has access to the molecular Hamiltonian both in its second-quantization representation (as a `SQOperator`) and in a qubit representation (as a `QubitOperator`) resulting from the Jordan-Wigner transformation. The molecule object is a key data structure in QFORTE that is passed to all algorithms to perform a quantum computation.

```
1 from qforte import *
2
3 # Define the molecular geometry.
4 geom = [('H', (0., 0., 0.)), ('H', (0., 0., 0.75))]
5
6 # Instantiate the system_factory object (also populates the integrals).
7 factory = system_factory(build_type='psi4', mol_geometry=geom, basis='sto-3g')
8
9 # Get the molecule object.
10 H2mol = factory.get_molecule()
```

Listing 7: Initializing the QFORTE molecule object.

3. ALGORITHMS IMPLEMENTED IN QFORTE

Quantum algorithms are implemented in QFORTE in the Python layer, and are built by composing abstract base classes (ABCs) that implement algorithms (and optionally) mixin classes that encode specific ansätze. For example, the lowest-level ABC in QFORTE is `Algorithm`, which is responsible for defining attributes and abstract functions common to all derived classes. The class `AnsatzAlgorithm`, derived from `Algorithm`, defines additional functionality that builds a quantum circuit for any algorithm that employs a parameterized ansatz. Each distinct algorithm im-

plemented in QFORTE is then defined by its own concrete class that inherits from various appropriate parent classes, as displayed in Fig. 2.

We consider the algorithms implemented in QFORTE partitioned into several overlapping categories. The first is variational hybrid algorithms^{11,12,65} in which a classical optimizer is utilized to minimize the energy expectation value. The second category is projective approaches^{13,53} where projective quantities are measured on a quantum device and then used to directly update or augment a classically parameterized unitary. The third category is quantum subspace diagonalization^{13,46,50,51,66} (QSD) in which a non-orthogonal many body basis is generated from a family of operators, and the matrix elements of a generalized eigenvalue problem are measured on a quantum device. We note that an important class of algorithms that is currently not implemented in QFORTE are those derived from quantum phase estimation^{48,49} where the eigenvalue of an Hamiltonian is estimated via a controlled time evolution of an initial state.

Currently, QFORTE contains black-box implementations of the following algorithms: (i) variational quantum eigensolver^{11,12} (VQE) and (ii) projective quantum eigensolver⁵³ (PQE) both with a disentangled⁶⁷ (factorized) unitary coupled cluster^{68–72} (dUCC) ansatz, (iii) the adaptive derivative-assembled pseudo-trotter (ADAPT)-VQE,⁴⁴ (iv) selected PQE⁵³ (SPQE), (v) a variant of quantum imaginary time evolution¹³ (QITE) and its quantum Lanczos¹³ (QLanczos) extension, (vi) quantum Krylov^{50,51} (QK), and (vii) its selected multireference variant⁵¹ (MRSQK). Each of these algorithms is implemented using the software components described in Sec. 2. In the following subsections we will briefly summarize each of these methodologies.

3.1 Variational quantum eigensolver (VQE) based on dUCC trial states. In the general VQE schema, one uses a unitary circuit $\hat{U}(\theta)$ parameterized by the parameter vector θ to construct a normalized trial state of the form

$$|\Psi_{\text{VQE}}\rangle = \hat{U}(\theta) |\Phi_0\rangle \quad (15)$$

from an easily prepared reference state $|\Phi_0\rangle$ (such as the Hartree–Fock determinant). One then minimizes the energy expectation value of the trial state

$$E_{\text{VQE}} = \min_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \langle \Phi | \hat{U}^\dagger(\boldsymbol{\theta}) \hat{\mathcal{H}} \hat{U}(\boldsymbol{\theta}) | \Phi \rangle \quad (16)$$

This minimization problem is solved by a classical optimization algorithm that calls a quantum routine to access the energy (and optionally gradients^{73,74}) of the trial state. Energy and gradients may be computed by averaging the expectation value of each term in the qubit Hamiltonian [Eq. (2)], for example, in the case of the energy

$$E(\boldsymbol{\theta}) = \sum_{\ell} h_{\ell} \langle \Phi | \hat{U}^\dagger(\boldsymbol{\theta}) \hat{P}_{\ell} \hat{U}(\boldsymbol{\theta}) | \Phi \rangle \quad (17)$$

QFORTE implements VQE with a disentangled (or factorized) UCC ansatz, $\hat{U}_{\text{dUCC}}(\mathbf{t})$. We assume a single determinant reference state $|\Phi_0\rangle = |\phi_1\phi_2\cdots\rangle$ specified by occupied spin orbitals $\{\phi_i\}$ and unoccupied (virtual) spin orbitals $\{\phi_a\}$. The operator $\hat{\tau}_{\mu} \equiv \hat{\tau}_{ij\dots}^{ab\dots} = \hat{a}_a^\dagger \hat{a}_b^\dagger \cdots \hat{a}_j \hat{a}_i$ is a particle-hole excitation operator that transforms the reference determinant $|\Phi_0\rangle$ into the excited determinant $|\Phi_{\mu}\rangle = \hat{\tau}_{\mu} |\Phi_0\rangle$. The dUCC ansatz is then constructed as a product of exponentiated anti-hermitian operators $\hat{k}_{\mu} \equiv \hat{\tau}_{\mu} - \hat{\tau}_{\mu}^\dagger$ multiplied by the corresponding amplitudes $t_{\mu} = (\mathbf{t})_{\mu}$ as

$$\hat{U}_{\text{dUCC}}(\mathbf{t}) = \prod_{\mu} e^{t_{\mu} \hat{k}_{\mu}} \quad (18)$$

The dUCC circuit is built automatically in QFORTE by initializing a SQOperator list that represent the \hat{k}_{μ} , transforming \hat{k}_{μ} to a QuantumOperator list via the Jordan–Wigner mapping, and constructing the circuit for the exponential of each sub-term [Eq. (9)]. We note that optimization in QFORTE relies on the SciPy scientific computing library.⁷⁵ Any valid SciPy optimizer may be selected when calling the run() function of a VQE algorithm. The ordering of the operators $e^{t_{\mu} \hat{k}_{\mu}}$ entering Eq. (18) is defined by the binary representation of the corresponding determinants $|\Phi_{\mu}\rangle = \hat{\tau}_{\mu} |\Phi_{\text{HF}}\rangle$ in the occupation number representation. The maximum excitation level (single

and double, triple, etc. . .) for the operators in Eq. (18) can be of arbitrary rank and is specified by the user as an option passed to the `run()` function.

3.2 Adaptive VQE. We have also implemented the ADAPT-VQE approach, and adaptive variant of dUCC-VQE.⁴⁴ In ADAPT-VQE, the unitary ansatz at macro-iteration k is defined as the product of exponentials of selected operators

$$\hat{U}_{\text{ADAPT}}^{(k)}(\mathbf{t}^{(k)}) = \prod_{j=1}^k e^{t_{\mu_j}^{(k)} \hat{\kappa}_{\mu_j}} \quad (19)$$

where $\hat{\kappa}_{\mu_j}$ is an operator in the pool \mathcal{P} and $t_{\mu_j}^{(k)}$ its corresponding amplitude. In QFORTE it is possible to construct \mathcal{P} in a variety of ways, including the generalized single and double excitation/de-excitation operators described in the original ADAPT-VQE publication.⁴⁴ To grow the ADAPT ansatz, all possible unitaries with an additional exponential are considered

$$\hat{U}_{\text{ADAPT}}^{(k+1)}(\mathbf{t}^{(k)}, t_\nu) = e^{t_\nu \hat{\kappa}_\nu} \hat{U}_{\text{ADAPT}}^{(k)}(\mathbf{t}^{(k)}) \quad (20)$$

For each operator, the corresponding energy gradient with respect to t_ν is evaluated

$$g_\nu = \langle \Psi_{\text{VQE}} | [\hat{\mathcal{H}}, \hat{\kappa}_\nu] | \Psi_{\text{VQE}} \rangle \quad (21)$$

and the one with the largest gradient magnitude $|g_\nu|$ is selected. In the next macro-iteration, this new operator is appended to ansatz and all the parameters $t_{\mu_j}^{(k+1)}$ are optimized employing the general VQE schema. The procedure is terminated when the norm of the gradient vector for the pool falls below a user-provided threshold that controls the accuracy of the ansatz.

3.3 Projective quantum eigensolver (PQE) based on dUCC trial states. In the dUCC projective quantum eigensolver approach, we consider a trial state of the form $|\Psi\rangle = \hat{U}(\mathbf{t}) |\Phi_0\rangle$, where $\hat{U}(\mathbf{t})$ is defined by Eq. (18) and contains arbitrary particle-hole operators. PQE aims to solve the following unitarily-transformed version of the Schrödinger equation, obtained by left-multiplying the original

one with $\hat{U}^\dagger(\mathbf{t})$,

$$\hat{U}^\dagger(\mathbf{t})\hat{\mathcal{H}}\hat{U}(\mathbf{t})|\Phi_0\rangle = E|\Phi_0\rangle \quad (22)$$

Instead of finding the optimal parameters via variational minimization (as is done in VQE), PQE seeks to minimize the residual condition

$$r_\mu(\mathbf{t}) \equiv \langle \Phi_\mu | \hat{U}^\dagger(\mathbf{t})\hat{\mathcal{H}}\hat{U}(\mathbf{t}) | \Phi_0 \rangle = 0, \text{ for all } \Phi_\mu \in Q \quad (23)$$

where the residual $r_\mu(\mathbf{t})$ is a matrix element of the unitarily-transformed Hamiltonian $\hat{U}^\dagger(\mathbf{t})\hat{\mathcal{H}}\hat{U}(\mathbf{t})$ between the excited determinant Φ_μ and the reference state Φ_0 . In practice, we only consider enforcing the residual condition for a subset Q of excited determinants (e.g., all single and double excitations).

An important feature of dUCC-PQE is that the parameter vector \mathbf{t} can be updated by measuring the residuals via a simple quasi-Newton iteration approach

$$t_\mu^{(n+1)} = t_\mu^{(n)} + \frac{r_\mu^{(n)}}{\Delta_\mu} \quad (24)$$

where the superscript “(n)” indicates the amplitude at iteration n . The quantities Δ_μ are standard Møller–Plesset denominators $\Delta_\mu \equiv \Delta_{ij\dots}^{ab\dots} = \epsilon_i + \epsilon_j + \dots - \epsilon_a - \epsilon_b \dots$ where ϵ_i are Hartree–Fock orbital energies.

The residuals r_μ can be easily determined from symmetric expectation values and can therefore be measured via operator averaging on a quantum device. Specifically, each element r_μ is obtained as a sum of three expectation values

$$\begin{aligned} r_\mu(\mathbf{t}) &= \langle \Omega_\mu | \hat{U}^\dagger(\mathbf{t})\hat{\mathcal{H}}\hat{U}(\mathbf{t}) | \Omega_\mu \rangle \\ &\quad - \frac{1}{2} \langle \Phi_0 | \hat{U}^\dagger(\mathbf{t})\hat{\mathcal{H}}\hat{U}(\mathbf{t}) | \Phi_0 \rangle \\ &\quad - \frac{1}{2} \langle \Phi_\mu | \hat{U}^\dagger(\mathbf{t})\hat{\mathcal{H}}\hat{U}(\mathbf{t}) | \Phi_\mu \rangle \end{aligned} \quad (25)$$

where Ω_μ is an easily preparable superposition of Φ_0 and Φ_μ

$$|\Omega_\mu\rangle = e^{\frac{\pi}{4}\hat{k}_\mu} |\Phi_0\rangle = \frac{1}{\sqrt{2}} |\Phi_0\rangle + \frac{1}{\sqrt{2}} |\Phi_\mu\rangle \quad (26)$$

3.4 Selected PQE. QFORTE implements a selected variant of PQE (SPQE) that, similarly to ADAPT, utilizes a dUCC ansatz constructed iteratively from a pool of arbitrary-order particle-hole operators. In brief, the selection procedure is done by construction of a normalized state $|\tilde{r}\rangle$ that approximates the residual and is defined as

$$\begin{aligned} |\tilde{r}\rangle &= \hat{U}^\dagger(\mathbf{t}) e^{i\Delta t \hat{H}} \hat{U}(\mathbf{t}) |\Phi_0\rangle \\ &= (1 + i\Delta t \hat{U}^\dagger(\mathbf{t}) \hat{H} \hat{U}(\mathbf{t})) |\Phi_0\rangle + \mathcal{O}(\Delta t^2) \end{aligned} \quad (27)$$

For small values of Δt , measurement of the state $|\tilde{r}\rangle$ in the computational basis yields an excited determinant Φ_ν with probability $|C_\nu|^2 \approx (\Delta t)^2 |\langle \Phi_\nu | \hat{U}^\dagger(\mathbf{t}) \hat{H} \hat{U}(\mathbf{t}) | \Phi_0 \rangle|^2$, a quantity proportional to the residual squared $|r_\nu|^2$. We may then approximate the values of the (normalized) squared residuals as

$$|\tilde{r}_\nu|^2 \approx \frac{N_\nu}{M} \quad (28)$$

where N_ν is the number of times the state Φ_ν is measured from M preparations of $|\tilde{r}\rangle$. A cumulative thresholding procedure is then utilized to add new operators \hat{k}_ν to the ansatz, enforcing the condition

$$\sum_{\hat{k}_\nu \notin \mathcal{A}}^{\text{excluded}} |r_\nu|^2 \approx \sum_{\hat{k}_\nu \notin \mathcal{A}}^{\text{excluded}} \frac{|\tilde{r}_\nu|^2}{\Delta t^2} \leq \Omega^2 \quad (29)$$

where Ω is a user-specified convergence threshold parameter. Contrary to ADAPT-VQE where an operator may occur multiple times in the ansatz, in SPQE each particle-hole operator may be selected only once. In QFORTE the time step is taken as a parameter of the calculation and the Suzuki-Trotter approximation^{76,77} is used for the time evolution operator. This selection strategy is particularly appealing for strongly correlated systems because it does not require the candidate operators \hat{k}_ν to be restricted to any particular excitation order.

3.5 Quantum imaginary time evolution. The quantum imaginary time evolution (QITE) algorithm¹³ obtains the ground state $|\Psi_0\rangle$ by propagating a trial state $|\Phi\rangle$ (assuming $\langle\Psi_0|\Phi\rangle \neq 0$) via imaginary time evolution $e^{-\beta\hat{\mathcal{H}}}$ in the limit of infinite propagation time,

$$|\Psi_0\rangle = \lim_{\beta \rightarrow \infty} \frac{1}{\sqrt{N(\beta)}} e^{-\beta\hat{\mathcal{H}}} |\Phi\rangle \quad (30)$$

The factor $1/\sqrt{N(\beta)} = |\langle\Phi|e^{-2\beta\hat{\mathcal{H}}}\Phi\rangle|^{-1/2}$ guarantees normalization of the evolved state. Since the imaginary time evolution operator is non-unitary, its direct implementation on quantum computers is impractical. QITE circumvents this limitation by constructing a unitary that approximates the action of the imaginary time evolution operator for a small time step $\Delta\beta$. This unitary is obtained by matching a state propagated from imaginary time β to $\beta + \Delta\beta$ with the unitary $\exp(-i\Delta\beta\hat{A})$, where \hat{A} is Hermitian:

$$|\psi(\beta + \Delta\beta)\rangle = N(\Delta\beta)^{-1/2} e^{-\Delta\beta\hat{\mathcal{H}}} |\psi(\beta)\rangle = e^{-i\Delta\beta\hat{A}} |\psi(\beta)\rangle \quad (31)$$

The operator \hat{A} can be written as a linear expansion of Pauli strings $\hat{\rho}_\mu = \prod_l \hat{\sigma}_{\mu_l}^{(l)}$ such that $\hat{A} = \sum_{\mu \in \mathcal{P}} \alpha_\mu \hat{\rho}_\mu$. Here, \mathcal{P} is a subset with dimension M of all possible $4^{N_{\text{qb}}}$ Pauli strings, $\mu \equiv (\mu_1, \mu_2, \dots, \mu_{N_{\text{qb}}})$ is a multi-index describing a unique Pauli operator product, and $\mu_l \in \{I, X, Y, Z\}$. Equations for QITE are obtained by expanding Eq. (31) up to linear terms and left-projecting onto $\langle\Phi|\hat{A}^\dagger$, yielding the M -dimensional linear system $\mathbf{S}\boldsymbol{\alpha} = \mathbf{b}$. The elements of \mathbf{S} , and \mathbf{b} , as well as the value of $N(\Delta\beta)$ can be determined via measurement of simple symmetric expectation values:¹³

$$N(\Delta\beta) \approx 1 - 2\Delta\beta \langle\Phi|\hat{\mathcal{H}}|\Phi\rangle \quad (32)$$

$$S_{\mu\nu} = \langle\Phi|\hat{\rho}_\mu^\dagger \hat{\rho}_\nu |\Phi\rangle \quad (33)$$

and,

$$b_\mu = \frac{-i}{\sqrt{N(\Delta\beta)}} \langle\Phi|\hat{\rho}_\mu^\dagger \hat{\mathcal{H}} |\Phi\rangle \quad (34)$$

Here $|\Phi\rangle$ is an approximation to the exact time evolved state $|\psi(\beta)\rangle$. We note that the overall QITE procedure is iterative, and requires that the linear system be solved $M = \beta/\Delta\beta$ times to reach the target evolution time β .

In QFORTE, the Pauli strings that enter into \mathcal{P} are generated automatically from a user-specified manifold of second-quantized operators. Specifically, given a set of excitation/de-excitation operators, these are first Jordan-Wigner transformed, and all unique terms containing an odd number of Y gates are included in \mathcal{P} . We note that the QFORTE implementation of QITE differs from the “inexact QITE” described in Ref. 13, in which a subgroup of important Pauli operators is chosen for *each* k -local Hamiltonian term.

3.6 Quantum Lanczos. The quantum Lanczos algorithm¹³ expands an eigenstate as a linear combination of states obtained via QITE:

$$|\Psi\rangle = \sum_{n=0}^M c_n |\psi(\beta_n)\rangle \quad (35)$$

Variational minimization of the energy of the state Ψ with respect to the coefficients c_n leads to the generalized eigenvalue problem $\mathbf{H}\mathbf{c} = \mathbf{S}\mathbf{c}E$, where the matrix elements $(\mathbf{H})_{mn} = \langle\psi(\beta_m)|\hat{\mathcal{H}}|\psi(\beta_n)\rangle$ and $(\mathbf{S})_{mn} = \langle\psi(\beta_m)|\psi(\beta_n)\rangle$, with $\beta_m = m\Delta\beta$, are obtained from the QITE subroutine. A convenient feature of QL is that the matrix elements can be (approximately) evaluated in terms of the normalization coefficients N and symmetric energy expectation values:

$$S_{mn} \approx \langle\Phi|e^{-m\Delta\beta\hat{A}}e^{-n\Delta\beta\hat{A}}|\Phi\rangle = \frac{N(\beta_m)N(\beta_n)}{N^2(\beta_k)} \quad (36)$$

and,

$$\begin{aligned} H_{mn} &\approx \langle\Phi|e^{-m\Delta\beta\hat{A}}\hat{\mathcal{H}}e^{-n\Delta\beta\hat{A}}|\Phi\rangle \\ &= \frac{N(\beta_m)N(\beta_n)}{N^2(\beta_k)} \langle\psi(\beta_k)|\hat{\mathcal{H}}|\psi(\beta_k)\rangle \end{aligned} \quad (37)$$

where $2k = m + n$. This is a significant simplification that allows the determination of all quantities needed for QLanczos without using ancilla qubits.

3.7 Quantum Krylov (QK). In quantum Krylov diagonalization,^{50,51} a general state is written as a linear combination of the basis $\{\psi_n\}$ generated from real-time Hamiltonian evolutions at a given set of $s + 1$ time steps $\{t_0, t_1, \dots, t_s\}$ (typically chosen of the form $t_n = n\Delta t$) from a reference state Φ :

$$|\Psi\rangle = \sum_{n=0}^s c_n |\psi_n\rangle = \sum_{n=0}^s c_n e^{-it_n \hat{H}} |\Phi\rangle \quad (38)$$

Like in the case of QLanczos, variational minimization of the energy of the state Ψ leads to a generalized eigenvalue problem $\mathbf{H}\mathbf{c} = \mathbf{S}\mathbf{c}E$, where the elements of the overlap matrix (\mathbf{S}) and Hamiltonian (\mathbf{H}) are given by $S_{mn} = \langle \psi_m | \psi_n \rangle$ and $H_{mn} = \langle \psi_m | \hat{H} | \psi_n \rangle$, respectively. In QFORTE, the quantum circuit used to approximate the real-time evolution is generated via Eq. (9) using the Suzuki-Trotter decomposition

$$e^{-it\hat{H}} = e^{-it\sum_{\ell}\theta_{\ell}\hat{P}_{\ell}} \approx \left(\prod_{\ell} e^{-it\theta_{\ell}\hat{P}_{\ell}/r} \right)^r \quad (39)$$

with r Trotter steps. The quantum circuits used to evaluate \mathbf{S} and \mathbf{H} are implemented in QFORTE via a variant of the Hadamard test, requiring only an additional ancillary qubit.⁷⁸ The time step (Δt), number of time evolutions states (s), and number of Trotter steps (r) are all given as user-specified values. We note that QFORTE can also realize QK via exact real-time evolution (as opposed to those resulting from Trotterized dynamics) using sparse matrix operations.

3.8 Multireference selected QK. A selected multireference variant of QK (MRSQK) is also implemented in QFORTE. The base procedure is identical to that of QK described in the above section, but several orthogonal reference states $|\Phi_I\rangle$ are included in the subspace and time evolved in order to improve numerical stability and target states with multireference character. The MRSQK wave

function is thus given by

$$|\Psi\rangle = \sum_{\alpha} c_{\alpha} |\psi_{\alpha}\rangle = \sum_{l=1}^d \sum_{n=0}^s c_l^{(n)} e^{-it_n \hat{H}} |\Phi_l\rangle \quad (40)$$

In MRSQK, a preliminary single-reference QK calculation is performed in order to determine the set of important references $\{\Phi_l\}$. The resulting single-reference QK wave function $|\tilde{\Psi}\rangle = \sum_n \tilde{c}_n |\psi_n\rangle$ is used to construct a list of determinants with importance value $P_{\mu} = |\langle\phi_{\mu}|\tilde{\Psi}\rangle|^2$, since the probability of measuring a determinant ϕ_{μ} is equal to $P_{\mu} = |\langle\phi_{\mu}|\tilde{\Psi}\rangle|^2$. In QFORTE, the quantity P_{μ} is approximated by measuring each element of the Krylov basis and estimating the total probability as a weighted sum over references via

$$P_{\mu} = \left| \sum_{\alpha} \langle\phi_{\mu}|\psi_{\alpha}\rangle c_{\alpha} \right|^2 \approx \sum_{\alpha} |\langle\phi_{\mu}|\psi_{\alpha}\rangle|^2 |c_{\alpha}|^2 \quad (41)$$

Once formed, the list of the most important determinants is augmented to guarantee that all spin arrangements of open-shell determinants are included, and the d most important references are used in the MRSQK subspace. The number of references (d), the MRSQK time step Δt_{mr} , the number of time evolutions per reference (s), and the parameters for the preliminary single-reference QK calculation are all specified by the user at runtime.

4. REPRESENTATIVE TIMINGS

By implementing an optimized low-level state-vector emulator, the algorithms implemented in QFORTE can be applied in a reasonable amount of time to small-sized molecular systems. This enables the rapid development and generation of paper-quality benchmark data for new algorithms implemented in QFORTE. To illustrate this point, we report timings for operations common to many quantum algorithms, such as application of circuits and evaluation of expectation values. As an example, we consider the cost to generate a state-vector corresponding to a disentangled UCC ansatz (see Eq. 18) with particle-hole single and double excitations (dUCCSD) for a family of molecular hydrogen systems $\text{H}_2\text{--H}_{10}$. The dUCCSD state is produced by applying the corresponding unitary

\hat{U}_{SD} to a Hartree–Fock reference. In this example, we employ a minimum basis set, corresponding to computations on 4–20 qubits. We also consider the time required to evaluate the Hamiltonian expectation value of the dUCCSD state via exact operator averaging: $\langle \mathcal{H}_q \rangle = \sum_{\ell}^{N_{\text{ps}}} h_{\ell} \langle \hat{P}_{\ell} \rangle$. Figure 5 shows that QFORTE can accomplish such operations on the order of fractions of a second to a few minutes (on a laptop computer) for the systems considered here. In particular, operations on systems with 6 electrons can be performed in the order of less than a second, enabling rapid testing of algorithms on relatively complex problems. Based on these results, we estimate that the black-box algorithms currently implemented in QFORTE can be used to study systems up to a size of approximately 10 spatial orbitals in approximately one to four hours depending on the specific algorithm and run parameters.

We also benchmarked the efficiency of QFORTE’s state vector emulator against those implemented in CIRQ¹⁸ and QSIM³² (each using a single thread). In Fig. 6 we report the average time (over 1000 applications) required to apply a trial circuit on a particular simulator. The trial circuit is similar to that used for benchmarking in Ref. 29 and is composed of a Hadamard transform, a cascade of CNOT operations on neighboring qubits, and another Hadamard transform. We observe that for small simulations (4–12 qubits), QFORTE is faster than CIRQ or QSIM, with the speedup afforded by QFORTE in this regime being likely due to a higher constant overhead in the other packages. For computations with 14 and 16 qubits, the timing of the three packages is comparable, while for more than 20 qubits both CIRQ and QSIM are faster than QFORTE, with QSIM achieving speedups of up to approximately 10 times.

5. EXAMPLE: DEVELOPING NEW ALGORITHMS WITH QFORTE

In this section we discuss some examples of how QFORTE can be used to facilitate the implementation of new quantum algorithms. We also then show how QFORTE can be used to produce comparative studies of different algorithms. In addition to the example described below, QFORTE has several Jupyter-notebook tutorials available on topics ranging from basic API use, to detailed instructions for various algorithm implementations, to running jobs in a black-box fashion.

As an example, we consider the workflow necessary to experiment with a new VQE ansatz based

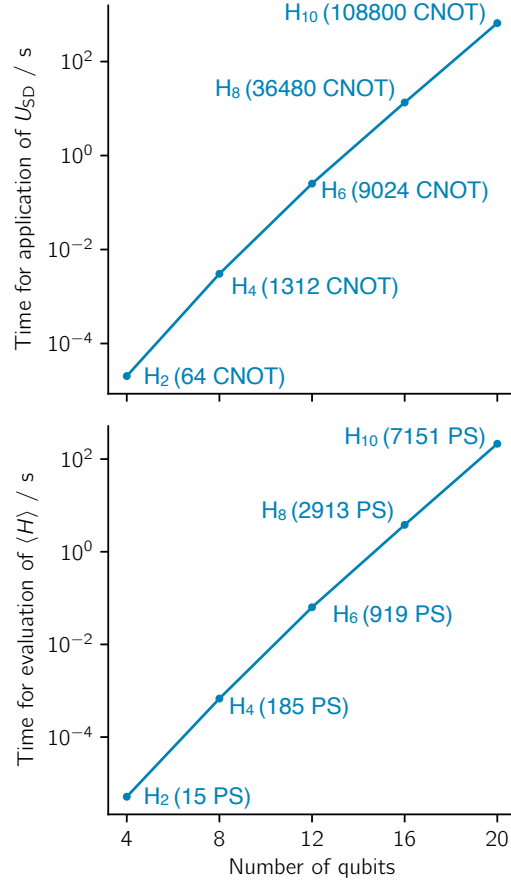


Figure 5: Timings for (top) the application of disentangled UCCSD circuits to a quantum states of increasing dimension, and for (bottom) the evaluation the molecular hydrogen Hamiltonians of increasing size (H₂–H₁₀). The number of CNOT gates in the dUCCSD circuits and the number of Pauli strings (PS) in the Hamiltonian are reported in parentheses beside the corresponding systems in the top and bottom plots, respectively. All cluster amplitudes in the dUCCSD circuits were initialized to 1.0. All timings shown were generated using a single thread on a laptop computer with an Intel i5 3.1 GHz processor.

on the theory of paired CC doubles.^{79,80} We will formulate the ansatz in the disentangled unitary coupled cluster form [Eq. (18)], such that operators entering into the resulting (disentangled) paired UCC doubles (p-dUCCD) ansatz maintain a seniority-zero trial state (include only contributions from closed-shell determinants). We would also like to enforce that the ansatz includes only particle-hole operators, such that the final ansatz has the form:

$$\hat{U}_{\text{p-dUCCD}}(\mathbf{t}) = \prod_i^{\text{occ}} \prod_a^{\text{vir}} e^{t_i^a (\hat{a}_{i\beta}^\dagger \hat{a}_{i\alpha}^\dagger \hat{a}_{a\alpha} \hat{a}_{a\beta} - \hat{a}_{a\beta}^\dagger \hat{a}_{a\alpha}^\dagger \hat{a}_{i\alpha} \hat{a}_{i\beta})} \quad (42)$$

where the indices i , and a pertain to occupied or virtual spatial orbitals, respectively, for a specified

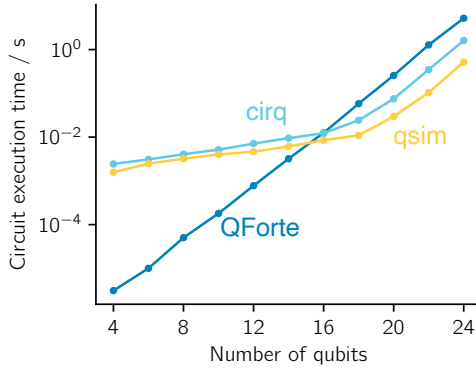


Figure 6: Timings using QFORTE, CIRQ, and QSIM for the application of a trial circuit comprised of a Hadamard transform, a cascade of CNOT gates, and another Hadamard transform. Timings are reported as an average over 1000 circuit applications. All timings shown were generated on a laptop computer with an Intel i5 3.1 GHz processor.

Hartree-Fock reference state.

The Python class structure in QFORTE can easily facilitate the implementation of the p-dUCCD-VQE variant. To begin, we define a mixin class (pdUCCD) that defines only the function `ansatz_circuit()`. This function is responsible for returning the parameterized unitary circuit $\hat{U}_{\text{p-dUCCD}}(\mathbf{t})$ comprised of gates of the form shown in Fig. 4 for the exponentials of Pauli-strings. The `ansatz_circuit()` function provided by the mixin class will be called by the function `AnsatzAlgorithm.energy_feval()` which applies $\hat{U}_{\text{p-dUCCD}}(\mathbf{t})$ to a `Computer()` (initialized to the Hartree-Fock reference) and returns the energy expectation value (without noise by default). Next, we define a new class, `pdUCCDVQE`, derived from the VQE abstract base class and the `pdUCCD` mixin class. To complete the implementation of the `pdUCCDVQE` child class, we simply need to define two functions: `run()` and `solve()`. The `run()` function takes the user-defined parameters such as convergence thresholds, defines algorithm specific attributes such as the number of occupied and virtual orbitals, and calls all necessary subroutines to run the algorithm.

Finally, the `solve()` function calls the user-specified classical optimization algorithm (BFGS⁸¹⁻⁸⁴ by default), and defines: (i) the number of classical parameters used, (ii) the number of CNOT gates in the ansatz circuit (without considering advanced compilation techniques⁸⁵), and (iii) to total number of Pauli-string evaluations ($\langle\langle\Phi_0|\hat{U}_{\text{p-dUCCD}}^\dagger(\mathbf{t})\hat{P}_\ell\hat{U}_{\text{p-dUCCD}}(\mathbf{t})|\Phi_0\rangle\rangle$). An overview of the steps described above is given in Lst. 8. We note that in practice QFORTE requires that derived

classes define printing and attribute verification functions in addition to the three described above, but we do not show these in Lst. 8 for brevity.

After implementing the p-dUCCD-VQE method, we would like to compare it to other quantum algorithms, like VQE based on a conventional disentangled UCC ansatz. Once an algorithm has been implemented, running jobs in QFORTE is very simple, as we only need to pass a molecule object to an algorithm constructor (see Sec. 2.3). For example, Lst. 9 shows how to calculate the potential energy curve for H_4 using the new p-dUCCD-VQE method, VQE with a disentangled UCC single and double excitations ansatz (dUCCSD-VQE), dUCCSD optimized via PQE (dUCCSD-PQE), and quantum Krylov diagonalization with a Hartree-Fock reference (QK). After running the algorithms, it is possible to extract salient information such as the predicted energy, and computational resource estimates. Using the code in Lst. 9, we can produce the potential energy curve scans shown in Fig. 7 and obtain the computational resources estimates reported in Tab. 1. In this example, we conclude that although the p-dUCCD-VQE circuit is very compact (from a circuit depth and classical parameterization standpoint), it does not afford the same variational flexibility as the other dUCC-based methods or QK.

Table 1: Computational resource estimates and mean signed error for p-dUCCD-VQE, dUCCSD-VQE, dUCCSD-PQE and QK($s=3$, $\Delta t = 0.5$ a.u.) computed for the dissociation of linear H_4 in a STO-3G basis. N_{par} is the number of classical parameters used in the ansatz circuit for VQE/PQE or the dimension of the generalized eigenvalue problem for QK, N_{CNOT} is the number of CNOT gates in the ansatz (for VQE/PQE) or Trotterized time-evolution circuit (for QK), \bar{N}_{PSE} is the average number of Pauli-string evaluations over the entire potential energy curve, and MSE is the mean signed energy error (in mE_h , with respect to the exact energy) computed over the entire potential curve (H-H near-neighbor bond distances in the range 0.5–2.0 Å).

Method	N_{par}	N_{CNOT}	\bar{N}_{PSE}	MSE (mE_h)
p-dUCCD-VQE	4	192	11063	80.922
dUCCSD-VQE	14	736	134804	1.204
dUCCSD-PQE	14	736	94283	1.204
QK	4	2656	2972	23.483

6. CONCLUSION

In this article we introduce the new open-source software package QFORTE. QFORTE aids the development and testing of quantum algorithms for molecular electronic structure, and has been used by our research group to implement the algorithms introduced in Refs. 51 and 53. We also

```

1 from qforte import *
2 from scipy.optimize import minimize
3
4 class pdUCCD:
5     def ansatz_circuit(self, params):
6         Kq = QubitOperator()
7
8         for i in range(self._nocc):
9             for a in range(self._nvir):
10                mu = i*self._nocc + a
11                k_mu = SQOperator()
12                ia = 2*i
13                ib = 2*i+1
14                aa = 2*self._nocc + 2*a
15                ab = 2*self._nocc + 2*a+1
16                k_mu.add( 1.0*params[mu], [ab, aa], [ia, ib])
17                k_mu.add(-1.0*params[mu], [ib, ia], [aa, ab])
18                Kq.add(k_mu.jw_transform())
19
20        U, phase = trotterize(Kq)
21        return U
22
23 class pdUCCDVQE(pdUCCD, VQE):
24     def run(self, optimizer='BFGS'):
25         self._optimizer = optimizer
26         self._nocc = int(sum(self._ref) / 2)
27         self._nvir = int((self._nqb - sum(self._ref)) / 2)
28         self._npar = int(self._nocc*self._nvir)
29         self._params_0 = [0.0 for k in range(self._npar)]
30         self.solve()
31
32     def solve(self):
33         res = minimize(self.energy_feval, self._params_0, method=self.
34         _optimizer)
35         self._final_res = res
36         self._Egs = res.fun
37         self._n_classical_params = len(res.x)
38         self._n_cnot = self.build_Uvqc(res.x).get_num_cnots()
39         self._n_pauli_trm_measures = self._Nl * res.nfev

```

Listing 8: Example Python code i) defining a mixin class pdUCCD responsible for the construction of the circuit for the p-dUCCD ansatz, and ii) defining a derived pdUCCDVQE class responsible for the optimization of the ansatz. Note that QFORTE requires that all derived classes additionally define printing and attribute verification functions in addition to those shown here.

```

1 from qforte import *
2 import numpy as np
3
4 for r in np.linspace(0.5, 2.0, 30):
5
6     # Specify geometry as a function of r.
7     geom = [('H', (0., 0., 0.0)),
8             ('H', (0., 0., 1*r)),
9             ('H', (0., 0., 2*r)),
10            ('H', (0., 0., 3*r))]
11
12    # Run classical scf with Psi4 backend.
13    mol = system_factory(build_type='psi4', mol_geometry=geom, basis='sto-3g')
14
15    # Instantiate and run new VQE algorithm.
16    pdUCCD_VQE = pdUCCDVQE(mol)
17    pdUCCD_VQE.run()
18
19    # Instantiate and run black-box algorithms.
20    dUCCSD_VQE = UCCNVQE(mol)
21    dUCCSD_VQE.run(pool_type='SD', opt_thresh = 1.0e-4)
22
23    dUCCSD_PQE = UCCNVQE(mol)
24    dUCCSD_PQE.run(pool_type='SD', res_vec_thresh = 1.0e-4)
25
26    QK = SRQK(mol)
27    QK.run(s=3)

```

Listing 9: Example of how to compute a potential energy curve of the linear H₄ molecule using both the pdUCCDVQE algorithm implemented in List. 8 and various black-box algorithms implemented in QFORTE.

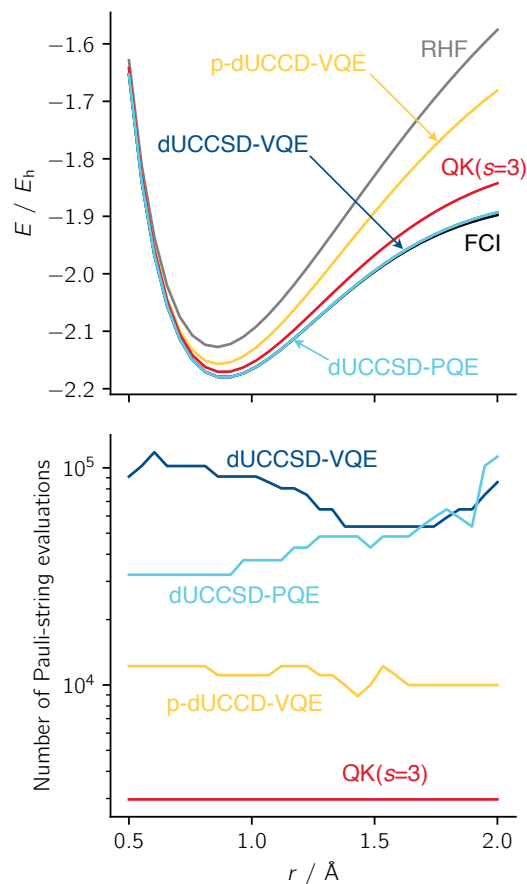


Figure 7: Ground state potential energy curve (top) and number of Pauli-string evaluations (bottom) computed with the p-dUCCD-VQE method implemented in Lst. 8, dUCCSD-VQE, dUCCSD-PQE, and QK with three time evolved basis states for the dissociation of linear H_4 in a STO-3G basis. The QK calculation used a time step of $\Delta t = 0.5$ a.u. and a Trotterized time evolution circuit with a single Trotter step ($r = 1$). The restricted Hartree-Fock (RHF) and FCI curves are also reported for reference.

demonstrate, for systems as large as H_{10} (10 electrons in 10 orbitals mapped to 20 qubits), that QFORTE can be used to perform important subroutines such as application of the compiled dUCCSD circuit ($\sim 10^5$ CNOT gates) and the evaluation of the Hamiltonian expectation value (~ 7000 Pauli strings) in a modest amount of time ($\sim 10\text{--}30$ s). The ability of QFORTE to facilitate black-box calculations with a wide variety of quantum algorithms using only a classical electronic structure package as a dependency makes it a useful tool for comparing quantum algorithms based on the accuracy of their outputs and their required computational resources. Moreover, the easy-to-use basic components (implemented as C++ classes exposed in Python), combined with a simple Python class structure, allow QFORTE to function as an excellent platform for implementing and testing new

quantum algorithms. Note that each complementary task in QFORTE’s quantum workflow pipeline (see Fig. 1), can likewise be accomplished by a variety of other packages. In some cases packages targeting each specific use case (i.e. fermionic encoding, circuit simulation, modeling realistic error channels, etc..) may provide more features or better performance. However, the novelty of QFORTE is its ability to execute the entire pipeline in a single package and using a concise input format.

In the future, we plan to expand the features and quantum algorithms implemented in QFORTE with the hope that its unique capabilities will provide a useful tool to other researchers. For example, we plan to add support in the low-level layer of QFORTE for spin and point-group symmetries and enable the treatment of open-shell systems. Another desirable feature is supporting fermionic encodings beyond the currently-available Jordan-Wigner transformation. On the front of the emulator performance, a significant speedup could be achieved by applying particle number and spin symmetry restrictions to the state vector, such that the application of fermionic operators can be performed efficiently using well-established quantum chemistry techniques,⁸⁶ extending the scope of routine computation to systems with more than 24 qubits. Such an approach has recently been realized in the so-called fermionic quantum emulator.²² Additionally we plan to implement API(s) to several of the leading platforms (such as QISKIT and PYQUIL) capable of facilitating calculations on real quantum hardware.

The current set of algorithms and features implemented by QFORTE is by no means representative of the full spectrum of recent advances in the field of quantum simulation.

On the algorithm side, we plan to extend the type of ansätze supported in QFORTE (see, e.g., Refs. 42,43,87–89). We are also interested in implementing additional excited-state methods (see, e.g., Refs. 46,66,90,91). Other desirable features include techniques for quantum resource reduction such as qubit tapering,⁹² Hamiltonian factorization,⁹³ and circuit compilation.⁸⁵ It is our hope that QFORTE will become a valuable asset to the quantum simulation community. We welcome feedback and contributions from any who wish to see their work represented in our package.

Acknowledgement

This work was supported by the U.S. Department of Energy under Award No. DE-SC0019374. N.H.S. was supported by a fellowship from The Molecular Sciences Software Institute under NSF grant ACI-1547580. QFORTE was predominantly developed as a Molecular Sciences Software Institute fellowship project. N.H.S. would like to thank his mentor at the institute Jonathan Moussa for his thoughts and advice. In addition to development by N.H.S. and F.A.E., the authors acknowledge contributions to the QFORTE code from Nan He, Renke Huang, Jonathon Misiewicz, and Ilias Magoulas.

REFERENCES

- (1) Arute, F.; Arya, K.; Babbush, R.; Bacon, D.; Bardin, J. C.; Barends, R.; Biswas, R.; Boixo, S.; Brandao, F. G.; Buell, D. A. Quantum supremacy using a programmable superconducting processor. *Nature* **2019**, *574*, 505–510.
- (2) 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.
- (3) 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.
- (4) Colless, J. I.; Ramasesh, V. V.; Dahlen, D.; Blok, M. S.; Kimchi-Schwartz, M.; McClean, J.; Carter, J.; De Jong, W.; Siddiqi, I. Computation of molecular spectra on a quantum processor with an error-resilient algorithm. *Phys. Rev. X* **2018**, *8*, 011021.
- (5) Shen, Y.; Zhang, X.; Zhang, S.; Zhang, J.-N.; Yung, M.-H.; Kim, K. Quantum implementation of the unitary coupled cluster for simulating molecular electronic structure. *Phys. Rev. A* **2017**, *95*, 020501.
- (6) Hempel, C.; Maier, C.; Romero, J.; McClean, J.; Monz, T.; Shen, H.; Jurcevic, P.; Lanyon, B. P.; Love, P.; Babbush, R., et al. Quantum chemistry calculations on a trapped-ion quantum simulator. *Phys. Rev. X* **2018**, *8*, 031022.
- (7) Nam, Y.; Chen, J.-S.; Pienta, N. C.; Wright, K.; Delaney, C.; Maslov, D.; Brown, K. R.; Allen, S.; Amini, J. M.; Apisdorf, J., et al. Ground-state energy estimation of the water molecule on a trapped-ion quantum computer. *npj Quantum Inf.* **2020**, *6*, 1–6.

- (8) Lin, C. Y.-Y.; Zhu, Y. Performance of QAOA on typical instances of constraint satisfaction problems with bounded degree. *arXiv preprint arXiv:1601.01744* **2016**,
- (9) Wang, Z.; Hadfield, S.; Jiang, Z.; Rieffel, E. G. Quantum approximate optimization algorithm for maxcut: A fermionic view. *Phys. Rev. A* **2018**, *97*, 022304.
- (10) Preskill, J. Quantum Computing in the NISQ era and beyond. *Quantum* **2018**, *2*, 79.
- (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) McClean, J. R.; Romero, J.; Babbush, R.; Aspuru-Guzik, A. The theory of variational hybrid quantum-classical algorithms. *New J. Phys.* **2016**, *18*, 023023.
- (13) Motta, M.; Sun, C.; Tan, A. T.; O'Rourke, M. J.; Ye, E.; Minnich, A. J.; Brandão, F. G.; Chan, G. K.-L. Determining eigenstates and thermal states on a quantum computer using quantum imaginary time evolution. *Nat. Phys.* **2019**, *16*, 1–6.
- (14) McArdle, S.; Endo, S.; Aspuru-Guzik, A.; Benjamin, S. C.; Yuan, X. Quantum computational chemistry. *Rev. Mod. Phys.* **2020**, *92*, 015003.
- (15) Bauer, B.; Bravyi, S.; Motta, M.; Kin-Lic Chan, G. Quantum algorithms for quantum chemistry and quantum materials science. *Chem. Rev.* **2020**, *120*, 12685–12717.
- (16) Smart, S. E.; Mazziotti, D. A. Quantum Solver of Contracted Eigenvalue Equations for Scalable Molecular Simulations on Quantum Computing Devices. *Phys. Rev. Lett.* **2021**, *126*, 070504.
- (17) Aleksandrowicz, G.; Alexander, T.; Barkoutsos, P.; Bello, L.; Ben-Haim, Y.; Bucher, D.; Cabrera-Hernández, F. J.; Carballo-Franquis, J.; Chen, A.; Chen, C.-F.; Chow, J. M.; Córcoles-Gonzales, A. D.; Cross, A. J.; Cross, A.; Cruz-Benito, J.; Culver, C.; González, S. D. L. P.; Torre, E. D. L.; Ding, D.; Dumitrescu, E.; Duran, I.; Eendebak, P.; Everitt, M.; Sertage, I. F.;

- Frisch, A.; Fuhrer, A.; Gambetta, J.; Gago, B. G.; Gomez-Mosquera, J.; Greenberg, D.; Hamamura, I.; Havlicek, V.; Hellmers, J.; Herok, L.; Horii, H.; Hu, S.; Imamichi, T.; Itoko, T.; Javadi-Abhari, A.; Kanazawa, N.; Karazeev, A.; Krsulich, K.; Liu, P.; Luh, Y.; Maeng, Y.; Marques, M.; Martín-Fernández, F. J.; McClure, D. T.; McKay, D.; Meesala, S.; Mezzacapo, A.; Moll, N.; Rodríguez, D. M.; Nannicini, G.; Nation, P.; Ollitrault, P.; O’Riordan, L. J.; Paik, H.; Pérez, J.; Phan, A.; Pistoia, M.; Prutyaynov, V.; Reuter, M.; Rice, J.; Davila, A. R.; Rudy, R. H. P.; Ryu, M.; Sathaye, N.; Schnabel, C.; Schoute, E.; Setia, K.; Shi, Y.; Silva, A.; Siraichi, Y.; Sivarajah, S.; Smolin, J. A.; Soeken, M.; Takahashi, H.; Tavernelli, I.; Taylor, C.; Taylour, P.; Trabing, K.; Treinish, M.; Turner, W.; Vogt-Lee, D.; Vuillot, C.; Wildstrom, J. A.; Wilson, J.; Winston, E.; Wood, C.; Wood, S.; Wörner, S.; Akhalwaya, I. Y.; Zoufal, C. Qiskit: An Open-source Framework for Quantum Computing. **2019**,
- (18) Developers, C. Cirq. 2021; <https://doi.org/10.5281/zenodo.4750446>, See full list of authors on Github: <https://github.com/quantumlib/Cirq/graphs/contributors>.
- (19) Microsoft, The qsharp user guide - microsoft quantum. **2020**,
- (20) Smith, R. S.; Curtis, M. J.; Zeng, W. J. A practical quantum instruction set architecture. *arXiv preprint arXiv:1608.03355* **2016**,
- (21) Killoran, N.; Izaac, J.; Quesada, N.; Bergholm, V.; Amy, M.; Weedbrook, C. Strawberry fields: A software platform for photonic quantum computing. *Quantum* **2019**, 3, 129.
- (22) 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.
- (23) McClean, J.; Rubin, N.; Sung, K.; Kivlichan, I. D.; Bonet-Monroig, X.; Cao, Y.; Dai, C.; Fried, E. S.; Gidney, C.; Gimby, B., et al. OpenFermion: the electronic structure package for quantum computers. *Quantum Sci. Technol.* **2020**,
- (24) Smelyanskiy, M.; Sawaya, N. P.; Aspuru-Guzik, A. qHiPSTER: The quantum high performance software testing environment. *arXiv preprint arXiv:1601.07195* **2016**,

- (25) Luo, X.-Z.; Liu, J.-G.; Zhang, P.; Wang, L. Yao. jl: Extensible, efficient framework for quantum algorithm design. *Quantum* **2020**, *4*, 341.
- (26) Suzuki, Y.; Kawase, Y.; Masumura, Y.; Hiraga, Y.; Nakadai, M.; Chen, J.; Nakanishi, K. M.; Mitarai, K.; Imai, R.; Tamiya, S., et al. Qulacs: a fast and versatile quantum circuit simulator for research purpose. *arXiv preprint arXiv:2011.13524* **2020**,
- (27) 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., et al. Tequila: A platform for rapid development of quantum algorithms. *Quantum Sci. Technol.* **2021**, *6*, 024009.
- (28) Bergholm, V.; Izaac, J.; Schuld, M.; Gogolin, C.; Alam, M. S.; Ahmed, S.; Arrazola, J. M.; Blank, C.; Delgado, A.; Jahangiri, S., et al. PennyLane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968* **2018**,
- (29) Steiger, D. S.; Häner, T.; Troyer, M. ProjectQ: an open source software framework for quantum computing. *Quantum* **2018**, *2*, 49.
- (30) Sun, Q.; Berkelbach, T. C.; Blunt, N. S.; Booth, G. H.; Guo, S.; Li, Z.; Liu, J.; McClain, J. D.; Sayfutyarova, E. R.; Sharma, S., et al. PySCF: the Python-based simulations of chemistry framework. *Wiley Interdiscip. Rev. Comput. Mol. Sci.* **2018**, *8*, e1340.
- (31) Smith, D. G.; Burns, L. A.; Simmonett, A. C.; Parrish, R. M.; Schieber, M. C.; Galvelis, R.; Kraus, P.; Kruse, H.; Di Remigio, R.; Alenaizan, A., et al. Psi4 1.4: Open-source software for high-throughput quantum chemistry. *J. Chem. Phys.* **2020**, *152*, 184108.
- (32) team, Q. A.; collaborators, qsim. 2020; <https://doi.org/10.5281/zenodo.4023103>.
- (33) Boixo, S.; Isakov, S. V.; Smelyanskiy, V. N.; Neven, H. Simulation of low-depth quantum circuits as complex undirected graphical models. *arXiv preprint arXiv:1712.05384* **2017**,
- (34) Bravyi, S.; Browne, D.; Calpin, P.; Campbell, E.; Gosset, D.; Howard, M. Simulation of quantum circuits by low-rank stabilizer decompositions. *Quantum* **2019**, *3*, 181.

- (35) Gray, J.; Kourtis, S. Hyper-optimized tensor network contraction. *Quantum* **2021**, *5*, 410.
- (36) Huang, C.; Zhang, F.; Newman, M.; Cai, J.; Gao, X.; Tian, Z.; Wu, J.; Xu, H.; Yu, H.; Yuan, B., et al. Classical simulation of quantum supremacy circuits. *arXiv preprint arXiv:2005.06787* **2020**,
- (37) Huang, Y.; Love, P. Approximate stabilizer rank and improved weak simulation of Clifford-dominated circuits for qudits. *Physical Review A* **2019**, *99*, 052307.
- (38) Huang, Y.; Love, P. Feynman-path-type simulation using stabilizer projector decomposition of unitaries. *Physical Review A* **2021**, *103*, 022428.
- (39) Markov, I. L.; Shi, Y. Simulating quantum computation by contracting tensor networks. *SIAM Journal on Computing* **2008**, *38*, 963–981.
- (40) Barkoutsos, P. K.; Gonthier, J. F.; Sokolov, I.; Moll, N.; Salis, G.; Fuhrer, A.; Ganzhorn, M.; Egger, D. J.; Troyer, M.; Mezzacapo, A., et al. Quantum algorithms for electronic structure calculations: Particle-hole Hamiltonian and optimized wave-function expansions. *Phys. Rev. A* **2018**, *98*, 022322.
- (41) Romero, J.; Babbush, R.; McClean, J. R.; Hempel, C.; Love, P. J.; Aspuru-Guzik, A. Strategies for quantum computing molecular energies using the unitary coupled cluster ansatz. *Quantum Sci. Technol.* **2019**, *4*, 014008.
- (42) Wecker, D.; Hastings, M. B.; Troyer, M. Progress towards practical quantum variational algorithms. *Phys. Rev. A* **2015**, *92*, 042303.
- (43) Ryabinkin, I. G.; Yen, T.-C.; Genin, S. N.; Izmaylov, A. F. Qubit Coupled Cluster Method: A Systematic Approach to Quantum Chemistry on a Quantum Computer. *J. Chem. Theory Comput.* **2018**, *14*, 6317–6326.
- (44) 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.

- (45) 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 Comput.* **2020**, *16*, 1055–1063.
- (46) McClean, J. R.; Kimchi-Schwartz, M. E.; Carter, J.; de Jong, W. A. Hybrid quantum-classical hierarchy for mitigation of decoherence and determination of excited states. *Phys. Rev. A* **2017**, *95*, 042308.
- (47) Kitaev, A. Y. Quantum measurements and the Abelian stabilizer problem. *arXiv:9511026* **1995**,
- (48) Abrams, D. S.; Lloyd, S. Simulation of Many-Body Fermi Systems on a Universal Quantum Computer. *Phys. Rev. Lett.* **1997**, *79*, 2586–2589.
- (49) Abrams, D. S.; Lloyd, S. Quantum algorithm providing exponential speed increase for finding eigenvalues and eigenvectors. *Phys. Rev. Lett.* **1999**, *83*, 5162–5165.
- (50) Parrish, R. M.; McMahon, P. L. Quantum Filter Diagonalization: Quantum Eigendecomposition without Full Quantum Phase Estimation. *arXiv:1909.08925* **2019**,
- (51) Stair, N. H.; Huang, R.; Evangelista, F. A. A Multireference Quantum Krylov Algorithm for Strongly Correlated Electrons. *J. Chem. Theory Comput.* **2020**, *16*, 2236–2245.
- (52) Klymko, K.; Mejuto-Zaera, C.; Cotton, S. J.; Wudarski, F.; Urbanek, M.; Hait, D.; Head-Gordon, M.; Whaley, K. B.; Moussa, J.; Wiebe, N., et al. Real time evolution for ultracompact Hamiltonian eigenstates on quantum hardware. *arXiv:2103.08563* **2021**,
- (53) Stair, N. H.; Evangelista, F. A. Simulating Many-Body Systems with a Projective Quantum Eigensolver. *PRX Quantum* **2021**, *2*, 030301.
- (54) Smart, S. E.; Mazziotti, D. A. Quantum-classical hybrid algorithm using an error-mitigating N -representability condition to compute the Mott metal-insulator transition. *Phys. Rev. A* **2019**, *100*, 022517.

- (55) Smart, S. E.; Boyn, J.-N.; Mazziotti, D. A. Resolving Correlated States of Benzene on a Quantum Computer with an Error-Mitigated Quantum Contracted Eigenvalue Solver. 2021.
- (56) Boyn, J.-N.; Lykhin, A. O.; Smart, S. E.; Gagliardi, L.; Mazziotti, D. A. Quantum-Classical Hybrid Algorithm for the Simulation of All-Electron Correlation. 2021.
- (57) Jakob, W.; Rhinelander, J.; Moldovan, D. pybind11—Seamless operability between C++ 11 and Python. *URL: <https://github.com/pybind/pybind11>* **2017**,
- (58) Yung, M.-H.; Casanova, J.; Mezzacapo, A.; McClean, J.; Lamata, L.; Aspuru-Guzik, A.; Solano, E. From transistor to trapped-ion computers for quantum chemistry. *Sci. Rep.* **2014**, *4*, 3589.
- (59) Crawford, T. D.; Schaefer, H. F. An introduction to coupled cluster theory for computational chemists. *Rev. Comput. Chem.* **2000**, *14*, 33–136.
- (60) Jordan, P.; Wigner, E. P. *The Collected Works of Eugene Paul Wigner*; Springer, 1993; pp 109–129.
- (61) Bravyi, S. B.; Kitaev, A. Y. Fermionic quantum computation. *Ann. Phys.* **2002**, *298*, 210–226.
- (62) Havlíček, V.; Troyer, M.; Whitfield, J. D. Operator locality in the quantum simulation of fermionic models. *Phys. Rev. A* **2017**, *95*, 032332.
- (63) Setia, K.; Whitfield, J. D. Bravyi-Kitaev Superfast simulation of electronic structure on a quantum computer. *J. Chem. Phys.* **2018**, *148*, 164104.
- (64) Setia, K.; Bravyi, S.; Mezzacapo, A.; Whitfield, J. D. Superfast encodings for fermionic quantum simulation. *Phys. Rev. Res.* **2019**, *1*, 033033.
- (65) Cerezo, M.; Arrasmith, A.; Babbush, R.; Benjamin, S. C.; Endo, S.; Fujii, K.; McClean, J. R.; Mitarai, K.; Yuan, X.; Cincio, L., et al. Variational quantum algorithms. *arXiv preprint arXiv:2012.09265* **2020**,

- (66) Huggins, W. J.; Lee, J.; Baek, U.; O’Gorman, B.; Whaley, K. B. A non-orthogonal variational quantum eigensolver. *New J. Phys.* **2020**,
- (67) 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.
- (68) Szalay, P. G.; Nooijen, M.; Bartlett, R. J. Alternative ansätze in single reference coupled-cluster theory. III. A critical analysis of different methods. *J. Chem. Phys.* **1995**, *103*, 281–298.
- (69) Taube, A. G.; Bartlett, R. J. New perspectives on unitary coupled-cluster theory. *International journal of quantum chemistry* **2006**, *106*, 3393–3401.
- (70) Cooper, B.; Knowles, P. J. Benchmark studies of variational, unitary and extended coupled cluster methods. *J. Chem. Phys.* **2010**, *133*, 234102.
- (71) Evangelista, F. A. Alternative single-reference coupled cluster approaches for multireference problems: The simpler, the better. *J. Chem. Phys.* **2011**, *134*, 224102.
- (72) Harsha, G.; Shiozaki, T.; Scuseria, G. E. On the difference between variational and unitary coupled cluster theories. *J. Chem. Phys.* **2018**, *148*, 044107.
- (73) Schuld, M.; Bergholm, V.; Gogolin, C.; Izaac, J.; Killoran, N. Evaluating analytic gradients on quantum hardware. *Phys. Rev. A* **2019**, *99*, 032331.
- (74) Kottmann, J. S.; Anand, A.; Aspuru-Guzik, A. A Feasible Approach for Automatically Differentiable Unitary Coupled-Cluster on Quantum Computers. *arXiv preprint arXiv:2011.05938* **2020**,
- (75) Virtanen, P.; Gommers, R.; Oliphant, T. E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J., et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nat. Methods* **2020**, *17*, 261–272.
- (76) Trotter, H. F. On the product of semi-groups of operators. *Proc. Am. Math. Soc.* **1959**, *10*, 545–551.

- (77) Suzuki, M. Improved Trotter-like formula. *Phys. Lett. A* **1993**, *180*, 232–234.
- (78) Aharonov, D.; Jones, V.; Landau, Z. A polynomial quantum algorithm for approximating the Jones polynomial. *Algorithmica* **2009**, *55*, 395–421.
- (79) Limacher, P. A.; Ayers, P. W.; Johnson, P. A.; De Baerdemacker, S.; Van Neck, D.; Bultinck, P. A new mean-field method suitable for strongly correlated electrons: Computationally facile antisymmetric products of nonorthogonal geminals. *J. Chem. Theory Comput.* **2013**, *9*, 1394–1401.
- (80) Bulik, I. W.; Henderson, T. M.; Scuseria, G. E. Can single-reference coupled cluster theory describe static correlation? *J. Chem. Theory Comput.* **2015**, *11*, 3171–3179.
- (81) Broyden, C. G. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA J. Appl. Math.* **1970**, *6*, 76–90.
- (82) Fletcher, R. A new approach to variable metric algorithms. *Comput. J.* **1970**, *13*, 317–322.
- (83) Goldfarb, D. A family of variable-metric methods derived by variational means. *Math. Comput.* **1970**, *24*, 23–26.
- (84) Shanno, D. F. Conditioning of quasi-Newton methods for function minimization. *Math. Comput.* **1970**, *24*, 647–656.
- (85) Hastings, M. B.; Wecker, D.; Bauer, B.; Troyer, M. Improving Quantum Algorithms for Quantum Chemistry. *Quantum Info. Comput.* **2015**, *15*, 1–21.
- (86) Knowles, P. J.; Handy, N. C. A new determinant-based full configuration interaction method. *Chem. Phys. Lett.* **1984**, *111*, 315–321.
- (87) Lee, J.; Huggins, W. J.; Head-Gordon, M.; Whaley, K. B. Generalized unitary coupled cluster wave functions for quantum computation. *J. Chem. Theory Comput.* **2018**, *15*, 311–324.

- (88) Foss-Feig, M.; Hayes, D.; Dreiling, J. M.; Figgatt, C.; Gaebler, J. P.; Moses, S. A.; Pino, J. M.; Potter, A. C. Holographic quantum algorithms for simulating correlated spin systems. *Phys. Rev. Res.* **2021**, *3*, 033002.
- (89) Haghshenas, R.; Gray, J.; Potter, A. C.; Chan, G. K. The Variational Power of Quantum Circuit Tensor Networks. *arXiv preprint arXiv:2107.01307* **2021**,
- (90) Seki, K.; Yunoki, S. Quantum power method by a superposition of time-evolved states. *Phys. Rev. X Quantum* **2021**, *2*, 010333.
- (91) Ollitrault, P. J.; Kandala, A.; Chen, C.-F.; Barkoutsos, P. K.; Mezzacapo, A.; Pistoia, M.; Sheldon, S.; Woerner, S.; Gambetta, J. M.; Tavernelli, I. Quantum equation of motion for computing molecular excitation energies on a noisy quantum processor. *Phys. Rev. Res.* **2020**, *2*, 043140.
- (92) Bravyi, S.; Gambetta, J. M.; Mezzacapo, A.; Temme, K. Tapering off qubits to simulate fermionic Hamiltonians. *arXiv preprint arXiv:1701.08213* **2017**,
- (93) Huggins, W. J.; McClean, J. R.; Rubin, N. C.; Jiang, Z.; Wiebe, N.; Whaley, K. B.; Babbush, R. Efficient and noise resilient measurements for quantum chemistry on near-term quantum computers. *npj Quantum Inf.* **2021**, *7*, 1–9.

TOC Graphic

