Received: 5 February 2020 | Revised: 17 April 2020 | Accepted: 4 May 2020

DOI: 10.1002/wcms.1482

ADVANCED REVIEW

WIREs COMPUTATIONAL MOLECULAR SCIENCE WILEY

DFTpy: An efficient and object-oriented platform for orbital-free DFT simulations

Xuecheng Shao[1] | Kaili Jiang[1] | Wenhui Mi[1] | Alessandro Genova[1,3] | Michele Pavanello[1,2]

[1]Department of Chemistry, Rutgers University, Newark, New Jersey

[2]Department of Physics, Rutgers University, Newark, New Jersey

[3]Kitware Inc., 1712 U.S. 9 Suite 300, Clifton Park, New York, New York

Correspondence
Michele Pavanello, Department of Physics, Rutgers University, 101 Warren Street, Newark, NJ 07102.
Email: m.pavanello@rutgers.edu

Xuecheng Shao, Wenhui Mi, and Kaili Jiang, Department of Chemistry, Rutgers University, Newark, New Jersey.
Email: xs161@newark.rutgers.edu (X.S.), wenhui.mi@rutgers.edu (W.M.), and kj385@newark.rutgers.edu (K.J.)

Funding information
Basic Energy Sciences, Grant/Award Number: DE-SC0018343; Directorate for Computer and Information Science and Engineering, Grant/Award Number: ACI-1931473; Division of Chemistry, Grant/Award Number: CHE-1553993

Abstract

In silico materials design is hampered by the computational complexity of Kohn–Sham DFT, which scales cubically with the system size. Owing to the development of new-generation kinetic energy density functionals (KEDFs), orbital-free DFT (OFDFT) can now be successfully applied to a large class of semiconductors and such finite systems as quantum dots and metal clusters. In this work, we present DFTpy, an open-source software implementing OFDFT written entirely in Python 3 and outsourcing the computationally expensive operations to third-party modules, such as NumPy and SciPy. When fast simulations are in order, DFTpy exploits the fast Fourier transforms from PyFFTW. New-generation, nonlocal and density-dependent-kernel KEDFs are made computationally efficient by employing linear splines and other methods for fast kernel builds. We showcase DFTpy by solving for the electronic structure of a million-atom system of aluminum metal which was computed on a single CPU. The Python 3 implementation is object-oriented, opening the door to easy implementation of new features. As an example, we present a time-dependent OFDFT implementation (hydrodynamic DFT) which we use to compute the spectra of small metal clusters recovering qualitatively the time-dependent Kohn–Sham DFT result. The Python codebase allows for easy implementation of application programming interfaces. We showcase the combination of DFTpy and ASE for molecular dynamics simulations of liquid metals. DFTpy is released under the MIT license.

This article is categorized under:
Software > Quantum Chemistry
Electronic Structure Theory > Density Functional Theory
Data Science > Computer Algorithms and Programming

KEYWORDS

DFT, materials science, TDDFT

1 | INTRODUCTION

1.1 | Theoretical background

Orbital-free Density Functional Theory (OFDFT) is an emerging technique for modeling materials (bulk and nanoparticles) with an accuracy nearing the one of Kohn–Sham DFT (KSDFT) and with an algorithm that is almost linear scaling, $O(N\log(N))$, both in terms of work and memory requirements.[1–3] The most efficient OFDFT software[2–5] can approach million-atom system sizes while still accounting for the totality of the valence electrons. The central ingredient to OFDFT is the employment of pure Kinetic Energy Density Functionals (KEDFs). Commonly adopted KEDF approximants are not accurate enough to describe strongly directional chemical bonds—a category which unfortunately includes most molecules.[6,7] However, new-generation nonlocal KEDFs allow OFDFT to model quantum dots and semiconductors.[8,9] Hence, OFDFT is to be considered an emerging technique for computational materials science, chemistry, and physics.

In OFDFT, the electronic structure is found by direct minimization of the DFT Lagrangian,

$$\mathcal{L}[\rho] = E[\rho] - \mu\left(\int \rho(\mathbf{r})\mathrm{d}\mathbf{r} - N_\mathrm{e}\right), \tag{1}$$

where $E[\rho]$ is the electronic energy density functional, and N_e the number of valence electrons, taking the form,

$$E[\rho] = T_\mathrm{s}[\rho] + E_\mathrm{H}[\rho] + E_\mathrm{xc}[\rho] + \int v_\mathrm{ext}(\mathbf{r})\rho(\mathbf{r})\ \mathrm{d}\mathbf{r}, \tag{2}$$

where E_H is the Hartree energy, E_xc the exchange-correlation (xc) energy, T_s the noninteracting kinetic energy and $v_\mathrm{ext}(\mathbf{r})$ is the external potential (in OFDFT, typically given by local pseudopotentials).

Minimization of the Lagrangian with respect to the electron density function, $\rho(\mathbf{r})$, yields the density of the ground state. In other words,

$$\rho(\mathbf{r}) = \underset{\rho}{\operatorname{argmin}}\{\mathcal{L}[\rho]\}. \tag{3}$$

Equivalently, $\rho(\mathbf{r})$ can be obtained by solving the Euler–Lagrange equation,

$$\frac{\delta E[\rho]}{\delta\rho(\mathbf{r})} - \mu = 0, \tag{4}$$

which is expanded as follows

$$\frac{\delta T_s[\rho]}{\delta\rho(\mathbf{r})} + v_s(\mathbf{r}) - \mu = 0, \tag{5}$$

where we grouped $v_s(\mathbf{r}) = \frac{\delta E_H[\rho]}{\delta\rho(\mathbf{r})} + \frac{\delta E_{xc}[\rho]}{\delta\rho(\mathbf{r})} + v_\mathrm{ext}(\mathbf{r})$.

In conventional KSDFT, the KEDF potential, $\frac{\delta T_s[\rho]}{\delta\rho(\mathbf{r})}$, is not evaluated and instead the kinetic energy is assumed to be only a functional of the KS orbitals which in turn are functionals of the electron density. In OFDFT, the KEDF potential is available by direct evaluation of the functional derivative of an approximate KEDF. Thus, the Euler equation (5) can be tackled directly. In practice, most often the minimization of the Lagrangian in Equation (1) is carried out directly with Equation (5) as its gradient.

1.2 | OFDFT software background

In this work, we present DFTpy, a flexible and object-oriented implementation of OFDFT. The software builds all the needed energy and potential terms so that the minimization of the energy functional can be carried out. The optimization itself can be done by several commonly adopted nonlinear, multivariable optimizers (such as quasi-Newton methods).

DFTpy situates itself in a fairly uncultivated field, as unlike KSDFT, OFDFT software are few.[2,4,10–12] As most projects, DFTpy started out as a toy project collecting Python 3 classes defining NumPy.Array subclasses and associated methods for handling functions on regular grids. Functionalities included interpolations and conversion between file types. This was released under the moniker PBCpy.[13] The next step for DFTpy came in 2018 when classes related to the basic energy terms in materials were developed. Hartree energy based on NumPy's fast Fourier transforms (FFTs), exchange-correlation, and KEDF functionals based on pyLibXC.[14] Efforts to formalize the previous implementation culminated in recent months with a strong focus on the efficiency of the codebase for its application to million atom systems.

The current state-of-the-art in OFDFT software is PROFESS,[2] GPAW,[10] ATLAS,[12] and DFT-FE.[4] GPAW, DFT-FE, and ATLAS are real-space codes implementing either finite-element or finite-difference methods. Similarly to PROFESS, DFTpy relies on Fourier space not only for the treatment of Coulomb interactions but also for the computation of gradient and Laplacian operations (needed for instance for the von Weizsäcker term).

The distinguishing new features of DFTpy lie in its object-oriented core design composed of several important abstractions: Grid, Field (i.e., functions on grids), FunctionalClass (i.e., an abstraction encoding an energy functional). These enable fast implementations of new functionalities. As an example, in this work, we showcase a new time-dependent OFDFT[15–18] implementation for the computation of optical spectra within an OFDFT framework, and an application programming interface (API) combining DFTpy with Atomic Simulation Environment (ASE),[19] which is a set of tools and Python modules for atomistic simulations, for the realization of molecular dynamics simulations.

More specifically, DFTpy distills efficient methods for the computation of structure factors via the smooth particle-mesh Ewald method,[20,21] and an in-house, line-search-based electron density optimization algorithm which has the ability to dynamically adjust the effective grid cutoff during the optimization. To the best of our knowledge, DFTpy contains the most efficient implementation to date of new-generation nonlocal KEDFs. These functionals are known to give a major boost to the performance of semilocal and nonlocal KEDFs but are associated with an unsustainable increase in the computational cost. DFTpy solves the problem by implementing an evaluation of the KEDF functional derivative (potential) that exploits linear splines, bringing down the computational cost to less than 20 times the one of a GGA KEDF.

The paper is organized as follows. We first describe the most important classes defining the DFTpy codebase. We proceed to describe the core aspects responsible for the efficient implementation. Finally, we provide the reader with two examples: first showcasing DFTpy timings and linear scalability with system size and then DFTpy's ease of implementation of new methods by presenting a time-dependent OFDFT implementation that we apply to the computation of optical spectra of small metallic clusters.

2 | CLASSES AND SOFTWARE WORKFLOW

2.1 | DFTpy classes

DFTpy bases itself on PBCpy, a collection of classes for handling functions and fields of arbitrary rank in periodic boundary conditions.[13] PBCpy's main classes are Grid and Field which are both NumPy.Array subclasses.

The Grid class (comprising of BaseGrid, DirectGrid, and ReciprocalGrid) is aware of all the attributes needed to define a grid, such as the lattice vectors, the number of space discretization points in each direction. Subclasses of Grid include RealSpaceGrid and ReciprocalSpaceGrid which are self-explanatory.

The Field class (comprising of BaseField, DirectField, and ReciprocalField) encodes a function defined on a Grid. There are several methods bound to this class. For example, if Field is defined on a Direct/ReciprocalGrid it contains .fft/.ifft, the forward/inverse Fourier Transform. Additional bound methods include spline interpolations, integrals, and the appropriately extended definitions of the common algebraic operations (=, +, *, /). Fields can be of arbitrary rank. For instance, the electron density is a rank one field, while the density gradient is a rank three field whether they are represented in real or reciprocal space.

DFTpy features classes, such as FunctionalClass for the evaluation of the various terms in the energy: the kinetic energy density functional, KEDF, the exchange-correlation functional, `XC`, the electron-ion local pseudopotential, IONS, and the Hartree functional, HARTREE. Such a software structure is compatible with virtually all types of

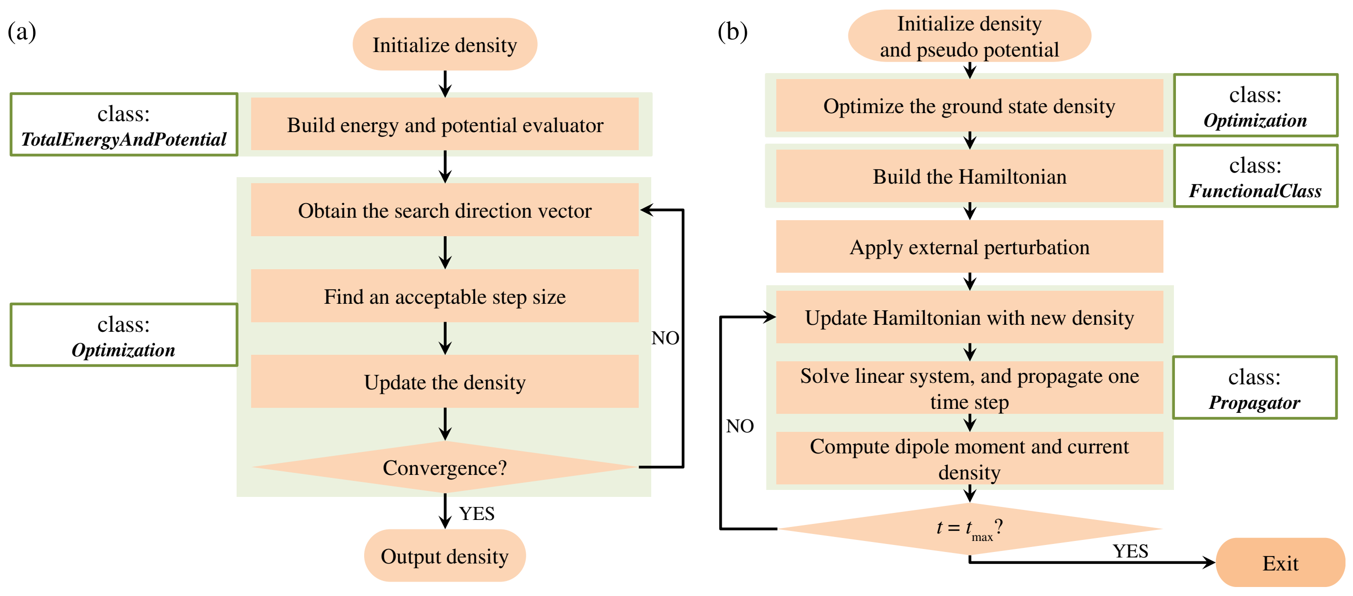
(a)
Initialize density
class: TotalEnergyAndPotential
Build energy and potential evaluator
Obtain the search direction vector
Find an acceptable step size
class: Optimization
Update the density
Convergence?
NO
YES
Output density
(b)
Initialize density and pseudo potential
Optimize the ground state density
class: Optimization
Build the Hamiltonian
class: FunctionalClass
Apply external perturbation
Update Hamiltonian with new density
Solve linear system, and propagate one time step
class: Propagator
Compute dipole moment and current density
NO
t = t_max?
YES
Exit

FIGURE 1 Flowcharts for (a) a density optimization job and (b) a TD-OFDFT job (see text for details). On the side of the flowcharts, we indicate in green boxes the names of the Python classes involved

electronic structure methods, and not only OFDFT. Thus, we expect in future releases of DFTpy to also include KSDFT as well as APIs at the level of the energy functional for external KSDFT codes and particularly those offering efficient Python interfaces.[10,22,23]

2.2 | Other classes and APIs

DFTpy contains classes for handling the optimization of the electron density and for handling the user interface. The optimization class is a standard optimizer which will probably be spun off as its own module in later releases. The user interface consists of a Python dictionary collecting the parameters for the calculation and an API to ASEs input/output geometry handler.[19] With ASE, DFTpy can read and write virtually any file format.

DFTpy has been conceived to ease the developments of new methods and to leverage the many modules already available. Too often junior scientists spend time reinventing common software simply because their platform is not flexible enough to interface easily with other modules. We showcase this with a simple example, using the capability of ASE to run molecular dynamics with DFTpy as the external engine. We developed a DFTpyCalculator class which is in the form of an ASE Calculator class, set in the ASE.Atoms class. In Section 3, we present a simple example of MD simulation carried out with DFTpy+ASE.

Two workflow examples are given in Figure 1. In inset (a) of the figure, we show a flowchart describing the main steps carried out by a density optimization job. Only 3 classes are involved: FunctionalClass, EnergyAndPotentialEvaluator and Optimization. Figure 1b shows the flowchart of a TD-OFDFT job for which an additional class is required. Namely, Propagator needed for handling the TDDFT propagation step (vide infra). Examples and Jupyter Notebooks related to the density optimization class and the DFTpy+ASE API are available at DFTpy's manual[24] and git repository.[25]

3 | DETAILS ENABLING COMPUTATIONAL EFFICIENCY

3.1 | New-generation nonlocal KEDFs

New-generation nonlocal KEDFs began with the breakthrough development of the Huang-Carter functional (HC) in 2010.[26] For the first time, this functional could reliably approach semiconductors and inhomogeneous systems with a robust algorithm. Unfortunately, HC was deemed too computationally expensive to become a workhorse for realistically sized model systems. This prompted a number of additional development by several groups,[27–30] including our recent work.[8,31] In this section, we will focus on the functionals developed by our group, and specifically LMGP and LWT family of functionals. However, the techniques and conclusions drawn here are general and encompass other new-generation functionals, such as HC[26] and LDAK.[30]

In general, the nonlocal KEDFs consist of three terms: Thomas-Fermi (TF),[32–34] von Weizsäcker (vW),[35] and nonlocal term. First, the local TF term is given by

$$T_{\mathrm{TF}}[\rho] = \frac{3}{10}\left(3\pi^2\right)^{2/3}\int \rho^{5/3}(\mathbf{r})d\mathbf{r}. \tag{6}$$

Second, the semilocal vW term is written as

$$T_{\mathrm{vW}}[\rho] = \int \sqrt{\rho(\mathbf{r})}\left(-\frac{1}{2}\nabla^2\right)\sqrt{\rho(\mathbf{r})}d\mathbf{r}. \tag{7}$$

Finally, the nonlocal term of the KEDFs is given by:

$$T_{\mathrm{NL}}[\rho] = \int \rho^{\alpha}(\mathbf{r})\omega_{\mathrm{NL}}[\rho](\mathbf{r},\mathbf{r}')\rho^{\beta}(\mathbf{r}')d\mathbf{r}d\mathbf{r}', \tag{8}$$

where α and β are suitable parameters, and $\omega_{\mathrm{NL}}[\rho](\mathbf{r}, \mathbf{r}')$ is a kernel usually assumed to be a function of only $|\mathbf{r} - \mathbf{r}'|$ and as such is represented in reciprocal space by a one-dimensional function, $\omega_{\mathrm{NL}}(q)$. When the Wang–Teter functional is used,[36]

$$\omega_{\mathrm{NL}}(q) = \omega_{\mathrm{WT}}(q) = C_{\mathrm{WT}}G_{\mathrm{NL}}(\eta(q)) \tag{9}$$

where $\eta(q) = \frac{q}{2k_{\mathrm{F}}}$ with $k_{\mathrm{F}} = (3\pi^2\rho)^{\frac{1}{3}}$ is the Fermi wavevector, $C_{\mathrm{WT}} = \frac{6}{25}\left(3\pi^2\right)^{2/3}$ and $G_{\mathrm{NL}}(\eta)$ is defined in Equation (11). The WT functional can be improved to satisfy functional integration relations[31] by the addition of one correction term giving rise to the MGP family of functionals. Namely,

$$\omega_{x,y}(q) = \omega_{\mathrm{WT}}(q) - x\mathrm{C}_{\mathrm{WT}}\int_0^1 dt\, t^y \frac{dG_{NL}(\eta(q,t))}{dt}. \tag{10}$$

where

$$G_{NL}(\eta) = \left(\frac{1}{2} + \frac{1-\eta^2}{4\eta}\ln\left|\frac{1+\eta}{1-\eta}\right|\right)^{-1} - 3\eta^2 - 1. \tag{11}$$

MGP is given by $(x, y) = (1, 5/6)$, MGPA by $(x, y) = (1/2, 5/6)$ and MGPG by $(x, y) = (1, 5/3)$. The only difference between MGP/A/G is the way a kernel is symmetrized. We refer the interested reader to the supplementary information of Reference 31.

In Reference 8, we developed a technique to generalize WT as well as MGP/A/G functionals to approach localized, finite systems by invoking spline techniques to obtain kernels no longer dependent only on the average electron density but instead dependent locally on the full electron density function. The resulting functionals are dubbed LWT, LMGP/A/G depending on the kernels mentioned in Equations (9) and (10).

The implementation of these new functionals in DFTpy requires the following four steps (s1–s4):

s1. Determine the maximum/minimum value of k_{F} and generate a set of k_{F} values in the $\left[k_F^{\max}, k_F^{\min}\right]$ interval by an arithmetic or geometric progression. This is an $O(N)$ operation with a very small prefactor.

s2. Evaluate the kernel for each of the k_{F}s using splines either in real or reciprocal space. At the beginning of computation, the kernel is calculated at some discrete points of η. This calculation is done only once. The kernel evaluation is an $O(N)$ operation with a potentially large prefactor depending on the type of spline used.

# atoms	13	171	1,099	12,195
T_{NL}	87.14	227.22	759.64	8,010.22
$T_{\mathrm{TF}} + T_{\mathrm{vW}}$	6.15	16.54	50.83	463.36

TABLE 1 Timings for the evaluation of new-generation KEDFs (LMGP) with DFTpy for Al clusters of varying sizes

Note: 40 k_{F} values between $k_{\mathrm{F}}^{\min}$ and $k_{\mathrm{F}}^{\max}$ generated with an arithmetic progression is used in all systems. The kernel is interpolated using the nearest-neighbor method and the linear spline is employed to interpolate the nonlocal KEDF potential over the k_{F} values.

s3. Compute the KEDF potential from Equation (8) with the different kernels. This is an $O(N\log N)$ operation due to the FFTs needed to evaluate the convolution in Equation (8).
s4. Interpolate the KEDF potential over the values of k_{F} onto $k_{\mathrm{F}}[\rho(\mathbf{r})]$ to obtain the final nonlocal KEDF potential and energy. This is an $O(N)$ operation with a potentially large prefactor depending on the type of spline used.

The timings associated with T_{NL} (we chose LMGP[8]) compared to $T_{\mathrm{TF}} + T_{\mathrm{vW}}$ are summarized in Table 1 for Aluminum clusters of different sizes ranging from 13 to 12,195 atoms. The structures are generated with ASE adopting a 15 Å vacuum layer in each direction to ensure the interactions between atoms and their periodic images are negligible. Inspecting Table 1, we note that the cost of the additional nonlocal functionals is less than 20 times the one of the semilocal functionals. Thus, it is reasonable to conclude that DFTpy's implementation of new-generation nonlocal KEDFs opens the door to predictive, ab initio simulations of mesoscale systems (>10 nm).

We should make the following remarks: (a) The results presented in Table 1 are a reference only to isolated systems. For bulk systems, a much smaller k_{F} grid is needed and the cost is therefore much reduced. Testing shows that the cost becomes less than half of the one in the table for similarly sized bulk systems. (b) The arithmetic progression used to generate the η grid can be improved and optimized. For example, we found that using geometric progressions can reduce the number of needed η points and thus further reduce the cost compared to Table 1. (c) If the values of $k_F^{\min}$ and $k_F^{\max}$ are kept constant throughout the density optimization, convergence, and the algorithm's stability further improve.

3.2 | Density optimization strategies

Finding a solution to Equation (4) is nontrivial. A stable optimization method is found by recasting Equation (4) in terms of $\psi(\mathbf{r}) = \sqrt{\rho(\mathbf{r})}$,[37]

$$\frac{\delta E[\psi^2]}{\delta \psi(\mathbf{r})} - 2\mu\psi(\mathbf{r}) = 0, \tag{12}$$

in this way, there is no need to impose the constraint, $\rho(\mathbf{r}) > 0$.

The algorithms employed to carry out the optimization have a long history and in many respects, they determine the computational efficiency of the entire OFDFT code. In DFTpy, we follow the common prescription. Given an initial $\psi(\mathbf{r})$, the following steps are repeated until convergence is reached:

1. Obtain the search direction vector $p_{\mathrm{k}}(\mathbf{r})$ with an optimization method of choice (e.g., conjugated gradient).
2. Find an acceptable step size λ_{k} along the vector $p_{\mathrm{k}}(\mathbf{r})$ using a line search strategy.
3. Generate a new $\psi_{\mathrm{k}+1}(\mathbf{r})$ from $\psi_{\mathrm{k}}(\mathbf{r})$, λ_{k}, and $p_{\mathrm{k}}(\mathbf{r})$.

For Step 1, three main types of optimization methods are implemented in DFTpy: nonlinear conjugate gradient (CG),[38–44] limited memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS)[45] and truncated Newton (TN) methods.[46] We tested the TN method to be the fastest method in DFTpy for most systems. However, in many instances (e.g., isolated systems), the TN method incurs into a high failure rate. Because, in L-BFGS, there is a need to store the last several updates of ψ and gradient, the memory cost is larger than for other methods. CG, instead, is the most stable among these methods, with several available options for updating p_{k}. In DFTpy, line search can be performed by the algorithms in SciPy.Optimize.

There are two ways to carry out an optimization: one is direct minimization of the energy functional, and another is the optimization of the residual [i.e., the result of the evaluation of Equation (12)]. The optimizing function, ψ_{k+1}, can be updated by $\psi_{k+1} = \psi_k + \lambda_k p_k$, then normalized to N_e. However, such a scaling scheme is not always stable. An alternative approach is to use an orthogonalization scheme prescribing p_k to be orthogonal to ψ_k and normalized to N_e. The update can take the form[47] $\psi_{k+1} = \psi_k \cos(\lambda_k) + p_k \sin(\lambda_k)$.

For those systems with inhomogeneous electron densities (such as clusters), convergence is very slow and can be very time consuming. For this reason, in DFTpy we implemented a multi-step density optimization scheme. In this scheme, the number of grid points needed to represent the electron density is determined dynamically and typically increases together with the optimization steps. We start out by carrying out a full density optimization on a coarse grid and then we interpolate the converged density onto a finer grid leading to substantial savings. For example, if the grid spacing of the coarse grid is twice larger than the finer grid, the timing is decreased by 1/8. For this scheme, the bigger the density inhomogeneities in the ground state density, the greater the efficiency improvement. In the next section, we will present an analysis of the timings and overall computational savings yielded by the new multi-step optimization method.

3.3 | Leveraging existing techniques

DFTpy leverages fast algorithms, such as FFTs for Fourier transforms,[48] and Particle-Mesh Ewald (PME) scheme for the computation of ionic structure factors.[21,49,50] These are the most time-consuming operations when large scale simulations are targeted.[51,52]

For FFTs, DFTpy encodes two modules: Numpy.fft and pyFFTW.[53] While Numpy.fft is a portable FFT implementation, pyFFTW is perhaps the most efficient FFT under a Python environment that shares the same interface of Numpy.fft. As FFTs are one of the most time-consuming operations, it is worth to further improve them. For example, Reikna[54] seems to offer a better interface to PyCUDA (the Python APIs for CUDA software to run on GPUs) compared to pyFFTW. Additionally, Google's TensorFlow[55] also provides a GPU enabled FFT implementation (via Cuda FFT). Thus, in future DFTpy releases, we will develop APIs to both Reikna and TensorFlow modules.

Regarding the computation of the ionic structure factor, when a large number of ions is considered in the simulation (e.g., more than 1,000 ions), the vanilla $O(N^2)$ method is no longer viable and, instead, the PME method is commonly employed. To the best of our knowledge, there are no tested, open-source Python modules for PME. Thus, DFTpy has an in-house PME implementation, taking advantage of SciPy methods when possible. However, this may change in future releases if such a PME Python module had to become available.

4 | TIMINGS AND ASSESSMENT OF EFFICIENCY

Throughout this section, the calculations are carried out with the bulk-derived local pseudopotentials[56] (BLPS) and optimal effective local pseudopotentials (OEPP),[57] and the LDA xc functional parametrization by Perdew and Zunger.[58] Timing tests are performed starting from a face-centered cubic (fcc) Aluminum crystal with a lattice constant of 4.05 Å, and a kinetic energy cutoff of 600 eV. This is sufficient to converge the total energy to below 1 meV/atom.

4.1 | Optimization of the electron density

Figure 2a shows the total wall times required for the electron density optimization of systems containing up to ∼10,000 Al atoms using several optimization methods: CG, TN (regular, residual minimization, and scaled, i.e., normalizing the density to the number of electrons), and L-BFGS. All methods show an approximate linear scaling execution time with system size. TN performs better than L-BFGS and CG methods. The residual minimization (RM) scheme with TN method, also presented in the figure, performs comparably to the energy minimization, and the scaling scheme shows good performance. We conclude that TN provides the most efficient optimization. Thus, TN is adopted for all the following calculations of bulk systems.

The performance of the multi-step density optimization scheme described in 3.2 in comparison to a vanilla density optimization of Al clusters is shown in Figure 2b, inset. In the calculation, we used the same structures as in

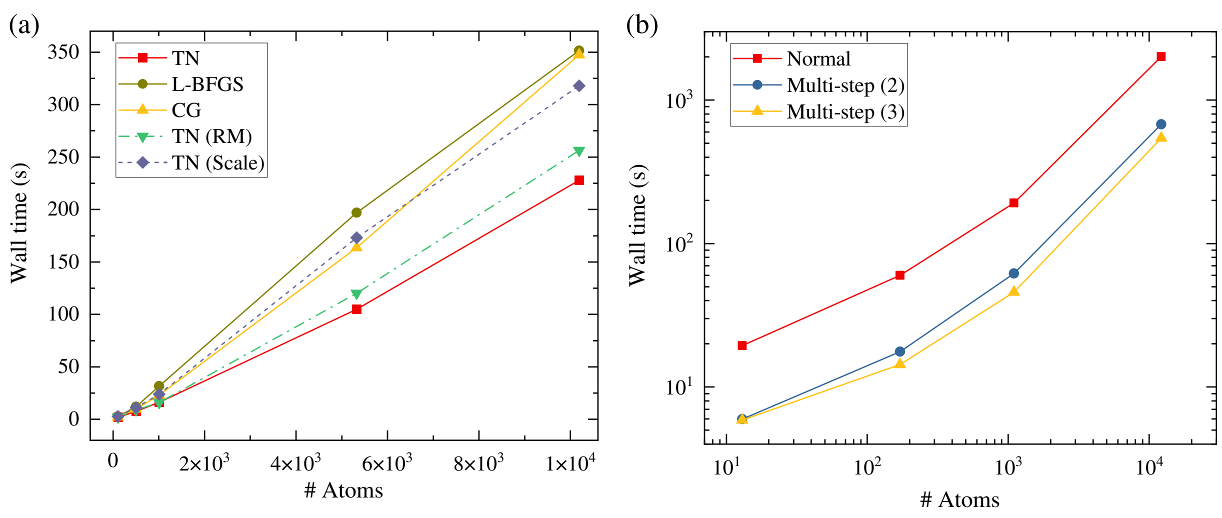

FIGURE 2 Timings (wall time) for density optimizations carried out with different optimization methods. (a) Comparing optimizers on bulk Al supercells. (b) Comparing two- and three-step optimization to a vanilla density optimization for Al clusters

Table 1, and for KEDF, we use $T_{TF} + T_{vW}$. For each step, CG is found to be more stable than TN method for isolated systems and is used in the density optimization. The results show that the multi-step scheme speeds up the calculation by a factor of 2, demonstrating the high-efficiency of this multi-step scheme. In particular, a two-step scheme already brings most of the achievable savings, and a three-step scheme further improves, even though by a much smaller margin.

4.2 | Linear scalability up to 1 million atoms

OFDFT methods are developed because they hold the promise to be able to describe realistically sized systems. In materials science, typical system sizes considered by the experiments involve thousands to well over millions of atoms. Will KSDFT ever be able to approach such systems? While it is hard to make a prediction at this particular point in history with quantum computing and machine learning spearheading new and potentially disruptive avenues of exploration, it is clear that current KSDFT algorithms (with exception of divide and conquer methods leveraging a mixture of KSDFT and OFDFT such as subsystem DFT[59]) and software are far from being able to approach million-atom system sizes. OFDFT is developed to precisely fill this gap.[3,50]

DFTpy enters this playing field with an essentially single-core implementation (possibly enhanced by multithreading from OpenMP implementations of some underlying modules which are, however, not used in this work). We stress here that a single core is perhaps all that is needed when system sizes of such dimensions are approached. This is because the complexity of sampling becomes a true computational bottleneck. Several thousands or even millions of structures need to be sampled in large-scale simulations, which make farming-type parallelization more efficient than single executions of parallel codes.

To our knowledge, the largest system size ever approached by single-processor OFDFT software is 13,500 atoms.[51] At the same time, the largest system ever approached by parallel OFDFT codes reached ∼4 million atoms using 2,048 processors.[52] To test the computational usefulness and efficiency of DFTpy, we perform a density optimization on the fcc Al supercell up to 1,000,188 atoms with a single processor. The total time and time-per-call for the total potential as a function of the number of atoms are presented in Figure 3. From the figure, we can see that DFTpy still shows approximately linear scaling behavior with the number of atoms even for the large systems considered. The total time for simulating the ∼1-million atom system on a single core is only ∼32 hr and can be further reduced to ∼20 hr by using the slightly lower cutoff of 500 eV which can still converge the total energy to within 1 meV/atom. We also notice that the FFT only accounts for ∼25% of the total time, and surprisingly the time cost of T_{TF} and LDA exchange-correlation are comparable to the FFT.

Thus, even though Python brings many important qualities to the developed software, it also poses a few headaches. The example just mentioned shows that operations as simple as the power (i.e., $a = b^c$, involved in the evaluation of

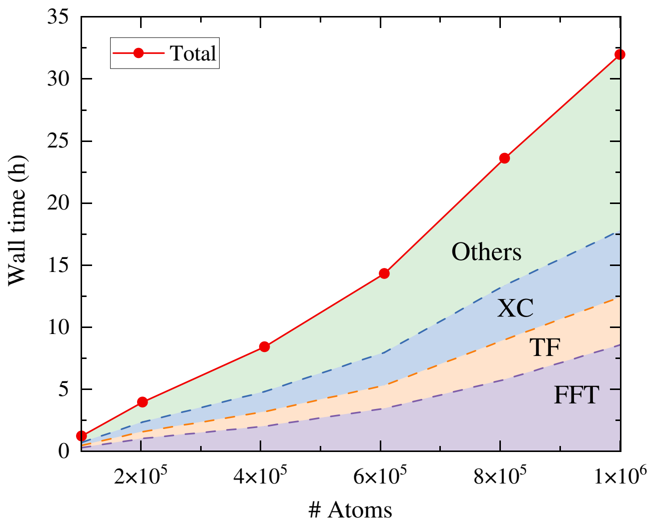

FIGURE 3 Timings (wall time) for density optimization on fcc aluminum for systems up to 1 million atoms with the truncated Newton method

TABLE 2 Bulk properties of fcc Al calculated by KSDFT and OFDFT methods

	V_0	E_0	B_0
KSDFT	15.644	−57.951	82.94
OFDFT	15.819	−57.934	84.97
Relaxation	15.821	−57.934	–

Note: V_0 is the equilibrium volume (Å^3/atom), E_0 is the total energy (eV/atom), and B_0 is the bulk modulus (GPa). "Relaxation" refers to values obtained by OFDFT after a structure relaxation using DFTpy+ASE.

LDA functionals such as T_{TF} and Dirac's exchange) can be inefficient when NumPy is used. Even though this comes at a linear cost, the prefactor is substantial, making the evaluation of the Thomas–Fermi functional much too expensive as seen in the figure. This issue will be tackled in future releases of DFTpy, for example by employing Pythran[60] or low-level languages for such operations.

4.3 | DFTpy+ASE: Dynamics of liquid aluminum

Molecular dynamics (MD) is a widely used simulation technique in materials science and chemistry, useful to study structural and dynamic properties of materials. It is quite straightforward to develop an API that combines DFTpy and ASE to perform MD simulations.

To showcase this API, we target a known success story for OFDFT. That is the simulation of structure and dynamics of liquid metals, and particularly liquid Al. In Table 2, we first show that DFTpy with the Wang–Teter (WT) functional is capable of predicting the correct equation of state for bulk Al. The equilibrium bulk structure is found numerically as well as via optimization (again, carried out via DFTpy+ASE) which agree with the fitted results from total energy values.

We then proceeded to carry out MD simulations in the canonical ensemble (i.e., constant number of particles, volume and temperature: NVT) for liquid Al at the experimental density 2.35 g/cm^3 and the temperature of 1,023 K.[61] We first consider a small system size of 108 atoms and then we also tackle a 1,372 atom system. The time step used is 2 fs, and a Langevin thermostat[62] is used. Except a uniform density as the initial guess density in the first step, the initial density is given by optimization density of the previous step in the following steps, which further reduces the wall time. Figure 4 shows that our simulation results are in very good agreement with experimental data. DFTpy simulates the 108 atoms for 20,000 steps in only 37,368 s (∼10 hr). To study finite-size effects on the $g(r)$, we also carried out a simulation with a larger cell containing 1,372 atoms. The results in the figure show that finite-size effects are negligible for this system. Here, $g(r)$ were averaged over 10,000 steps after equilibration.

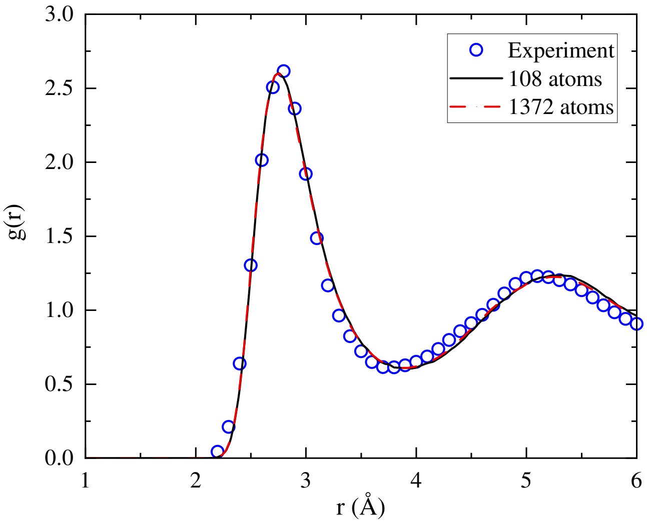

FIGURE 4 Pair distribution functions $g(r)$ for liquid Al at experimental conditions compared to X-ray diffraction data[61]

5 | EASE OF IMPLEMENTATION OF NEW METHODS

5.1 | Time-dependent OFDFT

The hydrodynamic approach to time-dependent DFT (TD-DFT) has shown great promise for understanding plasmonics,[17,63,64] and the response of bulk metals,[65] metal surfaces,[66,67] and metal clusters.[15,68,69] Its applications, however, have been limited to model systems, such as jellium,[67] jellium spheres,[15] and other models.[64] Even though these models are useful, as they provide a qualitative picture of the physics, a predictive and quantitative model can only be achieved when the atomistic details of the systems are taken into account. This is exactly our aim in this new implementation in DFTpy. Thus, in this section, we present an implementation of atomistic hydrodynamic TD-DFT which we call TD-OFDFT, hereafter.

The theory follows closely OFDFT,[65,70] and introduces a "collective orbital" $\psi(\mathbf{r})$, where $|\psi(\mathbf{r})|^2 = \rho(\mathbf{r})$. We then solve the associated Schrödinger-like equation. Namely,

$$\hat{H}\psi(\mathbf{r}) = \mu\psi(\mathbf{r}), \tag{13}$$

where

$$\hat{H} = -\frac{1}{2}\nabla^2 + \frac{\delta T_{\mathrm{S}}^{\mathrm{Pauli}}}{\delta\rho(\mathbf{r})} + v_{\mathrm{S}}(\mathbf{r}). \tag{14}$$

The Laplacian term comes from the minimization of the von Weizsäcker (vW) term, $T_{\mathrm{S}}^{\mathrm{vW}}$. $T_{\mathrm{S}}^{\mathrm{Pauli}} = T_{\mathrm{S}} - T_{\mathrm{S}}^{\mathrm{vW}}$ is the remaining part of the noninteracting kinetic energy and is included in the TD-DFT effective potential.

A similar approach can be formulated for the time-dependent extension[68,71] requiring the current density $\mathbf{j}(\mathbf{r}, t) = \rho(\mathbf{r}, t)\nabla S(\mathbf{r}, t)$, where $S(\mathbf{r}, t)$ is a scalar velocity field. Thus, we write the time-dependent collective orbital in the form of $\psi(\mathbf{r},t) \equiv \sqrt{\rho(\mathbf{r},t)}e^{iS(\mathbf{r},t)}$, and then solve a time-dependent Schrödinger-like equation,

$$i\frac{\partial\psi(\mathbf{r},t)}{\partial t} = \hat{H}\psi(\mathbf{r},t). \tag{15}$$

Following Equation (14), the Hamiltonian in the above equation has the form,

$$\hat{H} = -\frac{1}{2}\nabla^2 + \frac{\delta T_{\mathrm{S}}^{\mathrm{Pauli}}}{\delta\rho(\mathbf{r},t)} + v_{\mathrm{S}}(\mathbf{r},t), \tag{16}$$

which we implement in the adiabatic LDA (ALDA) approximation.

This formalism can be exploited in several flavors: real-time propagations,[65,68] and perturbatively.[72] In this work, we choose the former, as described in the following section. We remark recent efforts to achieve an exact factorization of the wavefunction into marginal and conditional terms[73,74] leading to a one-electron-like equation similar to Equation (15).

5.1.1 | Implementation of real-time TD-OFDFT

We implemented a Crank–Nicolson propagator with a predictor–corrector to any desired order. The relevant equation to solve for this implicit propagator is,[75]

$$\left(1-i\frac{\mathrm{d}t}{2}\hat{H}\right)\psi(t+\mathrm{d}t)=\left(1+i\frac{\mathrm{d}t}{2}\hat{H}\right)\psi(t). \tag{17}$$

The real-time TD-OFDFT simulation follows the workflow:

1. Optimize the ground state density.
2. Build the Hamiltonian in Equation (14).
3. Apply an external perturbation to displace the system from the ground electronic state (vide infra).
4. Calculate the new potential with the density $\rho(t_\mathrm{i})$ and update the Hamiltonian with the new potential.
5. Solve the linear system in Equation (17), and propagate 1-time step from t_i to $t_{\mathrm{i}+1}$ (including the predictor–corrector step).
6. Compute dipole moment and current density.
7. Loop steps 4–6 until the total propagation time is reached.

A simple Jupyter notebook encoding the TD-OFDFT scheme is available in the *notebooks* section of the GitLab repository,[25] as well as in the *tutorials* tab of DFTpy's manual.[24]

5.1.2 | Optical spectra of Mg_8 and Mg_{50} clusters

We choose Mg metal clusters as the systems of interest. The system is optimized to its ground state density $\rho_0(\mathbf{r})$. At $t = 0$, we introduce a laser kick with strength k in the x-direction by setting the collective phase, $S(\mathbf{r}, t = 0) = -ikx$,

$$\psi(\mathbf{r},t=0)=\psi(\mathbf{r})e^{-ikx}, \tag{18}$$

where $\psi(\mathbf{r})=\sqrt{\rho_0(\mathbf{r})}$. We then propagate the system in real-time and obtain the time-dependent dipole moment change

$$\delta\mu(t)=\int\mathbf{r}(\rho(\mathbf{r},t)-\rho_0(\mathbf{r}))\mathrm{d}\mathbf{r}. \tag{19}$$

The oscillator strength is calculated using the following equation:

$$\sigma(\omega)=\omega\mathrm{Im}[\delta\tilde{\mu}(\omega)]. \tag{20}$$

For simplicity, we employ the Thomas–Fermi–von Weizsäcker functional,[35] which was shown to perform well for finite, isolated systems such as the metal clusters considered in this work.[76] We use the OEPP local pseudopotentials[57] and the Perdew–Zunger LDA xc functional.[58] A kinetic energy cutoff of 400 and 850 eV was used for Mg_8 and Mg_{50}, respectively. We indicate by TD-KSDFT the TD-DFT calculations carried out with the exact noninteracting kinetic energy (i.e., Kohn–Sham) which are performed with the embedded Quantum Espresso (eQE) code.[77] In follow-up work,

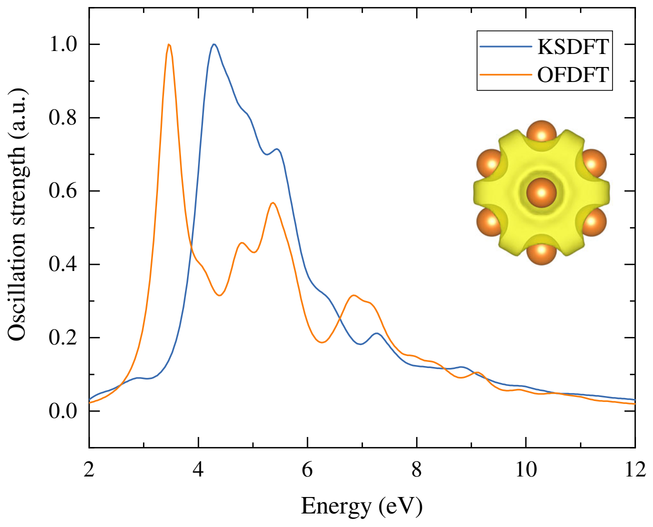

FIGURE 5 Comparison of optical spectra obtained with TD-OFDFT and TD-KSDFT for Mg_8. A view of the total density is given in the inset

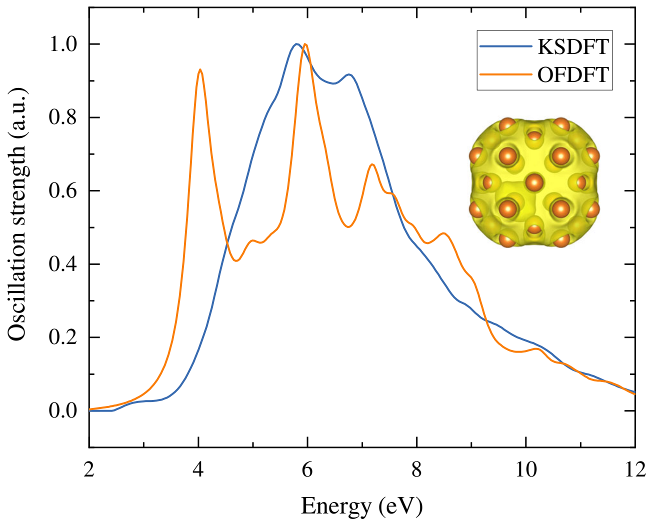

FIGURE 6 Comparison of optical spectra computed with TD-OFDFT and TD-KSDFT for Mg_{50}. A view of the total density is given in the inset

we will apply the TD-OFDFT technique to a wider class of clusters (metal and semiconducting) where we will be able to determine the ranges of applicability of TD-OFDFT.

Metal clusters have been a common application of time-dependent Thomas–Fermi methods, including hydrodynamic OFDFT.[70] The general consensus is that the larger the metal cluster, the closer the agreement with KSDFT. Banerjee and Harbola[15] showed that when OFDFT is applied to jellium spheres corresponding to cluster sizes of 100 atoms, the deviation between OFDFT and KSDFT in terms of the value of the static dipole polarizability goes below 20%. For cluster sizes corresponding to 1,000 atoms, the deviation goes below 2%.

Similarly, Figures 5 and 6 show that our TD-OFDFT calculations yield spectra for Mg_8 and Mg_{50} that are in fair agreement with TD-KSDFT.

The agreement, however, is stronger in the Mg_{50} system where the width and shape of the spectral envelope is better reproduced. The reason for such an agreement likely is the fact that Mg_{50} can develop a uniform electron gas-like electronic structure in its core, a type of structure well characterized by a single orbital. Even though the agreement for the spectra of Mg clusters are only qualitative, we expect TD-OFDFT to perform similarly or better for other metal clusters (e.g., Na clusters).[15,68]

5.1.3 | Comparison of KS-DFT and OFDFT orbitals

An interesting question is whether the collective orbitals recovered by the solution of Equation (13) resemble the KS orbitals. In principle, the collection of occupied and virtual KS orbitals form a complete basis, and so do the OFDFT

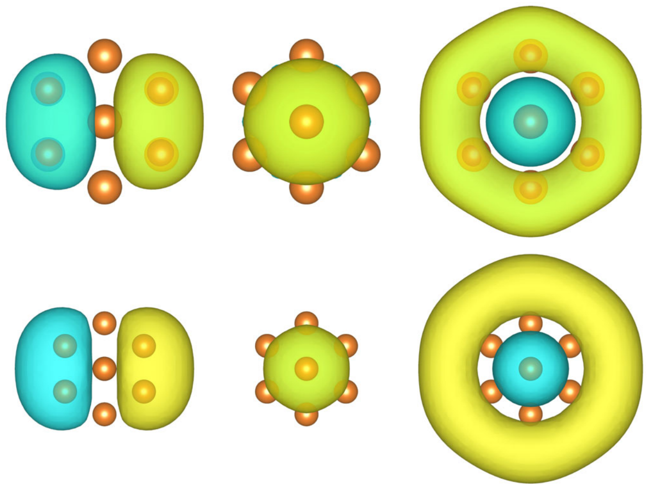
FIGURE 7 Mg_8 KSDFT virtual orbitals: LUMO, LUMO+2, and LUMO+6 (above). OFDFT "collective" virtual orbitals: LUMO, LUMO+3, and LUMO+6 (below)

TABLE 3 Distribution of the singular values of the overlap matrix, S_{ij}, between KS and OF-DFT orbitals for Mg_8

	Number of singular values	
Range	**Occ. + Virt.**	**Virt. Only**
0.9–1.0	14	7
0.8–0.9	2	1
0.7–0.8	1	1
0.6–0.7		1
0.5–0.6		1
0.4–0.5		1
0.3–0.4		4

Note: A selection of virtuals can be inspected in Figure 7.

collective orbitals. Thus, if we had to compare a large number of KS and collective orbitals, we would find that they span the exact same Hilbert space. For these reasons, we consider the Mg_8 system and limit the comparison to the low-lying orbitals. Specifically, we compare orbitals within 5.0 eV from the Fermi energy, which corresponds to the first peak in the optical spectra. These comprise 17 OFDFT collective orbitals (1 occupied and 16 virtual) and 32 KS orbitals (8 occupied and 24 virtual). Three KSDFT and OFDFT virtuals are displayed in Figure 7.

A direct comparison of OF and KS orbitals cannot be done visually. Therefore, we set up a rectangular overlap matrix, $S_{ij} = \left\langle \psi_i^{\mathrm{KS}} | \psi_j^{\mathrm{OF}} \right\rangle$ and compute its singular value decomposition. The distribution of the singular values are collected in Table 3.

If occupied and virtuals are included in the singular value decomposition, the OF orbitals can be essentially exactly represented as a linear combination of the KS orbitals (the majority of the singular values are close to 1). However, if only virtual orbitals within a 5.0 eV energy window from the Fermi energy are considered, the OF orbitals can only be partially decomposed into KS virtual orbitals. Thus, the comparison shows that OFDFT and KSDFT orbitals are similar if occupied and virtuals are compared. The virtual spaces, however, are only partially similar.

6 | CONCLUSIONS AND FUTURE DIRECTIONS

The Python revolution in computational electronic structure theory began almost two decades ago. It initially involved the emergence of wrappers for traditional software.[19,78,79] Initial attempts to output full-fledged quantum chemistry implementations came as early as 2004.[80] A defining moment was the 2015 release of PySCF[22] which featured an essentially complete quantum chemistry code (including advanced post-HF methods) with a software that leveraged C routines for Gaussian integrals and Python for essentially anything else.

With DFTpy, we merely follow this revolution, by developing a Python implementation for OFDFT simulations. The object-oriented nature of DFTpy provides an almost barrierless entry to advanced coding. We give an example of this by showcasing a new time-dependent OFDFT implementation and associated applications to the optical spectra of Mg clusters. In addition to the clear advantages compared to other, more traditional OFDFT codes based on low-level programming languages, DFTpy implements new-generation nonlocal KEDFs with density-dependent kernels in a fairly efficient way. An analysis of timings shows that the cost associated with the new nonlocal functionals is less than 20 times that of semilocal functionals when isolated systems are approached (such as surfaces or clusters) and less than seven times when bulk systems are considered. This is an important advance, making such functionals feasible for large scale simulations.

DFTpy classes and structure are general and could support a KS-DFT implementation and APIs to other Python codebases, such as PySCF, GPAW, and PSI4. In doing so, in the near future, we will implement a set of classes that will handle embedding schemes (from many-body expansions to density and quantum embedding). In this way, we will be able to seamlessly combine portions of a mesoscopic system computed at the OFDFT level and others at the KS-DFT level pushing the boundaries of time and length scales that can be approached by ab initio methods.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grants No. CHE-1553993 and ACI-1931473. The TD-OFDFT development is supported by the US Department of Energy, Office of Basic Energy Sciences, under Award Number DE-SC0018343. The authors acknowledge the Office of Advanced Research Computing (OARC) at Rutgers, The State University of New Jersey for providing access to the Amarel cluster and associated research computing resources that have contributed to the results reported here (URL: http://oarc.rutgers.edu).

CONFLICT OF INTEREST

The authors have declared no conflicts of interest for this article.

AUTHOR CONTRIBUTIONS

Xuecheng Shao: Conceptualization; data curation; software; supervision; writing-original draft. **Kaili Jiang:** Investigation; software; writing-original draft. **Wenhui Mi:** Conceptualization; investigation; writing-original draft. **Alessandro Genova:** Conceptualization; software. **Michele Pavanello:** Conceptualization; project administration; software; supervision; writing-original draft.

ORCID

Wenhui Mi https://orcid.org/0000-0002-1612-5292
Michele Pavanello https://orcid.org/0000-0001-8294-7481

RELATED WIREs ARTICLE

The Chronus Quantum software package

REFERENCES

1. Witt WC, Beatriz G, Dieterich JM, Carter EA. Orbital-free density functional theory for materials research. *J Mater Res*. 2018;*33*:777–795.
2. Chen M, Xia J, Huang C, et al. Introducing profess 3.0: An advanced program for orbital-free density functional theory molecular dynamics simulations. *Comput Phys Commun*. 2015;*190*:228–230.
3. Shao X, Xu Q, Wang S, Lv J, Wang Y, Ma Y. Large-scale ab initio simulations for periodic system. *Comput Phys Commun*. 2018;*233*: 78–83.
4. Gavini V, Knap J, Bhattacharya K, Ortiz M. Non-periodic finite-element formulation of orbital-free density functional theory. *J Mech Phys Solids*. 2007;*55*:669–696.
5. Chen M, Jiang X-W, Zhuang H, Wang L-W, Carter EA. Petascale orbital-free density functional theory enabled by small-box algorithms. *J Chem Theory Comput*. 2016;*12*:2950–2963.
6. Xia J, Carter EA. Density-decomposed orbital-free density functional theory for covalently bonded molecules and materials. *Phys Rev B*. 2012;*86*:235109.
7. Xia J, Huang C, Shin I, Carter EA. Can orbital-free density functional theory simulate molecules? *J Chem Phys*. 2012;*136*:084102.
8. Mi W, Pavanello M. Orbital-free dft correctly models quantum dots when asymptotics, nonlocality and nonhomogeneity are accounted for. *Phys Rev B*. 2019;*100*:041105(R).

9. Xu Q, Wang Y, Ma Y. Nonlocal kinetic energy density functional via line integrals and its application to orbital-free density functional theory. *Phys Rev B*. 2019;*100*:205132.
10. Lehtomäki J, Makkonen I, Caro MA, Harju A, Lopez-Acevedo O. Orbital-free density functional theory implementation with the projector augmented-wave method. *J Chem Phys*. 2014;*141*:234102.
11. Karasiev VV, Sjostrom T, Trickey S. Finite-temperature orbital-free DFT molecular dynamics: Coupling profess and quantum espresso. *Comput Phys Commun*. 2014;*185*:3240–3249.
12. Mi W, Shao X, Su C, et al. Atlas: A real-space finite-difference implementation of orbital-free density functional theory. *Comput Phys Commun*. 2016;*200*:87–95.
13. Genova A. Pbcpy: A python3 package providing some useful abstractions to deal with molecules and materials under periodic boundary conditions (pbc). 2018.
14. Lehtola S, Steigemann C, Oliveira MJ, Marques MA. Recent developments in libxc—A comprehensive library of functionals for density functional theory. *SoftwareX*. 2018;*7*:1–5.
15. Banerjee A, Harbola MK. Hydrodynamic approach to time-dependent density functional theory: Response properties of metal clusters. *J Chem Phys*. 2000;*113*:5614–5623.
16. Tokatly I, Pankratov O. Hydrodynamic theory of an electron gas. *Phys Rev B*. 1999;*60*:15550–15553.
17. Banerjee A, Harbola MK. Hydrodynamical approach to collective oscillations in metal clusters. *Phys Lett A*. 2008;*372*:2881–2886.
18. Zaremba E, Tso HC. Thomas-fermi-dirac-von weizsäcker hydrodynamics in parabolic wells. *Phys Rev B*. 1994;*49*:8147–8162.
19. Larsen AH, Mortensen JJ, Blomqvist J, et al. The atomic simulation environment—A python library for working with atoms. *J Phys Condens Matter*. 2017;*29*:273002.
20. Darden T, York D, Pedersen L. Particle mesh ewald: An n·log(n) method for ewald sums in large systems. *J Chem Phys*. 1993;*98*:10089–10092.
21. Essmann U, Perera L, Berkowitz ML, Darden T, Lee H, Pedersen LG. A smooth particle mesh ewald method. *J Chem Phys*. 1995;*103*:8577–8593.
22. Sun Q, Berkelbach TC, Blunt NS, et al. Pyscf: The python-based simulations of chemistry framework. *WIREs Comput Mol Sci*. 2018;*8*:e1340.
23. Smith DGA, Burns LA, Sirianni DA, et al. Psi4numpy: An interactive quantum chemistry programming environment for reference implementations and rapid development. *J Chem Theory Comput*. 2018;*14*:3504–3511.
24. Dftpy Manual. 2020. Available from: https://dftpy.rutgers.edu
25. Dftpy. 2020. Available from: https://gitlab.com/pavanello-research-group/dftpy
26. Huang C, Carter EA. Nonlocal orbital-free kinetic energy density functional for semiconductors. *Phys Rev B*. 2010;*81*:045206.
27. Constantin LA, Fabiano E, Sala FD. Nonlocal kinetic energy functional from the jellium-with-gap model: Applications to orbital-free density functional theory. *Phys Rev B*. 2018;*97*:205137.
28. Luo K, Karasiev VV, Trickey S. A simple generalized gradient approximation for the noninteracting kinetic energy density functional. *Phys Rev B*. 2018;*98*:041111.
29. Constantin LA, Fabiano E, Śmiga S, Della Sala F. Jellium-with-gap model applied to semilocal kinetic functionals. *Phys Rev B*. 2017;*95*:115153.
30. Xu Q, Lv J, Wang Y, Ma Y. Nonlocal kinetic energy density functionals for isolated systems obtained via local density approximation kernels. *Phys Rev B*. 2020;*101*:045110.
31. Mi W, Genova A, Pavanello M. Nonlocal kinetic energy functionals by functional integration. *J Chem Phys*. 2018;*148*:184107.
32. Thomas LH. The calculation of atomic fields. *Mathematical Proceedings of the Cambridge philosophical Society*. Volume 23. Cambridge, MA: Cambridge University Press, 1927; p. 542–548.
33. Fermi E. Statistical method to determine some properties of atoms. *Rend Accad Naz Lincei*. 1927;*6*:5.
34. Fermi E. Eine statistische methode zur bestimmung einiger eigenschaften des atoms und ihre anwendung auf die theorie des periodischen systems der elemente. *Z Phys*. 1928;*48*:73–79.
35. von Weizsäcker CF. Zur Theorie der Kernmassen. *Z Phys*. 1935;*96*:431–458.
36. Wang L-W, Teter MP. Kinetic-energy functional of the electron density. *Phys Rev B*. 1992;*45*:13196–13220.
37. Wang YA, Govind N, Carter EA. Orbital-free kinetic-energy density functionals with a density-dependent kernel. *Phys Rev B*. 1999;*60*:16350–16358.
38. Hestenes MR, Stiefel E. *Methods of conjugate gradients for solving linear systems*. Vol 49. Washington, DC: NBS, 1952.
39. Fletcher R, Reeves CM. Function minimization by conjugate gradients. *Comput J*. 1964;*7*:149–154.
40. Polak E, Ribiere G. Note sur la convergence de méthodes de directions conjuguées. *ESAIM: Math Model Num Anal*. 1969;*3*:35–43.
41. Polyak BT. The conjugate gradient method in extremal problems. *USSR Comput Math Math Phys*. 1969;*9*:94–112.
42. Fletcher R. *Practical methods of optimization: Vol. 1. Unconstrained optimization*. Hoboken, NJ: John Wiley & Sons, 1980.
43. Liu Y, Storey C. Efficient generalized conjugate gradient algorithms, part 1: Theory. *J Optim Theory Appl*. 1991;*69*:129–137.
44. Dai Y-H, Yuan Y. A nonlinear conjugate gradient method with a strong global convergence property. *SIAM J Optim*. 1999;*10*:177–182.
45. Liu DC, Nocedal J. On the limited memory bfgs method for large scale optimization. *Math Program*. 1989;*45*:503–528.
46. Nocedal J, Wright S. *Numerical optimization*. Berlin: Springer, 2006.
47. Jiang H, Yang W. Conjugate-gradient optimization method for orbital-free density functional calculations. *J Chem Phys*. 2004;*121*:2030–2036.

48. Frigo M, Johnson SG. The design and implementation of FFTW3. *Proc IEEE*. 2005;*93*:216–231.
49. Choly N, Kaxiras E. Fast method for force computations in electronic structure calculations. *Phys Rev B*. 2003;*67*:155101.
50. Hung L, Carter EA. Accurate simulations of metals at the mesoscale: Explicit treatment of 1 million atoms with quantum mechanics. *Chem Phys Lett*. 2009;*475*:163–170.
51. Ho GS, Lignères VL, Carter EA. Introducing profess: A new program for orbital-free density functional theory calculations. *Comput Phys Commun*. 2008;*179*:839–854.
52. Shao X, Mi W, Xu Q, Wang Y, Ma Y. O (n log n) scaling method to evaluate the ion–electron potential of crystalline solids. *J Chem Phys*. 2016;*145*:184110.
53. Gomersall H. pyfftw. 2016.
54. Opanchuk B. *Reikna, a pure python gpgpu library*. Melbourne: Swinburne University of Technology, 2019.
55. Abadi M, Barham P, Chen J, et al. Tensorflow: A system for large-scale machine learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, 2016; p. 265–283.
56. Huang C, Carter EA. Transferable local pseudopotentials for magnesium, aluminum and silicon. *Phys Chem Chem Phys*. 2008;*10*: 7109–7120.
57. Mi W, Zhang S, Wang Y, Ma Y, Miao M. First-principle optimal local pseudopotentials construction via optimized effective potential method. *J Chem Phys*. 2016;*144*:134108.
58. Perdew JP, Zunger A. Self-interaction correction to density-functional approximations for many-electron systems. *Phys Rev B*. 1981;*23*: 5048–5079.
59. Krishtal A, Sinha D, Genova A, Pavanello M. Subsystem density-functional theory as an effective tool for modeling ground and excited states, their dynamics and many-body interactions. *J Phys Condens Matter*. 2015;*27*:183202.
60. Guelton S, Brunet P, Amini M, Merlini A, Corbillon X, Raynaud A. Pythran: Enabling static optimization of scientific python programs. *Comput Sci Discov*. 2015;*8*:014001.
61. Waseda Y. *The structure of non-crystalline materials: Liquids and amorphous solids*. New York: McGraw-Hill, 1980.
62. Allen MP, Tildesley DJ. *Computer simulation of liquids*. Oxford: Oxford University Press, 2017.
63. Domps A, Suraud E, Reinhard P-G. Geometrical and quantal fragmentation of optical response in. *Eur Phys J D Atom Mol Opt Plasma Phys*. 1998;*2*:191–194.
64. Ciracì C, Sala FD. Quantum hydrodynamic theory for plasmonics: Impact of the electron density tail. *Phys Rev B*. 2016;*93*:205405.
65. White TG, Richardson S, Crowley BJB, Pattison LK, Harris JWO, Gregori G. Orbital-free density-functional theory simulations of the dynamic structure factor of warm dense aluminum. *Phys Rev Lett*. 2013;*111*:175002.
66. Bennett AJ. Influence of the electron charge distribution on surface-plasmon dispersion. *Phys Rev B*. 1970;*1*:203–207.
67. Liebsch A. *Electronic excitations at metal surfaces*. New York, NY: Springer US, 1997.
68. Domps A, Reinhard P-G, Suraud E. Time-dependent Thomas-fermi approach for electron dynamics in metal clusters. *Phys Rev Lett*. 1998;*80*:5520–5523.
69. Calvayrac F, Reinhard P-G, Suraud E, Ullrich C. Nonlinear electron dynamics in metal clusters. *Phys Rep*. 2000;*337*:493–578.
70. Harbola MK. Differential virial theorem and quantum fluid dynamics. *Phys Rev A*. 1998;*58*:1779–1782.
71. Giannoni M, Vautherin D, Veneroni M, Brink D. Variational derivation of nuclear hydrodynamics. *Phys Lett B*. 1976;*63*:8–10.
72. Banerjee A, Harbola MK. Calculation of van der waals coefficients in hydrodynamic approach to time-dependent density functional theory. *J Chem Phys*. 2002;*117*:7845–7851.
73. Abedi A, Maitra NT, Gross EKU. Exact factorization of the time-dependent electron-nuclear wave function. *Phys Rev Lett*. 2010;*105*: 123002. https://doi.org/10.1103/physrevlett.105.123002
74. Schild A, Gross E. Exact single-electron approach to the dynamics of molecules in strong laser fields. *Phys Rev Lett*. 2017;*118*:163202. https://doi.org/10.1103/physrevlett.118.163202
75. Castro A, Marques MAL, Rubio A. Propagators for the time-dependent Kohn-Sham equations. *J Chem Phys*. 2004;*121*:3425–3433.
76. Chan GK-L, Cohen AJ, Handy NC. Thomas–Fermi–Dirac–von Weizsäcker models in finite systems. *J Chem Phys*. 2001;*114*:631–638.
77. Genova A, Ceresoli D, Krishtal A, Andreussi O, DiStasio RA Jr, Pavanello M. Eqe: An open-source density functional embedding theory code for the condensed phase. *Int J Quantum Chem*. 2017;*117*:e25401.
78. Ong SP, Richards WD, Jain A, et al. Python materials genomics (pymatgen): A robust, open-source python library for materials analysis. *Comput Mater Sci*. 2013;*68*:314–319.
79. Jacob CR, Beyhan SM, Bulo RE, et al. PyADF—a scripting framework for multiscale quantum chemistry. *J Comput Chem*. 2011;*32*: 2328–2338.
80. Muller RP, PyQuante: Python Quantum Chemistry. 2004.

How to cite this article: Shao X, Jiang K, Mi W, Genova A, Pavanello M. DFTpy: An efficient and object-oriented platform for orbital-free DFT simulations. *WIREs Comput Mol Sci*. 2021;11:e1482. https://doi.org/10.1002/wcms.1482