The Journal of Chemical Physics

The CECAM electronic structure library and the modular software development paradigm

Cite as: J. Chem. Phys. **153**, 024117 (2020); https://doi.org/10.1063/5.0012901
Submitted: 06 May 2020 . Accepted: 08 June 2020 . Published Online: 13 July 2020

Micael J. T. Oliveira, Nick Papior, Yann Pouillon, Volker Blum, Emilio Artacho, Damien Caliste, Fabiano Corsetti, Stefano de Gironcoli, Alin M. Elena, Alberto García, Víctor M. García-Suárez, Luigi Genovese, William P. Huhn, Georg Huhs, Sebastian Kokott, Emine Küçükbenli, Ask H. Larsen, Alfio Lazzaro, Irina V. Lebedeva, Yingzhou Li, David López-Durán, Pablo López-Tarifa, Martin Lüders, Miguel A. L. Marques, Jan Minar, Stephan Mohr, Arash A. Mostofi, Alan O'Cais, Mike C. Payne, Thomas Ruh, Daniel G. A. Smith, José M. Soler, David A. Strubbe, Nicolas Tancogne-Dejean, Dominic Tildesley, Marc Torrent, and Victor Wen-zhe Yu

COLLECTIONS

Note: This article is part of the JCP Special Topic on Electronic Structure Software.

This paper was selected as Featured

The Journal of Chemical Physics ARTICLE scitation.org/journal/jcp

The CECAM electronic structure library and the modular software development paradigm

Cite as: J. Chem. Phys. 153, 024117 (2020); doi: 10.1063/5.0012901
Submitted: 6 May 2020 • Accepted: 8 June 2020 •
Published Online: 13 July 2020

Micael J. T. Oliveira,[1,a] Nick Papior,[2,b] Yann Pouillon,[3,4,c] Volker Blum,[5,6,d] Emilio Artacho,[7,8,9,e] Damien Caliste,[10] Fabiano Corsetti,[11,12] Stefano de Gironcoli,[13] Alin M. Elena,[14] Alberto García,[15] Víctor M. García-Suárez,[16] Luigi Genovese,[10] William P. Huhn,[5] Georg Huhs,[17] Sebastian Kokott,[18] Emine Küçükbenli,[13,19] Ask H. Larsen,[4,20] Alfio Lazzaro,[21] Irina V. Lebedeva,[22] Yingzhou Li,[23] David López-Durán,[22] Pablo López-Tarifa,[24] Martin Lüders,[1,14] Miguel A. L. Marques,[25] Jan Minar,[26] Stephan Mohr,[17] Arash A. Mostofi,[11] Alan O'Cais,[27] Mike C. Payne,[9] Thomas Ruh,[28] Daniel G. A. Smith,[29] José M. Soler,[30] David A. Strubbe,[31] Nicolas Tancogne-Dejean,[1] Dominic Tildesley,[32] Marc Torrent,[33,34] and Victor Wen-zhe Yu[5]

AFFILIATIONS

[1]Max Planck Institute for the Structure and Dynamics of Matter, D-22761 Hamburg, Germany
[2]DTU Computing Center, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark
[3]Departamento CITIMAC, Universidad de Cantabria, Santander, Spain
[4]Simune Atomistics, 20018 San Sebastián, Spain
[5]Department of Mechanical Engineering and Materials Science, Duke University, Durham, North Carolina 27708, USA
[6]Department of Chemistry, Duke University, Durham, North Carolina 27708, USA
[7]CIC Nanogune BRTA and DIPC, 20018 San Sebastián, Spain
[8]Ikerbasque, Basque Foundation for Science, 48011 Bilbao, Spain
[9]Theory of Condensed Matter, Cavendish Laboratory, University of Cambridge, Cambridge CB3 0HE, United Kingdom
[10]Department of Physics, IRIG, Univ. Grenoble Alpes and CEA, F-38000 Grenoble, France
[11]Departments of Materials and Physics, and the Thomas Young Centre for Theory and Simulation of Materials, Imperial College London, London SW7 2AZ, United Kingdom
[12]Synopsys Denmark, 2100 Copenhagen, Denmark
[13]Scuola Internazionale Superiore di Studi Avanzati, 34136 Trieste, Italy
[14]Scientific Computing Department, Daresbury Laboratory, Warrington WA4 4AD, United Kingdom
[15]Institut de Ciència de Materials de Barcelona (ICMAB-CSIC), Bellaterra E-08193, Spain
[16]Departamento de Física, Universidad de Oviedo & CINN, 33007 Oviedo, Spain
[17]Barcelona Supercomputing Center (BSC), 08034 Barcelona, Spain
[18]Fritz Haber Institut, 14195 Berlin, Germany
[19]John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, Massachusetts 02138, USA
[20]Nano-Bio Spectroscopy Group and ETSF, Departamento de Física de Materiales, Universidad del País Vasco UPV/EHU, 20018 San Sebastián, Spain
[21]Department of Chemistry, University of Zürich, CH-8057 Zürich, Switzerland
[22]CIC Nanogune BRTA, 20018 San Sebastián, Spain
[23]Department of Mathematics, Duke University, Durham, North Carolina 27708-0320, USA
[24]Centro de Física de Materiales, Centro Mixto CSIC-UPV/EHU, 20018 San Sebastián, Spain
[25]Institut für Physik, Martin-Luther-Universität Halle-Wittenberg, 06120 Halle (Saale), Germany
[26]New Technologies Research Centre, University of West Bohemia, 301 00 Plzen, Czech Republic

[27]Institute for Advanced Simulation (IAS), Jülich Supercomputing Centre (JSC), Forschungszentrum Jülich GmbH, 52425 Jülich, Germany
[28]Institute of Materials Chemistry, TU Wien, 1060 Vienna, Austria
[29]Molecular Sciences Software Institute, Blacksburg, Virginia 24060, USA
[30]Departamento e Instituto de Física de la Materia Condensada (IFIMAC), Universidad Autónoma de Madrid, 28049 Madrid, Spain
[31]Department of Physics, University of California, Merced, California 95343, USA
[32]School of Chemistry, University of Southampton, Southampton SO17 1BJ, United Kingdom
[33]CEA, DAM, DIF, F-91297 Arpajon, France
[34]Université Paris-Saclay, CEA, Laboratoire Matière en Conditions Extrêmes, 91680 Bruyères-le-Châtel, France

Note: This article is part of the JCP Special Topic on Electronic Structure Software.
[a]**Electronic mail:** micael.oliveira@mpsd.mpg.de
[b]**Electronic mail:** nickpapior@gmail.com
[c]**Electronic mail:** yann.pouillon@materialsevolution.es
[d]**Electronic mail:** volker.blum@duke.edu
[e]**Author to whom correspondence should be addressed:** ea245@cam.ac.uk

ABSTRACT

First-principles electronic structure calculations are now accessible to a very large community of users across many disciplines, thanks to many successful software packages, some of which are described in this special issue. The traditional coding paradigm for such packages is *monolithic*, i.e., regardless of how modular its internal structure may be, the code is built independently from others, essentially from the compiler up, possibly with the exception of linear-algebra and message-passing libraries. This model has endured and been quite successful for decades. The successful evolution of the electronic structure methodology itself, however, has resulted in an increasing complexity and an ever longer list of features expected within all software packages, which implies a growing amount of replication between different packages, not only in the initial coding but, more importantly, every time a code needs to be re-engineered to adapt to the evolution of computer hardware architecture. The Electronic Structure Library (ESL) was initiated by CECAM (the European Centre for Atomic and Molecular Calculations) to catalyze a paradigm shift away from the monolithic model and promote modularization, with the ambition to extract common tasks from electronic structure codes and redesign them as open-source libraries available to everybody. Such libraries include "heavy-duty" ones that have the potential for a high degree of parallelization and adaptation to novel hardware *within them*, thereby separating the sophisticated computer science aspects of performance optimization and re-engineering from the computational science done by, e.g., physicists and chemists when implementing new ideas. We envisage that this modular paradigm will improve overall coding efficiency and enable specialists (whether they be computer scientists or computational scientists) to use their skills more effectively and will lead to a more dynamic evolution of software in the community as well as lower barriers to entry for new developers. The model comes with new challenges, though. The building and compilation of a code based on many interdependent libraries (and their versions) is a much more complex task than that of a code delivered in a single self-contained package. Here, we describe the state of the ESL, the different libraries it now contains, the short- and mid-term plans for further libraries, and the way the new challenges are faced. The ESL is a community initiative into which several pre-existing codes and their developers have contributed with their software and efforts, from which several codes are already benefiting, and which remains open to the community.

 https://doi.org/10.1063/5.0012901

I. INTRODUCTION

Electronic structure theory is among the most productive branches of computational science today.[1] The necessary underlying level of theory—Dirac's equation—is analytically known exactly.[2] It is applicable to condensed matter physics, chemistry, and materials science and, in fact, touches all branches of engineering—whenever either modified or completely new technologically more capable materials are needed. Practical, i.e., numerically tractable, approximations to Dirac's equation can be used to predict the properties of molecules, solids, liquids, and interfaces, including their responses to environmental stimuli (fields, currents, mechanical stimuli, etc.). They typically provide sufficient accuracy and reliability[3] to formulate experimentally testable hypotheses and, ultimately, accelerate the discovery and development of "new" molecules and materials. The growth of the field is reflected in a plethora of existing and new software developments that implement the aspects of electronic structure theory either for specialized or rather broad general use cases. The community-wide `psi-k.net` website lists over 30 "codes" at the time of writing (December 2019), and 74 individual code projects are listed at the "Community Code Database" of the U.S. based Molecular Software Sciences Institute (MolSSI),[4] another community-bridging organization working to support a broad set of "codes" and their users.[5,6]

While the electronic structure community (ESC) is thus extremely active in developing software that enables a host of

scientific insights, developments have historically occurred in the form of different individual software packages that are largely distinct from each other at the code level. Notable exceptions are numerical and/or performance related libraries that are often generic to the broader computational community, e.g., basic linear algebra subroutines (BLAS),[7] exploiting parallelism at the message passing interface (MPI) level,[8] higher-level linear algebra utilities (most importantly LAPACK[9] and its parallel counterpart, ScaLAPACK[10]), or fast Fourier transform of the West (FFTW).[11]

ESC software development has historically taken place within a model of largely monolithic programs in which, on top of a main quantum engine, all further developments are incorporated incrementally. The (now) more traditional electronic structure codes are steadily growing, each incorporating all or many of the developments that have become standard in the community. This model is illustrated in Fig. 1(a). Furthermore, each code needs re-engineering to adapt to the constant hardware evolution, most notably in high-performance computing, and most of the re-engineering is carried out on tasks that are common to all or most of the codes. In addition to this obvious inefficiency, two other important problems are inherent in the monolithic model. First, it stifles innovation: novel methodological (physics) ideas within the wider community can only be implemented by joining any of the pre-existing efforts. It is increasingly hard to start a project from scratch. This problem is partly addressed by the open-source model of programming, well established in some modern electronic structure projects, to which novel ideas can be incorporated by external coders, at least in principle (note, however, that poor quality, undocumented open source code does not fulfill this requirement). Second, the monolithic model allows very little differentiation in the profiles of human resources needed for the project: there is a need for people with expertise in the state-of-the-art for both computational science (e.g., physics and chemistry) and computer science (e.g., software engineering, performance optimization, hardware architecture, and numerical analysis).

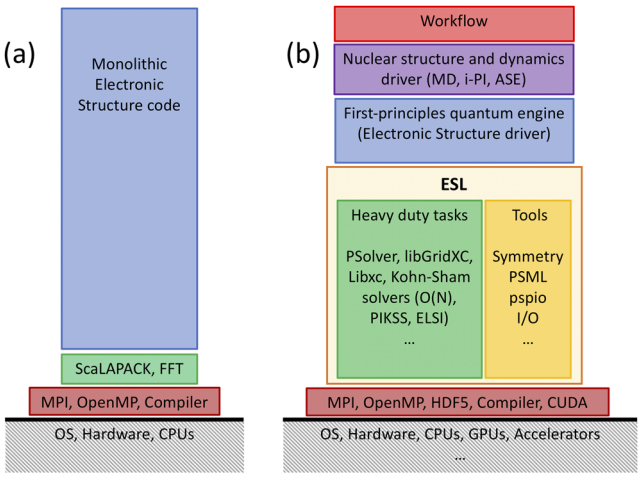

FIG. 1. Comparison of the traditional monolithic and emerging modular paradigms in electronic structure coding. One of today's electronic-structure codes—large blue box in panel (a)—thins down into the higher-level electronic structure driver that defines the particular code—blue box in panel (b), allowing for a more specialized and sustainable development of the different parts of the software. The acronyms indicate operating system (OS), fast Fourier transforms (FFT), input/output (I/O), molecular dynamics (MD), and linear scaling with the number of atoms, $\mathcal{O}(N)$, in addition to the central and graphical processing units, CPU and GPU, respectively. Steering upper-level drivers are distinguished between versatile toolkits, such as ASE, and handlers of massive amounts of quantum-engine replicas, such as i-PI. In addition to the well-known MPI, OpenMP, CUDA, and HDF libraries, other acronyms and names related to libraries are described in Secs. I, II, and IV.

II. SHARED LIBRARIES AND THE ESL

A. The library sharing movement

Partly in response to the problems mentioned above, and partly following the spirit of the open access movement and inspired by well-established practices in software engineering, the computational physics and chemistry communities have witnessed the appearance of libraries—understanding this term broadly—which perform particular, well-defined tasks that are common to many codes. We will not review this movement here but will illustrate it with some examples from the ESC. Take, for instance, the exploitation of symmetry in computational simulation of both molecular and crystalline systems. This involves a well-defined set of tasks, from recognizing the symmetry group for a specified structure, to the labeling of eigenstates according to irreducible representations, including the reduction in the eigenproblem complexity or the optimization of Brillouin-zone sampling. Several libraries have appeared within this free sharing movement to perform these tasks (e.g., the `spglib` library[12]).

The handling of symmetry is an example of a very general pre- and post-processing tool whose function can be defined completely independently and is one of many other similar possibilities that constitute opportunities for creating libraries. Another notable case is that of `wannier90`,[13] which not only calculates maximally localized Wannier functions[14] from the outputs of electronic structure codes but can also determine many properties using these Wannier functions. Remarkably, the authors' ambition from the very beginning of the project was to maximize its applicability to all classes of electronic structure methods, and they have managed to limit code dependencies to an absolute minimum. This has enabled a very widespread adoption within the wider electronic structure community (see Sec. IV K).

In addition to these kinds of tools, other sharing/library efforts have been appearing, which we can characterize as top-level steering codes and low-level routines, as shown in Fig. 1(b), which illustrates the new emerging paradigm. Among the former is the integration of electronic structure codes as "solvers" or "quantum engines" into broader frameworks, typically handling the nuclear degrees of freedom, such as the python-based Atomic Simulation Environment (ASE)[15] or the i-PI framework[16] for classical and path-integral molecular dynamics (MD). Both of these support a large number of underlying electronic structure codes. Also in the top-level category, much effort is now being dedicated to general-purpose workflow tools that steer and automate the running of electronic structure codes in complex procedures and encompass the ambition of versatility (see, e.g., the AiiDA project[17]), providing a much more detailed picture of how electronic structure methods are applied as

compared to a decade ago. As a pioneering example of a low-level shared library, we mention Libxc,[18,19] which implements hundreds of local and semilocal exchange–correlation functionals and is now very widely used (see Sec. IV B).

There are many other tasks and needs in the electronic structure that may be generically abstracted in the form of shared libraries, with common frameworks and shared workloads in order to more readily achieve the maturity of the established functionality, numerical correctness, and continued development of new functionality at the same time. Developing electronic-structure software based on common standards, libraries, application programming interfaces (APIs), and flexible software components is a trend that is therefore gaining prominence in the field.

Additionally, at a social level, such shared developments bring different communities together and reinforce the existing collaborations within the communities themselves. Significant challenges on this path are often simple, related to human time and workload and include the following: identifying and locating an existing solution to a code problem at the time when it is needed; finding and reading documentation to understand and co-develop software originally written by others; being able to download, install, and successfully link to an array of disparate software pieces on a given, often individualized, computed platform; and having an effective pathway to communicate with the developers of the library for advice and to offer feedback and suggestions for improvement. These issues are not specific to the ESC but rather reflect generic challenges that confront all shared software development efforts.

B. ESL

1. Concept

This is where the "Electronic Structure Library" (ESL)[20] enters, the subject of the present paper. A key goal of the ESL is to alleviate and overcome the issues mentioned above, creating an effective collaboration platform for shared software developments, where these make sense. Our vision is to sustain a community that develops, distributes, and oversees electronic structure libraries for the benefit of all electronic structure codes.

The ESL started in 2014 as a CECAM initiative, with the aim of stimulating the segregation of well-defined tasks into shared libraries, pushing the model of Fig. 1(b), and confronting the challenges it entails. From the beginning, the work of the ESL has been done by programmers actively involved in successful electronic structure codes and the ESL initiative has been supported by the development teams of these codes, which include ABINIT,[21] Siesta,[22] Octopus,[23] Quantum ESPRESSO (QE),[24] BigDFT,[25] FHI-aims,[26] and gpaw.[27]

The initial efforts focused on three aspects: (i) identifying the existing libraries suitable for inclusion in the ESL, (ii) extracting and re-coding as libraries a number of sub-packages from the community codes, and (iii) incorporating these libraries into other participant codes.

Ongoing efforts within the ESL include improving the coordination between and interoperability of the various software modules, expanding their integration into large software development projects (e.g., some of the main electronic structure codes in the community), and making it easier to seamlessly distribute a consistent bundle of libraries and software modules (see Sec. V).

A key enabler in all this process has been the will to overcome the monolithic mentality, both at a scientific level (one research group, one code) and at a business model level (free software vs open-source vs proprietary), allowing collaborations between communities and making new public–private partnerships possible.

In addition to the obvious goal of avoiding re-inventing the wheel for every code, by re-coding the well-known algorithms for well-established tasks, two other important advantages are foreseen. The first relates to human resources. Electronic structure codes encompass sophisticated physics and sophisticated software engineering. The monolithic development model demands highly educated personnel with expertise in both areas. An efficient segregation of tasks into libraries would allow an abstraction of the low-level detail for physicists or chemists coding at a high-level, while software engineers could maintain and evolve the low-level software without needing a high-level of expertise in the science used.

A related second advantage is that a widely used ESL library or set of libraries, with well-defined APIs, would offer a good target for re-coding for software engineers working close to the cutting-edge of hardware developments and high performance computing (HPC) centers. This is, of course, a continuous process as it has to be done at each step of hardware evolution. Indeed, it has already happened with, e.g., Intel offering their own implementation of linear-algebra libraries adapted to their own compilers and processors. The ESL should be able to offer many more targets for optimization. It should also be remembered that there is currently a substantial level of resource dedicated to re-engineering codes for new hardware, both at individual HPC centers and national (or trans-national, e.g., European Union) funded research agencies. These efforts are usually directed toward particular codes. Dedicating these efforts to libraries would be more efficient, serve the community more widely, and would also be easier to maintain as libraries are naturally composed of independent modules. Ideally, scientists should aspire to adapt their codes to new computers and to new computer paradigms as this is usually the only way to access the largest computational systems. Through the ESL, this could be achieved just by linking to the latest library implementation for a given computer architecture.

These elements of the ESL vision rely, however, on the conversion into libraries of massively parallel heavy-duty code, which is an extremely ambitious goal. There has been an emphasis on heavy-duty tasks in the ESL efforts so far, although work has not focused exclusively on this. These efforts are described in Sec. IV. The segregation of heavy-duty libraries involves many new challenges, which we now describe.

2. Challenges

In addition to the challenges mentioned above referring to shared software in general, the model proposed here faces a number of additional important challenges. First, building a binary code (compilation and linking), which depends on many libraries, and often their specific version is substantially more difficult than for a self-contained (monolithic) program. Furthermore, the complexity of the heterogeneous environments typically encountered at HPC installations makes the build even harder and more diverse. Ours is not the first community to face these problems, and a significant part of the ESL effort is expended in the bundling and building strategies for the ESL, as described in Sec. V.

A second important challenge is the loss of the global coherence in data structures and parallelization that monolithic programs can adopt (although it is not always possible or convenient). This implies the need for conversion routines to adapt data structures from one section of the code to another. Again, this challenge is not new, and it represents an intrinsic element of this modular paradigm. Associated with these conversions, and in general, with the whole strategy, is an expected loss of efficiency, compared to that achievable within perfectly coded (and constantly maintained) monolithic programs. However, the savings in (limited) human resources that modularity brings are likely to outweigh a loss in (continuously expanding) hardware cycles. An analogy can be made with the controversies in the early 1970s regarding the use of high-level languages (instead of machine language) for the implementation of system software.[28] It is now clear that the apparently wasteful road led to significant progress.

Another challenge faced by the ESL to date stems from the fact that the majority of the libraries currently in the ESL have been extracted from the pre-existing electronic structure codes. This means that the API and internals of the library were chosen with its parent environment in mind. Finally, the issue of licensing should be mentioned. Different libraries are released under different licenses, which may impose conditions on the licenses under which the using codes are distributed. This represents a challenge as well, although of a different kind.

III. COMMON ELEMENTS OF ELECTRONIC STRUCTURE CODES

Before describing the existing library implementations in the ESL, we give here a brief overview of the task that they are supposed to handle or, to put it more simply, what are the common elements of the electronic structure codes. From the many available methods to approximate Dirac's equation in a computationally tractable form, the majority fall into one of two broad classes: density functional theory (DFT) and wavefunction based methods. In this paper, we concentrate on the former, although many of the tools described here are also useful for other methods also based on effective single-particle models, such as Hartree–Fock.

In the non-relativistic limit, the equations to be solved for ground-state DFT are the Kohn–Sham equations,[29]

$$\hat{h}_{\mathrm{KS}}[n]\phi_i(\boldsymbol{r}) = \epsilon_i\phi_i(\boldsymbol{r}), \tag{1}$$

where ϕ_i and ϵ_i are the Kohn–Sham (KS) orbitals and eigenenergies, respectively, and $\hat{h}_{\mathrm{KS}}$ is the Kohn—-Sham Hamiltonian. The Hamiltonian is usually decomposed in the following way:

$$\hat{h}_{\mathrm{KS}}[n] = \hat{t}_s + v_{\mathrm{ext}} + v_{\mathrm{H}}[n] + v_{\mathrm{xc}}[n], \tag{2}$$

where v_{ext} is the external potential (typically the potential generated by the nuclei), v_{H} is the Hartree potential, and v_{xc} is the exchange and correlation potential. $\hat{t}_s$ is the single-particle kinetic energy operator. In non-relativistic form,

$$\hat{t}_s = -\frac{1}{2}\nabla^2. \tag{3}$$

However, practically, every electronic structure code employs at least a scalar-relativistic variant of $\hat{t}_s$ (the applicability of the non-relativistic expression is limited to the lightest chemical elements only). In codes employing pseudopotential-type techniques (see below), relativity is usually incorporated implicitly through the form of the projectors. In all-electron codes, explicit scalar-relativistic forms of $\hat{t}_s$ are used. The Kohn–Sham equations are a set of one-particle equations that need to be solved self-consistently as several terms in Eq. (2) are functionals of the electronic density,

$$n(\boldsymbol{r}) = \sum_i f_i|\phi_i(\boldsymbol{r})|^2. \tag{4}$$

f_i are occupation numbers, ensuring that the orbitals are only occupied as far as there are electrons (i.e., $\sum_i f_i = N_{\mathrm{el}}$, where N_{el} is the number of electrons in the system). Any code that aims to solve the Kohn–Sham equations must therefore perform the following tasks: (1) given a set of atomic coordinates, evaluate v_{ext}; (2) evaluate $v_{\mathrm{H}}[n]$; (3) evaluate $v_{\mathrm{xc}}[n]$; (4) solve the eigenvalue problem of Eq. (1); and (5) find the density that solves the self-consistency problem. Each of these steps thus represents an opportunity for electronic structure packages to share and reuse code:

1. To reduce the computational cost, many DFT codes use the pseudopotential approximation.[30] The pseudopotentials are normally generated by specialized codes that output them using a particular one of the existing file formats. Therefore, codes that want to use a specific pseudopotential are required to know the corresponding file format to parse the corresponding information.
2. The Hartree potential $v_{\mathrm{H}}[n](\boldsymbol{r})$ is defined as

$$v_{\mathrm{H}}[n](\boldsymbol{r}) = \int \mathrm{d}\boldsymbol{r}'\,\frac{n(\boldsymbol{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|}. \tag{5}$$

   Direct evaluation of this integral is not usually numerically efficient, and it is common practice to instead solve the corresponding Poisson equation.
3. Many hundreds of different approximations to the exchange–correlation functional have been proposed, some of which require the evaluation of long, complex mathematical expressions. Implementing such approximations is thus a tedious, error prone task.
4. Many different methods exist in the literature for solving eigenvalue problems such as Eq. (1). Upon discretization of the orbitals ϕ, one can write the problem in the language of matrices and vectors. Then, solving Eq. (1) reduces to the standard linear algebra problem of diagonalizing a matrix, in this case the Hamiltonian matrix. For cases where the size of this matrix is too large for direct diagonalization, either due to the memory or computational time required, iterative eigensolvers can be used, which only require the result of the Hamiltonian operating on an orbital, or alternative formulations of the problem can be solved, such as the ones based on Green's functions or Fermi-operator expansions.
5. Finding the density that solves the self-consistency problem is usually done iteratively: starting from a guess for the density, one solves Eq. (1), thus obtaining a new set of orbitals ϕ, which, in turn, are used to obtain a new density. The process is then

repeated using the new density until the changes in the density are smaller than some defined threshold. Since the total computational cost strongly depends on how fast the iterative procedure converges, many methods are available to accelerate this process.

In the case of wavefunction based methods, many of these require a solution to the Hartree–Fock equations as a starting point. These equations share many similarities with the Kohn–Sham equations: both are sets of one-particle equations that need to be solved self-consistently. This further increases the opportunities for code sharing and reuse among electronic structure packages.

To numerically solve either the Kohn–Sham or Hartree–Fock equation, the relevant quantities are typically discretized in some way, either using basis-sets or grids. Each type of discretization requires specialized functions that can, in principle, be shared among codes that use the same basis-set or type of grid. For example, atom-centered basis sets require the efficient evaluation of one- and two-particle integrals.

Along with the common elements of electronic structure packages that are directly related to the equations solved, other types of operations are also performed by most ES codes. The prime examples are I/O operations, which range from parsing an input file to writing physical quantities of interest to disk for visualization or further processing.

IV. EXISTING LIBRARY IMPLEMENTATIONS IN THE ESL

In the following, we briefly present the libraries and packages that are currently part of the ESL, giving a brief description of their scope, history, and use cases. They are tabulated in Table I.

A. PSolver

Electrostatic potentials play a fundamental role in nearly any field of physics and chemistry. It is, therefore, essential to have efficient algorithms to find the electrostatic potential V arising from a charge distribution ρ [associated with the particle density n in Eqs. (4) and (5)] in a dielectric medium described by the dielectric constant $\epsilon(\mathbf{r})$ or, in other words, to solve the generalized Poisson's equation,

$$\nabla \cdot \epsilon(\mathbf{r}) \nabla \phi(\mathbf{r}) = -4\pi\rho(\mathbf{r}). \tag{6}$$

The large variety of situations in which this equation is encountered led us to address this problem for different choices of the boundary conditions (BC). The long-range behavior of the inverse Laplacian operator makes this problem strongly dependent on the BC of the system. Therefore, any method aiming at providing a solution to Eq. (6) has to deal with the BC, which, for instance, could be either periodic or free (otherwise referred to as "isolated" or "open") along each of the three directions x, y, z. In the case of fully periodic BC, the most natural (and efficient) approach to the problem

TABLE I. Libraries included in the ESL and ESL bundle. The first 12 are dedicated to electronic structure (ES) functionality; the last ten are tools of more general (GEN) applicability beyond electronic structure theory. They are described in Sec. IV, except for Futile, which is described in Sec. VI B 2. The licensed acronyms expand as follows: GPL: GNU General Public License, in versions 2.0[31] and 3.0;[32] LGPL: GNU Lesser GPL, version 3.0;[33] MPL: Mozilla Public License, version 2.0;[34] MIT: Massachusetts Institute of Technology License;[35] CeCILL-C: The CeCILL-C Free Software License Agreement;[36] BSD: Berkeley Software Distribution license, in either the 2-clause[37] or 3-clause[38] version.

Library	Functionality	License
PSolver	ES: Poisson solver for 0, 1, 2, and 3 dimensions, varying dielectrics and Poisson–Boltzmann	GPL-2.0
Libxc	ES: Pointwise evaluation of exchange and correlation for LDAs and GGAs	MPL-2.0
libvdwxc	ES: Evaluation of van der Waals non-local exchange and correlation	GPL-3.0
libGridXC	ES: Evaluation of exchange and correlation in regular grids incl. non-local van der Waals DFs	BSD 3-clause
pspio	ES: Input/output of pseudopotentials in most popular formats	LGPL-3.0
libPSML	ES: Standardized pseudopotential markup language specification and associated library	BSD 3-clause
ESCDF	ES: Electronic-structure data format specification and associated library	LGPL-3.0
ELSI	ES: Unified interface calling a variety of Hamiltonian solver libraries	BSD 3-clause
PEXSI	ES: Pole expansion and selective inversion solver library	BSD 3-clause
LibOMM	ES: Iterative minimization non-orthogonal solver	BSD 2-clause
PIKSS	ES: Parallel iterative Kohn–Sham solvers	GPL-3.0
`wannier90`	ES: Postprocessing to obtain maximally localized Wannier functions and derived quantities	GPL-2.0
ELPA	GEN: High-performance dense eigenvalue solver library	LGPL-3.0
NTPoly	GEN: Sparse linear-scaling solver library	MIT
SLEPc-SIPs	GEN: Shift-and-invert parallel slicing solver	BSD 2-clause
SuperLU_DIST	GEN: Sparse linear system solver	BSD 3-clause
Scotch	GEN: Graph partitioning library	CeCILL-C
MatrixSwitch	GEN: Matrix-format-independent abstraction layer of linear algebra operations	BSD 2-clause
flook	GEN: Connection between Fortran and Lua for embedded scripting code control	MPL-2.0
LibFDF	GEN: Flexible data format for the input of control parameters	BSD 3-clause
xmlf90	GEN: Fortran library to parse and write the well-formed XML files	BSD 2-clause
Futile	GEN: Low-level toolbox (handles YAML-code mapping, dynamic memory, timing, error, etc.)	GPL-3.0

is the reciprocal space treatment. It amounts to expanding both the density and the potential as superpositions of plane waves (PW) (Fourier series), thereby Eq. (6) becoming—for a homogeneous dielectric—algebraic in the Fourier components of ρ and V. This equation is readily solved, and the result is finally transformed back into real space. Forward and backward transformations are carried out via Fast Fourier Transforms (FFT); hence, the overall computational scaling of the method with respect to the number N of grid points is a rather appealing $\mathcal{O}(N \log N)$.

The situation is less straightforward for the same problem but different BC, e.g., free (isolated) BC. In this case, the solution of Poisson's equation in vacuum can formally be obtained from a three-dimensional integral,

$$V(\mathbf{r}) = \int d\mathbf{r}' G(|\mathbf{r} - \mathbf{r}'|)\rho(\mathbf{r}'), \tag{7}$$

where $G(r) = 1/r$ is the Green function of the Laplacian operator in the unconstrained $\mathbb{R}^3$ space. The long range nature of the kernel operator G does not allow us to approximate free BC with a very large periodic volume. Consequently, the description of non-periodic systems using a periodic formalism always introduces long-range interactions between supercells that compromise the results.

Due to the simplicity of plane wave methods, various attempts have been made to generalize the reciprocal space approach to free BC.[39–41] All of them use FFT at some point and thus have a $\mathcal{O}(N \log N)$ scaling. These methods use *ad hoc* screening functions to subtract the spurious interactions between super-cells. They have some restrictions and cannot be used blindly. For example, the method of Füsti-Molnar and Pulay[40] is only efficient for spherical geometries, and the method of Martyna and Tuckerman[41] requires artificially large simulation boxes that are computationally expensive. Nonetheless, the usefulness of reciprocal space methods has been demonstrated for a variety of applications, and plane-wave based approaches are widely used in the chemical physics community.

Two-dimensional periodic systems, such as surfaces, are another prominent choice of BC. The many surface-specific experimental techniques developed in recent years produce important results that can greatly benefit from theoretical interpretation and analysis. The development of efficient computational techniques for systems with such boundary conditions thus became very important. A number of explicit Poisson solvers have been developed in this framework[42–44] based on a reciprocal space treatment. Essentially, these Poisson solvers are constructed by implementing a suitable generalization for surface BC of the same methods that were developed for isolated systems. As for the free BC case, screening functions are applied to subtract the artificial interaction between the supercells in the non-periodic direction. Therefore, they exhibit the same kind of intrinsic limitations, e.g., good accuracy is only achieved inside the bulk of the computational region, with the consequent need for artificially large simulation boxes, which may increase the computational overhead.

Following these considerations, a series of efficient and accurate Poisson solvers have been developed that compatible with all possible combinations of mixed isolated/periodic boundary conditions. The solvers also support screened and unscreened Coulomb operators in vacuum[45–47] and distributed non-uniform dielectrics including the Poisson–Boltzmann equation.[48,49] In contrast to Poisson solvers based solely on a reciprocal space treatment, the fundamental operations of this Poisson solver are based on a mixed reciprocal-real space representation of the charge density. This allows different boundary conditions in different directions to be naturally satisfied. Screening functions or other approximations are thus not needed.

The basic advantage of this approach is that the real-space values of the potential $V(\mathbf{r})$ are obtained to very high accuracy on the uniform mesh of the simulation domain, via a direct solution of Poisson's equation by convolving the density with the appropriate Green's function of the Laplacian. As already mentioned, the Green's function can be discretized for the most common types of boundary conditions encountered in electronic structure calculations, namely, free, wire, slab, and periodic. This approach can therefore be straightforwardly used in all DFT codes that are able to express the densities $\rho(\mathbf{r})$ on uniform real-space grids. This is very common because the XC correlation potential is usually calculated on such a grid, at least in pseudopotential-based codes. This approach has also proved to be, in its parallel central processing unit (CPU) version, the fastest in most cases[50] and is therefore integrated in various DFT codes such as ABINIT,[21] CP2K,[51] Octopus,[23,52] and CONQUEST.[53]

To conclude, the Poisson solver algorithm has already been ported on Graphic Processing Units (GPU)[54] and is readily available in the ESL package. It enables affordable calculation of exact exchange operators in large systems.[55]

B. Libxc

The exchange–correlation functional is at the heart of density-functional theory,[29] and it is ultimately responsible for the accuracy of any such electronic structure calculation. It is, therefore, perhaps not surprising that hundreds of different approximations to this term have been proposed over the last few decades. Most of these can be classified into five families, usually often identified as different rungs of Jacob's ladder,[56] leading from the Hartree world to the Heaven of chemical accuracy. The rungs correspond to the local-density approximation (LDA), the generalized-gradient approximation, the meta-generalized-gradient approximation, functionals that depend on the occupied Kohn–Sham orbitals, and finally, functionals that also depend on the virtual orbitals. Libxc[18,19] is a library that contains the mathematical expressions for functionals belonging to the first three families, together with the semi-local parts for the functionals of the last two rungs.

Libxc has, by now, a long history, with its roots at the beginning of this century and version 1.0.0 appearing in 2010 (the current stable version is 4.3.4). The number of functionals included has increased steadily over the years with more than 500 functionals, arising from more than half a century of theoretical developments, implemented to date. Recently, the library was completely restructured to allow the definition of the functionals to be written in Maple,[57] which simplifies the insertion of new functionals (Maple's symbolic language is considerably simpler than C and well adapted for mathematical manipulations). Moreover, all derivatives are evaluated symbolically by Maple. This significantly reduces the possibility of errors in the implementation and opens the way for the evaluation of higher derivatives of the functionals.

Currently, Libxc supports up to fourth-derivatives, required, for example, for the calculation of Hessians of potential energy surfaces for excited-states.

There are a number of advantages of Libxc for the users of electronic structure codes. First, they have instant access to nearly all the exchange–correlation functionals ever developed. Furthermore, most functionals are implemented in Libxc shortly after their publication, giving access to the latest theoretical developments in density-functional theory often only requiring a simple recompilation of the library. Finally, it makes the comparison of different codes and methods much simpler. Libxc is by now used by more than 30 electronic structure codes, developed by the Physics communities (such as Abinit,[58] BigDFT,[25] FHI-aims,[26] and WIEN2k[59]), the Quantum Chemistry community (such as Psi4,[60] Orca,[61] PySCF,[62] or Turbomole[63]), commercially developed codes (QuantumATK[64]), as well as other libraries, e.g., libGridXC (see Sec. IV D). Libxc guarantees reliable, bug free implementations of the functionals, which are often cross-checked with the reference code from the original authors of the functionals. Finally, Libxc provides a simple means to perform benchmark calculations in a variety of physical systems and using diverse numerical methods (see, e.g., Ref. 65).

C. libvdwxc

libvdwxc[66] is a software library that evaluates the non-local correlation term for density functionals in the vdW-DF family[67,68] such as vdW-DF,[69] vdW-DF2,[70] and vdW-DF-cx.[71] It also implements the recent spin-generalization of these functionals.[72] It is written in C and released under the GNU GPL license.

Libxc evaluates functionals point-wise and hence supports only local and semi-local functionals. The purpose of libvdwxc is to complement Libxc by providing just the missing non-local term. libGridXC contains an alternative implementation (see Sec. IV D).

The vdW-DF functionals are the sum of three terms: the correlation energy from LDA; the exchange energy from a GGA functional, which is often chosen differently for different vdW functionals; and finally, the non-local vdW correlation energy, which is characteristic of the vdW-DF family. This latter term is an integral over a kernel function $\phi(\mathbf{r}, \mathbf{r}')$,

$$E_c^{\mathrm{nl}}[n] = \frac{1}{2} \iint n(\mathbf{r}) \phi(\mathbf{r}, \mathbf{r}') n(\mathbf{r}') \, \mathrm{d}\mathbf{r} \, \mathrm{d}\mathbf{r}'. \tag{8}$$

Direct integration of this expression scales as $\mathcal{O}(N^2)$ and is very expensive, so most codes use the spline interpolation method due to Román-Pérez and Soler.[73] This reduces the integral to a convolution in Fourier space whose computational cost is only $\mathcal{O}(N \log N)$.

The algorithm uses a number (conventionally 20) of helper functions,[73] $\theta_n(\mathbf{r})$, and their Fourier transforms. This still requires more memory and computation time than a standard GGA functional. libvdwxc focuses on parallel scalability in order that this computation will not become a bottleneck. It works in parallel using MPI with the Fourier transform library FFTW.[11,74] For parallel computations, the grids use the 1D block distribution of FFTW. libvdwxc additionally supports the PFFT library,[75] an extension to FFTW that improves scalability for massively parallel architectures.

libvdwxc takes the density and its gradient on a uniform 3D grid as an input. The grid directions need not be orthogonal. It calculates the total energy and its derivatives at each point, following Libxc conventions for ease of integration with DFT codes.

D. libGridXC

The libGridXC library[76] started life as SiestaXC, a collection of modules within Siesta to compute the exchange–correlation energy and potential in DFT calculations for atomic and periodic systems. The "grid" part of the name refers to the discretization for charge density and potential used in those calculations. The original code included a set of low-level routines to compute the exchange–correlation energy density and potential, $\epsilon_{xc}(\mathbf{r})$ and $V_{xc}(\mathbf{r})$, respectively, at a point for (semilocal) LDA and GGA functionals (i.e., a subset of the functionality now offered by Libxc), and two high-level routines to handle the computations in the whole domain (with radial or 3D-periodic grids), including computations of any gradients and integrations, needed. The most relevant feature of SiestaXC was its pioneering implementation of efficient and practical algorithms for van der Waals functionals,[73] in particular, for the evaluation of the non-local correlation term. These algorithms have found their way into numerous other implementations, as exemplified in Sec. IV C on libvdwxc. Another strength of the code is its support for non-collinear spin densities, as needed, in particular, for calculations with spin–orbit-coupling. Like libvdwxc, it inputs the density on a uniform grid, not necessarily orthogonal, and outputs the XC energy and potential on the same grid. In contrast with libvdwxc, the density gradient is evaluated internally.

The current libGridXC retains most of the SiestaXC functionality and enhances it by offering an interface to Libxc that supports a much wider selection of XC functionals. The code, written in Fortran, has been streamlined and re-packaged into a proper standalone library, with an automatic build-system. It is used by modern versions of Siesta, it is being adopted in ABINIT, and it is also being considered for BigDFT and other codes.

E. pspio

For a long time, the development of pseudopotentials has generally been coupled to a parent DFT code. This has resulted in a proliferation of file formats and incompatibilities, preventing or severely limiting collaboration involving different codes. Even worse, some versions of a pseudopotential format are not compatible with some versions of the DFT code they originated from. To address this issue, many discussions took place from 2002 on to define a common file format for pseudopotentials. While this led to the successful creation of the PAW-XML format for projector augmented-wave (PAW) datasets,[77] no agreement was reached at the time for norm-conserving pseudopotentials (see, however, Sec. IV F).

pspio takes exactly the reverse perspective: since many file formats exist and will continue to exist for the foreseeable future, let us design and implement a library that is able to read and write all of them, including the different versions of each format. Any pseudopotential generator or DFT code using pspio will thereby be free of file-format problems. However, pspio is not intended to act as a "universal translator," which would basically require the implementation of a pseudopotential generator within the library. Indeed, different file formats store different quantities, some of which have

to be reconstructed to convert one format to another. As a consequence, direct format conversion is only possible in a very limited number of cases.

pspio currently supports the `FHI98PP`, `ABINIT6`, and `UPF-1` file formats. Support for Siesta `PSF` and `ONCV` formats is currently being tested. It can be found in Ref. 78.

F. libPSML

Several well-known programs generate pseudopotentials in a variety of formats, tailored to the needs of specific electronic-structure codes. While some generators are now able to output data in different bespoke formats, and some simulation codes are now able to read different pseudopotential formats (with the help from pspio in Sec. IV E, for example), the common historical pattern in the design of those formats has been generator produced data for a single particular simulation code, most likely to be the one maintained by the same group. The consequence was often that a number of implicit assumptions, shared by generator and user, have entered into the formats and fossilized there.

This leads to practical problems, not only of programming, but also of interoperability and reproducibility, which depend on spelling out a large number of details that are not always well known or documented for all codes or existing formats.

PSML (for PSeudopotential Markup Language)[79,80] is a file format for norm-conserving pseudopotential data, which is designed to encapsulate, to the greatest extent possible, the abstract concepts in the domain's ontology and to provide appropriate metadata and provenance information. PSML files can be produced by the oncvpsp[30] and atom[81] pseudopotential generator programs and are a download-format option in the Pseudo-Dojo database of curated pseudopotentials.[82,83]

The software library libPSML[79,80] can be used by electronic structure codes to transparently extract the information in a PSML file and adapt it to their own data structures, or to create converters for other formats. It is currently used by Siesta and ABINIT, making full pseudopotential interoperability possible and thus facilitating comparisons of calculation results.

A feature of the PSML format and library is worth noting: the exchange–correlation flavor used in the generation of the pseudopotential is encoded in the PSML file as a set of Libxc "ids." It exemplifies the importance of software standards in scientific computing and their implementation in widely available libraries. Given the comprehensive support for functionals in Libxc, this is very close to a "universal" specification. The combination of libPSML and Libxc (with maybe libGridXC as an intermediate layer) is thus a basic ingredient for interoperability.

G. Electronic structure common data format (ESCDF)

The electronic structure common data format (ESCDF)[84] and the accompanying library libescdf[85] are currently being developed with the aim of simplifying a number of I/O related issues: (1) many codes deal with the same information, foremost structural data about the system of interest, which could easily be interchangeable between codes, and for which a common format would be useful; (2) having a common standard available would simplify workflow systems, chaining, e.g., *ab initio* calculations with post-processing spectroscopy calculations and data visualization; and (3) parallel I/O of large datasets for the general output or for code-specific restart files is becoming increasingly important and having a common tool to facilitate this at a low level would help many code developers. Over the last few decades, there have been several attempts to introduce such common standards in the ESC, with varying degrees of success. The main challenge is that much of the data are not actually interchangeable between codes, which are based on different computational methods. For instance, it is, in general, not meaningful to use wave functions generated in a plane wave pseudo-potential method in an all-electron LAPW code. ESCDF acknowledges that fact and does not try to impose a rigid standard but, rather, to provide lower level tools, which define a common vocabulary for writing data and to provide the necessary meta-data to clearly describe how the data in a given file are represented. This ambition for flexibility is further illustrated in the specification of structure, which allows for periodicity in any dimension (0–3), as used by the PSolver (Sec. IV A), and even beyond, as for non-periodic embedding in infinite or semi-infinite structures, as used by multiple-scattering methods.[86–88]

The ideas behind the ESCDF are, to a large extent, based on the ETSF-IO library and associated specifications,[89,90] which it tries to extend and modernize by moving from netCDF-4 to HDF-5[91] as the underlying technology. They also build on the wavefunction format of the BerkeleyGW code,[92] which was defined as both a specification and a library of reading and writing routines, used by Octopus and other DFT codes in preparing inputs for *GW* and Bethe–Salpeter calculations. The development has been driven by a collaboration of ETSF developers and new developers, in particular, from the EUSpec[93] network, which had the specific goal of providing tools for chaining calculations and post-processing tools. The ESCDF specifications have also been aligned as much as possible with the existing specifications from the NOMAD project[94] and have also informed NOMAD about their new extensions. The ESCDF specifications are developed and maintained by a dedicated curating team (CT).

One of the core features of the implementation of libescdf is the separation of the format specification and the library code. The specifications are defined in a JSON file, which can easily be extended without the need to change the code in the library. The specific code for the library is then generated automatically from the JSON file, and the format documentation is also auto-generated from this central specifications file. This strategy will make the library more maintainable and effectively decouples the science from the underlying software design.

Currently, both the specifications and the library are still under development, with several sections of the former already complete. As soon as it is possible, libescdf will be interfaced with the ESL Demonstrator project and included in the ESL Bundle (see Sec. V).

H. ELSI and supported solver libraries: ELPA, PEXSI, NTPoly, SLEPc-SIPs, SuperLU-DIST, Scotch

This group of libraries solves or circumvents the Kohn–Sham or generalized Kohn–Sham eigenproblem, i.e., the central problem of electronic structure calculations. They can be used in conjunction with the open-source ELSI library (ELectronic Structure Infrastructure, https://elsi-interchange.org), but the associated solvers can be

and are also used in a standalone fashion with different electronic structure packages. ELSI provides a unified software interface that connects electronic structure codes to various high-performance solver libraries to solve or circumvent eigenproblems encountered in electronic structure theory.[95] In addition to providing interfaces, matrix conversion, etc., ELSI also abstracts common tasks in handling eigenvalue problems in an electronic structure code. The tasks handled by ELSI and related solvers often amount to the most compute-intensive ones in electronic structure codes. These ESL components therefore already offer support for several past and present pre-exascale hardware developments, notably Intel's many-core architectures as well as NVidia's GPUs. Solvers currently supported in ELSI include conventional dense eigensolvers (ELPA,[96,97] EigenExa,[98] LAPACK,[9] and MAGMA[99]), the orbital minimization method (LibOMM[100]), sparse iterative eigensolvers (SLEPc[101] and SLEPc-SIPs), the pole expansion and selected inversion method (PEXSI[102]), and linear scaling density matrix purification methods (NTPoly[103]). As sketched in Fig. 2, an electronic structure code interfacing to ELSI automatically has access to all the eigensolvers and density matrix solvers supported in ELSI. In addition, the ELSI interface is able to convert arbitrarily distributed dense and sparse matrices to the specification expected by the solvers, taking this burden away from the electronic structure code. A comprehensive review of the capabilities of the latest version of ELSI, including parallel solution of problems found in spin-polarized systems (two spin channels) and periodic systems (multiple ***k***-points), scalable matrix I/O, density matrix extrapolation, and iterative eigensolvers in a reverse communication interface (RCI) framework, is presented in a separate publication.[104]

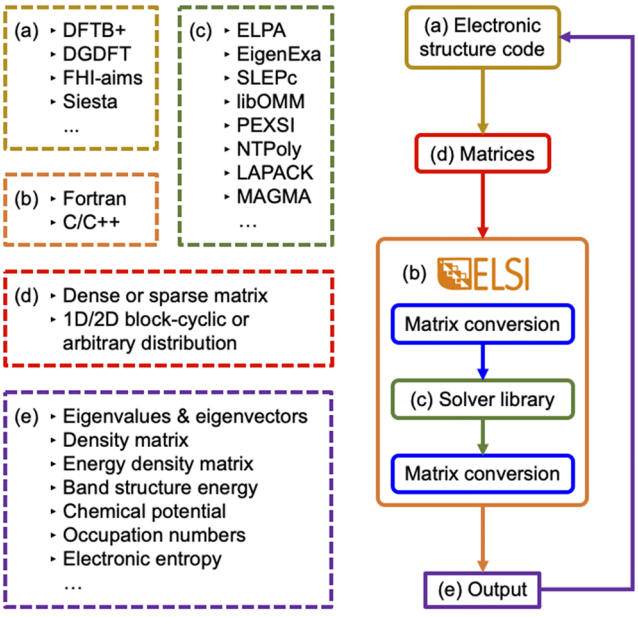

FIG. 2. Interaction of the ELSI interface with electronic structure codes. An electronic structure code has access to various eigensolvers and density matrix solvers via the ELSI API. Whenever necessary, ELSI handles the conversion between different units, conventions, matrix formats, and programming languages. (a) lists the electronic structure codes that currently use ELSI. (b)–(e) list the programming language, solvers, matrix formats, and output quantities, respectively, supported by ELSI.

The development of ELSI, including its API design, internal data structure, build system, testing, and integration with electronic structure codes, was driven from its inception by contributions and feedback from the community. In workshops organized by the ESL, ELSI has been a primary focus from the outset. Moreover, developers and users of several electronic structure codes participate in open ELSI monthly video meetings to exchange ideas, ensuring a direct information flow in order to develop ELSI as a software package that fits the needs of as many electronic structure projects as possible. To date, the ELSI interface has been adopted by the DFTB+,[105] DGDFT,[106] FHI-aims,[26] and SIESTA[22] codes. To aid in the selection of the solver that is best suited for a particular application problem, ELSI provides a series of benchmarks to assess the performance of the solvers for different problem types and on different computer architectures.[95,104] This benchmark effort has been greatly accelerated by a separate FortJSON library, shipped with ELSI, which enables the output of runtime parameters, matrix dimensions, timing statistics, etc., into a standard JSON file. Thanks to the popularity and portability of JSON, ELSI log files written by FortJSON can be easily processed and analyzed by the existing tools. Comparing different solvers in different codes on an equal footing is thus significantly simplified by the ELSI infrastructure.

ELSI ships with its own tested versions of several individual solver libraries (which are also included in the ESL) but, additionally, linking against already compiled upstream versions from each solver library is supported as much as possible. The installation of the different components is handled by a single CMake-based build system that either compiles the redistributed source code of the solvers or links ELSI against user-supplied solver libraries.

I. LibOMM non-orthogonal eigensolver

Now integrated into the larger bundle of eigensolvers provided by ELSI, LibOMM was developed during the first ESL workshop as a standalone library and can still be used as such. It is written in Fortran with C bindings and can be compiled either for serial or MPI parallel operation.

The orbital minimization method (OMM) is an iterative solver method based on finding the set of Wannier functions describing the occupied subspace by the minimization of a specially defined energy functional. The peculiarity of the OMM is that only an unconstrained minimization is required, thus avoiding a potentially expensive orthogonalization step; the properties of the functional drive the Wannier functions toward orthonormality as it is minimized. The OMM has an interesting history (discussed briefly in Ref. 100) stemming from research on linear-scaling DFT methods. The LibOMM library, however, is based on a later re-implementation of the method used in SIESTA as an efficient cubic-scaling solver for the basis of finite-range numerical atomic orbitals.[100] As such, the library provides a tensorial correction for non-orthogonal basis sets, either via a Cholesky factorization of the overlap matrix or a preconditioner suitable for localized orbitals.

The library is built for maximum efficiency in data reuse; the API is designed for the repeated calls within an outer self-consistency loop in the host code. Data are reused between calls in

two ways: (a) some matrices, such as the coefficient matrix of the Wannier functions, are repeatedly passed in at each call, updated during the call and passed out again, and (b) other data are allocated and stored internally by the library, and therefore, a final call to free all memory must be performed.

The library is also written to be agnostic with respect to both the data storage scheme of the matrices and the implementation of the matrix operations. This is achieved by making use of an underlying library for matrix operations, MatrixSwitch, described in Sec. IV L.

J. PIKSS: Parallel iterative Kohn–Sham solvers

In addition to ELSI and its supported solvers (Secs. IV H and IV I), the parallel iterative Kohn–Sham (KS) solvers library PIKSS is a bundle of several iterative diagonalization eigensolvers that have been extracted from the Quantum ESPRESSO (QE) suite and recast in an independent, code-agnostic fashion. It includes the popular Davidson diagonalization and band-by-band conjugate gradient minimization methods but also implements the more recently developed Projected Preconditioned Conjugate Gradient (PPCG)[107] and Parallel Orbital (ParO) update solvers[108] that allow new parallelization paradigms.

As ideal within the ESL concept, the library is designed such that the interaction with the main electronic structure code is via routine library calls with a well-specified API. The operations performed by the library depend on the chosen diagonalization method but generally include the application of the Hamiltonian to a set of candidate wavefunctions, computation of the overlap matrix, approximation of inverse matrices, etc.

Unlike the original, strictly plane-wave implementation in QE, the KS-Solvers library allows for any internal representation of the wavefunction and Hamiltonian. To further exemplify how to expand the usability of the solvers, a reverse communication interface version for one of the solvers (Davidson diagonalization) is also provided. The library is currently hosted in Ref. 109.

Initially integrated within KS-Solvers but currently being developed as a stand-alone library, the miniPWPP module serves as a demonstrator for the KS-Solver library. miniPWPP, a barebone DFT implementation based on a planewave-empirical pseudopotential framework, demonstrates the usage of the several methods within KS-Solvers library. It allows performance comparison of them on different hardware platforms (e.g., CPU and GPU), and with different parallelization paradigms (e.g., over bands and task groups). Both the KS-Solvers and miniPWPP, along with other libraries (i.e., FFTXlib and LAXlib) that originated from Quantum ESPRESSO suite and tailored for plane wave basis, will be inserted in the ESL bundle in future releases.

K. `wannier90`

Wannier functions (WFs)[110,111] provide a localized real-space representation of the electronic structure of materials that is complementary to the reciprocal-space representation of Bloch bands. The freedom associated with the choice of gauge of Bloch states can be used to construct exponentially localized WFs.[112] The so-called *maximally localized Wannier functions* (MLWFs)[14] are obtained by choosing the gauge that minimizes the total quadratic spread of the WFs. Both the case of isolated bands,[14] i.e., a composite set of bands that is separated from other bands in the Brillouin zone (BZ) by energy gaps, and the case of entangled bands[113] can be treated.

MLWFs are used routinely, for example, to analyze and understand chemical bonding, to perform high-accuracy fine-grained interpolation of quantities in the BZ (such as band energies, Berry phase properties, and electron–phonon interactions), to characterize topological materials, to construct compact tight-binding models of materials, and to compute charge transport properties. We direct the reader to Refs. 111 and 114 for details of the underlying theory of MLWFs and their diverse applications. Instead, here, we focus on (1) the aspects of the `wannier90` code[13,114,115] and its development that have enabled it to emerge as a paradigmatic example of an interoperable software tool and (2) future plans that will take the code further in directions that reflect the broader philosophy of the ESL.

From its conception,[116] `wannier90` was designed to make the addition of new functionality as easy as possible, by being modular, well-documented and well-commented, and to be as independent as possible from the underlying code that calculates the Bloch bands from which the MLWFs are constructed. As such, `wannier90` requires only the matrix elements $M_{mn}^{(\mathbf{k},\mathbf{b})} = \langle u_{m\mathbf{k}} | u_{n\mathbf{k}+\mathbf{b}} \rangle$, where $u_{n\mathbf{k}}(\mathbf{r})$ is the cell-periodic part of the Bloch function $\psi_{n\mathbf{k}}(\mathbf{r}) = u_{n\mathbf{k}}(\mathbf{r}) e^{i\mathbf{k}\cdot\mathbf{r}}$, together with an initial guess for the choice of gauge. The latter can be obtained either by projecting an appropriate set of atomic-like orbitals $g_n(\mathbf{r})$ onto the initial Bloch states or by using the recently implemented "selected-columns of the density matrix" (SCDM) method,[117–119] that does not demand the human intervention often needed to define good projection functions.

Since these matrix elements, together with the eigenvalues of the single-particle Hamiltonian, are independent of the specific implementational details of the underlying electronic structure code (e.g., basis set, grids, symmetry operations, level of theory, and pseudopotentials), `wannier90` is fully interoperable with any code that is able to calculate them. The onus is largely on the developers of electronic structure codes, therefore, to develop and maintain their own interface that provides these quantities. `wannier90` allows for two interface modes: (1) via reading and writing files to/from disk and running `wannier90` as a separate external executable and (2) via calls to the `wannier90` library directly from within a program. Electronic structure codes that interface to `wannier90` include Quantum ESPRESSO,[24] ABINIT,[58] VASP,[120] SIESTA,[22] WIEN2k,[59] Fleur,[121] Octopus,[23,52] ELK,[122] BigDFT,[25] GPAW,[27] pyscf,[62] and openmx.[123] New developments in `wannier90`, therefore, are available to the vast majority of the user community rapidly, which serves us to accelerate research.

The most recent major release (v3.x) of `wannier90` is able to compute a growing range of properties,[114] a range that is increasingly difficult to maintain in one code with a small group of developers. Furthermore, there is a growing community of researchers and codes, including Gollum,[124] WannierTools,[125] NanoTCAD ViDES,[126] Yambo,[127] Z2Pack,[128] Triqs,[129] and EPW[130] that use `wannier90` to calculate an even wider range of properties. For these reasons, in 2016, ten years after its first release,[116] `wannier90` transitioned to a community-development model in which the code is hosted on GitHub,[115] and community-driven developments are invited via a fork and pull-request approach. Code integrity is maintained via a documented coding style guide that contributors must adhere to, together with nightly automated building and testing on a Buildbot[131] test farm, and continuous integration with Travis CI,[132]

whereby a pull-request triggers a suite of test calculations and is blocked if any tests fail.

What does the future hold for `wannier90`? Currently, only a small subset of the full functionality of the code is accessible in the library mode of `wannier90`, and only in serial processing. The next major planned development, therefore, is to completely re-engineer the library mode of the code such that the full functionality of `wannier90` (including parallel processing) is accessible via library calls from within, e.g., an overarching workflow, a dynamical simulation, or a self-consistent field iteration. This would enable advanced materials properties to be calculated seamlessly *on the fly*. Using the code in this way is made significantly more practical due to recent developments in generating MLWFs automatically with minimal user-intervention.[117] Some challenges will need to be overcome, however, including: determining the optimal strategy for parallelization given that this is likely to conflict with that of the host electronic structure code and handling errors thread-safely yet unobtrusively. When this development is complete, like the ouroboros that swallows its own tail,[133] we envisage the main `wannier90` code becoming a wrapper for its own library calls.

L. MatrixSwitch

The MatrixSwitch library was developed alongside LibOMM during the first ESL workshop but is independent of it. Its aim is to act as an intermediary layer between high-level routines for physics-related algorithms and low-level routines dealing with matrix storage and manipulation, allowing the former to be written in a way that is close to mathematical notation while also enabling seamless switching between different matrix storage formats and implementations of the matrix operations. As new formats are introduced in MatrixSwitch, they can immediately be used in the high-level routines without any further modification of the code. Both dense and sparse formats are supported, as well as serial and parallel distributions.

At the center of MatrixSwitch is the `matrix` object, a Fortran derived type defined by the library, which acts as a wrapper for the specific storage format. A small number of basic operations are defined for this object, such as setting and getting elements, matrix–matrix multiplication, matrix addition, and traces. The API is also easily extensible to include more complex matrix operations that are not part of the standard set, as the object is quite transparent and can be unpacked when needed to operate directly on the underlying data structures.

An additional feature of the library is that it facilitates its usage only for a subset of the host code (e.g., in a specific module), by permitting the pre-existing matrix data conforming to one of the supported storage formats to be simply registered by MatrixSwitch without the need for copying, converting, or allocating new memory. The registered matrix can then be used for any MatrixSwitch operation as if it were natively managed.

As well as being used by LibOMM, MatrixSwitch is also currently being used in parts of Siesta (e.g., for the recently developed real-time time-dependent DFT algorithm) and in smaller codes used for individual research projects.[134] MatrixSwitch has recently been extended[135] to use the DBCSR library[136,137] (Distributed Block Compressed Sparse Row), a linear-algebra parallel engine for sparse matrices. The original Siesta linear-scaling solver based on OMM with finite support solutions[138] is being refactored to use MatrixSwitch and thus take the advantage of the DBCSR backend. This strategy will be extended to other related solvers.[139]

M. flook

The flook library was developed with the objective to control flow and move lightweight operations into the scriptable code. It provides a simple way to code top steering tools [see Fig. 1(b)] such as molecular dynamics (MD) methods, which are called only once every MD step and are typically light operations, but it also allows deeper-level control of the code, such as the tuning of mixing parameters or stopping calculations when certain criteria are met, for example.

The scripting can be efficiently implemented by embedding a Lua interpreter into the application program. Lua is a lightweight embeddable scripting language.[140] The software library flook enables Fortran and Lua to communicate together in a seamless way by passing variables to and from "tables" in Lua. Having a hook between Lua and Fortran empowers end-users to create their own scripts in Lua in order to extend the functionality of codes. Since any data can be moved between Fortran and Lua, the Lua script that implements a particular functionality can be used to replace that functionality inside the core Fortran code, if so desired.

This methodology works by assigning Lua functions to be called in Fortran. By populating a table with the internal data-structure in Fortran using dictionaries, one can easily enable variable passing between Fortran and Lua using a single line of code. As an example, here is a snippet of Fortran and Lua code, which enables access to the atomic coordinates in the Siesta DFT package.

(a) ***Fortran code:***

```
type(dictionary_t): variables
! Add atomic coordinates to table of variables
variables = variables//('geom.xa'.kvp.xa)
```

(b) ***Lua code:***

```
- Retrieve atomic coordinates and manipulate
local xa = siesta.geom.xa
```

Currently flook is used in Siesta to extend molecular dynamics methods and customize outputs, force-constant calculations, and convergence of precision parameters. It also exposes convergence variables, which allows the user to change these parameters while a calculation is running. A derived project [flos (https://github.com/siesta-project/flos)] implements the scriptable Lua functions that may be used for other projects using flook.

N. LibFDF

FDF stands for Flexible Data Format, designed within the Siesta project to simplify the handling of input options. It is based on a keyword/value paradigm (including physical units when relevant) and is supplemented by a block interface for arbitrarily complex blobs of data.

LibFDF[141] is the official implementation of the FDF specifications for use in client codes. At present, the FDF is used extensively by Siesta, and it has been an inspiration for several other code-specific input formats.

New input options can be implemented very easily. When a keyword is not present in the FDF file, the corresponding program variable is assigned a pre-programmed default value. This enables programmers of client codes to insert new input statements anywhere in the code, without worrying about "reserving a slot" in a possibly already crowded fixed-format input file.

O. xmlf90

xmlf90 is a package to handle XML in Fortran. It has two major components: (*i*) A XML parsing library, with the most complete programming interface based on the very successful SAX (Simple API for XML) model,[142] although a partial DOM interface and a very experimental XPATH interface are also present. The SAX parser, in particular, was designed to be a useful tool in the extraction and analysis of data in the context of scientific computing, and thus, the priorities were efficiency and the ability to deal with potentially large XML files while maintaining a small memory footprint. (*ii*) A library (xmlf90-wxml) that facilitates the writing of well-formed XML, including features such as automatic start-tag completion, attribute pretty-printing, and element indentation. There are also helper routines to handle the output of numerical arrays. xmlf90 is the parsing engine for the libPSML library of Sec. IV F.

V. THE ESL BUNDLE

Adopting a modular approach has many advantages: smaller software units to develop and maintain, easier testing of each component, faster propagation of fixes, better separation of technical domains, and reduced duplication of code, to cite a few obvious ones. However, it also implies some risks: if not addressed explicitly, the asynchronous evolution of the individual components—also known as modules—quickly becomes a severe obstacle to the improvement and maintenance of the whole. Libxc is an emblematic example of this kind of situation: whenever a new exchange–correlation functional is implemented in Libxc, a few dozen DFT codes are just one compilation away from using it. However, if the API of Libxc changes, each DFT code has to update its interface to Libxc to be able to use the new version, which results in a situation where some codes use one version of the library, while others are stuck with an older version.

When one module depends on another, or a number of others, a few aspects have to be considered with great care:

- A given version of the dependent module is compatible with only some versions of those it is dependent on.
- Even when the two versions are compatible, not all configuration options available will actually work, i.e., the two modules may have conflicting requirements in some situations.
- In addition to technical aspects, social considerations have to be taken into account, in particular, when the two modules are developed by different teams.

To mitigate these risks, we provide all the ESL software libraries in the form of a bundle. The ESL Bundle provides a set of ready-to-use software modules such that to each version of the ESL Bundle, there corresponds a well-defined set of module versions that are compatible among themselves. The contents of the ESL Bundle are curated through the activities of the ESL and supervised by the ESL Steering Committee (for more details about the governing structure of the ESL, see Appendix A). Adding, updating, or removing modules is discussed during ESL workshops and Steering Committee meetings until an agreement is reached, before being thoroughly tested to detect possible compatibility issues. The validation of any change within the ESL Bundle sometimes involves a high level of complexity, which is why it is performed by a team of volunteers and includes manual steps. Indeed, the ESL Bundle is meant to be used in production by ESC codes, not just to be successfully installed on a given set of systems. What will finally decide whether a module can be updated will be the usability of its new version by these ESC codes, which is why complementary tests should be conducted with the codes themselves before releasing a new version of the ESL Bundle. Figure 3 summarizes the dependencies between individual modules, which are actively monitored in collaboration with their respective developers. Changes and improvements brought by the ESL are reported to the original developers of the affected modules and contributed back to the upstream module whenever possible.

However, providing a bundle by itself is not enough. To ensure its usability, the ESL Bundle must be easy to compile and install on different platforms and by users with different needs and goals. With so many different components, written in different programming languages and using different build systems, this is far from being a trivial task. This is why the ESL Bundle needs to be distributed in different forms, each targeting a different use case. We describe two

FIG. 3. Internal dependencies between the components of the ESL Bundle. White background: components extracted from electronic-structure codes participating in the ESL. Blue background: components created within the ESL or through its activities. Orange background: components maintained outside the ESL. The complete tree of solvers accessed by ELSI appears in Fig. 2.

of these distribution channels in greater detail in Subsections V A and V B. A third distribution channel currently under consideration is to provide the ESL Bundle as Debian and RPM packages. Several components, such as Libxc, are already included in the official repositories of several popular Linux distributions, such as Debian or Fedora, as well as in the MacPorts package manager for macOS, but we would like to extend this to the whole ESL Bundle. A fourth channel of distribution for the bundle is the collection of docker images released publicly on Docker Hub, at https://hub.docker.com/u/eslib. After each release, a new docker image is built using JHBuild scripts, tagged, and used to test the ESL Demonstrator. These images can be handy for quick access to a binary distribution of the bundle, of benefit to both developers and curious users.

A. JHBuild bundler

To provide the ESL Bundle in a fully self-contained way with a common installation interface for all of its components, we use the JHBuild framework.[143] JHBuild is an actively maintained Python build framework used by the GNOME project,[144] an open-source desktop environment for Unix-like operating systems, which has been solving the same challenges as the ESL over the last two decades. JHBuild is able to build a collection of modules, that it names *modulesets*, from a minimal amount of information: download URL, type of build system, and one-to-one dependencies, plus optional on-the-fly corrections (patches). JHBuild determines the correct order of compilation of the modules by itself and strictly separates the aspects related to the modulesets from those belonging to the build environment. The latter is achieved by using configuration files to tune the build parameters, globally or for each module. In the case of the ESL Bundle, we provide a curated collection of such configuration files for Linux-based systems and macOS. The use of JHBuild greatly simplifies the installation of the ESL Bundle and is ideal for developers or users of electronic structure codes that require one or more components to be installed on their personal computers but do not care too much about performance.

B. HPC-oriented distribution

Once we consider software provisioning for HPC resources, where software such as the ESL Bundle should leverage the available hardware and seamlessly integrate into the existing software stack, the situation becomes vastly more challenging. In this context, the ESL is far from alone in the depth and complexity of its software stack. Application developers, HPC sites, and end users around the world spend significant amounts of time creating and verifying optimized software installations for such resources. Although the problems that arise with installing scientific software are ubiquitous, there is currently inadequate collaboration between HPC sites and/or HPC domains. At the "*Getting Scientific Software Installed*" Birds-of-a-Feather session at SC'19 less than 30% of the survey respondents answered "yes" when asked whether they work together with other HPC sites on software installation.

EasyBuild[145] is a tool for providing optimized, reproducible, multi-platform scientific software installations in a consistent, efficient, and user-friendly manner. EasyBuild is currently used by well over 100 HPC sites worldwide (including Jülich Supercomputing Centre, CSCS, Compute Canada, SURFsara, and SNIC). Leveraging EasyBuild for HPC-oriented distribution provides the ESL with an HPC-oriented build infrastructure that can quickly and reliably distribute the bundle to a large number of HPC sites.

EasyBuild employs the so-called compiler toolchains, or simply *toolchains* for short, which are a major facilitator in handling the build and installation processes. A typical toolchain consists of one or more compilers, usually put together with some libraries, which provide specific functionality, e.g., for using an MPI stack for distributed computing, or which provide optimized routines for commonly used math operations, e.g., the well-known BLAS/LAPACK APIs for linear algebra routines. For each software package being built, the toolchain to be used must be specified. Notably, EasyBuild already supports over 1800 software packages, including many of the (direct and indirect) dependencies of the ESL Bundle. These verified and consistent infrastructures allow ESL development efforts to focus primarily on its component libraries, which can be synchronized with the EasyBuild toolchain release cycle (which currently has two updates per year). This is why EasyBuild replaces JHBuild for the distribution of the ESL Bundle on HPC environments.

VI. USE CASES IN END USER CODES

Figure 4 illustrates the usage of ESL packages by the electronic structure programs engaged in the ESL project. There are more cases of usage not covered here, namely, other electronic structure codes that use some of the libraries described above. As stated previously, ESL collects both libraries that have been built or extracted from codes purposely for ESL, together with independently developed and maintained libraries, e.g., Libxc and `wannier90`, which predate ESL, and whose authors agree with (and contribute to) their incorporation into the ESL. The libraries in Fig. 4 are the ones that are (or are being) included in the ESL Bundle described in Sec. V.

The lines in Fig. 4 show dependencies between components of the ESL and user codes. The red lines indicate dependencies on libraries (depicted as ovals) that have been extracted as independent library components of the ESL from the connected user code (the rectangles). Of course, the original codes use them as well. The blue lines show which user codes use components that were independently developed in the ESL. Some of the libraries, although not extracted from any given code, were developed by code developers of a particular code. However, such connections are not indicated in this figure. In the following, we describe the links and ESL usage illustrated in Fig. 4 for the codes shown, starting with the ESL Demonstrator, a very lean electronic structure code created from scratch in a couple of weeks and built on the ESL.

A. ESL Demonstrator

Four years after the ESL was initiated, the ESL team realized it had gathered sufficient libraries to account for nearly all the complex parts of a simple DFT code. In February 2018, the fifth ESL workshop was focused on building an entire DFT code from scratch, within a fortnight. The purpose of such a demonstrator code is to demonstrate the usage of ESL libraries and to provide a framework to test the ESL Bundle. It is not in any way intended to be a competitor to the existing DFT codes. Instead, it can be seen as being part of the ESL documentation, guiding new users and developers of the ESL. As such, some effort has been made to make it clear, simple, and easily extendable.

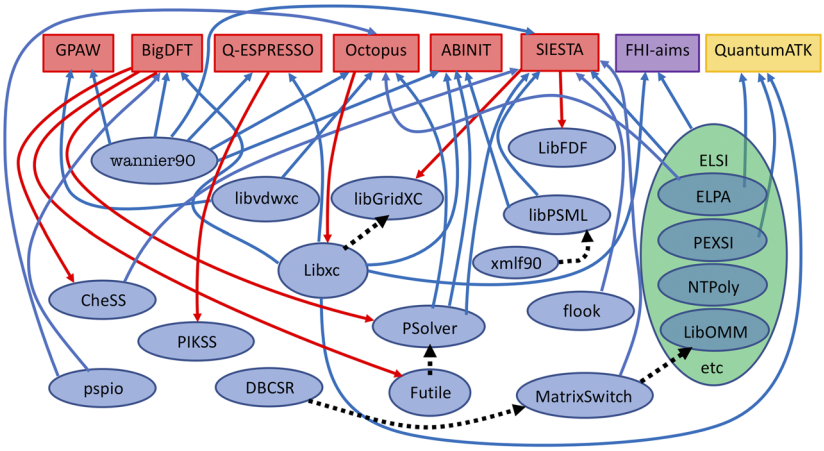

FIG. 4. Use of ESL libraries within electronic structure codes. The rectangular boxes at the top show electronic structure programs (end users), red indicating open-source programs (licensed via GPL), while FHI-aims is distributed by a registered non-profit organization (Molecular Simulations from First Principles e.V., MS1P, https://ms1p.org/) under a proprietary license, and QuantumATK is under commercial license distributed by the software company Synopsys. This set contains codes connected to the ESL, mostly via contributors to the ESL being developers of these codes (there are many other codes that use at least some of the depicted libraries). Blue ellipses indicate libraries described in this paper. The larger (green) ellipse corresponds to ELSI as a general interface to several Hamiltonian solvers, with its associated libraries. Arrows indicate dependencies. Thick black dashed lines indicate libraries dependent on other libraries, blue lines show libraries directly used by the codes, and red lines indicate libraries that were re-engineered by extracting them from a particular code (and are also used by that code). DBCSR[136,137] is included here because it has been coupled[135] to MatrixSwitch as a parallel sparse linear-algebra engine. ELPA is also used by ABINIT, QuantumESPRESSO, and GPAW.

The resulting code, the ESL Demonstrator, is a functional DFT code, which makes extensive use of the ESL libraries presented in Fig. 5. It uses pspio for reading pseudopotentials, PSolver to calculate the Hartree potential, LibFDF as the input engine, libGridXC to calculate the XC potential on a grid, Libxc to evaluate the XC functional on the points of that grid, ELSI for calculating eigenstates, and flook to make scriptable control flows. During the development, it was also decided to follow the Sphinx documentation style to retain a unified documentation scheme.

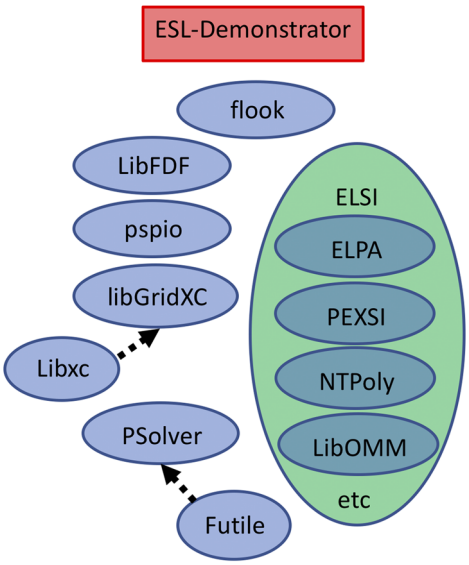

FIG. 5. Libraries used in the full DFT program developed as a demonstrator for the ESL. It allows the user to choose between plane-waves and atomic orbitals as basis sets. The libraries themselves are as in Fig. 4, and as described in this section.

The development of the demonstrator code was carefully divided between teams formed from the 14 people who attended the workshop coding session. The tasks were assigned to suit the individual expertize within each team while also taking into account the backbone code of the demonstrator. In particular, there are several parts of a DFT program required: (1) user input, (2) Hartree potential, (3) XC potential, (4) eigenstate solver, and (5) scriptable work-flows. While most codes use either a plane-wave or a localized orbital basis, the ESL Demonstrator allows users to use either of the two. Such a decision makes the code slightly more complicated, but it allows an increase in the range of libraries used and provides newcomers to DFT codes an easy access point to two classes of basis sets that require very different numerical methods.

The ESL Demonstrator successfully uses the aforementioned libraries. It currently only allows non-spin-polarized, Γ-point calculations, and serial execution. Some of the missing features are due to shortcomings in the ESL Bundle. For example, it currently contains no libraries to generate **k**-point grids, which restricts the ESL Demonstrator to Γ-point only calculations. Other missing features are not due to the underlying libraries but simply reflect the early stage of development of the ESL Demonstrator. Our plan is to keep extending the ESL Demonstrator to cover more features provided by the existing ESL libraries and by new libraries added to ESL Bundle in the meantime. Work is underway on allowing parallel execution and spin-polarized calculations.

An important effect of developing the ESL Demonstrator is the exposure of possible bugs, missing features, and testing how integrable and inter-operable libraries actually are. Indeed, the development of the demonstrator led to the discovery of certain bugs and build problems in ESL libraries. The ESL Demonstrator acts a *de facto* test for the ESL Bundle where a successful build and run of the demo are a prerequisite to release a new bundle. The code is hosted at https://gitlab.com/ElectronicStructureLibrary/esl-demo.

B. ESL in participating codes

1. ABINIT

ABINIT pioneered the open-source model within the electronic-structure community. It is a plane-wave-based code that has allowed contributions from quite an open community of developers, some of them coding new features, tools, etc., in modules being integrated into the code. In this sense, it can claim to have taken the first intellectual step within the electronic-structure community that led to the ESL. In addition to this contribution to the ESL concept, ABINIT benefits from the usage of Libxc for the local and semi-local exchange–correlation terms, and it now incorporates libGridXC on top of Libxc for the global grid treatment on non-local functionals. It also makes use of PSolver for the computation of the Hartree terms to energy and potential, and the PSML standard and associated library for the pseudopotential input.

2. BigDFT

For several years already, the monolithic sources of BigDFT have been divided into several subdirectories, that slowly became independent from each other and were finally separated into their own modules, living in a separate Git tree and that are shipped with their own build system. This direction of work was seen by the developer team as a way to keep the development sustainable in terms of functionalities and maintenance. It started with a library implementing a tool box for Fortran, Futile that is available in the ESL. This tool box started with the in-memory representation of a YAML document[146] but was quickly extended to keep track of dynamic memory allocations, time measurements, error handling, etc. The versatile Poisson solver used in BigDFT is now completely independent from its origins and also available in the ESL. Some other components of BigDFT available in the ESL were developed from the start as separated libraries, such as the sparse matrix library CheSS.[147] BigDFT is also taking advantage of codes developed outside the project. Like numerous other DFT codes, it is using Libxc for the exchange and correlation calculation. An interface exists to post-process the calculated wave functions using `wannier90`. While initially coded for the Hutter–Goedecker–Hartwigsen pseudopotential formalism,[148] a link with pspio was written to allow a greater range of pseudopotentials. Finally, BigDFT is distributed as a bundle, such as the ESL. It takes care of the compilation and linking of the various libraries and the end project itself to deliver a single executable to the end user.

3. FHI-aims

FHI-aims developers have greatly contributed to the ESL through joint involvement in the broader, U.S. NSF-funded ELSI project, described in Sec. IV H. ELSI was inspired by the ESL and represented an early U.S. initiative in this effort, in an otherwise primarily European endeavor. Through ELSI, FHI-aims now benefits from a range of Hamiltonian solvers in a seamless framework. This list includes ELPA, which originated within FHI-aims and is maintained as a standalone solver library led by the Max-Planck Computing and Data Facility, as well as PEXSI, NTPoly, and all other solvers supported by ELSI. The libraries used by FHI-aims are not restricted to solvers, as Libxc is supported for the calculation of the exchange–correlation contribution for local and semi-local functionals.

4. GPAW

The GPAW code allows for different modes of operation, according to the way Kohn–Sham wavefunctions are represented.[27,149] They are based on, namely, finite differences (FD), plane waves (PW), or linear combination of atomic orbitals (LCAO). The LCAO mode uses ELPA for fast parallel diagonalization of the Hamiltonian matrix. GPAW also uses Libxc plus libvdwxc to support LDA, GGA, meta-GGA, and vdW-DF exchange–correlation functionals.

5. Multiple scattering codes

Codes built around the computation of the Kohn–Sham Green's function, by means of multiple scattering (MS) theory, give immediate access to spectroscopic properties, transport, and many other response functions. In addition, these methods can deal with many different problems in electronic structure theory for systems with and without periodicity, such as disordered alloys and semi-infinite surfaces.[87] Multiple-scattering codes typically import data such as self-consistent Kohn–Sham potentials or charge densities from other ES codes. The set of ESCDF format specifications and the associated library is ideal for that purpose. It is being already used by data transfers between codes such as the Munich SPR-KKR[87,88] and MSSpec.[86]

6. Octopus

Octopus is currently interfaced to several libraries that are part of the ESL Bundle. The Libxc library, although it is now completely independent, was originally developed within Octopus. When treating finite systems in Octopus, the default method to solve Poisson's equation is the one provided by PSolver. The evaluation of exchange and correlation functionals that depend explicitly on the density is done exclusively using the Libxc and libvdwxc libraries. Support for reading pseudopotentials using the pspio library is also provided, as well as the possibility of using `wannier90` to compute maximally localized Wannier functions from the Bloch states. Finally, the ELPA library can be used whenever direct diagonalization of matrices is required.

7. QuantumATK

QuantumATK is a commercially developed platform that includes its own LCAO and plane-wave DFT solvers as well as semi-empirical tight-binding and force-fields. The code is the closed-source, but makes use of several external software libraries; among these are three libraries in the ESL Bundle: Libxc, ELPA, and PEXSI (the last two included independently of ELSI). ELPA and PEXSI are

used not only for the LCAO-DFT solver but also for the various semi-empirical tight-binding solvers.

QuantumATK is a unique case among the list of current codes using ESL libraries for a number of reasons: (a) its closed-source and commercial nature mean that there are strict constraints on the licensing of libraries it can use (the most common being MIT, BSD, and LGPL); (b) it uses C++ as its backend language (with a Python frontend), and the ESL libraries are therefore linked to C++ rather than Fortran or C; and (c) executables are compiled and shipped for Windows as well as Linux.

8. Quantum ESPRESSO

This plane-wave program distribution contains a variety of optimized iterative Hamiltonian solvers tailored for the plane-wave basis. Within the ESL effort, they were extracted and isolated into the PIKKS KS-Solvers suite and library, together with additional components to perform fast-Fourier transforms (FFTXlib) and parallel linear algebra operations (LAXlib). These will be inserted in the ESL bundle in future releases. The use of these components is demonstrated in a simple empirical pseudopotential code that can be used as a tool for further developments.[109] Quantum ESPRESSO codes link to these libraries, as well as to other ESL libraries of different origins, such as Libxc, ELPA, and `wannier90`.

9. SIESTA

Two libraries now in the ESL (libGridXC and LibFDF) originated as modules within Siesta. Several more (flook, xmlf90, and libPSML) were developed with general usefulness in mind but also to address the issues of relevance to that program. Siesta is thus an important contributor to the ESL. In the opposite direction, Siesta benefits from the other ESL-provided functionality, most clearly in the area of solvers, with an interface to ELSI that has significantly extended the choices available and enhanced the performance of the code. The Libxc library is also used through the interface to libGridXC. `wannier90` has also been fully incorporated as a library. Work is now being done to incorporate the new functionality available in the PSolver library in siesta, and there are plans to benefit from some of the low-level utilities in the Futile package.

VII. FUTURE

The ESL represents a channel for possible spontaneous utility projects to develop and link into present and new electronic structure packages. In this sense, the future evolution of the ESL from the point of view of the sub-packages it contains is quite open.

For the mid- and long-term future of the ESL, a key metric of success will be wide usage. In addition to communication (as done in this paper and on the web), the following aspects will be important. (i) Content—useful features. High-level programmers should be able to find in the ESL key tools for their programs. (ii) Performance. The libraries will need to be maintained, keeping up with hardware evolution, and maintaining competitive standards of efficiency and scalability. An important component of future performance will be the definition and stabilization of APIs, in addition to good standards of documentation. (iii) Easy use. The library has to be user friendly, not necessarily for the end user, but for computational-scientist coders, who will implement new codes and/or features linking to the ESL. (iv) Easy build. End users of programs that link to the ESL should be able to compile their codes reasonably easily.

Concerning content, there is a list of candidate libraries to include, as well as modules in present programs that can be extracted as libraries. In the short term, there are packages (some of them mentioned above) that are being prepared for inclusion into the ESL and its bundle. This is the case for the CheSS library,[147] which implements a linear-scaling Hamiltonian solver based on a Fermi-operator expansion. It arises from the BigDFT program, but it already works as a separate library and is already used by other codes, such as Siesta. Similarly, the connection between MatrixSwitch and DBCSR, illustrated in Fig. 4, will soon be bundled into the ESL. In addition, a candidate for bundling into the ESL is the libPAW library,[150] currently distributed in the ABINIT package, but also used in BigDFT and other codes. It is a collection of objects and routines intended to facilitate the porting of the projector augmented-wave method "out of the box" onto any ES code, regardless of the basis used for the wave functions. DFTB+[105,151] developers are also joining the ESL effort contributing their semi-empirical electronic structure engine and the stand-alone SAYDX library (Structured Array Data Exchange),[152] which is an auxiliary library that provides a platform for exchanging array data using a simple tree structure. It offers a framework to build, manipulate, and query such array data trees, as well as send and receive them through various transport layers. They will be incorporated into future ESL bundles.

Although the ESL and this paper focus on electrons, an important line of future work is the incorporation of upper-level steering packages and libraries, prominently molecular-dynamics engines, and, more generally, codes dealing with the nuclear degrees of freedom, both classically and quantum-mechanically. Large-scale first-principles condensed-matter and molecular simulations are extremely versatile, but most of their applicability demands an efficient treatment of both electrons and nuclei, which will benefit from (*i*) improved robust communication between nuclear-dynamics drivers and electronic-structure engines on varied platforms, and (*ii*) hierarchical parallelization of the integrated code, to allow very large scale simulations on massively parallel computers. Library solutions for both problems and, especially, their integration, represents a promising direction for the community, building on initiatives such as i-PI.[16] It is a line of work that would involve the core of the CECAM community, not only the electronic side, representing a great opportunity for the future of condensed-matter and molecular simulations.

VIII. CONCLUSIONS

The electronic structure library project presented here is an initiative to stimulate, coordinate, and amplify the efforts in library sharing already started within the electronic structure community. It was initiated by CECAM, which continues its support together with the E-CAM European Centre of Excellence, spearheading a push within the community for a better model of electronic structure software development, which, it is hoped, will enhance dynamism, versatility, maintainability and optimization of electronic structure codes. It will rationalize coding effort by avoiding useless

repetition and by separating different types of coding tasks to be carried out by people with suitable profiles and backgrounds, distinguishing between computational scientists and computer scientists or software engineers. We believe that it will allow the re-engineering efforts needed for the deployment of electronic codes on novel computer architectures to be carried out more efficiently, widely, and by professionals close to hardware companies and HPC centers.

Importantly, it is a community effort, pushed by people involved in the development of very prominent and popular electronic structure codes, representing a wide spectrum of the community. Most of the library packages presented in this paper were extracted from those codes, and many of these are currently being used by codes other than their parent codes. There has been an emphasis on library packages for highly parallel heavy-duty tasks, the sharing of which is more challenging, but very important for the ambitions of the ESL.

In addition to extracting, generating, and documenting the library packages and adapting their APIs for general use, part of the ESL effort is dedicated to facing the new challenges arising with the model, most prominently, the integration of units with different data structures and parallelization, and the bundling of the set of packages in the ESL library for consistent and automatic building and compiling.

Finally, as a community effort, the ESL community welcomes new additions to the ESL, and, of course, the use of the ESL or its components by any electronic structure programmer, or indeed any other community, as well as user feedback.

AUTHORS' CONTRIBUTIONS

All authors have contributed to this paper by coding and organizing the coding events. M.J.T.O., N.P., Y.P., and V.B. have contributed to the ESL by coordinating the project at coding events and in between them; M.J.T.O., N.P., and Y.P. have maintained the software infrastructure. Most authors have contributed to the writing of this paper, either of particular sections or by revising it. M.J.T.O. and E.A. have coordinated the writing.

ACKNOWLEDGMENTS

The authors would like to thank CECAM for launching and pushing the ESL, as well as hosting part of its infrastructure, and partly funding the extended workshops where most of the coding was done, both in the Lausanne headquarters and in the Dublin, Trieste, and Zaragoza nodes. Within CECAM, the authors particularly thank Sara Bonella, Bogdan Nichita, and Ignacio Pagonabarraga. The authors also acknowledge all the people who have supported and contributed to the ESL in different ways, including Luis Agapito, Xavier Andrade, Balint Aradi, Emanuele Bosoni, Lori A. Burns, Christian Carbogno, Ivan Carnimeo, Abel Carreras Conill, Alberto Castro, Michele Ceriotti, Anoop Chandran, Wibe de Jong, Pietro Delugas, Thierry Deutsch, Hubert Ebert, Aleksandr Fonari, Luca Ghiringhelli, Paolo Giannozzi, Matteo Giantomassi, Judit Gimenez, Ivan Girotto, Xavier Gonze, Benjamin Hourahine, Jürg Hutter, Thomas Keal, Jan Kloppenburg, Hyungjun Lee, Liang Liang, Lin Lin, Jianfeng Lu, Nicola Marzari, Donal MacKernan, Layla Martin-Samos, Paolo Medeiros, Fawzi Mohamed, Jens Jørgen Mortensen, Sebastian Ohlmann, David O'Regan, Charles Patterson, Etienne Plésiat, Markus Rampp, Laura Ratcliff, Stefano Sanvito, Paul Saxe, Matthias Scheffler, Didier Sebilleau, Søren Smidstrup, James Spencer, Atsushi Togo, Joost Vandevondele, Matthieu Verstraete, and Brian Wylie.

The authors would also like to thank the Psi-k network for having partially funded several of the ESL workshops. A.O., E.A., D.L.-D., S.G., E.K., A.A.M., and M.C.P. received funding from the European Union's Horizon 2020 research and innovation program under Grant Agreement No. 676531 (Centre of Excellence project E-CAM). The same project has partly funded the extended software development workshops in which most of the ESL coding effort has happened. A.G., S.M., and E.A. acknowledge support from the European Union's Horizon 2020 research and innovation program under Grant Agreement No. 824143 (Centre of Excellence project MaX). M.A.L.M. acknowledges partial support from the DFG through Project No. MA-6786/1. D.G.A.S. was supported by the U.S. National Science Foundation (NSF) (Grant No. ACI-1547580). M.C.P. acknowledges support from the EPSRC under Grant No. EP/P034616/1. A.A.M. acknowledges support from the Thomas Young Centre under Grant No. TYC-101, the Wannier Developers Group, and all of the authors and contributors of the `wannier90` code (see Ref. 115 for a complete list). A.M.E. acknowledges support from CoSeC, the Computational Science Centre for Research Communities, through CCP5: The Computer Simulation of Condensed Phases (EPSRC Grant Nos. EP/M022617/1 and EP/P022308/1). A.G. and J.M.S. acknowledge Spain's Ministry of Science (Grant No. PGC2018-096955-B-C42). E.A., A.G., and J.M.S. acknowledge Spain's Ministry of Science (Grant No. FIS2015-64886-C5). Y.P., D.L.-D., and E.A. acknowledge support from the Spanish MINECO and EU Structural Investment Funds (Grant No. RTC-2016-5681-7). M.L. acknowledges support from the EPRSC under Grant No. EP/M022668/1. M.L., M.J.T.O., and Y.P. acknowledge support from the EU COST action (Grant No. MP1306). J.M. was supported by the European Regional Development Fund (ERDF), project CEDAMNF (Reg. No. CZ.02.1.01/0.0/0.0/15-003/0000358). V.W.-Z.Y., W.P.H., Y.L., and V.B. acknowledge support from the National Science Foundation under Award No. ACI-1450280 (the ELSI project). V.W.-Z.Y. also acknowledges a MolSSI fellowship (NSF Award No. ACI-1547580). Simune Atomistics S.L. is thanked for allowing A.H.L. and Y.P. to contribute to the ESL, as is Synopsys, Inc., for the partial availability of F.C.

APPENDIX A: COMMUNITY ORGANIZATION AND STEERING STRUCTURE OF THE ESL

Formally, the ESL project was kick-started at a workshop organized at CECAM-HQ by Artacho, Payne, and Tildesley. Hosting around 20 participants, the workshop was held during the summer of 2014 over a period of six weeks. After extensive discussions, the objectives and scope of the library were agreed, and the basic infrastructure was put in place. At this point, the key element was the ESL wiki containing information about the existing libraries and modules. Also at this time, a governance structure was put into place consisting in a Curating Team (CT) and a Scientific Advisory Board.

Since then, more workshops have been organized, roughly one per year, where the ESL has been changed, improved, and expanded. It has evolved from a repository of information about software libraries and tools in the domain of the electronic structure to a curated bundle of tightly integrated software libraries. As the project evolved and mutated, its governance adapted to better serve its objectives. In 2019, the Advisory Board was replaced by a Steering Committee (SC). The SC proposes and defines the guidelines that the CT should follow. There are quarterly meetings, which are open to the public and focus on at least three tasks: (i) deciding which new libraries should be added to the ESL Bundle, (ii) proposing which versions of the existing ESL software should be shipped, and (iii) discussions of topics for coming workshops. The SC aims to include as many developers of software included in the ESL Bundle and from codes using it as possible. It currently has 12 members, and all CT members are part of the SC as well. More recently, the curating team has been expanded from three to six members. The CT manages everyday activities within the ESL by holding monthly meetings, creating proposal drafts, and communicating with code developers and the SC. Each member of the CT is tasked with supervising one specific aspect of the ESL. These include, amongst others, bundle maintenance, organization of the ESL workshop, the ESL website, and ESL documentation. Note that there are no competing interests between CT and SC. The ESL initiative aims to hold at least one workshop a year. These have a duration of 14 days of which 2 days are for discussions and the remaining 12 days are for hands-on development activities. The focus of the workshops shifts each year with the topic decided by the SC.

APPENDIX B: SUSTAINABILITY AND SOFTWARE ENGINEERING OF THE ESL DEMONSTRATOR

Continuous Integration (CI) is a software engineering practice that allows code integration from multiple contributors automatically into the main repository of a project. The process is enabled by a set of tools and stages that assert the correctness of the code at each change. We strongly believe that CI is a critical requirement for any scientific software project in order to maintain a sufficient level of quality over time.

CI is used within the ESL in a systematic way for the development of the ESL Bundle and the ESL Demonstrator, as well as to check that the ESL Demonstrator can keep relying on new versions of the ESL Bundle. The ESL Demonstrator is a basic example of the ESC code built exclusively with ESL components to explain to developers how to use them in their own codes (see Sec. VI A). As such, it has to be permanently kept in a working state. Some of the individual components of the ESL also benefit from CI, upon choice of their respective developers.

As an example, the ESL Demonstrator relies on a series of widespread tools and technologies: Gitlab CI,[153] CTest from CMake,[154] YAML,[146] and Docker/Docker Hub.[155] All these ingredients are glued together to provide development workflows implemented in the ESL (see Fig. 6). After each commit, the CI infrastructure automatically checks that the code successfully builds and the corresponding tests pass.

Docker is one of the tools designed to help with running and deployment of applications by using container technology. It is relatively straightforward to use, and we deploy the entire ESL Bundle on it. We offer three Linux flavors for our Docker images: Ubuntu, Fedora, and OpenSUSE. These Docker images are the ones we use in the build and testing stage. The images are public and distributed via Docker Hub.

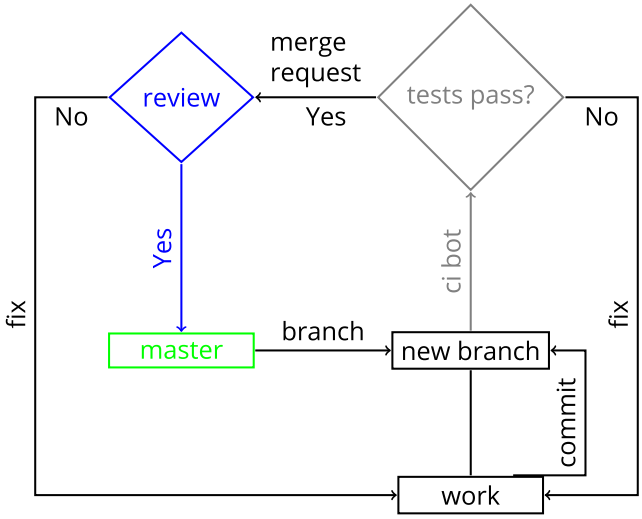

FIG. 6. Continuous integration workflow integrated with git.

The EasyBuild framework is of great help in this context. It has support for generating Singularity and Docker container recipes, which will use EasyBuild to build and install reference software stacks. The latter will then be used within the CI infrastructure of the ESL, which mostly uses container-based runners in the cloud.

Gitlab CI is the integrated Gitlab tool for continuous integration, continuous delivery, and deployment, and is highly configurable. At the moment, we are using it only for CI.

CTest is a testing tool distributed as part of CMake. It integrates seamlessly with CMake, which is our build system of choice, and one can easily use it to run unit tests or regression tests. We choose to use the latter due to the lack of any maintained unit testing framework for Fortran. A test is deemed passed or failed based on matching a regular expression at the end of execution. We also defined a target that monitors the code coverage of our tests. In order to help with the automation of testing, the output of the ESL Demonstrator is YAML-compliant, helping us to easily check the output. For checking the output, we rely on a simplified version of Siesta's YAML output testing.

In the ESL workflow (see Fig. 6), the user starts from a validated version of the *master* repository by branching their own branch, a *user branch*. Once the work envisaged is done and the user commits the changes to the branch, the Gitlab CI automatically runs the designed tests. If the tests fail, the user corrects the errors and commits again. Once the tests pass, the code is ready for a *merge request*. A merge request is issued by the user for inclusion in the master branch of the project. A *peer review process* then kicks in. Two reviewers have to agree for the code to be included in the master branch. If issues are found, the code is returned to the user to fix the issues. If both reviewers agree, then the code is integrated.

DATA AVAILABILITY

Data sharing is not applicable to this article as no new data were created or analyzed in this study. Repositories for all ESL software packages mentioned in this work have been properly cited and are publicly available.

REFERENCES

[1] P. Mavropoulos and P. Dederichs, Statistical data about density functional calculations, Ψ_k Scientific Highlight of the Month No. 135, https://psi-k.net/download/highlights/Highlight_135.pdf, April 2017.

[2] P. A. M. Dirac, "Quantum mechanics of many-electron systems," Proc. R. Soc. London, Ser. A **123**, 714–733 (1929).

[3] N. Mardirossian and M. Head-Gordon, "Thirty years of density functional theory in computational chemistry: An overview and extensive assessment of 200 density functionals," Mol. Phys. **115**, 2315–2372 (2017).

[4] See, https://molssi.org/software-search/ for the Molecular Sciences Software Institute (MolSSI), 2016.

[5] N. Wilkins-Diehr and T. D. Crawford, "NSF's inaugural software institutes: The science gateways community institute and the molecular sciences software institute," Comput. Sci. Eng. **20**, 26–38 (2018).

[6] A. Krylov, T. L. Windus, T. Barnes, E. Marin-Rimoldi, J. A. Nash, B. Pritchard, D. G. A. Smith, D. Altarawy, P. Saxe, C. Clementi, T. D. Crawford, R. J. Harrison, S. Jha, V. S. Pande, and T. Head-Gordon, "Perspective: Computational chemistry software and its advancement as illustrated through three grand challenge cases for molecular science," J. Chem. Phys. **149**, 180901 (2018).

[7] See http://www.netlib.org/blas/blast-forum for BLAS technical forum, 1979.

[8] See https://www.mpi-forum.org for EMPI forum, since 1991.

[9] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide* (SIAM, 1999).

[10] See http://www.netlib.org/scalapack for ScaLAPACK, since 1992.

[11] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," Proc. IEEE **93**, 216–231 (2005), special issue on Program Generation, Optimization, and Platform Adaptation.

[12] A. Togo and I. Tanaka, "Spglib: A software library for crystal symmetry search," arXiv:1808.01590 (2018).

[13] See https://www.wannier.org/ for the `wannier90` code.

[14] N. Marzari and D. Vanderbilt, "Maximally localized generalized Wannier functions for composite energy bands," Phys. Rev. B **56**, 12847–12865 (1997).

[15] A. H. Larsen, J. J. Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dułak, J. Friis, M. N. Groves, B. Hammer, C. Hargus, E. D. Hermes, P. C. Jennings, P. B. Jensen, J. Kermode, J. R. Kitchin, E. L. Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J. B. Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K. S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, and K. W. Jacobsen, "The atomic simulation environment—A python library for working with atoms," J. Phys.: Condens. Matter **29**, 273002 (2017).

[16] 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, "i-PI 2.0: A universal force engine for advanced molecular simulations," Comput. Phys. Commun. **236**, 214–223 (2019).

[17] G. Pizzi, A. Cepellotti, R. Sabatini, N. Marzari, and B. Kozinsky, "AiiDA: Automated interactive infrastructure and database for computational science," Comput. Mater. Sci. **111**, 218–230 (2016).

[18] M. A. L. Marques, M. J. T. Oliveira, and T. Burnus, "Libxc: A library of exchange and correlation functionals for density functional theory," Comput. Phys. Commun. **183**, 2272–2281 (2012).

[19] S. Lehtola, C. Steigemann, M. J. Oliveira, and M. A. Marques, "Recent developments in libxc—A comprehensive library of functionals for density functional theory," SoftwareX **7**, 1–5 (2018).

[20] ESL: The Electronic Structure Library, Main site: http://esl.cecam.org, Repository: https://github.com/ElectronicStructureLibrary and in https://gitlab.com/ElectronicStructureLibrary.

[21] X. Gonze, B. Amadon, P. Anglade, J. Beuken, F. Bottin, P. Boulanger, F. Bruneval, D. Caliste, R. Caracas, M. Côté, T. Deutsch, L. Genovese, P. Ghosez, M. Giantomassi, S. Goedecker, D. Hamann, P. Hermet, F. Jollet, G. Jomard, S. Leroux, M. Mancini, S. Mazevet, M. Oliveira, G. Onida, Y. Pouillon, T. Rangel, G. Rignanese, D. Sangalli, R. Shaltaf, M. Torrent, M. Verstraete, G. Zerah, and J. Zwanziger, "ABINIT: First-principles approach to material and nanosystem properties," Comput. Phys. Commun. **180**, 2582–2615 (2009).

[22] J. M. Soler, E. Artacho, J. D. Gale, A. García, J. Junquera, P. Ordejón, and D. Sánchez-Portal, "The SIESTA method for *ab initio* order-*N* materials simulation," J. Phys.: Condens. Matter **14**, 2745–2779 (2002).

[23] N. Tancogne-Dejean, M. J. T. Oliveira, X. Andrade, H. Appel, C. H. Borca, G. Le Breton, F. Buchholz, A. Castro, S. Corni, A. A. Correa, U. De Giovannini, A. Delgado, F. G. Eich, J. Flick, G. Gil, A. Gomez, N. Helbig, H. Hübener, R. Jestädt, J. Jornet-Somoza, A. H. Larsen, I. V. Lebedeva, M. Lüders, M. A. L. Marques, S. T. Ohlmann, S. Pipolo, M. Rampp, C. A. Rozzi, D. A. Strubbe, S. A. Sato, C. Schäfer, I. Theophilou, A. Welden, and A. Rubio, "Octopus, a computational framework for exploring light-driven phenomena and quantum dynamics in extended and finite systems," J. Chem. Phys. **152**, 124119 (2020).

[24] P. Giannozzi, O. Andreussi, T. Brumme, O. Bunau, M. Buongiorno Nardelli, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, M. Cococcioni, N. Colonna, I. Carnimeo, A. Dal Corso, S. de Gironcoli, P. Delugas, R. A. DiStasio, A. Ferretti, A. Floris, G. Fratesi, G. Fugallo, R. Gebauer, U. Gerstmann, F. Giustino, T. Gorni, J. Jia, M. Kawamura, H.-Y. Ko, A. Kokalj, E. Küçükbenli, M. Lazzeri, M. Marsili, N. Marzari, F. Mauri, N. L. Nguyen, H.-V. Nguyen, A. Otero-de-la-Roza, L. Paulatto, S. Poncé, D. Rocca, R. Sabatini, B. Santra, M. Schlipf, A. P. Seitsonen, A. Smogunov, I. Timrov, T. Thonhauser, P. Umari, N. Vast, X. Wu, and S. Baroni, "Advanced capabilities for materials modelling with quantum ESPRESSO," J. Phys.: Condens. Matter **29**, 465901 (2017).

[25] L. Genovese, A. Neelov, S. Goedecker, T. Deutsch, S. A. Ghasemi, A. Willand, D. Caliste, O. Zilberberg, M. Rayson, A. Bergman, and R. Schneider, "Daubechies wavelets as a basis set for density functional pseudopotential calculations," J. Chem. Phys. **129**, 014109 (2008).

[26] V. Blum, R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, K. Reuter, and M. Scheffler, "*Ab initio* molecular simulations with numeric atom-centered orbitals," Comput. Phys. Commun. **180**, 2175–2196 (2009).

[27] J. J. Mortensen, L. B. Hansen, and K. W. Jacobsen, "Real-space grid implementation of the projector augmented wave method," Phys. Rev. B **71**, 035109 (2005).

[28] D. M. Ritchie, The development of the C language, https://www.bell-labs.com/usr/dmr/www/chist.html, 1993.

[29] R. M. Martin, *Electronic Structure: Basic Theory and Practical Methods* (Cambridge University Press, Cambridge, 2004).

[30] D. R. Hamann, "Optimized norm-conserving Vanderbilt pseudopotentials," Phys. Rev. B **88**, 085117 (2013).

[31] See https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html for the GNU General Public License, version 2.0.

[32] See https://www.gnu.org/licenses/gpl-3.0.en.html for the GNU General Public License, version 3.0.

[33] See https://www.gnu.org/licenses/lgpl-3.0.en.html for the GNU Lesser General Public License, version 3.0.

[34] See https://www.mozilla.org/MPL/2.0 for Mozilla Public License version 2.0.

[35] See https://opensource.org/licenses/MIT for the MIT license.

[36] See https://spdx.org/licenses/CECILL-C.html for the CeCILL-C Free Software License Agreement.

[37] See https://opensource.org/licenses/BSD-2-Clause for 2-clause BSD license.

[38] See https://opensource.org/licenses/BSD-3-Clause for 3-clause BSD license.

[39] R. W. Hockney, "Potential calculation and some applications," Methods Comput. Phys. **9**, 135–211 (1970).

[40] L. Füsti-Molnar and P. Pulay, "Accurate molecular integrals and energies using combined plane wave and Gaussian basis sets in molecular electronic structure theory," J. Chem. Phys. **116**, 7795–7805 (2002).

[41] G. J. Martyna and M. E. Tuckerman, "A reciprocal space based method for treating long range interactions in *ab initio* and force-field-based calculations in clusters," J. Chem. Phys. **110**, 2810–2821 (1999).
[42] P. Mináry, M. E. Tuckerman, K. A. Pihakari, and G. J. Martyna, "A new reciprocal space based treatment of long range interactions on surfaces," J. Chem. Phys. **116**, 5351–5362 (2002).
[43] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles* (McGraw-Hill, 1981).
[44] J. J. Mortensen and M. Parrinello, "A density functional theory study of a silica-supported zirconium monohydride catalyst for depolymerization of polyethylene," J. Phys. Chem. B **104**, 2901–2907 (2000).
[45] L. Genovese, T. Deutsch, A. Neelov, S. Goedecker, and G. Beylkin, "Efficient solution of Poisson's equation with free boundary conditions," J. Chem. Phys. **125**, 074105 (2006).
[46] L. Genovese, T. Deutsch, and S. Goedecker, "Efficient and accurate three-dimensional Poisson solver for surface problems," J. Chem. Phys. **127**, 054704 (2007).
[47] A. Cerioni, L. Genovese, A. Mirone, and V. A. Sole, "Efficient and accurate solver of the three-dimensional screened and unscreened Poisson's equation with generic boundary conditions," J. Chem. Phys. **137**, 134108 (2012).
[48] G. Fisicaro, L. Genovese, O. Andreussi, N. Marzari, and S. Goedecker, "A generalized Poisson and Poisson-Boltzmann solver for electrostatic environments," J. Chem. Phys. **144**, 014103 (2016).
[49] G. Fisicaro, L. Genovese, O. Andreussi, S. Mandal, N. N. Nair, N. Marzari, and S. Goedecker, "Soft-sphere continuum solvation in electronic-structure calculations," J. Chem. Theory Comput. **13**, 3829–3845 (2017).
[50] P. García-Risueño, J. Alberdi-Rodriguez, M. J. Oliveira, X. Andrade, M. Pippig, J. Muguerza, A. Arruabarrena, and A. Rubio, "A survey of the parallel performance and accuracy of Poisson solvers for electronic structure calculations," J. Comput. Chem. **35**, 427–444 (2014).
[51] J. Hutter, M. Iannuzzi, F. Schiffmann, and J. VandeVondele, "cp2k: Atomistic simulations of condensed matter systems," Wiley Interdiscip. Rev.: Comput. Mol. Sci. **4**, 15–25 (2014).
[52] X. Andrade, J. Alberdi-Rodriguez, D. A. Strubbe, M. J. T. Oliveira, F. Nogueira, A. Castro, J. Muguerza, A. Arruabarrena, S. G. Louie, A. Aspuru-Guzik, A. Rubio, and M. A. L. Marques, "Time-dependent density-functional theory in massively parallel computer architectures: The octopus project," J. Phys.: Condens. Matter **24**, 233202 (2012).
[53] M. J. Gillan, D. R. Bowler, A. S. Torralba, and T. Miyazaki, "Order-*N* first-principles calculations with the CONQUEST code," Comput. Phys. Commun. **177**, 14–18 (2007).
[54] N. Dugan, L. Genovese, and S. Goedecker, "A customized 3D GPU Poisson solver for free boundary conditions," Comput. Phys. Commun. **184**, 1815–1820 (2013).
[55] L. E. Ratcliff, A. Degomme, J. A. Flores-Livas, S. Goedecker, and L. Genovese, "Affordable and accurate large-scale hybrid-functional calculations on GPU-accelerated supercomputers," J. Phys.: Condens. Matter **30**, 095901 (2018).
[56] J. P. Perdew, "Jacob's ladder of density functional approximations for the exchange-correlation energy," AIP Conf. Proc. **577**, 1–20 (2001).
[57] See https://maplesoft.com for Maple, Maplesoft, a division of Waterloo Maple Inc., Waterloo, Ontario.
[58] X. Gonze, F. Jollet, F. Abreu Araujo, D. Adams, B. Amadon, T. Applencourt, C. Audouze, J.-M. Beuken, J. Bieder, A. Bokhanchuk, E. Bousquet, F. Bruneval, D. Caliste, M. Côté, F. Dahm, F. Da Pieve, M. Delaveau, M. Di Gennaro, B. Dorado, C. Espejo, G. Geneste, L. Genovese, A. Gerossier, M. Giantomassi, Y. Gillet, D. R. Hamann, L. He, G. Jomard, J. Laflamme Janssen, S. Le Roux, A. Levitt, A. Lherbier, F. Liu, I. Lukačević, A. Martin, C. Martins, M. J. Oliveira, S. Poncé, Y. Pouillon, T. Rangel, G.-M. Rignanese, A. H. Romero, B. Rousseau, O. Rubel, A. A. Shukri, M. Stankovski, M. Torrent, M. J. Van Setten, B. Van Troeye, M. J. Verstraete, D. Waroquiers, J. Wiktor, B. Xu, A. Zhou, and J. W. Zwanziger, "Recent developments in the ABINIT software package," Comput. Phys. Commun. **205**, 106–131 (2016).
[59] P. Blaha, K. Schwarz, G. K. Madsen, D. Kvasnicka, J. Luitz, R. Laskowski, F. Tran, and L. D. Marks, *WIEN2k: An Augmented Plane Wave Plus Local Orbitals Program for Calculating Crystal Properties* (Technische Universitat, 2019), ISBN: 3-950103112.
[60] R. M. Parrish, L. A. Burns, D. G. A. Smith, A. C. Simmonett, A. E. DePrince, E. G. Hohenstein, U. Bozkaya, A. Y. Sokolov, R. Di Remigio, R. M. Richard, J. F. Gonthier, A. M. James, H. R. McAlexander, A. Kumar, M. Saitow, X. Wang, B. P. Pritchard, P. Verma, H. F. Schaefer, K. Patkowski, R. A. King, E. F. Valeev, F. A. Evangelista, J. M. Turney, T. D. Crawford, and C. D. Sherrill, "Psi4 1.1: An open-source electronic structure program emphasizing automation, advanced libraries, and interoperability," J. Chem. Theory Comput. **13**, 3185–3197 (2017).
[61] F. Neese, "The ORCA program system," Wiley Interdiscip. Rev.: Comput. Mol. Sci. **2**, 73–78 (2012).
[62] Q. Sun, T. C. Berkelbach, N. S. Blunt, G. H. Booth, S. Guo, Z. Li, J. Liu, J. D. McClain, E. R. Sayfutyarova, S. Sharma, S. Wouters, and G. K. L. Chan, "PySCF: The python-based simulations of chemistry framework," Wiley Interdiscip. Rev.: Comput. Mol. Sci. **8**, e1340 (2018).
[63] R. Ahlrichs, M. Bär, M. Häser, H. Horn, and C. Kölmel, "Electronic structure calculations on workstation computers: The program system turbomole," Chem. Phys. Lett. **162**, 165–169 (1989).
[64] S. Smidstrup, T. Markussen, P. Vancraeyveld, J. Wellendorff, J. Schneider, T. Gunst, B. Verstichel, D. Stradi, P. A. Khomyakov, U. G. Vej-Hansen, M.-E. Lee, S. T. Chill, F. Rasmussen, G. Penazzi, F. Corsetti, A. Ojanperä, K. Jensen, M. L. N. Palsgaard, U. Martinez, A. Blom, M. Brandbyge, and K. Stokbro, "QuantumATK: An integrated platform of electronic and atomic-scale modelling tools," J. Phys.: Condens. Matter **32**, 015901 (2019).
[65] P. Borlido, T. Aull, A. W. Huran, F. Tran, M. A. L. Marques, and S. Botti, "Large-scale benchmark of exchange–correlation functionals for the determination of electronic band gaps of solids," J. Chem. Theory Comput. **15**, 5069–5079 (2019).
[66] A. H. Larsen, M. Kuisma, J. Löfgren, Y. Pouillon, P. Erhart, and P. Hyldgaard, "libvdwxc: A library for exchange–correlation functionals in the vdW-DF family," Modell. Simul. Mater. Sci. Eng. **25**, 065004 (2017).
[67] K. Berland, V. R. Cooper, K. Lee, E. Schröder, T. Thonhauser, P. Hyldgaard, and B. I. Lundqvist, "van der Waals forces in density functional theory: A review of the vdW-DF method," Rep. Prog. Phys. **78**, 066501 (2015).
[68] P. Hyldgaard, K. Berland, and E. Schröder, "Interpretation of van der Waals density functionals," Phys. Rev. B **90**, 075148 (2014).
[69] D. C. Langreth, M. Dion, H. Rydberg, E. Schröder, P. Hyldgaard, and B. I. Lundqvist, "van der Waals density functional theory with applications," Int. J. Quantum Chem. **101**, 599–610 (2005).
[70] D. C. Langreth, B. I. Lundqvist, S. D. Chakarova-Käck, V. R. Cooper, M. Dion, P. Hyldgaard, A. Kelkkanen, J. Kleis, L. Kong, S. Li, P. G. Moses, E. Murray, A. Puzder, H. Rydberg, E. Schröder, and T. Thonhauser, "A density functional for sparse matter," J. Phys.: Condens. Matter **21**, 084203 (2009).
[71] K. Berland and P. Hyldgaard, "Exchange functional that tests the robustness of the plasmon description of the van der Waals density functional," Phys. Rev. B **89**, 035412 (2014).
[72] T. Thonhauser, S. Zuluaga, C. A. Arter, K. Berland, E. Schröder, and P. Hyldgaard, "Spin signature of nonlocal correlation binding in metal-organic frameworks," Phys. Rev. Lett. **115**, 136402 (2015).
[73] G. Román-Pérez and J. M. Soler, "Efficient implementation of a van der Waals density functional: Application to double-wall carbon nanotubes," Phys. Rev. Lett. **103**, 096102 (2009).
[74] S. G. Johnson and M. Frigo, "Implementing FFTs in practice," in *Fast Fourier Transforms*, edited by C. S. Burrus (Connex-ions, Rice University, Houston, TX, 2008), Chap. 11.
[75] M. Pippig, "PFFT: An extension of FFTW to massively parallel architectures," SIAM J. Sci. Comput. **35**, C213–C236 (2013).
[76] See https://gitlab.com/siesta-project/libraries/libgridxc for libGridXC.
[77] F. Jollet, M. Torrent, and N. Holtzwarth, XML specification for atomic PAW datasets, https://esl.cecam.org/mediawiki/index.php/paw-xml.
[78] See https://gitlab.com/ElectronicStructureLibrary/libpspio for PSPIO library.
[79] A. García, M. J. Verstraete, Y. Pouillon, and J. Junquera, "The PSML format and library for norm-conserving pseudopotential data curation and interoperability," Comput. Phys. Commun. **227**, 51–71 (2018).
[80] See https://siesta-project.github.io/psml-docs; accessed November 2019.

[81]Atom code for the generation of norm-conserving pseudopotentials, the version maintained by the Siesta project can be accessed at http://icmab.es/siesta/Pseudopotentials/index.html, an alternative version is available at http://bohr.inesc-mn.pt/~jlm/pseudo.html; accessed July 2017.
[82]M. J. van Setten, M. Giantomassi, E. Bousquet, M. J. Verstraete, D. R. Hamann, X. Gonze, and G.-M. Rignanese, "The PSEUDODOJO: Training and grading a 85 element optimized norm-conserving pseudopotential table," Comput. Phys. Commun. **226**, 39–54 (2018).
[83]See https://www.pseudo-dojo.org; accessed November 2019.
[84]See https://esl.cecam.org/ESCDF_-_Electronic_Structure_Common_Data_Format for information on the ESCDF format specification.
[85]See https://esl.cecam.org/Libescdf for the libESCDF software, source, and documentation.
[86]D. Sébilleau, C. Natoli, G. M. Gavaza, H. Zhao, F. Da Pieve, and K. Hatada, "MsSpec-1.0: A multiple scattering package for electron spectroscopies in material science," Comput. Phys. Commun. **182**, 2567–2579 (2011).
[87]H. Ebert, D. Ködderitzsch, and J. Minár, "Calculating condensed matter properties using the KKR-Green's function method—Recent developments and applications," Rep. Prog. Phys. **74**, 096501 (2011).
[88]H. Ebert, J. Braun, D. Ködderitzsch, and S. Mankovsky, "Fully relativistic multiple scattering calculations for general potentials," Phys. Rev. B **93**, 075145 (2016).
[89]See https://www.etsf.eu/ for the ETSF library software, source, specifications, and documentation.
[90]See https://esl.cecam.org/ETSF_File_Format_Specifications for the file format specifications in the ETSF data standard.
[91]See https://www.hdfgroup.org/solutions/hdf5/ for information on the large-scale data format in HDF-5 and its associated library for parallel I/O handling.
[92]J. Deslippe, G. Samsonidze, D. A. Strubbe, M. Jain, M. L. Cohen, and S. G. Louie, "BerkeleyGW: A massively parallel computer package for the calculation of the quasiparticle and optical properties of materials and nanostructures," Comput. Phys. Commun. **183**, 1269–1289 (2012).
[93]See https://euspec.eu/ for information on the EUSpec network.
[94]See https://nomad-coe.eu/ for information on the NOMAD project and its data specifications.
[95]V. W.-Z. Yu, F. Corsetti, A. García, W. P. Huhn, M. Jacquelin, W. Jia, B. Lange, L. Lin, J. Lu, W. Mi, A. Seifitokaldani, Á. Vázquez-Mayagoitia, C. Yang, H. Yang, and V. Blum, "ELSI: A unified software interface for Kohn–Sham electronic structure solvers," Comput. Phys. Commun. **222**, 267–285 (2018).
[96]A. Marek, V. Blum, R. Johanni, V. Havu, B. Lang, T. Auckenthaler, A. Heinecke, H.-J. Bungartz, and H. Lederer, "The ELPA library: Scalable parallel eigenvalue solutions for electronic structure theory and computational science," J. Phys.: Condens. Matter **26**, 213201 (2014).
[97]P. Kůs, A. Marek, S. S. Köcher, H.-H. Kowalski, C. Carbogno, C. Scheurer, K. Reuter, M. Scheffler, and H. Lederer, "Optimizations of the eigensolvers in the ELPA library," Parallel Comput. **85**, 167–177 (2019).
[98]T. Imamura, S. Yamada, and M. Machida, "Development of a high-performance eigensolver on a peta-scale next-generation supercomputer system," Prog. Nucl. Sci. Technol. **2**, 643–650 (2011).
[99]J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki, "Accelerating numerical dense linear algebra calculations with GPUs," in *Numerical Computations with GPUs* (Springer, 2014), pp. 3–28.
[100]F. Corsetti, "The orbital minimization method for electronic structure calculations with finite-range atomic basis sets," Comput. Phys. Commun. **185**, 873–883 (2014).
[101]V. Hernandez, J. E. Roman, and V. Vidal, "SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems," ACM Trans. Math. Software **31**, 351–362 (2005).
[102]L. Lin, M. Chen, C. Yang, and L. He, "Accelerating atomic orbital-based electronic structure calculation via pole expansion and selected inversion," J. Phys.: Condens. Matter **25**, 295501 (2013).
[103]W. Dawson and T. Nakajima, "Massively parallel sparse matrix function calculations with NTPoly," Comput. Phys. Commun. **225**, 154–165 (2018).
[104]V. W.-Z. 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, A. Vázquez-Mayagoitia, C. Yang, and V. Blum, "ELSI—An open infrastructure for electronic structure solvers," arXiv:1912.13403 [physics.comp-ph] (2019).
[105]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," J. Chem. Phys. **152**, 124101 (2020).
[106]W. Hu, L. Lin, and C. Yang, "DGDFT: A massively parallel method for large scale density functional theory calculations," J. Chem. Phys. **143**, 124110 (2015).
[107]E. Vecharynski, C. Yang, and J. E. Pask, "A projected preconditioned conjugate gradient algorithm for computing many extreme eigenpairs of a Hermitian matrix," J. Comput. Phys. **290**, 73–89 (2015).
[108]Y. Pan, X. Dai, S. de Gironcoli, X.-G. Gong, G.-M. Rignanese, and A. Zhou, "A parallel orbital-updating based plane-wave basis method for electronic structure calculations," J. Comput. Phys. **348**, 482–492 (2017).
[109]See https://gitlab.e-cam2020.eu/esl/PIKSS for PIKSS.
[110]G. H. Wannier, "The structure of electronic excitation levels in insulating crystals," Phys. Rev. **52**, 191 (1937).
[111]N. Marzari, A. A. Mostofi, J. R. Yates, I. Souza, and D. Vanderbilt, "Maximally localized Wannier functions: Theory and applications," Rev. Mod. Phys. **84**, 1419–1475 (2012).
[112]C. Brouder, G. Panati, M. Calandra, C. Mourougane, and N. Marzari, "Exponential localization of Wannier functions in insulators," Phys. Rev. Lett. **98**, 046402 (2007).
[113]I. Souza, N. Marzari, and D. Vanderbilt, "Maximally localized Wannier functions for entangled energy bands," Phys. Rev. B **65**, 035109 (2001).
[114]G. Pizzi, V. Vitale, R. Arita, S. Blügel, F. Freimuth, G. Géranton, M. Gibertini, D. Gresch, C. Johnson, T. Koretsune, J. Ibañez-Azpiroz, H. Lee, J.-M. Lihm, D. Marchand, A. Marrazzo, Y. Mokrousov, J. I. Mustafa, Y. Nomura, Y. Nohara, L. Paulatto, S. Poncé, T. Ponweiser, J. Qiao, F. Thöle, S. S. Tsirkin, M. Wierzbowska, N. Marzari, D. Vanderbilt, I. Souza, A. A. Mostofi, and J. R. Yates, "Wannier90 as a community code: New features and applications," J. Phys.: Condens. Matter **32**, 165902 (2020).
[115]See https://github.com/wannier-developers/wannier90 for the `wannier90` GitHub repository.
[116]A. A. Mostofi, J. R. Yates, Y.-S. Lee, I. Souza, D. Vanderbilt, and N. Marzari, "Wannier90: A tool for obtaining maximally-localised Wannier functions," Comput. Phys. Commun. **178**, 685–699 (2008).
[117]V. Vitale, G. Pizzi, A. Marrazzo, J. R. Yates, N. Marzari, and A. A. Mostofi, "Automated high-throughput Wannierisation," npj Comput. Mat. **6**, 66 (2020).
[118]A. Damle, L. Lin, and L. Ying, "Compressed representation of Kohn–Sham orbitals via selected columns of the density matrix," J. Chem. Theory Comput. **11**, 1463–1469 (2015).
[119]A. Damle and L. Lin, "Disentanglement via entanglement: A unified method for Wannier localization," Multiscale Model. Simul. **16**, 1392–1410 (2018).
[120]G. Kresse and J. Furthmüller, "Efficiency of *ab-initio* total energy calculations for metals and semiconductors using a plane-wave basis set," Comput. Mater. Sci. **6**, 15–50 (1996).
[121]S. Blügel and G. Bihlmayer, "Full-potential linearized augmented planewave method," in *Computational Nanoscience: Do it Yourself!*, edited by J. Grotendorst, S. Blügel, and D. Marx (John von Neumann Institute for Computing, Jülich, 2006), Vol. 31, pp. 85–129.
[122]See https://elk.sourceforge.net for the Elk code, 2019.
[123]H. Weng, T. Ozaki, and K. Terakura, "Revisiting magnetic coupling in transition-metal-benzene complexes with maximally localized Wannier functions," Phys. Rev. B **79**, 235118 (2009).
[124]J. Ferrer, C. J. Lambert, V. M. García-Suárez, D. Z. Manrique, D. Visontai, L. Oroszlany, R. Rodríguez-Ferradás, I. Grace, S. W. D. Bailey, K. Gillemot, H. Sadeghi, and L. A. Algharagholy, "GOLLUM: A next-generation simulation tool for electron, thermal and spin transport," New J. Phys. **16**, 093029 (2014).

[125]Q. Wu, S. Zhang, H.-F. Song, M. Troyer, and A. A. Soluyanov, "Wanniertools: An open-source software package for novel topological materials," Comput. Phys. Commun. **224**, 405–416 (2018).
[126]See https://vides.nanotcad.com for NanoTCAD ViDES.
[127]A. Marini, C. Hogan, M. Grüning, and D. Varsano, "yambo: An *ab initio* tool for excited state calculations," Comput. Phys. Commun. **180**, 1392–1403 (2009).
[128]D. Gresch, G. Autès, O. V. Yazyev, M. Troyer, D. Vanderbilt, B. A. Bernevig, and A. A. Soluyanov, "Z2Pack: Numerical implementation of hybrid Wannier centers for identifying topological materials," Phys. Rev. B **95**, 075146 (2017).
[129]O. Parcollet, M. Ferrero, T. Ayral, H. Hafermann, I. Krivenko, L. Messio, and P. Seth, "TRIQS: A toolbox for research on interacting quantum systems," Comput. Phys. Commun. **196**, 398–415 (2015).
[130]S. Poncé, E. Margine, C. Verdi, and F. Giustino, "EPW: Electron-phonon coupling, transport and superconducting properties using maximally localized Wannier functions," Comput. Phys. Commun. **209**, 116–133 (2016).
[131]See https://www.buildbot.net for Buildbot.
[132]See https://www.travis-ci.org for Travis-CI.
[133]See https://en.wikipedia.org/wiki/Ouroboros for Ouroboros Wikipedia, the free encyclopedia; accessed 21 April 2020.
[134]F. Corsetti, A. A. Mostofi, and J. Lischner, "First-principles multiscale modelling of charged adsorbates on doped graphene," 2D Materials **4**, 025070 (2017).
[135]See https://e-cam.readthedocs.io/en/latest/Electronic-Structure-Modules/modules/MatrixSwitchDBCSR/readme.html#id8; accessed February 2020.
[136]U. Borštnik, J. VandeVondele, V. Weber, and J. Hutter, "Sparse matrix multiplication: The distributed block-compressed sparse row library," Parallel Comput. **40**, 47–58 (2014).
[137]See https://github.com/cp2k/dbcsr and https://www.cp2k.org/dbcsr; accessed February 2020.
[138]P. Ordejón, D. A. Drabold, R. M. Martin, and M. P. Grumbach, "Linear system-size scaling methods for electronic-structure calculations," Phys. Rev. B **51**, 1456 (1995).
[139]D. R. Bowler and T. Miyazaki, "Methods in electronic structure calculations," Rep. Prog. Phys. **75**, 036503 (2012).
[140]R. Ierusalimschy, *Programming in Lua*, 4th ed. (Feisty Duck Digital Book Distribution, 2016).
[141]See https://gitlab.com/siesta-project/libraries/libfdf for LibFDF.
[142]See https://en.wikipedia.org/wiki/Simple_API_for_XML for SAX, Simple API for XML.
[143]See https://wiki.gnome.org/Projects/Jhbuild for Jhbuild, since 2003.
[144]See https://www.gnome.org for Gnome project, since 1999.
[145]D. Alvarez, A. O'Cais, M. Geimer, and K. Hoste, "Scientific software management in real life: Deployment of easybuild on a large scale system," in *2016 Third International Workshop on HPC User Support Tools (HUST)* ((IEEE Press/Wiley and Sons, Hoboken, NJ, 2016), pp. 31–40, ISBN: 978-1-5090-3874-9.
[146]See https://yaml.org for Yaml, since 2001.
[147]S. Mohr, W. Dawson, M. Wagner, D. Caliste, T. Nakajima, and L. Genovese, "Efficient computation of sparse matrix functions for large-scale electronic structure calculations: The chess library," J. Chem. Theory Comput. **13**, 4684–4698 (2017).
[148]C. Hartwigsen, S. Goedecker, and J. Hutter, "Relativistic separable dual-space Gaussian pseudopotentials from H to Rn," Phys. Rev. B **58**, 3641 (1998).
[149]A. H. Larsen, M. Vanin, J. J. Mortensen, K. S. Thygesen, and K. W. Jacobsen, "Localized atomic basis set in the projector augmented wave method," Phys. Rev. B **80**, 195112 (2009).
[150]T. Rangel, D. Caliste, L. Genovese, and M. Torrent, "A wavelet-based projector augmented-wave (PAW) method: Reaching frozen-core all-electron precision with a systematic, adaptive and localized wavelet basis set," Comput. Phys. Commun. **208**, 1–8 (2016).
[151]B. Aradi, B. Hourahine, and T. Frauenheim, "DFTB+, a sparse matrix-based implementation of the DFTB method," J. Phys. Chem. A **111**, 5678–5684 (2007).
[152]See https://github.com/aradi/libsaydx for SAYDX—Structured Array Data Exchange.
[153]See https://about.gitlab.com/product/continuous-integration for Gitlab-CI, since 2011.
[154]See https://gitlab.kitware.com/cmake/community/wikis/doc/ctest/Testing-With-CTest for CTest from CMake, since 2000.
[155]See https://www.docker.com for Docker, since 2013.