The Journal of Chemical Physics

RESEARCH ARTICLE | JULY 03 2024

Integrated workflows and interfaces for data-driven semi-empirical electronic structure calculations

Special Collection: Modular and Interoperable Software for Chemical Physics

Pavel Stishenko ; Adam McSloy; Berk Onat ; Ben Hourahine ; Reinhard J. Maurer ; James R. Kermode ; Andrew Logsdail

J. Chem. Phys. 161, 012502 (2024)
https://doi.org/10.1063/5.0209742

The Journal of Chemical Physics SOFTWARE pubs.aip.org/aip/jcp

Integrated workflows and interfaces for data-driven semi-empirical electronic structure calculations

Cite as: J. Chem. Phys. 161, 012502 (2024); doi: 10.1063/5.0209742
Submitted: 22 March 2024 • Accepted: 7 June 2024 •
Published Online: 3 July 2024

Pavel Stishenko,[1] Adam McSloy,[2] Berk Onat,[2] Ben Hourahine,[3,a)] Reinhard J. Maurer,[4] James R. Kermode,[2,b)] and Andrew Logsdail[1,c)]

AFFILIATIONS

[1] Cardiff Catalysis Institute, School of Chemistry, Cardiff University, Park Place, Cardiff CF10 3AT, United Kingdom
[2] Warwick Centre for Predictive Modelling, School of Engineering, University of Warwick, Coventry CV4 7AL, United Kingdom
[3] SUPA, Department of Physics, John Anderson Building, University of Strathclyde, 107 Rottenrow, Glasgow G4 0NG, United Kingdom
[4] Department of Chemistry, University of Warwick, Coventry CV4 7AL, United Kingdom and Department of Physics, University of Warwick, Coventry CV4 7AL, United Kingdom

Note: This paper is part of the JCP Special Topic on Modular and Interoperable Software for Chemical Physics.
[a)] benjamin.hourahine@strath.ac.uk
[b)] j.r.kermode@warwick.ac.uk
[c)] **Author to whom correspondence should be addressed:** LogsdailA@cardiff.ac.uk

ABSTRACT

Modern software engineering of electronic structure codes has seen a paradigm shift from monolithic workflows toward object-based modularity. Software objectivity allows for greater flexibility in the application of electronic structure calculations, with particular benefits when integrated with approaches for data-driven analysis. Here, we discuss different approaches to create deep modular interfaces that connect big-data workflows and electronic structure codes and explore the diversity of use cases that they can enable. We present two such interface approaches for the semi-empirical electronic structure package, DFTB+. In one case, DFTB+ is applied as a library and *provides* data to an external workflow; in another, DFTB+ *receives* data via external bindings and processes the information subsequently within an internal workflow. We provide a general framework to enable data exchange workflows for embedding new machine-learning-based Hamiltonians within DFTB+ or enabling deep integration of DFTB+ in multiscale embedding workflows. These modular interfaces demonstrate opportunities in emergent software and workflows to accelerate scientific discovery by harnessing existing software capabilities.

I. INTRODUCTION

Semi-empirical electronic structure methods, such as Density Functional Tight-Binding (DFTB) theory,[1,2] have a long-standing history of enabling fast and robust predictions on a diverse range of materials for time and length scales.[2] The atomistic resolution accessible with DFTB (up to ~10^{18} atoms)[3] is traditionally out of reach for conventional first-principles calculations, making these approaches particularly appealing for large-scale simulation of dynamical chemical processes. Semi-empirical approaches can provide robust accuracy for conventional organic molecular materials[4,5] or inorganic materials.[6] Integration of these approaches in complex automated computational workflows and machine learning (ML) surrogate models is now timely given the impact of these new capabilities on computational materials science and chemistry over the last decade.[7–9]

The concept of using data-driven approaches to transfer first-principles information into second-principles electronic structure codes has a long history in the construction of tight binding parameterizations, for example, through global stochastic optimization of

tight-binding parameters or repulsive potentials via particle swarm optimization.[10–12] With the emergence of modern ML methods, the prospect of closer integration and direct learning between methods has been explored by several studies. As an example, Stöhr *et al.* have used ML interatomic potentials to represent the repulsive potential in DFTB,[13] and several studies have shown that ML surrogate models can accurately predict first-principles electronic structure in local orbital representation.[14–19] In the context of Density Functional Theory (DFT) and other semi-empirical methods, ML has also been used to represent electronic Hamiltonian parameters,[20,21] including the `TBmalt` approach, which provides end-to-end learning of parameters based on target properties.[22] The proof-of-principle applications show what is possible in this space, but standardized workflows or integrated approaches are yet to emerge.

The majority of well-established electronic structure software packages have developed as monolithic codebases with a single entry point, due to extensive investment in their development before the widespread adoption of integrated workflows. The consequence is a bottleneck for the integration between electronic structure and big-data approaches. Electronic structure software packages with modularity in their design that provide external accessibility to inner functionality have become more prominent in recent years, reflecting the evolving landscape of computational materials science and the uptake of objective, modular programming and data-driven workflows (e.g., `GPAW`,[23] `Psi4`,[24] and `pySCF`[25]) or UNIX philosophy[26] inspired designs such as `WIEN2k`;[27] furthermore, established packages have sought to reshape their designs with library components that allow execution through an externally driven interface. These retrofitted approaches typically rely on file input/output (I/O) and parsing, with only basic variable communication (e.g., MPI communicators), although alternative strategies with in-memory data transmission have increased in popularity.

Examples of established strategies include the automated building of deep PYTHON interfaces to Fortran codes using `f90wrap`[28] (e.g., `QEPy` for `Quantum Espresso`,[29] `quippy` for `QUIP`,[30] `CasPyTep` for `CASTEP`,[31] and a Python wrapper for the Bader code[32]), socket-type interfaces (`i-PI`,[33] `MDI`[34]), and library extraction (`tblite` in `CREST`,[35] `ELSI`[36]). High-level Python packages, such as the Atomic Simulation Environment (`ASE`),[37] have emerged as a *de facto* standard for building frameworks for atomistic simulation workflows. Examples of workflow engines include `AiiDA`,[38] `Chemoton`,[39] `CENSO`,[40] `BigChem`,[41] `QCEngine`,[34] etc., which enable some classes of high-level algorithms to be written once and reused between codes; however, this generality is commonly restricted to atomic and molecular properties rather than electronic structure.

Recently, the emergence of automated and machine-learning-augmented workflows and the establishment of extensive materials databases using FAIR principles, i.e., findable, accessible, interoperable, and reusable,[42] have led to a need for more flexible infrastructure capabilities in the design of electronic structure software. In particular, there is evidence of a clear need for greater modularity and interoperability in code design, which should support strong interfaces between electronic structure codes and external software packages. Such developments would circumvent traditional bottlenecks in data communication and accelerate discoveries facilitated by electronic structure theory.

Currently, there remains limited demonstration and standardization of deployable, user-ready interfaces that facilitate interaction and application of external workflows and data-driven frameworks with first-principles and semi-empirical electronic structure software packages. In particular, there is a need for deep module interfaces that can expose the extensive and well-developed functionality in existing, established software. The interfaces should be simple code-level interfaces rather than commonly shallow interfaces that work predominantly with complex file I/O or external scripted workflows. The ability to use such deep interfaces, which allow large data objects to be manipulated during run-time, will reward the community with computationally efficient approaches concerning both data processing and data storage. The list of software that may benefit from using deep interfaces includes charge partitioning codes,[43–45] electronic structure analysis,[46] and the growing family of electronic structure machine learning models.[18–20,22] The Atomic Simulation Interface (`ASI`)[47] is a recent example interface that is built to import and export electronic structure information from quantum chemistry codes during runtime with minimal performance penalty, as demonstrated via coupling with the `DFTB+` and `FHI-aims` software packages.[48]

The commonly articulated strategies for integration of electronic structure software packages can be categorized in order of depth and complexity of the interface as follows:

Data parsing via file I/O operations: This typically focuses on input and output data files only. This provides no data accessibility at the mid-process stage and has limited data precision and data object sizes.

Socket (and alternative) data transport protocols: Small data objects are communicated in byte format via a lower-level transport method.[33,34] This provides data-accessibility mid-process but is limited in terms of data object size.

Directly connecting to an API: An Application Programming Interface (API) provides a well-defined set of function calls. This provides data accessibility mid-process, can handle flexible data object sizes, and provides fixed API standards.

Flexible interfacial wrappers: An intermediate package manages couplings between different API standards across multiple languages. This provides data-accessibility mid-process and is flexible in both data object size and API definitions.

Despite the recognized disadvantages of file I/O-based interfacing, this approach is the most commonly used option. Reasons that file I/O interfacing remains popular include that the file system interface is universally provided by the operating system, that it does not impose any restrictions on data format, and that it is intrinsically asynchronous. These features are useful if codes on both ends of the interface are not specified *a priori*, which is especially applicable in the case of closed-source software. Motivations for deeper interfaces include the complexity of I/O data formats and the stability of their layout over time, the synchronization of I/O data, and inherent performance limitations when using I/O, especially at high frequency. The ongoing efforts toward standardization of input and output formats, such as `QCEngine`,[34] JavaScript Object Notation (JSON)[49] in `ORCA`,[50] or HDF5-based TREXIO,[51] offer the potential to alleviate some of these challenges.

Recent work on the `DFTB+` software package[52,53] has investigated how deep interfaces can be established and used for the

benefit of workflow-based computational simulation. Herein, we discuss general strategies for interfacing with electronic structure codes before presenting two interfaces that are capable of either being *driven by an external workflow* to provide data[47] or *driving a workflow* with external bindings used to obtain data from an external engine.[18] The interfaces follow different strategies and philosophies and address distinct potential use cases, such as embedding and ML workflows.

II. GENERAL CONSIDERATIONS ON DEEP INTERFACES TO ELECTRONIC STRUCTURE CODES

The contemporary difficulties of interfacing with electronic structure software packages are rooted in the assumption that the transient data objects that describe the electronic structure of a simulated system are not valuable outside of the code. Therefore, such data structures are often placed deep in the code foundations, and exposing these data structures for read or write operations typically requires intrusive changes in the core codebase.

An additional difficulty when interfacing with electronic structure software packages is the substantial size of electronic structure descriptors (electronic density, Kohn–Sham orbitals, and matrices of Kohn–Sham equations). Simulations often take up the majority of available random access memory, and transferring large amounts of data between formats or copying between codes can be limited by accessible memory and become a performance bottleneck. Unnecessary copying of the data between inter-operating codes can be avoided by providing direct access to the memory buffers via shared memory, memory-mapped files, or by running these codes as libraries in a single process; however, such approaches face other obstacles caused by data efficiency practices in high-performance computing, such as the reuse of large arrays for storage of various different data. For example, many routines in the Basic Linear Algebra Subprograms (`BLAS`) library return their results in their input buffers,[54] overwriting the data in repeated execution. If one wants to, e.g., export a large array, a pointer to the internal data buffer is insufficient, and instead the data must be accessed when the desired data are actually in that array. In practice, accessing the data in this manner means that the code execution must be paused and restarted at various (initially unplanned) moments, often deep in the call stack. Given that many electronic structure software packages were initially designed as standalone applications, such changes in the control flow can be intrusive and error-prone.

Approaches to suspend and restart a subroutine map onto two different code interfacing paradigms (see Fig. 1). One option is refactoring and splitting code into two separate subroutines that are called immediately before and after any data import or export. An alternative option is invoking a user-provided callback function to perform data read or write. The former method essentially converts the code into a library, inverting the control of the workflow; if the routine that is split is nested deep within the call stack, every function along the call stack must also be split. The latter method introduces local inversions of control through callbacks, but the modified code generally still drives the overall workflow.

In general, the considerations outlined may be viewed as significant drawbacks to deep interfaces, especially if the interfaced code is poorly structured, fragile, or tightly coupled; in such cases, the development and maintenance of deep interfaces may become involved and laborious. However, the potential benefits are significant and, if pursued carefully, present opportunities for realizing new science. In the following, we present realizations of the two highlighted deep interfacing paradigms inside the `DFTB+` software package, outlining our experiences with realization and showcasing the potential value.

III. METHODS

A. The DFTB family of models

The DFTB method,[1] and related semi-empirical tight-binding models,[55] approximate density functional theory (DFT). By expanding the Kohn–Sham functional around an approximate reference density, the total energy expressions are written as a sum of a (generally) attractive electronic band-structure contribution, an electrostatic energy, and a repulsive energy (which corresponds to the double-counting terms in DFT). The expressions for the electrostatic contributions are derived with respect to fluctuations from a reference charge density, which itself is assumed to be the sum of a set of neutral atomic densities corresponding to the structure being modeled. The electronic structure Hamiltonian itself is typically evaluated using reference neutral atoms and atomic dimers. The 2-center integrals are for a minimal, non-orthogonal, atomic valence basis (neglecting crystal field and 4-center contributions[1]). Depending on the choice of Hamiltonian (DFTB or xTB), these values are

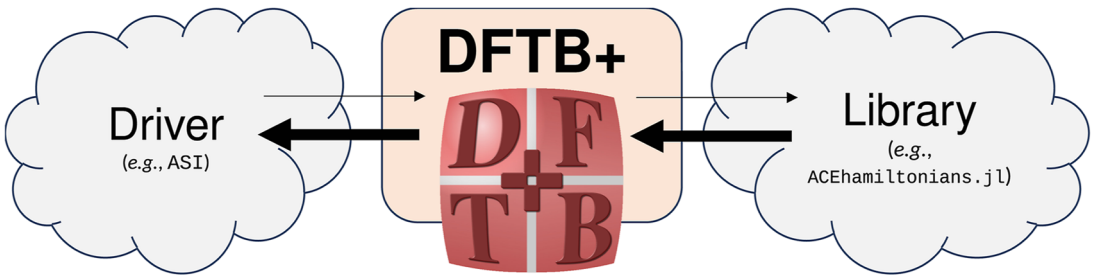

FIG. 1. Schematic representation of interfacing paradigms where `DFTB+` is used: as a resource, driven by an external package (left-hand side), or instead drives communication with an external package as part of its own workflow (right-hand side). In both cases, the majority of data communication is returned to the software driving the relationship (i.e., the client), as indicated by the asymmetry in the arrows representing data flow.

(typically) obtained from DFT calculations, either by tabulation or by fitting empirical expressions.

The charge fluctuations from the neutral reference are expressed using Mulliken (gross) charges,[56] and, depending on the Hamiltonian, the electrostatics are restricted to atomic monopoles or selected multipole contributions. The electrostatic potential is then evaluated at each atomic site, with the resulting 2-center contributions approximating the integrals as a product of the overlap between sites and the average of their potentials (for the monopole).

The exchange–correlation contributions are included in the parameterization of the reference neutral system, combined with taking a suitable atomic limit for the electrostatic energy of the charge fluctuations.[57,58] The double-counting terms in the energy expressions are represented as fitted inter-atomic potentials[1,52] or as parameterized inter-atomic integrals.[55]

B. The DFTB+ code

The DFTB+ code implements various DFTB and xTB models. Interactions between atoms are internally represented using a data structure based on spatial atomic neighbors,[59] and most terms are evaluated in real space; hence, the majority of the code is boundary-condition independent. Therefore, for periodic structures (and other space-filling geometries), the Hamiltonian and overlap matrices are transformed into a crystal-momentum ($\mathbf{k}$) dependent dense Hamiltonian. The resulting set of secular equations for the band structure is solved either via conventional diagonalization (LAPACK[60] or ScaLAPACK[61]), via hybrid central processing unit (CPU)-graphics processing unit (GPU) calculations (MAGMA[62,63] or ELPA[64]), or through one of the eigenvalue or density-matrix distributed solvers provided by the ELSI project.[36]

The DFTB+ codebase[65] is primarily written in FORTRAN 2008, with components in C/C++ and PYTHON 3. An API is provided in these languages to use the code as an external library for energy/force calculations or other modes such as real-time electronic propagation, e.g., Ehrenfest dynamics.[66] The software is licensed under the GNU Lesser General Public License 3.0 (or later),[67] chosen based on the code's library capability. Continuous integration of DFTB+ is performed via the project GitHub repository,[65] with custom regression testing scripts and a unit test system using the FyTest framework.[68] The code is internally documented with Doxygen[69] and FORD[70] compatible comments.

C. Extensions enabled by the present work

The external Hamiltonian evaluation via ACEhamiltonians.jl[18] (Sec. IV) is performed in real space and, hence, can be evaluated for the general range of boundary conditions supported by DFTB+. These include conventional molecular/cluster structures in free space or periodic boundary conditions; more general boundary conditions can also be evaluated by DFTB+, such as Green's function embedding[71] or helical structures.[72] The externally provided electronic structure model is built piece-wise from local geometric cluster fragments to give coverage of the entire geometry, which then includes the boundary conditions managed by DFTB+.

The ASI bindings (Sec. V) directly exchange the dense Hamiltonian matrices to be diagonalized and/or the dense single-particle density matrix between codes. The direct communication enables direct comparison of the semi-empirical Hamiltonians against local or non-local first principles models. The local potential exchange via ASI (Sec. V B 1) also enables the use of various forms of external electrostatic embedding models, along with testing the approximations in self-consistent semi-empirical Hamiltonians against the first principles of local potentials.

D. Computational details

To demonstrate the capabilities of the ASI interface, we have calculated the band structure of Al with the Hamiltonian ($\mathbf{H}$) and overlap ($\mathbf{S}$) matrices evaluated in the all-electron full-potential numerical atomic orbital software package FHI-aims (Version: 230905)[48] that implements ASI API version 1.1.[47] The ground state electron density of the Al bulk crystal with a lattice parameter of 2.024 Å was evaluated using DFT with the Perdew–Burke–Ernzerhof (PBE) exchange–correlation functional,[73] a scalar-relativistic zeroth order regular approximation (ZORA) correction,[48] and a $2 \times 2 \times 2$ Γ-centered $\mathbf{k}$-grid. A "minimal" basis set was used for Al, which consists of 13 numerical orbitals (3s, 2p, and 1d orbitals). $\mathbf{H}$ was subsequently evaluated along the $\mathbf{k}$-path W–X–Γ–L–K–Γ–L, with 50 sampling points in each section of the pathway. $\mathbf{H}$ and $\mathbf{S}$ were exported via the ASI callback function from FHI-aims to DFTB+. The DFTB+ computation was configured to use the same basis and path in $\mathbf{k}$-space, with the eigensolver of DFTB+ used to obtain the band structure.

In the case of the ACEhamiltonians interface, the same FHI-aims configurations were used. The $\mathbf{H}$ and $\mathbf{S}$ matrices were exported from FHI-aims as real-space matrices using ACEhamiltonians version v0.1.0.[18]

IV. ACEhamiltonians AS A LIBRARY TO PROVIDE EXTERNAL HAMILTONIAN EVALUATION FOR DFTB+

Within DFTB+, an interface was constructed to facilitate communication between the ACEhamiltonians and DFTB+ software packages. The interface enables data-driven ACEhamiltonans models for $\mathbf{H}$ and $\mathbf{S}$ to be combined with the robust functionality of the DFTB+ framework. The interface design provides threefold benefits: (i) modularity, by providing a means by which observables can be computed using ACEhamiltonians models without having to unnecessarily extend the ACEhamiltonians codebase itself; (ii) performance, by using an optimized production-level framework such as DFTB+, especially when dealing with larger systems where domain decomposition-based parallelism is essential; and (iii) accessibility, as such an interface reduces the barrier associated with using an ACEhamiltonians model by allowing combination with widely used software that has a broad userbase.

A. Interface description

Communication between the FORTRAN-based DFTB+ and ACEhamiltonians follows the general structure illustrated in Fig. 2, except that the JULIA-based ACEhamiltonians are facilitated via an intermediary C layer. This interfacial layer ensures that the modifications to DFTB+, which allow the invocation of external models, are not restricted to one external framework or programming language. The translation layer was written in low-level C, which

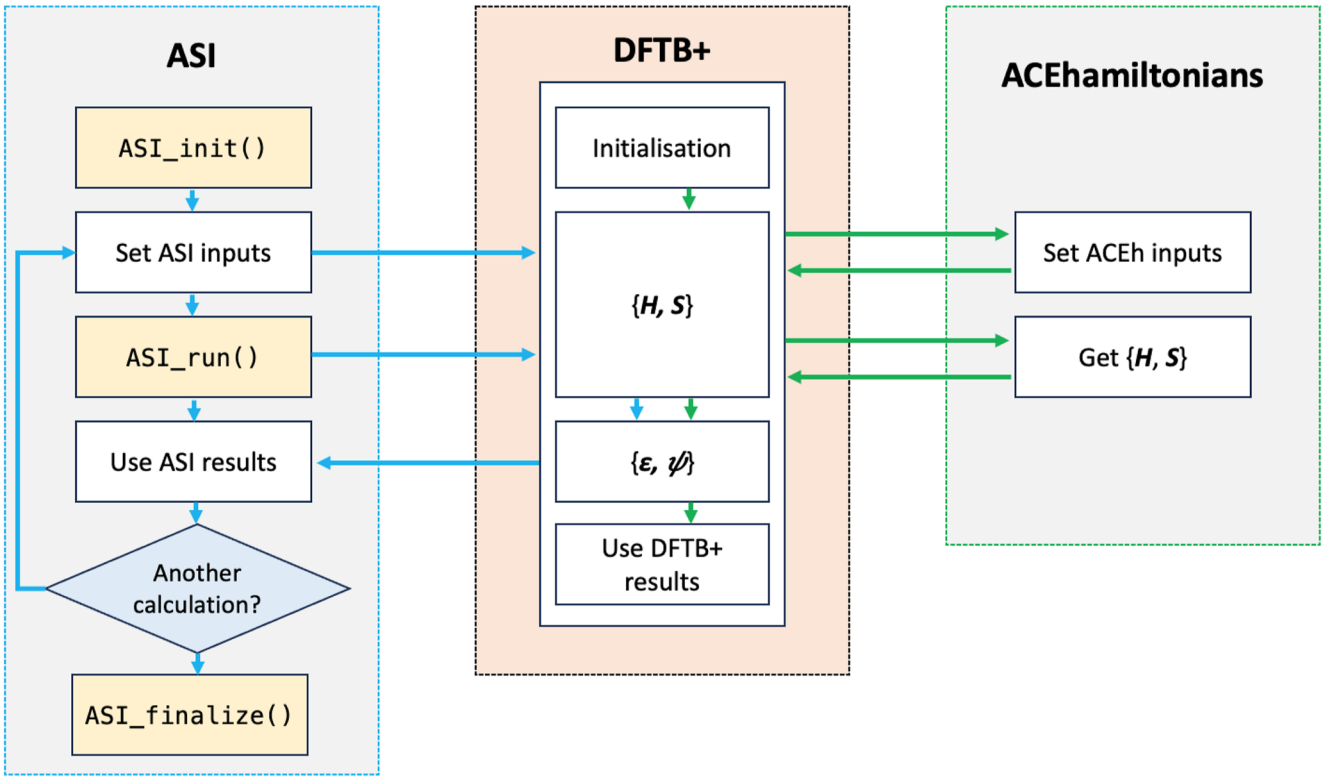

FIG. 2. Schematic representation of the specific workflows invoked between `DFTB+`, `ASI`, and `ACEhamiltonians`. Boxes and arrows in blue are part of the `ASI` execution pathway, and in green for the `ACEhamiltonians` pathway.

provides good interoperability with other languages through external bindings. When executed, `DFTB+` calls the `ACEhamiltonians` interface to invoke a setup subroutine, during which an initial bidirectional exchange of information occurs. The exchange allows `DFTB+` to specify the chemical species present in the target system; the `ACEhamiltonians` interface responds by providing the environmental and interaction cutoff distances, followed by the number of orbitals present for each species along with their occupancy and azimuthal quantum numbers.

`DFTB+` constructs all relevant atom and bond clusters with the information obtained. The clusters are provided to the interface, along with a list of indices specifying which block of the Hamiltonian/overlap matrix is represented by each cluster. The information is stored internally in the `ACEhamiltonians` interface until a new set of clusters is provided, such as would be expected during a molecular dynamics simulation, or the model cleanup subroutine is invoked, which clears memory in preparation for code termination.

Subsequently, `DFTB+` calls the prediction subroutine in the `ACEhamiltonians` interface, providing pointers to the Hamiltonian and overlap matrices that are to be populated. The interface loops over the atom clusters and populates the associated on-site Hamiltonian matrix block by block for each atom by evaluating the model. During each loop, the `ACEhamiltonians` function responsible for constructing on-site blocks is called; the coordinates and species of the atoms are provided, along with the model that is to be evaluated and the block of the Hamiltonian matrix into which the results should be placed; and the on-site blocks of the overlap matrix are set to an identity matrix. The process is repeated with the bond clusters to fill in the offsite blocks of the Hamiltonian and overlap matrices.

As shown in Fig. 3, the clusters needed to compute onsite blocks are spherical and atom-centered, while those for offsite blocks, which represent interactions between orbitals on two distinct atoms, are cylindrical and bond-centered. Atomic coordinates are provided relative to the origin atom i, with which atomic clusters are constructed for every atom in the structure. The atomic cluster for atom i can be defined as the subset of atoms that satisfy $r_{ij} \leq r_{cut}$, where r_{ij} is the distance between atom i and some other atom j, and the environmental cutoff distance r_{cut} is a free parameter. Bond clusters are created for all atom pairs $\{i,j\}$ for which atom i resides in the origin cell, and $r_{ij} \leq r_{bond}$ holds true, where r_{bond} specifies the interaction cutoff distance. For a given atom pair $\{i,j\}$, the bond cluster is the subset of atoms whose perpendicular distance to the open line segment between atoms i and j does not exceed the specified environmental cutoff distance r_{cut}. All coordinates are specified at the midpoint of the bond. Further details of the `DFTB+` external API are given in the supplementary material.

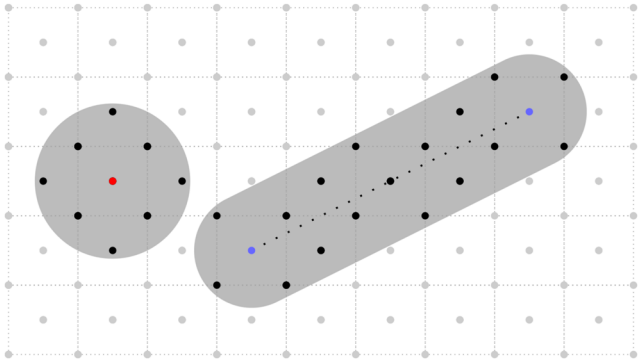

FIG. 3. Schematic representation of an atom-centered cluster (central atom shown in red) and a bond-centered cluster formed between a pair of atoms (shown in blue) in a periodic crystal lattice. Shaded regions indicate areas where an atom would be considered to be part of the cluster. Black and gray colored atoms are used to indicate those that are and are not part of a cluster, respectively.

B. Results

Figure 4 presents the band structure of a pristine aluminum FCC unit cell as obtained via the DFTB+ API (red dots), alongside the same calculation performed using the ACEhamiltonians package directly (black line). The results agree quantitatively in the occupied levels; however, local deviations become apparent within the higher energy unoccupied levels. Notably, the band structure exhibits a mean absolute eigenvalue error of $\sim 10^{-2}$ eV. This is greater than would naturally be expected given that the two results are generated using the same underlying model and share many of the same prediction subroutines. In an effort to determine the source of the observed deviation, the DFTB+ API was used to generate and, subsequently, write out the Hamiltonian and overlap matrices. These were then used by ACEhamiltonians to reconstruct the band structure. The resulting band structure was within machine precision of that derived using ACEhamiltonians directly, which demonstrates that the discrepancy originates from the means by which the band structures were calculated rather than the underlying matrices (i.e., the API is not the source of the difference). The discrepancy stems from the different eigensolvers used by the ACEhamiltonians and DFTB+ codebase: when using a common matrix source, the subroutines of the ACEhamiltonians package produce band structures that agree ($\sim 8 \times 10^{-6}$ eV) with those generated by FHI-aims, which alleviates the discrepancies in the higher energy levels.

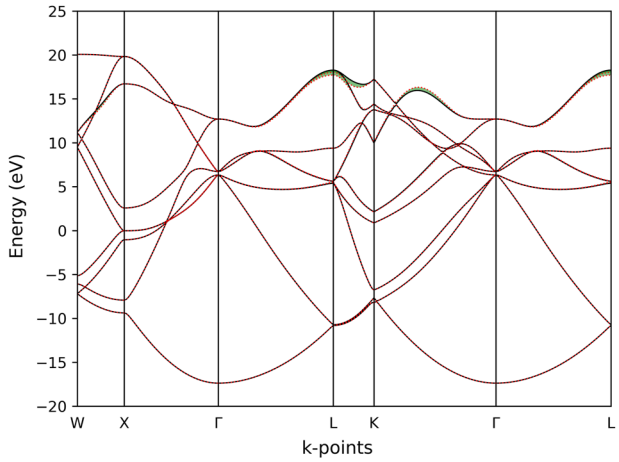

FIG. 4. Band structure of a pristine aluminum FCC unit cell as calculated directly by the ACEhamiltonians package (black) and obtained using the DFTB+ API with the same model (red dotted). Green shading is used to highlight areas of discrepancy in the unoccupied states.

V. ATOMIC SIMULATION INTERFACE (ASI) AS A DRIVER THAT USES DFTB+

The ability to drive calculations externally and use a specific package to evaluate system properties on demand motivates the development of an infrastructure where DFTB+ can be deployed as a software library. Modern PYTHON coding developments provide capacity for high-level interfaces reliant on file I/O for data transfer, but deep integration via pre-compiled software languages can enable more efficient and accurate data communication and software applications. Recent efforts toward this software paradigm have seen the development of the Atomic Simulation Interface (ASI), with the primary purpose of conveniently connecting ASI-enabled codes in multiscale simulation workflows, such as hybrid quantum/molecular mechanics (QM/MM), multiscale quantum mechanical embedding (QM/QM), or integration with machine learning (ML) frameworks (QM/ML).

A. Interface description

ASI has been developed as a plain C API, again demonstrating the use of a low-level language enabling compatibility across software infrastructure. The key feature of the ASI is the provision of an efficient and portable method to transmit large data arrays, relevant to electronic structure models, between software packages. ASI itself is fundamentally an API specification, similar to MPI or BLAS standards, that ensures compatibility; the complete ASI API specification is available as a C header file with comments in Doxygen[69] format, along with HTML pages generated by Doxygen[69] from the aforementioned C header file. The ASI API is designed to be implemented by software packages to provide programmatic access to their internal data structures. We refer to software that implements ASI APIs as *ASI-enabled codes*, and we refer to software that invokes ASI API functions as ASI *clients*.

In the current example, DFTB+ is an ASI-enabled code with functionality provided for the communication of key electronic data structures, such as Hamiltonian ($\mathbf{H}$) and overlap ($\mathbf{S}$) matrices, as well as less complex data objects, e.g., variables and arrays, such as energies (E) and forces on atomic centers ($-\nabla E$). The DFTB+ ASI is implemented as a separate C library that links with the DFTB+ library and ASI clients. A PYTHON wrapper for the ASI API, asi4py, provides compatibility with PYTHON workflows and is available for installation via the pip command line tool. The convenience of asi4py complements the deep integration of the ASI interface and provides a user-friendly way to create ASI clients in Python.

The key ASI functions can be broken into four groups: control flow, atomic information, electrostatic potential, and electronic structure matrices.[47] The necessary intrusions to implement in an existing codebase are minimal. For the application of DFTB+ using the ASI standard, the DFTB+ package is compiled as a shared object library to allow dynamic linkage with the client. The workflow is driven by the ASI client; thus, after ASI initialization, key data objects are communicated to/from DFTB+ and callback functions are registered before the request for execution of a DFTB+ calculation. Callback functions give direct access to internal data objects within the DFTB+ process via pointers without unnecessary copying, thus causing near-zero computational and communication overhead and also adapting to the chosen parallelization scheme. The callback functions are invoked during the execution process, providing external access to data objects when they are calculated. Derived quantities, such as energy, forces, stress, and atomic charges, are also available. Once all necessary operations on the exposed data objects have been completed, finalization is performed, which includes the release of allocated memory. The ASI workflow is presented in Fig. 2 and contrasted against the ACEhamiltonians interface. Further details of the key ASI functions are provided in the supplementary material.

B. Results

1. Electrostatic embedding

The ASI functions that allow communication of the electrostatic potential can facilitate electrostatic QM/QM embedding. Figure 5 compares the total intermolecular interaction energy of two water molecules evaluated with DFTB+ and the electrostatic component of that interaction as evaluated with a PYTHON script that orchestrates two DFTB+ instances using the asi4py library interface. In the latter case, each DFTB+ instance calculates a single water molecule, and the electrostatic potential from one molecule is then exported from one DFTB+ instance using the ASI_calc_esp function and transferred via MPI calls to the second DFTB+ instance, where it is included via the callback installed by the ASI_register_external_potential function.

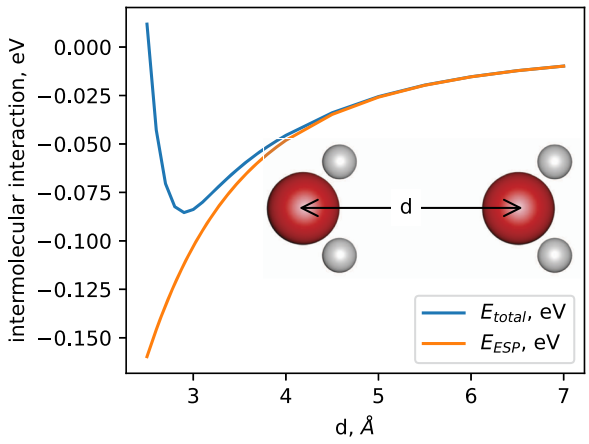

FIG. 5. Example of electrostatic embedding in DFTB+ achieved with the ASI interface. The distance (d) between two water dimers is shown in the inset. The graph shows the interaction energy as a function of d (blue line) and the electrostatic embedding energy evaluated with the ASI API in a self-consistent manner (red line). The lines are shown to converge at large d.

The calculation is performed self-consistently: the energy of both molecules is initially calculated with zero external potential; then the calculation for each molecule is repeated using the electrostatic potential provided by the other molecule. The calculation should be repeated until self-consistency is achieved. Convergence criteria should be defined and checked by ASI clients; for a simple system with two water molecules simulated by separate DFTB+ instances, five iterations are sufficient to reach 10^{-5} eV accuracy on the distances from 2.5 Å and above (see Fig. 5). Figure 5 shows that the electrostatic potential is dominant for the intermolecular interaction at large distances (>4 Å), which is the expected behavior.

2. Electronic structure transfer

The ASI implementation in DFTB+ supports the import of Hamiltonian ($\mathbf{H}$) and overlap ($\mathbf{S}$) matrices. With this functionality, data objects evaluated in electronic structure software packages can be imported into DFTB+ and evaluated. The potential of this functionality is demonstrated with the computation of the electronic band structure for bulk Al (supplementary material, Fig. S1), where $\mathbf{H}$ and $\mathbf{S}$ have been computed with the software package FHI-aims. FHI-aims supports the ASI API, and with the ASI data transfer protocols, it is possible to calculate and analyze the band structure in DFTB+. The resulting band structure is given in the supplementary material; the result overlays the band structure achieved with the standalone ACEhamiltonians approach and matches the FHI-aims native calculation of the same data, showing the versatility of this modular interface.

VI. CONCLUSIONS

As workflows in computational materials science become more complex, codes need to become more interoperable. Potential paradigms when interfacing electronic structure software with other codes are: the software can act as the *driver*, requesting information; or as the *library*, being queried for information. In both cases, data transfer is bidirectional, although asymmetric. With the emergence of ML workflows, there are many opportunities to achieve synergy between semi-empirical electronic structure methods and data-driven approaches.[9] To yield usable software solutions that enable complex simulations or data-driven workflows, robust interfaces between different codes must be established.

Here, we have reported examples of electronic structure interfaces implemented in the DFTB+ code that explore the driver and library paradigms. We explain the general considerations and traits of the interfaces and showcase possible use cases by communicating electronic structure information in the form of the Hamiltonian in local basis representation and evaluating the embedded electrostatic potential.

Both interfaces have the potential to provide exciting future capabilities. The ASI bindings can, in principle, be used for a self-consistent workflow, either driven inside DFTB+ or externally. Similarly, the ACEhamiltonians framework could exchange atomic properties such as charge, enabling self-consistent updates of the

supplied model. Either option would then also immediately be compatible with a subset of the DFTB+ capabilities beyond ground state calculations, such as Δ-SCF excitations.[52] Similarly, calculations using a density-functional ground state reference, which then uses the DFTB approximated random-phase excitation poles, become possible.[74] Generalization to spin-polarization or extending the real-time electronic propagation to receive an external model are also interesting further applications. Another extension built on top of the current work would be to exchange derivatives of the external models with respect to atomic displacements, enabling forces/strains from the Hellmann–Feynman theorem, or higher-order response properties using the internal DFTB+ coupled perturbed routines.[75]

In summary, the presented outcomes demonstrate the potential for flexible and powerful usage of components of the DFTB+ package by harnessing modularity. There is ample space for further integration of data workflows. The modularity of the package integration presents insertion points that can be used for evaluating a range of data objects in a variety of software packages, using the best implementations of any given step when these may be in separate software.

SUPPLEMENTARY MATERIAL

The accompanying supplementary material provides the following: a comparison of band structures calculated with FHI-aims and DFTB+, details of the DFTB+ external API, and details of the ASI API.

ACKNOWLEDGMENTS

This work was financially supported by the Leverhulme Trust Research Project (Grant No. RPG-2017-191) and the NOMAD Center of Excellence (European Commission Grant Agreement No. 951786). A.L. and P.S. acknowledge funding from the UKRI Future Leaders Fellowship program (Grant Nos. MR/T018372/1 and MR/Y034279/1). R.J.M. acknowledges funding from the UKRI Future Leaders Fellowship program (Grant Nos. MR/S016023/1 and MR/X023109/1) and UKRI Frontier Research (Grant No. EP/X014088/1). P.S., R.J.M., and A.L. acknowledge funding from the ARCHER2 eCSE Program (Grant No. eCSE03-10). We acknowledge computational resources provided by the Scientific Computing Research Technology Platform of the University of Warwick, Supercomputing Wales, for access to the Hawk HPC facility, which is part-funded by the European Regional Development Fund via the Welsh Government, the EPSRC-funded HPC Midlands+ consortium (Grant No. EP/T022108/1), the UK Car-Parinello consortium (Grant No. EP/P022065/1), and the UK High Performance Computing "Materials Chemistry Consortium" (Grant Nos. EP/R029431/1 and EP/X035859/1) for access to the ARCHER2 high-performance computing facility.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

P.S. and A.M. contributed equally to this work.

Pavel Stishenko: Methodology (equal); Software (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Adam McSloy**: Software (equal); Visualization (equal); Writing – review & editing (equal). **Berk Onat**: Software (equal). **Ben Hourahine**: Conceptualization (equal); Funding acquisition (equal); Methodology (equal); Project administration (equal); Supervision (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Reinhard J. Maurer**: Conceptualization (equal); Funding acquisition (equal); Project administration (equal); Supervision (equal); Writing – original draft (equal); Writing – review & editing (equal). **James R. Kermode**: Conceptualization (equal); Funding acquisition (equal); Methodology (equal); Project administration (equal); Supervision (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Andrew Logsdail**: Conceptualization (equal); Funding acquisition (equal); Project administration (equal); Supervision (equal); Writing – original draft (equal); Writing – review & editing (equal).

DATA AVAILABILITY

The DFTB+ software package is available at https://github.com/dftbplus/dftbplus. The v24.2 release will contain all functionality outlined in this manuscript; the described changes for the ASI binding or to connect to the ACEhamiltonians are undergoing review and are currently available at Refs. 76 and 77 respectively. Full documentation is available at https://dftbplus.org/. The ACEhamiltonians v0.1.0 software package is available at https://github.com/ACEsuit/ACEhamiltonians.jl. The ASI v1.1 software package and DFTB+ implementation are available at https://gitlab.com/pvst/asi. The interface specification is available at https://pvst.gitlab.io/asi.

REFERENCES

[1] M. Elstner and G. Seifert, Philos. Trans. R. Soc., A **372**, 20120483 (2014).
[2] C. Bannwarth, B. Hourahine, and J. Moussa, "IOP roadmap: Software for electronic structure based simulations in chemistry and materials," in *Electronic Structure*, edited by T. Windus and V. Blum (Institute of Physics, 2024).
[3] Y. Nishimura and H. Nakai, Chem. Lett. **50**, 1546 (2021).
[4] M. Gaus, A. Goez, and M. Elstner, J. Chem. Theory Comput. **9**, 338 (2013).
[5] M. Mortazavi, J. G. Brandenburg, R. J. Maurer, and A. Tkatchenko, J. Phys. Chem. Lett. **9**, 399 (2018).
[6] G. Jha and T. Heine, J. Chem. Theory Comput. **18**, 4472 (2022).
[7] J. A. Keith, V. Vassilev-Galindo, B. Cheng, S. Chmiela, M. Gastegger, K.-R. Müller, and A. Tkatchenko, Chem. Rev. **121**, 9816 (2021).
[8] J. Westermayr, M. Gastegger, K. T. Schütt, and R. J. Maurer, J. Chem. Phys. **154**, 230903 (2021).
[9] N. Fedik, B. Nebgen, N. Lubbers, K. Barros, M. Kulichenko, Y. W. Li, R. Zubatyuk, R. Messerly, O. Isayev, and S. Tretiak, J. Chem. Phys. **159**, 110901 (2023).
[10] J. M. Knaup, B. Hourahine, and T. Frauenheim, J. Phys. Chem. A **111**, 5637 (2007).

[11]N. F. Aguirre, A. Morgenstern, M. J. Cawkwell, E. R. Batista, and P. Yang, J. Chem. Theory Comput. **16**, 1469 (2020).
[12]A. S. Hutama, C.-p. Chou, Y. Nishimura, H. A. Witek, and S. Irle, J. Phys. Chem. A **125**, 2184 (2021).
[13]M. Stöhr, L. Medrano Sandonas, and A. Tkatchenko, J. Phys. Chem. Lett. **11**, 6835 (2020).
[14]K. T. Schütt, M. Gastegger, A. Tkatchenko, K.-R. Müller, and R. J. Maurer, Nat. Commun. **10**, 5024 (2019).
[15]M. Gastegger, A. McSloy, M. Luya, K. T. Schütt, and R. J. Maurer, J. Chem. Phys. **153**, 044123 (2020).
[16]O. Unke, M. Bogojeski, M. Gastegger, M. Geiger, T. Smidt, and K.-R. Müller, in *Advances in Neural Information Processing Systems* (Curran Associates, Inc., 2021), Vol. 34, pp. 14434–14447.
[17]J. Nigam, M. J. Willatt, and M. Ceriotti, J. Chem. Phys. **156**, 014115 (2022).
[18]L. Zhang, B. Onat, G. Dusson, A. McSloy, G. Anand, R. J. Maurer, C. Ortner, and J. R. Kermode, npj Comput. Mater. **8**, 158 (2022).
[19]H. Li, Z. Wang, N. Zou, M. Ye, R. Xu, X. Gong, W. Duan, and Y. Xu, Nat. Comput. Sci. **2**, 367 (2022).
[20]P. O. Dral, O. A. Von Lilienfeld, and W. Thiel, J. Chem. Theory Comput. **11**, 2120 (2015).
[21]G. Zhou, N. Lubbers, K. Barros, S. Tretiak, and B. Nebgen, Proc. Natl. Acad. Sci. U. S. A. **119**, e2120333119 (2022).
[22]A. McSloy, G. Fan, W. Sun, C. Hölzer, M. Friede, S. Ehlert, N.-E. Schütte, S. Grimme, T. Frauenheim, and B. Aradi, J. Chem. Phys. **158**, 034801 (2023).
[23]J. J. Mortensen, L. B. Hansen, and K. W. Jacobsen, Phys. Rev. B **71**, 035109 (2005).
[24]D. G. Smith, L. A. Burns, A. C. Simmonett, R. M. Parrish, M. C. Schieber, R. Galvelis, P. Kraus, H. Kruse, R. Di Remigio, A. Alenaizan *et al.*, J. Chem. Phys. **152**, 184108 (2020).
[25]Q. Sun, X. Zhang, S. Banerjee, P. Bao, M. Barbry, N. S. Blunt, N. A. Bogdanov, G. H. Booth, J. Chen, Z.-H. Cui *et al.*, J. Chem. Phys. **153**, 024109 (2020).
[26]M. McIlroy, E. Pinson, and B. Tague, Bell Syst. Tech. J. **57**, 1899 (1978).
[27]P. Blaha, K. Schwarz, F. Tran, R. Laskowski, G. K. H. Madsen, and L. D. Marks, J. Chem. Phys. **152**, 074101 (2020).
[28]J. R. Kermode, J. Phys.: Condens. Matter **32**, 305901 (2020).
[29]X. Shao, O. Andreussi, D. Ceresoli, M. Truscott, A. Baczewski, Q. Campbell, and M. Pavanello (2024). "QEpy: Quantum ESPRESSO in Python," GitLab. https://gitlab.com/shaoxc/qepy
[30]G. Csányi, S. Winfield, J. R. Kermode, A. De Vita, A. Comisso, N. Bernstein, and M. C. Payne, IOP Comput. Phys. Newsl. 2007, 1–24.
[31]S. J. Clark, M. D. Segall, C. J. Pickard, P. J. Hasnip, M. I. J. Probert, K. Refson, and M. C. Payne, Z. Kristallogr. - Cryst. Mater. **220**, 567 (2005).
[32]J. R. Kermode, Fork of Bader charge analysis code with Python interface.
[33]V. Kapil, M. Rossi, O. Marsalek, R. Petraglia, Y. Litman, T. Spura, B. Cheng, A. Cuzzocrea, R. H. Meißner, D. M. Wilkins, B. A. Helfrecht, P. Juda, S. P. Bienvenue, W. Fang, J. Kessler, I. Poltavsky, S. Vandenbrande, J. Wieme, C. Corminboeuf, T. D. Kühne, D. E. Manolopoulos, T. E. Markland, J. O. Richardson, A. Tkatchenko, G. A. Tribello, V. Van Speybroeck, and M. Ceriotti, Comput. Phys. Commun. **236**, 214 (2019).
[34]T. Barnes (2022). "The MolSSI driver interface library," GitHub. https://molssi-mdi.github.io/MDI_Library
[35]P. Pracht, S. Grimme, C. Bannwarth, F. Bohle, S. Ehlert, G. Feldmann, J. Gorges, M. Müller, T. Neudecker, C. Plett, S. Spicher, P. Steinbach, P. A. Wesołowski, and F. Zeller, J. Chem. Phys. **160**, 114110 (2024).
[36]V. W. Yu, C. Campos, W. Dawson, A. García, V. Havu, B. Hourahine, W. P. Huhn, M. Jacquelin, W. Jia, M. Keçeli, R. Laasner, Y. Li, L. Lin, J. Lu, J. Moussa, J. E. Roman, Á. Vázquez-Mayagoitia, C. Yang, and V. Blum, Comput. Phys. Commun. **256**, 107459 (2020).
[37]A. Hjorth Larsen, J. Jørgen Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dułak, J. Friis, M. N. Groves, B. Hammer, C. Hargus *et al.*, J. Phys.: Condens. Matter **29**, 273002 (2017).
[38]G. Pizzi, A. Cepellotti, R. Sabatini, N. Marzari, and B. Kozinsky, Comput. Mater. Sci. **111**, 218 (2016).
[39]J. P. Unsleber, S. A. Grimmel, and M. Reiher, J. Chem. Theory Comput. **18**, 5393 (2022).
[40]S. Grimme, F. Bohle, A. Hansen, P. Pracht, S. Spicher, and M. Stahn, J. Phys. Chem. A **125**, 4039 (2021).
[41]C. B. Hicks and T. J. Martinez, J. Chem. Phys. **160**, 142501 (2024).
[42]M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne *et al.*, Sci. Data **3**, 160018 (2016).
[43]G. Henkelman, A. Arnaldsson, and H. Jónsson, Comput. Mater. Sci. **36**, 354 (2006).
[44]T. A. Manz and N. G. Limas, RSC Adv. **6**, 47771 (2016).
[45]T. A. Manz and N. G. Limas, Chargemol program for performing DDEC analysis, version 3.4.
[46]A. A. Mostofi, J. R. Yates, G. Pizzi, Y.-S. Lee, I. Souza, D. Vanderbilt, and N. Marzari, Comput. Phys. Commun. **185**, 2309 (2014).
[47]P. V. Stishenko, T. W. Keal, S. M. Woodley, V. Blum, B. Hourahine, R. J. Maurer, and A. J. Logsdail, J. Open Source Software **8**, 5186 (2023).
[48]V. Blum, R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, K. Reuter, and M. Scheffler, Comput. Phys. Commun. **180**, 2175 (2009).
[49]F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, in *Proceedings of the 25th International Conference on World Wide Web* (International World Wide Web Conferences Steering Committee, 2016), pp. 263–273.
[50]F. Neese, F. Wennmohs, U. Becker, and C. Riplinger, J. Chem. Phys. **152**, 224108 (2020).
[51]E. 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, J. Chem. Phys. **158**, 174801 (2023).
[52]B. Hourahine, B. Aradi, V. Blum, F. Bonafé, A. Buccheri, C. Camacho, C. Cevallos, M. Y. Deshaye, T. Dumitrică, A. Dominguez, S. Ehlert, M. Elstner, T. van der Heide, J. Hermann, S. Irle, J. J. Kranz, C. Köhler, T. Kowalczyk, T. Kubař, I. S. Lee, V. Lutsker, R. J. Maurer, S. K. Min, I. Mitchell, C. Negre, T. A. Niehaus, A. M. N. Niklasson, A. J. Page, A. Pecchia, G. Penazzi, M. P. Persson, J. Řezáč, C. G. Sánchez, M. Sternberg, M. Stöhr, F. Stuckenberg, A. Tkatchenko, V. W.-z. Yu, and T. Frauenheim, J. Chem. Phys. **152**, 124101 (2020).
[53]B. Hourahine, B. Aradi, V. Blum, F. Bonafé, A. Buccheri, C. Camacho, C. Cevallos, M. Y. Deshaye, T. Dumitrică, A. Dominguez, S. Ehlert, M. Elstner, T. van der Heide, J. Hermann, S. Irle, J. Jakowski, J. J. Kranz, C. Köhler, T. Kowalczyk, T. Kubař, I. S. Lee, V. Lutsker, R. J. Maurer, S. K. Min, I. Mitchell, C. Negre, T. A. Niehaus, A. M. N. Niklasson, A. J. Page, A. Pecchia, G. Penazzi, M. P. Persson, J. Řezáč, C. G. Sánchez, M. Sternberg, M. Stöhr, F. Stuckenberg, A. Tkatchenko, V. W.-z. Yu, and T. Frauenheim, J. Chem. Phys. **157**, 039901 (2022).
[54]L. S. Blackford, A. Petitet, R. Pozo, K. Remington, R. C. Whaley, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry *et al.*, ACM Trans. Math. Software **28**, 135 (2002).
[55]C. Bannwarth, E. Caldeweyher, S. Ehlert, A. Hansen, P. Pracht, J. Seibert, S. Spicher, and S. Grimme, Wiley Interdiscip. Rev.: Comput. Mol. Sci. **11**, e01493 (2020).
[56]R. S. Mulliken, J. Chem. Phys. **23**, 1833 (1955).
[57]M. Elstner, D. Porezag, G. Jungnickel, J. Elsner, M. Haugk, T. Frauenheim, S. Suhai, and G. Seifert, Phys. Rev. B **58**, 7260 (1998).
[58]C. Bannwarth, S. Ehlert, and S. Grimme, J. Chem. Theory Comput. **15**, 1652 (2019).
[59]B. Aradi, B. Hourahine, and T. Frauenheim, J. Phys. Chem. A **111**, 5678 (2007).
[60]E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. (Society for Industrial and Applied Mathematics, Philadelphia, PA, 1999).
[61]L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide* (Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997).

[62]J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki, *Numerical Computations with GPUs* (Springer, 2014), Vol. 1.
[63]V.-Q. Vuong, C. Cevallos, B. Hourahine, B. Aradi, J. Jakowski, S. Irle, and C. Camacho, J. Chem. Phys. **158**, 084802 (2023).
[64]V. W. Yu, J. Moussa, P. Kůs, A. Marek, P. Messmer, M. Yoon, H. Lederer, and V. Blum, Comput. Phys. Commun. **262**, 107808 (2021).
[65]B. Hourahine, B. Aradi, V. Blum, F. Bonafé, A. Buccheri, C. Camacho, C. Cevallos, M. Y. Deshaye, T. Dumitrică, A. Dominguez, S. Ehlert, M. Elstner, T. van der Heide, J. Hermann, S. Irle, J. J. Kranz, C. Köhler, T. Kowalczyk, T. Kubař, I. S. Lee, V. Lutsker, R. J. Maurer, S. K. Min, I. Mitchell, C. Negre, T. A. Niehaus, A. M. N. Niklasson, A. J. Page, A. Pecchia, G. Penazzi, M. P. Persson, J. Řezáč, C. G. Sánchez, M. Sternberg, M. Stöhr, F. Stuckenberg, A. Tkatchenko, V. W.-z. Yu, and T. Frauenheim (2023). "DFTB+, a software package for efficient approximate density functional theory based atomistic simulations," Zenodo, V.23.1, https://doi.org/10.5281/zenodo.8117766
[66]F. P. Bonafé, B. Aradi, B. Hourahine, C. R. Medrano, F. J. Hernández, T. Frauenheim, and C. G. Sánchez, J. Chem. Theory Comput. **16**, 4454 (2020).
[67]Free Software Foundation, GNU Lesser General Public License, version 3, 2007.
[68]B. Aradi (2021). "FyTest–instant Fortran unit testing," GitHub. https://github.com/aradi/fytest
[69]D. van Heesch, Doxygen, https://www.doxygen.nl/, 2024.
[70]C. MacMackin (2023). "FORtran documenter," GitHub. https://github.com/Fortran-FOSS-Programmers/ford
[71]A. Pecchia and A. D. Carlo, Rep. Prog. Phys. **67**, 1497 (2004).
[72]I. Nikiforov, B. Hourahine, B. Aradi, T. Frauenheim, and T. Dumitrică, J. Chem. Phys. **139**, 094110 (2013).
[73]J. P. Perdew, K. Burke, and M. Ernzerhof, Phys. Rev. Lett. **77**, 3865 (1996).
[74]R. Rüger, E. van Lenthe, T. Heine, and L. Visscher, J. Chem. Phys. **144**, 184103 (2016).
[75]D. Maag, J. Böser, H. A. Witek, B. Hourahine, M. Elstner, and T. Kubař, J. Chem. Phys. **158**, 124107 (2023).
[76]P. Stishenko and B. Hourahine (2024). "Pull request no. 1335: 'Asirebase,'" GitHub. https://github.com/dftbplus/dftbplus/pull/1335
[77]B. Hourahine (2024). "Pull request no. 1420: 'External model interface,'" GitHub. https://github.com/dftbplus/dftbplus/pull/1420