The Journal of Chemical Physics

RESEARCH ARTICLE | NOVEMBER 22 2021

Quantum Chemistry Common Driver and Databases (QCDB) and Quantum Chemistry Engine (QCENGINE): Automation and interoperability among computational chemistry programs

FREE

Daniel G. A. Smith ; Annabelle T. Lolinco ; Zachary L. Glick ; Jiyoung Lee ; Asem Alenaizan ; Taylor A. Barnes ; Carlos H. Borca ; Roberto Di Remigio ; David L. Dotson ; Sebastian Ehlert ; Alexander G. Heide ; Michael F. Herbst ; Jan Hermann ; Colton B. Hicks ; Joshua T. Horton ; Adrian G. Hurtado ; Peter Kraus ; Holger Kruse ; Sebastian J. R. Lee ; Jonathon P. Misiewicz ; Levi N. Naden ; Farhad Ramezanghorbani ; Maximilian Scheurer ; Jeffrey B. Schriber ; Andrew C. Simmonett ; Johannes Steinmetzer ; Jeffrey R. Wagner ; Logan Ward ; Matthew Welborn; Doaa Altarawy ; Jamshed Anwar ; John D. Chodera ; Andreas Dreuw ; Heather J. Kulik ; Fang Liu ; Todd J. Martínez ; Devin A. Matthews ; Henry F. Schaefer, III ; Jiří Šponer ; Justin M. Turney ; Lee-Ping Wang ; Nuwan De Silva ; Rollin A. King ; John F. Stanton ; Mark S. Gordon ; Theresa L. Windus ; C. David Sherrill ; Lori A. Burns

J. Chem. Phys. 155, 204801 (2021)
https://doi.org/10.1063/5.0059356

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

Quantum Chemistry Common Driver and Databases (QCDB) and Quantum Chemistry Engine (QCENGINE): Automation and interoperability among computational chemistry programs

Cite as: J. Chem. Phys. 155, 204801 (2021); doi: 10.1063/5.0059356
Submitted: 8 June 2021 • Accepted: 1 October 2021 •
Published Online: 22 November 2021

Daniel G. A. Smith,[1,a] Annabelle T. Lolinco,[2] Zachary L. Glick,[3] Jiyoung Lee,[2,b] Asem Alenaizan,[3,c] Taylor A. Barnes,[1] Carlos H. Borca,[3] Roberto Di Remigio,[4,d] David L. Dotson,[5] Sebastian Ehlert,[6] Alexander G. Heide,[7] Michael F. Herbst,[8] Jan Hermann,[9] Colton B. Hicks,[10,11] Joshua T. Horton,[12] Adrian G. Hurtado,[13] Peter Kraus,[14] Holger Kruse,[15] Sebastian J. R. Lee,[16] Jonathon P. Misiewicz,[7,e] Levi N. Naden,[1] Farhad Ramezanghorbani,[17] Maximilian Scheurer,[18] Jeffrey B. Schriber,[3] Andrew C. Simmonett,[19] Johannes Steinmetzer,[20] Jeffrey R. Wagner,[5,21] Logan Ward,[22] Matthew Welborn,[1,a] Doaa Altarawy,[1,23] Jamshed Anwar,[12] John D. Chodera,[24] Andreas Dreuw,[18] Heather J. Kulik,[25] Fang Liu,[25,e] Todd J. Martínez,[10,11] Devin A. Matthews,[26,f] Henry F. Schaefer III,[7] Jiří Šponer,[15] Justin M. Turney,[7] Lee-Ping Wang,[27] Nuwan De Silva,[2,g] Rollin A. King,[28] John F. Stanton,[29] Mark S. Gordon,[30] Theresa L. Windus,[30] C. David Sherrill,[3] and Lori A. Burns[3,h]

AFFILIATIONS

[1] Molecular Sciences Software Institute, Blacksburg, Virginia 24060, USA
[2] Department of Chemistry, Iowa State University, Ames, Iowa 50011, USA
[3] Center for Computational Molecular Science and Technology, School of Chemistry and Biochemistry, and School of Computational Science and Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332, USA
[4] Department of Chemistry, Centre for Theoretical and Computational Chemistry, UiT, The Arctic University of Norway, N-9037 Tromsø, Norway
[5] Open Force Field Initiative, University of Colorado Boulder, Boulder, Colorado 80309, USA
[6] Mulliken Center for Theoretical Chemistry, Institut für Physikalische und Theoretische Chemie, Universität Bonn, Beringstraße 4, D-53115 Bonn, Germany
[7] Center for Computational Quantum Chemistry, University of Georgia, Athens, Georgia 30602, USA
[8] Applied and Computational Mathematics, RWTH Aachen University, Schinkelstr. 2, 52062 Aachen, Germany
[9] FU Berlin, Department of Mathematics and Computer Science, 14195 Berlin, Germany
[10] Department of Chemistry, Stanford University, Stanford, California 94305, USA
[11] SLAC National Accelerator Laboratory, Stanford University, Menlo Park, California 94025, USA
[12] Department of Chemistry, Lancaster University, Lancaster LA1 4YW, United Kingdom
[13] Institute for Advanced Computational Science, Stony Brook University, Stony Brook, New York 11794-5250, USA
[14] School of Molecular and Life Sciences, Curtin University, GPO Box U1987, Perth 6845, WA, Australia
[15] Institute of Biophysics of the Czech Academy of Sciences, Královopolská 135, 612 65 Brno, Czech Republic
[16] California Institute of Technology, Pasadena, California 91125, USA
[17] Department of Chemistry, University of Florida, Gainesville, Florida 32611, USA

[18]Interdisciplinary Center for Scientific Computing, Heidelberg University, Im Neuenheimer Feld 205, 69120 Heidelberg, Germany
[19]Laboratory of Computational Biology, National Institutes of Health–National Heart, Lung and Blood Institute, Bethesda, Maryland 20892, USA
[20]Institute of Physical Chemistry, Friedrich Schiller University Jena, Jena, Germany
[21]Department of Pharmaceutical Sciences, University of California Irvine, Irvine, California 92697, USA
[22]Data Science and Learning Division, Argonne National Laboratory, Lemont, Illinois 60439, USA
[23]Department of Computer and Systems Engineering, Alexandria University, Alexandria 21544, Egypt
[24]Computational and Systems Biology Program, Sloan Kettering Institute, Memorial Sloan Kettering Cancer Center, New York, New York 10065, USA
[25]Department of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA
[26]The Institute for Computational Engineering and Sciences, The University of Texas at Austin, Austin, Texas 78712, USA
[27]Department of Chemistry, University of California Davis, Davis, California 95616, USA
[28]Department of Chemistry, Bethel University, St. Paul, Minnesota 55112, USA
[29]Quantum Theory Project, The University of Florida, 2328 New Physics Building, Gainesville, Florida 32611-8435, USA
[30]Department of Chemistry and Ames Laboratory, Iowa State University, Ames, Iowa 50011, USA

[a)]**Present address:** Entos, Inc., 4470 W. Sunset Blvd. Suite 107 PMB 94758, Los Angeles, California 90027, USA.
[b)]**Present address:** Oden Institute for Computational Engineering and Sciences, University of Texas at Austin, Austin, Texas 78712, USA.
[c)]**Present address:** Chemistry Department, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia.
[d)]**Present address:** EuroCC National Competence Centre Sweden, Uppsala University, Lägerhyddsvägen 2 SE-752 37 Uppsala, Sweden.
[e)]**Present address:** Department of Chemistry, Emory University, Atlanta, Georgia 30322, USA.
[f)]**Present address:** Department of Chemistry, Southern Methodist University, Dallas, Texas 75275-0314, USA.
[g)]**Present address:** Department of Physical and Biological Sciences, Western New England University, Springfield, Massachusetts 01119, USA.
[h)]**Author to whom correspondence should be addressed:** lori.burns@gmail.com

ABSTRACT

Community efforts in the computational molecular sciences (CMS) are evolving toward modular, open, and interoperable interfaces that work with existing community codes to provide more functionality and composability than could be achieved with a single program. The Quantum Chemistry Common Driver and Databases (QCDB) project provides such capability through an application programming interface (API) that facilitates interoperability across multiple quantum chemistry software packages. In tandem with the Molecular Sciences Software Institute and their Quantum Chemistry Archive ecosystem, the unique functionalities of several CMS programs are integrated, including CFOUR, GAMESS, NWChem, OpenMM, Psi4, Qcore, TeraChem, and Turbomole, to provide common computational functions, i.e., energy, gradient, and Hessian computations as well as molecular properties such as atomic charges and vibrational frequency analysis. Both standard users and power users benefit from adopting these APIs as they lower the language barrier of input styles and enable a standard layout of variables and data. These designs allow end-to-end interoperable programming of complex computations and provide best practices options by default.

 https://doi.org/10.1063/5.0059356

I. INTRODUCTION

The number of quantum chemistry (QC) programs is continuously increasing, building a rich spectrum of capabilities where varied levels of accuracy, performance, distributed computing, graphics processing units (GPU)-acceleration, or licensing can be obtained. While this is generally beneficial to the end user, the diversity of custom input and output makes it difficult to switch between programs without learning the vagaries of each. Even the simplest research tasks using QC programs require mastering layers of expertise. On the input side, users must know what model chemistry will treat the molecular system of interest with adequate physics in tractable time, as well as pertinent modifications like density-fitting (DF), convergence, and active space, which are all questions of scientific expertise (I-a). [Labels of non-scientific (i.e., beyond I-a) input I-x or output O-x problems enumerated here are referenced by solutions in Sec. II.] They must know the names given by a QC program to the knobs that dial up the model chemistry and modifications, a question of domain-specific-language (DSL) expertise (here, "domain" is the QC software) (I-b). They would benefit from knowing the insider best-practice knobs that select the most efficient algorithms, approximations, and implementations specialized to the model chemistry, a question of program expertise (I-c). They must know the structure of the input specification by which the QC program receives

instruction, a question of formatting and DSL expertise (I-d). Last on the input side, they must know the dance of files, environment variables, and commands to launch the job, a question of program operational expertise (I-e).

On the output and analysis side, further skills are required to process the program-specific ASCII or structured data file. Users must know what strings in the output mark the desired result, a matter of DSL expertise (O-a). If the targeted quantity is not explicitly printed but is derivable, they must know the arithmetic or unit conversion, a question of QC expertise (O-b). If individual energies or derivatives are to be combined to create a more sophisticated model chemistry (e.g., basis set extrapolation,[1,2] focal-point methods,[3–5] G3 or HEAT procedures,[6,7] or empirical correction[8]) or for molecular systems decomposition or perturbation [e.g., many-body expansion (MBE), counterpoise procedure,[9] geometry optimization, or finite difference derivatives], users may be able to use routines built into QC programs (needing DSL expertise) but more generally must script the procedure themselves, requiring QC and programming expertise (O-c). More elaborately, they may want to combine the results with other programs—requiring recognizing and compensating for default knobs that render program results unmixable—a matter of QC and program expertise (I-f). Finally, users may hope that completed calculations can be stored and queried or even reused, matters of database expertise (O-d). Efforts to reduce non-scientific expertise burdens on the user have traditionally aggregated QC methods, geometry optimizers, and sundry procedures into vertically integrated "software silos" that, by increasing the DSL burden, risk locking users into one or a few programs. We pursue reducing the non-scientific expertise burdens on users by restructuring the QC software ecosystem while minimally disrupting longstanding, robust, and debugged computational molecular sciences (CMS) codes.

As a concrete example, in a high-accuracy spectroscopic application (see Sec. III), a user might want to include numerous small corrections, such as electron correlation effects beyond coupled-cluster (CC) through perturbative triples [CCSD(T)],[10] basis set extrapolation, relativistic corrections,[11] and Born–Oppenheimer (BO) diagonal corrections.[12,13] The best implementation of each of these terms is not necessarily found in a single QC program. Careful users can evaluate different terms using different programs through a post-processing script to obtain a focal-point energy, but more complex procedures such as geometry optimizations[14] are difficult due to tight coupling in QC programs that generally do not allow arbitrary gradients to be injected into the iterative optimizer.

Finally, in the emerging "data age" of computational chemistry, users increasingly want to treat QC results as a commodity, obtaining them on demand as part of complex workflows or generating datasets of millions of computations to use in force field (FF) parameterization, methodology assessment, machine learning (ML), or other data-driven pipelines. These users must be able to set up, execute, and extract computational results as easily as possible.

To address such challenges, the differing needs of workflows for uniform interaction with CMS codes have been separated into different layers of concern, resulting in the development of QCEngine and Quantum Chemistry Common Driver and Databases (QCDB).[15]

- Consider a new QC practitioner learning which density functional theory (DFT) program best suits the local hardware or accessing the latest ML FF for many molecules. Such users would benefit from a uniform application programming interface (API) to evaluate these diverse capabilities without requiring knowledge of the specifics of each program's DSL. QCEngine is designed to provide this uniform API and is an I/O runner around individual CMS codes' core single-point capabilities. QCEngine communicates through a JavaScript Object Notation (JSON) Schema,[16] denoted QCSchema, thus automatically generating program input files from a consistent and simple molecule and method specification.
- Next, consider the systematic study of dipole moments at different levels of theory from different programs or a FF developer training on the many symmetry-adapted perturbation theory (SAPT) component results over thousands of molecules. These applications would benefit from output layout uniformity and programmatic access to detailed results. QCEngine covers these cases by harvesting binary, structured, or text output into standardized QCSchema fields.
- Next, consider the maintainers of a CMS code whose users have been making the same formatting and incomplete input mistakes for the past decade and have been petitioning for quality-of-life features that would incur poor complexity-to-benefit ratio if implemented within the native framework and languages. These barriers to research would benefit from a shim layer in an easy and expressive language. QCDB provides a flexible input framework, helpful keyword validation, access to multijob procedures like MBE, and a place (besides documentation) to inject advice like context-dependent defaults.
- Now, consider a spectroscopist modeling a molecule with a composite method or the QC beginner hoping to avoid learning multiple DSLs. These circumstances would benefit from uniformity of input and results across programs. QCDB compensates for variable defaults and conventions so that multi-program model chemistries can be safely defined and simple methods accessed interchangeably.
- Finally, consider the experienced QC practitioner who writes inputs from memory and who turns keyword knobs as nimbly as organ stops but who would like to try another optimizer or an MBE procedure or not worry about capitalization and spaces today. This situation would benefit from a light hand in developing the QCSchema translation and common driver API so that existing expertise in direct interaction with CMS codes (DSL for keywords, for example) is applicable to these current projects.

In enabling uniformity at the input, output, and cross-program layers, both QCEngine and QCDB have striven to make their input predictable from customary input and to make customary output available.

Central to the ability of QCArchive[17] and QCDB to provide generic I/O, driver, and database interfaces to CMS codes is a common standard QC data format. Of course, to develop such a standard information exchange format for all QC programs and to encourage its adoption by QC packages is a difficult approach for a single research group, or even a handful of research groups, to successfully prescribe to a broad developer community. However, here the Molecular Sciences Software Institute (MolSSI),[18] funded by the

U. S. National Science Foundation, provides a unique opportunity to sponsor community discussions and to advocate for standards. Members of our collaborative team and the codes represented have worked closely with MolSSI on their development of a QCSCHEMA[19] for quantum chemistry information exchange, and we have adopted it for QCENGINE and QCDB.

There have been previous efforts to provide a unified interface to set up, drive, and analyze QC computations. For example, NEWTON-X[20] and FMS90[21–23] perform nonadiabatic dynamics computations using any of several QC programs. The Quantum Thermochemistry Calculator (QTC)[24] interfaces to a handful of QC programs to provide unified thermochemistry analysis functions independent of the QC data source. Especially tailored to deal with excited state optimizations is PYSISYPHUS, an external optimizer that localizes stationary points on potential energy surfaces by means of intrinsic reaction coordinate (IRC) integration, chain-of-state optimization, and surface walking for several QC codes through a uniform interface.[25] Among more general-purpose programs, CUBY[26,27] is a uniform driver and workflow manager that works with multiple QC and force field tools. CUBY allows the combination of methods across its interfaced programs and provides mixed quantum mechanics/molecular mechanics (QM/MM) and molecular dynamics capabilities. The WEBMO project is another that drives several QC programs as backends from a largely unified web portal frontend.[28] Another popular tool is the Atomic Simulation Environment (ASE),[29] which provides a Python interface to more than 40 QC or force field codes, along with drivers for geometry optimization and transition state searching with the nudged elastic band method and analysis and visualization functions. A recipes collection (ASR)[30] supplies further spectroscopy and analysis tools. ASE and ASR are focused on solid-state computations; while molecular computations are also possible, they do not provide the level of detail required for the majority of quantum chemistry workflows. Compared to ASE, QCDB is more focused on high-accuracy quantum chemistry (providing, for example, built-in support for focal-point methods). Newer entrants to the field of computational chemistry workflow tools at the scope of QCARCHIVE (rather than the narrower modular components QCENGINE and QCDB discussed here) include AIIDA,[31,32] which at present is materials focused, and CHEMSHELL,[33] which focuses on multiscale simulations. By interfacing with QCARCHIVE, QCDB can also focus on high-throughput quantum chemistry and on creating large databases for force field parameterization and machine-learning purposes. Although not focused on running CMS codes, the CCLIB[34,35] and HORTON[36] projects also have extensive capabilities to regularize output and post-processing.

We describe the modular software built to facilitate interoperability, the community QC codes, and the technical challenges associated with an interoperability project in Sec. II. An example application demonstrating the use of multiple QC codes to perform very high accuracy computations of spectroscopic constants of some diatomic molecules is presented in Sec. III.

II. FEATURES AND DESIGN PHILOSOPHY

Discussed are the present software projects and their place in the CMS ecosystem in Sec. II A, interfaced software providing single-point energies and properties in Sec. II B, interfaced and built-in software providing more complex procedures in Sec. II C, how these are all linked by a common driver in Sec. II D, and further details about implementing interoperability in Sec. II E.

A. QCSCHEMA and the quantum chemistry software ecosystem

The modular software components in our layered approach to QC interoperability and high-throughput computing are shown in Fig. 1. All are open-source projects, and community feedback and contributions through GitHub are welcome (links at Sec. V; QCENGINE documentation includes the general process for adding a new QC program). The QCSCHEMA[19] definitions layer is foundational and encodes the community-developed data layouts and model descriptions useable in any language, from C++ to Rust to JavaScript to Fortran. Above that is the QCELEMENTAL[37] data and models layer that implements QCSCHEMA and imposes a Python language restriction to gain sophisticated validation and feature-rich models. Next is the QCENGINE[38] execution layer that adapts CMS codes for standardized QCSCHEMA communication and imposes an execution environment restriction to gain easy access to many programs. Last is the QCFRACTAL[39] batch execution and database layer that imposes some calculation flexibility restrictions to gain multi-site distributed compute orchestration and provide structured-data storage and querying capabilities. [This layer, beyond the scope of the present work, addresses (O-d).] Together these compose the QCARCHIVE INFRASTRUCTURE, the Python software stack that backs the MolSSI QCARCHIVE project.[17,40] Enhancing QCENGINE is the QCDB[41] interoperability layer that imposes feature-registration and cross-program defaults restrictions to gain input uniformity and multi-program workflows.

QCELEMENTAL[37] (see Fig. 1) provides data and utilities (like a QCSCHEMA implementation) useable by all QC packages. For data, it exposes NIST Periodic Table and CODATA physical constants through a lightweight API and provides internally consistent unit conversion aided by the external module PINT.[42] QCELEMENTAL supports multiple dataset versions for CODATA and for properties such as covalent and van der Waals radii. Additionally, QCELEMENTAL provides a Python reference implementation for the MolSSI QCSCHEMA data layouts, including `Molecule` (example is given in Snippet 2), job input specification `AtomicInput` [examples at Figs. 2(b)–2(d)], and job output record `AtomicResult`. In addition to enforcing the basic key/value data layout inherent to a schema, QCELEMENTAL uses the external module PYDANTIC[43] to collocate physics validation, serialization routines, extra helper functions (like `Molecule` parsing, alignment, and output formatting), and schema generation into a *model* for the QCSCHEMA. Historically, many QCELEMENTAL capabilities were developed for QCDB in PSI4 and then refactored into QCELEMENTAL for broader community accessibility free from PSI4 and compiled-language dependence. QCENGINE and QCDB use all the QCELEMENTAL capabilities mentioned, particularly for QCSCHEMA communication and for uniform treatment of fragmented, ghosted, and mixed-basis molecules across differing QC program features.

QCENGINE[38] provides a uniform execution interface whereby community CMS codes consume QCSCHEMA `AtomicInputs` and emit `AtomicResults` via adaptors, called `ProgramHarnesses`. Depending on the degree of programmatic access a QC package

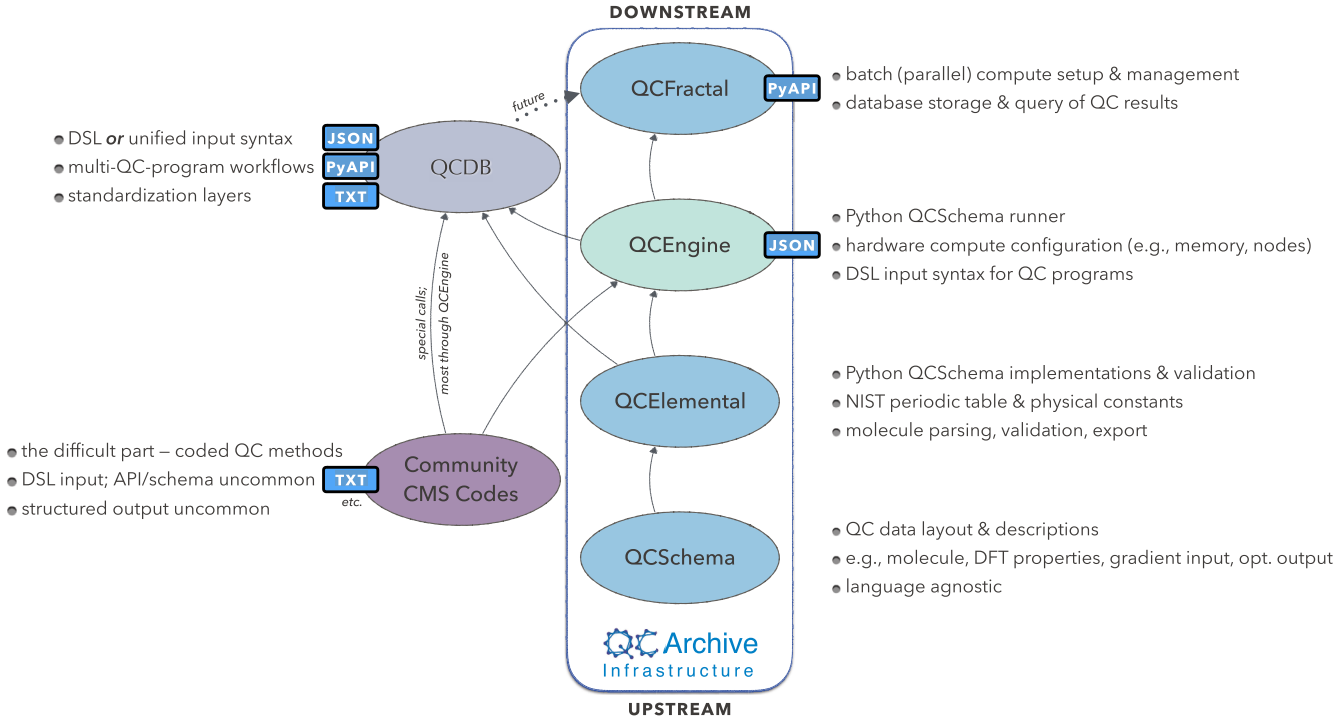

FIG. 1. Modular ecosystem around QCENGINE and QCDB. QCENGINE is the central, QCSCHEMA-based QC program runner in the QCARCHIVE INFRASTRUCTURE software stack, while QCDB adds additional interoperability features atop it. User input routes to QC computations are shown as one or more turquoise boxes—"TXT" for a command-line interface, "PyAPI" for an interactive application programming interface in Python, or "JSON" for single-command QCSCHEMA communication through command-line or Python.

provides, the `ProgramHarness` may be simple, as for a package that already provides a QCSCHEMA interface; moderate, as for a package that supports a Python API or has serialized output, be it binary, Extensible Markup Language (XML), or JSON; or involved, as for an executable with ASCII I/O; further details may be found in Sec. II E 10. A typical `ProgramHarness` consists of taking an `AtomicInput`, translating it into input file(s) and execution conditions, executing it, collecting all useful output, parsing the results into an `AtomicResult`, and returning it to the user. A `ProgramHarness` is written to cover analytic single-point computations, namely, energies, gradients, Hessians, and properties, as discussed further in Sec. II B. Adaptors for more complicated actions are classified as `ProcedureHarnesses` and are discussed in Sec. II C. QCENGINE additionally collects runtime data such as elapsed time, the hardware architecture of the host machine, memory consumption of the job, software environment details, and execution provenance (e.g., program, version, and module). As suggested by Fig. 1, adaptors written in QCDB have been migrated to QCENGINE so that both projects access more QC codes and share the maintenance and development burden.

QCDB[41] supplements QCENGINE's program and procedure capabilities with interoperability-enhanced `ProgramHarnesses` and multi-program procedures; furthermore, it links QCENGINE calls into an interactive driver interface. From the user's viewpoint, this layered approach to uniform QC computation is shown in Fig. 2 by an open-shell CCSD single-point energy. Running a QC code directly, as in Fig. 2(a), requires considerable DSL knowledge for method, basis, and keywords, not to mention details of layout and execution; essentially only the geometry (black text) is uniform. By molding the text inputs of Fig. 2(a) into the QCSCHEMA data layout Fig. 2(b), QCENGINE unifies the gray-shaded fields but still requires DSL from multiple codes. QCDB imposes more dependencies, like its own basis set library and utilities, to allow uniform basis specification and molecule symmetry as in Fig. 2(c). By imposing keyword registration and precedence logic, QCDB can provide the uniform and single-DSL input of Fig. 2(d). In practice, QCDB harnesses are minimal wrappers around QCENGINE harnesses.

By choosing an entry point (software component in Fig. 1) and interface (CLI, Python API, JSON), external projects can satisfy a number of interoperability use cases: convention for data layout (stop after QCSCHEMA), molecule string parsing (stop after QCELEMENTAL), uniform CMS execution (stop after QCENGINE), tolerant Python interface to single venerable CMS code (QCDB), or multicode workflows (QCDB).

B. Program capabilities

For several community codes or *programs* [Fig. 3(i); not comprehensive] capable of computing analytic energies, gradients,

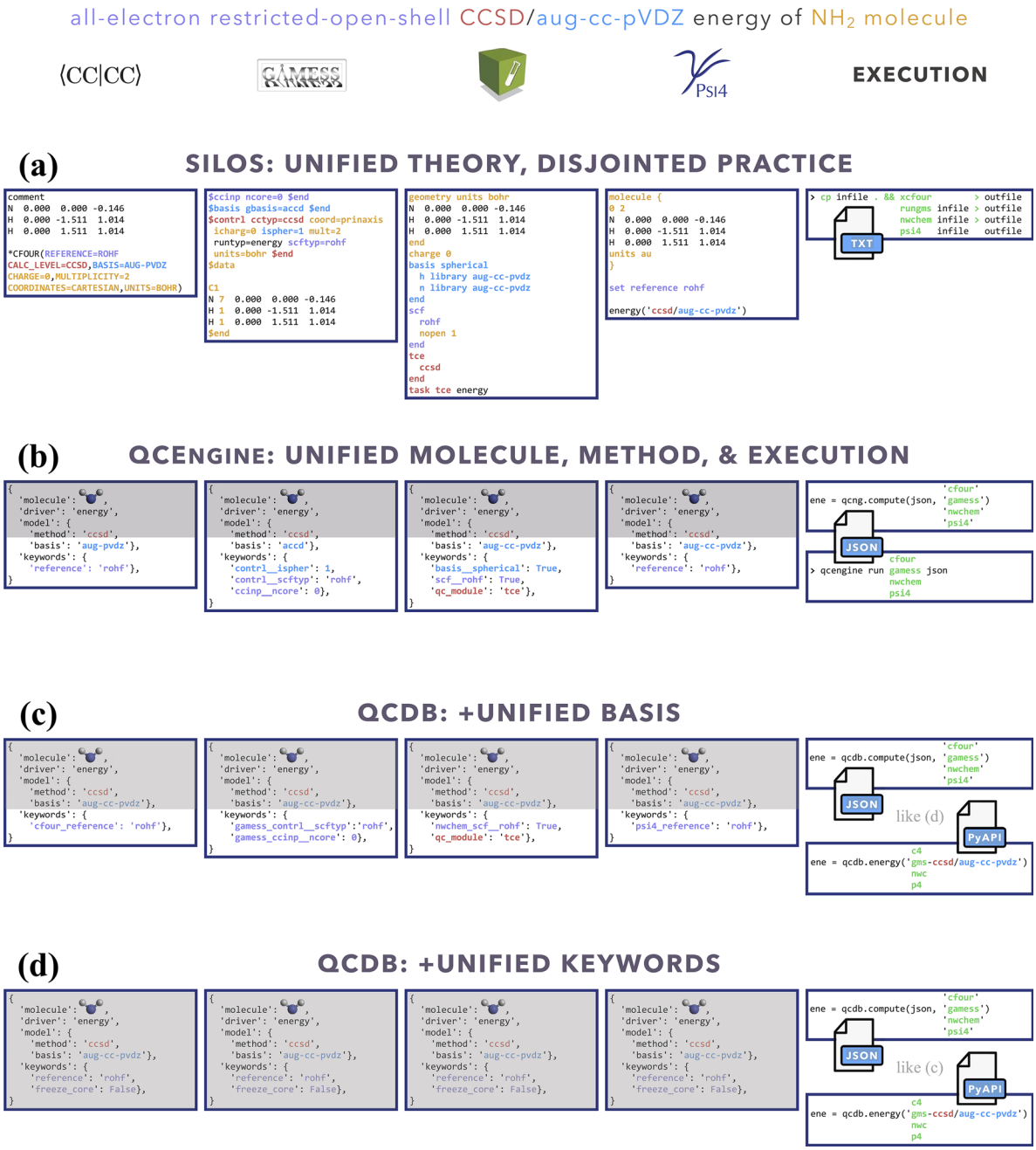

FIG. 2. Degrees of unifying access to quantum chemical calculations illustrated through an open-shell CCSD energy computation. Black text and gray shading are aspects *not* requiring user knowledge of multiple DSLs. See penultimate paragraph of Sec. II A for discussion.

or Hessians, the authors have written QCSCHEMA adaptors for QCENGINE known as `ProgramHarnesses` [Fig. 3(ii)]. The primary returns can be full scalars or arrays, as for most QC methods, or partial, as for dispersion corrections. So long as program communication fits into the `AtomicResult` data layout, semi-empirical and molecular mechanics programs can also formulate QCENGINE adaptors. A summary of interfaced codes can be seen in Table I. QCDB asserts greater control over codes to assure consistent output values, so its capabilities are centered on CFOUR, GAMESS, NWCHEM, PSI4, and select partial calculators

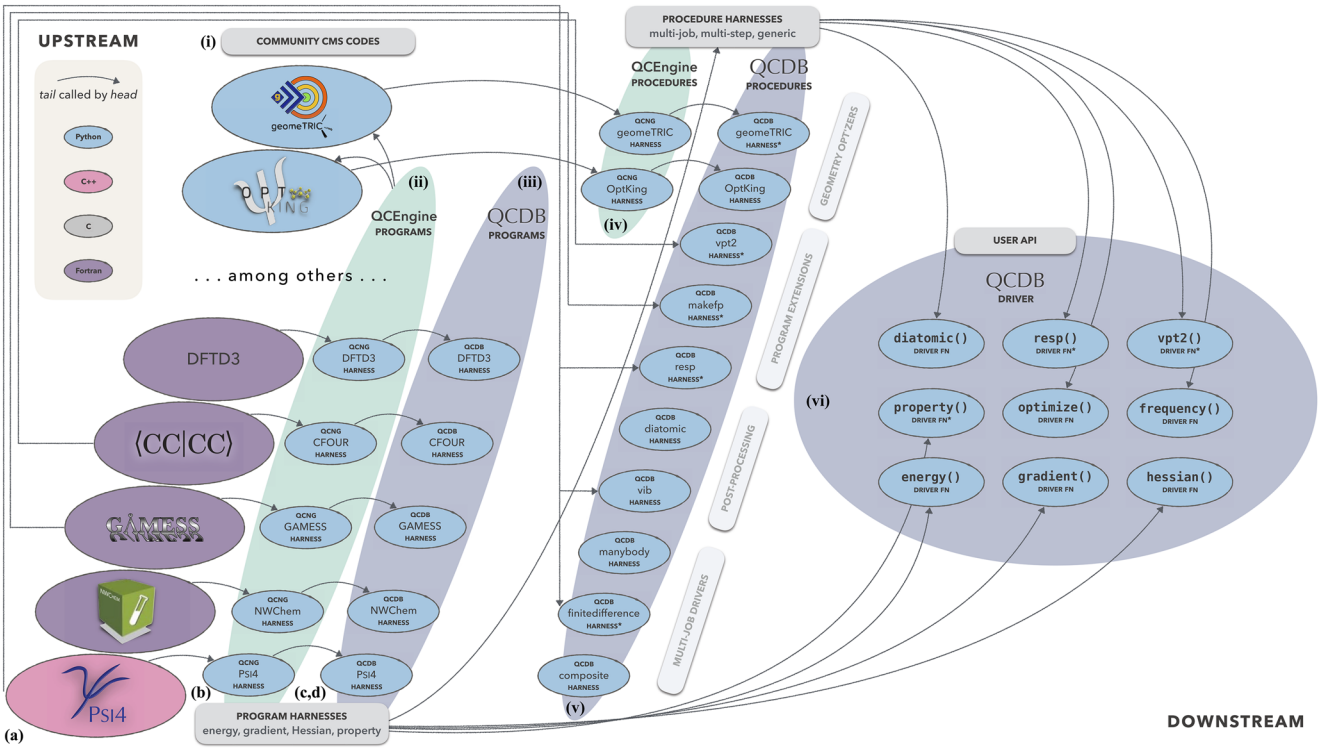

FIG. 3. Layout and access pattern between selected existing and planned (marked by *) community quantum chemistry codes, QCENGINE, and QCDB. Community codes (i) in a variety of languages are wrapped in QCSCHEMA input/output by a QCENGINE harness (ii) and (iv), which may be light (if the code has an API or structured output) or heavy (if only text output available). The QCDB harnesses (iii) and (v) add unifying and ease-of-use layers atop the QCENGINE calls. Whereas analytic energies and derivatives are classified as *programs* (ii) and (iii) and call QC codes directly, multi-stage and post-processing jobs are written as *procedures* (iv) and (v) for composability and distributability and call programs in turn. The QCDB driver provides API access to both sets. Labels (a)–(d) correspond to the stages of unified input in Fig. 2.

[Fig. 3(iii)]. Note that output harvesting capabilities (results available programmatically as opposed to text files) may lag behind those for input execution. A test suite that ensures matching values can be extracted from different programs has been established for both QCENGINE and QCDB to document differing conventions (e.g., canonicalization for ROHF CC, all-electron vs frozen-core). Uncovered incorrect values or missing properties have been reported to the code developers for further investigation.

1. ADCC

The interface to ADCC allows for computations of excited states based on the algebraic-diagrammatic construction (ADC) scheme for the polarization propagator. Several methods are available, including ADC(2), ADC(2)-x, and ADC(3), together with the respective core-valence separation (CVS) and spin-flip variants. For all aforementioned methods, excitation energies and properties are accessible. The interface uses PSI4 to compute the SCF reference state first and then calls adcc via its Python API. A minimum adcc v0.15.1 is required.

2. CFOUR

Many CFOUR features are available to both QCENGINE and QCDB, including most ground-state many-body perturbation theory and coupled-cluster energies, gradients, and Hessians: Hartree–Fock, MP2, MP3, MP4, CCSD, CCSD(T) with RHF, UHF, and ROHF references. Excited states are available for running but not parsing. Special features include CC with quadruple excitations through the NCC module, the ability to compute the diagonal Born–Oppenheimer correction using coupled-cluster theory, and, after revision, second-order vibrational perturbation theory (VPT2) (see Sec. II C 6). The interface generates text input and collects mixed text and binary output. A minimum CFOUR v2.0 is required.

3. GAMESS

The GAMESS interface for QCENGINE and QCDB provides Hartree–Fock, DFT, MP2, and coupled-cluster methods. Special features include full configuration interaction. In the future, the GAMESS interface will also provide effective fragment potential (EFP) capability through potential file generation (see Sec. II C 7)

TABLE I. Interfaced programs in QCEngine and QCDB. For each, availability of one or more methods for energy (E), gradient (G), and Hessian (H) is shown, as well as collection of properties (e.g., one-electron energy or dipole) and wavefunction quantities (e.g., number of basis functions and orbitals). Symbols are present (✓), absent (✗), or inapplicable (). Non-QC programs are not suitable for QCDB. Program I/O is handled primarily through QCSchema API (Q), API (A), structured XML, JSON, binary (S), or text (T).

CMS program	QCEngine					QCDB			Cite	I/O
	E	G	H	Prop.	Wfn	E	G	H		
Quantum chemistry										
ADCC	✓	✗	✗	✓	✗	✗	✗	✗	44 and 45	A
CFOUR	✓	✓	✓	✓	✗	✓	✓	✓	46	TS
GAMESS	✓	✓	✓	✓	✗	✓	✓	✓	47	T
Molpro	✓	✓	✗	✓	✗	✗	✗	✗	48 and 49	S
MRChem	✓	✗	✗	✓	✗	✗	✗	✗	50 and 51	S
NWChem	✓	✓	✓	✓	✗	✓	✓	✓	52	T
Psi4	✓	✓	✓	✓	✓	✓	✓	✓	53	Q
Q-Chem	✓	✓	✓	✗	✗	✗	✗	✗	54	TS
Qcore	✓	✓	✓	✗	✓	✗	✗	✗	55	S
TeraChem	✓	✓	✗	✓	✗	✗	✗	✗	56 and 57	Q,T
Turbomole	✓	✓	✓	✗	✗	✗	✗	✗	58 and 59	T
Semi-empirical										
MOPAC	✓	✓	✗	✓	✗				60	T
xtb	✓	✓	✗	✓	✗				61	Q
Molecular mechanics										
OpenMM	✓	✓	✗	✓					62	A
RDKit	✓	✓	✗	✓					63	A
Analytical corrections										
DFTD3	✓	✓	✗	✗		✓	✓	✗	8 and 64	T
DFTD4	✓	✓	✗	✗		✗	✗	✗	65 and 66	Q
gCP	✓	✓	✗	✗		✗	✗	✗	67 and 68	T
MP2D	✓	✓	✗	✗		✓	✓	✗	69 and 70	T
Machine learning inference										
TorchANI	✓	✓	✓	✓					71–73	A

and running pure EFP calculations on molecular clusters, energy ("`gms-efp`"). A particular complication for GAMESS is the controlled molecule and custom basis syntax, which led to QCDB feeding only symmetry-unique atoms and their full basis sets into the GAMESS input file. As QCEngine does not have symmetry capabilities, QCEngine-based GAMESS calculations are restricted to C_1. The interface generates text input and collects text output. The harness has been tested with the GAMESS 2017 R1 version.

4. Molpro

Energies and gradients are available in QCEngine from Hartree–Fock, DFT, MP2, CCSD, and CCSD(T) levels of theory, including some local methods. The interface generates text input and collects XML output. A minimum Molpro v2018.1 is required.

5. MRChem

Thanks to a harness to the MRChem software package, quasi-exact energies and selected properties in the multiwavelet, multiresolution basis are available with QCEngine. MRChem provides an efficient implementation for Hartree–Fock and DFT. Electric dipoles, quadrupoles, static and frequency-dependent polarizabilities, magnetizabilities, and NMR shielding constants are available. At variance with GTO-based quantum chemical software packages, the basis used in MRChem is adaptively refined: thanks to the multiwavelet framework, these results are exact to within the user-requested precision.[74] As a practical consequence, only the method keyword is required to define an input model to MRChem. JSON files are used to handle communication between QCEngine and MRChem. The harness can leverage the hybrid MPI/OpenMP parallelization of MRChem, provided suitable resources are available. A minimum MRChem v1.0.0 is required.

6. NWChem

The NWChem interface for QCEngine and QCDB provides a large selection of the quantum mechanical methods available, including Hartree–Fock, DFT, MP2, and coupled-cluster methods [both the code automatically derived and implemented with the Tensor Contraction Engine[75] (TCE) and the hand-coded implementations, where available]. Additional calculations available in the `TCE` include configuration interaction through single, doubles, triples, and quadruples level of theory and MBPT methods through the fourth order. Special features include CCSDTQ energies, excited states through equation of motion (EOM) coupled-cluster energies, and relativistic approximations. The interface generates text input and collects text output. The harness has been tested with NWChem v6.6 and v7.0.

7. Psi4

Essentially, all Psi4 features are available to QCEngine and QCDB, as Psi4 communicates natively in QCSchema (`psi4 - qcschema in.json`) and QCDB began as the Psi4 driver. These include conventional and density-fitted Hartree–Fock, DFT, MP2, and coupled-cluster methods. Special features are symmetry-adapted perturbation theory, coupled-cluster response properties, density-fitted CCSD(T) gradients, and optimized-orbital MP2, MP2.5, and MP3 energies and gradients. Wavefunction information is returned in QCSchema format. The interface generates JSON (QCSchema) input and collects JSON output. A minimum Psi4 v1.3 is required for QCEngine and v1.4 for QCDB.

8. Q-Chem

Energies, gradients, Hessians, and some properties are available in QCEngine at the SCF (Hartree–Fock and tens of DFT functionals) and MP2 levels (both conventional and density-fitted). The interface generates text input and collects mixed text and binary output. A minimum Q-Chem v5.1 is required.

9. Qcore

Energies, gradients, and Hessians are available in QCEngine from Hartree–Fock, DFT, and extended tight-binding (xTB). Qcore along with Psi4 are the two programs that can return wavefunction information in QCSchema. The interface generates JSON input and collects JSON output. A minimum of Qcore v0.7.1 is required.

10. TeraChem

TeraChem features two modes for driving computations via QCEngine: a standard text interface and a typed Protocol Buffers[76] interface. The former generates text input and collects text output to provide energies and gradients from Hartree–Fock and DFT levels of theory. A minimum TeraChem v1.5 is required.

TeraChem's Protocol Buffers (TCPB) server[57] interface offers a second way to drive computations using QCEngine. It provides energies and gradients from Hartree–Fock and DFT levels of theory, molecular properties including dipoles, charges, and spins, and limited wavefunction data including alpha- and beta-spin orbitals and orbital occupations. The TCPB interface also accelerates calculations by performing GPU initialization routines once at server startup. As a result, subsequent computations can begin instantaneously, thereby providing substantial speed-up for small systems (~ 10 heavy atoms) and minor speed-up for medium systems (~100 atoms).[77] The TCPB interface requires the installation of an additional Python package tcpb[78] minimum v0.7.0 to power the QCEngine integration. Subsequent updates to the tcpb package will expand the set of properties and wavefunction data available from TeraChem via QCEngine.

11. Turbomole

Energies, gradients, and Hessians are available in QCEngine for Hartree–Fock, many DFT functionals, and define-fitted MP2, MP3, MP4, and CC2. Turbomole's interactive `define` function for processing input proved an extra challenge to integrate with QCSchema. The interface generates interactive text input and collects text output. The harness has been tested with Turbomole v7.3 and v7.4.

12. XTB

The interface uses the Python API of XTB, which provides QCSchema support, to generate JSON (QCSchema) input and collect JSON output. A minimum of XTB v6.3 is required.

13. dftd3 and dftd4

A Python API to Grimme's dftd3 executable for computing variants of -D2 and -D3 for arbitrary QCSchema `Molecule` with automatic or custom parameter sets has been available in Psi4 for several years.[8,79,80] This has been adapted as a ProgramHarness for QCEngine and QCDB. The interface generates text input and collects text output. A minimum of dftd3 v3.2.1 is required.

For the separate dftd4 software, the interface uses the Python API, which provides QCSchema support, to generate JSON (QCSchema) input and collect JSON output. A minimum of dftd4 v3.1 is required.

14. gCP

Energies and gradients are available for the geometrical counterpoise correction GCP program developed by Kruse and Grimme that corrects the inter- and intramolecular basis set superposition error (BSSE) in Hartree–Fock and DFT calculations.[68] It also offers the GCP-part of the "3c" correction used in composite methods like HF-3c or PBEh-3c.[81] The interface generates text input and collects text output. The harness was tested with gCP v2.02.

C. Procedure capabilities

Whenever a quantum chemistry work sequence takes in QC-program-agnostic energies, gradients, Hessians, or properties (i.e., `AtomicResults`) but requires multiple ones (e.g., a finite difference derivative) or needs additional software [e.g., EFP potentials or symmetry-adapted linear combination (SALC) coordinates] or needs to take action in multiple stages (e.g., a geometry optimizer) or could combine `AtomicResults` from different programs (e.g., a composite method), it is classified in QCEngine or QCDB as a *procedure* [see Fig. 3(iv-v)]. Procedures are implemented in a `ProcedureHarness` to facilitate modularity and address O-c. Because procedures act upon generalized quantities, any code interfaced with QCEngine or QCDB gets all of the applicable procedures "for free." Together, programs and procedures are elements that can be composed into workflows both simple (e.g., opt + freq + vib) or complex as in Sec. III.

Presently available in QCENGINE are the GEOMETRIC, PYBERNY, and (Python) OPTKING geometry optimizers, the first of which has been used extensively (>380k optimizations) by the Open Force Field[82] community. Presently available or anticipated (*) for QCDB are the `Composite`, `FiniteDifference`,* `ManyBody`, `diatomic`, and `vib` routines inherited from the PSI4 recursive driver.[14] The PSI4 OPTKING geometry optimizer, written in C++, has been redeveloped in Python as a more versatile tool for future development and with the independence suitable for QCDB, while RESP* and CRYSTALATTE* have been expanded from PSI4 to work with QCDB. Procedures `makefp`* and `vpt2`* make use of specially extractable features from GAMESS and CFOUR, respectively, and require installation of the parent code. Similarly, FINDIF retains for the short term a dependence on PSI4. Note that the full capabilities from proven software components that were once or are presently partially or fully interfaced are in the procedure descriptions below. Procedures in QCENGINE and QCDB have passed through the proof-of-principle stage and are presently being reworked and expanded into the below forms; current availability is limited.

1. Geometry optimizers

To be used by QCENGINE or QCDB, a geometry optimizer must be able to take an input geometry in Cartesian coordinates and to take an arbitrarily sourced gradient and produce a next-candidate geometry displacement rather than be in control of both gradient and geometry-step stages. Regrettably, this eliminates most optimizers embedded in QC programs. Some alternatives are Wang's GEOMETRIC project,[83,84] which uses the TRIC coordinate system to specialize in interfragment and constrained optimizations, King's OPTKING,[85] which is a conventional IRC- and TS-capable QC optimizer, and Hermann's PYBERNY,[86] also a QC-focused optimizer. OPTKING can apply flexible convergence criteria including those related to energy change and the maximum or root-mean-square of the gradient or displacement, and it has the most common settings for many embedded/native optimizers conveniently accessible as keywords. QCENGINE presently has available GEOMETRIC, PYBERNY, and the `Python` OPTKING, while QCDB only has the original C++ OPTKING. After a planned driver update, all three Python optimizers will work with QCENGINE and hence with QCDB. All optimizers communicate through schema, in particular, a QCSCHEMA `OptimizationInput` that contains an ordinary `AtomicInput` as template for the gradient engine. Optimizations are called through QCENGINE using `qcng.compute_procedure({"input_molecule":...,"keywords":{"program":"gamess"},"input_specification":{"model":{"method":"mp2","basis":"6-31G"}}}, "geomeTRIC")` or `qcdb.optking("gms-mp2/6-31G")`, where the latter can take as model chemistry any sensible combination of other procedures (i.e., `qcdb.optking("gms-mp2/[23]zapa-nr", bsse_type="cp")`).

2. vib: Harmonic vibrational analysis

The harmonic vibrational analysis routine is automatically run after any `qcdb.frequency()` computation.[87] Taking in a Hessian matrix, the molecule, basis set information, and optional dipole derivatives, `vib()` performs the usual solution of whole or partial Hessians into normal modes and frequencies, reduced masses, turning points, and infrared intensities, all returned in schema. Other features include rotation-translation space projection, isotopic substitution analysis, Molden output, and a full thermochemical report incorporating the best features of several QC programs' vibrational output.

3. FiniteDifference: Derivatives

As QCENGINE and QCDB are focused on interfacing QC programs' analytic quantum chemical methods or unique features, user calls for non-analytic derivatives in QCDB are by default routed through the finite difference procedure.[87] This procedure (originally from PSI4) performs three- or five-point stencils for gradients and Hessians (full or partial), communicates through schema, and is parallelism-ready. The alternative of letting the internal finite difference of a QC program run and then parsing output files for multiple energies or gradients has been implemented in some cases, but this is not preferred (nor for internal geometry optimization).

4. Composite: Composite method and basis extrapolation treatments

Whenever an additive model chemistry is designated that involves differences of method (i.e., a focal point analysis or "delta" correction), basis [i.e., a complete basis set (CBS) extrapolation], keywords (e.g., all-electron minus frozen-core), or any combination thereof, the Composite procedure can encode it. Here, one can mix QC programs to perform conventional coupled cluster with CFOUR and DF-MP2 with PSI4, for example. Implementing new basis extrapolation formulas is simple, and it works on gradients and Hessians, as well as energies. If a subsidiary method energy can be obtained in the course of a target method, the procedure will recognize and avoid the unnecessary calculation (thus a TQ MP2 correlation energy extrapolation atop a DTQ HF energy will do 3, not 5, jobs). Input specification can be through API, schema, or strings (a user-friendly example is in the final paragraph of Sec. II E 5). All Composite communication is through schema, and the procedure is parallelism-ready.

5. ManyBody: Fragmentation and many-body approaches

All fragmentation and basis set superposition error (BSSE) treatments are collected into the ManyBody wrapper for many-body expansion (MBE) inherited from PSI4. The fragmentation pattern known from the QCSCHEMA `Molecule` is applied to determine the degree of decomposition into monomers, dimers, etc., up to the full molecule, or the user can set the `max_nbody` level. Total quantities (energy, gradient, or Hessian) and interaction quantities are accessible through uncounterpoise (noCP), couterpoise (CP), and Valiron–Mayer functional counterpoise (VMFC) schemes.[9,88,89] Geometry optimization with many-body-adapted quantities is also available. The wrapper can act on uniform single-method quantities or apply different model chemistries to each expansion level or interface with `Composite` or `FiniteDifference` results or both. All `ManyBody` communication is through schema, and the procedure is parallelism-ready.

6. vpt2: Anharmonic vibrational analysis

Anharmonic vibrational analysis has long been a feature of CFOUR. It requires a high-quality harmonic frequency procedure as

input. It then performs further Hessian computations at geometry displacements along the normal coordinates. These are then combined into a third-order and partial fourth-order potential followed by vibrational analysis. Although many analytic Hessians are available in CFOUR itself, the `qcdb.vpt2()` procedure focuses on the formulation through analytic *gradients*, as being suited to distributed computing and generalization to program-generic gradients. Thus, CFOUR is a helper program that, with the QCDB procedure, can perform anharmonic analyses of, for example, CCSD (from CFOUR gradients called through QCDB), DFT (from another QC program's gradients), or CBS (that produces a generalized gradient). All `qcdb.vpt2()` communication is through schema, and the procedure is parallelism-ready.

A complication is that the `vpt2()` procedure is essentially a series of invocations of CFOUR subcommands like `xcubic`, which expect files in native `JOBARC` form with energies, dipoles, and gradients. To accommodate this, QCDB uses Python modules to write imitations of the native files in string representations of binary form, which is lossless. Hence, a PSI4 DFT gradient is represented as a `JOBARC` to pass through the CFOUR mechanisms.

7. makefp: EFP library generation

The two engines for computing EFP interactions, LIBEFP[90,91] and GAMESS,[47] use the same parameter file for storing the EFP potential at a given basis set and monomer geometry. Only GAMESS can generate that file, and the routine has been wrapped by QCDB for access through `qcdb.makefp()`. The resulting `.efp` file contents are returned in the QCSCHEMA output and so are available for writing to a personal library or to feed to subsequent `qcdb.energy("gms-efp")` (or `"lefp-efp"` or `"p4-efp"`) calls to determine non-covalent interactions between EFP fragments. Certain EFP integrations await expansion of QCSCHEMA `Molecule`.

8. Diatomic: Spectroscopic constants

The electronic potential analysis for diatomic molecules has long been encoded in PSI4 as a post-processing procedure from a list of electronic energies along the interatomic coordinate. This has been reworked as a procedure and is demonstrated in Sec. III.

9. RESP: Charge fitting

The restrained electrostatic potential (RESP) charge model[92] is obtained by an iterative fitting of the electrostatic potential emerging from QC calculations on one or several conformers of a molecule to a classical point-charge potential. An existing RESP plugin[93,94] drives the property calculations with PSI4, and this has been expanded to alternately draw from GAMESS using the QCDB API.

10. CrystaLattE: Crystal lattice energies

The process of estimating the lattice energy of a molecular monocrystal via the many-body expansion is encoded in the CRYSTALATTE software.[95,96] Starting with extracting a subsample from a `cif` file, the program handles fragmentation into dimers, trimers, etc., identifies unique N-mers, prepares QC inputs, and keeps track of many-body results into final quantities. Although the thousands of component calculations mean that it will only become practical after QCDB upgrades to the distributed driver (see Sec. II D), CRYSTALATTE is ready to be integrated in serial mode in QCDB.

D. QCDB common driver

The driver component of QCDB [Fig. 3(vi)] is the fairly lightweight coordinator code that (1) facilitates the interactive API of `set_molecule, set_keywords, energy ("nwc-b3lyp/6-31g*"), print(variable("b3lyp dipole"))` rather than communicating through QCSCHEMA; (2) imposes cross-QC-program suggestions like tightening convergence for higher derivatives or for finite difference; and (3) weaves together procedures and programs so that `optimize("mp6")` commences finite difference or `energy("ccsd/cc-pv[tq]z", bsse_type="vmfc")` runs `ManyBody, Composite,` and program harnesses in the right sequence. The driver is primarily concerned with processing user-friendly input ["User API" in Fig. 3(vi)] into QCSCHEMA as directly as possible and then routing it into a program harness [Fig. 3(iii) for analytic single-points] or through procedures [Fig. 3(v)] on their way to program harnesses (e.g., for `Composite, FiniteDifference`) or through procedures after program harnesses [e.g., for `resp(), vib()`]. In order to make good use of the QCDB common driver, a QC program must register capabilities and information. These include the available analytic methods (for appropriate use of finite difference), insider best-practice options from the program's developers (see Sec. II E 9), and all keywords and their defaults (for flexible and informative keyword validation through Python).

The common driver is based upon the PSI4 v1.0 *recursive* driver described in Ref. 14 that unifies many complex treatments (e.g., MBE and CBS) into a few user-facing functions that focus on *what*, not *how*. After polishing in PSI4 v1.5, a new *distributed* driver with the same interface but tuned to QCSCHEMA communication and embarrassingly parallel execution through QCARCHIVE INFRASTRUCTURE will be substituted. See Sec. IV and Fig. 2 of Ref. 53 for details.

E. Technical aspects to interoperability

Details of specifying and running QC computations, particularly arbitrating the expression of QCSCHEMA by QCENGINE and QCDB, are collected below. Readers who prefer a software overview should proceed to Sec. III. Symbols like (I-b) mark strategies for overcoming or unifying the expertise barriers to using QC programs enumerated in the initial paragraphs of Sec. I.

1. Memory

User specification of memory resources is managed by QCENGINE and is outside the QCSCHEMA. By default, the job is given all of the compute node's memory (less some buffer). If user-specified, input units are in GiB, e.g., `qcdb` or `qcng.compute(..., local_options={"memory": 10})` (I-b). In either case, the memory quantity is translated into DSL keyword names such as `memory_size` and `mem_unit` for CFOUR. Because QCENGINE exercises total control over memory, any specification misplaced as a keyword into QCSCHEMA is ignored and overwritten in QCENGINE or raises an error if conflicting in QCDB. An exception is cases like

NWChem, where aggregated memory is managed by QCEngine but distribution between heap, stack, and global is editable through keywords (e.g., `memory__total` or `memory__stack`).

2. Disk

The working directory and execution environment are also governed by QCEngine, and user modifications are outside QCSchema. Each job is run in a quarantined scratch directory created for it and populated by input and any auxiliary files. Execution occurs through Python `subprocess` (or less often through Python API). Output files and any program-specific files in text or binary format (including the generated input) are collected and returned in QCSchema fields before scratch directory deletion (I-e).

3. Parallelism

The execution flags or environment variables that control CMS program parallelism and their single- or multi-node capabilities are built into their respective QCEngine harnesses. A job gets the full single-node resources (max cores and near-max memory) assigned to it by default; multinode execution (only for NWChem at present) requires explicit specification. Assigning instead an *optimal* portion of the full resources on the basis of method and memory could be implemented in a harness, but none presently do. User specification of parallelism is managed by QCEngine and is outside QCSchema [e.g., `qcdb` or `qcng.compute(..., local_options={"ncores": 4})`] (I-e).

4. Molecule specification

`Molecule` specification is the most important aspect that QCEngine and QCDB control via QCSchema to the exclusion of a program's DSL. The QCSchema `Molecule` can store mass, isotope, charge/multiplicity, fragmentation, ghostedness, and connectivity information (and more), along with the basic element and Cartesian geometry data (I-d). All quantities are stored in amu or Bohr to avoid imprecision from multiple unit conversions through different revisions of physical constants.

Initializing a molecule can occur through a variety of string formats (of Cartesian coordinates) or directly by arrays. Extensive validation and application of physics-based defaults follows such that string Snippet 1 becomes (Ref. 97 for details) the schema Snippet 2. In the QCDB API, molecules can additionally be specified via Z-matrix, mixed Cartesian/Z-matrix, and with variable and deferred coordinates. QCSchema `Molecule` holds almost all data relevant to molecular system specification in QC, including EFP fragments, which are parseable without additional software and are stored in a secondary object. Items that appear in the molecule specification sections of some programs but do not fit in QCSchema `Molecule`, such as the stars signaling optimizable

```
O 0 0 0
H 2 0 0
--
@22Ne 5 0 0
units bohr
```

SNIPPET 1. A string molecule input with complicating mass number, fragments, and implicit multiplicity.


```
{"atom_labels": ["", "", ""],
 "atomic_numbers": [ 8,  1, 10],
 "fix_com": False,
 "fix_orientation": False,
 "fragment_charges": [0.0, 0.0],
 "fragment_multiplicities": [2, 1],
 "fragments": [[0, 1], [2]],
 "geometry": [[0., 0., 0.],
       [2., 0., 0.],
       [5., 0., 0.]],
 "mass_numbers": [16,  1, 22],
 "masses": [15.99491462,  1.00782503, 21.99138511],
 "molecular_charge": 0.0,
 "molecular_multiplicity": 2,
 "name": "HNeO",
 "provenance": {"creator": "QCElemental",
                "routine": "qcelemental.molparse.from_schema",
                "version": "v0.8.0"},
 "real": [ True,  True, False],
 "schema_name": "qcschema_molecule",
 "schema_version": 2,
 "symbols": ["O", "H", "Ne"],
 "validated": True}
```


SNIPPET 2. QCSchema MOLECULE from Snippet 1. Translation described at Ref. 97.

internal coordinates in CFOUR, reside in an extras section. (EFP and extras are future extensions.)

Like memory or other aspects monopolized by QCSchema, user specification of the molecule in the DSL through keywords (e.g., `scf__nopen` in NWChem or `contrl__icharg` in GAMESS) is ignored and overwritten in QCEngine or raises an error if inconsistent in QCDB.

A requirement for combining vector data from multiple jobs is that the data be in a common frame of reference. Although each QC program has a standard internal orientation, these can be different between programs or between input specifications, and not all programs can return quantities in an arbitrary input frame and atom ordering. To smooth over inconsistent capabilities, the input geometry and the output geometry are both collected from output data, and an aligner computes the displacement, rotation matrix, and atom mapping needed to transform between them. Then, any vector results have the appropriate transformations applied so that all results in `AtomicResult` are in input orientation (O-a). This occurs for both QCEngine and QCDB when the `Molecule` fields `fix_com` and `fix_orientation` are True. (Here, "fix" is used in the "fasten" sense, not the "repair" sense.) When False, QCEngine returns in program native frame, while QCDB returns in Psi4 native frame.

5. Methods

Perhaps the most compelling element of QCSchema is the ability to request methods by a single string rather than piecemeal (e.g., `"blyp-d3(bj)"`, `"mp2"`, `"cis"` in place of `{"method": "blyp","dft_d": "d3_bj"}`, `{"mplevl": 2}`, `{"calclevel": "hf","excite": "cis"}`), thereby closely tying results to the model section (with subfields method and basis) of the data layout (barring algorithm, space, auxiliary basis set choices). As far as possible, all method specification and no extraneous information are consolidated into the `atomicinput.model.method` field. This is the primary translation

effort of each QCENGINE harness, as shown by the uniformity of the field in Fig. 2(b). In calling QCENGINE, the user supplies the canonical method name (I-b). There is no compensation for program peculiarities; for example, `"b3lyp"` returns different answers if submitted to programs that have made a different choice of VWN3 vs VWN5, consistent with the principle that users can translate an input directly into QCSCHEMA.

A complication to this principle is when programs conflate non-method information like algorithm (e.g., `rimp2`) or alternate code paths (e.g., `task tce energy`) into the primary method call. To maintain QCSCHEMA integrity for `model.method`, the project invents top-level keywords like `{"qc_module": "tce"}` to allow deliberate choice of the `TCE` over hand-coded CC in NWCHEM and `{"mp2_type": "df"}` to instruct DF in GAMESS, NWCHEM, or Q-CHEM. Keyword `qc_module` can also control choice of `VCC/ECC/NCC` in CFOUR and `DFMP2/DFOCC/DETCI` in PSI4, although these also have local knobs `cfour_cc_program` and `psi4_qc_module`.

Method specification in QCDB is similar to QCENGINE except a compound program-method argument like `optimize("nwc-mp2")` is used. This difference is historical and endures for ease of specifying composite model chemistries like `gradient("p4-mp2/cc-pv[56]Z + d: nwc-ccsd/cc-pv[tq]z + d: c4-ccsdtq/cc-pvdz")`[98] employing PSI4, NWCHEM, and CFOUR for different stages. Additionally, QCDB tests the major methods to ensure the same string yields the same result (I-f). It also maintains a list of capabilities, so, for example, ROHF CCSD in NWCHEM can be automatically routed to TCE [see Fig. 2(d)]. User specification of method information in keywords instead of through the model field is overwritten without warning in QCENGINE, while in QCDB, contradictory information yields an error.

6. Basis sets

Notwithstanding the curation efforts of the Basis Set Exchange[99] (BSE), every QC program maintains an internal library of basis sets with uneven upstream (from the basis set developer) updates applied, uneven downstream (by the program owner) specializations applied, and different spellings for accessing a given basis, not to mention different data formats. In QCENGINE, only the internal library of a program is used, accessed from the `atomicinput.model.basis` field. Thus, due to DSL, the same string value directed toward different programs can lead to different results, and different strings can lead to the same results, as in Fig. 2(b). To allow consistency between programs and to reduce user DSL demands, QCDB pulls basis sets from a single library (PSI4's in .gbs format, which is amply stocked with Pople, Dunning, Peterson, Karlsruhe, and other orbital and fitting basis sets) and performs the translation into the custom per-atom specification and format for each program, including setting spherical or Cartesian for *d*-shells and higher according to basis set design. In this way, a standard case-insensitive label and a consistent interface to custom and mixed basis sets is available (I-b). Alternatively, QCDB can act like QCENGINE to access a program's internal basis set library through program-specific keywords (e.g., `set gamess_basis__gbasis accd` vs `set basis aug-cc-pvdz`). While the PSI4 basis set library is used at present, future work will switch to the new MolSSI BSE.

7. Execution

Apart from CMS programs, QCENGINE requires only QCELEMENTAL and some common Python packages. It is readily installed by `conda install qcengine -c conda-forge` or `pip install qcengine`. Execution occurs through CLI or one-call API with JSON-like input. For example, if `AtomicInput` specification `{..., "model": {"method": "ccsd","basis": "aug-cc-pvdz"}}` was in a file, `qcengine spec run cfour` would run CFOUR and return QCSCHEMA `AtomicResult` (I-e). If the specification was a dictionary in a Python script, then `qcengine.compute(spec, "cfour")` produces the same results, as in the "execution" column of Fig. 2(b). QCENGINE can be run through a queue manager, but for more than incidental jobs, users should consider the job orchestration capabilities of QCFRACTAL.

QCDB requires only QCENGINE and is installed similarly by `conda install qcdb -c psi4`. Execution modes CLI and one-call API are called analogously, only replacing qcng by qcdb (and ccsd by c4-ccsd) as shown in Figs. 2(c) and 2(d). Additionally, though, QCDB can function through an interactive driver API to reuse molecule and keyword sets and perform more complex sequences. This is shown in Snippet 3 that scans an energy potential and then performs a computation at the optimum distance at a better level of theory. This is analogous to the Psi-API mode in PSI4. A simplified, plain-text input that gets processed into the API and is analogous to the PSIthon mode of PSI4 will be available after further integration with PSI4; an example is at Snippet 4.

```
import qcdb
nefh = qcdb.set_molecule("""Ne
                            --
                            F 1 R
                            H 2 1.0 1 135.0""")
qcdb.set_options({"e_convergence": 7,
                  "mp2_type": "df"})
results = {r / 100: None for r in range(200, 400, 10)}
for intra in results:
    nefh.R = intra
    results[intra] = qcdb.energy("p4-mp2/jun-cc-pvtz")
rmin = min(results, key=results.get)
qcdb.set_options({"e_convergence": 9})
nefh.R = rmin
model = "p4-mp2/aug-cc-pv[tq]z + d:c4-ccsd(t)/aug-cc-pvtz"
ene = qcdb.energy(model, bsse_type="cp")
print(f"Ne...FH at optimal dist. {rmin} A has IE {ene} E_h.")
```

SNIPPET 3. An interfragment potential energy scan followed by composite energy in QCDB.

8. Modes

QCDB operates in two modes, which treat keywords, particularly keyword defaults, differently. QCDB supports distinct modes of operation to tailor its capabilities toward driver integration of multiple programs (when unified results are needed) or toward interfacing a single program (when user familiarity is preferred). Most controlling is the driver or *unified* mode, which endeavors to elicit from different QC programs identical results out of identical input conditions (roughly the combination of method, basis, reference,

active space, and integrals treatment) (I-f). Here, the driver imposes QCDB-level defaults such as non-DF algorithms, all-electron spaces, and graduated convergence criteria for energy vs analytic derivative vs finite difference derivative. This mode is required for multi-program procedure runs [e.g., `energy("p4-mp2/cc-pv[tq]z + d:c4-ccsd/cc-pvtz")`] and is active by default.

Another mode, denoted *sandwich* since the QCDB pre- and post-processing is less intrusive, is for users focusing on a single QC program who want the driver routines, method mapping [e.g., `energy("gms-ccsd(t)",bsse_type="vmfc")`], and I/O-wrapping advantages of QCDB but do not want surprise resets of their accustomed defaults. Driver-suggested QCDB-level (e.g., frozen-core), driver-level (e.g., graduated derivative convergence), and best-practices (e.g., module selection) defaults are all turned off. This mode is effectively how QCENGINE runs.

Some background facts to illustrate the modes:

- For the default MP2 algorithm, PSI4 uses DF, while CFOUR, GAMESS, NWCHEM, and QCDB use CONV.
- The CFOUR, GAMESS, NWCHEM, PSI4, and QCDB default HF density convergences are 10^{-7}, 10^{-5}, 10^{-4}, 10^{-8}, and 10^{-8}, respectively.
- For the CCSD energy from CFOUR, the default CC module is `VCC`, while QCDB best-practice is `ECC`.
- The NWCHEM default `task ccsd energy` does not run for open-shell, while QCDB uses the CCSD module for RHF and `TCE` module for ROHF.
- GAMESS freezes core by default, while CFOUR, NWCHEM, PSI4, and QCDB correlate all electrons.

In the unified mode, `energy("gms-mp2")` and `energy("p4-mp2")` both run all-electron MP2 without DF and with 10^{-8} convergence. After setting ROHF, `energy("c4-ccsd")` runs through `ECC`, and `energy("nwc-ccsd")` runs through TCE, again both HF to 10^{-8} and all-electron. In contrast, sandwich mode `energy("gms-mp2")` produces a conventional frozen-core MP2 energy converged to 10^{-5}, while `energy("p4-mp2")` produces a DF all-electron value converged to 10^{-8}. In the ROHF CCSD case, the CFOUR job runs as all-electron through VCC with HF converged to 10^{-7}, while the NWCHEM submission declines to run.

9. Keywords

QC programs have hundreds of keywords controlling their operation on matters of substance (e.g., `RAS3`), strategy (e.g., `DIIS`), computer science (e.g., `INTS_TOLERANCE`), and research convenience (e.g., `DFT_NEW`). The variety in spelling and text arrangement by which the same ideas are communicated to different QC programs is staggering (and a considerable barrier to trying new codes). The necessity to represent any (single-stage, single-program) input file as QCSCHEMA requires mapping rules so that a user familiar with the native DSL can readily translate into the key/value representation of an `AtomicInput`'s `keywords` field. The primary guideline is that the right-hand side value must be a simple data quantity in natural Python syntax (e.g., CFOUR's `3-1-1-0/3-0-1-0` becomes `[[3, 1, 1, 0], [3, 0, 1, 0]]`), and the left-hand side key is a string that encodes any level of nesting with double-underscore (e.g., GAMESS's `contrl__scftyp` or NWCHEM's `dft__convergence__density`). A present/absent keyword (as opposed to a key/value pair) becomes a boolean, such as NWCHEM `scf__rohf`. The `ProgramHarness` handles formatting the keywords field (back) into the input grammar (I-d), including quashing unnecessary case-sensitivity (e.g., `Qz2p` converts to lowercase for CFOUR, while a filename option passes unchanged). For QCDB, prefixing a keyword by program name targets it toward a particular program; hence, `reference` becomes `cfour_reference` or `psi4_reference`.

The greatest challenge to mapping rules is that some programs have an input structure that blurs module nesting vs keyword name vs keyword value. An extra mapping rule not strictly required by QCENGINE is for keywords to be independent and granular such that they are one-to-one with other programs, not overworked like `dft__grid={"lebedev": (99, 11), "treutler": True}` (insufficiently granular) nor underworked like `scf__rhf=False` plus `scf__uhf=True` (insufficiently independent). QCDB uses internal aliasing and mutually exclusive groups to help keyword specification be intuitive for native users.

Making a QCSCHEMA fed to multiple programs produce uniform output is not within the scope of QCENGINE. Barriers to accessing multiple QC backends through a single DSL or, more intricately, to compatibly mixing backends include (a) heterogeneous control knobs across QC programs each with its own keyword set and (b) incompatible results due to different defaults yielding slightly different answers. QCDB takes up the task of uniting keywords into a single DSL for a further layer of interoperability. Unlike QCENGINE, QCDB registers valid keywords for each QC program and can apply custom validation functions to each. Additionally registered are unified keywords so that, for example, setting `REFERENCE` is translated into `CFOUR_REFERENCE` or `GAMESS_CONTRL__SCFTYP`, as shown in Figs. 2(c) and 2(d) (I-b, I-f). As mentioned above, insisting on granular keywords for the QCSCHEMA representation allows cleaner mapping between QC programs. As mentioned below, QCDB also encodes best-practice keywords to allow shorter inputs, context-dependent defaults, and bridging the developer-user knowledge gap. QCSCHEMA or QCDB API offer ample opportunities for users to submit contradictory input specification, several of which are shown in Snippet 4.

```
memory 300 mb
molecule {
 H
 H 1 0.74
}
set {
 basis 6-31g               # ok, new info
 cfour_calc_level ccsd     # clash w/"c4-hf" below
 cfour_deriv_level first   # clash w/energy() below (use gradient())
 cfour_memory_size 9000000 # clash w/300 mb above
 cfour_multiplicity 3      # clash w/implicit singlet of mol above
 cfour_units angstrom      # ok, consistent w/mol above
}

energy("c4-hf")
```

SNIPPET 4. Contradictory input opportunities.

QCDB resolves competing keyword suggestions and requirements by the user, driver, schema, and best practices into a final keyword set that is passed to QCENGINE for final formatting. Because of this step, incompatible keywords pass without warning in QCENGINE, while in QCDB, contradictory information yields an error.

Codebase authors know best how to run a computation, but they may have conveyed that knowledge only through documentation and forum posts. Due to the unwieldiness of large legacy codebases and the circuity of research (and the burden of backward compatibility), it can happen that a method needs several keywords to express it or that valuable approximations or code-routing do not get turned on by default. Due to its layered Python/C++ structure, PSI4 naturally has a place to express such "best-practice" defaults based on method, basis, system size, etc. The advantage is that simple method + basis inputs yield production-grade results. Thus, QCDB takes advantage of working with codebase authors and the intermediate Python layer to implement best-practice keywords based on available calculation data (I-c). These take the form of routing to the best (or only capable) module for a given method, reference, derivative level, and active space; of supplying sensible defaults such as the number of electrons or roots; of tuning convergence to the derivative and needed precision (analytic vs finite difference) at hand; or of specifying C_1 or highest-Abelian symmetry to modules with symmetry restrictions. Such options can be overridden by the user and can be disabled in sandwich mode (Sec. II E 8). These defaults are themselves subject to change as recommendations evolve, but their state is readily viewed in program inputs.

10. QCVariables

The QC output stream, whether ASCII, binary, or structured, is read immediately after program execution. Scalar and array result quantities, such as `PBE TOTAL ENERGY`, `MP4 CORRELATION ENERGY` and `PBE TOTAL GRADIENT`, `CCSD DIPOLE`, are extracted and held as significant-figure-preserving floats or NumPy arrays, respectively, and are known collectively as QCVariables (O-a). Extraction uses the most precise available source, whether the standard output stream or available auxiliary files (e.g., CFOUR `GRD`). The internal geometry is always collected, and any vector results are manipulated in concert with it, as described in Sec. II E 4. For QCENGINE, many of the same harvested quantities are directed into QCSCHEMA `AtomicResultProperties` lists. Results are available programmatically through `qcdb.variable("mp2 total energy")` or `atomicresult.properties.mp2_total_energy` in QCDB and QCENGINE, respectively.

A mild vexation in QC output files is that they contain different quantities such as total vs correlation energy or opposite-spin vs triplet energy that are interconvertible but not directly comparable. QCVariables enforce the consistency of common QC definitions and encode common combining rules (O-b). They are applied in post-processing to ensure that a maximum of data gets harvested from each run, that exactly the same quantities are collected from each QC program, and that trivially defined methods such as SCS(N)-MP2 and B3LYP-D3(BJ) need not clutter either the QC code or its parsing.

Using binary representations of floats rather than truncated strings from output files is a powerful argument for API integration rather than parsing. Binary representation is essential when dealing with many numbers with slight differences, such as finite differences or MBE sums. Programs with Python APIs (and that use APIs for internal inter-language transfer like between C++ and Python in PSI4) can transfer data with full precision; for QCENGINE, these are, for example, ADCC, OPENMM, RDKit, TORCHANI, DFTD4, PSI4, TCPB TERACHEM, and XTB. Of these, the last four have implemented QCSCHEMA directly for API access. An intermediate step is to use structured output like XML or JSON from MOLPRO, MRCHEM, and QCORE. For certain programs, a combination of reading available binary files (e.g., 99.0 for return energy in Q-CHEM and `JOBARX/JAINDX` for certain QC results and organizational data in CFOUR) and text parsing is employed. Results from other programs are collected solely through text parsing: e.g., DFTD3, GAMESS, GCP, MOPAC, MP2D, NWCHEM, the classic interface to TERACHEM, and TURBOMOLE. Although results are collected into QCSCHEMA from QC programs at the greatest accessible precision, in order to maintain that precision among the data transfers and storage of the QCDB and QCARCHIVE INFRASTRUCTURE ecosystem, the QCELEMENTAL implementation of QCSCHEMA (nominally a JSON Schema,[16] which does not handle binary or `numpy.ndarray`) includes MessagePack[100] serialization.

III. EXAMPLE: DIATOMIC SPECTROSCOPIC CONSTANT FITTING

With contemporary QC software, it is entirely possible to approach the *ab initio* limit in the description of diatomic molecules.[101] Such spectroscopically accurate calculations require extrapolating to the full configuration interaction and complete basis set limits under the non-relativistic Born–Oppenheimer (BO) approximation, followed by usually negligible corrections to account for both relativistic effects and the BO approximation itself. Not only does this type of calculation present a remarkable computational challenge [as it is significantly more expensive than CCSD(T), the usually sufficient target of quantum chemistry], it can also be practically difficult to incorporate multiple corrections and extrapolations into a workflow. While all of the necessary features are present across various QC software packages, no single package implements everything (let alone has the best implementation). Furthermore, enforcing consistent geometries, basis sets, convergence criteria, frozen orbitals, etc. between programs is a cumbersome, often error-prone task. The QCDB driver remedies this problem by providing an easy-to-use Python interface to multiple QC programs.

To showcase this capability of the QCDB driver, the ground states of a few diatomic molecules (BH, HF, and C_2) are optimized at essentially the *ab initio* limit, and spectroscopic constants are computed and compared to experiment. Previous studies estimating the *ab initio* limit for the full set of standard spectroscopic constants of these molecules have been reported (see, e.g., Refs. 102–104). The present study provides improved treatments for some of the small corrections and/or includes more correction terms. Here, we include corrections for electron correlation beyond CCSD(T), basis set effects beyond an already high-quality core-valence quadruple/quintuple-ζ extrapolation, relativistic effects, and the Born–Oppenheimer diagonal correction using four different QC programs through the unified QCDB interface. The effect of each correction is examined separately, as well as the cumulative effect of all corrections. Understanding the cost and importance of each correction is helpful for designing reasonable extrapolations for larger systems.

TABLE II. Composite level of theory for spectroscopic constants and associated QC programs.

Name	Method	Program
E_{Base}	CCSD(T)/cc-pCV[Q5]Z	NWChem
ΔE_{Basis}	MP2/(aug-cc-pCV[56]Z − cc-pCV[Q5]Z)	Psi4
ΔE_{DBOC}	CCSD/cc-pCVDZ	CFOUR
ΔE_{Rel}	X2C-CCSD(T)/cc-pCVTZ	Psi4
ΔE_{CCSDTQ}	[CCSDTQ − CCSD(T)]/cc-pVTZ	CFOUR
ΔE_{FCI}	(FCI − CCSDTQ)/cc-pVDZ	GAMESS/ CFOUR

A spectroscopically accurate model chemistry energy (E_{Total}) is defined as a base energy (E_{Base}) with five separate corrections,

$$E_{\text{Total}} = E_{\text{Base}} + \Delta E_{\text{Basis}} + \Delta E_{\text{DBOC}} + \Delta E_{\text{Rel}} + \Delta E_{\text{CCSDTQ}} + \Delta E_{\text{FCI}}. \quad (1)$$

Each energy and the QC program(s) used to obtain it is defined in Table II.

The rovibrational spectrum of a diatomic molecule is often expressed with Dunham's expansion,

$$E_{vJ} = h\sum_{kl} Y_{kl}\left(v+\frac{1}{2}\right)^k [J(J+1)]^l. \quad (2)$$

The first few Dunham coefficients correspond to well-studied spectroscopic constants,

$$Y_{10} = \omega_e, Y_{20} = -\omega_e x_e, Y_{01} = B_e, Y_{02} = -\bar{D}_e, Y_{11} = -\alpha_e. \quad (3)$$

The following truncation of the expansion is used to describe a diatomic:

$$E \approx U(r_e) + h\omega_e\left(v+\frac{1}{2}\right) + hB_eJ(J+1) - h\omega_e x_e\left(v+\frac{1}{2}\right)^2 - h\alpha_e\left(v+\frac{1}{2}\right)J(J+1) - h\bar{D}_eJ^2(J+1)^2. \quad (4)$$

TABLE III. Comparison between theory and experiment for bond lengths (Å) and spectroscopic constants (cm^{-1}) of three diatomic molecules. All Δ terms correspond to the difference between a value and the base CCSD(T)/cc-pCV[Q5]Z calculation. Experimental values from Ref. 106 (BH), 107 (HF), and 108 (C_2). All published experimental uncertainties are smaller than the displayed precision of the spectroscopic values presented here.

Molecule and method	r_e	ω_e	$\omega_e x_e$	B_e	D_e	α_e
BH						
Base	1.228 90	2371.24	49.4	12.088	0.001 257	0.423
ΔBasis	+0.000 18	−0.44	−0.4	−0.004	−0.000 001	−0.001
ΔDBOC	+0.000 65	−2.33	−0.2	−0.013	−0.000 002	+0.000
ΔRel	−0.000 01	−0.57	+0.1	+0.000	+0.000 001	+0.000
ΔCCSDTQ	+0.000 19	−2.07	+0.1	−0.004	+0.000 001	+0.001
ΔFCI	+0.000 00	+0.00	−0.2	+0.000	+0.000 000	+0.000
ΔTotal	+0.001 01	−5.41	−0.5	−0.020	+0.000 000	+0.000
Total	1.230 00	2365.83	49.0	12.068	0.001 256	0.423
Experiment	1.232 16	2366.72	49.3	12.026	0.001 235	0.422
HF						
Base	0.916 54	4147.01	90.5	20.968	0.002 144	0.793
ΔBasis	+0.000 17	−1.79	−0.7	−0.008	−0.000 001	−0.002
ΔDBOC	+0.000 01	+0.32	−0.2	−0.001	−0.000 001	+0.000
ΔRel	+0.000 06	−3.54	−1.3	−0.003	+0.000 003	+0.000
ΔCCSDTQ	+0.000 21	−4.49	+0.1	−0.009	+0.000 002	+0.002
ΔFCI	+0.000 01	−0.19	+0.0	+0.000	+0.000 000	+0.000
ΔTotal	+0.000 47	−9.70	−2.2	−0.021	+0.000 004	+0.000
Total	0.917 00	4137.31	88.3	20.947	0.002 148	0.792
Experiment	0.916 808	4138.32	89.0	20.956	0.002 151	0.798
C_2						
Base	1.240 39	1873.63	12.6	1.826	0.000 007	0.017
ΔBasis	+0.000 16	−1.01	+0.0	+0.000	+0.000 000	+0.000
ΔDBOC	+0.000 01	+0.09	+0.0	+0.000	+0.000 000	+0.000
ΔRel	−0.000 16	−0.41	+0.1	+0.000	+0.000 000	+0.000
ΔCCSDTQ	+0.001 46	−11.76	+0.8	−0.004	+0.000 000	+0.001
ΔFCI	+0.001 00	−4.58	+0.0	−0.003	+0.000 000	+0.000
ΔTotal	+0.002 48	−17.81	+0.8	−0.007	+0.000 000	+0.001
Total	1.242 87	1855.82	13.4	1.819	0.000 007	0.018
Experiment	1.242 44	1855.01	13.6	1.820	0.000 007	0.018

The spectroscopic constants are then describable in terms of the electronic PES $U(r)$ and its derivatives,

$$I_e \equiv \mu r_e^2 \quad B_e \equiv \frac{h}{8\pi^2 I_e} \quad \omega_e \equiv \frac{1}{2\pi}\left[\frac{U''(r_e)}{\mu}\right]^{1/2}, \tag{5}$$

$$\omega_e x_e \equiv \frac{B_e^2 r_e^4}{4h\omega_e^2}\left[\frac{10B_e r_e^2[U'''(r_e)]^2}{3h\omega_e^2} - U^{iv}(r_e)\right], \tag{6}$$

$$\alpha_e \equiv \frac{2B_e^2}{\omega_e}\left[\frac{2B_e r_e^3 U'''(r_e)}{h\omega_e^2} + 3\right] \quad \tilde{D}_e \equiv \frac{4B_e^3}{\omega_e^2}. \tag{7}$$

Note that these are all "equilibrium" constants, i.e., they are with respect to the bottom of the potential well (but with inclusion of the Born–Oppenheimer diagonal correction).

Accessed through the QCDB interface, the PSI4 diatomic procedure fits a set of points $[r, E(r)]$ to this truncation, solving for the spectroscopic constants via a least-squares optimization.[105] This procedure was used in the following way for each diatomic:

1. Through the QCDB driver, E_{Total} was calculated at seven values of r, spaced 0.005 Å apart and centered approximately at the minimum of the PES. The spectroscopic constants were calculated with PSI4, including an approximate r_e.
2. This seven-point calculation was repeated using the approximate r_e from the first step as the central point. The spectroscopic constants calculated from these PES points are those tabulated here.

Basis sets with spherical harmonics were used in all calculations, and basis set coefficients were standardized across all programs via QCDB. Electrons in core orbitals were frozen for computations using the cc-pVXZ basis set family, which lack core correlation functions. Energies were converged to at least 10^{-10} Hartrees in all programs. Even tighter convergence would be beneficial for the numerical differentiation performed in the fitting. Numerical tests suggest that this precision in energy can lead to uncertainties in α_e [proportional to $U'''(r_e)$] and $\omega_e x_e$ [proportional to $U^{\text{iv}}(r_e)$] as large as 0.0001 and 0.2 cm^{-1}, respectively.

The calculations of all diatomics and spectroscopic constants are presented in Table III, and the results for r_e and ω_e are shown in Fig. 4 for easier analysis. Prior to discussing the chemical and computational implications of these results, it is worthwhile to first note that the corrections for BH closely match those of a previous study[103] by Temelso *et al.* (which used a similar but less exact extrapolation). This validates these results from a software perspective: each program must be using correct geometries, basis sets, convergence criteria, etc. The finite-difference nature of the fitting procedure makes close agreement between programs particularly important.

The total extrapolation procedure shows remarkable agreement with experiment for bond lengths r_e (within 0.0005 Å) except for BH, off by 0.0022 Å. However, this extrapolation lacks nonadiabatic BO effects, which were found by Martin[102] to be unusually high for BH, ~0.0025 Å. This is rather close to the overall difference of 0.0022 Å between experiment and our best estimate.

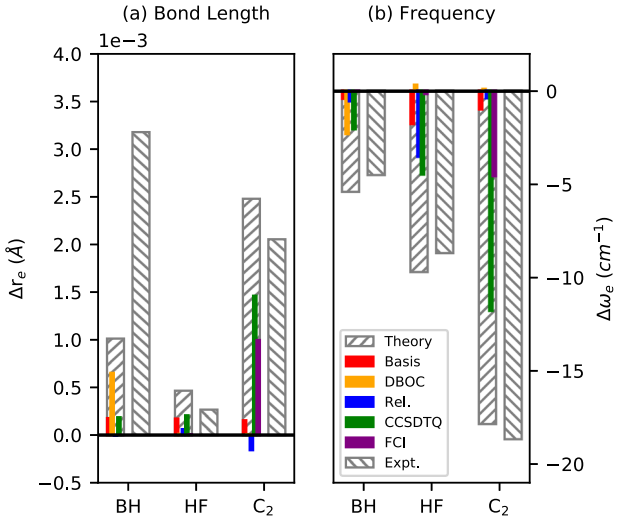

FIG. 4. Influence of post-CCSD(T)/CBS corrections on two spectroscopic constants, (a) r_e and (b) ω_e, and three diatomic systems, BH, HF, and C_2. For each system, the right gray bar is the difference between the experimental constant and the constant calculated at the E_{Base} [CCSD(T)/CBS] level of theory. The left gray bar is the difference in constants calculated at the E_{Total} and E_{Base} levels of theories. Within the left gray bar, contributions from each correction are shown as colored bars. Data are from Table III.

Theoretical harmonic frequencies ω_e are in excellent agreement with experiment, off by only 1 cm^{-1}. The rotational constant B_e is also well predicted, within 0.01 cm^{-1} for HF and C_2 and off by a somewhat larger 0.04 cm^{-1} for BH. The latter error may be largely due to already-noted non-BO effects, which cause a larger discrepancy in r_e for BH. $\omega_e x_e$ is in good agreement with experiment, matching within 0.2–0.4 cm^{-1} for BH and C_2 but is off by a larger 1.6 cm^{-1} for HF. It is not clear that the corrections employed here actually improve this constant, and the remaining discrepancy could be due to the numerical precision limitations discussed earlier. $\tilde{D}_e$ is very well predicted already by the base method, and the various corrections are extremely small. Similarly, α_e appears to not require corrections on top of the base method, each of which changes it by only ±0.002 cm^{-1} or less. Final values are within 0.005 cm^{-1} of experiment.

Figure 4 shows that the sum of the small corrections matches experiment very well for r_e and ω_e, except for the bond length of BH, where non-BO effects are important as noted above. All of the small corrections considered can be important for r_e and ω_e, although there is no consistency about their relative importance from one molecule to another. For example, the DBOC is rather important for BH (which has the lightest nuclei), but not for HF and even less so for C_2. Similarly, the FCI correction (beyond CCSDTQ) is negligible for BH and HF but is important for C_2 (worth 0.001 Å and 4.6 cm^{-1}) In total, the corrections for C_2 lower the value of ω_e by a surprisingly large 17.81 cm^{-1} from the base CCSD(T) value, which is very close to the experimental ω_e (18.61 cm^{-1} lower than the base). A large majority of this change is due to missing electron correlation: the CCSDTQ correction is responsible for about 12 cm^{-1} and the FCI correction by about another 5 cm^{-1}. This is presumably due to the much larger degree of electron correlation in C_2,

arising from the close near-degeneracy of the [core]$2\sigma_g^2 2\sigma_u^2 1\pi_x^2 1\pi_y^2$ and [core]$2\sigma_g^2 1\pi_x^2 1\pi_y^2 3\sigma_g^2$ configurations.

IV. SUMMARY AND CONCLUSIONS

Users increasingly desire programmatic (i.e., API: application programming interface) access to QC results, either for their convenience or for incorporation into automated workflows. The interface, volume, and intricacy requirements of that access vary widely across applications and increasingly involve *uniform* results across QC programs. The QCElemental, QCEngine, and QCDB software modules [the former two being part of the Molecular Sciences Software Institute[18] (MolSSI) QCArchive[17] project] provide a framework to facilitate interoperability among community computational molecular sciences (CMS) programs.

QCArchive and QCDB have been designed to work with emerging tools and standards developed by MolSSI, particularly the QCSchema JSON format for information passing. QCElemental provides implementations and validators around QCSchema objects, while QCEngine provides QCSchema I/O adaptors for CMS codes. In addition to wrapping nearly a dozen QC programs for uniform execution and programmatic access to results, QCEngine interfaces with geomeTRIC and other geometry optimizers that can, in turn, call QCEngine for QC gradients. QCEngine easily expands to additional CMS codes, has parallel execution capabilities through QCFractal, and by definition allows uniform execution, yet it is not in itself a coherent QC driver due to the differing implementations, conventions, defaults, and DSL of QC codes.

The Quantum Chemistry Common Driver and Databases (QCDB) project provides a simple and powerful driver front-end to multiple QC programs, allowing users automatic access to several features formerly requiring specialized scripts or laborious post-processing. These include built-in composite methods, many-body expansion procedures, vibrational analysis, and combinations thereof for not only energies but also gradients, Hessians, and geometry optimizations. By adding the basis set, keywords, and result tools for uniformity and interoperability, QCDB also allows mixing and matching capabilities of multiple quantum chemistry programs within a single computation. These features have been demonstrated with an application computing spectroscopic constants of diatomic molecules at the *ab initio* limit, including corrections for post-CCSD(T) electron correlation, beyond-cc-pCV[Q5]Z basis set effects, relativistic effects, and the Born–Oppenheimer diagonal correction, combining total energies computed by CFOUR, GAMESS, NWChem, and Psi4.

V. EXTERNAL MATERIAL

Software repositories and documentation are available for QCElemental at https://github.com/MolSSI/QCElemental/ and https://molssi.github.io/QCElemental/, for QCEngine at https://github.com/MolSSI/QCEngine/ and https://molssi.github.io/QCEngine/, for QCDB at https://github.com/qcdb/qcdb/ and https://qcdb.github.io/qcdb/, and for general QCArchive INFRASTRUCTURE at http://docs.qcarchive.molssi.org/. These programs remain in active development. Production computations are under way using many features of the software, and test suites are expected to pass. However, users are encouraged to contact the developers as they venture afield of the verified tests. Many snippets from this work, including an abbreviated diatomic fitting, are demonstrated in the test suite: https://github.com/qcdb/qcdb/blob/master/qcdb/tests/test_manuscript.py.

ACKNOWLEDGMENTS

Several of the co-authors have been supported in their development of QCDB and QCEngine and affiliated projects by the U.S. National Science Foundation through Grant Nos. ACI-1449723, CHE-1566192, ACI-1609842, ACI-1547580, ACI-1047772, ACI-1450217, ACI-2003931, CHE-1664325, and CHE-2134792, by the Office of Basic Energy Sciences Computational Chemical Sciences (CCS) Research Program (Grant Nos. AL-18-380-057 and DE-SC0018412), and by the Exascale Computing Project (Grant No. 17-SC-20-SC), a collaborative effort of the U.S. Department of Energy (DOE) Office of Science and the National Nuclear Security Administration.

The Molecular Sciences Software Institute acknowledges the Advanced Research Computing at Virginia Tech for providing computational resources and technical support.

D.G.A.S. also acknowledges the Open Force Field Consortium and Initiative for financial and scientific support.

A.G.H. was supported, in part, by the National Science Foundation under Grant No. OAC-1931387 at Stony Brook University and made use of the high-performance SeaWulf computing system, which was made possible by the National Science Foundation (Grant No. 1531492).

L.W. was additionally supported by DOE's Advanced Scientific Research Office (ASCR) under Contract No. DE-AC02-06CH11357.

H.J.K. and F.L. were partially supported by Grant No. DE-SC001896.

L.-P.W. acknowledges Grant No. ACS-PRF 58158-DNI6 and National Institutes of Health (Grant No. R01GM132386).

J.D.C. acknowledges support from NIH Grant No. P30 CA008748, NIH Grant No. R01 GM132386, and the Sloan Kettering Institute.

R.D.R. acknowledges support from the European High-Performance Computing Joint Undertaking under Grant Agreement No. 951732 and partial support from the Research Council of Norway through its Centres of Excellence scheme (Project No. 262695) and through its Mobility Grant scheme (Project No. 261873).

H. K. and J.Š. acknowledge funding from the Praemium academiae (CAS).

M.F.H. has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant Agreement No 810367).

T.J.M. and C.B.H. acknowledge support from the Office of Naval Research (Grant Nos. N00014-18-1-2659 and N00014-17-1-2875). This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1656518 for C.B.H. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

The contents of this paper are solely the responsibility of the authors and do not necessarily represent the views of the commercial partners of the Open Force Field Consortium.

AUTHOR DECLARATIONS

Conflict of Interest

J.D.C. is a current member of the Scientific Advisory Board of OpenEye Scientific Software, Redesign Science, and Interline Therapeutics and has equity interests in Redesign Science and Interline Therapeutics.

DATA AVAILABILITY

The data that support the findings of this study are available within the article.

REFERENCES

[1] D. G. Truhlar, "Basis-set extrapolation," Chem. Phys. Lett. **294**, 45–48 (1998).

[2] A. Halkier, T. Helgaker, P. Jørgensen, W. Klopper, H. Koch, J. Olsen, and A. K. Wilson, "Basis-set convergence in correlated calculations on Ne, N_2, and H_2O," Chem. Phys. Lett. **286**, 243–252 (1998).

[3] A. L. L. East and W. D. Allen, "The heat of formation of NCO," J. Chem. Phys. **99**, 4638–4650 (1993).

[4] A. G. Császár, W. D. Allen, and H. F. Schaefer, "In pursuit of the *ab initio* limit for conformational energy prototypes," J. Chem. Phys. **108**, 9751–9764 (1998).

[5] M. S. Schuurman, S. R. Muir, W. D. Allen, and H. F. Schaefer, "Toward subchemical accuracy in computational thermochemistry: Focal point analysis of the heat of formation of NCO and [H, N, C, O] isomers," J. Chem. Phys. **120**, 11586–11599 (2004).

[6] L. A. Curtiss, K. Raghavachari, P. C. Redfern, V. Rassolov, and J. A. Pople, "Gaussian-3 (G3) theory for molecules containing first and second-row atoms," J. Chem. Phys. **109**, 7764–7776 (1998).

[7] A. Tajti, P. G. Szalay, A. G. Császár, M. Kállay, J. Gauss, E. F. Valeev, B. A. Flowers, J. Vázquez, and J. F. Stanton, "HEAT: High accuracy extrapolated *ab initio* thermochemistry," J. Chem. Phys. **121**, 11599–11613 (2004).

[8] S. Grimme, J. Antony, S. Ehrlich, and H. Krieg, "A consistent and accurate *ab initio* parametrization of density functional dispersion correction (DFT-D) for the 94 elements H–Pu," J. Chem. Phys. **132**, 154104 (2010).

[9] S. F. Boys and F. Bernardi, "The calculation of small molecular interactions by the differences of separate total energies. Some procedures with reduced errors," Mol. Phys. **19**, 553–566 (1970).

[10] K. Raghavachari, G. W. Trucks, J. A. Pople, and M. Head-Gordon, "A fifth-order perturbation comparison of electron correlation theories," Chem. Phys. Lett. **157**, 479–483 (1989).

[11] B. Thaller, I. P. Grant, H. M. Quiney, D. Andrae, J. Desclaux, T. Saue, L. Labzowsky, I. Goidenko, E. Engel, M. Dolg, J. Sapirstein, N. Christensen, A. Wolf, M. Reiher, B. A. Hess, W. Kutzelnigg, D. Sundholm, K. Fægri, K. G. Dyall, P. Schwerdtfeger, and L. Visscher, in *Relativistic Electronic Structure Theory—Part 1: Fundamentals*, edited by P. Schwerdtfeger (Elsevier, Amsterdam, 2002).

[12] M. Born and R. Oppenheimer, "Zur quantentheorie der molekeln," Ann. Phys. **389**, 457–484 (1927).

[13] W. Kutzelnigg, "The adiabatic approximation I. The physical background of the Born–Handy ansatz," Mol. Phys. **90**, 909–916 (1997).

[14] C. E. Warden, D. G. A. Smith, L. A. Burns, U. Bozkaya, and C. D. Sherrill, "Efficient and automated computation of accurate molecular geometries using focal-point approximations to large-basis coupled-cluster theory," J. Chem. Phys. **152**, 124109 (2020).

[15] The QCDB project name, Quantum Chemistry Common Driver and Databases, describes its early scope. The database aspect has since departed and been properly developed in the QCARCHIVE project, particularly QCFRACTAL. See Sec. II A and Ref. 17 for details.

[16] JSON SCHEMA: A vocabulary that allows you to annotate and validate JSON documents. For the current version, see https://json-schema.org/; accessed January 2020.

[17] D. G. A. Smith, D. Altarawy, L. A. Burns, M. Welborn, L. N. Naden, L. Ward, S. Ellis, B. P. Pritchard, and T. D. Crawford, "The MolSSI QCARCHIVE project: An open-source platform to compute, organize, and share quantum chemistry data," Wiley Interdiscip. Rev.: Comput. Mol. Sci. **11**, e1491 (2021).

[18] 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).

[19] D. G. A. Smith, B. de Jong, L. A. Burns, G. Hutchison, and M. D. Hanwell, QCSCHEMA: A schema for quantum chemistry. For the current version, see https://github.com/MolSSI/QCSchema; accessed January 2020.

[20] M. Barbatti, G. Granucci, M. Persico, M. Ruckenbauer, M. Vazdar, M. Eckert-Maksić, and H. Lischka, "The on-the-fly surface-hopping program system NEWTON-X: Application to ab initio simulation of the nonadiabatic photodynamics of benchmark systems," J. Photochem. Photobiol., A **190**, 228–240 (2007).

[21] A. Toniolo, A. L. Thompson, and T. J. Martínez, "Excited state direct dynamics of benzene with reparameterized multi-reference semiempirical configuration interaction methods," Chem. Phys. **304**, 133–145 (2004).

[22] B. G. Levine, J. D. Coe, A. M. Virshup, and T. J. Martínez, "Implementation of *ab initio* multiple spawning in the Molpro quantum chemistry package," Chem. Phys. **347**, 3–16 (2008).

[23] A. Gaenko, A. DeFusco, S. A. Varganov, T. J. Martínez, and M. S. Gordon, "Interfacing the ab initio multiple spawning method with electronic structure methods in GAMESS: Photodecay of *trans*-azomethane," J. Phys. Chem. A **118**, 10902–10908 (2014).

[24] M. Keceli and S. Elliott, Quantum Thermochemistry Calculator; https://github.com/PACChem/QTC; accessed 25 November 2019.

[25] J. Steinmetzer, S. Kupfer, and S. Gräfe, "pysisyphus: Exploring potential energy surfaces in ground and excited states," Int. J. Quantum Chem. **121**, e26390 (2021).

[26] J. Řezáč, CUBY—Ruby framework for computational chemistry, version 4, http://cuby4.molecular.cz; accessed 22 November 2019.

[27] J. Řezáč, "CUBY: An integrative framework for computational chemistry," J. Comput. Chem. **37**, 1230–1237 (2016).

[28] W. F. Polik and J. R. Schmidt, "WebMO: Web-based computational chemistry calculations in education and research," Wiley Interdiscip. Rev.: Comput. Mol. Sci. **2021**, e1554.

[29] 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).

[30] M. Gjerding and A. H. Larsen, ASR: Atomic simulation recipes: Recipes for calculating material properties. For the current version, see https://gitlab.com/asr-dev/asr; accessed September 2021. Documentation at https://asr.readthedocs.io/en/latest/index.html.

[31] S. P. Huber, S. Zoupanos, M. Uhrin, L. Talirz, L. Kahle, R. Häuselmann, D. Gresch, T. Müller, A. V. Yakutovich, C. W. Andersen, F. F. Ramirez, C. S. Adorf, F. Gargiulo, S. Kumbhar, E. Passaro, C. Johnston, A. Merkys, A. Cepellotti, N. Mounet, N. Marzari, B. Kozinsky, and G. Pizzi, "AIIDA 1.0, a scalable computational infrastructure for automated reproducible workflows and data provenance," Sci. Data **7**, 300 (2020).

[32] M. Uhrin, S. P. Huber, J. Yu, N. Marzari, and G. Pizzi, "Workflows in AIIDA: Engineering a high-throughput, event-based engine for robust and modular computational workflows," Comput. Mater. Sci. **187**, 110086 (2021).

[33] Y. Lu, M. R. Farrow, P. Fayon, A. J. Logsdail, A. A. Sokol, C. R. A. Catlow, P. Sherwood, and T. W. Keal, "Open-source, Python-based redevelopment of the CHEMSHELL multiscale QM/MM environment," J. Chem. Theory Comput. **15**, 1317–1328 (2019).

[34] N. M. O'Boyle, A. L. Tenderholt, and K. M. Langner, "CCLIB: A library for package-independent computational chemistry algorithms," J. Comput. Chem. **29**, 839–845 (2008).

[35]K. M. Langner and E. Berquist, CCLIB: Parsers and algorithms for computational chemistry logfiles. For the current version, see https://github.com/cclib/cclib; accessed May 2021, https://doi.org/10.5281/zenodo.1407790.
[36]T. Verstraelen, P. Tecmer, F. Heidar-Zadeh, C. E. González-Espinoza, M. Chan, T. D. Kim, K. Boguslawski, S. Fias, S. Vandenbrande, D. Berrocal, and P. W. Ayers, HORTON, version 2.1.1, a helpful open-source research tool for n-fermion systems, see http://theochem.github.com/horton/.
[37]D. G. A. Smith, L. A. Burns, L. Naden, and M. Welborn, QCELEMENTAL: Periodic table, physical constants, and molecule parsing for quantum chemistry. For the current version, see https://github.com/MolSSI/QCElemental; accessed January 2020.
[38]D. G. A. Smith, S. Lee, L. A. Burns, and M. Welborn, QCENGINE: Quantum chemistry program executor and IO standardizer (QCSCHEMA). For the current version, see https://github.com/MolSSI/QCEngine; accessed January 2020.
[39]D. G. A. Smith, M. Welborn, D. Altarawy, and L. Naden, QCFRACTAL: A distributed compute and database platform for quantum chemistry. For the current version, see https://github.com/MolSSI/QCFractal; accessed January 2020.
[40]D. G. A. Smith, L. A. Burns, D. Altarawy, L. Naden, and M. Welborn, QCARCHIVE: A central source to compile, aggregate, query, and share quantum chemistry data, https://qcarchive.molssi.org; accessed January 2020.
[41]L. A. Burns, A. T. Lolinco, Z. L. Glick, J. Lee, and N. D. Silva, QCDB: Quantum chemistry common driver and databases. For the current version, see https://github.com/qcdb/qcdb; accessed January 2020.
[42]H. Grecco, PINT: Operate and manipulate physical quantities in Python. For the current version, see https://github.com/hgrecco/pint; accessed April 2020.
[43]S. Colvin, PYDANTIC: Data parsing and validation using Python type hints. For the current version, see https://github.com/samuelcolvin/pydantic; accessed April 2020.
[44]M. F. Herbst, M. Scheurer, T. Fransson, D. R. Rehn, and A. Dreuw, "adcc: A versatile toolkit for rapid development of algebraic-diagrammatic construction methods," Wiley Interdiscip. Rev.: Comput. Mol. Sci. **10**, e1462 (2020).
[45]M. F. Herbst and M. Scheurer, ADCC: Seamlessly connect your program to ADC. For the current version, see https://github.com/adc-connect/adcc; accessed January 2020. For the originating project, see https://adc-connect.org.
[46]D. A. Matthews, L. Cheng, M. E. Harding, F. Lipparini, S. Stopkowicz, T.-C. Jagau, P. G. Szalay, J. Gauss, and J. F. Stanton, "Coupled-cluster techniques for computational chemistry: The CFOUR program package," J. Chem. Phys. **152**, 214108 (2020).
[47]G. M. J. Barca, C. Bertoni, L. Carrington, D. Datta, N. De Silva, J. E. Deustua, D. G. Fedorov, J. R. Gour, A. O. Gunina, E. Guidez, T. Harville, S. Irle, J. Ivanic, K. Kowalski, S. S. Leang, H. Li, W. Li, J. J. Lutz, I. Magoulas, J. Mato, V. Mironov, H. Nakata, B. Q. Pham, P. Piecuch, D. Poole, S. R. Pruitt, A. P. Rendell, L. B. Roskop, K. Ruedenberg, T. Sattasathuchana, M. W. Schmidt, J. Shen, L. Slipchenko, M. Sosonkina, V. Sundriyal, A. Tiwari, J. L. Galvez Vallejo, B. Westheimer, M. Włoch, P. Xu, F. Zahariev, and M. S. Gordon, "Recent developments in the general atomic and molecular electronic structure system," J. Chem. Phys. **152**, 154102 (2020).
[48]H.-J. Werner, P. J. Knowles, G. Knizia, F. R. Manby, M. Schütz, P. Celani, W. Györffy, D. Kats, T. Korona, R. Lindh, A. Mitrushenkov, G. Rauhut, K. R. Shamasundar, T. B. Adler, R. D. Amos, S. J. Bennie, A. Bernhardsson, A. Berning, D. L. Cooper, M. J. O. Deegan, A. J. Dobbyn, F. Eckert, E. Goll, C. Hampel, A. Hesselmann, G. Hetzer, T. Hrenar, G. Jansen, C. Köppl, S. J. R. Lee, Y. Liu, A. W. Lloyd, Q. Ma, R. A. Mata, A. J. May, S. J. McNicholas, W. Meyer, T. F. Miller III, M. E. Mura, A. Nicklass, D. P. O'Neill, P. Palmieri, D. Peng, K. Pflüger, R. Pitzer, M. Reiher, T. Shiozaki, H. Stoll, A. J. Stone, R. Tarroni, T. Thorsteinsson, M. Wang, and M. Welborn, molpro, version 2019.2, a package of *ab initio* programs, 2019, see https://www.molpro.net.
[49]H.-J. Werner, P. J. Knowles, F. R. Manby, J. A. Black, K. Doll, A. Heßelmann, D. Kats, A. Köhn, T. Korona, D. A. Kreplin, Q. Ma, T. F. Miller, A. Mitrushchenkov, K. A. Peterson, I. Polyak, G. Rauhut, and M. Sibaev, "The Molpro quantum chemistry package," J. Chem. Phys. **152**, 144107 (2020).
[50]R. Bast, M. Bjørgve, R. Di Remigio, A. Durdek, L. Frediani, G. Gerez, S. R. Jensen, J. Juselius, R. Monstad, and P. Wind, MRCHEM: MultiResolution Chemistry, 2020.
[51]S. R. Jensen, S. Saha, J. A. Flores-Livas, W. Huhn, V. Blum, S. Goedecker, and L. Frediani, "The elephant in the room of density functional theory calculations," J. Phys. Chem. Lett. **8**, 1449–1457 (2017).
[52]E. Aprà, E. J. Bylaska, W. A. de Jong, N. Govind, K. Kowalski, T. P. Straatsma, M. Valiev, H. J. J. van Dam, Y. Alexeev, J. Anchell, V. Anisimov, F. W. Aquino, R. Atta-Fynn, J. Autschbach, N. P. Bauman, J. C. Becca, D. E. Bernholdt, K. Bhaskaran-Nair, S. Bogatko, P. Borowski, J. Boschen, J. Brabec, A. Bruner, E. Cauët, Y. Chen, G. N. Chuev, C. J. Cramer, J. Daily, M. J. O. Deegan, T. H. Dunning, M. Dupuis, K. G. Dyall, G. I. Fann, S. A. Fischer, A. Fonari, H. Früchtl, L. Gagliardi, J. Garza, N. Gawande, S. Ghosh, K. Glaesemann, A. W. Götz, J. Hammond, V. Helms, E. D. Hermes, K. Hirao, S. Hirata, M. Jacquelin, L. Jensen, B. G. Johnson, H. Jónsson, R. A. Kendall, M. Klemm, R. Kobayashi, V. Konkov, S. Krishnamoorthy, M. Krishnan, Z. Lin, R. D. Lins, R. J. Littlefield, A. J. Logsdail, K. Lopata, W. Ma, A. V. Marenich, J. Martin del Campo, D. Mejia-Rodriguez, J. E. Moore, J. M. Mullin, T. Nakajima, D. R. Nascimento, J. A. Nichols, P. J. Nichols, J. Nieplocha, A. Otero-de-la Roza, B. Palmer, A. Panyala, T. Pirojsirikul, B. Peng, R. Peverati, J. Pittner, L. Pollack, R. M. Richard, P. Sadayappan, G. C. Schatz, W. A. Shelton, D. W. Silverstein, D. M. A. Smith, T. A. Soares, D. Song, M. Swart, H. L. Taylor, G. S. Thomas, V. Tipparaju, D. G. Truhlar, K. Tsemekhman, T. Van Voorhis, Á. Vázquez-Mayagoitia, P. Verma, O. Villa, A. Vishnu, K. D. Vogiatzis, D. Wang, J. H. Weare, M. J. Williamson, T. L. Windus, K. Woliński, A. T. Wong, Q. Wu, C. Yang, Q. Yu, M. Zacharias, Z. Zhang, Y. Zhao, and R. J. Harrison, "NWCHEM: Past, present, and future," J. Chem. Phys. **152**, 184102 (2020).
[53]D. G. A. Smith, L. A. Burns, A. C. Simmonett, R. M. Parrish, M. C. Schieber, R. Galvelis, P. Kraus, H. Kruse, R. D. Remigio, A. Alenaizan, A. M. James, S. Lehtola, J. P. Misiewicz, M. Scheurer, R. A. Shaw, J. B. Schriber, Y. Xie, Z. L. Glick, D. A. Sirianni, J. S. O'Brien, J. M. Waldrop, A. Kumar, E. G. Hohenstein, B. P. Pritchard, B. R. Brooks, H. F. Schaefer III, A. Y. Sokolov, K. Patkowski, A. E. DePrince III, U. Bozkaya, R. A. King, F. A. Evangelista, J. M. Turney, T. D. Crawford, and C. D. Sherrill, "PSI4 1.4: Open-source software for high-throughput quantum chemistry," J. Chem. Phys. **152**, 184108 (2020).
[54]Y. Shao, Z. Gan, E. Epifanovsky, A. T. Gilbert, M. Wormit, J. Kussmann, A. W. Lange, A. Behn, J. Deng, X. Feng, D. Ghosh, M. Goldey, P. R. Horn, L. D. Jacobson, I. Kaliman, R. Z. Khaliullin, T. Kuś, A. Landau, J. Liu, E. I. Proynov, Y. M. Rhee, R. M. Richard, M. A. Rohrdanz, R. P. Steele, E. J. Sundstrom, H. L. Woodcock III, P. M. Zimmerman, D. Zuev, B. Albrecht, E. Alguire, B. Austin, G. J. O. Beran, Y. A. Bernard, E. Berquist, K. Brandhorst, K. B. Bravaya, S. T. Brown, D. Casanova, C.-M. Chang, Y. Chen, S. H. Chien, K. D. Closser, D. L. Crittenden, M. Diedenhofen, R. A. DiStasio, Jr., H. Do, A. D. Dutoi, R. G. Edgar, S. Fatehi, L. Fusti-Molnar, A. Ghysels, A. Golubeva-Zadorozhnaya, J. Gomes, M. W. Hanson-Heine, P. H. Harbach, A. W. Hauser, E. G. Hohenstein, Z. C. Holden, T.-C. Jagau, H. Ji, B. Kaduk, K. Khistyaev, J. Kim, J. Kim, R. A. King, P. Klunzinger, D. Kosenkov, T. Kowalczyk, C. M. Krauter, K. U. Lao, A. D. Laurent, K. V. Lawler, S. V. Levchenko, C. Y. Lin, F. Liu, E. Livshits, R. C. Lochan, A. Luenser, P. Manohar, S. F. Manzer, S.-P. Mao, N. Mardirossian, A. V. Marenich, S. A. Maurer, N. J. Mayhall, E. Neuscamman, C. M. Oana, R. Olivares-Amaya, D. P. O'Neill, J. A. Parkhill, T. M. Perrine, R. Peverati, A. Prociuk, D. R. Rehn, E. Rosta, N. J. Russ, S. M. Sharada, S. Sharma, D. W. Small, A. Sodt, T. Stein, D. Stück, Y.-C. Su, A. J. Thom, T. Tsuchimochi, V. Vanovschi, L. Vogt, O. Vydrov, T. Wang, M. A. Watson, J. Wenzel, A. White, C. F. Williams, J. Yang, S. Yeganeh, S. R. Yost, Z.-Q. You, I. Y. Zhang, X. Zhang, Y. Zhao, B. R. Brooks, G. K. Chan, D. M. Chipman, C. J. Cramer, W. A. Goddard III, M. S. Gordon, W. J. Hehre, A. Klamt, H. F. Schaefer III, M. W. Schmidt, C. D. Sherrill, D. G. Truhlar, A. Warshel, X. Xu, A. Aspuru-Guzik, R. Baer, A. T. Bell, N. A. Besley, J.-D. Chai, A. Dreuw, B. D. Dunietz, T. R. Furlani, S. R. Gwaltney, C.-P. Hsu, Y. Jung, J. Kong, D. S. Lambrecht, W. Liang, C. Ochsenfeld, V. A. Rassolov, L. V. Slipchenko, J. E. Subotnik, T. V. Voorhis, J. M. Herbert, A. I. Krylov, P. M. Gill, and M. Head-Gordon, "Advances in molecular quantum chemistry contained in the Q-Chem 4 program package," Mol. Phys. **113**, 184–215 (2015).
[55]F. Manby, T. Miller, P. Bygrave, F. Ding, T. Dresselhaus, F. Batista-Romero, A. Buccheri, C. Bungey, S. Lee, R. Meli *et al.*, "Entos: A quantum molecular simulation package," in *ChemRxiv* (Cambridge Open Engage, 2019).
[56]I. S. Ufimtsev and T. J. Martínez, "Quantum chemistry on graphical processing units. 3. Analytical energy gradients, geometry optimization, and first principles molecular dynamics," J. Chem. Theory Comput. **5**, 2619–2628 (2009).
[57]S. Seritan, C. Bannwarth, B. S. Fales, E. G. Hohenstein, S. I. L. Kokkila-Schumacher, N. Luehr, J. W. Snyder, C. Song, A. V. Titov, I. S. Ufimtsev, and T. J.

Martínez, "TeraChem: Accelerating electronic structure and *ab initio* molecular dynamics with graphical processing units," J. Chem. Phys. **152**, 224110 (2020).
[58]F. Furche, R. Ahlrichs, C. Hättig, W. Klopper, M. Sierka, and F. Weigend, "Turbomole," Wiley Interdiscip. Rev.: Comput. Mol. Sci. **4**, 91–100 (2014).
[59]S. G. Balasubramani, G. P. Chen, S. Coriani, M. Diedenhofen, M. S. Frank, Y. J. Franzke, F. Furche, R. Grotjahn, M. E. Harding, C. Hättig, A. Hellweg, B. Helmich-Paris, C. Holzer, U. Huniar, M. Kaupp, A. Marefat Khah, S. Karbalaei Khani, T. Müller, F. Mack, B. D. Nguyen, S. M. Parker, E. Perlt, D. Rappoport, K. Reiter, S. Roy, M. Rückert, G. Schmitz, M. Sierka, E. Tapavicza, D. P. Tew, C. van Wüllen, V. K. Voora, F. Weigend, A. Wodyński, and J. M. Yu, "TURBOMOLE: Modular program suite for *ab initio* quantum-chemical and condensed-matter simulations," J. Chem. Phys. **152**, 184107 (2020).
[60]J. J. P. Stewart, MOPAC: Semiempirical quantum chemistry. For the current version, see http://OpenMOPAC.net/; accessed January 2020, for Stewart Computational Chemistry, Colorado Springs, CO.
[61]C. Bannwarth, E. Caldeweyher, S. Ehlert, A. Hansen, P. Pracht, J. Seibert, S. Spicher, and S. Grimme, "Extended tight-binding quantum chemistry methods," Wiley Interdiscip. Rev.: Comput. Mol. Sci. **11**, e1493 (2021).
[62]P. Eastman, J. Swails, J. D. Chodera, R. T. McGibbon, Y. Zhao, K. A. Beauchamp, L.-P. Wang, A. C. Simmonett, M. P. Harrigan, C. D. Stern, R. P. Wiewiora, B. R. Brooks, and V. S. Pande, "OpenMM 7: Rapid development of high performance algorithms for molecular dynamics," PLoS Comput. Biol. **13**, e1005659 (2017).
[63]G. Landrum, RDKIT: Cheminformatics and machine-learning software in C++ and Python. For the current version, see 10.5281/zenodo.591637; accessed January 2020. For the originating project, see https://www.rdkit.org/.
[64]S. Grimme, J. Antony, S. Ehrlich, and H. Krieg, DFTD3: Dispersion correction for DFT, Hartree–Fock, and semi-empirical quantum chemical methods. For the current version, see https://github.com/loriab/dftd3; accessed January 2020. For the originating project, see https://www.chemie.uni-bonn.de/pctc/mulliken-center/software/dft-d3.
[65]S. Ehrlich and E. Caldewayher, DFTD4: Generally applicable atomic-charge dependent London dispersion correction. For the current version, see https://github.com/dftd4/dftd4; accessed April 2021.
[66]E. Caldeweyher, C. Bannwarth, and S. Grimme, "Extension of the D3 dispersion coefficient model," J. Chem. Phys. **147**, 034112 (2017).
[67]H. Kruse and S. Grimme, GCP: Geometrical counterpoise correction for DFT and Hartree–Fock quantum chemical methods. For the current version, see https://www.chemie.uni-bonn.de/pctc/mulliken-center/software/gcp/gcp; accessed January 2020.
[68]H. Kruse and S. Grimme, "A geometrical correction for the inter- and intra-molecular basis set superposition error in Hartree–Fock and density functional theory calculations for large systems," J. Chem. Phys. **136**, 154101 (2012).
[69]C. Greenwell, MP2D: A program for calculating the MP2D dispersion energy. For the current version, see https://github.com/Chandemonium/MP2D; accessed January 2020.
[70]J. Řezáč, C. Greenwell, and G. J. O. Beran, "Accurate noncovalent interactions via dispersion-corrected second-order Møller–Plesset perturbation theory," J. Chem. Theory Comput. **14**, 4711–4721 (2018).
[71]X. Gao, TORCHANI: Accurate neural network potential on PyTorch. For the current version, see https://github.com/aiqm/torchani; accessed January 2020.
[72]J. S. Smith, O. Isayev, and A. E. Roitberg, "ANI-1: An extensible neural network potential with DFT accuracy at force field computational cost," Chem. Sci. **8**, 3192–3203 (2017).
[73]X. Gao, F. Ramezanghorbani, O. Isayev, J. S. Smith, and A. E. Roitberg, "TorchANI: A free and open source PyTorch-based deep learning implementation of the ANI neural network potentials," J. Chem. Inf. Model. **60**, 3408–3415 (2020).
[74]L. Frediani, E. Fossgaard, T. Flå, and K. Ruud, "Fully adaptive algorithms for multivariate integral equations using the non-standard form and multiwavelets with applications to the Poisson and bound-state Helmholtz kernels in three dimensions," Mol. Phys. **111**, 1143–1160 (2013).
[75]S. Hirata, "Tensor contraction engine: Abstraction and automated parallel implementation of configuration-interaction, coupled-cluster, and many-body perturbation theories," J. Phys. Chem. A **107**, 9887–9897 (2003).
[76]PROTOCOL BUFFERS: A language-neutral, platform-neutral extensible mechanism for serializing structured data. For the current version, see https://developers.google.com/protocol-buffers/docs/reference/proto3-spec; accessed May 2021.
[77]S. Seritan, K. Thompson, and T. J. Martínez, "TeraChem cloud: A high-performance computing service for scalable distributed GPU-accelerated electronic structure calculations," J. Chem. Inf. Model. **60**, 2126–2137 (2020).
[78]S. Seritan, C. B. Hicks, and J. E. Ford, TCPB: Python client for TeraChem's protobuf server mode. For the current version, see https://github.com/mtzgroup/tcpb-client; accessed May 2021.
[79]S. Grimme, "Semiempirical GGA-type density functional constructed with a long-range dispersion correction," J. Comput. Chem. **27**, 1787–1799 (2006).
[80]D. G. A. Smith, L. A. Burns, K. Patkowski, and C. D. Sherrill, "Revised damping parameters for the D3 dispersion correction to density functional theory," J. Phys. Chem. Lett. **7**, 2197–2203 (2016).
[81]E. Caldeweyher and J. G. Brandenburg, "Simplified DFT methods for consistent structures and energies of large systems," J. Phys.: Condens. Matter **30**, 213001 (2018).
[82]See https://openforcefield.org for OpenForceField.
[83]L.-P. Wang, D. G. A. Smith, and Y. Qiu, GEOMETRIC: A geometry optimization code that includes the TRIC coordinate system. For the current version, see https://github.com/leeping/geomeTRIC; accessed January 2020.
[84]L.-P. Wang and C. Song, "Geometry optimization made simple with translation and rotation coordinates," J. Chem. Phys. **144**, 214108 (2016).
[85]A. Heide and R. A. King, OptKing: A Python version of the Psi4 geometry optimizer. For the current version, see https://github.com/psi-rking/optking; accessed January 2020.
[86]J. Hermann, PYBERNY: Molecular structure optimizer. For the current version, see https://github.com/jhrmnn/pyberny; accessed January 2020. Also, for Version 0.6.2 https://doi.org/10.5281/zenodo.3695038.
[87]The finite difference and vibrational analysis procedures have been extracted from PSI4 as an independent module in QCDB. For the moment, there is a lingering library dependence on PSI4 for SALCs, so it, too, must be installed (as indicated in Fig. 3).
[88]B. H. Wells and S. Wilson, "van der Waals interaction potentials: Many-body basis set superposition effects," Chem. Phys. Lett. **101**, 429–434 (1983).
[89]P. Valiron and I. Mayer, "Hierarchy of counterpoise corrections for *N*-body clusters: Generalization of the Boys-Bernardi scheme," Chem. Phys. Lett. **275**, 46–55 (1997).
[90]I. Kaliman, LIBEFP: Parallel implementation of the effective fragment potential method. For the current version, see https://github.com/ilyak/libefp; accessed January 2020.
[91]I. A. Kaliman and L. V. Slipchenko, "LIBEFP: A new parallel implementation of the effective fragment potential method as a portable software library," J. Comput. Chem. **34**, 2284–2292 (2013).
[92]C. I. Bayly, P. Cieplak, W. Cornell, and P. A. Kollman, "A well-behaved electrostatic potential based method using charge restraints for deriving atomic charges: The RESP model," J. Phys. Chem. **97**, 10269–10280 (1993).
[93]A. Alenaizan, RESP: A restrained electrostatic potential (RESP) plugin to PSI4. For the current version, see https://github.com/cdsgroup/resp; accessed January 2020.
[94]A. Alenaizan, L. A. Burns, and C. D. Sherrill, "Python implementation of the restrained electrostatic potential charge model," Int. J. Quantum Chem. **120**, e26035 (2020).
[95]C. H. Borca, CRYSTALATTE: Automating the calculation of crystal lattice energies. For the current version, see https://github.com/carlosborca/CrystaLattE; accessed January 2020.
[96]C. H. Borca, B. W. Bakr, L. A. Burns, and C. D. Sherrill, "CrystaLattE: Automated computation of lattice energies of organic crystals exploiting the many-body expansion to achieve dual-level parallelism," J. Chem. Phys. **151**, 144103 (2019).
[97]The QCElemental molecule parsing machinery is also used for PSI4, so its documentation and http://docs.qcarchive.molssi.org/projects/QCElemental/en/latest/model_molecule.html can be helpful. The following describes the

particular case in the text. In Snippet 1, the `units bohr` string indicates that the Cartesian coordinates are already in QCSCHEMA's required atomic units, so these are unchanged in Snippet 2's `geometry` field. The Snippet 1 strings `O, H,` and `Ne` specify the elements and are processed into Snippet 2 fields atomic_numbers and symbols. The prefix character `@` to neon in Snippet 1 indicates it's a ghost atom, so the Snippet 2 field `real` shows a T, T, F pattern. `Gh(22Ne)` would have been equivalent to the given `@22Ne`. The prefix string 22 to neon in Snippet 1 specifies the mass number, much like a nuclide symbol. Thus the Snippet 2 fields `mass_numbers` and `masses` use default values for the oxygen and hydrogen but ^{22}Ne values for neon. `@Ne@21.99138511` to specify the mass value would have been equivalent. The strings `no_com` and `no_reorient` were not given in Snippet 1, so the fields `fix_com` and `fix_orientation` in Snippet 2 are F, meaning that the origin and frame of geometry are incidental to the `Molecule` specification. A user label like `O1` or `O_bigbasis` is parsed, but since Snippet 1 doesn't include any, the `atom_labels` field of Snippet 2 are empty strings. The`--`line of Snippet 1 indicates there are two `fragments` in the molecule, the first with two atoms and the second with one. This is encoded in the fragments field of Snippet 2. No charge/multiplicity lines are present in Snippet 1, either overall or per-fragment, so defaults are assigned. The second fragment is all ghosts and so is a neutral singlet. Electrons are never added or removed to the specification, so the first fragment is assigned neutral doublet, and the overall molecule is a neutral doublet. These defaults are reflected in the Snippet 2 fields `molecular_charge, molecular_multiplicity, fragment_charges,` and `fragment_multiplicities`. The string parser also stamps the schema name and provenance information in Snippet 2.

[98]In full, the command requests a Dunning 5ζ to 6ζ Helgaker-formula extrapolation of the MP2 correlation gradient performed by PSI4 with a coupled-cluster singles and doubles excitations correction (CCSD−MP2) at the Dunning triple-ζ to quadruple-ζ Helgaker-formula extrapolation gradient performed by NWCHEM with a CC up to quadruples excitations at cc-pVDZ performed by CFOUR, all atop an implicit 6-ζ Hartree–Fock.

[99]B. P. Pritchard, D. Altarawy, B. Didier, T. D. Gibson, and T. L. Windus, "New basis set exchange: An open, up-to-date resource for the molecular sciences community," J. Chem. Inf. Model. **59**, 4814–4820 (2019).
[100]I. Naoki, MESSAGEPACK-PYTHON: MessagePack serializer implementation for Python. For the current version, see https://github.com/msgpack/msgpack-python; accessed January 2020. For the originating project, see https://msgpack.org/.
[101]L. Bytautas, N. Matsunaga, T. Nagata, M. S. Gordon, and K. Ruedenberg, "Accurate *ab initio* potential energy curve of F_2. III. The vibration rotation spectrum," J. Chem. Phys. **127**, 204313 (2007).
[102]J. M. L. Martin, "Benchmark ab initio potential curves for the light diatomic hydrides. Unusually large nonadiabatic effects in BeH and BH," Chem. Phys. Lett. **283**, 283–293 (1998).
[103]B. Temelso, E. F. Valeev, and C. D. Sherrill, "A comparison of one-particle basis set completeness, higher-order electron correlation, relativistic effects, and adiabatic corrections for spectroscopic constants of BH, CH^+, and NH," J. Phys. Chem. A **108**, 3068–3075 (2004).
[104]J. S. Boschen, D. Theis, K. Ruedenberg, and T. L. Windus, "Accurate ab initio potential energy curves and spectroscopic properties of the four lowest singlet states of C_2," Theor. Chem. Acc. **133**, 1425 (2013).
[105]J. D. Bender, S. Doraiswamy, D. G. Truhlar, and G. V. Candler, "Potential energy surface fitting by a statistically localized, permutationally invariant, local interpolating moving least squares method for the many-body potential: Method and application to N_4," J. Chem. Phys. **140**, 054302 (2014).
[106]W. T. M. L. Fernando and P. F. Bernath, "Fourier transform spectroscopy of the $A^1\Pi$-$X^1\Sigma^+$ transition of BH and BD," J. Mol. Spectrosc. **145**, 392–402 (1991).
[107]K. P. Huber and G. Herzberg, *Constants of Diatomic Molecules* (Van Nostrand Reinhold, New York, 1979).
[108]M. Douay, R. Nietmann, and P. F. Bernath, "New observations of the $A^1\Pi_u - X^1\Sigma_g^+$ transition (Phillips system) of C_2," J. Mol. Spectrosc. **131**, 250–260 (1988).