The Journal of Open Source Software

parsnip: Streamlined Crystallographic Data Parsing for Assembly Science and Engineering

Jenna Bradley [1] **and Sharon C. Glotzer** [1,2]

1 Materials Science and Engineering, University of Michigan, United States of America **2** Department of Chemical Engineering, University of Michigan, United States of America

DOI: 10.21105/joss.10407

Software

- Review
- Repository
- Archive

Editor: Lucy Whalley

Reviewers:

- @ml-evs
- @janash

Submitted: 02 January 2026
Published: 12 August 2026

Summary

`parsnip` provides a lightweight, precise, and domain agnostic interface for parsing data encoded in the Crystallographic Information File (CIF) 1.1 (Brown & McMahon, 2002), 2.0 (Bernstein et al., 2016), and Macromolecular CIF (mmCIF) (Bourne et al., 1997) formats. Designed for programmatic analysis of crystalline systems, `parsnip` offers a scriptable API and a suite of convenient data retrieval methods for automated lookups of structural information. Its minimal dependency set and high-level interface allows for the rapid incorporation of `parsnip` into existing libraries within the molecular simulation ecosystem (Ramasubramani et al., 2020).

`parsnip`'s primary functionality lies in its ability to accurately reconstruct unit cells from simulation data and from experimental data recorded with limited precision, often with only a few decimal places. Through a combination of rational and floating-point arithmetic, we achieve class-leading accuracy when reconstructing large and complex structures. `parsnip`'s detailed processing of structural data yields better alignment with reported space group and point group symmetries than existing tools (Hester, 2006; Larsen et al., 2017; Wojdyr, 2022), providing an ideal foundation for studies centered on material design.

`parsnip` supports a dictionary-like lookup format for both scalar and tabular data, both of which can be expanded with Unix-style wildcards to simplify complex queries. Convenient methods for parsing unit cell parameters, reconstructing particle positions, and identifying site symmetry data are exposed to streamline common workflows in materials data science. `parsnip`'s clear documentation of conventions and units eliminates ambiguities common to interdisciplinary research. `parsnip`'s use of NumPy structured arrays for data storage simplifies integration into Python, C, and FORTRAN libraries, resulting in a stable, scalable dependency for scientific codebases in materials research at the atomic, molecular, and colloidal scales.

Statement of Need

More than thirty-five years of material data is encoded in the CIF file format, with terabytes of elemental and protein structures freely available to researchers (Gražulis et al., 2009; Ong et al., 2015; wwPDB consortium et al., 2019). While early CIF parsers were predominantly written in C and Fortran, the growth of Python opened new opportunities for simple, scriptable access to crystallographic data. **PyCifRW** is one of the earliest such tools, offering a complete and specification compliant parser for the CIF format (Hester, 2006). Soon thereafter, **ASE**, a tool designed to initialize atomistic simulations, added support for CIF files as an alternative to NetCDF inputs for DFT simulations (Bahn & Jacobsen, 2002; Larsen et al., 2017). This marked a transition from pure IO libraries to mixed IO and analysis tools, with both **ASE** and later projects like **pymatgen** including code to characterize and manipulate structures (Ong et al., 2013). One of the most recent CIF libraries, **gemmi**, also follows this trend, although significant amounts of code are committed to quickly and accurately parsing the CIF grammar.

While all of these tools provide excellent interfaces for researchers working with atomic materials, the structure and typing of alternative libraries is ill-suited to interdisciplinary research, where the building blocks of crystal structures include atoms, molecules, macromolecules, and nanoparticles. The explosion of simulation research in superatomic systems has driven a need for scalable, array-formatted crystallographic data that easily translates across simulation frameworks and system length-scales. `parsnip` addresses this need by providing a simple, intuitive, and well-documented software frontend that integrates tightly with existing standards for molecular simulation and analysis (Anderson et al., 2020; Thompson et al., 2022).

State of the Field

`parsnip` supports colloidal and mesoscale materials research in addition to the atomic and protein datasets for which the CIF and mmCIF specifications were originally designed. This is a fundamental change to the scope of the library, and necessitates novel tooling to support the greater diversity of systems. For example, while both `parsnip` and the **gemmi** (Wojdyr, 2022) library support macromolecular data encoded in the mmCIF format, `parsnip` standardizes its API to ensure all inputs are handled in a consistent, programmatic way. In contrast, **gemmi** has separate parsing methods and data structures for each file type, improving performance at the expense of generality.

Existing crystallography libraries like **ASE** (Larsen et al., 2017) and **pymatgen** (Ong et al., 2013) encode atomic information into the data structures and types of parsed information, requiring postprocessing for studies where that data is unnecessary or incorrect. Rather than associating pure crystallographic data with atomic symbols or valence states by default, `parsnip` provides only the information required to reconstruct a particular structure unless otherwise queried. This approach offers benefits for users who do require atomic information, as we are able to provide access to arbitrary data associated with the basis positions, rather than fixed keys as required by other tools.

Software Design

`parsnip`'s design enables studies of crystallinity in colloidal matter, a feature set that is not met by any available tools in the field. By separating units and particle data from pure structural information, we are able to provide a general interface for the study of ordered matter across scales. We also abstract away some portions of the CIF syntax to simplify the API – most notable, we aggregate across data blocks to ensure all relevant information is accessible through a uniform grammar of queries. This choice enables a user interface that more closely resembles other structural data formats like *XYZ*, *MOL*, and *VTP* (*MOL File Format*, 2025; Schroeder et al., 2006).

Our unique approach to the CIF specification extends to the design of our parser as well. While most existing tools in the space use parser generators based on the IUCR's formal grammar, we identified a non-negligible subset of CIF files that break the formal specification but nevertheless contain useful data. To overcome this, `parsnip` does not validate the entire syntax tree of the CIF grammar: rather, we eagerly consume nodes near the leaves of the tree that appear to contain data. Most commonly, this allows `parsnip` to parse data entries with missing or incorrectly escaped delimiters, correctly extracting data that would otherwise be lost. This is a departure from the standard "validating" parser strategy, but it enables fast and robust data extraction without significant increases to code complexity. As detailed in our performance guide, `parsnip` completes a standardized CIF benchmark 3 to 100 times faster than most open-source libraries, though significantly slower than the novel **gemmi** parser (Wojdyr, 2022).

While our parsing technique alone provides significant accuracy benefits, there are still many files that cannot be accurately reconstructed by other tools. Standard, "symmetrized" CIF

data requires the application of symmetry operations to reconstruct a lattice. These operations are the sum of an experimentally-determined value Wyckoff position and a rational translation, with values wrapped into the range $[0, 1)$. Although Wyckoff positions can have arbitrary real values, the data stored in CIF files is necessarily finite. For this reason, the actual set of valid, parsable positions is the group of rational numbers modulo one. Rather than evaluating expressions in floating point arithmetic like other CIF libraries, `parsnip` evaluates unit cell positions in the correct rational form. We then convert back to floating point values for a tolerance-based check to remove duplicate atoms, which catches edge cases in recorded data where values are not rounded consistently (e.g. $(1/3, 2/3) \rightarrow (0.3333, 0.6666)$).

Research Impact Statement

`parsnip` is designed and optimized for integration with larger materials science codes, with a minimal dependency set and pure Python interface. This has facilitated its incorporation into the **freud** analysis library, which uses `parsnip` to build reference structures for high-throughput simulation analysis (Ramasubramani et al., 2020). When studying complex crystals, standard characterization techniques often fail to uniquely resolve a structure of interest. Simplifying access to a massive array of crystal structures stored in databases like the Crystallography Open Database (COD) (Gražulis et al., 2009) enables the construction of reference datasets for both classical and machine-learned characterization techniques.

We performed tests of parsnip and three other CIF parsing libraries against a subset of the Crystallography Open Database (COD), choosing the correct structure to be the most frequently reported value across the four packages. Table 1 shows that `parsnip`'s rational parsing approach was able to correctly extract 96.5% of structures, more than any other library we could find. We note that our results use a single, fixed parsing precision for all 10,094 files. However, as discussed in `parsnip`'s documentation, tailoring the parse precision to match the precision of the data in the file yields even better results.

Table 1: Comparison of unit-cell reconstruction consensus for 10094 CIF files from the COD. "Matches Consensus" indicates the total number of correctly-reconstructed crystals and "Failed to Parse" indicates files that could not be read at all.

Library	Matches Consensus	Mismatches Consensus	Failed to Parse	Percent Matching Consensus
`parsnip`	**9740**	**17**	337	**96.5%**
ASE	9247	37	810	91.6%
pymatgen	9255	35	804	91.7%
gemmi	8277	1817	**0**	82.0%

`parsnip` equals or outperforms the tested suite of libraries in accurately reproducing the space group of crystal data. Table 2 demonstrates this efficacy, using the 9112 CIF files from the previous test that included explicit space group information. Using `spglib==2.7.0` with `symprec=1e-3`, the reconstructions are evaluated against the original file's keys and classified into four categories: "Correct Space Group" (matches the original record); "Symmetry Too High" (often due to incorrect rounding); "Symmetry Too Low" (often due to extraneous sites); and "Undetermined" (no space group identified at the given precision). We note that while `parsnip` does not have the lowest total number of "Symmetry Too High" cases, the additional errors primarily occur where numeric placeholder values like `-1` are present, as we treat such entries as valid data where other tools do not.

Table 2: Comparison of space-group accuracy for 9114 CIF files from the COD with stored space-group data.

Library	Correct Space Group	Symmetry Too High	Symmetry Too Low	Undetermined
parsnip	**8800**	105	**43**	**164**
ASE	8471	106	357	178
pymatgen	8108	98	420	486
gemmi	8359	**82**	46	625

`parsnip` also supports Unix-style wildcard queries, simplifying common lookup patterns like cell parameter extraction and space group identification. Single-character wildcards enable specification-compliant queries into heterogeneous databases of both CIF and mmCIF files, further accelerating programmatic materials exploration. Most importantly, this allows users to extract equivalent data entries whose naming depends on specification. For example, there are many cases where the mmCIF key-naming convention differs from standard CIF files by a single character. The following code block shows two examples where this wildcard syntax simplifies user scripts. Although the **gemmi** library does support a similar style of wildcard through their `gemmi grep` command-line tool, its use is limited to bash scripting, and each wildcard query requires the file to be re-parsed in its entirety (Wojdyr, 2022).

```python
from parsnip import CifFile

crystal = CifFile("my_data.cif")
protein = CifFile("protein.mmcif")

cif_cell_keys = ("_cell_length_a", "_cell_length_b", "_cell_length_c")
mmc_cell_keys = ("_cell.length_a", "_cell.length_b", "_cell.length_c")

# Wildcard to extract all cell lengths
assert all(crystal[cif_cell_keys] == crystal["_cell_length*"])

# Wildcard to extract the same data from CIF and mmCIF
assert all(protein[mmc_cell_keys] == protein["_cell?length*"])
assert all(crystal[mmc_cell_keys] == crystal["_cell?length*"])
```

Acknowledgments

This research was supported by a Vannevar Bush Faculty Fellowship sponsored by the Department of the Navy, Office of Naval Research under ONR award number N00014-22-1-2821.

AI Usage Disclosure

GitHub Copilot's automated PR review was evaluated for use with this project, but was disabled due to the unhelpful nature of the output. No generative AI tools were used in the development of this software, the writing of this manuscript, or the preparation of supporting materials.

References

Anderson, J. A., Glaser, J., & Glotzer, S. C. (2020). HOOMD-blue: A Python package for high-performance molecular dynamics and hard particle Monte Carlo simulations. *Computational Materials Science*, *173*, 109363. https://doi.org/10.1016/j.commatsci.2019.109363

Bahn, S. R., & Jacobsen, K. W. (2002). *An object-oriented scripting interface to a legacy electronic structure code* [Article]. *4*(3), 56–66. https://doi.org/10.1109/5992.998641

Bernstein, H. J., Bollinger, J. C., Brown, I. D., Gražulis, S., Hester, J. R., McMahon, B., Spadaccini, N., Westbrook, J. D., & Westrip, S. P. (2016). Specification of the Crystallographic Information File format, version 2.0. *Journal of Applied Crystallography*, *49*(1), 277–284. https://doi.org/10.1107/S1600576715021871

Bourne, P. E., Berman, H. M., McMahon, B., Watenpaugh, K. D., Westbrook, J. D., & Fitzgerald, P. M. D. (1997). [30] Macromolecular crystallographic information file. In *Methods in Enzymology* (Vol. 277, pp. 571–590). Elsevier. https://doi.org/10.1016/S0076-6879(97)77032-0

Brown, I. D., & McMahon, B. (2002). CIF: the computer language of crystallography. *Acta Crystallographica Section B*, *58*(3 Part 1), 317–324. https://doi.org/10.1107/S0108768102003464

Gražulis, S., Chateigner, D., Downs, R. T., Yokochi, A. F. T., Quirós, M., Lutterotti, L., Manakova, E., Butkus, J., Moeck, P., & Le Bail, A. (2009). Crystallography Open Database – an open-access collection of crystal structures. *Journal of Applied Crystallography*, *42*(4), 726–729. https://doi.org/10.1107/S0021889809016690

Hester, J. R. (2006). A validating CIF parser: *PyCIFRW*. *Journal of Applied Crystallography*, *39*(4), 621–625. https://doi.org/10.1107/S0021889806015627

Larsen, A. H., Mortensen, J. J., Blomqvist, J., Castelli, I. E., Christensen, R., Dułak, M., Friis, J., Groves, M. N., Hammer, B., Hargus, C., Hermes, E. D., Jennings, P. C., Jensen, P. B., Kermode, J., Kitchin, J. R., Kolsbjerg, E. L., Kubal, J., Kaasbjerg, K., Lysgaard, S., … Jacobsen, K. W. (2017). The atomic simulation environment—a python library for working with atoms. *Journal of Physics: Condensed Matter*, *29*(27), 273002. https://doi.org/10.1088/1361-648X/aa680e

MOL file format. (2025). https://doi.org/10.1351/goldbook.MT06966

Ong, S. P., Cholia, S., Jain, A., Brafman, M., Gunter, D., Ceder, G., & Persson, K. A. (2015). The materials application programming interface (API): A simple, flexible and efficient API for materials data based on REpresentational state transfer (REST) principles. *Computational Materials Science*, *97*, 209–215. https://doi.org/10.1016/j.commatsci.2014.10.037

Ong, S. P., Richards, W. D., Jain, A., Hautier, G., Kocher, M., Cholia, S., Gunter, D., Chevrier, V. L., Persson, K. A., & Ceder, G. (2013). Python Materials Genomics (pymatgen): A robust, open-source python library for materials analysis. *Computational Materials Science*, *68*, 314–319. https://doi.org/10.1016/j.commatsci.2012.10.028

Ramasubramani, V., Dice, B. D., Harper, E. S., Spellings, M. P., Anderson, J. A., & Glotzer, S. C. (2020). Freud: A software suite for high throughput analysis of particle simulation data. *Computer Physics Communications*, *254*, 107275. https://doi.org/10.1016/j.cpc.2020.107275

Schroeder, W., Martin, K., & Lorensen, B. (2006). *The visualization toolkit (4th ed.)*. Kitware. ISBN: 978-1-930934-19-1

Thompson, A. P., Aktulga, H. M., Berger, R., Bolintineanu, D. S., Brown, W. M., Crozier, P. S., Veld, P. J. in 't, Kohlmeyer, A., Moore, S. G., Nguyen, T. D., Shan, R., Stevens, M. J., Tranchida, J., Trott, C., & Plimpton, S. J. (2022). LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Comp. Phys. Comm.*, *271*, 108171. https://doi.org/10.1016/j.cpc.2021.108171

Wojdyr, M. (2022). GEMMI: A library for structural biology. *Journal of Open Source Software*, *7*(73), 4200. https://doi.org/10.21105/joss.04200

wwPDB consortium, Burley, S. K., Berman, H. M., Bhikadiya, C., Bi, C., Chen, L., Costanzo, L. D., Christie, C., Duarte, J. M., Dutta, S., Feng, Z., Ghosh, S., Goodsell, D. S., Green, R. K., Guranovic, V., Guzenko, D., Hudson, B. P., Liang, Y., Lowe, R., … Ioannidis, Y. E. (2019). Protein Data Bank: The single global archive for 3D macromolecular structure data. *Nucleic Acids Research*, *47*(D1), D520–D528. https://doi.org/10.1093/nar/gky949