PDBRust: A High-Performance Rust Library for Protein Structure File Parsing and Analysis

Hosein Fooladi
University of Vienna, Austria
hosein.fooladi@univie.ac.at
ORCID: 0000-0002-3124-2761

Abstract

We present PDBRust, an open-source Rust library for parsing and analyzing Protein Data Bank (PDB) and mmCIF structure files. PDBRust provides a unified interface for both file formats with automatic format detection, comprehensive structural analysis capabilities, and seamless integration with the RCSB PDB database. Benchmarks demonstrate 40–260× speedups compared to equivalent Python implementations for common operations. The library is designed for high-throughput structural bioinformatics pipelines and machine learning applications requiring efficient processing of large protein structure datasets. PDBRust is available at https://github.com/HFooladi/pdbrust and through the Rust package registry at https://crates.io/crates/pdbrust.

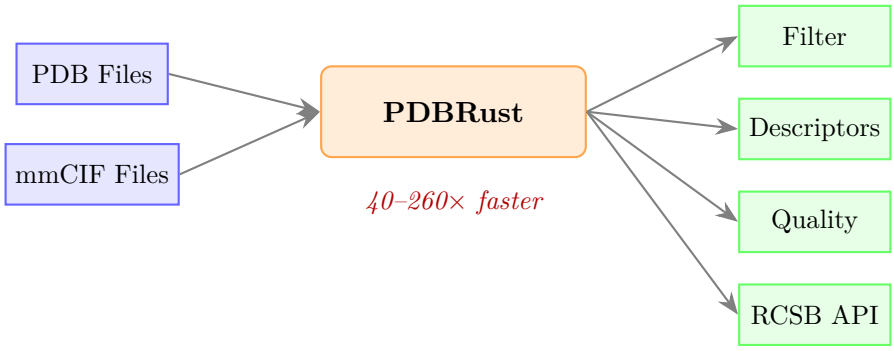

Figure 1: PDBRust architecture: unified parsing of PDB/mmCIF files with modular analysis features.

1 Introduction

Protein structure analysis is fundamental to computational biology, drug discovery, and machine learning applications in structural bioinformatics. The Protein Data Bank (PDB) Berman et al. [2000] contains over 200,000 experimentally determined structures, with data available in both the legacy PDB format and the modern mmCIF format. Processing these structures efficiently is essential for large-scale analyses, yet existing tools often sacrifice performance for ease of use.

Python libraries such as Biopython Cock et al. [2009] and MDAnalysis Michaud-Agrawal et al. [2011] provide comprehensive functionality but face inherent performance limitations due to interpreter overhead and memory management. For applications requiring processing of thousands of structures—such as training machine learning models on structural data or genome-wide structural analysis—these limitations become significant bottlenecks.

We developed PDBRust to address this gap, providing a Rust library that combines the safety and performance characteristics of systems programming with an ergonomic API suitable

for scientific computing. Rust's zero-cost abstractions, memory safety guarantees, and lack of garbage collection overhead make it well-suited for computationally intensive structural biology applications.

2 Features and Implementation

2.1 Core Parsing

PDBRust supports both PDB and mmCIF file formats through a unified `PdbStructure` data model. The parser automatically detects file format based on content and handles:

- ATOM/HETATM coordinate records with full metadata
- SEQRES sequence information
- SSBOND disulfide bond definitions
- CONECT connectivity records
- Multi-model structures (NMR ensembles)
- Alternate conformations (altlocs)

2.2 Modular Feature System

PDBRust employs Rust's feature flag system to provide optional functionality while keeping the core library lightweight. Available modules include:

- **filter**: Structure filtering, chain extraction, ligand removal, and coordinate transformations
- **descriptors**: Structural metrics including radius of gyration, amino acid composition, and geometric properties
- **quality**: Structure quality assessment for analysis readiness
- **rcsb**: Direct integration with RCSB PDB search API and file download

2.3 API Design

The library provides a fluent API for method chaining, enabling concise expression of complex operations:

```
let structure = parse_pdb_file("protein.pdb")?;

// Fluent API for structure manipulation
let cleaned = structure
    .remove_ligands()
    .keep_only_chain("A")
    .keep_only_backbone();

// Compute structural descriptors
let rg = cleaned.radius_of_gyration();
let composition = cleaned.aa_composition();
```

Listing 1: Example: Structure cleaning and analysis

3 Performance

We benchmarked PDBRust against equivalent Python implementations using the `libraryPDB` package on a representative protein structure (PDB: 1A2B, 4,728 atoms). Table 1 summarizes the results.

Table 1: Performance comparison between PDBRust and Python (libraryPDB)

Operation	Python (ms)	Rust (ms)	Speedup
Parse PDB file	15.2	0.36	42×
Remove ligands	8.0	0.03	267×
Get CA coordinates	7.2	0.03	240×
Radius of gyration	2.0	0.05	40×
Max CA distance	70.4	0.27	261×

The substantial speedups (40–260×) arise from several factors: (1) PDBRust parses the structure once and reuses it across operations, whereas Python libraries often re-parse for each call; (2) Rust's zero-cost abstractions eliminate runtime overhead; (3) efficient memory layout improves CPU cache utilization for coordinate-heavy operations.

4 Use Cases

PDBRust is designed for:

- **Machine learning pipelines**: Efficient feature extraction from large structure datasets for training geometric deep learning models
- **High-throughput screening**: Rapid processing of structure libraries for virtual screening campaigns
- **Structural bioinformatics**: Genome-wide analysis of protein structures and structure-function relationships
- **Web services**: Backend processing for structure analysis web applications requiring low latency

5 Availability

PDBRust is open-source software released under the MIT license. The source code is available at https://github.com/HFooladi/pdbrust. The library can be installed via the Rust package manager:

```
cargo add pdbrust --features "filter,descriptors,rcsb"
```

Documentation is available at https://docs.rs/pdbrust.

6 Conclusion

PDBRust provides a high-performance, memory-safe library for protein structure file parsing and analysis. By leveraging Rust's systems programming capabilities, it achieves 40–260× speedups over Python alternatives while maintaining an ergonomic API. The modular design allows users to include only the functionality they need, and the RCSB integration enables seamless access to the Protein Data Bank. PDBRust is suitable for large-scale structural bioinformatics applications where performance is critical.

Acknowledgments

We thank the Rust and structural biology communities for valuable feedback during development.

References

Helen M. Berman, John Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242, 2000. doi: 10.1093/nar/28.1.235.

Peter J. A. Cock, Tiago Antao, Jeffrey T. Chang, Brad A. Chapman, Cymon J. Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, and Michiel J. L. de Hoon. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, 2009. doi: 10.1093/bioinformatics/btp163.

Naveen Michaud-Agrawal, Elizabeth J. Denning, Thomas B. Woolf, and Oliver Beckstein. MDAnalysis: a toolkit for the analysis of molecular dynamics simulations. *Journal of Computational Chemistry*, 32(10):2319–2327, 2011. doi: 10.1002/jcc.21787.