Aalborg Universitet

cclib 2.0

An updated architecture for interoperable computational chemistry

Berquist, Eric; Dumi, Amanda; Upadhyay, Shiv; Abarbanel, Omri D; Cho, Minsik; Gaur, Sagar; Gano Gil, Victor Hugo; Hutchison, Geoffrey R; Lee, Oliver S; Rosen, Andrew S; Schamnad, Sanjeed; Schneider, Felipe S S; Steinmann, Casper; Stolyarchuk, Maxim; Vandezande, Jonathon E; Zak, Weronika; Langner, Karol M

Published in:
Journal of Chemical Physics

DOI (link to publication from Publisher):
10.1063/5.0216778

Creative Commons License
CC BY 4.0

Publication date:
2024

Document Version
Publisher's PDF, also known as Version of record

Link to publication from Aalborg University

Citation for published version (APA):
Berquist, E., Dumi, A., Upadhyay, S., Abarbanel, O. D., Cho, M., Gaur, S., Gano Gil, V. H., Hutchison, G. R., Lee, O. S., Rosen, A. S., Schamnad, S., Schneider, F. S. S., Steinmann, C., Stolyarchuk, M., Vandezande, J. E., Zak, W., & Langner, K. M. (2024). cclib 2.0: An updated architecture for interoperable computational chemistry. *Journal of Chemical Physics*, *161*(4), Article 042501. https://doi.org/10.1063/5.0216778

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal -

The Journal of Chemical Physics

RESEARCH ARTICLE | JULY 25 2024

cclib 2.0: An updated architecture for interoperable computational chemistry

Special Collection: Modular and Interoperable Software for Chemical Physics

Eric Berquist ; Amanda Dumi ; Shiv Upadhyay ; Omri D. Abarbanel ; Minsik Cho ; Sagar Gaur ; Victor Hugo Cano Gil ; Geoffrey R. Hutchison ; Oliver S. Lee ; Andrew S. Rosen ; Sanjeed Schamnad ; Felipe S. S. Schneider ; Casper Steinmann ; Maxim Stolyarchuk ; Jonathon E. Vandezande ; Weronika Zak ; Karol M. Langner

J. Chem. Phys. 161, 042501 (2024)
https://doi.org/10.1063/5.0216778

Articles You May Be Interested In

Software Development Of XML Parser Based On Algebraic Tools

AIP Conference Proceedings (December 2011)

Massively scalable workflows for quantum chemistry: BigChem and ChemCloud

J. Chem. Phys. (April 2024)

Raw-to-repository characterization data conversion for repeatable, replicable, and reproducible measurements

J. Vac. Sci. Technol. A (January 2020)

14 January 2025 07:10:56

AIP Publishing

The Journal of Chemical Physics

Special Topics Open for Submissions

Learn More

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

cclib 2.0: An updated architecture for interoperable computational chemistry

Cite as: J. Chem. Phys. 161, 042501 (2024); doi: 10.1063/5.0216778
Submitted: 30 April 2024 • Accepted: 1 July 2024 •
Published Online: 25 July 2024

View Online

Export Citation

CrossMark

Eric Berquist,[1] Amanda Dumi,[1] Shiv Upadhyay,[2] Omri D. Abarbanel,[3] Minsik Cho,[4] Sagar Gaur,[5,6] Victor Hugo Cano Gil,[7] Geoffrey R. Hutchison,[3] Oliver S. Lee,[8,9] Andrew S. Rosen,[10,11] Sanjeed Schamnad,[12] Felipe S. S. Schneider,[13,14] Casper Steinmann,[15] Maxim Stolyarchuk,[16] Jonathon E. Vandezande,[17,18] Weronika Zak,[19] and Karol M. Langner[20,a)]

AFFILIATIONS

[1]Sandia National Laboratories, Albuquerque, New Mexico 87185, USA
[2]Department of Chemistry, University of Washington, Seattle, Washington 98195, USA
[3]Department of Chemistry, University of Pittsburgh, 219 Parkman Avenue, Pittsburgh, Pennsylvania 15260, USA
[4]Department of Chemistry, Massachusetts Institute of Technology, 77 Massachusetts Ave., Cambridge, Massachusetts 02139, USA
[5]MarkovML 23, Geary St. Suite 600, San Francisco, California 94108, USA
[6]International Institute of Information Technology, Prof. CR Rao Road Gachibowli, Hyderabad 500032, Telangana, India
[7]Department of Chemistry, Carleton University, Ottawa, Ontario K1S 5B6, Canada
[8]Organic Semiconductor Centre, EaStCHEM School of Chemistry, University of St Andrews, St. Andrews KY16 9ST, United Kingdom
[9]Organic Semiconductor Centre, SUPA School of Physics and Astronomy, University of St Andrews, St. Andrews KY16 9SS, United Kingdom
[10]Department of Materials Science and Engineering, University of California, Berkeley, California 94720, USA
[11]Materials Science Division, Lawrence Berkeley National Laboratory, Berkeley, California 94720, USA
[12]Amazon.com Services LLC, Seattle, Washington 98109-5210, USA
[13]Department of Chemistry, Federal University of Santa Catarina, Trindade, Florianópolis SC 88040-900, Brazil
[14]Cellertz Bio, Santo Antônio de Lisboa, Florianópolis SC 88050-000, Brazil
[15]Department of Chemistry and Bioscience, Aalborg University, DK-9230 Aalborg, Denmark
[16]Independent Researcher, Paris, France
[17]Max-Planck-Institut für Kohlenforschung, Kaiser-Wilhelm-Platz 1, 45470 Mülheim an der Ruhr, Germany
[18]Schrödinger, Inc., 1 Main St., Cambridge, Massachusetts 02142, USA
[19]Department of Computer Science, Loughborough University, Epinal Way, Loughborough, Leicestershire LE11 3TU, United Kingdom
[20]Google DeepMind, Mountain View, California 94043, USA

Note: This paper is part of the JCP Special Topic on Modular and Interoperable Software for Chemical Physics.
[a)]**Author to whom correspondence should be addressed:** langner@google.com

ABSTRACT

Interoperability in computational chemistry is elusive, impeded by the independent development of software packages and idiosyncratic nature of their output files. The cclib library was introduced in 2006 as an attempt to improve this situation by providing a consistent interface to the results of various quantum chemistry programs. The shared API across programs enabled by cclib has allowed users to focus on results as opposed to output and to combine data from multiple programs or develop generic downstream tools. Initial development, however, did not anticipate the rapid progress of computational capabilities, novel methods, and new programs; nor did it foresee the growing need for customizability. Here, we recount this history and present cclib 2, focused on extensibility and modularity. We also introduce

14 January 2025 07:10:56

© Author(s) 2024

recent design pivots—the formalization of cclib's intermediate data representation as a tree-based structure, a new combinator-based parser organization, and parsed chemical properties as extensible objects.

© 2024 Author(s). All article content, except where otherwise noted, is licensed under a Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/). https://doi.org/10.1063/5.0216778

I. INTRODUCTION

Electronic structure methods are continuously progressing and evolving. This progress is manifested by an ever-changing computational ecosystem, the constant introduction of new programs and numerical methods, and increased functionality in existing programs. Both well-established codes and smaller scale projects developed within individual research groups are subject to these forces. For users organizing and processing the data from simulations, these changes pose a host of challenges. A particularly difficult one is the interoperability needed for post-processing simulations or for data transfer between software packages.

Ideally, in all computational chemistry programs, calculation and analysis methods would be decoupled and interact using a standardized format file with programmatic access (i.e., JSON, HDF5, TOML, etc.). In practice, this ideal has been extremely hard to approach due to highly specialized types of data, the fast pace of software development, and the lack of an agreed upon standard. There is also ambiguity in determining what intermediates should be stored on disk or discarded and recomputed during post-processing. All this leaves a need for a tool to facilitate extracting results from various codes, computing derived values of interest, and communicating values between programs.

It was precisely to address the need for such an intermediate interoperability layer that cclib was initially developed.[1] During over a decade and a half of development, there have been complementary efforts in developing file formats that enable interoperability such as Blue Obelisk,[2] Chemical JSON,[3,4] QCSchema[5] and TREXIO,[6] NOMAD,[7] and IOData,[8] to name a few. Inevitably, the variety of simulation types in the field of electronic structure leads to edge cases that break a standardized output format. In practice, this means a standardized format will necessarily exclude some portion of the electronic structure community. Particularly ill-served are the developers of new methods and smaller software packages that deviate from traditional calculation types. For practical use by these niche users, a representation of a calculation's data must be modular and extensible if it is to fit subdomain-specific needs while still maintaining a simple interface.

cclib has played a role in several scientific applications, some of which are highlighted below, but as the needs of the community grew, restrictions in the initial design became apparent. In this paper, we take a look back at cclib's role in computational chemistry and discuss the major changes underway in the code base for version 2 that will provide more utility for both users and developers. The subsequent Sec. III revisits the original work and how users interact with the library today. Section IV discusses its limitations and how they are addressed by the new features in cclib version 2.

II. THE STATE AND ROLE OF cclib

The library (https://github.com/cclib/cclib) has long been a part of the computational chemistry ecosystem and, during this time, has been used for a variety of projects, software applications, and publications. As a snapshot of impact, the original cclib literature report[1] has been cited by over 10K distinct authors in more than 400 different journals; the GitHub repository is used by over 180 other repositories and currently has around 100 forks.

In this section, we summarize the original implementation and discuss how cclib facilitates research efforts in computational chemistry. A closer look at several ways in which it is incorporated into scientific workflows provides practical context and demonstrates the range of possible utility.

To paraphrase the original literature report,[1] cclib is open-source (with a 3-clause BSD license[9]), written in Python, and meant for parsing and interpreting the results of computational chemistry packages. Specifically, the goals of cclib are

- to extract (parse) data from the output files generated by multiple programs,
- to provide a consistent interface to the results of computational chemistry calculations, particularly those results that are useful for algorithms or visualization,
- to facilitate the implementation of algorithms that are not specific to a particular computational chemistry package, and
- to maximize interoperability with other open source computational chemistry and cheminformatic software libraries.

Simple programmatic access to computational chemistry simulation data is the central feature of cclib, which enabled many of the scientific applications discussed below. Underlying this access is a data structure called ccData—an object generated from parsing log files—that can be consumed by downstream projects and methods directly or converted internally to other representations through "bridges," as shown in Fig. 1. There are a number of ways in which cclib is integrated into a broader workflow, either directly via its Python API or indirectly through another wrapping program. By integrating cclib into a workflow, one has access to all of the data that cclib can parse, in a ccData object, which can in turn be used to analyze excitation energies, reaction pathways, vibration transitions, and other properties of interest. Currently, cclib can parse over 70 attributes from more than a dozen different packages.

To better demonstrate the possible ways of interacting with cclib, in Fig. 2 we outline three mechanisms in which cclib can assist with the visualization of computational chemistry data (i.e., molecular orbitals and electron densities). The first, indirect

14 January 2025 07:10:56

© Author(s) 2024

TABLE I. An overview of the basic concepts and some idiosyncratic terminology used in `cclib`, spanning v1 and v2.

Term	Description
Program/package	A computer program or code (not cclib) that calculates electronic structure or other molecular properties
Log/output file	A text or binary formatted file containing the electronic structure information or other results from a computational chemistry program
Parser	A function or piece of code in cclib that extracts data from a log file
Parser combinator	A parser that combines multiple, more narrowly scoped parsers
ccData	The intermediate representation of data parsed from a log file is available to users
Tree	A data structure that holds information about how multiple ccData objects relate to each other
ccCollection	A generalization of ccData, containing other ccData objects and a tree of relationships between them
Attribute	A chemical property or other piece of data exposed to the user in a ccData object
Method	Code in cclib that, based on a ccData object, calculates derived properties such as population analysis
Bridge	A component of cclib that allows one to move between ccData and similar objects in other libraries

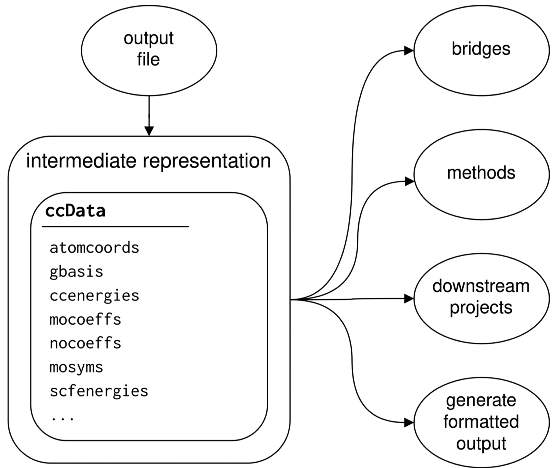


FIG. 1. Flow of information through the `cclib` library, highlighting the `ccData` intermediate representation alongside other components of the library. One or more output files are the typical starting point, which after parsing are distilled into a `ccData` object. From there, users may interact with other libraries to interoperate with other programs through bridges, use the data in downstream projects, write it to files in various output formats, or produce derived data using our collection of methods. Refer to Table I for an explanation of terminology.

mechanism is the integration of `cclib` within visualization software, as is the case with Avogadro.[10] Within Avogadro, a user can import a computational chemistry log file that will be internally parsed with `cclib` to extract the data needed for visualization. In this process, the user never explicitly interacts with `cclib`, yet it has played an integral role in the visualization by parsing the necessary data.

Another indirect pathway uses a software package that encapsulates `cclib` to generate files for subsequent consumption by a visualization program. For example, orbkit[11] includes a wrapper that uses `cclib` to parse output files and then generates a cube file from a `ccData` object, which can subsequently be processed by a visualization program. Another example of this pathway, outside the realm of visualization, is Goodvibes,[12,13] which has the ability to consume the `ccData` object and analyze molecular vibrations based on its parsed data.

The final mechanism is to use `cclib` directly from Python in a computational chemistry workflow. In this scenario, one parses a log file and then uses a `ccData` object for further analysis or to produce a specific file format for use in downstream code or later in another program. In the example visualization workflow presented in Fig. 2, a MOLDEN file is written after parsing a log file, a format readily accepted by many visualization programs.

While Fig. 2 only represents a fairly simple visualization pathway, `cclib` has facilitated other nontrivial scientific works, which we highlight here. At first glance, a random sampling of the latest citations for the original paper shows that most citations are actually for GaussSum, one of the earliest tools that integrated `cclib` as a core component.[1] The PubChemQC project, a large-scale molecular quantum chemistry database, used `cclib` to parse and organize computational chemistry data[14–16]—this endeavor was truly a high-throughput and data-oriented process, which contained 86×10^6 density functional theory calculations[15] and 221×10^6 semiempirical calculations.[16] St. John *et al.*[17] used `cclib` in a similar high-throughput workflow to generate a database of thermodynamic properties of organic molecules, which notably included 40 000 closed-shell molecules and 200 000 radicals. The generation of potential energy surfaces is a data intensive task that can yield great dividends since spectroscopic quantities and experimental observables can be obtained from a high quality potential energy surface. In the work of Abbott *et al.*,[18] `cclib` was used as one avenue to parse the many electronic structure calculations needed to fit a potential energy surface. Finally, we note that `cclib` can be used for "small data" chemistry as well. The theoretical work of Rahm and Hoffmann[19] proposed a new experimentally-observable quantity, allowing for an additional connection between experiment and theory. The authors also provided a script that used `cclib` to parse the required electronic structure information from several programs and then computed the new observable.[19]

14 January 2025 07:10:56

© Author(s) 2024

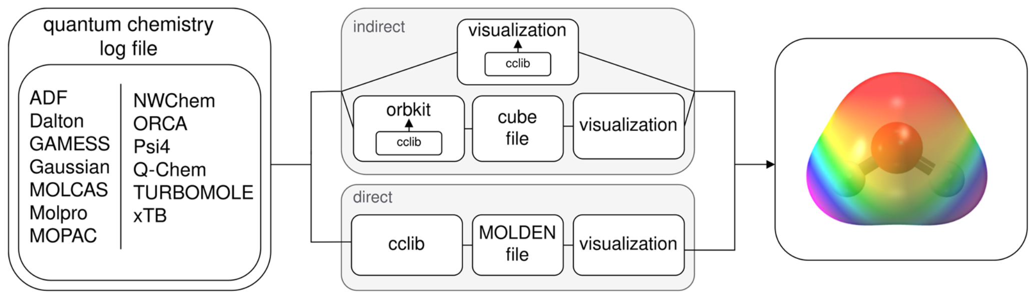


FIG. 2. Three typical mechanisms for using `cclib` in larger workflows, based on a hypothetical visualization goal. Software packages that `cclib` can extract data from are listed in the box on the left. Users may interact directly with `cclib` (lower gray box in center) to perform numerical analyses or to generate output files for other programs. Alternatively, other programs may rely on `cclib` in their own implementation to parse data, and users do not directly interact with cclib in this case (upper gray box in center).

Looking back on version 1 and the milestones we reached internally, it is worthwhile to mention aspects of the development ecosystem that enabled `cclib` to serve many different purposes, as well as the projects that benefited from incorporating `cclib`. The initial version of `cclib` supported five parsers, which have almost tripled to currently include ADF Dalton,[20,61] Gaussian,[21] GAMESS,[22] Jaguar,[23] MOLCAS,[24] Molpro,[25] NBO,[26] NWChem,[27] ORCA,[28] Psi4,[29] Q-Chem,[30] TURBOMOLE,[31] and xTB.[32] In addition, formatted checkpoint (FChk) files produced by Gaussian, Q-Chem, and Psi4 can be parsed, along with GAMESS `*.dat` files. Some programs also produce multiple output files by default, such as Molpro and Turbomole, requiring the parsing of multiple files and merging results into a single `ccData` object.

In addition to parsing log files, `cclib` offers the following analysis methods: C squared, Mulliken, Löwdin, Bickelhaupt, and Hirshfeld population analysis, density matrix calculation, Mayer's bond order, charge decomposition analyses, Bader's QTAIM, DDEC6, and nuclear properties. Furthermore, bridges exist to interface `ccData` objects to other programs, including Atomic Simulation Environment (ASE),[33] biopython,[34] Horton,[35] Open Babel,[36] Psi4,[29] PyQuante,[37] and PySCF.[38] In addition to following traditional semantic versioning practices, literature-citeable versions of `cclib` are also available through Zenodo, from 1.2[39] to 1.8,[40] with a new citation for each minor release. Releases of `cclib` can be installed through conda-forge,[41] PyPI,[42] Spack,[43] and Nix-QChem.[44]

Much of the success of `cclib` is a result of community contributions, which have been facilitated through open-source code development. The open-source software development paradigm is becoming widely adopted as a common approach to develop computational chemistry tools. As discussed previously, this approach to development has several advantages, such as community contributions, organic growth, distributed maintenance and consensus building, and rapid real-world user insight into the product. `cclib` has seen all these benefits as a result of its open source nature, and the v2 effort stems directly from it. For example, the issue of nonextendable attributes was highlighted early on in `cclib`'s development (GitHub issue No. 227), and a suitable solution was found through several iterations of discussion (GitHub issues No. 419 and No. 398).

Another contributing factor to `cclib`'s positive development workflow has been participation in Google Summer of Code program (GSoC, https://summerofcode.withgoogle.com/). Since 2016, `cclib` has participated in GSoC, under the Open Chemistry umbrella group alongside Avogadro, Open Babel,[36] DeepChem,[45] RDKit,[46] gnina,[47] 3Dmol.js,[48] NWChem,[27] and Psi4.[29] Open Chemistry maintains a yearly list of project ideas, the latest iteration of which is located at https://wiki.openchemistry.org/GSoC_Ideas_2024. `cclib` has gained a number of sizable contributions through GSoC, which would have been time-consuming or infeasible with only the core team.

III. DESIGN DECISIONS FOR THE FUTURE OF `cclib`

Although the initial design of `cclib` has facilitated real-world scientific applications, as described in the previous section, we encountered difficulties as the project's scope and capabilities scaled up. One of the challenges was that initially, in `cclib`, parsing was an all-or-nothing task. In other words, if an error occurred while parsing a log file, no data were returned, even if some subset of attributes were successfully parsed. As the number of attributes grew, this failure mode became more frequent. Version 1 of `cclib` was also unable to handle cases of nested output, programs called within other programs, which is an increasingly common way to use several programs such as NBO,[26] xTB,[32] and CFOUR.[49]

Another restriction of the initial design was that parsers were separate and monolithic for each supported package, which eventually made the parser code difficult to read, understand, extend, and maintain. Changes to the parsing of a particular attribute in the large, unified parsing code for a package often had unexpected secondary effects for other attributes. From a user's perspective, this problem made it hard to "turn off" attributes to save parsing time, which becomes an issue when parsing thousands or even millions of output files. For developers, extending or tweaking how a specific package or attribute is parsed became difficult. As a result of this parsing inflexibility, the `ccData` object was also inflexible to the introduction of new types of data and extending existing

14 January 2025 07:10:56

© Author(s) 2024

ones. Since `cclib` has a reasonably robust test and regression suite, the friction arose early, as tests unrelated to current work started to fail. In the end, all this prevented the rapid prototyping of new attributes and the extension or generalization of existing attributes.

In the following sections, we introduce the major design shifts v2 of the library brings to address these shortcomings: an intermediate tree-based internal data representation, a parser combinator framework for parsing, and a class-based representation for attributes. Each of these changes is discussed separately, but they are synergistic and work together to make `cclib` much more flexible and adaptable to future changes in computational chemistry. We note here that `cclib` uses Semantic versioning, and therefore, the changes introduced to the API are in general breaking. The changes are implemented in a way that minimizes the impact on users, but adjustments will be necessary to migrate existing use cases.

A. Toward a formalized intermediate data representation

Drawing inspiration from compiler theory, we co-opt the concept of an intermediate representation.[50] In the context of compilers, an intermediate representation is a language-agnostic way to capture the intentions of the original source code that can be used for downstream applications such as optimization and conversion to machine code. Using an intermediate representation, the same operations of optimization and conversion to machine code can be applied to a variety of programming languages.

There is a parallel situation in computation chemistry, where the variety of software packages generating output files with similar content corresponds to programming languages, and the consumption of data by downstream projects, used for analysis or downstream processing, corresponds to the optimization or conversion to machine code.

With these relations in mind, `ccData` is a sort of intermediate representation for `cclib` version 1, which is also evident in Fig. 1. We note that the `ccData` object was not explicitly designed but developed organically during the lifetime of `cclib`; its utility as an intermediate representation became apparent after extensive user feedback and development, underscoring how relevant it is to `cclib`'s functionality. Incremental design, however, has led to limitations in the initial approach. In the next section, these constraints and the design decisions motivating a new design are described.

B. Tree-based internal representation

One of the difficulties with `ccData` objects in version 1 was their rigidity. It is unable to support multiple related outputs in a general and consistent manner. For instance, when running multiple calculations in the same input file, such as a geometry optimization followed by a harmonic frequency analysis, there was no way to parse both the geometry optimization and frequency analysis and indicate their relationship in the data object offered to the user. Fragment or subsystem calculations are excellent examples where this need arises: basis set superposition error (BSSE), energy decomposition analysis (EDA), symmetry-adapted perturbation theory (SAPT), "Our own N-layered Integrated molecular Orbital and Molecular mechanics" (ONIOM), many-body expansions, and other forms of density embedding methods. The same difficulty arises when running "code-within-code" or "program-within-program" calculations, such as an NBO analysis inside of a Gaussian calculation or using Psi4 to drive CFOUR or MRCC.

In `cclib` version 2, we address these difficulties with `ccData` by adopting a tree-based internal representation. A tree structure allows one to express connections between data and organize related parsed pieces of data into a larger whole. We implement a tree-based representation by using two key data structures—a simple tree in which nodes are visited depth-first, where each node is composed of a `ccData` object and a `ccCollection` that serves to contain and navigate the tree structure.

For simple output files such as single point calculations, the tree definition is straightforward. Examples where the `ccCollection` is nontrivial and meaningful are given in listings 1 and 2. Listing 1 and Fig. 3 show the explicit construction of the tree and illustrate how a user interacts with it, and listing 2 shows the utility of the `ccCollection` for fragment-based calculations. In the latter of these two examples, the tree root represents the supermolecular system, and each child node holds information about each particular subsystem. Upon calling `ccread`, a parsing driver is constructed using this tree structure, and the parser visits each node of the tree when appropriate while extracting data from the log file (see the next section for more details on the parsing mechanics). Once parsing is complete, data can be accessed directly by indexing the resulting `ccCollection` object and inspecting the nodes of the tree. Note that the ccData nodes are stored as a flat list within the `ccCollection`, which is indexed given the structure of the tree.

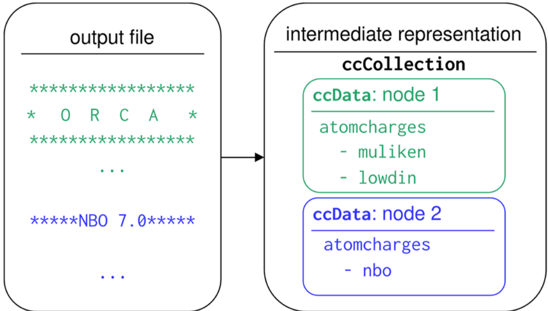


FIG. 3. The new `ccCollection` intermediate representation that is returned when parsing a log file in `cclib` 2. In this example, the input contains nested results from two different computational chemistry software packages, ORCA and NBO. Each node in the resulting `ccCollection` is a `ccData` object containing program-specific information.

14 January 2025 07:10:56

© Author(s) 2024

Listing 1. A demonstration of the new capability to parse output consisting of nested analyses from different packages. In this example, an NBO analysis is embedded inside an ORCA log file, and the latter is a child node of the former in the parse tree.

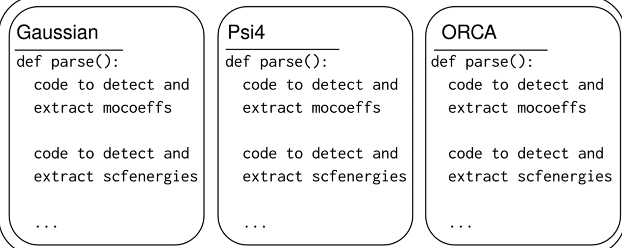
```
from cclib.io import ccread
from cclib.tree import Tree

# tree for code-in-code data
multiprogram_tree = Tree()
multiprogram_tree.add_root()  # node for ORCA data
multiprogram_tree.add_child(0) # node for NBO data

collection = ccread("water_mp2.out",
↪  tree=multiprogram_tree)

# OUTPUT
# ORCA atomcharges
collection[0].atomcharges
> {'mulliken': [-0.361718, 0.180855, 0.180864],
   'lowdin': [-0.2507, 0.125348, 0.125352]}
# NBO atomcharges
collection[1].atomcharges
> {'nbo': [-0.39652, 0.19826, 0.19826]}
```

Listing 2. An example of the structure of the tree resulting from parsing an output consisting of multiple fragment calculations. In this case, a BSSE correction calculation was carried out for a water dimer, where each water molecule is a separate fragment, and fragment data is stored in the child nodes of the root dimer node.

```
from cclib.io import ccread
from cclib.tree import Tree

bsse_tree = Tree()
bsse_tree.add_root()   # supramolecular calculation
bsse_tree.add_child(0) # monomer1 calculation
bsse_tree.add_child(0) # monomer2 calculation


collection = ccread("mp2_water_dimer_bsse.out",
↪  tree=bsse_tree)
print(f"Supramolecular energy
↪  {collection[0].scfenergies:}")
for i in range(1,3):
    print(f"Monomer {i:} energy
    ↪  {collection[i].scfenergies:}")


# Calculating BSSE
bsse = collection[0].scfenergies[0] -
↪  (collection[1].scfenergies[0] +
↪  collection[2].scfenergies[0])
print(f"BSSE {bsse}")

# OUTPUT
> Supramolecular energy [-4139.575596491689]
> Monomer 1 energy [-2069.713357897148]
> Monomer 2 energy [-2069.70834493012]
> BSSE -0.15389366442104802
```

C. Parser combinators

In version 2, a parser comprises other, smaller parsing functions, each of which is responsible for parsing a single property for



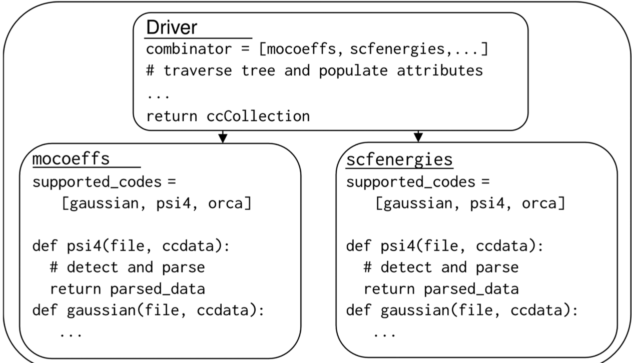


FIG. 4. Parsers were monolithic for each supported package in version 1 (above), while in version 2, parsers are composable from smaller components responsible for extracting individual attributes (below).

a specific software package. On the user side, the default behavior remains relatively unchanged in that `cclib` extracts all possible attributes unless configured otherwise. As an elaboration of the default behavior, this design change is meant to introduce customizability into how parsers work and enable flexible behavior. If only a certain attribute is required (i.e., only `scfenergies`), a unique combinator can be defined to only extract this information, skipping all other output. This particular approach is especially valuable to those working toward high-throughput processing or machine learning efforts, where performance can be optimized with such selective parsing. Parser combination also allows for easily extending what `cclib` extracts, giving users the ability to compose their own parser for properties that are not in the official release of the code or before such additions can be incorporated into the official release.[51,52] Figure 4 demonstrates how a combinator approach can lead to modular parser definitions for specific properties.

D. Attributes as classes

Another inflexibility of the original `ccData` is that parsed attributes are stored as primitive data types, and there may only be a single instance of each parsed attribute attached to each `ccData` instance. This provides a guaranteed lowest-common denominator for downstream consumers but misses out on the nuances of different packages, lacks provenance on exactly what was parsed, and cannot present variations of parsed data to users.

For example, when performing Configuration Interaction Singles (CIS) or Time Dependent Density Functional Theory (TDDFT)

14 January 2025 07:10:56

© Author(s) 2024

calculations with ORCA, spin–orbit coupling corrections may be requested, leading to multiple sets of excitation energies and transition moments; keeping both the uncorrected and corrected results is useful when studying the effect of relativity on excited states. However, since in v1, excited states and transition moments were stored in the rigidly typed attributes outlined in Table II, one set must be chosen. Some flexibility was given by the catch-all `transprop` attribute, where dictionary keys are the headers for each SOC-corrected spectrum. However, because there is no facility for handling the increased splitting for different spin states, only the uncorrected results were presented to the user in `etenergies`. In v2, the `etenergies` attribute also provides uncorrected excitation energies by default, and additional sets of energies are available on the object.

To add such flexibility and avoid overly narrow top-level attributes such as `etveldips` (specific to relativistic effects), attributes are now represented by objects. Each attribute type is declared by a class that inherits from a base `Attribute` class, as demonstrated in listing IV D. This design allows one to add custom (meta)data and behaviors where needed, which extend the data and methods common to all attributes.

One such planned extension is to tag numeric values where appropriate with units using the pint[53] library and implement methods that convert values between different unit systems. For example, in the case of energies, one may want to convert not only between atomic and SI units but also other commonly-used energy units such as electron volts, wavenumbers, and thermochemical calories. With the flexibility of an object-based design, we can implement this type of addition using Python decorators, method-based wrappers, or another coding paradigm. In fact, the implementation may change over time while keeping a constant user facing API as the library continues to evolve. For unit conversions, this will increase the clarity of the syntax and eliminate repetitive boilerplate while enabling custom conversions that we struggled to incorporate with the original design.

As another example, each attribute can be responsible for generating its own serialized representation. This becomes relevant when creating an output file from a `ccData` object. There is a default way to serialize all data types but an outer-level driver routine calls special methods on each attribute attached to the `ccData` object if they exist. The potentially customized representations are combined prior to export. For the `gbasis` attribute (which contains basis set parameters), such a customized output representation includes a `[GTO]` block for MOLDEN output, a `<Primitive Exponents>` block for WFX output, and a nested `atoms:orbitals:basis functions` object in Chemical JSON output.

There exist more kinds of derived data that are easier to accommodate with this paradigm, such as atomic charges calculated with different methods (Mulliken, Löwdin, CM5, Hirshfeld, etc.), electrostatic moments in various coordinate systems, and properties that are exclusive to specific computational packages. This design change also allows for fine grained control and precise tuning of features that are independent of the parsing program, such as type definition checks. All these improvements are critical to provide a consistent experience for downstream data consumers, and we look forward to enhancing many attributes with meaningful behaviors and seeing users contribute their own ideas.

Listing 3. In the new version of `cclib`, attributes are represented by dedicated objects, which implement the `Attribute` class by either inheritance or structural subtyping. Attributes defined in this way share common functionalities, such as validation (i.e., type checking), and allow for adding or modifying behaviors specific to particular attributes (such as unit conversion). This example is pseudo Python code, and the actual implementation may change as version 2 evolves into a non-alpha initial release.

```
class Attribute:
  name: str
  type: ...
  value: ...

  def validate(self) -> bool:
    """Checks value type and other constraints."""

class Etenergies(Attribute):
  name: str = "etenergies"
  type: np.ndarray

  def to_electronvolts(self) -> ndarray[float]:
      ...

class Mocoeffs(Attribute):
  name: str
  type: np.ndarray

  def is_orthogonal(self, S: aooverlaps) -> bool:
      ...
```

IV. LOOKING FORWARD

Compared to the original `cclib` paper[1] and the first version of the library, the design principles, architecture, and core development process have largely not changed over 18+ years of the project's existence. This speaks to the robustness of the API concepts `cclib` introduced, including attributes, post-parsing methods, and bridges to other programs. The library has grown and adapted over the years to changes in both the computational chemistry world and Python language. Perhaps the biggest change `cclib`'s developers have noticed, however, is the magnified importance of extensibility and collaborative development. Moving from Subversion to Git for version control and from Sourceforge to GitHub for hosting have been the two largest shifts so far to improve in this area. The open ended design of version 2, compared to the original, is the next natural step we intend to take with the project.

It is also worth reflecting on the tension that exists between libraries like `cclib` and their users. In this paper, we focused on several design ideas adopted from computer science (intermediate representations, combinatory parsing, trees, and classes). And the original paper describes in great detail the content and importance of unit and regression tests. Although the project has had little trouble guiding contributors of all backgrounds in adding new code and tests, most feature requests, bug reports, bug fixes, and feature additions have been from users who may see themselves more as scientists than software developers (based on anecdotal evidence). Even though version 2 represents a paradigm shift for how `cclib`

14 January 2025 07:10:56

© Author(s) 2024

TABLE II. All attributes used for holding data related to excited states in cclib v1.8.1. Overall, there are more than 70 attributes, which can be reviewed in the documentation at https://cclib.github.io/data.html.

Attribute name	Description	Python type
Etenergies	Energies of electronic transitions	numpy.ndarray[float] (1D)
Etoscs	Oscillator strengths of electronic transitions	numpy.ndarray[float] (1D)
Etdips	Electric transition dipoles of electronic transitions	numpy.ndarray[float] (2D)
Etveldips	Velocity–gauge electric transition dipoles of electronic transitions	numpy.ndarray[float] (2D)
Etmagdips	Magnetic transition dipoles of electronic transitions	numpy.ndarray[float] (2D)
Etrotats	Rotatory strengths of electronic transitions	numpy.ndarray[float] (1D)
Etsecs	Singly excited configurations for electronic transitions	list[list[tuple[tuple[int, int], tuple[int, int], float]]]
Etsyms	Symmetries of electronic transitions	list[str]
Transprop	All absorption and emission spectra	dict[str, (etenergies, etoscs)]

works and employs a more abstract design, the core purpose for cclib's existence remains: to facilitate interoperability and provide the simplest possible, streamlined analysis tools for computational chemistry workflows.

We recognize that this mission now requires support for a wider variety of scientific applications and easy extension to specific use cases. To this end, cclib v2 will continue to evolve in at least the following possible ways:

- The user facing API will converge as the non-alpha v2 release is finalized. The examples provided in this paper will always continue to work with v2, but we do expect simpler, more direct ways to emerge for the most basic functionality (like parsing simple output files). The final public API may be compatible with v1 syntax in many cases, but the backward incompatible assumption of a new major version will allow us to incrementally improve after extensive testing and in response to user feedback.
- A particular aspect of the emerging v2 API that we still find lacking is that trees representing parsed data need to be built explicitly. We can automate this step for specific examples today, and plan to add API components that do this for users and make it easy to contribute such recipes.
- The design pivots in v2 have yielded a more modular library with hierarchical parsers and data representations. Our primary goal when moving in this direction was to make it easier for users to contribute new components or to build their own capabilities without making changes to a monolithic codebase.
- Python is still the de facto programming language in scientific computing, but others are increasing in popularity; one recent GSoC project created cclib bindings for Julia (https://github.com/cclib/Cclib.jl), for example. The cclib library is in a good position to lean into this trend if it continues.

Most importantly, the design changes described here will allow users to parse more calculation types. There are many calculation types, and some make a universal intermediate representation of computational chemistry data challenging. We believe the flexible design principles baked into v2 will enable developers and the community to tackle the following challenging cases:

- Periodicity—Is the calculation aperiodic, low dimensionally periodic, or fully periodic? The shape and storage of quantities such as molecular coefficients (MOs) depend on the answers and could be accommodated with special parser components and attribute variants.
- Noncollinear and/or relativistic methods—The shapes of stored quantities (e.g., MOs and densities) are different when running noncollinear or relativistic calculations. Four component relativistic information, for example, could be added via an attribute class with a greater dimension for its data than the nonrelativistic case, and a new attribute parser would be used to parse the relevant information.
- Multicomponent or coupled calculations—A representation would need to be flexible enough to describe cases where the electronic degrees of freedom are coupled to quantum nuclei, vibrations and phonons, or photons.[54–60]
- New and advanced basis sets—Most codes use Gaussian basis sets, but others exist such as Slater, grid, wavelet, Gausslet, plane wave, etc. The parameters defining various types of basis sets have little similarity and greatly impact the size and dimensionality of computed data.
- Novel or non-standard calculation methods—Electronic structure is most often treated with density-based or wave function-based variational methods but the field is rich with a variety of other approaches such as perturbative, stochastic, and embedded methods. All of these could be supported by introducing targeted parser components and additional attributes.

V. CONCLUSIONS

cclib continues to play a role in the parsing, storage, and transfer of computational chemistry data, both in workflows that combine different programs and in post-processing workflows. While the initial version of the library has helped users in

14 January 2025 07:10:56

© Author(s) 2024

scientific studies for over a decade, it has become clear that a change in design is needed. In a new version of `cclib`, we build upon the original strengths of `cclib` and generalize them into a more flexible architecture with extensible components. This new version is a step improvement in three ways: the core parsing functionality is redefined in terms of combinators of modular primitive parsers, molecular properties, and other attributes become objects that users may build upon, and a tree data structure encapsulates parsed attributes in a hierarchical intermediate representation. We demonstrate novel capabilities by parsing two scenarios that were impossible with the previous version of `cclib`, namely, a fragment BSSE calculation and a "code-within-a-code" calculation where one program is called from another. These improved abstractions, alongside the new capabilities they unlock, represent a paradigm shift in the design of `cclib` that will assist current and future users of the library for hopefully at least another decade. Version 2 is currently available as an alpha release at https://github.com/cclib/cclib/releases.

SUPPLEMENTARY MATERIAL

The supplementary material contains two examples that demonstrate functionality not available previously, namely, parsing BSSE energies and parsing atomic charges computed by calling one program from inside another. Both examples include input and output files and Python scripts for cclib 2.0, and a README file provides a detailed overview.

ACKNOWLEDGMENTS

The authors would like to thank all the open source contributors to `cclib` since its inception, on the SourceForge platform and later on GitHub; there have been more than 70 distinct contributors across the nearly 5000 commits in the project's history so far. The authors would also like to acknowledge Google Summer of Code for funding student projects related to cclib via the Open Chemistry project (https://www.openchemistry.org/gsoc/) since 2016.

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC (NTESS), a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration (DOE/NNSA) under Contract No. DE-NA0003525. This written work is authored by an employee of NTESS. The employee, not NTESS, owns the right, title, and interest in and to the written work and is responsible for its contents. Any subjective views or opinions that might be expressed in the written work do not necessarily represent the views of the U.S. Government. The publisher acknowledges that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish, or reproduce the published form of this written work or allow others to do so for U.S. Government purposes. The DOE will provide public access to the results of federally sponsored research in accordance with the DOE Public Access Plan.

A.S.R. acknowledges support via a Miller Research Fellowship from the Miller Institute for Basic Research in Science, University of California, Berkeley. We also thank Kunal Sharma, who supported cclib development as part of a Google Summer of Code project in 2018, while a student at Department of Chemistry, Birla Institute of Technology and Science, Pilani, India.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

E.B, A.D, and S.U contributed equally to this work.

Eric Berquist: Conceptualization (equal); Software (equal); Supervision (equal); Writing – original draft (equal); Writing – review & editing (equal). **Amanda Dumi**: Conceptualization (equal); Software (equal); Supervision (equal); Writing – original draft (equal); Writing – review & editing (equal). **Shiv Upadhyay**: Conceptualization (equal); Software (equal); Supervision (equal); Writing – original draft (equal); Writing – review & editing (equal). **Omri D. Abarbanel**: Software (supporting). **Minsik Cho**: Software (supporting). **Sagar Gaur**: Software (supporting). **Victor Hugo Cano Gil**: Software (supporting). **Geoffrey R. Hutchison**: Conceptualization (supporting); Software (supporting); Supervision (supporting). **Oliver S. Lee**: Software (supporting). **Andrew S. Rosen**: Software (supporting). **Sanjeed Schamnad**: Software (supporting). **Felipe S. S. Schneider**: Software (supporting). **Casper Steinmann**: Software (supporting). **Maxim Stolyarchuk**: Software (supporting). **Jonathon E. Vandezande**: Software (supporting). **Weronika Zak**: Software (supporting). **Karol M. Langner**: Conceptualization (equal); Software (equal); Supervision (lead); Writing – original draft (equal); Writing – review & editing (equal).

DATA AVAILABILITY

The `cclib` code, input, and output files used to generate the values for the listings above can be found in the supplementary material. The data that support the findings of this study are openly available in cclib/cclib, at https://github.com/cclib/cclib.

REFERENCES

[1] 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).

[2] N. M. O'Boyle, R. Guha, E. L. Willighagen, S. E. Adams, J. Alvarsson, J.-C. Bradley, I. V. Filippov, R. M. Hanson, M. D. Hanwell, G. R. Hutchison, C. A. James, N. Jeliazkova, A. S. Lang, K. M. Langner, D. C. Lonie, D. M. Lowe, J. Pansanel, D. Pavlov, O. Spjuth, C. Steinbeck, A. L. Tenderholt, K. J. Theisen, and P. Murray-Rust, "Open data, open source and open standards in chemistry: The blue obelisk five years on," J. Cheminf. **3**, 37 (2011).

[3] M. D. Hanwell, W. A. de Jong, and C. J. Harris, "Open chemistry: RESTful web APIs, JSON, NWChem and the modern web application," J. Cheminf. **9**, 55 (2017).

[4] M. D. Hanwell, C. Harris, A. Genova, M. Haghighatlari, M. El Khatib, P. Avery, J. Hachmann, and W. A. de Jong, "Open Chemistry, Jupyterlab, REST, and quantum chemistry," Int. J. Quantum Chem. **121**, e26472 (2021).

[5] 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

14 January 2025 07:10:56

© Author(s) 2024

open-source platform to compute, organize, and share quantum chemistry data," Wiley Interdiscip. Rev.: Comput. Mol. Sci. **11**, e1491 (2021).
[6]E. Posenitskiy, V. G. Chilkuri, A. Ammar, M. Hapka, K. Pernal, R. Shinde, E. J. Landinez Borda, C. Filippi, K. Nakano, O. Kohulák, S. Sorella, P. de Oliveira Castro, W. Jalby, P. L. Ríos, A. Alavi, and A. Scemama, "TREXIO: A file format and library for quantum chemistry," J. Chem. Phys. **158**, 174801 (2023).
[7]M. Scheidgen, L. Himanen, A. N. Ladines, D. Sikter, M. Nakhaee, Á. Fekete, T. Chang, A. Golparvar, J. A. Márquez, S. Brockhauser, S. Brückner, L. M. Ghiringhelli, F. Dietrich, D. Lehmberg, T. Denell, A. Albino, H. Näsström, S. Shabih, F. Dobener, M. Kühbach, R. Mozumder, J. F. Rudzinski, N. Daelman, J. M. Pizarro, M. Kuban, C. Salazar, P. Ondračka, H.-J. Bungartz, and C. Draxl, "NOMAD: A distributed web-based platform for managingmaterials science research data," J. Open Source Softw. **8**, 5388 (2023).
[8]T. Verstraelen, W. Adams, L. Pujal, A. Tehrani, B. D. Kelly, L. Macaya, F. Meng, M. Richer, R. Hernández-Esparza, X. D. Yang, M. Chan, T. D. Kim, M. Cools-Ceuppens, V. Chuiko, E. Vöhringer-Martinez, P. W. Ayers, and F. Heidar-Zadeh, "IOData: A python library for reading, writing, and converting computational chemistry file formats and generating input files," J. Comput. Chem. **42**, 458–464 (2021).
[9]Open Source Initiative, 3-Clause BSD License (2024).
[10]M. D. Hanwell, D. E. Curtis, D. C. Lonie, T. Vandermeersch, E. Zurek, and G. R. Hutchison, "Avogadro: An advanced semantic chemical editor, visualization, and analysis platform," J. Cheminf. **4**, 17 (2012).
[11]G. Hermann, V. Pohl, J. C. Tremblay, B. Paulus, H.-C. Hege, and A. Schild, "ORBKIT: A modular python toolbox for cross-platform postprocessing of quantum chemical wavefunction data," J. Comput. Chem. **37**, 1511–1520 (2016).
[12]G. Luchini, R. Paton, J. Alegre-Requena, J. Rodríguez-Guerra, E. Berquist, J. Chen, IFunes, J. Velmiskina, froessler, H. Mayes, and S. S. S. Vejaykummar, and sibo (2022). "Patonlab/GoodVibes: Bug fixes & updated references," .
[13]G. Luchini, J. Alegre-Requena, I. Funes-Ardoiz, and R. Paton, "Goodvibes: Automated thermochemistry for heterogeneous computational chemistry data [version 1; peer review: 2 approved with reservations]," F1000Research **9**, 291 (2020).
[14]M. Nakata and T. Shimazaki, "PubChemQC project: A large-scale first-principles electronic structure database for data-driven chemistry," J. Chem. Inf. Model. **57**, 1300–1308 (2017).
[15]M. Nakata and T. Maeda, "PubChemQC B3LYP/6-31G*//PM6 data set: The electronic structures of 86 million molecules using B3LYP/6-31G* calculations," J. Chem. Inf. Model. **63**, 5734–5754 (2023).
[16]M. Nakata, T. Shimazaki, M. Hashimoto, and T. Maeda, "PubChemQC PM6: Data sets of 221 million molecules with optimized molecular geometries and electronic properties," J. Chem. Inf. Model. **60**, 5891–5899 (2020).
[17]P. C. St. John, Y. Guan, Y. Kim, B. D. Etz, S. Kim, and R. S. Paton, "Quantum chemical calculations for over 200,000 organic radical species and 40,000 associated closed-shell molecules," Sci. Data **7**, 244 (2020).
[18]A. S. Abbott, J. M. Turney, B. Zhang, D. G. A. Smith, D. Altarawy, and H. F. Schaefer, "PES-Learn: An open-source software package for the automated generation of machine learning models of molecular potential energy surfaces," J. Chem. Theory Comput. **15**, 4386–4398 (2019).
[19]M. Rahm and R. Hoffmann, "Toward an experimental quantum chemistry: Exploring a new energy partitioning," J. Am. Chem. Soc. **137**, 10282–10291 (2015).
[20]K. Aidas, C. Angeli, K. L. Bak, V. Bakken, R. Bast, L. Boman, O. Christiansen, R. Cimiraglia, S. Coriani, P. Dahle, E. K. Dalskov, U. Ekström, T. Enevoldsen, J. J. Eriksen, P. Ettenhuber, B. Fernández, L. Ferrighi, H. Fliegl, L. Frediani, K. Hald, A. Halkier, C. Hättig, H. Heiberg, T. Helgaker, A. C. Hennum, H. Hettema, E. Hjertenæs, S. Høst, I.-M. Høyvik, M. F. Iozzi, B. Jansík, H. J. Aa. Jensen, D. Jonsson, P. Jørgensen, J. Kauczor, S. Kirpekar, T. Kjærgaard, W. Klopper, S. Knecht, R. Kobayashi, H. Koch, J. Kongsted, A. Krapp, K. Kristensen, A. Ligabue, O. B. Lutnæs, J. I. Melo, K. V. Mikkelsen, R. H. Myhre, C. Neiss, C. B. Nielsen, P. Norman, J. Olsen, J. M. H. Olsen, A. Osted, M. J. Packer, F. Pawlowski, T. B. Pedersen, P. F. Provasi, S. Reine, Z. Rinkevicius, T. A. Ruden, K. Ruud, V. V. Rybkin, P. Sałek, C. C. M. Samson, A. S. de Merás, T. Saue, S. P. A. Sauer, B. Schimmelpfennig, K. Sneskov, A. H. Steindal, K. O. Sylvester-Hvid, P. R. Taylor, A. M. Teale, E. I. Tellgren, D. P. Tew, A. J. Thorvaldsen, L. Thøgersen, O. Vahtras, M. A. Watson, D. J. D. Wilson, M. Ziolkowski, and H. Ågren, "The Dalton quantum chemistry program system," Wiley Interdiscip. Rev.: Comput. Mol. Sci. **4**, 269–284 (2014).
[21]M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, G. A. Petersson, H. Nakatsuji, X. Li, M. Caricato, A. V. Marenich, J. Bloino, B. G. Janesko, R. Gomperts, B. Mennucci, H. P. Hratchian, J. V. Ortiz, A. F. Izmaylov, J. L. Sonnenberg, D. Williams-Young, F. Ding, F. Lipparini, F. Egidi, J. Goings, B. Peng, A. Petrone, T. Henderson, D. Ranasinghe, V. G. Zakrzewski, J. Gao, N. Rega, G. Zheng, W. Liang, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, K. Throssell, J. A. Montgomery, Jr., J. E. Peralta, F. Ogliaro, M. J. Bearpark, J. J. Heyd, E. N. Brothers, K. N. Kudin, V. N. Staroverov, T. A. Keith, R. Kobayashi, J. Normand, K. Raghavachari, A. P. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, J. M. Millam, M. Klene, C. Adamo, R. Cammi, J. W. Ochterski, R. L. Martin, K. Morokuma, O. Farkas, J. B. Foresman, and D. J. Fox, *Gaussian 16 Revision C.01*, Gaussian Inc., Wallingford CT, 2016.
[22]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. Wloch, 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).
[23]A. D. Bochevarov, E. Harder, T. F. Hughes, J. R. Greenwood, D. A. Braden, D. M. Philipp, D. Rinaldo, M. D. Halls, J. Zhang, and R. A. Friesner, "Jaguar: A high-performance quantum chemistry software program with strengths in life and materials sciences," Int. J. Quantum Chem. **113**, 2110–2142 (2013).
[24]F. Aquilante, J. Autschbach, R. K. Carlson, L. F. Chibotaru, M. G. Delcey, L. De Vico, I. F. Galván, N. Ferré, L. M. Frutos, L. Gagliardi, M. Garavelli, A. Giussani, C. E. Hoyer, G. Li Manni, H. Lischka, D. Ma, P. A. Malmqvist, T. Müller, A. Nenov, M. Olivucci, T. B. Pedersen, D. Peng, F. Plasser, B. Pritchard, M. Reiher, I. Rivalta, I. Schapiro, J. Segarra-Martí, M. Stenrup, D. G. Truhlar, L. Ungur, A. Valentini, S. Vancoillie, V. Veryazov, V. P. Vysotskiy, O. Weingart, F. Zapata, and R. Lindh, "MOLCAS 8: New capabilities for multiconfigurational quantum chemical calculations across the periodic table," J. Comput. Chem. **37**, 506–541 (2015).
[25]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).
[26]E. D. Glendening, C. R. Landis, and F. Weinhold, "*NBO 7.0*: New vistas in localized and delocalized chemical bonding theory," J. Comput. Chem. **40**, 2234–2241 (2019).
[27]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. Frúchtl, 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, A. Vázquez-Mayagoitia, P. Verma, O. Villa, A. Vishnu, K. D. Vogiatzis, D. Wang, J. H. Weare, M. J. Williamson, T. L. Windus, K. Wołiński, 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).

14 January 2025 07:10:56

© Author(s) 2024

[28] F. Neese, F. Wennmohs, U. Becker, and C. Riplinger, "The ORCA quantum chemistry program package," J. Chem. Phys. **152**, 224108 (2020).

[29] D. G. A. Smith, L. A. Burns, A. C. Simmonett, R. M. Parrish, M. C. Schieber, R. Galvelis, P. Kraus, H. Kruse, R. Di 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, I. Schaefer, F. Henry, A. Y. Sokolov, K. Patkowski, I. DePrince, A. Eugene, 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).

[30] E. Epifanovsky, A. T. B. Gilbert, X. Feng, J. Lee, Y. Mao, N. Mardirossian, P. Pokhilko, A. F. White, M. P. Coons, A. L. Dempwolff, Z. Gan, D. Hait, P. R. Horn, L. D. Jacobson, I. Kaliman, J. Kussmann, A. W. Lange, K. U. Lao, D. S. Levine, J. Liu, S. C. McKenzie, A. F. Morrison, K. D. Nanda, F. Plasser, D. R. Rehn, M. L. Vidal, Z.-Q. You, Y. Zhu, B. Alam, B. J. Albrecht, A. Aldossary, E. Alguire, J. H. Andersen, V. Athavale, D. Barton, K. Begam, A. Behn, N. Bellonzi, Y. A. Bernard, E. J. Berquist, H. G. A. Burton, A. Carreras, K. Carter-Fenk, R. Chakraborty, A. D. Chien, K. D. Closser, V. Cofer-Shabica, S. Dasgupta, M. de Wergifosse, J. Deng, M. Diedenhofen, H. Do, S. Ehlert, P.-T. Fang, S. Fatehi, Q. Feng, T. Friedhoff, J. Gayvert, Q. Ge, G. Gidofalvi, M. Goldey, J. Gomes, C. E. González-Espinoza, S. Gulania, A. O. Gunina, M. W. D. Hanson-Heine, P. H. P. Harbach, A. Hauser, M. F. Herbst, M. Hernández Vera, M. Hodecker, Z. C. Holden, S. Houck, X. Huang, K. Hui, B. C. Huynh, M. Ivanov, A. Jász, H. Ji, H. Jiang, B. Kaduk, S. Kähler, K. Khistyaev, J. Kim, G. Kis, P. Klunzinger, Z. Koczor-Benda, J. H. Koh, D. Kosenkov, L. Koulias, T. Kowalczyk, C. M. Krauter, K. Kue, A. Kunitsa, T. Kus, I. Ladjánszki, A. Landau, K. V. Lawler, D. Lefrancois, S. Lehtola, R. R. Li, Y.-P. Li, J. Liang, M. Liebenthal, H.-H. Lin, Y.-S. Lin, F. Liu, K.-Y. Liu, M. Loipersberger, A. Luenser, A. Manjanath, P. Manohar, E. Mansoor, S. F. Manzer, S.-P. Mao, A. V. Marenich, T. Markovich, S. Mason, S. A. Maurer, P. F. McLaughlin, M. F. S. J. Menger, J.-M. Mewes, S. A. Mewes, P. Morgante, J. W. Mullinax, K. J. Oosterbaan, G. Paran, A. C. Paul, S. K. Paul, F. Pavošević, Z. Pei, S. Prager, E. I. Proynov, A. Rák, E. Ramos-Cordoba, B. Rana, A. E. Rask, A. Rettig, R. M. Richard, F. Rob, E. Rossomme, T. Scheele, M. Scheurer, M. Schneider, N. Sergueev, S. M. Sharada, W. Skomorowski, D. W. Small, C. J. Stein, Y.-C. Su, E. J. Sundstrom, Z. Tao, J. Thirman, G. J. Tornai, T. Tsuchimochi, N. M. Tubman, S. P. Veccham, O. Vydrov, J. Wenzel, J. Witte, A. Yamada, K. Yao, S. Yeganeh, S. R. Yost, A. Zech, I. Y. Zhang, X. Zhang, Y. Zhang, D. Zuev, A. Aspuru-Guzik, A. T. Bell, N. A. Besley, K. B. Bravaya, B. R. Brooks, D. Casanova, J.-D. Chai, S. Coriani, C. J. Cramer, G. Cserey, I. DePrince, A. Eugene, J. DiStasio, A. Robert, A. Dreuw, B. D. Dunietz, T. R. Furlani, I. Goddard, A. William, S. Hammes-Schiffer, T. Head-Gordon, W. J. Hehre, C.-P. Hsu, T.-C. Jagau, Y. Jung, A. Klamt, J. Kong, D. S. Lambrecht, W. Liang, N. J. Mayhall, C. W. McCurdy, J. B. Neaton, C. Ochsenfeld, J. A. Parkhill, R. Peverati, V. A. Rassolov, Y. Shao, L. V. Slipchenko, T. Stauch, R. P. Steele, J. E. Subotnik, A. J. W. Thom, A. Tkatchenko, D. G. Truhlar, T. Van Voorhis, T. A. Wesolowski, K. B. Whaley, I. Woodcock, H. Lee, P. M. Zimmerman, S. Faraji, P. M. W. Gill, M. Head-Gordon, J. M. Herbert, and A. I. Krylov, "Software for the frontiers of quantum chemistry: An overview of developments in the Q-Chem 5 package," J. Chem. Phys. **155**, 084801 (2021).

[31] TURBOMOLE V7.2 2017, a development of University of Karlsruhe and Forschungszentrum Karlsruhe GmbH, 1989–2007, TURBOMOLE GmbH, since 2007; available from http://www.turbomole.com (2017).

[32] C. Bannwarth, E. Caldeweyher, S. Ehlert, A. Hansen, P. Pracht, J. Seibert, S. Spicher, and S. Grimme, "Extended tight-binding quantum chemistry methods," WIREs Comput. Mol. Sci. **11**, e1493 (2021).

[33] A. Hjorth Larsen, J. Jørgen 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. Bjerre Jensen, J. Kermode, J. R. Kitchin, E. Leonhard Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J. Bergmann 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).

[34] P. J. A. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski, and M. J. L. de Hoon, "Biopython: Freely available Python tools for computational molecular biology and bioinformatics," Bioinformatics **25**, 1422–1423 (2009).

[35] T. Verstraelen, P. Tecmer, F. Heidar-Zadeh, K. Boguslawski, M. Chan, Y. Zhao, T. D. Kim, S. Vandenbrande, D. Yang, C. E. González-Espinoza, S. Fias, P. A. Limacher, D. Berrocal, A. Malek, and P. W. Ayers, HORTON 2.0.1, http://theochem.github.com/horton/, 2015.

[36] N. M. O'Boyle, M. Banck, C. A. James, C. Morley, T. Vandermeersch, and G. R. Hutchison, "Open babel: An open chemical toolbox," J. Cheminf. **3**, 33 (2011).

[37] R. Muller (2024). "Pyquante2," https://github.com/rpmuller/pyquante2.

[38] Q. Sun, X. Zhang, S. Banerjee, P. Bao, M. Barbry, N. S. Blunt, N. A. Bogdanov, G. H. Booth, J. Chen, Z.-H. Cui, J. J. Eriksen, Y. Gao, S. Guo, J. Hermann, M. R. Hermes, K. Koh, P. Koval, S. Lehtola, Z. Li, J. Liu, N. Mardirossian, J. D. McClain, M. Motta, B. Mussard, H. Q. Pham, A. Pulkin, W. Purwanto, P. J. Robinson, E. Ronca, E. R. Sayfutyarova, M. Scheurer, H. F. Schurkus, J. E. T. Smith, C. Sun, S.-N. Sun, S. Upadhyay, L. K. Wagner, X. Wang, A. White, J. D. Whitfield, M. J. Williamson, S. Wouters, J. Yang, J. M. Yu, T. Zhu, T. C. Berkelbach, S. Sharma, A. Y. Sokolov, and G. K.-L. Chan, "Recent developments in the PySCF program package," J. Chem. Phys. **153**, 024109 (2020).

[39] K. M. Langner, N. M. O'Boyle, and A. L. Tenderholt, Release of cclib version 1.2 (2014).

[40] E. Berquist, A. Dumi, G. Hutchison, K. M. Langner, O. S. Lee, N. O'Boyle, F. S. S. Schneider, A. Tenderholt, and S. Upadhyay, Release of cclib version 1.8 (2023).

[41] See https://conda-forge.org/ for "conda-forge - Community-led recipes, infrastructure, and distributions for conda" (last accessed 15 January 2024).

[42] See https://pypi.org/ for "PyPI - The Python package index" (last accessed 15 January 2024).

[43] T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and S. Futral, "The spack package manager: Bringing order to HPC software chaos," in *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (IEEE, 2015), pp. 1–12.

[44] M. Kowalewski and P. Seeber, Int. J. Quantum. Chem. **122**, e26872 (2022).

[45] B. Ramsundar, P. Eastman, P. Walters, V. Pande, K. Leswing, and Z. Wu, *Deep Learning for the Life Sciences* (O'Reilly Media, 2019).

[46] G. Landrum, P. Tosco, B. Kelley, R. Rodriguez, D. Cosgrove, sriniker, R. Vianello, N. S. gedeck, G. Jones, E. Kawashima, D. Nealschneider, A. Dalke, B. Cole, M. Swain, S. Turk, A. Savelev, A. Vaucher, M. Wójcikowski, I. Take, V. F. Scalfani, R. Walker, K. Ujihara, D. Probst, G. godin, A. Pahl, J. Lehtivarjo, and F. Berenger, Jasondbiggs, and strets123 (2024).rdkit/rdkit: 2024_03_1 (q1 2024) release.

[47] A. T. McNutt, P. Francoeur, R. Aggarwal, T. Masuda, R. Meli, M. Ragoza, J. Sunseri, and D. R. Koes, "GNINA 1.0: Molecular docking with deep learning," J. Cheminf. **13**, 43 (2021).

[48] N. Rego and D. Koes, "3Dmol.js: molecular visualization with WebGL," Bioinformatics **31**(8), 1322–1324 (2015).

[49] 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).

[50] A. W. Appel, *Modern Compiler Implementationin ML*, 1st ed (Cambridge University Press, 1998).

[51] R. A. Frost, R. Hafiz, and P. Callaghan, "Parser combinators for ambiguous left-recursive grammars," in *Practical Aspects of Declarative Languages*, edited by P. Hudak and D. S. Warren (Springer Berlin Heidelberg, Berlin, Heidelberg, 2008), pp. 167–181.

[52] A. V. Aho and J. D. Ullman, *The Theory of Parsing, Translation, and Compiling* (Prentice-Hall, Inc., 1972).

[53] Pint Developers, Pint: Operate and manipulate physical quantities in Python (2012).

[54] I. L. Thomas, "Protonic structure of molecules. I. Ammonia molecules," Phys. Rev. **185**, 90–94 (1969).

[55] T. Ishimoto, M. Tachikawa, and U. Nagashima, "Review of multicomponent molecular orbital method for direct treatment of nuclear quantum effect," Int. J. Quantum Chem. **109**, 2677–2694 (2009).

14 January 2025 07:10:56

© Author(s) 2024

[56]F. Pavošević, T. Culpitt, and S. Hammes-Schiffer, "Multicomponent quantum chemistry: Integrating electronic and nuclear quantum effects via the nuclear–electronic orbital method," Chem. Rev. **120**, 4222–4253 (2020).
[57]F. Giustino, "Electron–phonon interactions from first principles," Rev. Mod. Phys. **89**, 015003 (2017).
[58]J. Flick, M. Ruggenthaler, H. Appel, and A. Rubio, "Atoms and molecules in cavities, from weak to strong coupling in quantum-electrodynamics (QED) chemistry," Proc. Natl. Acad. Sci. U. S. A. **114**, 3026–3034 (2017).
[59]R. F. Ribeiro, L. A. Martínez-Martínez, M. Du, J. Campos-Gonzalez-Angulo, and J. Yuen-Zhou, "Polariton chemistry: Controlling molecular dynamics with optical cavities," Chem. Sci. **9**, 6325–6339 (2018).
[60]J. T. Hugall, A. Singh, and N. F. van Hulst, "Plasmonic cavity coupling," ACS Photonics **5**, 43–53 (2018).
[61]G. te Velde, F. M. Bickelhaupt, E. J. Baerends, C. Fonseca Guerra, S. J. A. van Gisbergen, J. G. Snijders, and T. Ziegler, "Chemistry with ADF," J. Comput. Chem. **22**, 931 (2001); https://www.scm.com/doc/ADF/Required_citations.html#general-references.

14 January 2025 07:10:56

© Author(s) 2024