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

Pavel Stishenko* and Andrew Logsdail†
Cardiff Catalysis Institute, School of Chemistry, Cardiff University,
Park Place, Cardiff CF10 3AT, United Kingdom

Adam McSloy,* Berk Onat, and James R. Kermode‡
Warwick Centre for Predictive Modelling, School of Engineering,
University of Warwick, Coventry, CV4 7AL, United Kingdom

Ben Hourahine§
SUPA, Department of Physics, John Anderson Building, 107 Rottenrow,
University of Strathclyde, Glasgow G4 0NG, United Kingdom

Reinhard J. Maurer
Department of Chemistry, University of Warwick, Coventry, CV4 7AL, United Kingdom and
Department of Physics, University of Warwick, Coventry, CV4 7AL, United Kingdom

(Dated: March 26, 2024)

Modern software engineering of electronic structure codes has seen a paradigm shift from monolithic workflows towards 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; and 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 to 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.

* These authors contributed equally to this work.
† LogsdailA@cardiff.ac.uk
‡ j.r.kermode@warwick.ac.uk
§ benjamin.hourahine@strath.ac.uk

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 $\sim 10^{18}$ 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 parametrizations; 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 providing 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), though 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] and socket-type interfaces.[29, 30] High-level Pythonic packages, such as the Atomic Simulation Environment (ASE)[31], have emerged as a *de facto* standard for building atomistic simulation workflows,[32], enabling 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. Recently, the emergence of automated and machine-learning-augmented workflows, and establishment of extensive materials databases using FAIR principles,[33] has 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 allows 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 Atomic Simulation Interface (ASI)[34] 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.[35]

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

Data parsing via file I/O operations: typically focused 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.[29, 30] 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.

Recent work on the `DFTB+` software package[36, 37] 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 to electronic structure codes before presenting two interfaces that are capable of either being *driven by an external workflow* to provide data,[34] 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

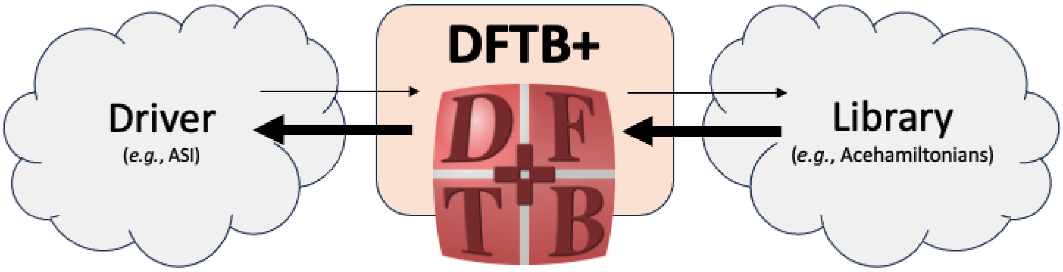

FIG. 1. Schematic representation of interfacing paradigms where `DFTB+` is used: as a resource, driven by an external package (left-hand side); or instead drives communication with an external package as part of its own workflow (right-hand side). In both cases, the majority of data communication is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

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 were 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, 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,[38] 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 is 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 have been 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 on to two different code interfacing paradigms (see Figure 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 the following, we present realizations of these paradigms inside the DFTB+ software package.

III. METHODS

A. The DFTB family of models

The DFTB method,[1] and related semi-empirical tight-binding models,[39] 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 from reference neutral atoms and atomic dimers. The 2-centre integrals are for a minimal, non-orthogonal, atomic valence basis (neglecting crystal field and 4-centre contributions[1]). Depending on the choice of Hamiltonian (DFTB or xTB) these values are (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[40] 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-centre 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.[41, 42] The double-counting terms in the energy expressions are represented as fitted inter-atomic potentials[1, 36] or as parameterized inter-atomic integrals.[39]

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[43] 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[44] or ScaLAPACK[45]), *via* hybrid CPU-GPU calculations (MAGMA[46, 47] or ELPA[48]) or through one of the eigenvalue or density-matrix distributed solvers provided by the ELSI project.[49]

The DFTB+ codebase[50] is primarily written in Fortran 2008, with components in C/C++ and Python3. 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,[51]). The software is licensed under the GNU Lesser General Public License 3.0 (or later),[52] chosen based on the code's library capability. Continuous integration of DFTB+ is performed via the project GitHub repository,[50] with custom regression testing scripts, plus a unit test system using the FyTest framework.[53] The code is internally documented with Doxygen[54] and FORD[55] compatible comments.

C. Extensions enabled by the present work

The external Hamiltonian evaluation via ACEhamiltonians.jl[18] (Section IV) is performed in real space, 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[56] or helical structures,[57]. The externally provided electronic structure model is built piece-wise from local geometric cluster fragments to give coverage of the entire geometry, that then includes the boundary conditions managed by DFTB+.

The ASI bindings (Section 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

(Section V B 1) also enables the use of various forms of external electrostatic embedding models, along with testing of the approximations in self-consistent semi-empirical Hamiltonians against the first principles local potentials.

D. Computational Details

To demonstrate the capabilities of the ASI interface, we have calculated the band structure of Al with the Hamiltonian (**H**) and overlap (**S**) matrices evaluated in the all-electron full-potential numerical atomic orbital software package FHI-aims (Version: 230905) [35] that implements ASI API version 1.1.[34] The ground state electron density of the Al bulk crystal with a lattice parameter of 2.024 Åwas evaluated using DFT with the PBE exchange-correlation functional,[58] a scalar-relativistic zeroth order regular approximation (ZORA) correction,[35] and a $2\times2\times2$ Γ-centered **k**-grid. A "minimal" basis set was used for Al, which consists of 13 numerical orbitals (3s, 2p, and 1d orbitals). **H** was subsequently evaluated along the **k**-path W-X-Γ-L-K-Γ-L, with 50 sampling points in each section of the pathway. **H** and **S** were exported via the ASI callback functions from FHI-aims to DFTB+. The DFTB+ computation was configured to use the same basis and path in **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 **H** and **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 **H** and **S** to be combined with the robust functionality of the DFTB+ framework. The interface design provides threefold benefit: (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; (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 Figure 2, except that the Julia-based ACEhamiltonians is facilitated via an intermediary C layer. This interfacial layer ensures that the modifications to DFTB+, which allow invocation of external models, are not restricted to one external framework or programming language. The translation layer was written in low-level C, which 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 are 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; 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 Figure 3, the clusters needed to compute onsite blocks are spherical and atom-centered, while those for offsite blocks, which represent interactions between orbitals on distinct two atoms, are cylindrical and bond-centered. Atomic coordinates are provided relative to the origin atom i, with which atomic clusters are constructed for every

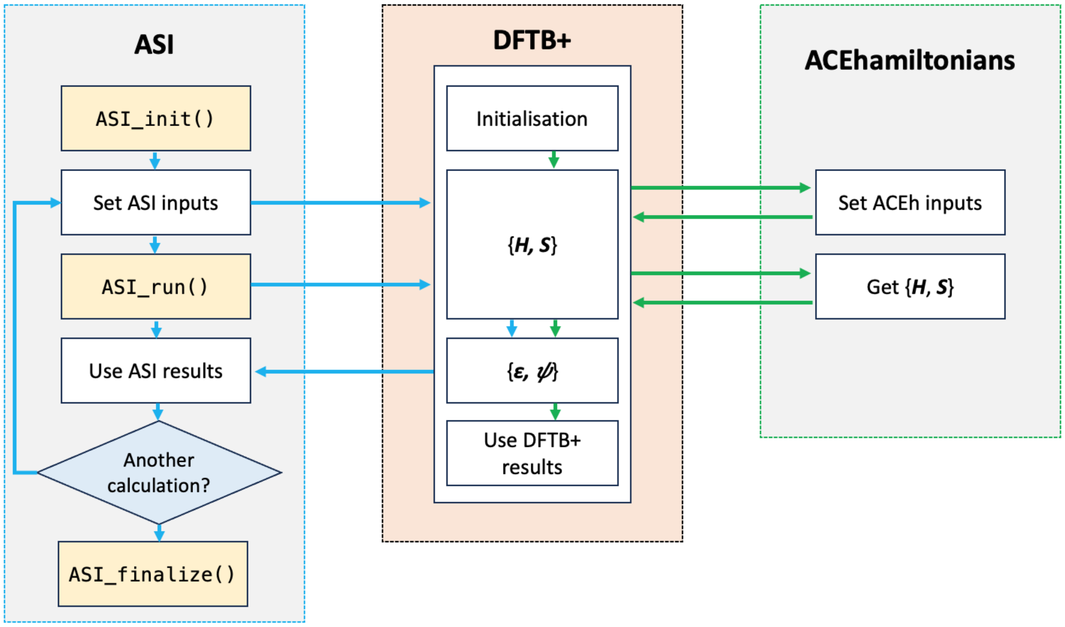

FIG. 2. Schematic representation of the specific workflows invoked between `DFTB+`, `ASI`, and `ACEhamiltonians`. Boxes and arrows in blue are part of the `ASI` execution pathway, and in green for the `ACEhamiltonians` pathway

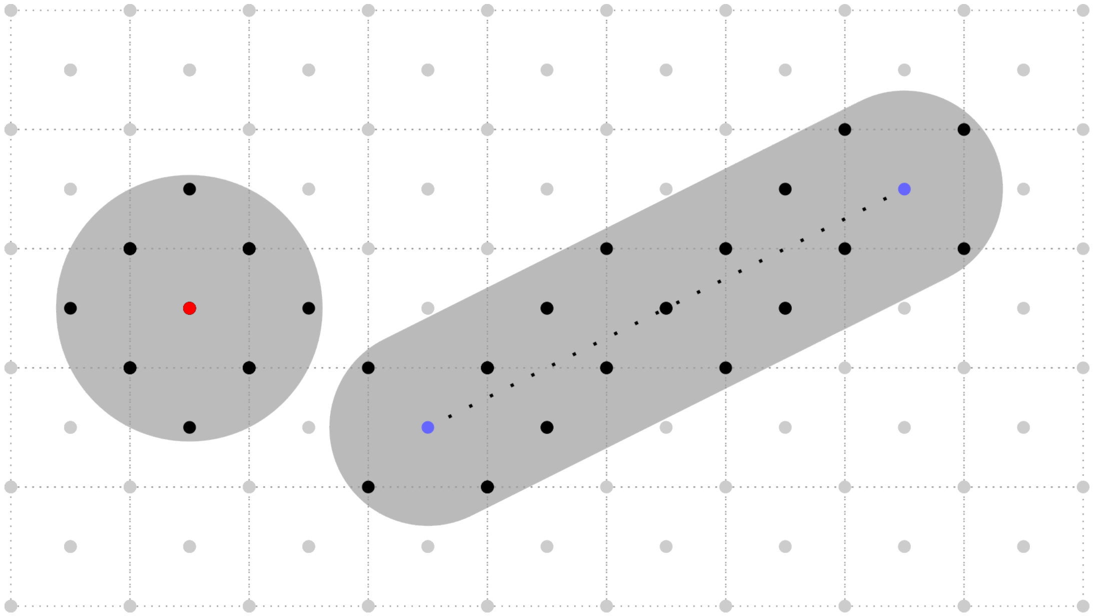

FIG. 3. Schematic representation of an atom-centred cluster (central atom shown in red), and a bond-centred cluster formed between a pair of atoms (shown in blue), in a periodic crystal lattice. Shaded regions indicate areas where an atom would be considered to be part of the cluster. Black and gray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

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 to the midpoint of the bond. Further details of the `DFTB+` external API are given in the Supporting Information (SI).

B. Results

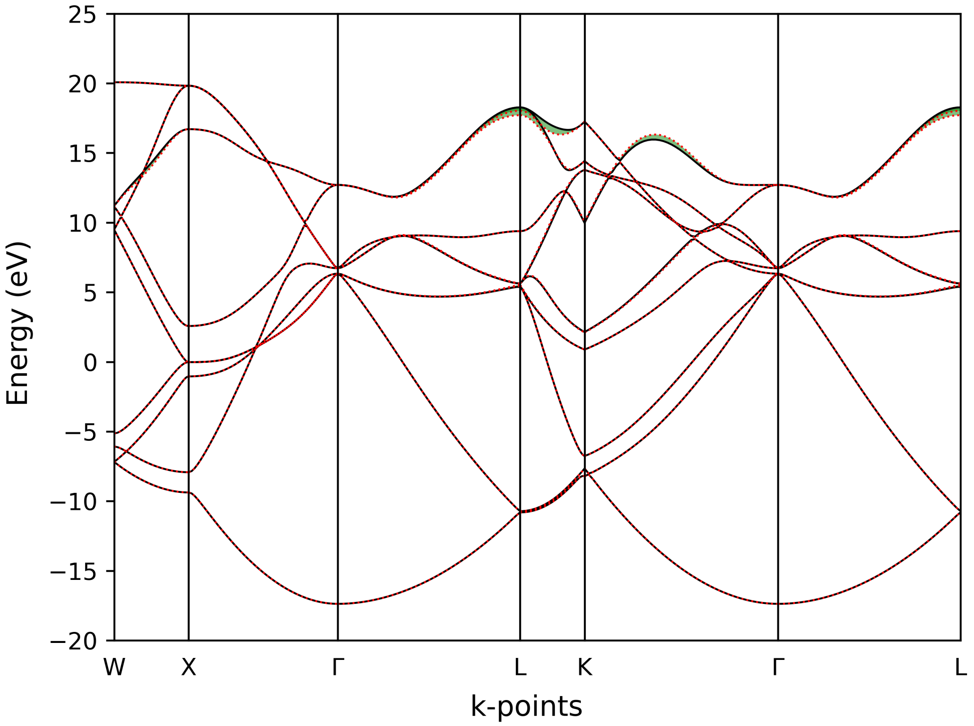

FIG. 4. Band structure of a pristine aluminum FCC unit cell as calculated directly by the `ACEhamiltonians` package (black), and obtained using the `DFTB+` API with the same model (red dotted). Green shading is used to highlight areas of discrepancy in the unoccupied states.

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 approximately 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`, and alleviates the discrepancies in the higher energy levels.

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 application. Recent efforts towards 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[54] format, along with HTML pages generated by Doxygen[54] 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 API as *ASI-enabled codes*, and we refer to software that invoke 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 (**H**) and overlap (**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.[34] 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 initialisation, key data objects are communicated to/from DFTB+ and callback functions registered before the request for execution of a DFTB+ calculation. Callback functions give direct access to data objects within DFTB+, thus causing near-zero computational cost, and also adapting to the chosen parallelisation schemes. The callback functions are invoked during the execution process, providing external access to data objects when calculated. Derived quantities, such as energy, forces, stress, 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 Figure 2, and contrasted against the ACEhamiltonians interface. Further details of the key ASI functions are provided in the SI.

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 separately 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 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.

The calculation is performed self-consistently: the energy of both molecules is calculated with zero external potential initially; 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 Figure 5). Figure 5 shows that the electrostatic potential is dominant for the intermolecular interaction at large distance (> 4 Å), which is the expected behaviour.

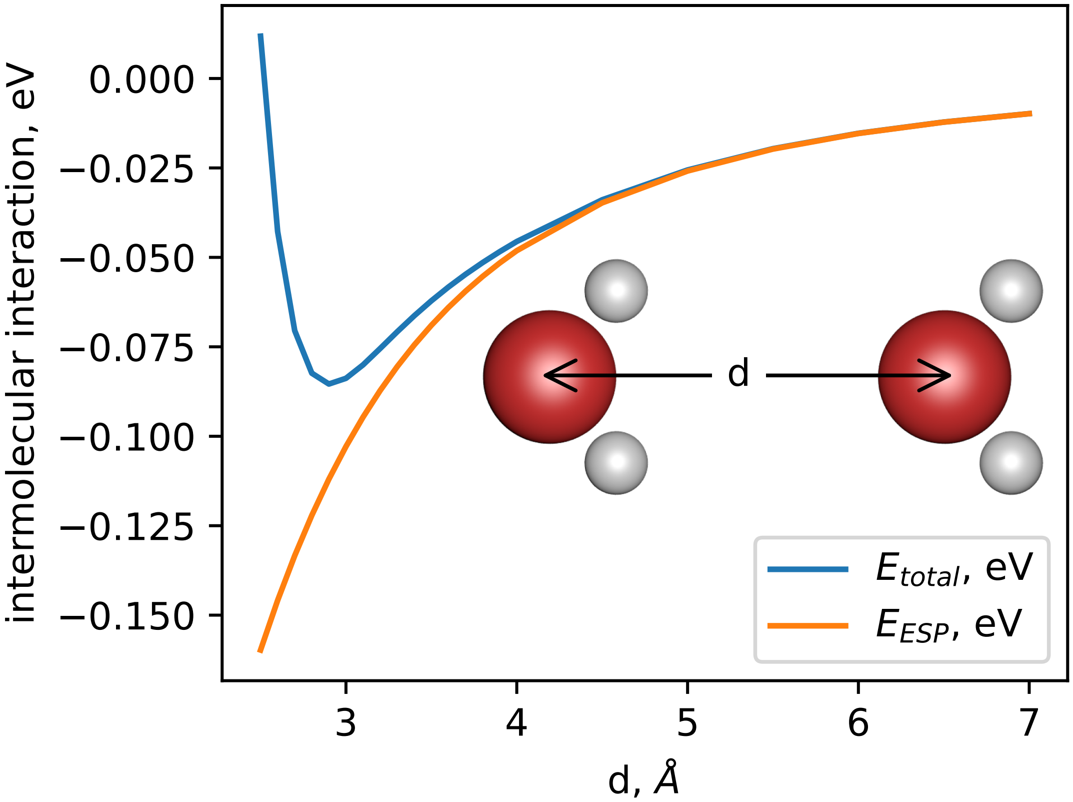

FIG. 5. Example of electrostatic embedding in DFTB+ achieved with the ASI interface. The distance (d) between two water dimers, as shown in the inset. The graph shows the interaction energy as a function of d (blue line), and the electrostatic embedding energy evaluated with `ASI` API in a self-consistent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

2. *Electronic structure transfer*

The `ASI` implementation in `DFTB+` supports the import of Hamiltonian (**H**) and overlap (**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 computation of the electronic band structure for bulk Al (Supporting Information, SI, Figure S1), where **H** and **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 analyse the band structure in `DFTB+`. The resulting band structure is given in the SI: 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 in 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, which 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 evaluation of emebedding electostatic 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.[36] Similarly, calculations using a density-functional ground state reference, which then uses the DFTB approximated random-phase excitation poles, becomes possible.[59] 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.[60]

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.

DATA AND CODE 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 a [61] and [62] 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 is available at https://gitlab.com/pvst/asi. The interface specification and DFTB+ implementation are available at https://pvst.gitlab.io/.

CREDIT AUTHOR STATEMENT

Pavel Stishenko: Methodology, Software, Writing - Original Draft, Writing - Review & Editing, Visualization. **Adam McSloy**: Software, Visualization, Writing - Review & Editing. **Berk Onat**: Software. **Ben Hourahine**: Conceptualization, Methodology, Writing - Original Draft, Writing - Review & Editing, Visualization, Supervision, Project administration, Funding acquisition. **Reinhard J. Maurer**: Conceptualization, Writing - Original Draft, Writing - Review & Editing, Supervision, Project administration, Funding acquisition. **James Kermode**: Conceptualization, Methodology, Writing - Original Draft, Writing - Review & Editing, Visualization, Supervision, Project administration, Funding acquisition. **Andrew Logsdail**: Conceptualization, Writing - Original Draft, Writing - Review & Editing, Supervision, Project administration, Funding acquisition.

ACKNOWLEDGMENTS

This work was financially supported by the Leverhulme Trust Research Project Grant (RPG-2017-191) and the NOMAD Centre of Excellence (European Commission grant agreement ID 951786). AJL and PVS acknowledge funding by the UKRI Future Leaders Fellowship program (MR/T018372/1, MR/Y034279/1). RJM acknowledges funding by the UKRI Future Leaders Fellowship program (MR/S016023/1, MR/X023109/1) and a UKRI frontier research grant (EP/X014088/1). PVS, RJM and AJL acknowledge funding from the ARCHER2 eCSE Programme (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 (EP/T022108/1), and the UK Car-Parinello consortium (EP/P022065/1) and the U.K. High Performance Computing "Materials Chemistry Consortium" (EP/R029431/1, EP/X035859/1) for access to the ARCHER2 high-performance

computing facility.

[1] M. Elstner and G. Seifert, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **372**, 20120483 (2014).

[2] C. Bannwarth, B. Hourahine, and J. Moussa, in *Software for electronic structure based simulations in chemistry and materials*, IOP Roadmap, edited by T. Windus and V. Blum (Electronic Structure, 2024) volume is still in review.

[3] Y. Nishimura and H. Nakai, Chemistry Letters **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, The Journal of Physical Chemistry Letters **9**, 399 (2018).

[6] G. Jha and T. Heine, Journal of Chemical Theory and Computation **18**, 4472 (2022).

[7] J. A. Keith, V. Vassilev-Galindo, B. Cheng, S. Chmiela, M. Gastegger, K.-R. Müller, and A. Tkatchenko, Chemical Reviews **121**, 9816 (2021).

[8] J. Westermayr, M. Gastegger, K. T. Schütt, and R. J. Maurer, The Journal of Chemical Physics **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, The Journal of Chemical Physics **159**, 110901 (2023).

[10] J. M. Knaup, B. Hourahine, and T. Frauenheim, The Journal of Physical Chemistry A **111**, 5637 (2007).

[11] N. F. Aguirre, A. Morgenstern, M. J. Cawkwell, E. R. Batista, and P. Yang, Journal of Chemical Theory and Computation **16**, 1469 (2020).

[12] A. S. Hutama, C.-p. Chou, Y. Nishimura, H. A. Witek, and S. Irle, The Journal of Physical Chemistry A **125**, 2184 (2021).

[13] M. Stöhr, L. Medrano Sandonas, and A. Tkatchenko, The Journal of Physical Chemistry Letters **11**, 6835 (2020).

[14] K. T. Schütt, M. Gastegger, A. Tkatchenko, K.-R. Müller, and R. J. Maurer, Nature Communications **10**, 5024 (2019).

[15] M. Gastegger, A. McSloy, M. Luya, K. T. Schütt, and R. J. Maurer, Journal of Chemical Physics **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*, Vol. 34 (Curran Associates, Inc., 2021) pp. 14434–14447.

[17] J. Nigam, M. J. Willatt, and M. Ceriotti, The Journal of Chemical Physics **156**, 014115 (2022).

[18] L. Zhang, B. Onat, G. Dusson, A. McSloy, G. Anand, R. J. Maurer, C. Ortner, and J. R. Kermode, npj Computational Materials **8**, 1 (2022), number: 1 Publisher: Nature Publishing Group.

[19] H. Li, Z. Wang, N. Zou, M. Ye, R. Xu, X. Gong, W. Duan, and Y. Xu, Nature Computational Science **2**, 367 (2022).

[20] P. O. Dral, O. A. Von Lilienfeld, and W. Thiel, Journal of Chemical Theory and Computation **11**, 2120 (2015).

[21] G. Zhou, N. Lubbers, K. Barros, S. Tretiak, and B. Nebgen, Proceedings of the National Academy of Sciences **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, The Journal of Chemical Physics **158**, 034801 (2023).

[23] J. J. Mortensen, L. B. Hansen, and K. W. Jacobsen, Phys. Rev. B Condens. Matter **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.*, The Journal of Chemical Physics **152** (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.*, The Journal of Chemical Physics **153** (2020).

[26] M. McIlroy, E. Pinson, and B. Tague, The Bell system technical journal **57**, 1899 (1978).

[27] P. Blaha, K. Schwarz, F. Tran, R. Laskowski, G. K. H. Madsen, and L. D. Marks, The Journal of Chemical Physics **152**, 074101 (2020).

[28] J. R. Kermode, Journal of Physics: Condensed Matter **32**, 305901 (2020).

[29] 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, Computer Physics Communications **236**, 214 (2019).

[30] The Molssi Driver Interface Library, `https://molssi-mdi.github.io/MDI_Library` (2022), accessed: 2022-10-24.

[31] A. H. Larsen, J. J. Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dułak, J. Friis, M. N. Groves, B. Hammer, C. Hargus, *et al.*, Journal of Physics: Condensed Matter **29**, 273002 (2017).

[32] G. Pizzi, A. Cepellotti, R. Sabatini, N. Marzari, and B. Kozinsky, Computational Materials Science **111**, 218 (2016).

[33] 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.*, Scientific data **3**, 1 (2016).

[34] P. V. Stishenko, T. W. Keal, S. M. Woodley, V. Blum, B. Hourahine, R. J. Maurer, and A. J. Logsdail, Journal of Open Source Software **8**, 5186 (2023).

[35] V. Blum, R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, K. Reuter, and M. Scheffler, Computer Physics Communications **180**, 2175 (2009).

[36] 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, Journal of Chemical Physics **152**, 124101 (2020).

[37] 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, The Journal of Chemical Physics **157**, 039901 (2022).

[38] 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 Transactions on Mathematical Software **28**, 135 (2002).

[39] C. Bannwarth, E. Caldeweyher, S. Ehlert, A. Hansen, P. Pracht, J. Seibert, S. Spicher, and S. Grimme, WIREs Comput. Mol. Sci. **11**, e01493 (2020).

[40] R. S. Mulliken, The Journal of Chemical Physics **23**, 1833 (2004).

[41] M. Elstner, D. Porezag, G. Jungnickel, J. Elsner, M. Haugk, T. Frauenheim, S. Suhai, and G. Seifert, Physical Review B **58**, 7260 (1998).

[42] C. Bannwarth, S. Ehlert, and S. Grimme, Journal of Chemical Theory and Computation **15**, 1652 (2019).

[43] B. Aradi, B. Hourahine, and T. Frauenheim, The Journal of Physical Chemistry A **111**, 5678 (2007).

[44] 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).

[45] 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).

[46] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki, Numerical Computations with GPUs , 1 (2014).

[47] V.-Q. Vuong, C. Cevallos, B. Hourahine, B. Aradi, J. Jakowski, S. Irle, and C. Camacho, The Journal of Chemical Physics **158** (2023).

[48] V. W. Yu, J. Moussa, P. Kůs, A. Marek, P. Messmer, M. Yoon, H. Lederer, and V. Blum, Computer Physics Communications **262**, 107808 (2021).

[49] 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, Álvaro Vázquez-Mayagoitia, C. Yang, and V. Blum, Computer Physics Communications **256**, 107459 (2020).

[50] 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, DFTB+, a software package for efficient approximate density functional theory based atomistic simulations, https://doi.org/10.5281/zenodo.8117766 (2023), 23.1.

[51] F. P. Bonafé, B. Aradi, B. Hourahine, C. R. Medrano, F. J. Hernández, T. Frauenheim, and C. G. Sánchez, Journal of Chemical Theory and Computation **16**, 4454 (2020).

[52] I. Free Software Foundation, GNU Lesser General Public License, version 3, (2007).

[53] B. Aradi, FyTest – instant Fortran unit testing, https://github.com/aradi/fytest (2021).

[54] D. van Heesch, Doxygen, https://www.doxygen.nl/ (1997 – 2024).

[55] C. MacMackin, FORtran Documenter, https://github.com/Fortran-FOSS-Programmers/ford (2015 – 2023).

[56] A. Pecchia and A. Di Carlo, Rep. Prog. Phys. **67**, 1497 (2004).

[57] I. Nikiforov, B. Hourahine, B. Aradi, T. Frauenheim, and T. Dumitrică, The Journal of Chemical Physics **139**, 094110 (2013).

[58] J. P. Perdew, K. Burke, and M. Ernzerhof, Physical review letters **77**, 3865 (1996).

[59] R. Rüger, E. van Lenthe, T. Heine, and L. Visscher, The Journal of Chemical Physics **144**, 184103 (2016).

[60] D. Maag, J. Böser, H. A. Witek, B. Hourahine, M. Elstner, and T. Kubař, The Journal of Chemical Physics **158** (2023).

[61] P. Stishenko and B. Hourahine, Pull request #1335: "asirebase", `https://github.com/dftbplus/dftbplus/pull/1335` (2024).

[62] B. Hourahine, Pull request #1420: "external model interface", `https://github.com/dftbplus/dftbplus/pull/1420` (2024).