The Journal of Open Source Software

Atomic Simulation Interface (ASI): application programming interface for electronic structure codes

Pavel V. Stishenko [1¶], **Thomas W. Keal** [2], **Scott M. Woodley** [3], **Volker Blum** [4], **Benjamin Hourahine** [5], **Reinhard J. Maurer** [6,7], **and Andrew J. Logsdail** [1¶]

1 Cardiff Catalysis Institute, School of Chemistry, Cardiff University, Cardiff, United Kingdom **2** Scientific Computing Department, STFC Daresbury Laboratory, Keckwick Lane, Daresbury, Warrington WA4 4AD, United Kingdom **3** Department of Chemistry, Kathleen Lonsdale Materials Chemistry, University College London, London, United Kingdom **4** Thomas Lord Department of Mechanical Engineering and Materials Science, Duke University, Durham, North Carolina 27708, United States **5** SUPA, Department of Physics, University of Strathclyde, John Anderson Building, 107 Rottenrow, Glasgow G4 0NG, United Kingdom **6** Department of Chemistry, University of Warwick, Coventry, CV4 7AL, United Kingdom **7** Department of Physics, University of Warwick, Coventry, CV4 7AL, United Kingdom ¶ Corresponding author

DOI: 10.21105/joss.05186

Software

- Review
- Repository
- Archive

Editor: Rachel Kurchin

Reviewers:

- @xwang862
- @junghans
- @srmnitc

Submitted: 12 December 2022
Published: 12 May 2023

Summary

The Atomic Simulation Interface (ASI) is a native C-style API for density functional theory (DFT) codes. ASI provides an efficient way to import and export large arrays that describe electronic structure (e.g. Hamiltonian, overlap, and density matrices) from DFT codes that are typically monolithic. The ASI API is designed to be implemented and used with minimal performance penalty, avoiding, where possible, unnecessary data copying. It provides direct access to the internal data structures of a code, and reuses existing data distribution over MPI nodes. The ASI API also defines a set of functions that support classical, AIMD (ab initio molecular dynamics), and hybrid QM/MM simulations: exporting potential energy, forces, atomic charges, and electrostatic potential at user defined points, as well as importing nuclear coordinates and arbitrary external electrostatic potentials. The ASI API is implemented in the DFTB+ (Hourahine et al., 2020) and FHI-aims (Blum et al., 2009) codes. A Python wrapper for easy access to ASI functions is also freely available (`asi4py`). We hope that the ASI API will be widely adopted and used for development of universal and interoperable DFT codes without sacrificing efficiency for portability.

Statement of need

Although numerous modern electronic structure codes have a common mathematical basis and often share core algorithm implementations (such as ESL (Oliveira et al., 2020), ELSI (Yu et al., 2018, 2020), libxc (Lehtola et al., 2018; Marques et al., 2012)), a portable and efficient way to access resulting electronic structure variables from the user side remains elusive. For classical, AIMD, and hybrid QM/MM calculations, a similar issue is solved by the widely adopted i-PI interface (Kapil et al., 2019), MolSSI Driver Interface (Barnes et al., 2021), ASE library (Larsen et al., 2017), and ChemShell environment (Lu et al., 2019), however, for electronic structure data, such as wave functions, band structure, Hamiltonian or density matrices, there is no widely adopted solution, probably due to diversity of basis sets. Therefore, many well-developed codes for electronic structure analysis and integrating machine learning are hard to employ due to their explicit dependence on the specific electronic structure code used. For example, the MPE solvent model (Filser et al., 2022) is implemented only in FHI-aims, there are SchNOrb (Schütt, Gastegger, et al., 2019) models that are available only for ORCA code (Neese et al.,

2020). Machine-learning models can be trained disregarding details of a specific electronic structure code's basis sets, therefore even without a universal representation of electronic quantities, a convenient and efficient way to access detailed electronic structure description will benefit the efforts towards modularization of electronic structure software.

State of the field

Demand to access electronic structure data is driven currently by the urge to apply recent machine-learning advances in the quantum chemistry field. After numerous successful applications of machine learning for direct prediction of energies and forces from atomic coordinates (Bartók et al., 2018; Z. Li et al., 2015; Schütt, Kessel, et al., 2019), data-driven models that predict electronic structure beyond energies and forces are being developed. Such data-driven models provide more interpretable outcomes, possess higher transferability, and can be used for prediction of a wider set of material properties. For example, Carleo & Troyer (2017) have employed reinforcement learning to compute ground-state and unitary time evolution of a few prototypical systems, and H. Li et al. (2018) have developed a deep-learning model that predicts a Hamiltonian matrix for subsequent DFTB calculations. The SchNOrb deep-learning framework uses neural tensor network representation of wave-functions for Hamiltonian matrix prediction (Schütt, Gastegger, et al., 2019).

Today, many electronic structure software packages use files to export or import information about electronic structure or potentials; this approach is implemented in ORCA (Neese et al., 2020), Quantum Espresso (Giannozzi et al., 2020), CP2K (Kühne et al., 2020), FHI-aims (Blum et al., 2009), DFTB+ (Hourahine et al., 2020), etc. Although file-based data exchange has advantages, it unavoidably introduces performance penalties and often takes additional coding efforts for data parsing and formatting, as storage formats are rarely good for active calculations. Rare exceptions such as GPAW (Mortensen et al., 2005), Psi4 (Turney et al., 2012), DFTK.jl (Herbst et al., 2021) do provide Python or Julia API's and thus simplify development of new functionality.

We believe the field will benefit from implementation of a universal API for access to DFT-related quantities in popular quantum chemistry codes. Even without universal specifications of basis sets, an API for access to Hamiltonian, overlap, and density matrices would be helpful for the applications mentioned above. It will pave the way to implementation of new machine learning models and electronic structure analysis tools, and will accelerate their adoption.

Functionality

The Atomic Simulation Interface (ASI) is a specification of pure C functions, designed to be implemented in existing quantum chemistry codes. The scope and capabilities of ASI mostly focus on efficient way to transfer electronic structure data. The complete ASI specification can be found on the project web page pvst.gitlab.io/asi. The plain C API was chosen for simplicity of implementation in Fortran codes and for simplicity of invocation from other languages including Python and Julia. For the sake of convenience, a Python wrapper for ASI functions has been created: `asi4py` is available for installation via `pip` (package installer for Python). The `asi4py` wrapper was designed to be used with the ASE framework (Larsen et al., 2017) implementing the ASE's `Calculator` interface. Therefore, any DFT code that implements ASI API automatically gets an ASE calculator with efficient data transfer.

There are four groups of key ASI functions that are briefly described in Table 1, united by their primary purpose: control flow, atomic information, electrostatic potential exchange, and transfer of large arrays describing electronic structure of the simulated system (currently routines for Hamiltonian, overlap and density matrices are included in the ASI specification). Table 1 lists only a subset of ASI functions, ommiting auxilary functions such as `ASI_get_basis_size`, `ASI_n_atoms`, *etc.*

Codes implementing the ASI API are expected to be built as a shared object library for dynamic linkage with the client code. Therefore, client codes get direct access to internal data structures of an ASI-implementing code, minimizing interoperability performance overhead. The suggested control-flow of a client code that employs ASI is shown in Figure 1.

Table 1. Key functions of the ASI API. For full list see ASI API Specification

Control flow	
`ASI_init`	Initialize calculation (load configuration files)
`ASI_run`	Do single-point calculation. Can be called multiple times with `ASI_set_atom_coords` for dynamics simulation or geometry optimization
`ASI_finalize`	Finalize calculations, free resources

Atomic information	
`ASI_set_atom_coords`	Sets atomic coordinates. Can be called (after `ASI_init`) multiple times.
`ASI_energy`	Returns total system energy.
`ASI_forces`	Returns pointer to the array of forces acting on atoms.
`ASI_atomic_charges`	Returns pointer to the array of atomic charges. Supported partitioning schemes depend on implementation

Electrostatic potential exchange	
`ASI_register_external_potential`	Set local electrostatic potential and its gradient at arbitrary points during calculations (during `ASI_run` call)
`ASI_calc_esp`	Calculate local electrostatic potential and its gradient in arbitrary points (after `ASI_run` call)

Electronic structure calculations	
`ASI_register_dm_init_callback`	Initialize SCF loop via density matrix
`ASI_register_dm_callback`	Get density matrix on each SCF iteration
`ASI_register_overlap_callback`	Get overlap matrix on each geometry change
`ASI_register_hamiltonian_callback`	Get Hamiltonian matrix on each SCF iteration

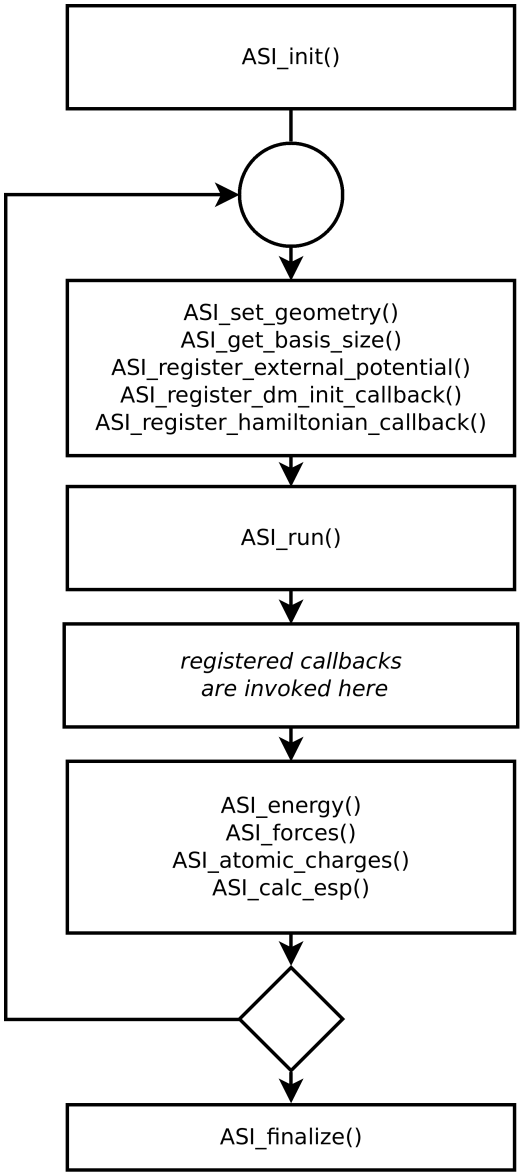

Figure 1: Suggested control-flow of a code that uses ASI API. The loop condition depends on the particular use case.

Given that the ASI API is expected to be implemented within existing codes using minimal changes necessary in a code base, we have made the group of control-flow functions as small as possible, preferring employment of callback functions. Registering a callback function gives the client code direct access to data objects of an ASI-implementing code and eliminates the need to copy them or to manage their lifetime. Callback functions that work with large matrices (Hamiltonian, overlap, and density matrices) support distributed storage via BLACS (Basic Linear Algebra Communication Subprograms) library (*Basic Linear Algebra Communication Subprograms*, n.d.). Each callback receives a BLACS descriptor of the matrix if MPI (Message Passing Interface (Walker & Dongarra, 1996)) parallelization is enabled. The dense storage format is currently supported, whilst support for sparse formats is expected in future versions.

The atomic information functions are designed to simplify calculation setup and for integration with classical simulation codes. The group of functions for electrostatic potential exchange are primarily meant for integration in QM/MM and AIMD workflows, for example into ChemShell

(Lu et al., 2019) framework or in IC-QMMM (Golze et al., 2013) calculations.

The group of functions for electronic structure calculations are designed to support development of new algorithms for density functional theory, for example density-matrix extrapolation (Polack et al., 2021), and employment of machine-learning techniques on the electronic structure level, such as SchNOrb model (Schütt, Gastegger, et al., 2019) or atomic cluster expansion of Hamiltonians (Zhang et al., 2022). New use cases are expected to emerge once the electronic structure properties are exposed via the ASI API.

Currently the ASI API is implemented in the open-source DFTB+ code, and in the FHI-aims code. Most of the ASI API is implemented in both codes with exception of `ASI_register_dm_init_callback` that is unavailable for DFTB+ due to different nature of the self-consistent loop in its algorithm. In addition, the set of supported charge partitioning schemes in the `ASI_atomic_charges` function depends on implementation. Although the currently available implementations both use localized basis sets, we do not foresee obstacles for ASI API implementation for plane-wave or hybrid basis sets.

For the sake of implementation simplicity, we tried to keep the number of ASI functions as small as possible. Therefore, implementations of client code in languages like C or Fortran can be cumbersome. To ease code development with the ASI API, we have created a Python wrapper for it: `asi4py`. `asi4py` provides a wrapping class `ASIlib` for a dynamically loaded shared library that implements ASI. That class forwards Python calls to native C calls via the `ctypes` library. Direct access to large matrices and arrays is provided via `NumPy` (Harris et al., 2020) arrays, so no data is copied during wrapping, and therefore performance overhead is minimal. For redistribution of BLACS matrices, access to necessary subroutines of the ScaLAPACK (Blackford et al., 1997) implementation linked with the loaded ASI library is provided via an additional package `scalapack4py`, so the existing BLACS contexts and MPI communicators are reused by a client code. To ease the calculation setup, an Atomic Simulation Environment (ASE) (Larsen et al., 2017) calculator interface is implemented by `ASI_ASE_calculator` class.

Use Case

A minimal example of ASI API usage with DFTB+ for access to Hamiltonian, overlap, and density matrices is shown below:

```
import os
import numpy as np
from ase.build import molecule
from asi4py import ASI_ASE_calculator

def write_input(asi):
  from ase.calculators.dftb import Dftb
  calc = Dftb(label='Some_cluster',
        Hamiltonian_SCC='Yes',
        Hamiltonian_MaxAngularMomentum_='',
        Hamiltonian_MaxAngularMomentum_O='"p"',
        Hamiltonian_MaxAngularMomentum_H='"s"')
  calc.write_input(asi.atoms, properties=['forces'])

atoms = molecule('H2O')

atoms.calc = ASI_ASE_calculator(os.environ['ASI_LIB_PATH'], write_input, None, atoms)
atoms.calc.asi.keep_density_matrix = True
atoms.calc.asi.keep_hamiltonian = True
atoms.calc.asi.keep_overlap = True
```

```
print(f'E = {atoms.get_potential_energy():.6f}')

S = atoms.calc.asi.overlap_storage[(1,1)]
H = atoms.calc.asi.hamiltonian_storage[(1,1)]
DM = atoms.calc.asi.dm_storage.get((1,1), None)
DM_cnt = atoms.calc.asi.dm_calc_cnt[(1,1)]

print(f'Number of electrons = {np.sum(S*DM):.6f}')
print(f'Sum of eigenvalues = {np.sum(H*DM):.6f}')
print(f'Number of iterations = {DM_cnt}')
```

In this example the `ASI_ASE_calculator` class is used for ASI library loading. The path to the library is specified in `ASI_LIB_PATH` environment variable. To create DFTB+ input files, the `ase.calculators.dftb.Dftb` class from ASE is used. The `ASIlib` object is aggregated by the calculator as `asi` property. Three boolean properties `keep_*` are used to enable storing of copies of corresponding matrices in dictionaries named `*_storage`. Keys of these dictionaries are pairs of 1-based indices of **k**-points and spin channels (always `(1,1)` for non-periodic spin-paired systems). The exemplar code just saves three matrices of a water molecule and prints out the system energy, number of electrons, sum of eigenvalues, and number of the density matrix evaluation iterations from the `dm_calc_cnt` dictionary.

Author Contribution Statement

Conceptualization by Andrew Logsdail and Reinhard Maurer. Coding and development by Pavel Stishenko with support from Ben Hourahine (DFTB+) and Volker Blum (FHI-aims). Project management and paper writing by all.

Acknowledgements

The ASI development was supported by ARCHER2 eCSE Programme (project eCSE03-10). A.L. acknowledges funding by the UKRI Future Leaders Fellowship Program (MR/T018372/1). R.M. acknowledges funding by the UKRI Future Leaders Fellowship Program (MR/S016023/1). Via our membership of the UK's HEC Materials Chemistry Consortium, which is funded by EPSRC (EP/R029431), this work used the ARCHER2 UK National Supercomputing Service (http://www.archer2.ac.uk). The authors are grateful to Mariana Rossi for her contribution to foundations of this work.

Citations

Barnes, T. A., Marin-Rimoldi, E., Ellis, S., & Crawford, T. D. (2021). The MolSSI driver interface project: A framework for standardized, on-the-fly interoperability between computational molecular sciences codes. *Computer Physics Communications*, *261*, 107688. https://doi.org/10.1016/j.cpc.2020.107688

Bartók, A. P., Kermode, J., Bernstein, N., & Csányi, G. (2018). Machine learning a general-purpose interatomic potential for silicon. *Phys. Rev. X*, *8*, 041048. https://doi.org/10.1103/PhysRevX.8.041048

Basic linear algebra communication subprograms. (n.d.). https://netlib.org/blacs/

Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., & Whaley, R. C. (1997). *ScaLAPACK users' guide*. Society for Industrial; Applied Mathematics.

Blum, V., Gehrke, R., Hanke, F., Havu, P., Havu, V., Ren, X., Reuter, K., & Scheffler, M. (2009). Ab initio molecular simulations with numeric atom-centered orbitals. *Computer Physics Communications*, *180*(11), 2175–2196. https://doi.org/10.1016/j.cpc.2009.06.022

Carleo, G., & Troyer, M. (2017). Solving the quantum many-body problem with artificial neural networks. *Science*, *355*(6325), 602–606. https://doi.org/10.1126/science.aag2302

Filser, J., Reuter, K., & Oberhofer, H. (2022). Piecewise multipole-expansion implicit solvation for arbitrarily shaped molecular solutes. *Journal of Chemical Theory and Computation*, *18*(1), 461–478. https://doi.org/10.1021/acs.jctc.1c00834

Giannozzi, P., Baseggio, O., Bonfà, P., Brunato, D., Car, R., Carnimeo, I., Cavazzoni, C., Gironcoli, S. de, Delugas, P., Ferrari Ruffino, F., Ferretti, A., Marzari, N., Timrov, I., Urru, A., & Baroni, S. (2020). Quantum ESPRESSO toward the exascale. *The Journal of Chemical Physics*, *152*(15), 154105. https://doi.org/10.1063/5.0005082

Golze, D., Iannuzzi, M., Nguyen, M.-T., Passerone, D., & Hutter, J. (2013). Simulation of adsorption processes at metallic interfaces: An image charge augmented QM/MM approach. *Journal of Chemical Theory and Computation*, *9*(11), 5086–5097. https://doi.org/10.1021/ct400698y

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Herbst, M. F., Levitt, A., & Cancès, E. (2021). DFTK: A julian approach for simulating electrons in solids. *Proc. JuliaCon Conf.*, *3*, 69. https://doi.org/10.21105/jcon.00069

Hourahine, B., Aradi, B., Blum, V., Bonafé, F., Buccheri, A., Camacho, C., Cevallos, C., Deshaye, M. Y., Dumitrică, T., Dominguez, A., Ehlert, S., Elstner, M., Heide, T. van der, Hermann, J., Irle, S., Kranz, J. J., Köhler, C., Kowalczyk, T., Kubař, T., … Frauenheim, T. (2020). DFTB+, a software package for efficient approximate density functional theory based atomistic simulations. *The Journal of Chemical Physics*, *152*(12), 124101. https://doi.org/10.1063/1.5143190

Kapil, V., Rossi, M., Marsalek, O., Petraglia, R., Litman, Y., Spura, T., Cheng, B., Cuzzocrea, A., Meißner, R. H., Wilkins, D. M., & others. (2019). I-PI 2.0: A universal force engine for advanced molecular simulations. *Computer Physics Communications*, *236*, 214–223.

Kühne, T. D., Iannuzzi, M., Del Ben, M., Rybkin, V. V., Seewald, P., Stein, F., Laino, T., Khaliullin, R. Z., Schütt, O., Schiffmann, F., Golze, D., Wilhelm, J., Chulkov, S., Bani-Hashemian, M. H., Weber, V., Borštnik, U., Taillefumier, M., Jakobovits, A. S., Lazzaro, A., … Hutter, J. (2020). CP2K: An electronic structure and molecular dynamics software package - quickstep: Efficient and accurate electronic structure calculations. *The Journal of Chemical Physics*, *152*(19), 194103. https://doi.org/10.1063/5.0007045

Larsen, A. H., Mortensen, J. J., Blomqvist, J., Castelli, I. E., Christensen, R., Dułak, M., Friis, J., Groves, M. N., Hammer, B., Hargus, C., Hermes, E. D., Jennings, P. C., Jensen, P. B., Kermode, J., Kitchin, J. R., Kolsbjerg, E. L., Kubal, J., Kaasbjerg, K., Lysgaard, S., … Jacobsen, K. W. (2017). The atomic simulation environment—a python library for working with atoms. *Journal of Physics: Condensed Matter*, *29*(27), 273002. https://doi.org/10.1088/1361-648x/aa680e

Lehtola, S., Steigemann, C., Oliveira, M. J. T., & Marques, M. A. L. (2018). Recent developments in libxc — a comprehensive library of functionals for density functional theory. *SoftwareX*, *7*, 1–5. https://doi.org/10.1016/j.softx.2017.11.002

Li, H., Collins, C., Tanha, M., Gordon, G. J., & Yaron, D. J. (2018). A density functional tight binding layer for deep learning of chemical hamiltonians. *Journal of Chemical Theory and*

Computation, *14*(11), 5764–5776. https://doi.org/10.1021/acs.jctc.8b00873

Li, Z., Kermode, J. R., & De Vita, A. (2015). Molecular dynamics with on-the-fly machine learning of quantum-mechanical forces. *Phys. Rev. Lett.*, *114*, 096405. https://doi.org/10.1103/PhysRevLett.114.096405

Lu, Y., Farrow, M. R., Fayon, P., Logsdail, A. J., Sokol, A. A., Catlow, C. R. A., Sherwood, P., & Keal, T. W. (2019). Open-source, python-based redevelopment of the ChemShell multiscale QM/MM environment. *Journal of Chemical Theory and Computation*, *15*(2), 1317–1328. https://doi.org/10.1021/acs.jctc.8b01036

Marques, M. A. L., Oliveira, M. J. T., & Burnus, T. (2012). Libxc: A library of exchange and correlation functionals for density functional theory. *Computer Physics Communications*, *183*(10), 2272–2281. https://doi.org/10.1016/j.cpc.2012.05.007

Mortensen, J. J., Hansen, L. B., & Jacobsen, K. W. (2005). Real-space grid implementation of the projector augmented wave method. *Phys. Rev. B*, *71*, 035109. https://doi.org/10.1103/PhysRevB.71.035109

Neese, F., Wennmohs, F., Becker, U., & Riplinger, C. (2020). The ORCA quantum chemistry program package. *The Journal of Chemical Physics*, *152*(22), 224108. https://doi.org/10.1063/5.0004608

Oliveira, M. J. T., Papior, N., Pouillon, Y., Blum, V., Artacho, E., Caliste, D., Corsetti, F., Gironcoli, S. de, Elena, A. M., García, A., García-Suárez, V. M., Genovese, L., Huhn, W. P., Huhs, G., Kokott, S., Küçükbenli, E., Larsen, A. H., Lazzaro, A., Lebedeva, I. V., … Yu, V. W. (2020). The CECAM electronic structure library and the modular software development paradigm. *The Journal of Chemical Physics*, *153*(2), 024117. https://doi.org/10.1063/5.0012901

Polack, É., Dusson, G., Stamm, B., & Lipparini, F. (2021). Grassmann extrapolation of density matrices for born–oppenheimer molecular dynamics. *Journal of Chemical Theory and Computation*, *17*(11), 6965–6973. https://doi.org/10.1021/acs.jctc.1c00751

Schütt, K. T., Gastegger, M., Tkatchenko, A., Müller, K.-R., & Maurer, R. J. (2019). Unifying machine learning and quantum chemistry with a deep neural network for molecular wavefunctions. *Nature Communications*, *10*(1). https://doi.org/10.1038/s41467-019-12875-2

Schütt, K. T., Kessel, P., Gastegger, M., Nicoli, K. A., Tkatchenko, A., & Müller, K.-R. (2019). SchNetPack: A deep learning toolbox for atomistic systems. *Journal of Chemical Theory and Computation*, *15*(1), 448–455. https://doi.org/10.1021/acs.jctc.8b00908

Turney, J. M., Simmonett, A. C., Parrish, R. M., Hohenstein, E. G., Evangelista, F. A., Fermann, J. T., Mintz, B. J., Burns, L. A., Wilke, J. J., Abrams, M. L., Russ, N. J., Leininger, M. L., Janssen, C. L., Seidl, E. T., Allen, W. D., Schaefer, H. F., King, R. A., Valeev, E. F., Sherrill, C. D., & Crawford, T. D. (2012). Psi4: An open-source ab initio electronic structure program. *WIREs Computational Molecular Science*, *2*(4), 556–565. https://doi.org/10.1002/wcms.93

Walker, D. W., & Dongarra, J. J. (1996). MPI: A standard message passing interface. *Supercomputer*, *12*, 56–68.

Yu, V. W., Campos, C., Dawson, W., García, A., Havu, V., Hourahine, B., Huhn, W. P., Jacquelin, M., Jia, W., Keçeli, M., Laasner, R., Li, Y., Lin, L., Lu, J., Moussa, J., Roman, J. E., Vázquez-Mayagoitia, Á., Yang, C., & Blum, V. (2020). ELSI — an open infrastructure for electronic structure solvers. *Computer Physics Communications*, *256*, 107459. https://doi.org/10.1016/j.cpc.2020.107459

Yu, V. W., Corsetti, F., García, A., Huhn, W. P., Jacquelin, M., Jia, W., Lange, B., Lin, L., Lu, J., Mi, W., & others. (2018). ELSI: A unified software interface for kohn–sham

electronic structure solvers. *Computer Physics Communications*, *222*, 267–285. https://doi.org/10.1016/j.cpc.2017.09.007

Zhang, L., Onat, B., Dusson, G., McSloy, A., Anand, G., Maurer, R. J., Ortner, C., & Kermode, J. R. (2022). Equivariant analytical mapping of first principles hamiltonians to accurate and transferable materials models. *Npj Computational Materials*, *8*(1). https://doi.org/10.1038/s41524-022-00843-2