

RESEARCH ARTICLE | OCTOBER 25 2024

Improved modularity and new features in *ipie*: Toward even larger AFQMC calculations on CPUs and GPUs at zero and finite temperatures

Special Collection: Modular and Interoperable Software for Chemical Physics

Tong Jiang  ; Moritz K. A. Baumgarten  ; Pierre-François Loos  ; Ankit Mahajan  ; Anthony Scemama  ; Shu Fay Ung  ; Jinghong Zhang  ; Fionn D. Malone; Joonho Lee  



J. Chem. Phys. 161, 162502 (2024)

<https://doi.org/10.1063/5.0225596>



View
Online



Export
Citation

Articles You May Be Interested In

KoopmanLab: Machine learning for solving complex physics equations

APL Mach. Learn. (September 2023)

Experimental realization of a quantum classification: Bell state measurement via machine learning

APL Mach. Learn. (September 2023)

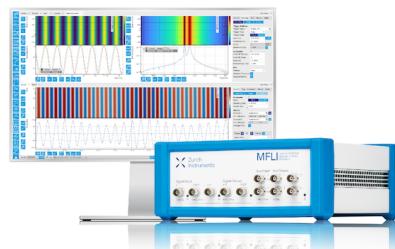
Challenge us.

What are your needs for periodic signal detection?



Zurich
Instruments

[Find out more](#)



Improved modularity and new features in ipie: Toward even larger AFQMC calculations on CPUs and GPUs at zero and finite temperatures

Cite as: J. Chem. Phys. 161, 162502 (2024); doi: 10.1063/5.0225596

Submitted: 25 June 2024 • Accepted: 4 October 2024 •

Published Online: 25 October 2024



View Online



Export Citation



CrossMark

Tong Jiang,¹ Moritz K. A. Baumgarten,¹ Pierre-François Loos,² Ankit Mahajan,³ Anthony Scemama,² Shu Fay Ung,³ Jinghong Zhang,¹ Fionn D. Malone,⁴ and Joonho Lee^{1,a)}

AFFILIATIONS

¹ Department of Chemistry and Chemical Biology, Harvard University, Cambridge, Massachusetts 02138, USA

² Laboratoire de Chimie et Physique Quantiques (UMR 5626), Université de Toulouse, CNRS, UPS, Toulouse, France

³ Department of Chemistry, Columbia University, New York, New York 10027, USA

⁴ Google Research, Venice, California 90291, USA

Note: This paper is part of the JCP Special Topic on Modular and Interoperable Software for Chemical Physics.

^{a)} Author to whom correspondence should be addressed: joonholee@g.harvard.edu

ABSTRACT

ipie is a Python-based auxiliary-field quantum Monte Carlo (AFQMC) package that has undergone substantial improvements since its initial release [Malone *et al.*, J. Chem. Theory Comput. **19**(1), 109–121 (2023)]. This paper outlines the improved modularity and new capabilities implemented in ipie. We highlight the ease of incorporating different trial and walker types and the seamless integration of ipie with external libraries. We enable distributed Hamiltonian simulations of large systems that otherwise would not fit on a single central processing unit node or graphics processing unit (GPU) card. This development enabled us to compute the interaction energy of a benzene dimer with 84 electrons and 1512 orbitals with multi-GPUs. Using CUDA and cupy for NVIDIA GPUs, ipie supports GPU-accelerated multi-slater determinant trial wavefunctions [Huang *et al.*, arXiv:2406.08314 (2024)] to enable efficient and highly accurate simulations of large-scale systems. This allows for near-exact ground state energies of multi-reference clusters, $[\text{Cu}_2\text{O}_2]^{2+}$ and $[\text{Fe}_2\text{S}_2(\text{SCH}_3)_4]^{2-}$. We also describe implementations of free projection AFQMC, finite temperature AFQMC, AFQMC for electron–phonon systems, and automatic differentiation in AFQMC for calculating physical properties. These advancements position ipie as a leading platform for AFQMC research in quantum chemistry, facilitating more complex and ambitious computational method development and their applications.

Published under an exclusive license by AIP Publishing. <https://doi.org/10.1063/5.0225596>

25 October 2024 10:32:22

I. INTRODUCTION

Auxiliary-field quantum Monte Carlo (AFQMC)^{1,2} has become increasingly popular in quantum chemistry^{3,4} and is furthermore being recognized as a useful approach in the context of quantum algorithms.^{5–11} A well-maintained AFQMC program with flexibility and robust performance will play a pivotal role at the intersection of many disciplines, including chemistry, physics, materials science, and quantum information science. Given the initial PAUXY development effort for rapid prototyping while lacking performance,^{12–16} the Python-based AFQMC program ipie was designed from scratch for high performance and ease of development. It was officially introduced as a production-level package in Ref. 17 under

the Apache License 2.0. ipie was optimized for high-performance computing architectures with both central and graphical processing units (CPUs and GPUs). High performance was largely achieved by integrating Numba’s JIT compilation¹⁸ to fine-tune the computational efficiency of specific kernels and MPI parallelism for effective distributed computing. The utility of ipie was showcased through the resolution of intricate quantum chemical challenges, notably the $[\text{Cu}_2\text{O}_2]^{2+}$ torture track.^{17,19} Rigorous benchmarks on CPU and GPU platforms position ipie competitively, displaying speed on par with—or surpassing—existing Python and C++ codes.^{17,20,21} Systematic timing benchmarks in Ref. 17 demonstrated the cost of typical AFQMC calculations using ipie, also compared to other available C++ codes. Since its release, ipie has gained popularity

beyond the original developer group, from academic researchers in quantum chemistry and physics to engineers in the technology sector who apply quantum simulations for material design and algorithm development.^{3,7,8,10,22–25}

In this article, we describe the recent development and current status of `ipie` (version v0.7.1),²⁶ introducing enhanced modularity, a suite of new features, interfaces to external packages, and associated numerical examples. Below are the key highlights.

A. High degree of modularity and customizability

The AFQMC driver has been restructured to be fully modular, allowing for a straightforward combination of features. This provides greater flexibility in adapting to a wide range of user demands. Advancements in AFQMC algorithms often focus on developing new trial wavefunctions to better control the fermionic sign/phase problem.^{5,27–29} The key routines in an AFQMC calculation compute intermediates using walker and trial wavefunctions, including overlap, force bias, Green's function, and energy estimators. Without altering the internal core code of `ipie`, users and developers can customize all components making up an AFQMC simulation, including trial wavefunctions, walkers, Hamiltonians, propagators, and estimators. The key objects are all structured using object-oriented programming (OOP) principles, facilitating straightforward customization through inheritance.

B. Development-friendly design

This improvement standardizes the component interfaces and workflow processes while offering a flexible system that adapts to various data types and user requirements. The abstract base classes serve as a foundational blueprint, ensuring all components adhere to a uniform structure and interact seamlessly. Complementing this, the factory methods for common workflows simplify the instantiation process, allowing users to set up standard calculations with minimal effort and reduced potential for errors. Integrating type-based dispatch through `Plum`³⁰ brings increased precision and efficiency in method handling, ensuring every component downstream dynamically adjusts its operations based on the specific trials and walkers requested.

Moreover, we have simplified the integration of `ipie` with external quantum chemistry packages. While the necessary integrals and orbitals for running AFQMC are most commonly obtained through an interface with PySCF,³¹ our simplified file format also ensures that other packages can be easily used. For sophisticated trial wavefunctions, such as multiple Slater determinant (MSD) trials derived from selected configuration interaction, interfaces with PySCF,³¹ Dice,^{21,32} and TREXIO³³ are available. An additional interface with the Fermionic Quantum Emulator (FQE)³⁴ has also been introduced, facilitating the conversion between `ipie`'s MSD wavefunction and quantum circuit wavefunctions, which facilitates AFQMC applications in the quantum information science (QIS) community.

C. New improvements and features

In the previous release paper,¹⁷ we introduced basic capabilities to perform phaseless AFQMC calculations with both single

Slater determinant and multi-Slater determinant trial wavefunctions. It supported CPU and GPU runs for AFQMC with single determinant trial wavefunctions, while only CPUs were supported for multi-Slater determinant calculations. In this release, several features have been added to `ipie` to improve memory management, accelerate performance, handle new problems, and enhance integration testing, among other advancements. Some notable features are as follows:

Tackling larger systems: To manage the high storage requirement of Cholesky vectors [i.e., $\mathcal{O}(N^3)$], `ipie` offers shared memory across MPI processes on the same node and distributed-memory among CPUs and GPUs.

Faster calculations for systems with multireference character: `ipie` adds support for GPU-accelerated AFQMC calculation with multi-Slater determinant trials, which enables more efficient calculations of the ground state of systems with multireference characters, for example, bond breaking problems³ and strongly correlated systems such as transition metal complexes.^{17,19,22,35}

New AFQMC developments: While primarily developed for phaseless AFQMC (ph-AFQMC) targeting *ab initio* systems, `ipie` also supports additional AFQMC methods, including free-projection AFQMC,²⁷ finite temperature AFQMC,¹⁴ AFQMC for coupled electron–phonon models,¹⁵ and automatic differentiation within AFQMC for calculating observables that do not commute with the Hamiltonian.³⁶ We also support complex-valued Hamiltonians that will be useful for performing prototypical solid-state calculations.

Integrated testing: `ipie` is equipped with improved integration testing, boosting the package's robustness and adaptability and significantly enhancing its reliability and usability for end-users and developers.

The organization of this paper is as follows: Sec. II overviews the theory of AFQMC; Sec. III details the components, software architecture, and workflow of `ipie`, including examples to illustrate the framework's adaptability for AFQMC development; Sec. IV introduces new features in `ipie` and provides corresponding examples; Sec. V outlines the interfaces to external packages; and Sec. VI concludes with a summary and outlook.

II. THEORY OF AFQMC

AFQMC is based on the following imaginary time evolution:

$$|\Psi_0\rangle \propto \lim_{\tau \rightarrow \infty} \exp(-\tau \hat{H}) |\Phi_0\rangle = \lim_{n \rightarrow \infty} (\exp(-\Delta\tau \hat{H}))^n |\Phi_0\rangle, \quad (1)$$

where $\Delta\tau$ is an infinitesimal time step, $|\Psi_0\rangle$ is the ground state wavefunction, and $|\Phi_0\rangle$ is an initial state satisfying $\langle \Phi_0 | \Psi_0 \rangle \neq 0$. While `ipie` supports some of the prototypical model Hamiltonians, its development has focused on the simulation of the *ab initio* Hamiltonian, which in second quantization is given by

$$\hat{H} = \sum_{p,q=1}^N h_{pq} \hat{a}_p^\dagger \hat{a}_q + \frac{1}{2} \sum_{p,q,r,s=1}^N g_{psqr} \hat{a}_p^\dagger \hat{a}_q^\dagger \hat{a}_r \hat{a}_s, \quad (2)$$

where the two-electron repulsion integral (ERI) is factorized with the Cholesky decomposition

$$g_{psqr} = (ps|qr) = \sum_{\gamma=1}^{N_y} L_{ps}^{\gamma} L_{qr}^{\gamma}. \quad (3)$$

With this factorization, we have

$$\hat{H} = \hat{v}_0 - \frac{1}{2} \sum_{\gamma=1}^{N_y} \hat{v}_{\gamma}^2, \quad (4)$$

where

$$\hat{v}_0 = \sum_{pq} \left[h_{pq} - \frac{1}{2} \sum_r (pr|rq) \right] \hat{a}_p^{\dagger} \hat{a}_q, \quad (5)$$

$$\hat{v}_{\gamma} = i \sum_{pq} L_{pq}^{\gamma} \hat{a}_p^{\dagger} \hat{a}_q. \quad (6)$$

The short-time propagator with Trotter decomposition is written as

$$e^{-\Delta\tau\hat{H}} = e^{-\frac{\Delta\tau}{2}\hat{v}_0} e^{\frac{\Delta\tau}{2}\sum\hat{v}_{\gamma}^2} e^{-\frac{\Delta\tau}{2}\hat{v}_0} + \mathcal{O}(\Delta\tau^3). \quad (7)$$

Upon applying the Hubbard–Stratonovich transformation,^{37,38} our effective propagator contains only one-body operators,

$$e^{-\Delta\tau\hat{H}} = \int d\mathbf{x} p(\mathbf{x}) \hat{B}(\mathbf{x}, \Delta\tau) + \mathcal{O}(\Delta\tau^2), \quad (8)$$

where $p(\mathbf{x})$ is the standard Gaussian distribution, $\mathbf{x} = (x_1, x_2, \dots, x_{N_y})$ are the auxiliary fields, and the one-body propagator \hat{B} is

$$\hat{B}(\mathbf{x}, \Delta\tau) = e^{-\frac{\Delta\tau}{2}\hat{v}_0} e^{-\sqrt{\Delta\tau}\mathbf{x} \cdot \hat{\mathbf{v}}} e^{-\frac{\Delta\tau}{2}\hat{v}_0}, \quad (9)$$

where $\hat{\mathbf{v}} = (\hat{v}_1, \hat{v}_2, \dots, \hat{v}_{N_y})$.

In AFQMC, each walker samples auxiliary fields and represents a statistical sample of the global wavefunction at imaginary time τ , written as

$$|\Psi(\tau)\rangle = \sum_i^{N_w} w_i(\tau) \frac{|\psi_i(\tau)\rangle}{\langle \Psi_T | \psi_i(\tau) \rangle}, \quad (10)$$

where $|\psi_i(\tau)\rangle$ is the wavefunction of the i th walker at time τ and $|\Psi_T\rangle$ is the trial wavefunction used for importance sampling. The energy estimator is then

$$E(\tau) = \sum_i^{N_w} w_i(\tau) E_{loc,i}(\tau) = \sum_i^{N_w} w_i(\tau) \frac{\langle \Psi_T | \hat{H} | \psi_i(\tau) \rangle}{\langle \Psi_T | \psi_i(\tau) \rangle}. \quad (11)$$

The walker state $|\psi_i(\tau)\rangle$ is updated by applying the discrete-time propagator, and the weight $w_i(\tau)$ is updated following the phaseless approximation:²

$$|\psi_i(\tau + \Delta\tau)\rangle = \hat{B}(\mathbf{x}_i - \bar{\mathbf{x}}_i, \Delta\tau) |\psi_i(\tau)\rangle, \quad (12)$$

$$w_i(\tau + \Delta\tau) = I_{ph}(\mathbf{x}_i, \bar{\mathbf{x}}_i, \tau, \Delta\tau) \times w_i(\tau). \quad (13)$$

We dynamically shift the distribution of auxiliary fields using the force bias $\bar{\mathbf{x}}_i$ defined by

$$\bar{\mathbf{x}}_i(\Delta\tau, \tau) = -\sqrt{\Delta\tau} \frac{\langle \Psi_T | \hat{\mathbf{v}} - \langle \hat{\mathbf{v}} \rangle_T | \psi_i(\tau) \rangle}{\langle \Psi_T | \psi_i(\tau) \rangle}, \quad (14)$$

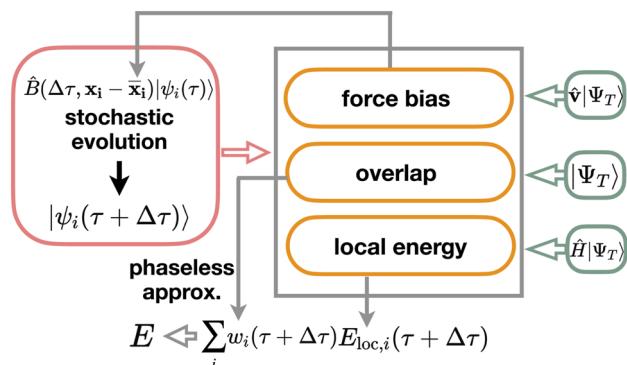


FIG. 1. Overview of phaseless AFQMC in terms of computational hotspots.

where $\langle \hat{\mathbf{v}} \rangle_T$ is usually called the mean-field shift in the AFQMC literature.⁴ Furthermore, the phaseless importance function^{1,2} used in the weight update (13) is given by

$$I_{ph}(\mathbf{x}_i, \bar{\mathbf{x}}_i, \tau, \Delta\tau) = |I(\mathbf{x}_i, \bar{\mathbf{x}}_i, \tau, \Delta\tau)| \times \max[0, \cos(\theta_i(\tau))], \quad (15)$$

which remains real and positive throughout the propagation. The so-called hybrid importance function is given by

$$I(\mathbf{x}_i, \bar{\mathbf{x}}_i, \tau, \Delta\tau) = S_i(\tau, \Delta\tau) e^{\mathbf{x}_i \cdot \bar{\mathbf{x}}_i - \bar{\mathbf{x}}_i \cdot \bar{\mathbf{x}}_i / 2}, \quad (16)$$

and the overlap ratio of the i th walker is

$$S_i(\tau, \Delta\tau) = \frac{\langle \Psi_T | \hat{B}(\Delta\tau, \mathbf{x}_i - \bar{\mathbf{x}}_i) | \psi_i(\tau) \rangle}{\langle \Psi_T | \psi_i(\tau) \rangle}. \quad (17)$$

We define the phase of the overlap as

$$\theta_i(\tau) = \arg[S_i(\tau, \Delta\tau)]. \quad (18)$$

The computation of local energies in Eq. (11), propagation of wavefunctions in Eq. (12), and evaluation of force biases in Eq. (14) are the primary computational hotspots in AFQMC calculations. We optimize these routines by employing optimized algorithms and adopting high-performance computing techniques. We provide an overview of the phaseless AFQMC algorithm described earlier in Fig. 1.

III. SOFTWARE ARCHITECTURE AND DESIGN PRINCIPLES

A. AFQMC driver

We overview the software architecture of ipie with emphasis on improved modularity. As shown in Fig. 2, ipie generates an AFQMC simulation by assembling the generic AFQMC driver from problem-specific components and then running it to execute the simulation. The driver requires several inputs to define the QMC simulation, including the trial wavefunction, walkers, and the Hamiltonian, as well as more generic parameters such as the number of blocks and time step. The following Python code snippet shows how the driver can be instantiated:

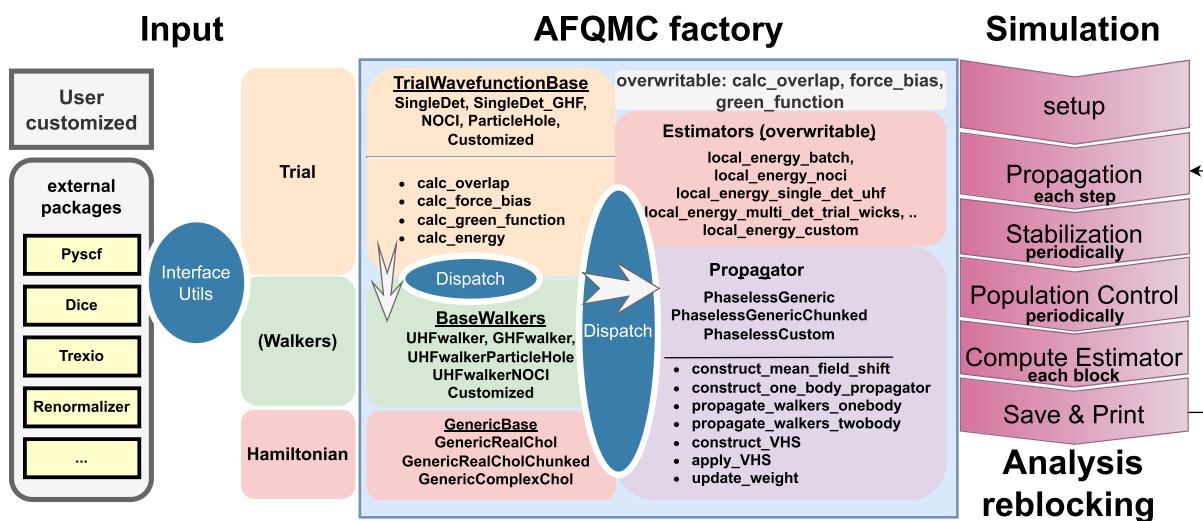


FIG. 2. The workflow of ipie.

Code Snippet 1 The AFQMC class within ipie.

```
class AFQMC(object):
    def __init__(self, system, hamiltonian, trial,
                 walkers, propagator, mpi_handler, params:
                 QMCPParams):
        ....
```

The code takes several objects needed for a typical AFQMC driver as follows:

- (1) System contains information that defines the problem, including the number of spin-up and spin-down electrons.
- (2) Hamiltonian stores molecular integrals as detailed in Sec. III B.
- (3) Trial represents a trial wavefunction object, for which ipie provides many options as detailed in Sec. III C.
- (4) Walkers manages all information about walkers during the imaginary time propagation. This includes the initial walker wavefunctions and the number of walkers, among other details. ipie allows walkers to be explicitly passed into the AFQMC driver or dynamically dispatched with default settings, as detailed in Sec. III D.
- (5) Propagator handles the update of walker weights and wavefunctions during the imaginary time evolution. ipie's AFQMC driver can take the propagator class as an input or dispatch it internally with default settings.
- (6) mpi_handler manages information related to the Message Passing Interface (MPI) for parallel computing, including rank details, chunking, and groups (i.e., a group is a collection of MPI processes). The parameter shared_comm in mpi_handler refers to the MPI rank within a group where the chunking integrals are distributed. The specifics of these are elaborated in Sec. IV A.
- (7) Params encompass other fundamental parameters for the QMC simulation, such as the time step.

While users can directly provide all of the aforementioned inputs to construct the AFQMC driver object, especially for development purposes, a factory method is provided to simplify the process greatly. This is especially useful for standard AFQMC calculations using single or multiple Slater determinant trials. This method significantly reduces the number of inputs required to construct the AFQMC object, as demonstrated in Code Snippet 2.

Code Snippet 2 Factory method for AFQMC driver.

```
def build(
    num_elec: Tuple[int, int],
    hamiltonian,
    trial_wavefunction
) -> "AFQMC":
    ....
```

As illustrated in Code Snippet 2 and Fig. 2, the construction of the AFQMC driver ultimately relies on two key inputs: the Hamiltonian and the trial wavefunction. The Hamiltonian object is assembled with one-electron integrals and the Cholesky decomposition (or density fitting) of two-electron integrals, detailed in Sec. III B. As for trial wavefunctions, ipie offers various options that utilize interfaces with external packages such as PySCF,³¹ Dice,²¹ and TREXIO³³ to facilitate standardized workflows. ipie can also accommodate user-customized trials, which offers flexibility in future developments. Benefiting from the flexibility of ipie, we are also able to incorporate more complicated trial wavefunctions, such as density matrix renormalization group (DMRG), and in Sec. III C we show the AFQMC calculation with matrix product states (MPS) trials by incorporating ipie with the DMRG package Renormalizer.³⁹

The build method in the AFQMC class is designed to streamline the setup of an AFQMC calculation by requiring only three inputs: the number of electrons, the Hamiltonian, and the trial wavefunction. Its main purpose is automating the

construction of key components such as walkers, propagators, estimators, etc. Users need only specify the hamiltonian and trial inputs `trial_wavefunction`, with the build method handling downstream instantiation of the rest.

Once the driver is built, wavefunction propagation, stabilization, population control, and estimator calculation are performed with the provided QMC parameters. Simulation outputs in both text file and hdf5 file formats are saved and updated after each block, which enables real-time reblocking analysis⁴⁰ via built-in tools.

B. Hamiltonian

The construction of a Hamiltonian object can be achieved in several ways. One can construct the object directly by inputting the one-electron integrals and the Cholesky decomposition [Eq. (3)] of either 8-fold (for real symmetric integrals) or 4-fold (for complex hermitian integrals) symmetric two-body integrals.

Code Snippet 3 Constructing the Hamiltonian object with provided electron integrals

```
from ipie.hamiltonians.generic import Generic as
    → HamGeneric
ham = HamGeneric(h1e, chol, ecore)
```

where `h1e`, `chol`, and `ecore` are the one-electron integrals, the Cholesky decomposition [Eq. (6)] of the two-electron integrals within the desired orbitals in Eq. (2), and the nuclear repulsion energy, respectively.

`ipie` also offers built-in functions for assembling the Hamiltonian object from PySCF calculations. One can provide (i) a PySCF check file, (ii) a PySCF `mol` object together with the molecular orbital (MO) coefficient matrix `mo_coeff`, or (iii) a `mol` object, `mo_coeff`, and a basis transformation matrix `X` that transforms MOs to orthogonal atomic orbitals (OAO), natural orbitals, and so on, depending on the single particle basis used in the AFQMC calculation.

Code Snippet 4 Constructing the Hamiltonian object from PySCF.

```
from ipie.utils.from_pyscf import generate_hamiltonian,
    → generate_hamiltonian_from_chk

# OAO basis
ham = generate_hamiltonian_from_chk(
    'scf.chk', use_mcscf=False,
    chol_cut=1e-5, num_frozen_core=0, ortho_ao=True
)

# MO basis
ham = generate_hamiltonian(mol, mo_coeff, h1e, mo_coeff,
    chol_cut=1e-5, num_frozen_core=0)
# basis with transform matrix X
ham = generate_hamiltonian(mol, mo_coeff, h1e, X,
    chol_cut=1e-5)
```

The argument `chol_cut` specifies the cutoff for the Cholesky decomposition; `use_mcscf` specifies whether to use the multi-configurational self-consistent field (MCSCF) MO coefficients; and

`num_frozen_core` specifies the number of frozen cores for the subsequent AFQMC calculations.

Given that the size of Cholesky vectors scales as $\mathcal{O}(N^3)$, with N as the number of orbitals, large systems with large basis sets can generate electron integrals of considerable size. For such systems, one cannot store copies of integrals for each MPI process and must resort to other strategies. Utilizing shared memory across different MPI processes, `ipie` offers an approach that allows processes on the same node to access Cholesky vectors in the same memory block. The following code snippet can be adopted to ensure the use of shared memory Hamiltonian objects:

Code Snippet 5 Constructing the Hamiltonian object with shared memory.

```
from ipie.utils.mpi import get_shared_comm
from ipie.hamiltonians.utils import get_hamiltonian
from mpi4py import MPI
shared_comm = get_shared_comm(MPI.COMM_WORLD)
ham = get_hamiltonian("ham.h5", comm=shared_comm,
    pack_chol=True)
```

This reads the integrals from the `ham.h5` file and generates the Hamiltonian object with shared integrals across all MPI processes. `pack_chol` refers to another strategy used by `ipie` to reduce memory usage by a factor of two and speed up the propagation. Namely, the permutation symmetry of Cholesky vectors is exploited to utilize only the upper-triangular part of the Cholesky vectors. Propagation using the packed matrix contracts only the symmetric part in Eq. (12) and is thus more efficient.

However, the shared memory strategy often creates a memory access overhead and reduces the MPI parallel efficiency when many cores are used. We also enable a chunking (or distributed-memory) strategy. The Cholesky vectors are divided into segments, or “chunked,” and distributed across different MPI processes to optimize memory usage and computational efficiency. The Cholesky vectors are divided into several chunks, each of which will be allocated to separate MPI processes. The specifics of this chunking process and the associated MPI communication strategy will be thoroughly explained in Sec. IV A. Given enough nodes, our approach completely removes the memory bottleneck in AFQMC and can work on both CPU and GPU nodes. We especially recommend our distributed-memory implementation for GPUs, where memory capacity is much more limited than that of CPUs.

C. Trial wavefunctions

Any new trial wavefunction may inherit from the `TrialWavefunctionBase` class, which contains methods to calculate essential quantities including the overlap [Eq. (17)], force bias [Eq. (14)], etc., as shown in Fig. 3. Some trials, such as single/multiple Slater determinants, also require computation of Green’s function and half rotation of electron repulsion integrals.¹³ There is flexibility to either inherit these functions directly or overwrite them with customized implementations, ensuring both consistency in fundamental operations and adaptability for specialized needs.

Single Slater determinant (SD) trials can be generated with

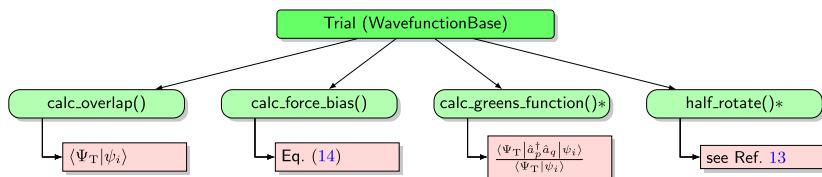


FIG. 3. The functions to be defined for a trial object. *: The function is optional depending on the trial type and is used in certain trials for computations like overlap, force bias, or local energy.

Code Snippet 6 The single determinant trial object.

```
from ipie.trial_wavefunction.single_det import SingleDet
trial = SingleDet(np.hstack([orbs_a, orbs_b]), nelec,
                  num_basis)
```

where `orbs_a` and `orbs_b` are the spin-up and spin-down coefficient matrices for the SD trials. Similarly, MSD trials can be generated with

Code Snippet 7 The multiple Slater determinant trial object.

```
from ipie.trial_wavefunction.particle_hole import
    ParticleHole
trial = ParticleHole(wfn, nelec, num_basis)
```

with `wfn` containing the occupation indices in the basis used in the configuration interaction (CI) expansion of the MSD and their coefficients.

A new customized trial can be defined straightforwardly in `ipie`. For instance, we consider a “noisy” SD trial that can be inherited from our existing SD trial class. The noisy trial adds Gaussian noise to the overlap evaluation and keeps everything else the same as for regular SD trials. The following code snippet achieves this:

Code Snippet 8 Example of constructing a custom noisy trial class.

```
class NoisySingleDet(SingleDet):
    def __init__(self, wavefunction, num_elec,
                 num_basis, noise_level=1e-12):
        super().__init__(wavefunction, num_elec,
                        num_basis)
        self._noise_level = noise_level
    def calc_overlap(self, walkers) -> np.ndarray:
        ovlp = super().calc_overlap(walkers)
        noise = np.random.normal(
            scale=self._noise_level,
            size=ovlp.size
        )
        return ovlp * (1 + noise)
```

Recently, we have also implemented a more sophisticated extension using a matrix product state (MPS) trial wavefunction.²⁵ MPS is the variational ansatz used in the density matrix renormalization group (DMRG) algorithm, and we obtain the MPS solution from external DMRG packages such as `Renormalizer`.³⁹ The theory explaining the structure of the MPS trial and associated functions is detailed in Ref. 25. A representative calculation is shown in Fig. 4.

D. Dispatchers of walkers, propagators, and estimators

As the program grows, additional complexity arises from the need to dispatch different features for arbitrary combinations of smaller building blocks. While there are numerous ways to achieve this, `ipie` previously relied on conditional `if` statements at the computational subroutines, which, despite its simplicity, will result in many intractable conditions as the number of features increases. The latest release of `ipie` thus utilizes the multiple dispatch library `Plum`³⁰ to dispatch features for different combinations of objects in a compact and flexible manner. This can be especially useful in AFQMC, where the behavior of many functions varies based on the type of trial, walkers, and Hamiltonians. In Fig. 5, we illustrate the dispatcher for walkers depending on the type of trial. As for `ipie`’s native trial wavefunctions, including both single and multiple (orthogonal and non-orthogonal) Slater determinants, once the trial and Hamiltonian objects are in place, the AFQMC driver can be formed. In instances where the walker object is not explicitly provided, the AFQMC driver constructs it inside the build function based on the type of the trial wavefunction via `Plum`. The appropriate energy estimators and propagators are also built according to the

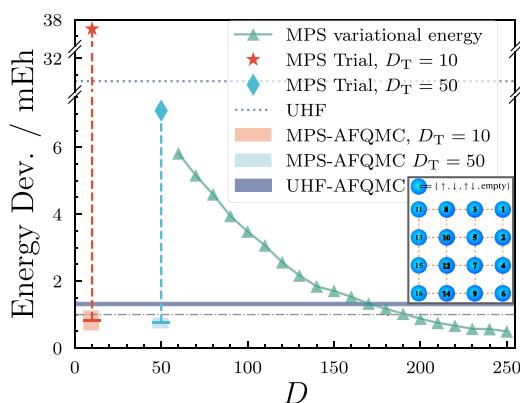


FIG. 4. Application of MPS-AFQMC to a two-dimensional hydrogen lattice with $r = 4.2a_0$. Energy deviation from exact results using MPS-AFQMC, UHF-AFQMC, and DMRG with different bond dimensions.²⁵

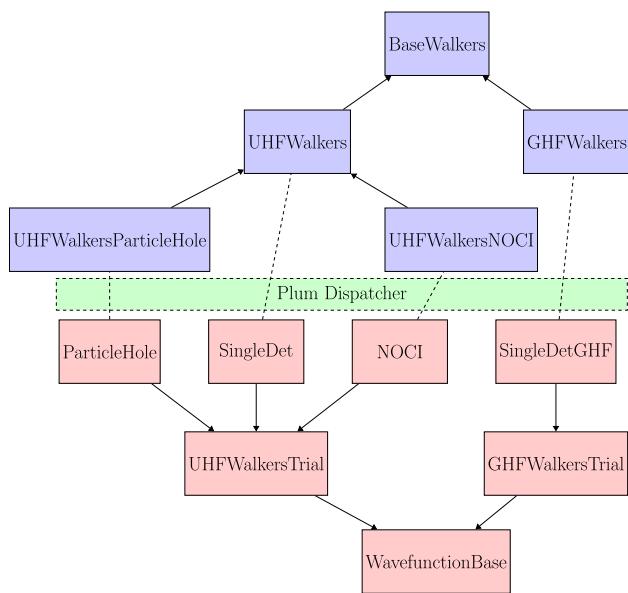


FIG. 5. Dispatcher for different walkers depending on the trial. Lines with arrows indicate class inheritance, while dashed lines signify the use of the Plum dispatcher for dispatching.

object type of the trial, walkers, and Hamiltonian, completing the necessary initialization for AFQMC calculations.

E. Estimators

ipie supports well-optimized energy estimators for wavefunctions, including SD,¹³ MSD,⁴¹ and non-orthogonal configuration interaction (NOCI) trials. It supports real and complex Hamiltonians on both CPUs and GPUs. As mentioned in Sec. III D, dispatchers can automatically determine a specific implementation of the estimator required for given user inputs. Users can also compose customized estimators for new types of trial wavefunctions. Considering the example in Code Snippet 8, we may define the corresponding energy estimator as

Code Snippet 9 Customized noisy estimator.

```

class NoisyEnergyEstimator(EnergyEstimator):
    def __init__(self, system, ham, trial):
        super().__init__(system=system, ham=ham,
                         trial=trial)
    def compute_estimator(self, system, walkers,
                          hamiltonian, trial, istep=1):
        trial.calc_greens_function(walkers)
        energy = local_energy_batch(system, hamiltonian,
                                    walkers, trial)
        self._data["ENumer"] = xp.sum(walkers.weight *
                                      energy[:, 0].real)
        self._data["EDenom"] = xp.sum(walkers.weight)
        self._data["E1Body"] = xp.sum(walkers.weight *
                                      energy[:, 1].real)
        self._data["E2Body"] = xp.sum(walkers.weight *
                                      energy[:, 2].real)
        return self._data
  
```

where `xp` serves as an abstract linear algebra backend, facilitating seamless integration of either NumPy for pure CPU-based computations or CuPy for GPU-accelerated tasks. In this estimator, we still evaluate the local energy as we do for any SD trial. This customized estimator can be passed to the AFQMC driver simply with

Code Snippet 10 Adding the customized noisy estimator to the AFQMC driver.

```

add_est = {"energy":
           NoisyEnergyEstimator(Generic(mol.nelec), ham, trial)
          }
afqmc.run(additional_estimators=add_est)
  
```

For more sophisticated cases such as MPS trials, in addition to the usual energy estimator, we can furthermore define other estimators such as one to measure the bond dimension of walkers when converted into an MPS.²⁵

IV. NEW FEATURES

Many new features have been added to ipie since our first release paper;¹⁷ we highlight some of the more important ones in this section.

A. Distributed Hamiltonian for limited memory

When using GPUs, we routinely encounter cases where the Cholesky vectors cannot entirely fit into a single GPU card's memory. Or one could imagine a large system where a single CPU node cannot fit the Cholesky vectors in memory anymore. In such circumstances, the Cholesky vectors are distributed over several GPU cards or CPU nodes that constitute a group. Each group stores the full Cholesky tensor as shown in Fig. 6(a). Whenever the tensor is required by the computational subroutines, for example, in computing the force bias or local energy, the walkers in each member of the group are communicated to other members in the same group, as illustrated in Fig. 6(b). We employ a cyclic data passing scheme of walkers among MPI processes to apply all existing Cholesky chunks to given walkers. With this strategy, one can accumulate locally contracted quantities before moving to the next QMC time step.

We performed timing benchmarks as an application by computing the parallel-displaced benzene dimer, a van der Waals complex taken from the S22 dataset.⁴² Disk usage for the benzene dimer calculations varies with the choice of basis set: 11 GB for the aug-cc-pVTZ basis and 67 GB for the aug-cc-pVQZ basis with a Cholesky decomposition threshold of 10^{-3} . The threshold is set based on the Hartree-Fock energy error around typical density fitting errors⁴³ ($<50 \mu E_h$ per atom here). We used our shared memory framework for CPU calculations and distributed Cholesky vectors and half-rotated integrals¹³ across multiple cards were used for GPU computations.

We analyze computational wall time as a function of the configuration (`nmembers` \times `ngroups`) in Fig. 7(a), where we also compare the GPU implementation against the CPU setup. We denote the number of cards over which the Cholesky vectors are distributed as `nmembers`, and every `nmembers` card forms a group, as described in

Fig. 6(a). We denote the number of groups as `nGroups`. For benzene dimer with aug-cc-pVTZ, a single A100 GPU card achieves performance around tenfold faster than 36 CPU cores. Comparisons between configurations of (1×2) vs (2×1) and (2×2) vs (4×1) , with an equivalent total number of GPU cards, demonstrated that configurations with a greater number of `nMembers` were slightly slower due to the increased communication cost that scales linearly with `nMembers`. We show how the communication within a group occurs in **Fig. 6(b)**. As shown in **Fig. 7(a)**, distributing the Hamiltonian across additional GPUs slightly slows down the performance, predominantly during the VHS step, which involves the construction of two-body propagators (where HS stands for Hubbard–Stratonovich) during propagation. A (2×2) setup shows $2\times$ faster speed compared to a (2×1) configuration, reaffirming that the distributed memory approach retains good parallel efficiency. While communication overhead is an inherent challenge in the distributed memory setup, this example shows the utility of our distributed-memory implementation.

Using our GPU implementation, we calculated the interaction energy of the benzene dimer with counterpoise correction.⁴⁸ We compared AFQMC results with those from other methodologies, as shown in **Figs. 7(b)** and **7(c)**. While coupled cluster with singles and doubles (CCSD) underestimates the binding energy, second-order Møller–Plesset perturbation theory overestimates. Crucially, we observed that RHF-AFQMC is on par with CCSD with perturbative triples [CCSD(T)]. We used 1224 walkers for this calculation, which were found to have negligible population control biases. In general, one should perform a convergence test to determine the number of walkers needed, as is standard in QMC calculations.³ We obtained more than 30 000 total blocks for both dimer and monomer calculations to compute the interaction energy with the error bars shown in **Figs. 7(b)** and **7(c)**. The raw data for all blocks can be obtained from the data repository in Data Availability. Such a large

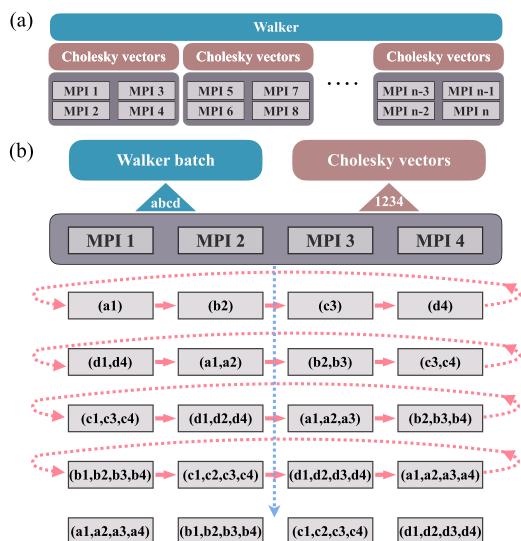


FIG. 6. (a) Distribution of walkers and the chunked Hamiltonian over multiple groups. (b) MPI communication between MPI processes within a group with the distributed Hamiltonian and walker batches.

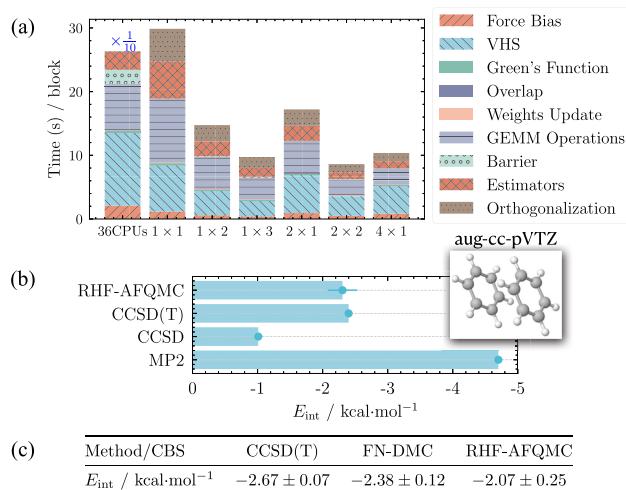


FIG. 7. (a) Timing benchmarks for AFQMC calculations on the benzene dimer with the aug-cc-pVTZ basis employing 36 CPUs with shared memory and multiple A100 GPUs with the distributed Hamiltonian. The data were obtained by averaging 100 blocks, each containing 25 steps with a time step of $0.005\text{E}_{\text{h}}^{-1}$. The interaction energy of the dimer computed from AFQMC is compared against other methods using the aug-cc-pVTZ basis in (b) and in the complete basis set limit obtained from a two-point basis extrapolation⁴⁴ from aug-cc-pVTZ and aug-cc-pVQZ calculations in (c). The MP2, CCSD, and CCSD(T) data using the aug-cc-pVTZ basis in (b) were obtained from Refs. 45 and 46. The CCSD(T) and FN-DMC data in (c) were extracted from Ref. 47. All AFQMC results were obtained with 1224 total walkers. The CPU calculations employed an Intel® Xeon® Platinum 8268 CPU @ 2.90 GHz. The GPU calculations employed an NVIDIA A100 GPU, accompanied by an Intel Xeon Platinum 8358 CPU @ 2.60 GHz.

number of blocks is generally unusual for AFQMC calculations; in the rest of the calculations shown in this paper, a few thousand blocks are sufficient to achieve the desired accuracy.

B. Support of complex Cholesky vectors

Most calculations in quantum chemistry deal with real-valued integrals. However, in some cases, especially in the presence of external magnetic fields, when considering spin-orbit coupling or employing Bloch orbitals in solid-state calculations, the wavefunctions (and orbitals) can become complex-valued.⁴⁹ This adds a layer of mathematical and computational complexity. To handle these cases, ipie supports complex Cholesky vectors; the number of auxiliary fields is hence twice as many as the number of Cholesky vectors. The implementation details are available in Ref. 50.

C. GPU accelerated MSD-AFQMC and timing benchmarks

ph-AFQMC with MSD trials is a powerful tool for systems with multi-reference characteristics, especially when the phaseless bias can be converged away. Building on top of our successful CPU implementation¹⁷ based on Wick's theorem,⁴¹ as detailed in Ref. 51, ipie has now enabled GPU-accelerated MSD-AFQMC implementations. Here, we summarize some of the important features of our implementation and discuss applications of this feature. Interested readers are referred to Ref. 51 for more details.

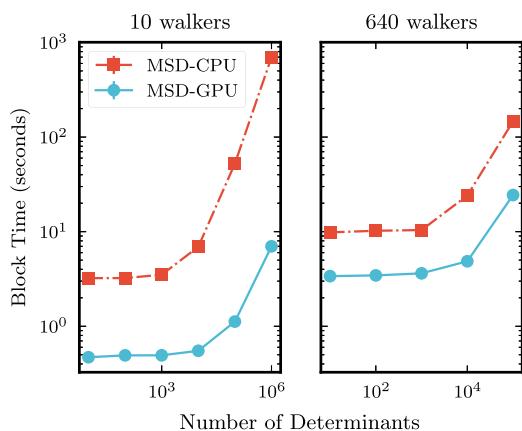


FIG. 8. (a) Comparisons of the time cost per block for AFQMC calculations on $[\text{Cu}_2\text{O}_2]^{2+}$ using the BS1 basis¹⁷ with 10 walkers on a single CPU core and a single A100 GPU. The CPU calculations employed an Intel Xeon Platinum 8336C CPU @ 2.30 GHz. (b) Time per block with 640 walkers on a single A100 against 32 CPU cores. Figure adapted from Ref. 51.

Similar to our single determinant GPU implementation, ipie employs `cupy.einsum` via the cuTENSOR library to enhance matrix multiplication performance. Additionally, Wick's algorithm is adapted for GPU execution using custom CUDA kernels, effectively speeding up the computational hotspots in MSD-AFQMC, such as Green's function, overlap, and local energy computations.

The GPU implementation of MSD-AFQMC is benchmarked on the example $[\text{Cu}_2\text{O}_2]^{2+}$ from ipie's first release paper.¹⁷ In addition to employing a total of 10 walkers to directly compare with our previous results in Ref. 17, calculations using a total of 640 walkers were also performed to obtain a more realistic comparison. With 10 walkers, the GPU implementation on a single NVIDIA A100 card is six times faster than using a single CPU core when the number of determinants in the trial is less than 10^3 . As the number of determinants increases, the performance improvement is even more pronounced: tenfold with 10^4 determinants and ~ 100 -fold with 10^6 determinants, as illustrated in Fig. 8(a). With 640 walkers, the GPU code on a single A100 card is compared against 32 CPU cores. As shown in Fig. 8(b), a fourfold speedup is observed with less than 10^2 determinants that increase to around sixfold with more determinants. The determinants were divided into chunks and computed sequentially to handle a larger number of determinants within the memory limits. This approach introduces an overhead that mostly affects the energy estimator, as reported in Ref. 51.

The timing and absolute energy benchmarks are further benchmarked on the (20o, 30e) active space of the $[\text{Fe}_2\text{S}_2(\text{SCH}_3)_4]^{2-}$ cluster,⁵¹ as shown in Fig. 9. In contrast to the 108-orbital calculation of $[\text{Cu}_2\text{O}_2]^{2+}$, a merely two-fold speedup is observed using a single A100 card compared to 32 CPUs. This speedup increases almost tenfold when the number of determinants exceeds 10^4 . As shown in Fig. 9(b), another observation is that using natural orbitals for AFQMC calculation leads to chemical accuracy with around 3×10^5 determinants in the trial. In contrast, reaching chemical accuracy is more challenging when using localized orbitals employed for DMRG calculations.³⁵ These observations highlight the critical

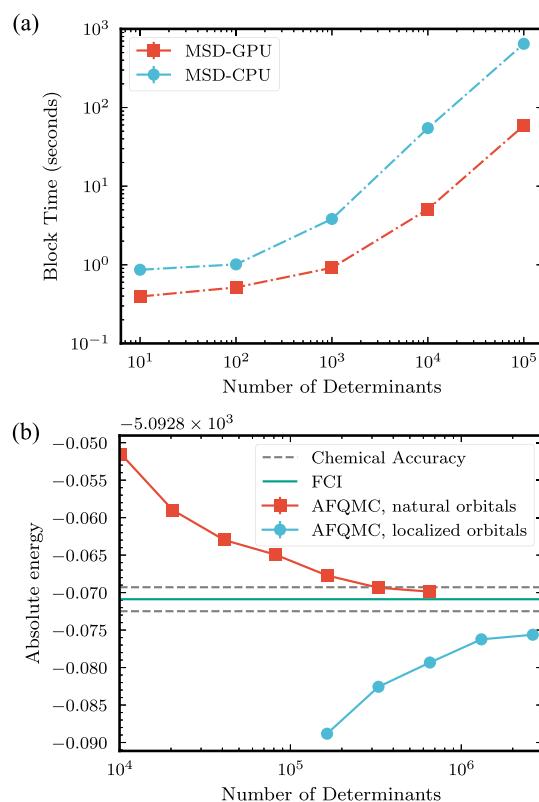


FIG. 9. Time cost and absolute energy benchmarks on the $[\text{Fe}_2\text{S}_2(\text{SCH}_3)_4]^{2-}$ cluster. (a) The time cost per block with 640 walkers on 32 CPU cores and a single A100 GPU. The CPU calculation employed an Intel Xeon Platinum 8336C CPU @ 2.30 GHz. (b) Absolute energies are derived using localized atomic orbitals and natural orbitals. Figure adapted from Ref. 51.

need to select an appropriate set of orbitals for AFQMC calculations involving strongly correlated systems. Additionally, it is noteworthy that the energies obtained using localized orbitals with a limited number of determinants are initially lower than the FCI energy, and they tend to converge upward toward the FCI reference as the number of determinants increases.

D. Free projection AFQMC

Free projection (fp-) AFQMC is a numerically exact method for calculating the eigenvalues of a Hamiltonian. While the more commonly used ph-AFQMC variant employs the phaseless constraint to control the sign problem, fp-AFQMC attempts to sample the ground state energy brute-force with exponential-scaling sample complexity. Despite this scaling, it is possible to perform relatively large active space calculations in practice using accurate trial states.

In fp-AFQMC, the ground-state energy is estimated by

$$E(\tau) = \frac{\langle \psi_l | \hat{H} e^{-\tau \hat{H}} | \psi_r \rangle}{\langle \psi_l | e^{-\tau \hat{H}} | \psi_r \rangle} = \frac{\int d\mathbf{x} p(\mathbf{x}) \langle \psi_l | \hat{H} \hat{B}(\mathbf{x}, \Delta\tau) | \psi_r \rangle}{\int d\mathbf{x} p(\mathbf{x}) \langle \psi_l | \hat{B}(\mathbf{x}, \Delta\tau) | \psi_r \rangle}, \quad (19)$$

where $|\psi_l\rangle$ and $|\psi_r\rangle$ are left and right trial states, respectively, $\hat{B}(\mathbf{x}, \Delta\tau)$ is defined in Eq. (9), and τ is the imaginary time. Analogously to the ph-AFQMC propagator sampling, the auxiliary fields \mathbf{x} are sampled from the Gaussian distribution $p(\mathbf{x})$. Unlike ph-AFQMC, we do not use the force bias to perform importance sampling. Instead, only mean-field subtraction is used. We also do not employ population control.

A judicious choice of $|\psi_l\rangle$ and $|\psi_r\rangle$ offers two advantages. First, more accurate trial states (e.g., selected CI) reduce the imaginary time required to project the ground state energy to a given accuracy, thereby reducing the noise in the energy estimate.^{27,41} Integrating the GPU-accelerated MSD code⁵¹ will also significantly accelerate the fp-AFQMC calculations with MSD trials. Second, since energies are measured using the state $|\psi_l\rangle$, the closer this state is to the ground state, the smaller the variance in the energy estimate due to the zero variance principle.⁵² Furthermore, one can use a CCSD wavefunction as the initial state $|\psi_r\rangle$, which reduces the projection time. The CCSD wavefunction is employed by performing a Hubbard–Stratonovich transformation on the exponential of the cluster operator.²⁷ Using trial states belonging to specific symmetry sectors also allows one to target the lowest energy states in the corresponding sectors.

As an illustrative example, we consider the D_{4h} symmetric transition state of cyclobutadiene. This state has a biradical character, which makes it a challenging problem for single reference electronic structure methods, including spin-restricted and spin-unrestricted CCSD(T) [RCCSD(T) and UCCSD(T)]. We performed fp-AFQMC calculations on this system using an MSD trial, obtained via heat-bath configuration interaction (HCl), as $|\psi_l\rangle$ and the spin-restricted CCSD (RCCSD) state as $|\psi_r\rangle$. The HCl state was obtained from an HCl calculation with a crude $\epsilon_1 = 10^{-4}$ in the full space except for four frozen HF orbitals. These orbitals were kept frozen in all correlated calculations. The geometry was taken from Ref. 27, which also reported fp-AFQMC energies for this system. As additional validation of our results, we converged the ph-AFQMC energy with respect to the number of determinants in the trial HCl state; we expect this energy to be nearly exact. Results are shown in Fig. 10.

The fp-AFQMC energy converges to the converged ph-AFQMC energy within statistical error bars. Our results suggest that hybrid (H)-AFQMC energies reported in Ref. 53 are likely biased as they are too much lower than our converged ph-AFQMC and fp-AFQMC energies. RCCSD(T) is about 13.5 mE_h higher than ph-AFQMC, while UCCSD(T) is closer but still about 4.5 mE_h higher. The differences in the barrier height largely come from transition state energies because all these methods work well for the equilibrium D_{2h} geometry.

E. AFQMC beyond the ground-state electronic structure energy

The preceding discussions concentrate on the ground-state *ab initio* electronic structure energy calculations. Apart from this, *ipie* now also accommodates other types of calculations within the framework of AFQMC, such as property calculations,³⁶ finite temperature calculations,^{14,54} and ground-state calculations of electron–phonon coupled model systems.¹⁵ The code structures for these different features mirror those of the zero temperature *ab initio*

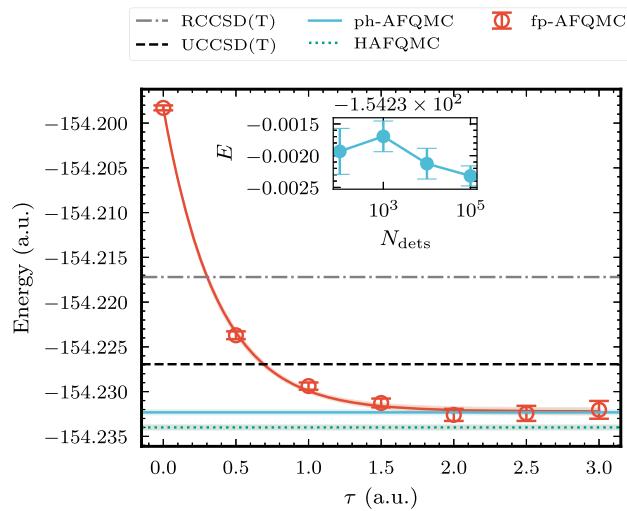


FIG. 10. Convergence of the fp-AFQMC energy for the cyclobutadiene transition state in the cc-pVQZ basis set with projection time. The inset shows the convergence of ph-AFQMC energy with respect to the number of determinants in the trial.

electronic structure implementation. We also note that *ipie*'s legacy code also supports the ground state and finite-temperature calculations of other model systems such as the Hubbard model, which we do not discuss in this paper.

1. Electronic structure at finite temperatures

An extension of the phaseless AFQMC method at zero temperature, finite-temperature AFQMC (FT-AFQMC) was developed to study systems at finite temperatures.¹⁴ It is customary (though not necessary⁵⁵) to work in the grand canonical ensemble described by the temperature T , volume V , and chemical potential μ . The central quantities of interest are thermal expectation values computed from the partition function

$$\Xi = \text{Tr} \left[e^{-\beta(\hat{H} - \mu \hat{N})} \right], \quad (20)$$

where \hat{N} is the total number operator and μ is the chemical potential. The Boltzmann factor in Eq. (20) can be interpreted as a propagator in imaginary time $\tau = \beta$.

Analogously to the zero temperature case, τ is first discretized into l intervals of length $\Delta\tau = \tau/l$,

$$\Xi = \text{Tr} \left[e^{-\beta(\hat{H} - \mu \hat{N})} \right] = \text{Tr} \left[\lim_{l \rightarrow \infty} \prod_{k=1}^{k=l} e^{-\Delta\tau(\hat{H} - \mu \hat{N})} \right], \quad (21)$$

the short-time propagator is then Trotter decomposed as

$$e^{-\Delta\tau(\hat{H} - \mu \hat{N})} \simeq e^{-\frac{\Delta\tau}{2}(\hat{v}_0 - \mu \hat{N})} e^{\frac{\Delta\tau}{2} \sum_y \hat{v}_y^2} e^{-\frac{\Delta\tau}{2}(\hat{v}_0 - \mu \hat{N})}, \quad (22)$$

and the application of the Hubbard–Stratonovich transformation gives

$$e^{-\Delta\tau(\hat{H} - \mu \hat{N})} \simeq \int d\mathbf{x} p(\mathbf{x}) \hat{B}(\mathbf{x}, \Delta\tau, \mu), \quad (23)$$

where the one-body propagator \hat{B} is now also function of μ ,

$$\hat{B}(\mathbf{x}, \Delta\tau, \mu) = e^{-\frac{\Delta\tau}{2}(\hat{v}_0 - \mu\hat{N})} e^{-\sqrt{\Delta\tau}\mathbf{x}\cdot\hat{\mathbf{v}}} e^{-\frac{\Delta\tau}{2}(\hat{v}_0 - \mu\hat{N})}. \quad (24)$$

The grand canonical partition function is thus evaluated as

$$\Xi = \text{Tr} \left[e^{-\beta(\hat{H} - \mu\hat{N})} \right] = \int d\mathbf{x}_1 \cdots d\mathbf{x}_l \underbrace{p(\mathbf{x}_1) \cdots p(\mathbf{x}_l)}_{p(\mathbf{x}_1, \dots, \mathbf{x}_l)} \times \text{Tr} \left[\prod_{k=1}^l \hat{B}(\mathbf{x}_k, \Delta\tau, \mu) \right], \quad (25)$$

with $p(\mathbf{x}_1, \dots, \mathbf{x}_l)$ being the probability of sampling a specific path designated by auxiliary fields $\mathbf{x}_1, \dots, \mathbf{x}_l$. Furthermore, the trace in Eq. (25) can be written in terms of a determinant,^{56–58} which finally yields

$$\Xi = \int d\mathbf{x}_1 \cdots d\mathbf{x}_l p(\mathbf{x}_1, \dots, \mathbf{x}_l) \det \left[\mathbf{I} + \prod_{k=1}^l \mathbf{B}(\mathbf{x}_k, \Delta\tau, \mu) \right], \quad (26)$$

where \mathbf{I} is the identity matrix and \mathbf{B} is a matrix representation of \hat{B} in a single-particle basis. It should be noted that the partition function itself is not explicitly calculated—expectation values derived from it are. For some generic observable \hat{O} , its expectation value is

$$\langle \hat{O} \rangle = \frac{1}{\Xi} \text{Tr} \left[e^{-\beta(\hat{H} - \mu\hat{N})} \hat{O} \right], \quad (27)$$

$$= \int d\mathbf{X} \frac{p(\mathbf{X}) \text{Tr}[A(\mathbf{X}, \Delta\tau, \mu)]}{\int d\mathbf{Y} p(\mathbf{Y}) \text{Tr}[A(\mathbf{Y}, \Delta\tau, \mu)]}, \quad (28)$$

$$\times \frac{\text{Tr}[A(\mathbf{X}, \Delta\tau, \mu)\hat{O}]}{\text{Tr}[A(\mathbf{X}, \Delta\tau, \mu)]}, \quad (29)$$

where we introduced the shorthand \mathbf{X} for the set of auxiliary fields along an imaginary path $\{\mathbf{x}_1, \dots, \mathbf{x}_l\}$ and,

$$A(\mathbf{X}, \Delta\tau, \mu) = \prod_{k=1}^l \hat{B}(\mathbf{x}_k, \Delta\tau, \mu). \quad (30)$$

Rewriting Eq. (27) in the form Eq. (29) allows $\langle \hat{O} \rangle$ to be estimated through an importance sampling procedure. The field configurations \mathbf{X} are obtained via Monte Carlo sampling from the modified probability distribution $\tilde{p}(\mathbf{X}) = p(\mathbf{X}) \text{Tr}[A(\mathbf{X}, \Delta\tau, \mu)]$, while the computed random variables are the *local* expectation values

$$O_L(\mathbf{X}, \Delta\tau, \mu) = \frac{\text{Tr}[A(\mathbf{X}, \Delta\tau, \mu)\hat{O}]}{\text{Tr}[A(\mathbf{X}, \Delta\tau, \mu)]}. \quad (31)$$

Similar to the zero temperature case, the importance sampling is implemented by initializing a set of walkers in the space of auxiliary fields with weights w_i and propagating them in imaginary time. We, therefore, evaluate Eq. (29) in practice as

$$\langle \hat{O} \rangle = \frac{\sum_i w_i O_L(\mathbf{X}_i)}{\sum_i w_i}. \quad (32)$$

An example of the imaginary time trace for observables is provided in Fig. 11, which reproduces Fig. 3 in Ref. 14.

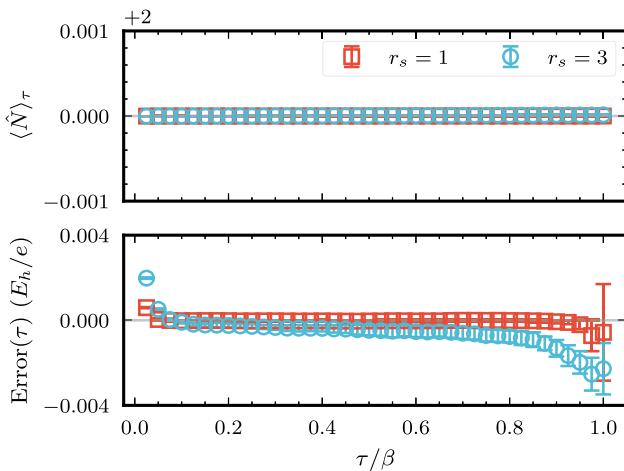


FIG. 11. Imaginary time trace from FT-AFQMC of the uniform electron gas model with 2 electrons. The top panel depicts the average electron number at imaginary time τ , $\langle \hat{N} \rangle_\tau$, while the bottom panel shows the error in the total energy compared to FCI, i.e., $E_{\text{FT-AFQMC}}(\tau) - E_{\text{FCI}}$. These results reproduce Fig. 3 in Ref. 14.

2. Electrons coupled to phonons

Utilizing the tools provided in `ipie`, one may write a new projector Monte Carlo method to calculate the ground state of electrons coupled to phonons (i.e., lattice vibration),¹⁵

$$\hat{H} = \underbrace{\sum_{pq} h_{pq} a_p^\dagger a_q}_{\hat{H}_{\text{el}}} + \underbrace{\sum_v \omega_v b_v^\dagger b_v}_{\hat{H}_{\text{ph}}} + \underbrace{\sum_{pqv} g_{pqv} a_p^\dagger a_q (b_v + b_v^\dagger)}_{\hat{H}_{\text{el-ph}}}, \quad (33)$$

where $b^{(\dagger)}$ represents the bosonic annihilation (and creation) operator, the first term represents the electronic band term, the second term represents the phonon band term, and the third term represents the coupling between electrons and phonons.

The corresponding ground-state projector is obtained from trotterizing the imaginary time propagator corresponding to Eq. (33),

$$e^{-\Delta\tau\hat{H}} = e^{-\frac{\Delta\tau}{2}\hat{H}_{\text{el}}} e^{-\frac{\Delta\tau}{2}\hat{H}_{\text{ph}}} e^{-\Delta\tau\hat{H}_{\text{el-ph}}} e^{-\frac{\Delta\tau}{2}\hat{H}_{\text{ph}}} e^{-\frac{\Delta\tau}{2}\hat{H}_{\text{el}}} + \mathcal{O}(\Delta\tau^3). \quad (34)$$

In our method, we choose walkers of the form $|\psi_w\rangle \otimes |\mathbf{X}_w\rangle \equiv |\psi_w(\tau), \mathbf{X}_w(\tau)\rangle$, with $|\psi_w\rangle$ being a single determinant and $|\mathbf{X}_w\rangle$ being the coordinates for phonon displacements. Using the Monte Carlo sampling, we can work in the position space of phonons without invoking any boson number truncation. This strategy differs from other standard Monte Carlo approaches in this area.^{59,60}

With importance sampling via a trial wavefunction, $|\Psi_T\rangle$, our global wavefunction is a weighted linear combination of walker vibronic wavefunctions,

$$|\Psi(\tau)\rangle = \sum_{w=1}^{N_{\text{walkers}}} w_w(\tau) \frac{|\psi_w(\tau), \mathbf{X}_w(\tau)\rangle}{\langle \Psi_T | \psi_w(\tau), \mathbf{X}_w(\tau) \rangle}. \quad (35)$$

The propagation of the bosonic degrees of freedom under \hat{H}_{ph} is performed via a diffusion Monte Carlo algorithm.⁶¹ With the walker

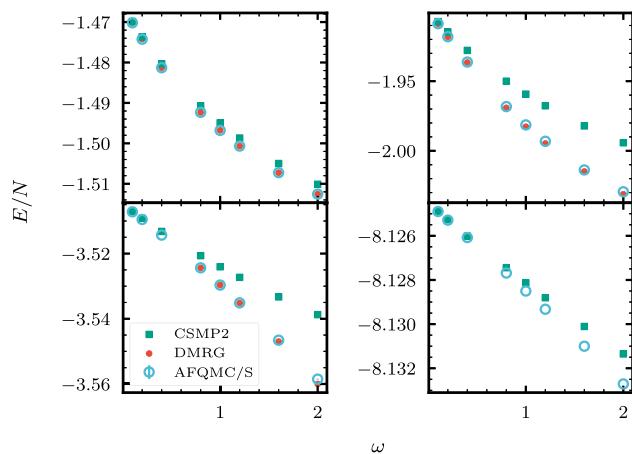


FIG. 12. Reproduction of Fig. 3 in Ref. 15. Results are shown for a 20-site Holstein model at half-filling. We used a coherent state trial to obtain the AFQMC/S results. CSMP2 data points correspond to perturbation theory-based computations based on molecular orbitals obtained from SCF calculations for a Holstein Fock operator.¹⁵ Panels (a)–(d) show scans along phonon frequencies ω for λ equals (a) 0.1, (b) 0.3, (c) 0.8, and (d) 2.

representation, the propagation with \hat{H}_{el} and $\hat{H}_{\text{el-ph}}$ is straightforward by exploiting the Thouless theorem. More details can be found in Ref. 15. As an example calculation, we picked the one-dimensional Holstein model under a periodic boundary condition,⁶² which involves $h_{pq} = -t(\delta_{p+1,q} + \delta_{p,q+1})$, $\omega_v = \omega$, and $g_{pqr} = g\delta_{pq}\delta_{pv}$. For 20-site at half-filling, we considered four different unitless electron-phonon coupling strengths $\lambda = \frac{g^2}{2t\omega}$, which are displayed in Fig. 12. It is evident from Fig. 12 that the coherent state trial is not an optimal choice for the intermediate coupling regime, where $2t\lambda < \omega$. By employing more sophisticated trial wavefunctions, we can obtain more reliable results also for the intermediate coupling regime.¹⁵ The computational bottleneck of our QMC algorithm is the evolution under the el-ph projection operator, generally scaling as $\mathcal{O}(N^3)$.

3. Automatic differentiable AFQMC

The computation of observables that do not commute with \hat{H} poses additional challenges in projector Monte Carlo, such as AFQMC. A recently proposed approach by Mahajan *et al.*³⁶ aims to compute the response estimator [Eq. (36)] which, though still not

exact due to the discontinuity in the distribution, is accurate enough to give reliable results,

$$\langle \hat{O} \rangle_{\text{response}} = \left. \frac{dE_{\text{AFQMC}}(\lambda)}{d\lambda} \right|_{\lambda=0}. \quad (36)$$

Here, λ is a parameter for the perturbed Hamiltonian $\hat{H}(\lambda) = \hat{H} + \lambda\hat{O}$, and AFQMC energy $E_{\text{AFQMC}}(\lambda)$ is given by the Monte Carlo estimator

$$E_{\text{AFQMC}}(\lambda) = \frac{\sum_i w_i(\lambda) \frac{\langle \Psi_T(\lambda) | \hat{H}(\lambda) | \psi_i(\lambda) \rangle}{\langle \Psi_T(\lambda) | \psi_i(\lambda) \rangle}}{\sum_i w_i(\lambda)}, \quad (37)$$

where $w_i(\lambda)$ is the weight for the i th walker, and $|\psi_i(\lambda)\rangle$ is the i th walker state at coupling strength λ .

We implemented this scheme for computing observables using the Automatic Differentiation (AD) functionality of PyTorch⁶³ as an add-on within `ipie`. It is noteworthy that storing the computation graph for an entire AFQMC run requires substantial memory. In practice, we use the concept of AD blocks.³⁶ We only track the computation graph within an AD block; there is no connection between different AD blocks. Using this approach, we manage memory costs by adjusting block size. We also use the gradient checkpointing technique⁶⁴ to reduce the memory cost further.

For relatively large systems, the AFQMC calculation is parallelized using MPI to distribute walkers over MPI tasks. The differentiation of the AFQMC global energy estimator is not embarrassingly parallel since walkers will mix between different MPI ranks via population control. The AD implementation in `ipie` does not support communication between MPI tasks. Therefore, we perform “local” population control within an MPI rank. Because the local population control does not mix the walkers between different ranks, it is valid to regard those estimates as independent samples. We thus perform block analysis on those samples to obtain the final result. Effectively, this amounts to running very low walker population simulations (50 walkers per task here), which is only practically possible for small system sizes where population control is not a concern. For intermediate system sizes, one could imagine using large memory nodes and OpenMP threading or using single GPUs with a sufficient amount of memory.

Here, we present the results of AD-AFQMC calculations on various molecular systems. The molecular integrals are obtained by PySCF,³¹ and the modified Cholesky decomposition is performed with `ipie`, with a threshold of 10^{-5} . We used a time step of 0.01 a.u., and periodic reorthogonalization of walkers is performed

TABLE I. Comparison of AD-AFQMC dipole moment (in a.u.) of various molecules at equilibrium geometry with the implementation in Ref. 36 and other quantum chemistry methods. The data using RCCSD and RCCSD(T) are also extracted from Ref. 36.

Molecule	This work	AD-AFQMC in Ref. 36	RCCSD	RCCSD(T)	Experiment
H ₂ O	0.723(2)	0.720(2)	0.7335	0.7247	0.730 ⁶⁶
NH ₃	0.592(2)	0.592(2)	0.6015	0.5938	0.581(1) ⁶⁷
CO	0.022(4)	0.019(4)	0.0199	0.0429	0.048(1) ⁶⁸
HCl	0.428(1)	0.429(1)	0.4318	0.4273	0.430 ⁶⁹
HBr	0.332(2)	0.329(2)	0.3289	0.3245	0.325 ⁶⁹

every five steps for all calculations. We used the restricted Hartree Fock (RHF) trial for all systems. For accurate statistical analysis, we perform block analysis for ≥ 200 gradient samples for each calculation. All AFQMC calculations are performed using the frozen-core approximation.

We benchmarked our implementation on various small molecules using the aug-cc-pVTZ⁶⁵ basis set and compared the results to Ref. 36 in Table I. All dipole moments align strongly with the AD-AFQMC results reported in Ref. 36. Furthermore, except for CO, AD-AFQMC matches experimental values better than RCCSD and exhibits accuracy comparable to RCCSD(T). This supports the widely held view that AFQMC's accuracy falls between CCSD and CCSD(T).

F. Enhanced integration testing and no-MPI mode

As ipie expands its functionalities, robust testing workflows become crucial. ipie supports a new integration testing framework that enhances the package's robustness and adaptability and significantly contributes to its reliability and ease of use for the end-users. The GitHub continuous integration (CI) workflow automatically tests new pull requests, ensuring that every change to the codebase does not break existing functionalities. The workflow encompasses a comprehensive test suite and executes various linting and code formatting checks before running serial and parallel unit tests and integration tests.

Recognizing the diverse computational environments in which ipie might be deployed, we introduced the no-MPI mode. This mode is designed for situations where MPI is unavailable or its use is not desired, offering greater flexibility for users and developers. The CI workflow includes a job that tests ipie's functionality without the MPI dependency. This mode is particularly beneficial for users who wish to perform quantum Monte Carlo simulations on personal computing setups or in environments where setting up MPI is challenging.

V. INTERFACES TO EXTERNAL PACKAGES

A. Dice interface and SHCI-AFQMC

ipie includes utilities for converting the output from Dice,²¹ a package that employs semistochastic heat bath configuration interaction (SHCI)^{21,32} as the complete active space (CAS) solver,

Code Snippet 11 Convert Dice output to the MSD trial in ipie.

```
python -u /path-to-ipie/tools/extract_dice.py --dice-wfn
      /path-to-Dice-output/dets.bin --sort --verbose
```

which produces the wfn.h5 file containing the coefficients and the indices of occupied orthogonal orbitals for spin-up and spin-down sectors that follows the block-formatted ($\alpha\alpha\cdots\beta\beta\cdots$) orbital convention in ipie. The SHCI calculation generates a good MSD trial for challenging systems.^{17,41} However, it is not a black-box approach and often requires careful handling, such as selecting CAS and using natural orbitals, as discussed in Sec. IV C.

These detailed discussions are beyond the scope of this article, and we briefly mention the procedures in the semi-black box example provided within ipie.¹⁷

1. Initiate a preliminary rough SHCI calculation in an extensive active space for the system under study.
2. Derive the SHCI one-electron reduced density matrix (1-RDM), extracting the resultant natural orbital occupation number (NOON) and natural orbitals.
3. Define an active space criterion based on a predetermined NOON threshold.
4. Adjust the orbitals through rotation, aligning them with the unitary transformation specified by the natural orbitals.
5. Execute a refined SHCI self-consistent field calculation within the new active space determined in the previous step.
6. Use this MSD trial in AFQMC.

This strategy ensures the trial wavefunction encapsulates static correlations within the active space via the MSD trial. At the same time, AFQMC incorporates the residual dynamic correlations. This procedure is folded into the factory utility method ipie.utils.from_dice.build_driver_from_shciscf.

B. TREXIO support and CIPSI-AFQMC

The TREXIO library and file format have been developed to offer a robust and efficient solution for storing and exchanging wavefunction parameters and matrix elements.³³ This library supports bindings in several programming languages, including Python, and can be conveniently installed via the pip package manager.

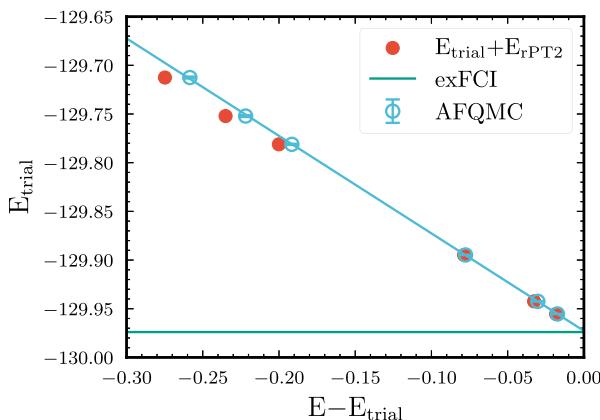
The compatibility of ipie with the TREXIO format facilitates its integration with various software packages. Specifically, it allows ipie to utilize trial wavefunctions produced by Quantum Package,⁷⁰ along with the associated one-electron integrals and Cholesky-decomposed electron repulsion integrals. This interface allowed us to check if AFQMC could complement configuration interaction using perturbative selection done iteratively (CIPSI) calculations to improve full configuration interaction (FCI) energy estimates of large systems.

Typically, to estimate the FCI energy from a CIPSI calculation, one extrapolates to zero the variational energy, E_{trial} , as a function of the renormalized second-order perturbative correction, E_{rPT2} .⁷⁰ However, the AFQMC energy, E_{AFQMC} , is anticipated to provide a closer approximation to the FCI energy than the sum of E_{trial} and E_{rPT2} . This expectation is particularly relevant for systems with large E_{rPT2} corrections.

We conducted AFQMC calculations on the nitroxyl and benzene molecules using CIPSI trial wavefunctions of increasing sizes. The results of these calculations are detailed in Table II. Supporting our expectation, Fig. 13 demonstrates that the AFQMC corrections align the data points along a straight line,⁷¹ validating the hypothesis that AFQMC energies are more reliable for the extrapolation toward the FCI value, especially in cases with large rPT2 corrections. Employing calculations with comparatively small wavefunctions for benzene, as shown in Table II, a three-point linear extrapolation based on the rPT2 correction yields a correlation energy of -858.6 mEh. In contrast, extrapolation using AFQMC energies results in a correlation energy of -862.2 mEh. This latter value is

TABLE II. Renormalized PT2 correction E_{rPT2} , variational energy E_{var} , and AFQMC energy E_{AFQMC} for the nitroxyl and benzene molecules as functions of the number of determinants in the variational space.

System, basis	N_{det}	E_{trial}	$E_{\text{trial}} + E_{\text{rPT2}}$	E_{AFQMC}
HNO, 6-31G	1	-129.712 554	-129.987 361	-129.971(1)
	2	-129.752 113	-129.987 158	-129.974(1)
	41	-129.781 149	-129.981 219	-129.9727(6)
	748	-129.894 775	-129.973 405	-129.9724(2)
	2838	-129.942 391	-129.975 057	-129.9726(1)
	14 201	-129.955 368	-129.973 620	-129.9727(1)
C_6H_6 , cc-pVDZ	1	-230.720 490 4	-231.393 30	-231.5887(7)
	23	-230.761 581 6	-231.421 67	-231.5866(7)
	828	-230.868 707 2	-231.446 02	-231.5864(6)
	15 690	-231.130 008 7	-231.493 30	-231.5848(4)
	109 869	-231.315 239 9	-231.530 22	-231.5842(4)

**FIG. 13.** Energy of the trial wavefunction (E_{trial}) as a function of ΔE , where $\Delta E = E_{\text{rPT2}}$ for CIPSI calculations and $\Delta E = E_{\text{AFQMC}} - E_{\text{trial}}$ for AFQMC. exFCI is the extrapolated FCI energy obtained from CIPSI calculations.

significantly closer to the correlation energy of -863.4 mEh achieved through CIPSI with 167×10^6 determinants.⁷²

These preliminary calculations enabled by the TREXIO interface illustrate that the integration of AFQMC with CIPSI emerges as a promising methodology for estimating the FCI energy of systems larger than those currently feasible.

C. FQE interfaces

The Fermionic quantum emulator (FQE)³⁴ is a lightweight fermionic circuit simulator, which is particularly useful in quantum computing where it aids in the development and testing of quantum algorithms tailored for fermionic systems. ipie provides the conversion between ipie's MSD wavefunction and the FQE wavefunction.

VI. CONCLUSIONS AND OUTLOOKS

This paper summarized the improvements and new features added in ipie since its original release.¹⁷ These improvements

enhance modularity and computational efficiency and offer intuitive user-end APIs. New features and interfaces aim to expand a broader spectrum of AFQMC calculations in quantum chemistry.

We summarize the following key features we highlighted in this paper:

1. *Distributed Hamiltonians to remove the memory bottleneck.* We demonstrated ipie's capacity for studying large systems deploying GPUs with significantly higher efficiency than CPU-based implementations, exemplified in our case study assessing the interaction energies in a benzene dimer.
2. *GPU support for MSD trial wavefunctions.* With customized CUDA kernels, we enabled an efficient realization of Wick's theorem. Timing benchmarks for $[\text{Cu}_2\text{O}_2]^{2+}$ and $[\text{Fe}_2\text{S}_2(\text{SCH}_3)_4]^{2-}$ were shown to achieve more than an order of magnitude speedup compared to our CPU implementation for a large MSD trial.⁵¹
3. *Support for complex-valued Cholesky vectors.* ipie can handle complex-valued Cholesky vectors that may arise when the underlying basis functions are complex-valued.
4. *Free-projection AFQMC.* A numerically exact AFQMC approach can be used to study small strongly correlated systems.
5. *Finite-temperature AFQMC.* A finite-temperature AFQMC algorithm based on the grand canonical ensemble was added.
6. *Electron–phonon QMC.* A QMC algorithm that computes the ground state of electron–phonon problems was added.
7. *Automatic differentiable AFQMC.* We offer AFQMC property calculations via automatic differentiation.
8. *External package interfaces.* ipie is now interfaced with PySCF, Dice, TREXIO, and FQE.

ipie has been mainly designed for *ab initio* calculations, but more supports for model Hamiltonians have been added and are under active development, including the Hubbard–Holstein,¹⁵ Peierls, uniform electron gas models,¹² etc. The embedding method is also an interesting feature and currently not supported in ipie, but it has been studied in the literature,^{73,74} where AFQMC was employed as a solver for a model Hamiltonian in the embedding method. With the improved modularity and usability, we believe it would be

relatively straightforward to implement. We also note that `ipie` currently uses double precision arithmetic for all calculations; although it also supports mixed precision, where single precision is used for propagations and double precision is used for local energy estimation, the stability and efficiency need to be further investigated. In addition, there is indeed potential for further optimization in the CPU implementation, particularly through enhancements with a hybrid MPI/OpenMP programming strategy, and we will continue to explore avenues along those lines.

We hope that `ipie` will serve as a community code base for developing *ab initio* AFQMC methods and their applications. Furthermore, as `ipie` is written mainly in Python, we anticipate its use in machine learning and quantum computing¹¹ communities will also grow.

ACKNOWLEDGMENTS

The work by T.J. and J.L. was supported by the Department of Energy (DOE) Office of Fusion Energy Sciences “Foundations for quantum simulation of warm dense matter” project and by Harvard University’s startup funds. Computations were carried out partly on the FASRC cluster supported by the FAS Division of Science Research Computing Group at Harvard University. This work also used the Delta system at the National Center for Supercomputing Applications through allocation Grant Nos. CHE230032, CHE230088, and PHY230192 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by National Science Foundation Grant Nos. 2138259, 2138286, 2138307, 2137603, and 2138296. P.F.L. and A.S. have received financial support from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation program (Grant Agreement No. 863481). A.S. was also supported by the European Center of Excellence in Exascale Computing (TREX), which has received funding from the European Union’s Horizon 2020—Research and Innovation program—under Grant Agreement No. 952165. S.F.U. acknowledges David Reichman for his support. We acknowledge Yifei Huang and Dingshun Lv for their GPU-MSD code contributions.⁵¹

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Tong Jiang: Data curation (lead); Formal analysis (equal); Investigation (equal); Software (equal); Visualization (lead); Writing – original draft (lead); Writing – review & editing (lead). **Moritz K. A. Baumgarten:** Data curation (equal); Formal analysis (equal); Investigation (equal); Software (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Pierre-François Loos:** Data curation (equal); Formal analysis (equal); Investigation (equal); Software (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Ankit Mahajan:** Data curation (equal); Formal analysis (equal);

Investigation (equal); Software (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal).

Anthony Scemama: Data curation (equal); Formal analysis (equal); Investigation (equal); Software (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal).

Shu Fay Ung: Data curation (equal); Formal analysis (equal); Investigation (equal); Software (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Jinghong Zhang:** Data curation (equal); Formal analysis (equal); Investigation (equal); Software (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal).

Fionn D. Malone: Conceptualization (equal); Software (lead); Writing – review & editing (equal). **Joonho Lee:** Conceptualization (lead); Data curation (equal); Formal analysis (lead); Funding acquisition (lead); Investigation (lead); Software (lead); Writing – original draft (lead); Writing – review & editing (equal).

DATA AVAILABILITY

The data that support the findings of this study are openly available in our Zenodo repository at <https://doi.org/10.5281/zenodo.12522916>.

REFERENCES

1. S. Zhang, J. Carlson, and J. E. Gubernatis, “Constrained path quantum Monte Carlo method for fermion ground states,” *Phys. Rev. Lett.* **74**, 3652–3655 (1995).
2. S. Zhang and H. Krakauer, “Quantum Monte Carlo method using phase-free random walks with slater determinants,” *Phys. Rev. Lett.* **90**, 136401 (2003).
3. J. Lee, H. Q. Pham, and D. R. Reichman, “Twenty years of auxiliary-field quantum Monte Carlo in quantum chemistry: An overview and assessment on main group chemistry and bond-breaking,” *J. Chem. Theory Comput.* **18**, 7024–7042 (2022).
4. M. Motta and S. Zhang, “Ab initio computations of molecular systems by the auxiliary-field quantum Monte Carlo method,” *WIREs Comput. Mol. Sci.* **8**, e1364 (2018).
5. W. J. Huggins, B. A. O’Gorman, N. C. Rubin, D. R. Reichman, R. Babbush, and J. Lee, “Unbiasing fermionic quantum Monte Carlo with a quantum computer,” *Nature* **603**, 416–420 (2022).
6. K. Wan, W. J. Huggins, J. Lee, and R. Babbush, “Matchgate shadows for fermionic quantum simulation,” *Commun. Math. Phys.* **404**, 629–700 (2023).
7. M. Amsler, P. Deglmann, M. Degroote, M. P. Kaicher, M. Kiser, M. Kühn, C. Kumar, A. Maier, G. Samsonidze, A. Schroeder *et al.*, “Classical and quantum trial wave functions in auxiliary-field quantum Monte Carlo applied to oxygen allotropes and a CuBr₂ model system,” *J. Chem. Phys.* **159**, 044119 (2023).
8. M. Kiser, A. Schroeder, G.-L. R. Anselmetti, C. Kumar, N. Moll, M. Streif, and D. Vodola, “Classical and quantum cost of measurement strategies for quantum-enhanced auxiliary field quantum Monte Carlo,” *New J. Phys.* **26**, 033022 (2024).
9. B. Huang, Y.-T. Chen, B. Gupt, M. Suchara, A. Tran, S. McArdle, and G. Galli, “Evaluating a quantum-classical quantum Monte Carlo algorithm with Matchgate shadows,” *Phys. Rev. Res.* (to be published) (2024).
10. M. Kiser, M. Beuerle, and F. Simkovic IV, “Contextual subspace auxiliary-field quantum Monte Carlo: Improved bias with reduced quantum resources,” *arXiv:2408.06160v2* (2024).
11. T. Jiang, J. Zhang, M. K. A. Baumgarten, M.-F. Chen, H. Q. Dinh, A. Ganeshram, N. Maskara, A. Ni, and J. Lee, “Walking through Hilbert space with quantum computers,” *arXiv:2407.11672v1* (2024).
12. J. Lee, F. D. Malone, and M. A. Morales, “An auxiliary-field quantum Monte Carlo perspective on the ground state of the dense uniform electron gas: An investigation with Hartree-Fock trial wavefunctions,” *J. Chem. Phys.* **151**, 064122 (2019).

- ¹³J. Lee and D. R. Reichman, "Stochastic resolution-of-the-identity auxiliary-field quantum Monte Carlo: Scaling reduction without overhead," *J. Chem. Phys.* **153**, 044131 (2020).
- ¹⁴J. Lee, M. A. Morales, and F. D. Malone, "A phaseless auxiliary-field quantum Monte Carlo perspective on the uniform electron gas at finite temperatures: Issues, observations, and benchmark study," *J. Chem. Phys.* **154**, 064109 (2021).
- ¹⁵J. Lee, S. Zhang, and D. R. Reichman, "Constrained-path auxiliary-field quantum Monte Carlo for coupled electrons and phonons," *Phys. Rev. B* **103**, 115123 (2021).
- ¹⁶J. Lee, F. D. Malone, M. A. Morales, and D. R. Reichman, "Spectral functions from auxiliary-field quantum Monte Carlo without analytic continuation: The extended Koopmans' theorem approach," *J. Chem. Theory Comput.* **17**, 3372–3387 (2021).
- ¹⁷F. D. Malone, A. Mahajan, J. S. Spencer, and J. Lee, "Ipie: A python-based auxiliary-field quantum Monte Carlo program with flexibility and efficiency on CPUs and GPUs," *J. Chem. Theory Comput.* **19**, 109–121 (2023).
- ¹⁸S. K. Lam, A. Pitrou, and S. Seibert, in Proceedings of the 2nd Workshop on the LLVM Compiler Infrastructure in HPC. LLVM'15, 2015.
- ¹⁹C. J. Cramer, M. Włoch, P. Piecuch, C. Puzzarini, and L. Gagliardi, "Theoretical models on the Cu₂O₂ torture track: Mechanistic implications for oxytyrosinase and small-molecule analogues," *J. Phys. Chem. A* **110**, 1991–2004 (2006).
- ²⁰P. R. Kent, A. Annaberdiyev, A. Benali, M. C. Bennett, E. J. Landinez Borda, P. Doak, H. Hao, K. D. Jordan, J. T. Krogel, I. Kylinpää *et al.*, "QMCPACK: Advances in the development, efficiency, and application of auxiliary field and real-space variational and diffusion quantum Monte Carlo," *J. Chem. Phys.* **152**, 174105 (2020).
- ²¹S. Sharma, A. A. Holmes, G. Jeannairet, A. Alavi, and C. J. Umrigar, "Semistochastic heat-bath configuration interaction method: Selected configuration interaction with semistochastic perturbation theory," *J. Chem. Theory Comput.* **13**, 1595–1604 (2017).
- ²²L. Hahn, P. Deglmann, and M. Kühn, "Chelate complexes of 3d transition metal ions—A challenge for electronic-structure methods?," *J. Chem. Theory Comput.* **20**, 4545 (2024).
- ²³V. P. Vysotskiy, C. Filippi, and U. Ryde, "Scalar relativistic all-electron and pseudopotential *ab initio* study of a minimal nitrogenase [Fe(SH)₄H]⁻ model employing coupled-cluster and auxiliary-field quantum Monte Carlo many-body methods," *J. Phys. Chem. A* **128**, 1358 (2024).
- ²⁴M. S. Chen, J. Lee, H.-Z. Ye, T. C. Berkelbach, D. R. Reichman, and T. E. Markland, "Data-efficient machine learning potentials from transfer learning of periodic correlated electronic structure methods: Liquid water at AFQMC, CCSD, and CCSD(T) accuracy," *J. Chem. Theory Comput.* **19**, 4510–4519 (2023).
- ²⁵T. Jiang, B. O'Gorman, A. Mahajan, and J. Lee, "Unbiasing fermionic auxiliary-field quantum Monte Carlo with matrix product state trial wavefunctions," *arXiv:2405.05440v1* (2024).
- ²⁶The ipie package (v0.7.1), <https://github.com/JoonhoLee-Group/ipie> (2024).
- ²⁷A. Mahajan and S. Sharma, "Taming the sign problem in auxiliary-field quantum Monte Carlo using accurate wave functions," *J. Chem. Theory Comput.* **17**, 4786–4798 (2021).
- ²⁸E. Vitali, P. Rosenberg, and S. Zhang, "Calculating ground-state properties of correlated fermionic systems with BCS trial wave functions in Slater determinant path-integral approaches," *Phys. Rev. A* **100**, 023621 (2019).
- ²⁹C.-C. Chang, B. M. Rubenstein, and M. A. Morales, "Auxiliary-field-based trial wave functions in quantum Monte Carlo calculations," *Phys. Rev. B* **94**, 235144 (2016).
- ³⁰"Plum: Multiple dispatch in Python," <https://github.com/beartype/plum>.
- ³¹Q. Sun, T. C. Berkelbach, N. S. Blunt, G. H. Booth, S. Guo, Z. Li, J. Liu, J. D. McClain, E. R. Sayfutyarova, S. Sharma *et al.*, "PySCF: The python-based simulations of chemistry framework," *WIREs Comput. Mol. Sci.* **8**, e1340 (2018).
- ³²A. A. Holmes, N. M. Tubman, and C. Umrigar, "Heat-bath configuration interaction: An efficient selected configuration interaction algorithm inspired by heat-bath sampling," *J. Chem. Theory Comput.* **12**, 3674–3680 (2016).
- ³³E. Posenitskiy, V. G. Chilkuri, A. Ammar, M. Hapka, K. Pernal, R. Shinde, E. J. Landinez Borda, C. Filippi, K. Nakano, O. Kohulák, S. Sorella, P. de Oliveira Castro, W. Jalby, P. L. Ríos, A. Alavi, and A. Scemama, "TREXIO: A file format and library for quantum chemistry," *J. Chem. Phys.* **158**, 174801 (2023).
- ³⁴N. C. Rubin, K. Gunst, A. White, L. Freitag, K. Throssell, G. K.-L. Chan, R. Babbush, and T. Shiozaki, "The fermionic quantum emulator," *Quantum* **5**, 568 (2021).
- ³⁵Z. Li and G. K.-L. Chan, "Spin-projected matrix product states: Versatile tool for strongly correlated systems," *J. Chem. Theory Comput.* **13**, 2681–2695 (2017).
- ³⁶A. Mahajan, J. S. Kurian, J. Lee, D. R. Reichman, and S. Sharma, "Response properties in phaseless auxiliary field quantum Monte Carlo," *J. Chem. Phys.* **159**, 184101 (2023).
- ³⁷J. Hubbard, "Calculation of partition functions," *Phys. Rev. Lett.* **3**, 77 (1959).
- ³⁸R. Stratonovich, "On a method of calculating quantum distribution functions," *Sov. Phys. Dokl.* **2**, 416–419 (1958), <https://zbmath.org/?q=an:0080.19804>.
- ³⁹J. Ren, W. Li, T. Jiang, Y. Wang, and Z. Shuai, "Time-dependent density matrix renormalization group method for quantum dynamics in complex systems," *WIREs Comput. Mol. Sci.* **12**, e1614 (2022).
- ⁴⁰H. Flyvbjerg and H. G. Petersen, "Error estimates on averages of correlated data," *J. Chem. Phys.* **91**, 461–466 (1989).
- ⁴¹A. Mahajan, J. Lee, and S. Sharma, "Selected configuration interaction wave functions in phaseless auxiliary field quantum Monte Carlo," *J. Chem. Phys.* **156**, 174111 (2022).
- ⁴²P. Jurečka, J. Šponer, J. Černý, and P. Hobza, "Benchmark database of accurate (MP2 and CCSD (T) complete basis set limit) interaction energies of small model complexes, DNA base pairs, and amino acid pairs," *Phys. Chem. Chem. Phys.* **8**, 1985–1993 (2006).
- ⁴³F. Weigend, M. Kattannek, and R. Ahlrichs, "Approximated electron repulsion integrals: Cholesky decomposition versus resolution of the identity methods," *J. Chem. Phys.* **130**, 164106 (2009).
- ⁴⁴A. Halkier, T. Helgaker, P. Jørgensen, W. Klopper, H. Koch, J. Olsen, and A. K. Wilson, "Basis-set convergence in correlated calculations on Ne, N₂, and H₂O," *Chem. Phys. Lett.* **286**, 243–252 (1998).
- ⁴⁵Benchmark data for S22 reaction set, 2024.
- ⁴⁶L. A. Burns, M. S. Marshall, and C. D. Sherrill, "Appointing silver and bronze standards for noncovalent interactions: A comparison of spin-component-scaled (SCS), explicitly correlated (F12), and specialized wavefunction approaches," *J. Chem. Phys.* **141**, 234111 (2014).
- ⁴⁷Y. S. Al-Hamdan, P. R. Nagy, A. Zen, D. Barton, M. Kállay, J. G. Brandenburg, and A. Tkatchenko, "Interactions between large molecules pose a puzzle for reference quantum mechanical methods," *Nat. Commun.* **12**, 3927 (2021).
- ⁴⁸S. F. Boys and F. Bernardi, "The calculation of small molecular interactions by the differences of separate total energies. Some procedures with reduced errors," *Mol. Phys.* **19**, 553–566 (1970).
- ⁴⁹S. Blaschke and S. Stopkowicz, "Cholesky decomposition of complex two-electron integrals over GIAOs: Efficient MP2 computations for large molecules in strong magnetic fields," *J. Chem. Phys.* **156**, 044115 (2022).
- ⁵⁰M. Suewattana, W. Purwanto, S. Zhang, H. Krakauer, and E. J. Walter, "Phaseless auxiliary-field quantum Monte Carlo calculations with plane waves and pseudopotentials: Applications to atoms and molecules," *Phys. Rev. B* **75**, 245123 (2007).
- ⁵¹Y. Huang, Z. Guo, H. Q. Pham, and D. Lv, "GPU-accelerated auxiliary-field quantum Monte Carlo with multi-slater determinant trial states," *arXiv:2406.08314v1* (2024).
- ⁵²R. Assaraf and M. Caffarel, "Zero-variance principle for Monte Carlo algorithms," *Phys. Rev. Lett.* **83**, 4682–4685 (1999).
- ⁵³Y. Chen, L. Zhang, W. E, and R. Car, "Hybrid auxiliary field quantum Monte Carlo for molecular systems," *J. Chem. Theory Comput.* **19**, 4484–4493 (2023).
- ⁵⁴Y.-Y. He, M. Qin, H. Shi, Z.-Y. Lu, and S. Zhang, "Finite-temperature auxiliary-field quantum Monte Carlo: Self-consistent constraint and systematic approach to low temperatures," *Phys. Rev. B* **99**, 045108 (2019).
- ⁵⁵T. Shen, Y. Liu, Y. Yu, and B. M. Rubenstein, "Finite temperature auxiliary field quantum Monte Carlo in the canonical ensemble," *J. Chem. Phys.* **153**, 204108 (2020).
- ⁵⁶R. Blankenbecler, D. J. Scalapino, and R. L. Sugar, "Monte Carlo calculations of coupled boson–fermion systems. I," *Phys. Rev. D* **24**, 2278–2286 (1981).
- ⁵⁷J. E. Hirsch, "Two-dimensional Hubbard model: Numerical simulation study," *Phys. Rev. B* **31**, 4403–4419 (1985).

- ⁵⁸R. R. d. Santos, "Introduction to quantum Monte Carlo simulations for fermionic systems," *Braz. J. Phys.* **33**, 36–54 (2003).
- ⁵⁹A. Macridin, "Phonons, charge and spin in correlated systems," Ph.D. thesis, University of Groningen, 2003.
- ⁶⁰D. J. J. Marchand, P. C. E. Stamp, and M. Berciu, "Dual coupling effective band model for polarons," *Phys. Rev. B* **95**, 035117 (2017).
- ⁶¹M. H. Kalos, D. Levesque, and L. Verlet, "Helium at zero temperature with hard-sphere and other forces," *Phys. Rev. A* **9**, 2178–2195 (1974).
- ⁶²T. Holstein, "Studies of polaron motion: Part I. The molecular-crystal model," *Ann. Phys.* **8**, 325–342 (1959).
- ⁶³A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems* (Curran Associates, Inc., 2019), Vol. 32, pp. 8024–8035.
- ⁶⁴T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," [arXiv:1604.06174v2](https://arxiv.org/abs/1604.06174v2) (2016).
- ⁶⁵T. H. Dunning, Jr., "Gaussian basis sets for use in correlated molecular calculations. I. The atoms boron through neon and hydrogen," *J. Chem. Phys.* **90**, 1007–1023 (1989).
- ⁶⁶S. L. Shostak, W. L. Ebenstein, and J. S. Muenter, "The dipole moment of water. I. Dipole moments and hyperfine properties of H₂O and HDO in the ground and excited vibrational states," *J. Chem. Phys.* **94**, 5875–5882 (1991).
- ⁶⁷F. Shimizu, "Stark spectroscopy of NH₃ ν₂ band by 10-μ CO₂ and N₂O lasers," *J. Chem. Phys.* **52**, 3572–3576 (1970).
- ⁶⁸J. Muenter, "Electric dipole moment of carbon monoxide," *J. Mol. Spectrosc.* **55**, 490–491 (1975).
- ⁶⁹F. Lovas, E. Tiemann, J. Coursey, S. Kotchigova, J. Chang, K. Olsen, and R. Dragoset, "Diatom spectral database," <https://dx.doi.org/10.18434/T4T59X> (2003).
- ⁷⁰Y. Garniron, T. Applencourt, K. Gasperich, A. Benali, A. Ferté, J. Paquier, B. Pradines, R. Assaraf, P. Reinhardt, J. Toulouse, P. Barbaresco, N. Renon, G. David, J.-P. Malrieu, M. Vérit, M. Caffarel, P.-F. Loos, E. Giner, and A. Scemama, "Quantum package 2.0: An open-source determinant-driven suite of programs," *J. Chem. Theory Comput.* **15**, 3591–3609 (2019).
- ⁷¹H. G. A. Burton and P.-F. Loos, "Rationale for the extrapolation procedure in selected configuration interaction," *J. Chem. Phys.* **160**, 104102 (2024).
- ⁷²P.-F. Loos, Y. Damour, and A. Scemama, "The performance of CIPSI on the ground state electronic energy of benzene," *J. Chem. Phys.* **153**, 176101 (2020).
- ⁷³B.-X. Zheng, J. S. Kretchmer, H. Shi, S. Zhang, and G. K.-L. Chan, "Cluster size convergence of the density matrix embedding theory and its dynamical cluster formulation: A study with an auxiliary-field quantum Monte Carlo solver," *Phys. Rev. B* **95**, 045103 (2017).
- ⁷⁴B. Eskridge, H. Krakauer, and S. Zhang, "Local embedding and effective down-folding in the auxiliary-field quantum Monte Carlo method," *J. Chem. Theory Comput.* **15**, 3949 (2019).