JOURNAL OF SYNCHROTRON RADIATION
ISSN 1600-5775

# Corvus: a framework for interfacing scientific software for spectroscopic and materials science applications

S. M. Story,[a] F. D. Vila,[a,b]* J. J. Kas,[a,b] K. B. Raniga,[c] C. D. Pemmaraju[b] and J. J. Rehr[a,b]

[a]Department of Physics, University of Washington, Seattle, WA 98195, USA, [b]Theory Institute for Materials and Energy Spectroscopies, SLAC National Accelerator Laboratory, Menlo Park, CA 94025, USA, and [c]School of Humanities and Sciences, Stanford University, Stanford, CA 94305, USA. *Correspondence e-mail: fdv@uw.edu

Received 27 March 2019
Accepted 23 May 2019

Edited by R. W. Strange, University of Essex, UK

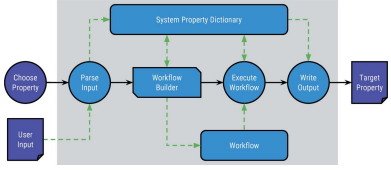
**Keywords:** Corvus; workflow manager; spectroscopy; materials science; property-driven.

**Supporting information**: this article has supporting information at journals.iucr.org/s

Corvus, a Python-based package designed for managing workflows of physical simulations that utilize multiple scientific software packages, is presented. Corvus can be run as an executable script with an input file and automatically generated or custom workflows, or interactively, in order to build custom workflows with a set of Corvus-specific tools. Several prototypical examples are presented that link density functional, vibrational and X-ray spectroscopy software packages and are of interest to the synchrotron community. These examples highlight the simplification of complex spectroscopy calculations that were previously limited to expert users, and demonstrate the flexibility of the Corvus infrastructure to tackle more general problems in other research areas.

## 1. Introduction

During the past decade, it has become increasingly important to combine the capabilities of multiple scientific software packages to simulate physical properties of complex condensed matter and molecular systems (Lawler *et al.*, 2008; Marini *et al.*, 2009; Valiev *et al.*, 2010). For example, calculations of excited states and spectra of materials often require conformational, electronic structure and response properties from one or more density functional theory (DFT) packages, which are subsequently passed to one or more analysis packages to calculate additional physical properties. In this way, building tools that interface various scientific software packages allows one to take advantage of the strengths of each, and to improve spectral simulations without having to re-implement complicated algorithms with disparate input–output schemes. Previous scientific workflow developments typically consist of the creation of purpose-oriented tools using a variety of languages and coding strategies. The result of such a strategy is the proliferation of *ad hoc* workflow tools that are neither expandable nor connectable. Consequently, there is a steep learning curve for implementation, development, deployment and usage of these tools, creating a high barrier for inexperienced users. In light of these issues, we now aim to develop a more robust framework that can, out-of-the-box, much more easily run any of our current ready-made scientific workflows. This allows users to focus on the physics as opposed to the computational details. Moreover, we also want a flexible and easily managed platform that would allow even non-expert users to construct or modify their own workflows. A variety of such workflow tools already exists (Pizzi *et al.*, 2016; Wilde *et al.*, 2011; Spjuth *et al.*, 2007; Goecks *et al.*, 2010; Altintas *et al.*, 2004; AlSairafi *et al.*, 2003). These



tools, and others meant to provide sophisticated computing paradigms such as federated and web-based computing, are rather general, and typically designed more for developers than general users. Therefore, we have developed a different all-encompassing physics-oriented framework that can generically accommodate various simple everyday workflows, spanning an arbitrary number of external software packages. This design has been implemented in the Python-based Corvus package, aimed at both less experienced users only wishing to run pre-built workflows that facilitates their research at the synchrotron, as well as expert users and developers interested in developing customizable interface and analysis tools. By catering to both target audiences, Corvus brings an attractive alternative to the scientific workflow landscape.

## 2. Corvus structure

Corvus is designed around two main layers: (1) a top-level layer that interacts with the user and executes the various steps defined in the workflow, and (2) lower-level infrastructure that interacts with the external scientific software to produce the desired output that is then translated into the physical properties required by the workflow. This stratification allows us to separate the implementation details specific to each external software package from the general workflow controls, to permit generic and modular workflow structures that can handle a wide variety of calculations.

Corvus relies on a *Workflow* object consisting of a sequence of modular steps, each providing part of the information needed to compute a desired physical property. These individual steps are dubbed *Exchanges*, since they take a set of input properties, pass it out to external software which then pass newly calculated output properties back in to a central workflow database. The code that deals with the external software is separated into independent components, or *Handlers*, that are specific to each external software package. These *Workflow* and *Handler* objects now contain the majority of our previous, *ad hoc*, workflow tools. Thus, Corvus is not simply a collection of purpose-oriented scripts but a broadly applicable scientific workflow infrastructure. The following sections give a more detailed description of the design and usage of Corvus.

### 2.1. Global structure of a Corvus simulation

The basic structure of Corvus consists of (1) the specification of the physical system being studied and its desired target properties, (2) the creation and execution of a given Corvus generated *Workflow* to compute them, and (3) a description and visualization of the resulting target properties. Fig. 1 shows a flowchart for the high-level infrastructure of a general Corvus calculation. In most common materials science and spectroscopy simulations, the input parameters used to control how a simulation runs (*e.g.* convergence conditions or grid sizes) are specific to how a given software package runs a particular type of simulation, and, in many cases, are spread over a multitude of input files. In Corvus, however, these types of parameters are easily deduced from the target property, and are managed automatically by the *Handlers*. In practical terms this means that the user only needs to provide a single minimal input file (see Section 2) which specifies the physical system. The input is parsed and stored in a lookup table or *System Property* dictionary, that contains both physical properties and control parameters for a given workflow. This dictionary will update as the Corvus *Workflow* executes, incrementally growing the knowledge base of the system with each *Exchange*.

After processing the information provided by the user, a *Workflow* can be read in from a file, or be automatically generated based on the target property and the scientific software available to Corvus. The structure of a generic *Workflow* will be discussed in Section 2.2. The automated workflow generation is currently carried out by a simple *Builder*. This *Builder* can handle basic tree-like workflows (Fig. 2) by proceeding backwards from the desired property that forms the root of the dependency tree. For dependencies that can be fulfilled by more than one *Handler* the code either

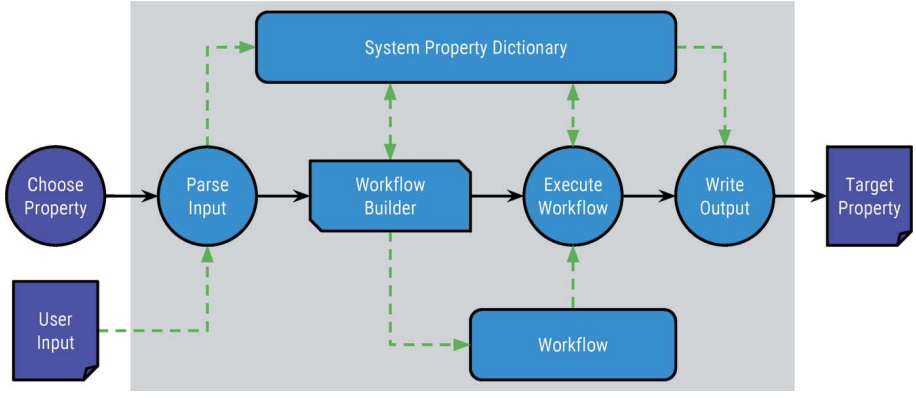



**Figure 1**
Overview of the top-level infrastructure in a Corvus run. The user's responsibility is constrained to choosing the target property and providing relevant input. Corvus creates a *System Property* dictionary which stores all information for the physical system being studied. This dictionary is continually updated as Corvus runs through the *Workflow*. The solid black arrows indicate progression in time, while the dashed green arrows represent data exchanges.

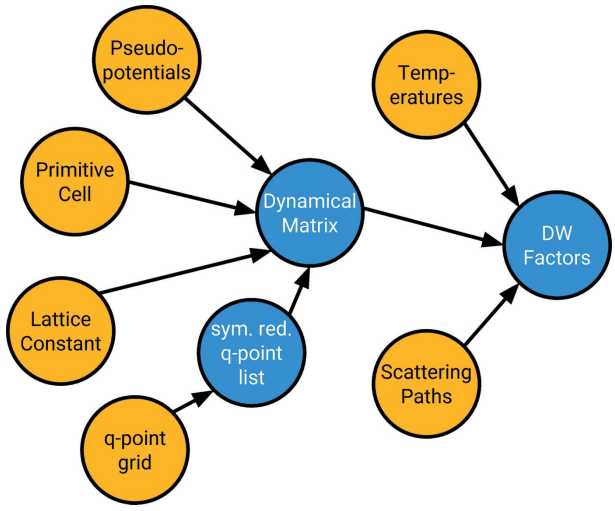



**Figure 2**
Illustration of a typical dependency graph for the main physical properties involved in the calculation of Debye–Waller (DW) factors using the *ABINIT* and *DMDW* software packages (as described in Section 3.2) User-provided properties are shown in orange, and the primary calculated properties are shown in blue. The automatic workflow *Builder* works backwards from the DW property determining its dependencies until all are satisfied either from previous workflow steps or from the user input.

uses internal defaults or the `usehandlers` input token, described below, to override the defaults and give users the option to select their desired *Handlers*. A fully general *Workflow* builder requires an algorithm that accounts for the fact that a single calculation can generate multiple desired properties, and that certain properties require loops and conditionals based on the input. This more general *Builder* will be developed in future versions.

The Corvus *Workflow* thus connects the sequence of *Exchanges* to be executed in order. We require the *Handlers* to abstract certain generic methods for running an *Exchange* using their associated external software (see Section 2.4). This allows Corvus to run through the entire workflow in an automated way. Each step in the calculation is run in a separate subdirectory so that the output from all the scientific software is later accessible in an organized fashion. Once finished, the output properties requested by the user are pulled from the property dictionary and written to file.

### 2.2. Scientific workflows driven by physical properties

The *Workflow* objects (Fig. 3) are designed to contain the minimal information required to allow Corvus to iterate through the sequence of calculations easily. More specifically, a *Workflow* consists of three main components: (1) a list of target properties requested by the user, (2) a list of *Exchange* operations to execute in order to produce the requested properties, and (3) the minimum list of input properties required from the user. Although the sequence of *Exchanges* must be executed in a specific order to produce the desired properties, as determined by the dependency tree spawned by the target property, the actual executions of the *Exchanges* are

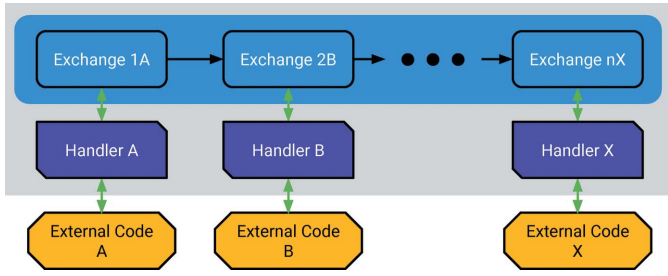



**Figure 3**
The Corvus *Workflow* object consists of a sequence of *Exchanges*. Each *Exchange* is an independent component of the *Workflow*, and uses its assigned *Handler* to run the external scientific software that produces the desired physical properties which are then passed back up to the *System Property* dictionary, so they can be accessed in subsequent *Exchanges*.

independent of each other. By breaking up the *Workflow* into discrete *Exchanges* it is easy to build in checkpointing and restarting of calculations. This is especially important when a user has a simulation with a workflow that starts with a computationally demanding calculation, but wants to rerun the simulation with different physical parameters that only affect later parts of the workflow.

The *Exchange* object is also lightweight, simply facilitating the exchange of properties from the system dictionary to a specified *Handler*, and *vice versa*. Thus *Exchanges* merely contain a specification of the *Handler* to be used for calculation, and a list of input and requested output properties. With this information, Corvus can automatically call upon the specified *Handler* at each stage and have it generate the necessary output from the provided input without any knowledge of the *Handler* implementation details.

In addition to the *Exchange* there are currently two other workflow sequence structures – *Updates* and *Loops*. An *Update* is used to overwrite a given property (or list of properties) in the system property dictionary. This is useful for tasks like initializing a property and replacing it with an optimized/converged value or parameter sweeps. The *Loop* structure is used to run the same *Exchange* multiple times, but iterating over a grid (or grids) of parameters by executing an *Update* for each *Exchange* iteration. These control structures cover a broad range of useful scientific workflows relevant to the spectroscopy, chemistry and materials science communities, as demonstrated by the examples presented below. Development to include more sophisticated possibilities like branching and controlled loops is currently under way.

### 2.3. System property lookup table

All input and calculated quantities are stored in a single lookup table, currently implemented as a Python dictionary. Throughout the execution of a workflow, physical properties of the system being studied can be polled, added and updated. Typical properties currently stored in the *System Properties* dictionary are the composition and structure of the system, its electronic structure, the dynamical matrix, *etc*. The *System Properties* dictionary is similar to the runtime database used by *NWChem* (Valiev *et al*., 2010). By taking advantage of the

Python dictionary structure, the state of the system can be easily updated or expanded. Moreover, given that the contents of the *System Properties* completely define the state of the system, Corvus can checkpoint simply by saving the *System Properties* dictionary. Another advantage is that the state of the system can be 'forked' and looped over simply by duplicating the dictionary or changing its internal properties during the workflow. Finally, although the *System Properties* dictionary is a single-run repository, its structure makes it very well suited to be reused in other calculations.

### 2.4. Interfacing external software with *Handlers*

As shown in the previous sections, the structural design of Corvus is such that the flow control can operate at the top level without being hindered by technical details specific to the external software used to calculate the physical properties. This makes building workflows nearly effortless, but relies on the necessary *Exchanges* being supported by the *Handlers*. This has the additional advantage of making the infrastructure tolerant to missing external packages in that, if a package is missing, the construction of the *Workflow* will fail and Corvus will generate an error before any calculations are attempted. As such, the vast majority of technical developments lies in the creation of appropriate *Handlers* for each software package module. These interfaces are in essence the evolution of our previous scientific workflow tools.

The *Handlers* are generally independent of one another, so that the development for supporting the external scientific software packages can be done independently as well, simplifying the work of developers. To interface with the top-level flow control, a *Handler* only needs to implement the following set of basic functions used by Corvus to create *Workflows* and run *Exchanges*:

`canProduce(output)` — returns true/false based on whether or not the associated external software can produce the given output(s).

`requiredInputFor(output)` — returns a list of input properties the associated external software requires to produce the given output(s). This includes any output properties not supported by this *Handler*.

`sequenceFor(output)` — returns the *Workflow* sequence required to produce the given output(s).

`prep()` — prepares a subdirectory where the external software is run.

`run(input, output)` — runs the external software using the input given the `input` dictionary, and returns the required output in the `output` dictionary.

`cleanup()` — does any necessary clean-up of the subdirectory before finishing up the *Exchange*.

In addition, *Handlers* can have two helper functions dedicated to generating the input and translating the output of the different external packages:

`generateInput(input, output)` — creates the necessary input files used by the external software to produce the desired output and returns the list of files.

`translateOutput(input, output)` — extracts the desired properties from the output generated by the external software and translates them into the internal property dictionary format and returns them as a dictionary to be stored in the system property dictionary.

### 2.5. Usage

Corvus currently supports the following command line arguments:

`-i/--input [filename]` — specify input file (default `Corvus.inp`).

`-w/--workflow` — specify pre-generated workflow file.

`--prefix [string]` — specify a prefix/label for subdirectories and output files, restricted to alphanumeric and [- _.] characters (default = 'Corvus').

`-c/--checkpoints` — enable checkpointing.

`-r/--resume` — load savefile ([prefix].nest) and continue from last savepoint.

`-s/--save [filename]` — specify savefile other than default.

`--parallelrun [command string]` — specify command prefix for parallelized program commands, *e.g.* "`mpirun -np 24`". This includes any options, environmental variables, settings, *etc.*, to control the parallel execution of an internal calculation.

For example, the command

```
run-corvus -i Cu-sf.inp -c --parallelrun
   "mpirun -np 96"
```

generates and runs a *Workflow* to produce the electron spectral function, where the target is selected in the `Cu-sf.inp` input file (Section 2), with checkpointing enabled, and using the standard MPI parallel jobs command `mpirun` with 96 processors.

The command

```
run-corvus -cr --prefix old --input
   Cu-refresh.inp
```

loads, with checkpointing enabled, the saved Corvus state file with prefix `old` (⇒ `old.nest`) and resumes where the previous *Workflow* left off, refreshing the values of any properties listed in the input file `Cu-refresh.inp`.

Finally,

```
run-corvus -w expansion.wf --parallelrun
   "aprun -np 240"
```

runs the pre-generated *Workflow* `expansion.wf` using 240 processors on a Cray supercomputer cluster, which uses the `aprun` command for parallel jobs.

In addition to operating as a *Workflow* driver, Corvus can be used as a software library for any of the workflow tools written by our developers, *e.g.* input file generators or output file parsers and translators, as they can be imported just as any other Python module. This also allows users to use the Corvus tools on-the-fly through the Python interpreter – useful for tasks such as *post hoc* analysis, or construction of workflows,

```
from corvus.structure.structures import *
import pickle

# Load in pre-made Workflow from file
wf = pickle.load(open('myworkflow.wf', 'r'))

# Want to change temperature to 300.0 K
tempUpdate = Update('temp', newValue=300.0)

# Change temperature at step 3 (index = 2)
wf.addExchangeAt(2, tempUpdate)
```

**Figure 4**
Example of how to change a property value within a *Workflow*. The required Corvus structures are imported from the Corvus module, a property *Update* is created, and *Exchange* is inserted into the workflow.

input files, *etc.* – or through user-made Python scripts. For example, if a user wanted to change the value of a certain property midway through a given *Workflow* this can be achieved as shown in Fig. 4: the *Workflow* `myworkflow.wf` is loaded, an update is generated for the `temp` property using the `Update` method, and this update is inserted into the *Workflow* at the desired point. Furthermore, by enabling checkpointing, the user ensures that the entire system property dictionary is saved to file. Since the saved file is formatted using Python's standard `pickle` module, the user can load the saved state, either in the Python interpreter or in any Python script, and manipulate any of the saved properties at will. This is a more attractive alternative to the user than having the additional step of parsing and translating output data files themselves. This command line structure is also well suited to the development of graphical user interfaces which can hide details of the syntax in these commands. A GUI, currently under development, would provide users with a graphical method for constructing, visualizing, debugging and running workflows, as well as visualizing output.

### 2.6. Input/output and configuration files

The input for Corvus has a generic property token driven format similar to that used in the *AI2PS*, *AI2NBSE* and *OCEAN* software packages developed in our group (Story *et al.*, 2014; Vinson *et al.*, 2011; Gilmore *et al.*, 2015). Their input-parsing tool has been translated into a Python version called *Parsnip*, which is an independent utility included in the Corvus package. The input grammar is composed of a token string followed by the input value(s) wrapped in curly braces (see Fig. 5 for details on the generic syntax). Property tokens must be one string of non-white-space characters. The input file supports standard Python-style commenting as well. This format is also used in the Corvus configuration and initialization files, which set paths to external software packages and default values for many parameters. This is very convenient since the input tokens can thus be easily checked against the currently implemented tokens. Moreover, new input tokens can be added to Corvus without any recoding by simply adding them to the configuration files. Once a token is added,

```
# comment
token1 {value} # comment
token2 {value value value}
token3 {
        value  # comment
        value  # comment
       }
token4  value
```

**Figure 5**
Example grammar for an input file read by *Parsnip*, the input parser used in Corvus. As shown with `token4`, if an input value is a single entry with no white-space, the curly braces can be dropped if desired. Comments can be denoted by the `#` and/or `!` characters.

its value is automatically available to all *Handlers*. Finally, the Corvus syntax and its internal Python representation as a simple dictionary makes it well suited to be converted into more standard formats such as XML and JSON. Corvus modules for this purpose are currently under development.

Since Corvus makes use of external software to calculate physical properties, there is output produced both by Corvus and the external packages themselves. The raw output from each *Exchange* is kept in the subdirectory where the external software was run. The output from Corvus is currently limited to the final target properties requested by the user, but the amount of desired output can be easily tailored to the needs of the user. In addition, all important data are stored in the saved *System Properties* dictionary, and can be easily accessed after the end of the calculation.

The list of input tokens and their default or expected values is too long to include here but can be found in the Corvus manual. Here we highlight a few examples of input tokens that are important to control the way a Corvus run is executed, some general methodological tokens, and a few of the physical properties that can set in the input:

`target_list` — a list of space-separated strings that controls the targets of a given *Workflow*.

`usehandlers` — a list of the *Handlers* to be used to control the generation of the *Workflow*. This token helps Corvus manage targets that can be produced by multiple *Handlers*.

`method` — this token helps the *Workflow* generator choose which *Handlers* can produce properties with a particular theoretical method, for instance, DFT, MP2 or CCSD.

`cell_vectors` — this token sets the simulation cell vectors for a periodic boundary conditions calculation.

### 2.7. Requirements and obtaining the code

One of the goals of the Corvus platform is to minimize installation and management issues for the user. We therefore kept this initial version of Corvus constrained to Python 2.7. Nevertheless, we are currently working on porting the code to Python 3, in preparation for this version becoming more widespread. Additional Python packages that provide useful analysis tools [*e.g.* Numpy (van der Walt *et al.*, 2011)] can be treated like any other external software package, and can be

enabled through their own *Handlers*. Consequently, users only need Python 2.7 to install Corvus, and the Corvus workflows that are possible will be determined by the specific external software packages available on the user's system. The code is open source (BSD 3-clause license) and can be found in the software website of the Theoretical Institute for Materials and Energy Spectroscopies (TIMES) (TIMES, 2019).

## 3. Example applications

Our initial applications of Corvus focused on the simulation of various spectral and thermal properties of molecular and condensed matter systems. The following sections briefly describe prototypical scientific workflows for a few key properties in order to demonstrate the capabilities of Corvus, in particular highlighting its usefulness to synchrotron scientists.

### 3.1. Dynamic disorder in XANES

Experimental X-ray absorption spectroscopy (XAS) measurements are often carried out at room temperature or in heated samples (Vinson *et al.*, 2014; Hedin & Rosengren, 1977; Fujikawa *et al.*, 1999; Uejio *et al.*, 2008; Pascal *et al.*, 2014; Nemausat *et al.*, 2015). These conditions induce thermal dynamical disorder which generally leads to the broadening, shift and, occasionally, the appearance of additional features in the spectra. Peak shifts can be caused by thermal expansion effects as the average bond distances change with temperature. Additional shoulder-like features involving new transitions can also appear, arising from dynamical symmetry-breaking around certain atoms. In this instance, X-ray dipole transitions that are symmetry-forbidden in the 0 K structure become allowed due to thermal fluctuations of the nuclear positions (Pascal *et al.*, 2014, 2015). These effects cannot be reproduced simply by broadening spectra computed with a 0 K structure, and simulations must take into account the temperature-induced dynamical disorder. One way to straightforwardly introduce such effects in first-principles XAS simulations is to average over *ab initio* molecular dynamics (AIMD) trajectories (Uejio *et al.*, 2008; Pascal *et al.*, 2014). This approach is most useful in large, disordered or anharmonic systems involving, for instance, surfaces, solvent–solute interactions, *etc.*, where explicit calculations of the vibrational frequencies and eigenmodes are either prohibitively expensive or inaccurate. Furthermore, such an approach can be generalized to excited-state non-adiabatic molecular dynamics as well (Attar *et al.*, 2017). Given that AIMD trajectory-averaged XAS of molecular and nanoscale systems are regularly needed, we implemented a Corvus workflow to automate the process within *NWChem* (Valiev *et al.*, 2010). Fig. 6 shows a diagram of the workflow, while Figs. 7 and 8 show the Corvus input file and the results obtained for the 300 K AIMD-averaged C *K*-edge XAS of chloroacetylene, respectively. It should be noted that, although in this example we use *NWChem* to compute the XAS spectrum, this step could also be accomplished using the Feff *Handler*, and, in the future, an *OCEAN* (Story *et al.*, 2014; Vinson *et al.*, 2011; Gilmore *et al.*, 2015) *Handler* that is currently under development. Moreover, unless otherwise stated, the input shown here and in other figures is all that is required to produce the requested target.

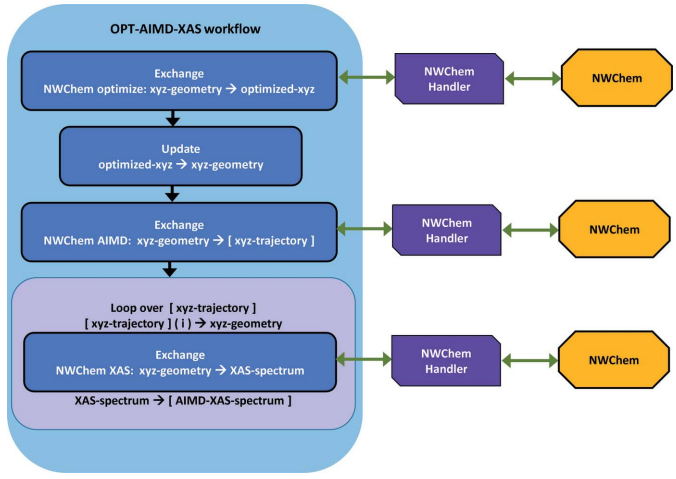



**Figure 6**
Workflow diagram for the calculation of AIMD-averaged X-ray spectra using *NWChem*.

```
target_list { opt_xasavg }
usehandlers { Nwchem }
cell_struc_xyz_red {
C    0.0000   0.0000 -1.8297
C    0.0000   0.0000 -0.6193
Cl   0.0000   0.0000  1.0348
H    0.0000   0.0000 -2.8988
}
nwchem.xas.xrayenergywin { -12.0 -08.0 }
nwchem.qmd.targ_temp 300
nwchem.xas.nroots 100
```

**Figure 7**
Sample input file for calculations of AIMD-averaged X-ray spectra using *NWChem*. The input declares the target, the structure of the molecule, the temperature for the MD, and the parameters for the XAS calculation.

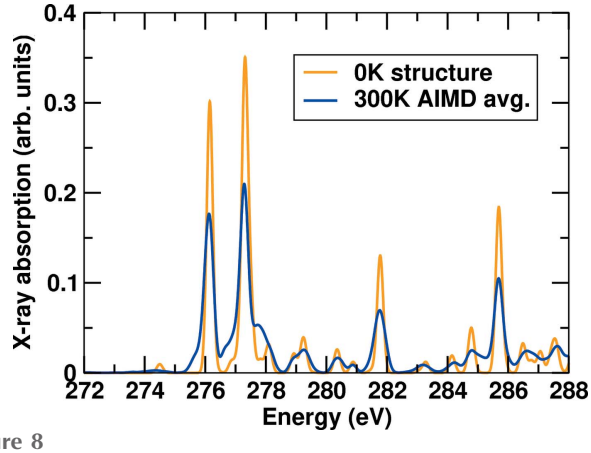



**Figure 8**
Comparison of the 0 K and AIMD-averaged C *K*-edge X-ray absorption spectra of chloroacetylene computed with Corvus and *NWChem*.

In this example, the user simply provides a starting structure in standard *xyz* format, and indicates that the output should be an AIMD-averaged X-ray absorption spectrum through the `target_list` option `opt_xasavg`. The option `opt_xasavg` indicates that the input structure should be optimized before starting the AIMD. The only additional information needed is the target temperature for the AIMD simulation *via* the `nwchem.qmd.targ_temp` option, as well as the number of roots and relevant energy range for a TDDFT simulation of the C *K*-edge X-ray absorption spectrum (Zhang *et al.*, 2012). Within *NWChem*, the number of roots dictates the total number of excitation energies calculated in solving the linear-response TDDFT Casida equations (Casida & Huix-Rotllant, 2012). Additionally for X-ray absorption spectra, a restricted energy window that includes the relevant core-orbitals as initial states needs to be indicated to target the desired subset of core-excitations (Zhang *et al.*, 2012). Together, these parameters control the range of energies over which spectral transitions are calculated. With this minimal input Corvus will carry out a series of steps to produce the AIMD averaged spectrum shown in Fig. 8. The internal Corvus *NWChem* input file is a combination of the user-provided input and an extensive set of sensible defaults that aim to strike a compromise between accuracy and efficiency. The user can of course override the defaults by setting them explicitly in the Corvus input file. In this example a total of 120 individual AIMD snapshots are used, each of which results in one XAS spectrum. In the last step Corvus consolidates the individual spectra, which are in the form of a list of excitation energies and corresponding oscillator strengths, into a single file to be post-processed by the user before plotting. The final AIMD averaged output spectrum also includes a 0.1 eV core-hole lifetime broadening. As expected, significant temperature broadening of the spectral peaks as well as some additional structure is observed.

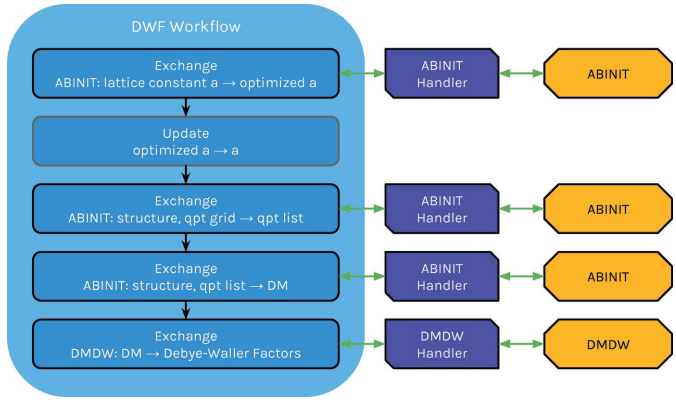



**Figure 9**
Workflow diagram for the calculation of EXAFS DW factors by passing *ABINIT* dynamical matrices to the *FEFF* standalone *DMDW* module.

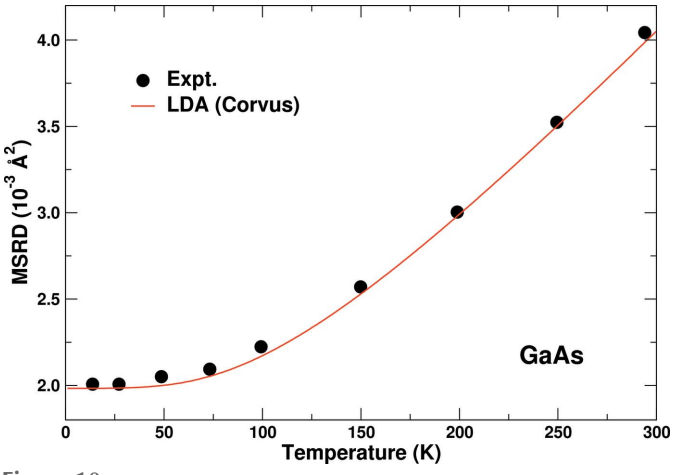



**Figure 10**
Comparison of the experimental (Ahmed *et al.*, 2013) and theoretical temperature variation of the near-neighbor mean square relative displacement (MSRD) for GaAs.

### 3.2. *Ab initio* Debye–Waller factors

Debye–Waller (DW) factors describe the *exponential damping* observed in X-ray spectra due to thermal vibrations and configurational disorder. The effect of vibrations is dominated by an average over the oscillatory behavior of the scattering paths – as the atoms oscillate, so do the lengths of the scattering paths – the largest contribution coming from the mean square relative displacement of the scattering path (Grimvall, 1986; Vila *et al.*, 2007). These DW factors can be efficiently and accurately calculated from a system's dynamical matrix (DM) of force constants, or Hessian, using the standalone *DMDW* (*Dynamical Matrix Debye–Waller*) module from the *FEFF* software package (Rehr *et al.*, 2009; Vila *et al.*, 2007). The force constants, however, cannot be calculated internally in *FEFF* and must be provided to the *DMDW* module from an external source. Currently Corvus includes handlers for *ABINIT* (Gonze *et al.*, 2016, 2009; Gonze & Lee, 1997) and *NWChem* (Valiev *et al.*, 2010) that can produce dynamical matrices and Hessians, respectively, and can be used in *DMDW* calculations. The typical Corvus workflow using the *ABINIT Handler* for such a calculation is shown in Fig. 9. In this workflow the complicated process (Vila *et al.*, 2007) of generating the appropriate reciprocal space *q*-point grid and converting the reciprocal space dynamical matrix to real space is done automatically before passing it to the *DMDW* module. Typical results from this workflow are presented in Fig. 10, which shows a comparison of the experimental (Ahmed *et al.*, 2013) and theoretical temperature variation of the near-neighbor mean square relative displacement for GaAs. It should be noted that, although these DW factors can be included in EXAFS fits *post facto*, this workflow can also be extended to compute EXAFS spectra with *ab initio* DW factors directly by replacing the *DMDW* step with a *FEFF* one that computes the DW factors internally and includes them in the EXAFS calculation. This workflow is currently under development.

Fig. 11 shows the result of a different Corvus workflow in which the *NWChem* software package is used to generate Hessians (the molecular equivalent of the dynamical matrix) for a variety of diatomic, triatomic and tetratomic molecules.

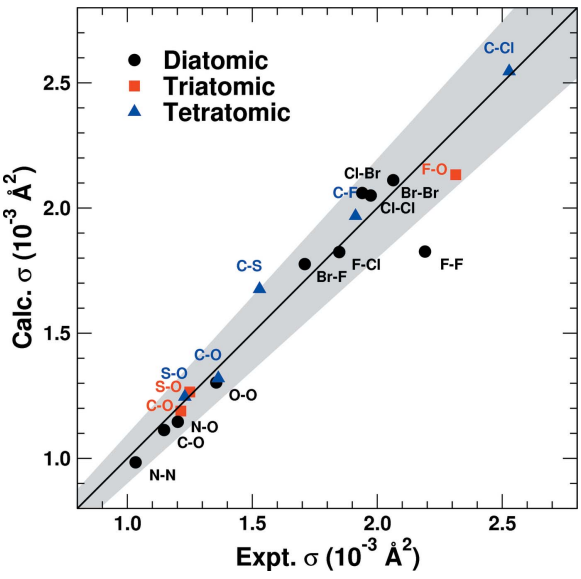



**Figure 11**
Comparison of the experimental (Dimakis & Bunker, 1998) and theoretical bond single-scattering DW factors at room temperature for a variety of diatomic, triatomic and tetratomic molecules. The grayed area indicates the typical uncertainty in experimentally determined DW factors (Crozier, 1997).

Typical input and output files for these types of calculations are shown in Figs. 12 and 13, and the supporting information shows a comparison between a given Corvus input and all the input files automatically generated for that workflow, highlighting the extreme simplification provided by Corvus. This workflow also includes an automatic optimization of the structures of the molecules. The simulated single-scattering DW factors for the different types of bonds are compared with experimental results (Dimakis & Bunker, 1998) and show that they are mostly within the expected uncertainty (Crozier, 1997) of the experimental methods (±10%, gray region on the plot).

### 3.3. Thermal expansion

A goal of many simulations is to calculate the temperature dependence of phonon properties by including thermal expansion effects. This entails first determining the structure of the system as a function of temperature, then calculating the physical properties with that structure. These types of calculations showcase the more advanced usage of Corvus, including parameter sweeps and dynamically updated properties. The thermal expansion of a system is defined by its equilibrium configuration at a given temperature for a range of temperatures. At thermal equilibrium, the Helmholtz free energy is minimized (Vila *et al.*, 2007). Therefore, the procedure for determining the lattice constant as a function of temperature is as follows: (1) For a range of temperatures calculate the Helmholtz free energy for a range of lattice constants at this temperature, and (2) find the minimum of the Helmholtz free energy with respect to lattice constant for each temperature. This can be accomplished by building up a

```
target_list { s2 }
usehandlers { Nwchem Dmdw }
struc_xyz {
          S         0.000000     0.000000     0.372789
          O         0.000000     1.245184    -0.372789
          O         0.000000    -1.245184    -0.372789
}
nwchem.basis {
* library "aug-cc-pVTZ"
}
dmdw.ioflag 0
dmdw.nlanc  3
dmdw.tempgrid {3 100.0 300.0}
dmdw.paths {
1
2 0 0 3.0
}
```

**Figure 12**
Sample input file for calculations of DW factors using a combination of *NWChem* and the *DMDW* module in *FEFF*. The input declares the list of targets, the structure of the molecule, the basis set for the *NWChem* calculations, and the parameters for the *DMDW* section.

```
{'1,2': {'Pathlength (Ang)': 1.451,
         'Temp (K)': [100.0, 200.0, 300.0],
         's^2': [1.258, 1.258, 1.265]},
 '1,3': {'Pathlength (Ang)': 1.451,
         'Temp (K)': [100.0, 200.0, 300.0],
         's^2': [1.258, 1.258, 1.265]},
 '2,1': {'Pathlength (Ang)': 1.451,
         'Temp (K)': [100.0, 200.0, 300.0],
         's^2': [1.258, 1.258, 1.265]},
.
.
```

**Figure 13**
Sample output file for calculations of DW factors using a combination of *NWChem* and the *DMDW* module in *FEFF*. The target property ('s^2') is reported as a function of temperature together with the parameters for the scattering path, such as the indices of the atoms involved and the length of the path. This output is in the form of a Python dictionary that can be easily imported into other analysis tools.

Helmholtz free energy surface as a function of both temperature and lattice constant, then minimizing at each temperature. A typical workflow for this process is shown in Fig. 14. *DMDW* itself produces results for a grid of temperatures, so Corvus can collect these results for a grid of lattice sizes to create the Helmholtz free energy surface. The collection of property calculations is finally passed to the Numpy *Handler*, which does the remaining analysis to determine the $a(T)$ curve. Fig. 15 presents typical results for the relative thermal expansion of Cu compared with experiment (Nix & MacNair, 1941; White, 1973).

### 3.4. Structural optimization and RIXS/XES

One very common and important task for analysis of spectroscopies is to optimize the structure of a system given an initial structure based on, for example, crystallographic or

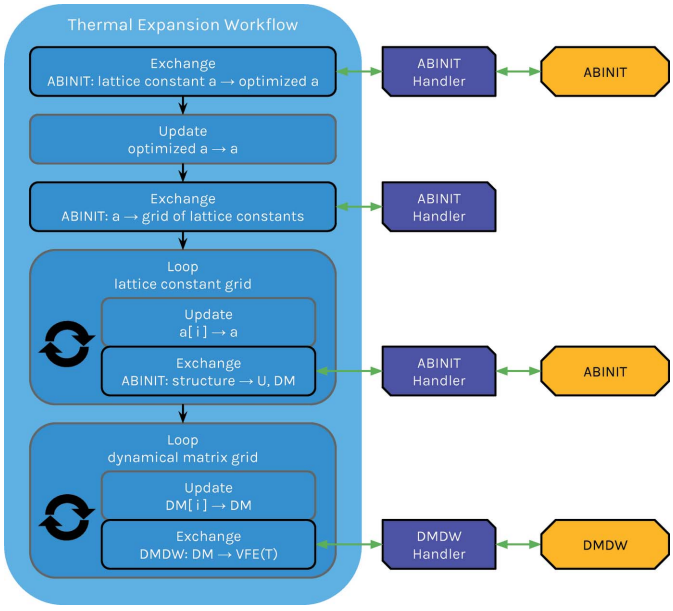



**Figure 14**
Corvus workflow for calculating thermal expansion using the *ABINIT* and *DMDW* Corvus handlers.

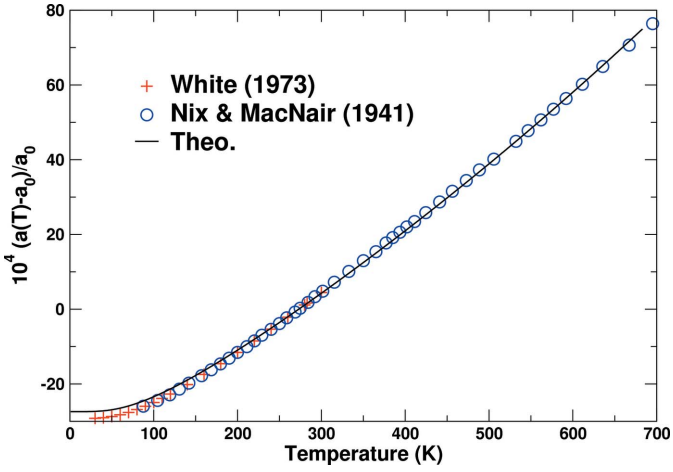



**Figure 15**
Theoretical and experimental (Nix & MacNair, 1941; White, 1973) relative thermal expansion for Cu.

molecular data of a similar system. The optimization can be performed using DFT or hybrid functionals, or more accurate methods. *FEFF* then takes the optimized structure as input, and can produce a variety of spectroscopic quantities. In this example, the X-ray emission spectrum (XES) and resonant inelastic X-ray scattering (RIXS) of the $[LMn(acac)N]^+$ cation in $[LMn(acac)N]BPh_4$ (shown at the top of Fig. 18, and where the 'L' stands for 1,4,7-trimethyl-1,4,7-triazacyclononane) are calculated at the initial structure using *FEFF*, optimization is performed using *ORCA* (Neese, 2012), and the spectra are calculated again using the optimized structure. This is a relatively simple workflow (see Fig. 16), but can be directed by the user, *i.e.* by specifying a set of targets as well as the order of calculation of those targets. While the input to the two software packages separately might be quite involved, especially

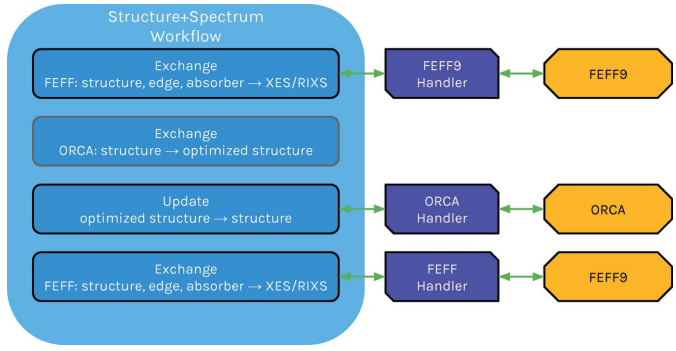



**Figure 16**
Diagram of workflow for XES and RIXS from initial and optimized structure.

```
# Choose target and handlers
target_list { feffRIXS }
usehandlers { Orca Feff }

# X-Ray edge
feff.edge { K }
# Lorenzian broadening of spectra
spectral_broadening { 0.7 }
# Charge of molecule
charge { 1 }
# Absorbing atom is the first in the cluster
absorbing_atom { 1 }
# Atomic cluster in Angstroms (xyz style format)
cluster {
Mn 0.000000  0.000000  0.000000
O  0.199422  0.594958  1.791970
N -0.731983 -1.757497  0.528479
O -1.808293  0.994578 -0.167048
N -0.008891 -0.580607 -2.000021
.
.
```

**Figure 17**
Sample input file for calculations of XES and RIXS using *ORCA* and *FEFF*.

for the calculation of RIXS, the automation has reduced the input to a minimal set of required properties, such as the coordinates of the atoms in the molecule (cluster), the absorbing atom, and the X-ray edge of interest, as shown in Fig. 17.

Fig. 18 shows the RIXS spectrum (middle) of $[LMn(acac)N]BPh_4$ calculated at the optimized structure, and the XES spectrum (bottom) calculated at the optimized structure compared with that calculated from the initial structure and experimental results (Smolentsev *et al.*, 2009). Note the appreciable improvement in agreement with experiment when the optimized structure is used.

## 4. Conclusions

We have developed Corvus, a general property-driven scientific workflow tool designed to interface multiple scientific

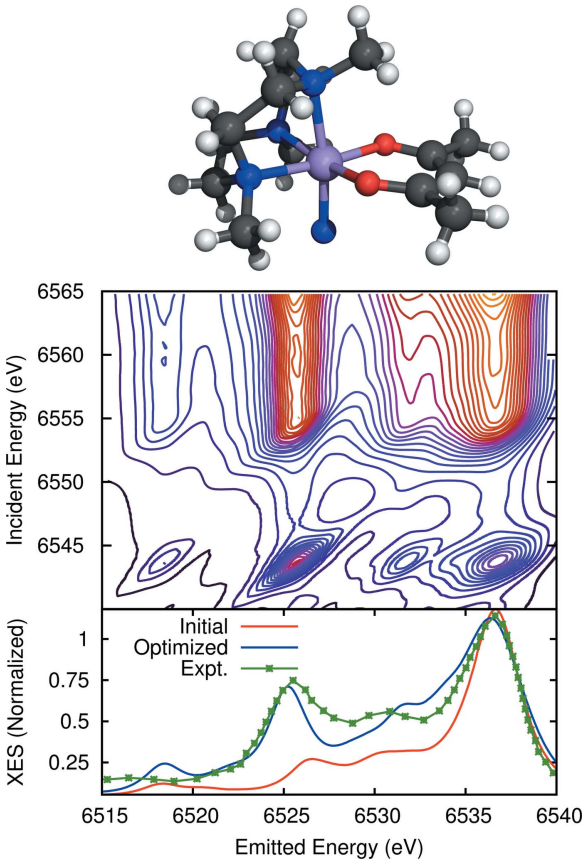



**Figure 18**
Structure of the LMn(acac)N]$^{+}$ cation (top) in [LMn(acac)N]$BPh_4$, where the 'L' stands for 1,4,7-trimethyl-1,4,7-triazacyclononane, its RIXS spectrum (middle) at the optimized structure, and theoretical XES (bottom) calculated at both the optimized (blue) and initial structure (red), compared with the experimental result (Smolentsev *et al.*, 2009) (green crosses).

software packages, highlighting its application to excited state and spectroscopic properties. In order to allow for automated calculations and easy construction of scientific workflows, Corvus keeps most technical details of the external scientific software well separated from the flow control. In addition, Corvus has hierarchical levels of complexity, allowing inexperienced users to perform single-shot workflows, and expert users to use it as an open library of functions for customized workflows. We have demonstrated the versatility of Corvus by applying it to calculations of excited state and spectroscopic properties, using a variety of possible workflows based on different external software packages, including *FEFF*, *NWChem*, *ABINIT* and *ORCA*. Many extensions are possible, both to other external packages for high-performance computing and exascale resources, through integration with other workflow tools such as SWIFT, and by creating modules that interface to data sources such as the Materials Project (Jain *et al.*, 2013). We are currently developing a smart workflow generator capable of creating more complex, non-linear data flows. Finally we are developing a graphical user interface that will enable users to create and execute their own data flows in a simplified, intuitive way.

## Funding information

This work was supported primarily by the Theory Institute for Materials and Energies Science (TIMES) at SLAC which is funded by the US Department of Energy, Office of Basic Energy Sciences, Division of Materials Sciences and Engineering, under Contract No. DE-AC02-76SF00515, with computational support from NERSC, a DOE Office of Science User Facility, under Contract No. DE-AC02-05CH11231.

## References

Ahmed, S. I., Aquilanti, G., Novello, N., Olivi, L., Grisenti, R. & Fornasini, P. (2013). *J. Chem. Phys.* **139**, 164512.
AlSairafi, S., Emmanouil, F.-S., Ghanem, M., Giannadakis, N., Guo, Y., Kalaitzopoulos, D., Osmond, M., Rowe, A., Syed, J. & Wendel, P. (2003). *Int. J. High. Perform. Comput. Appl.* **17**, 297–315.
Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B. & Mock, S. (2004). *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM2004)*, 21–23 June 2004, Santorini, Greece, pp. 423–424.
Attar, A. R., Bhattacherjee, A., Pemmaraju, C. D., Schnorr, K., Closser, K. D., Prendergast, D. & Leone, S. R. (2017). *Science*, **356**, 54–59.
Casida, M. E. & Huix-Rotllant, M. (2012). *Annu. Rev. Phys. Chem.* **63**, 287–323.
Crozier, E. (1997). *Nucl. Instrum. Methods Phys. Res. B*, **133**, 134–144.
Dimakis, N. & Bunker, G. (1998). *Phys. Rev. B*, **58**, 2467–2475.
Fujikawa, T., Rehr, J. J., Wada, Y. & Nagamatsu, S. (1999). *J. Synchrotron Rad.* **6**, 317–319.
Gilmore, K., Vinson, J., Shirley, E., Prendergast, D., Pemmaraju, C., Kas, J., Vila, F. & Rehr, J. (2015). *Comput. Phys. Commun.* **197**(Suppl. C), 109–117.
Goecks, J., Nekrutenko, A., Taylor, J. & Galaxy Team, T. (2010). *Genome Biol.* **11**, R86.
Gonze, X., Amadon, B., Anglade, P.-M., Beuken, J.-M., Bottin, F., Boulanger, P., Bruneval, F., Caliste, D., Caracas, R., Côté, M., Deutsch, T., Genovese, L., Ghosez, P., Giantomassi, M., Goedecker, S., Hamann, D., Hermet, P., Jollet, F., Jomard, G., Leroux, S., Mancini, M., Mazevet, S., Oliveira, M., Onida, G., Pouillon, Y., Rangel, T., Rignanese, G.-M., Sangalli, D., Shaltaf, R., Torrent, M., Verstraete, M., Zerah, G. & Zwanziger, J. (2009). *Comput. Phys. Commun.* **180**, 2582–2615.
Gonze, X., Jollet, F., Abreu Araujo, F., Adams, D., Amadon, B., Applencourt, T., Audouze, C., Beuken, J.-M., Bieder, J., Bokhanchuk, A., Bousquet, E., Bruneval, F., Caliste, D., Côté, M., Dahm, F., Da Pieve, F., Delaveau, M., Di Gennaro, M., Dorado, B., Espejo, C., Geneste, G., Genovese, L., Gerossier, A., Giantomassi, M., Gillet, Y., Hamann, D. R., He, L., Jomard, G., Laflamme Janssen, J., Le Roux, S., Levitt, A., Lherbier, A., Liu, F., Lukačević, I., Martin, A., Martins, C., Oliveira, M. J. T., Poncé, S., Pouillon, Y., Rangel, T., Rignanese, G., Romero, A. H., Rousseau, B., Rubel, O., Shukri, A. A., Stankovski, M., Torrent, M., Van Setten, M. J., Van Troeye, B., Verstraete, M. J., Waroquiers, D., Wiktor, J., Xu, B., Zhou, A. & Zwanziger, J. W. (2016). *Comput. Phys. Commun.* **205**, 106–131.
Gonze, X. & Lee, C. (1997). *Phys. Rev. B*, **55**, 10355–10368.
Grimvall, G. (1986). *Thermophysical Properties of Materials.* Amsterdam: Elsevier.
Hedin, L. & Rosengren, A. (1977). *J. Phys. F: Met. Phys.* **7**, 1339–1348.
Jain, A., Ong, S. P., Hautier, G., Chen, W., Richards, W. D., Dacek, S., Cholia, S., Gunter, D., Skinner, D., Ceder, G. & Persson, K. (2013). *APL Mater.* **1**, 011002.
Lawler, H. M., Rehr, J. J., Vila, F., Dalosto, S. D., Shirley, E. L. & Levine, Z. H. (2008). *Phys. Rev. B*, **78**, 205108.

Marini, A., Hogan, C., Grüning, M. & Varsano, D. (2009). *Comput. Phys. Commun.* **180**, 1392–1403.
Neese, F. (2012). *WIREs Comput. Mol. Sci.* **2**, 73–78.
Nemausat, R., Cabaret, D., Gervais, C., Brouder, C., Trcera, N., Bordage, A., Errea, I. & Mauri, F. (2015). *Phys. Rev. B*, **92**, 144310.
Nix, F. C. & MacNair, D. (1941). *Phys. Rev.* **60**, 597–605.
Pascal, T. A., Boesenberg, U., Kostecki, R., Richardson, T. J., Weng, T.-C., Sokaras, D., Nordlund, D., McDermott, E., Moewes, A., Cabana, J. & Prendergast, D. (2014). *J. Chem. Phys.* **140**, 034107.
Pascal, T. A., Pemmaraju, C. D. & Prendergast, D. (2015). *Phys. Chem. Chem. Phys.* **17**, 7743–7753.
Pizzi, G., Cepellotti, A., Sabatini, R., Marzari, N. & Kozinsky, B. (2016). *Comput. Mater. Sci.* **111**, 218–230.
Rehr, J. J., Kas, J. J., Prange, M. P., Sorini, A. P., Takimoto, Y. & Vila, F. D. (2009). *C. R. Phys.* **10**, 548–559.
Smolentsev, G., Soldatov, A. V., Messinger, J., Merz, K., Weyhermüller, T., Bergmann, U., Pushkar, Y., Yano, J., Yachandra, V. K. & Glatzel, P. (2009). *J. Am. Chem. Soc.* **131**, 13161–13167.
Spjuth, O., Helmus, T., Willighagen, E. L., Kuhn, S., Eklund, M., Wagener, J., Murray-Rust, P., Steinbeck, C. & Wikberg, J. E. (2007). *BMC Bioinformatics*, **8**, 59.
Story, S. M., Kas, J. J., Vila, F. D., Verstraete, M. J. & Rehr, J. J. (2014). *Phys. Rev. B*, **90**, 195135.
TIMES (2019). *Corvus*, https://web.stanford.edu/group/times/software.html.
Uejio, J. S., Schwartz, C. P., Saykally, R. J. & Prendergast, D. (2008). *Chem. Phys. Lett.* **467**, 195–199.
Valiev, M., Bylaska, E., Govind, N., Kowalski, K., Straatsma, T., Van Dam, H. J. J., Wang, D., Nieplocha, J., Apra, E., Windus, T. & de Jong, W. (2010). *Comput. Phys. Commun.* **181**, 1477–1489.
Vila, F. D., Rehr, J. J., Rossner, H. H. & Krappe, H. J. (2007). *Phys. Rev. B*, **76**, 014301.
Vinson, J., Jach, T., Elam, W. & Denlinger, J. (2014). *Phys. Rev. B*, **90**, 205207.
Vinson, J., Rehr, J. J., Kas, J. J. & Shirley, E. L. (2011). *Phys. Rev. B*, **83**, 115106.
Walt, S. van der, Colbert, S. C. & Varoquaux, G. (2011). *Comput. Sci. Eng.* **13**, 22–30.
White, G. K. (1973). *J. Phys. D Appl. Phys.* **6**, 2070–2078.
Wilde, M., Hategan, M., Wozniak, J. M., Clifford, B., Katz, D. S. & Foster, I. (2011). *Parallel Comput.* **37**, 633–652.
Zhang, Y., Biggs, J. D., Healion, D., Govind, N. & Mukamel, S. (2012). *J. Chem. Phys.* **137**, 194306.