Journal of Cheminformatics

SOFTWARE

Open Access

Helmkit: fast and robust conversion of HELM notation to atomistic representations for large-scale macromolecular informatics

Ramon Adàlia[1,2*], Gemma Sanjuan[1], Tomàs Margalef[1] and Ismael Zamora[2,3]

Abstract

The Hierarchical Editing Language for Macromolecules (HELM) provides a powerful framework for representing complex biomolecules, including peptides, oligonucleotides, and hybrid constructs, but existing tools for converting HELM notations to atomistic models suffer from limitations in speed, scope, and robustness. We introduce helmkit, an open-source Python library that enables direct, high-throughput conversion of HELM strings to RDKit molecular objects. Designed for general macromolecular structures, helmkit supports peptides, nucleic acids, chemical linkers, and hybrids, while natively handling inline monomers, special characters in names, and automatic inference of missing attachment points. Its streamlined architecture, with minimal dependencies and built-in parallelization, achieves processing speeds of up to 5,000 HELM entities per second. Validation on large-scale datasets from PubChem (878,442 entries) and CycPeptMPDB (7,298 entries) demonstrates near-perfect accuracy, with helmkit successfully parsing structures that fail in other libraries. By facilitating efficient, scalable analysis of diverse macromolecules, helmkit advances computational workflows in drug discovery, virtual screening, and biomolecular engineering.

Scientific contribution

We introduce a lightweight Python library that provides direct and accurate conversion from HELM notation to atomistic RDKit representations, addressing a practical limitation of existing cheminformatics tools. Unlike prior approaches, the library supports peptides, RNA, chemical linkers, and hybrid molecules within a single framework, while remaining robust to real-world HELM variants through built-in error correction and R-group inference. Validation on public datasets and high-throughput performance demonstrate that the method is reliable and suitable for large-scale molecular data processing.

Keywords HELM, Python, Macromolecules, RDKit, Cheminformatics, High-throughput

Introduction

Macromolecules such as peptides [5], oligonucleotides [22], and hybrid constructs [23] play a central role in modern biology and drug discovery. These molecules include peptides that act as hormones and signaling agents [3], oligonucleotides that regulate gene expression [1], and conjugates that combine small molecules, linkers, and biopolymers into multifunctional therapeutics [7]. Collectively, they span a continuum between small molecules and large biologics, offering

*Correspondence:
Ramon Adàlia
Ramon.Adalia@autonoma.cat
[1] Computer Architecture and Operating Systems, Universitat Autònoma de Barcelona, Carrer de les Sitges, 08193 Cerdanyola del Vallès, Barcelona, Spain
[2] Lead Molecular Design, S.L., Avinguda Cerdanyola 92-94 1.4, 08173 Sant Cugat del Vallès, Barcelona, Spain
[3] Mass Spec Analytica, Carrer Vallseca 93, 08195 Sant Cugat del Vallès, Barcelona, Spain

© The Author(s) 2026. **Open Access** This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by-nc-nd/4.0/.

tunable specificity, potency, and pharmacokinetic properties [25]. Their expanding use across oncology, metabolic disorders, and infectious diseases highlights the growing need for computational tools capable of handling their structural diversity.

With the increasing sophistication of designed macromolecules, including non-natural residues, cyclic topologies, chemical linkers, and hybrid architectures, manual analysis and representation have become unfeasible. Computational tools are essential for virtual screening, property prediction, structure generation, and synthetic planning. Central to these applications is the ability to accurately represent and manipulate macromolecular structures *in silico* [2]. However, existing cheminformatics tools are typically tailored to either linear biopolymers or small molecules, making them ill-suited for hierarchical structures that combine both [9].

Biopolymers can be represented in a variety of ways. Sequence-based formats such as FASTA [15] encode peptides or nucleic acids as strings of monomer symbols, providing a compact and human-readable representation. However, they cannot describe non-linear architectures, chemical linkers, or modified monomers. Conversely, atom-level formats such as SMILES [26], InChI [6], or MDL [4] describe molecules as arbitrary chemical graphs, allowing full flexibility but losing the hierarchical, monomer-based organization that facilitates sequence-level operations. Tasks such as identifying termini, counting residues, or mapping modifications become nontrivial graph traversal problems.

The Hierarchical Editing Language for Macromolecules (HELM) [28] was developed to bridge this gap. HELM combines the clarity of sequence-based notation with the expressiveness of chemical graph representations. It encodes macromolecules as sets of named monomers connected through explicitly defined attachment points. Each monomer's chemical structure and bonding information are defined in a monomer database, enabling the representation of peptides, RNA, DNA, chemical linkers, and hybrid constructs within a unified framework.

While several open-source tools can parse HELM, they often suffer from limited functionality and poor scalability. For example, the widely used RDKit [10] provides only basic parsing and cannot handle extended monomer definitions or nested polymer structures. The more recent `pyPept` library [14] supports custom monomers but focuses exclusively on peptides, lacks support for inline monomers, and struggles with datasets where monomer names contain special characters, which are issues common in public resources such as PubChem [8] or CycPeptMPDB [11]. Furthermore, its performance constraints, typically processing only a few molecules per second, limit its suitability for high-throughput workflows.

To overcome these limitations, we introduce `helmkit`, a dedicated Python library for converting HELM strings into RDKit molecular objects. Unlike peptide-specific tools, `helmkit` is designed for general hierarchical structures, supporting peptides, nucleic acids, chemical linkers, and hybrid combinations thereof. It natively handles inline monomers, custom monomer definitions, and special character encodings, ensuring compatibility with real-world datasets. The implementation emphasizes efficiency and robustness, with built-in parallelization and optimized data structures enabling throughput approaching 5,000 HELM entities per second on standard hardware. `helmkit` has been validated across large-scale datasets from PubChem and CycPeptMPDB, demonstrating both correctness and scalability for modern macromolecular informatics.

Related work

The development of tools for handling macromolecular structures has evolved alongside the increasing complexity of biotherapeutic design. Several open-source software packages support the parsing, manipulation, and conversion of HELM notation, reflecting its adoption as a standard for representing diverse biomolecules. These tools vary in scope, from general-purpose cheminformatics libraries to specialized parsers, but often exhibit limitations in scalability, flexibility, or integration with popular Python ecosystems such as RDKit. Below, we survey key existing approaches, highlighting their strengths and gaps that `helmkit` addresses.

General cheminformatics libraries with HELM support

Prominent open-source cheminformatics toolkits provide partial support for HELM, primarily focusing on basic parsing and conversion to molecular objects.

The RDKit library itself offers native HELM parsing through the `MolFromHELM` function, which converts HELM strings directly into `Chem.Mol` objects. This functionality is efficient for standard monomers, such as natural amino acids and nucleotides, and integrates seamlessly with RDKit's extensive suite of molecular manipulation tools. However, RDKit's HELM support is constrained to a predefined set of monomers; user-supplied custom monomers are not natively supported. As a result, it is unsuitable for diverse real-world datasets without significant preprocessing.

Other general libraries, such as the Chemistry Development Kit (CDK) [24] in Java, indirectly support HELM through integrations with specialized

toolkits (discussed below). CDK excels in graph-based molecular representations and can generate SMILES or other formats from parsed structures, but it does not natively parse HELM, requiring additional layers for macromolecular handling.

Pistoia alliance HELM toolkit and editors

The Pistoia Alliance, the organization behind the HELM standard, maintains a suite of open-source tools under the PistoiaHELM project on GitHub [17]. These are primarily implemented in Java and serve as reference implementations for HELM parsing and manipulation.

The HELM2 Notation Toolkit [19] is the core reference implementation for HELM version 2.0. It provides robust parsing of HELM strings, creation of HELM notation objects, validation, and basic calculations (e.g., molecular weight). Leveraging CDK for chemistry operations, it supports ambiguous structures and can be configured for external monomer databases via web services. While it enables conversion to atom-level representations such as SMILES through CDK, it does not directly output RDKit-compatible molecules and is not designed for Python workflows. An earlier HELM Notation Toolkit exists for version 1.0, with similar features but without ambiguity support.

Complementing the toolkit are graphical editors: the HELM Web-editor [20], a web-based tool for drawing and editing biomolecules, and the HELM Antibody Editor (HAbE) [18] for specialized antibody notations. These tools facilitate visual structure creation and export to HELM strings but are not optimized for batch processing or programmatic conversion. A.NET implementation of the HELM Toolkit is also available, offering similar parsing capabilities for Windows-centric environments, although it lacks the CDK integration for advanced conversions.

Overall, the Pistoia tools provide comprehensive HELM compliance and are ideal for validation or standalone Java/.NET applications. However, their language-specific nature limits their adoption in Python-dominated cheminformatics pipelines, and their performance in high-throughput scenarios is not emphasized.

Specialized python libraries

In the Python ecosystem, options for HELM handling are more limited, with a focus on domain-specific applications.

The pyPept library [14] is a notable Python tool for peptide structures, supporting HELM parsing and conversion to RDKit molecules. To the best of our knowledge, it is the only Python library capable of handling HELM with custom monomer libraries. It handles custom monomers via SDF files and includes features for 2D/3D conformer generation. However, as detailed in Sect. Differences with pyPept, pyPept is restricted to peptides, employs inefficient intermediate representations (e.g., BILN to sequence), requires multiple dependencies (e.g., pandas, Biopython, igraph), and struggles with monomer names containing special characters. Its performance, at a few molecules per second, hinders large-scale use, and error handling via process termination (`exit()`) is suboptimal for robust applications.

Gaps and opportunities

Existing tools demonstrate HELM's maturity but reveal key shortcomings: general libraries such as RDKit lack extensibility for custom macromolecules, Pistoia tools are not Python-native, and specialized options such as pyPept are narrow in scope and inefficient. None emphasize parallel processing, inline monomer support, or automatic R-group inference for underspecified datasets such as PubChem. `helmkit` fills these voids by offering a lightweight, Python-centric library with direct HELM-to-RDKit conversion, broad macromolecular compatibility, and high-throughput capabilities, making it a valuable addition to the ecosystem.

HELM notation

The Hierarchical Editing Language for Macromolecules (HELM) [28] is a line notation developed to represent complex biomolecular structures in a concise, hierarchical, and computationally accessible format. It overcomes the limitations of sequence-based or atom-only representations by organizing structural information across four levels: complex polymer, simple polymer, monomer, and atom. This hierarchy enables the accurate encoding of large, chemically modified biomolecules, including peptides, oligonucleotides, antibody–drug conjugates, and crosslinked systems, while remaining convertible to both atomic-level formats (e.g., SMILES, InChI) and biological sequences.

A complete HELM notation is structured as a string with sections delimited by dollar signs (`$`):

```
SimplePolymerList$ConnectionList$HydrogenBondList$AttributeList$
```

Only the simple polymer list is mandatory.

Simple polymers

A simple polymer represents a single linear chain of monomers sharing the same polymer type (e.g., `PEPTIDE`, `RNA`, `CHEM`). Each is denoted as:

```
PolymerID{SimplePolymerNotation}
```

where `PolymerID` follows the form `TYPE#` (e.g., `PEPTIDE1`, `RNA1`), uniquely identifying the chain within the notation. The simple polymer notation concatenates monomer IDs from left to right, following these rules:

- Single-letter IDs for natural monomers (e.g., `A`, `G`, `U`) are written directly.
- Multi-letter IDs for modified or non-natural monomers are enclosed in square brackets (e.g., `[dF]`, `[mR]`).
- Branch monomers are enclosed in parentheses and attached to the preceding backbone monomer (e.g., `R(A)` attaches base `A` to ribose `R`).
- Periods (`.`) separate monomers into logical groups.
- Backbone chain extension implicitly connects `R2` of one monomer to `R1` of the next, whereas branch chain extension connects `R3` in the backbone to `R1` in the branch.

Monomers need to be defined externally with a SMILES or Molfile structure, a unique ID within their polymer type, and labeled attachment points (`R1`, `R2`, etc.) with default capping groups.

For example, the peptide sequence:

```
PEPTIDE1{A.R.G.[dF].C.K.[ahA].E.D.A}
```

represents a 10-residue chain (N- to C-terminus) with two modified amino acids (`[dF]`, `[ahA]`).

An RNA strand using the sugar–base–phosphate model:

```
RNA1{R(A)P.[mR](G)[sP].R([5meC])P.[dR](U)}
```

encodes a 4-nucleotide sequence (5' to 3') with a modified sugar (`mR`, `dR`), a phosphate linker (`sP`), and a base (`5meC`).

The `CHEM` polymer type is reserved for nonspecific chemical modifiers (e.g., linkers). Due to the absence of predefined connection rules, each `CHEM` simple polymer contains exactly one monomer, and enclosing brackets can be omitted for multi-letter identifiers.

Complex polymers

Intrapolymer connections enable cyclic structures and branches; interpolymer connections support conjugates and crosslinks. Interconnections between simple polymers are specified in the connection list via the following format:

```
SourceID,TargetID,SrcPos:SrcR-TgtPos:TgtR
```

where the positions are 1-based indices within each simple polymer and where `R1`, `R2`, `R3`, etc., denote attachment points. For example:

```
RNA1,CHEM1,21:R2-1:R1
```

links the `R2` site of the 21st monomer in `RNA1` to `R1` of the single monomer in `CHEM1`.

Hydrogen bonds and annotations

Non-covalent interactions, such as base pairing in double-stranded RNA, are captured in the hydrogen bond list as follows:

```
Polymer1,Polymer2,Pos1:pair-Pos2:pair
```

Note that hydrogen bonds do not define bonds between specific atoms, but rather between two monomers, which is why the word `pair` is used instead of specific attachment points such as those in the connection list.

Finally, polymer-specific metadata are provided in the attribute list:

```
PEPTIDE1{Class:therapeutic}|RNA1{Strand:sense}
```

Implementation

Requirements and installation

The `helmkit` library is designed for compatibility with modern Python environments and depends only on Python version 3.11 or higher, along with the RDKit cheminformatics library. No additional third-party packages are required for core functionality. The package is officially distributed through PyPI and can be installed using `pip install helmkit`. Further instructions, including guidance on the development environment setup and custom monomer library configuration, are provided in the repository's `README.md` file.

helmkit design

The `helmkit` library is engineered with a focus on modularity, efficiency, and extensibility to facilitate seamless integration into cheminformatics pipelines. The primary entry point is the `Molecule` class constructor, which accepts a single HELM string as input. This

class encapsulates all necessary methods for parsing the HELM notation and converting it directly to an RDKit `Chem.Mol` object. Internally, the class maintains state for the RDKit molecule, inter-monomer bonds, individual monomers, and polymer chains.

The parsing process begins by dissecting the HELM string into its constituent sections: simple polymers, connections, hydrogen bonds (if present), and attributes. For each simple polymer, the monomers are parsed sequentially. In peptide chains, connections are formed by linking the R2 attachment point of one monomer to the R1 of the next. For RNA polymers, the logic accommodates branched structures: bases enclosed in parentheses are attached via their R1 to the R3 of the preceding sugar monomer, whereas the backbone continues from the R2 of the sugar to the next monomer (e.g., a phosphate). Unutilized R-groups are capped with appropriate substituents, such as hydrogens or hydroxyls, based on the monomer definitions.

Connections between polymers are applied post-parsing, and the specified attachment points are used to form covalent bonds. Hydrogen bonds in HELM are specified at the monomer level using `pair` annotations, which represent non-covalent interactions such as Watson–Crick base pairing in nucleic acids. Due to potential ambiguities in mapping these to specific atoms (e.g., multiple hydrogen bonds per pair), `helmkit` parses and stores them as metadata attributes on the `Molecule` object but does not integrate them into the covalent RDKit graph, due to their non-covalent nature and the ambiguity in atom-level mapping. Future extensions may include optional atom-level resolution based on standard pairing schemes.

Error handling is robust, with descriptive exceptions raised for invalid HELM syntax, undefined monomers, or incompatible attachments. These exceptions include contextual information such as the problematic HELM substring and the specific parsing stage where the error occurred, facilitating debugging. This design ensures that the library fails gracefully without terminating the parent process, enabling batch processing workflows where individual failures should not halt the entire operation.

Differences with pyPept

The `pyPept` library is a Python tool designed for parsing and manipulating peptide structures, supporting conversion from HELM to RDKit molecules via intermediate representations such as Boehringer Ingelheim Line Notation (BILN) and sequence objects. While `pyPept` is effective for its intended domain, `helmkit` addresses several limitations and design choices that impact both performance and usability. A comprehensive comparison of key differences is presented in Table 1.

The most fundamental architectural difference lies in the conversion pathway. Whereas `pyPept` employs a multi-step pipeline (HELM → BILN → Sequence → RDKit), `helmkit` performs direct conversion from HELM to RDKit, eliminating intermediate representations. This design choice reduces computational overhead, minimizes potential error sources, and simplifies the codebase. The streamlined approach contributes significantly to the performance improvements demonstrated in Sect. Performance, where `helmkit` achieves speedups of over 300× compared with `pyPept`.

In addition to architectural differences, `helmkit` extends support to a broader range of biomolecules.

Table 1 Feature comparison between `pyPept` and `helmkit`

Feature	`pyPept`	`helmkit`
Peptide support	✓	✓
Nucleic acid support	✗	✓
Chemical linker support	✗	✓
Hybrid macromolecules	✗	✓
Inline monomer definitions	✗	✓
Special characters in monomer names	✗	✓
Parallel parsing	✗	✓
Session-wide monomer cache	✗	✓
Single-file monomer library input	✗	✓
Exception-based error handling	✗	✓
Conversion path	HELM → BILN → Sequence → RDKit	HELM → RDKit
Dependencies	`rdkit`, `pandas`, `biopython`, `requests`, `igraph`	`rdkit` only

While `pyPept` is restricted to peptides, `helmkit` natively handles nucleic acids, chemical linkers, and hybrid macromolecules, making it suitable for a wider variety of HELM-encoded structures. This expanded scope is particularly valuable for applications involving RNA therapeutics, peptide–nucleic acid conjugates, and other complex biomolecular systems.

Operational differences also distinguish the two libraries. `helmkit` loads the monomer library once per session and reuses it across multiple parsing operations, whereas `pyPept` reloads the library for every peptide. This design, combined with the simpler dependency footprint (RDKit only, versus five dependencies for `pyPept`), reduces both the initialization overhead and the installation complexity. Additionally, `helmkit` requires only a file path to the monomer library, whereas `pyPept` demands a directory structure organized as a Python module with `__init__.py`.

Robustness is another key differentiator. `helmkit` tolerates monomer names containing special characters such as whitespaces, hyphens, and parentheses, provided that they do not disrupt bracket or parenthesis balancing. In contrast, `pyPept` enforces strict naming conventions and fails on such inputs. This flexibility is crucial for processing real-world datasets such as CycPeptMPDB, where `pyPept` often encounters parsing failures due to non-standard monomer names.

Finally, error handling approaches differ substantially. `helmkit` raises Python exceptions that can be caught and handled programmatically, enabling graceful error recovery and integration into larger workflows. `pyPept`, however, calls `exit()` upon encountering errors, which abruptly terminates the entire process. This design choice makes `pyPept` unsuitable for batch processing scenarios where individual failures should not halt the entire operation.

Custom monomer libraries

To accommodate diverse macromolecular datasets, `helmkit` allows loading custom monomer libraries via the `load_monomer_library` utility function, which accepts a path to an SDF file. This function employs a least-recently-used (LRU) cache to prevent redundant loading of the same library, optimizing memory and time in batch processing scenarios.

The loaded library is structured as a dictionary, with top-level keys representing monomer types (e.g., `aa` for amino acids, `rna` for nucleic acid components, and `chem` for chemical linkers). Each type maps to a sub-dictionary where keys are monomer names and values are dictionaries containing the RDKit molecule, R-group indices, substituents, and attachment points. These details must be annotated in the SDF file using tags compatible with `pyPept`: `symbol` (monomer ID), `m_Rgroups` (substituent specifications), `m_RgroupIdx` (R-group atom indices), and `m_type` (monomer category).

Unlike `pyPept`, `helmkit` infers attachment points automatically from the R-group indices, eliminating the need for explicit definitions. Additionally, the SDF file can reside at any filesystem path, without requiring encapsulation in a Python module.

In-line monomer support

Public repositories such as PubChem frequently embed monomer definitions directly within HELM strings using SMILES notation augmented with R-group placeholders. These inline definitions typically omit unused substituents, including only attachment-relevant R-groups. For instance:

```
PEPTIDE1{[*C(=O)CCC(C(=O)O)N |$_R2;;;;;;;;;$|].(C,[dC])}$$$$V2.0
```

`helmkit` natively parses such inline monomers by extracting the SMILES, identifying R-group labels (e.g., `_R2`), and integrating them into the molecule assembly. This feature ensures compatibility with real-world data sources without requiring external monomer databases.

Automatic R-group inference

In cases where inline or custom monomers lack explicit R-group definitions, which occurs in PubChem for several in-line amino acids, `helmkit` employs heuristic substructure matching to infer attachment points. This is activated only for peptide monomers to maintain specificity.

The matching is performed using SMARTS (SMiles ARbitrary Target Specification) patterns, a cheminformatics language that extends SMILES to describe molecular substructures with precise atom and bond constraints. SMARTS allows specification of atom types, hybridization states, hydrogen counts, and bond orders, enabling robust pattern matching against molecular graphs.

For the N-terminal R1 (amine group), the library searches for patterns such as `[#6][NX3H][#6]` (alpha-amino carbon) or `[NX3H2][#6]` (primary amine). If exactly one match is found, a placeholder atom is appended as the R1 group, with the attachment indexed accordingly.

For the C-terminal R2 (carboxyl group), patterns such as `[CX3H1]=O` (aldehyde) or `CX3[OH]` (carboxylic acid) are matched. A similar placeholder is added if exactly one match exists.

This inference prevents parsing failures and enables the processing of underspecified monomers, with the placeholders later replaced during chain assembly.

Parallel processing

For high-throughput applications, `helmkit` provides the `load_in_parallel` function. This accepts a list of HELM strings and an optional custom monomer library path, returning a list of RDKit molecules. Leveraging Python's multiprocessing module, it distributes parsing across available CPU cores, potentially achieving linear speedups on multicore systems.

Usage examples

This section demonstrates how to use `helmkit` for converting HELM notations into atomistic representations. The examples cover basic usage, custom monomer libraries, parallel processing, and complex real-world scenarios.

Basic HELM to RDKit conversion

The simplest use case for `helmkit` is converting a HELM string into an RDKit molecular object. The following example demonstrates parsing a HELM notation containing two peptide chains connected by a covalent bond:

```
from helmkit import Molecule

helm = "PEPTIDE1{E}|PEPTIDE2{C.G}$PEPTIDE1,PEPTIDE2,1:R3-1:R1$$$"
molecule = Molecule(helm)
rdkit_mol = molecule.mol
```

By default, `helmkit` uses an internal library containing 644 amino-acid monomers and 343 RNA monomers, covering both common and unusual building blocks. The peptide monomers originate from `pypept`, CycPept-MPDB, and a small set manually curated to match data from PubChem. The RNA monomers are based on the Pistoia Alliance HELM set, supplemented with additional entries curated from PubChem. This default library provides immediate, configuration-free support for most standard macromolecules.

Key attributes are available to inspect the resulting structure:

- `molecule.monomers`: A list of dictionaries, each describing a single monomer (chain identifier, name, type, abbreviation, etc.).
- `molecule.monomer_indices`: An array mapping each atom in the molecule to its parent monomer. For example, `[0, 0, 0, 1, 1]` means that the first 3 atoms belong to monomer 0 and the next 2 to monomer 1.
- `molecule.bond_indices`: The list of 0-based indices of all inter-monomer bonds in the atomistic structure.
- `molecule.hydrogen_bonds`: A list of tuples, each describing a single hydrogen bond (source chain, source monomer, target chain, target monomer).

Custom monomer libraries

Custom monomer databases can be loaded in SDF format to override the default set:

```
from helmkit import load_monomer_library

monomer_library = load_monomer_library("/path/to/monomers.sdf")
molecule = Molecule(helm, monomer_library)
```

PEPTIDE1{A.L}|RNA1{p.r(U)}|CHEM1{PAD4}$PEPTIDE1,CHEM1,2:R2-1:R1|RNA1,CHEM1,1:R1-1:R2$$$

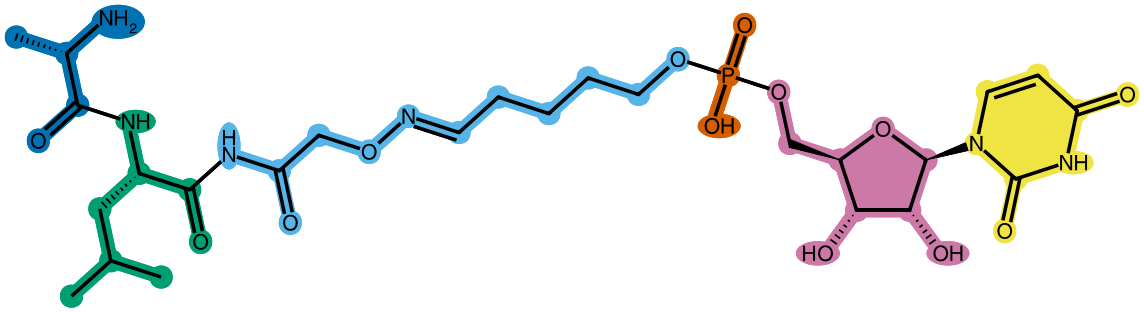


Fig. 1 Peptide-RNA conjugate using the phosphoramidite derivative 4 linker and the corresponding HELM

PEPTIDE964{[medl-].T.[Mono26].[Mono27].L.P.[Me_Tyr(Me)]}$PEPTIDE964,PEPTIDE964,2:R3-7:R2$$$

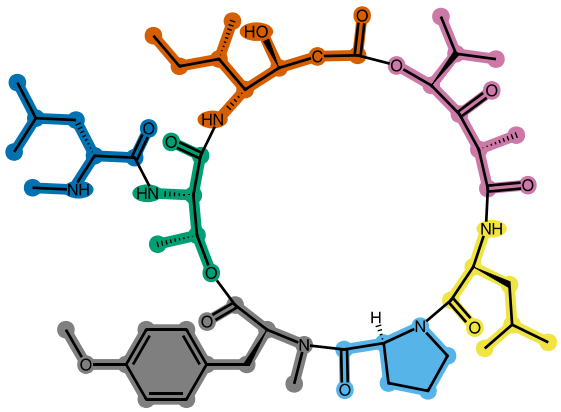


Fig. 2 Lariat peptide from CycPeptMPDB with special characters in the monomer identifiers of the corresponding HELM

PEPTIDE1{[C1CC(N(C1)C(=O)O)C=O].[*C(*)C |$_R1;;_R2;$|].[c1ccc(cc1)CCC(C(=O)O)N]}$$$$

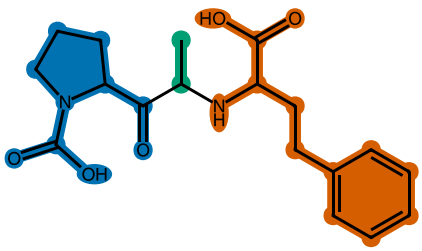


Fig. 3 Example peptide from PubChem, featuring monomers with missing R-groups in their inline SMILES

PEPTIDE1{F.C.Y.[dW].K.T.C.T}|CHEM1{LUDT}$PEPTIDE1,PEPTIDE1,2:R3-7:R3|PEPTIDE1,CHEM1,1:R1-1:R1$$$

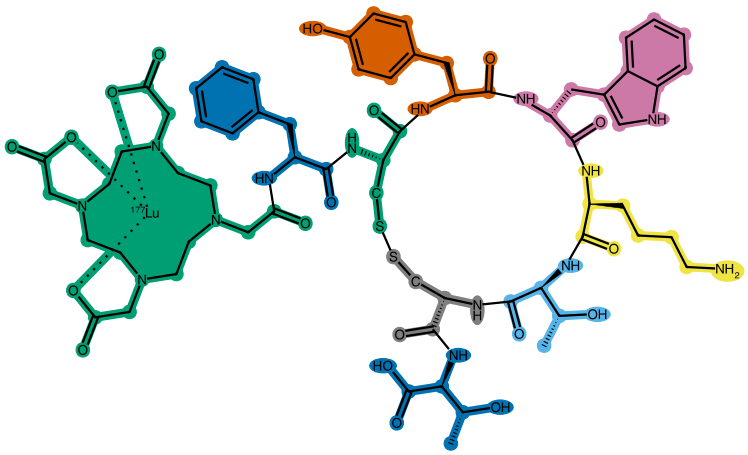


Fig. 4 ^{177}Lu-DOTATATE and the corresponding HELM encoding

PEPTIDE1{[Glp].H.W.S.T.K.L.R.P.G.[am]}|CHEM1{GLUT}|CHEM2{DOXO}$PEPTIDE1,CHEM1,6:R3-1:R2|CHEM1,CHEM2,1:R1-1:R1$$$

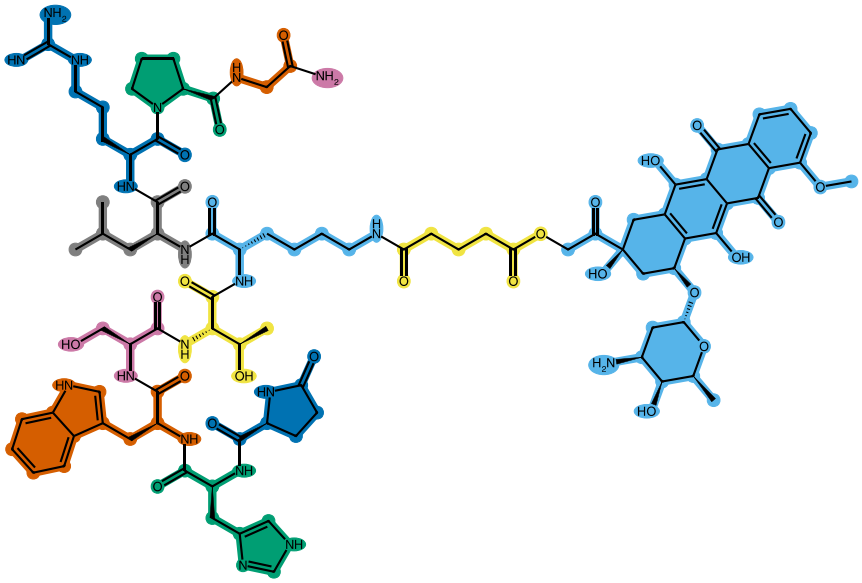


Fig. 5 Zoptarelin doxorubicin and the corresponding HELM encoding

Parallel batch processing

For high-throughput tasks involving large batches of HELM strings, `helmkit` supports parallel parsing using all available CPU cores:

```
from helmkit import load_in_parallel

helms = ["PEPTIDE1{A.R.G}$$$$", "PEPTIDE1{S.G.T}$$$$"]
molecules = load_in_parallel(helms, monomer_library)
```

Examples

The following examples illustrate `helmkit`'s capabilities with complex macromolecular structures. Figure 1 presents a HELM string defining a peptide–RNA conjugate using the PAD4 (phosphoramidite derivative 4) linker [12]. Note that chain integer identifiers only require uniqueness within each chain type.

Figure 2 shows a HELM string for a lariat-shaped peptide containing non-standard amino acids from the CycPeptMPDB database [27]. `helmkit` can process special characters and complex monomer names that often break other HELM toolkits.

`helmkit` also reliably parses HELM notations containing monomers defined by inline SMILES and can infer missing R-groups, which is something essential for poorly specified inputs. Figure 3 illustrates such a scenario with an example from PubChem [13], where the first and third monomers lack explicit R2 and R1, but are correctly interpreted by the library.

Real-world therapeutic conjugates emphasize why robust HELM support matters in practice. Clinical candidates frequently mix canonical residues, heavily modified monomers, and small organic payloads inside a single notation, stressing parser flexibility. Figure 4 shows the approved ^{177}Lu-DOTATATE radioligand [16], where a cyclic octreotide core is connected both internally (disulfide between residues 2 and 7) and externally to a DOTA-lutetium payload. Similarly, Fig. 5 shows zoptarelin doxorubicin [21], a GnRH decapeptide linked through a glutaric spacer to a doxorubicin payload, illustrating two independent `CHEM` polymers attached to a single peptide chain. These examples underline how `helmkit` faithfully resolves complex attachment logic into RDKit graphs that can be directly used for property calculations or visualization.

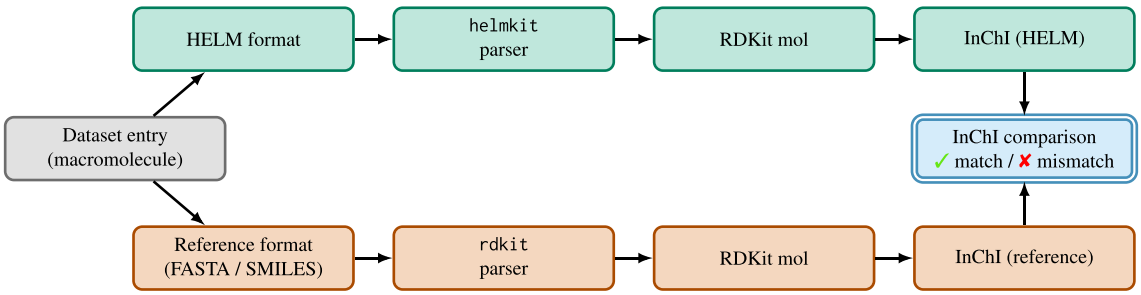


Fig. 6 Validation flow for HELM parsing with `helmkit`

Table 2 Summary of dataset-level success rates for the synthetic and large-scale tests

Synthetic tests		
Dataset	**helmkit**	**pyPept**
Random peptides (1,000)	✓ 1, 000/1, 000 (100%)	✘ 992/1, 000 (99.2%)
Random RNA (1,000)	✓ 1, 000/1, 000 (100%)	✘ 0/1, 000 (0%)
Real-world datasets		
CycPeptMPDB	✓ 7, 298/7, 298 (100%)	✘ 4, 694/7, 298 (64.3%)
PubChem peptides	✓ 864, 102/867, 922 (99.56%)	✘ 90, 095/867, 922 (10.38%)
PubChem RNA	✓ 10, 355/10, 520 (98.43%)	✘ 0/10, 520 (0%)

Results

Testing

The validation of `helmkit` employed a multi-tiered approach combining systematic random generation and large-scale dataset validation to ensure both correctness and robustness across diverse macromolecular structures. The testing strategy addresses the inherent complexity of HELM parsing, where errors in monomer connection, R-group handling, or attachment point resolution can propagate to produce incorrect molecular graphs that may still be chemically valid but structurally distinct from the intended representation.

Integration tests were implemented using Python's `pytest` framework, exercising core parsing functionality, monomer library loading, R-group inference, and connection handling. To validate end-to-end correctness, tests compared HELM-derived RDKit molecules against equivalent structures parsed from alternative representations using well-established tools. The International Chemical Identifier (InChI) served as the primary validation metric, providing a canonical representation that is sensitive to structural differences while remaining invariant to the representation format. For each test case, InChI strings computed from HELM-parsed molecules were compared against reference InChI values derived from equivalent input formats. Figure 6 illustrates this process.

Systematic validation employed randomly generated macromolecules to exercise the parser across a broad parameter space. Monomers were sampled from standard biological sets: the 20 canonical amino acids for peptides, and the four standard RNA bases (A, C, G, U) along with ribose sugar and phosphate linker for nucleic acids. Peptides were constructed as linear chains spanning 2–50 residues, while RNA structures tested both common monomer orderings: `[sugar](base)[linker]` and `[linker][sugar](base)` formats, with chain lengths ranging from 5–40 nucleotides. For each set, 1,000 random HELM strings were generated, parsed, converted to an RDKit molecule, and cross-validated by parsing the equivalent sequence representation (FASTA) using RDKit's native parsers and comparing the resulting InChI identifiers.

Large-scale validation leveraged real-world datasets to assess the performance on structures encountered in production environments. The PubChem database provided 878,442 entries containing HELM notations for peptides and RNA, including 867,922 peptide entries as of August 11, 2025 and 10,520 RNA records. Prior to validation, double-bond stereochemistry and isotopic specifications were intentionally stripped, as these features cannot be consistently or unambiguously represented within the constraints of HELM. The CycPeptMPDB dataset (version 1.2) comprised 7,298 cyclic and lariat peptides, including complex topologies and non-standard modifications that challenge parsing robustness. For each entry with available monomer definitions, the HELM string was parsed, converted to an RDKit molecule, and the computed InChI was compared against reference values. Discrepancies underwent manual inspection to distinguish library errors from database inconsistencies, leading to iterative refinements in R-group inference algorithms and connection-handling logic.

Table 2 summarizes the full set of validation results, which demonstrate that `helmkit` performs accurately across all datasets. For PubChem, `pyPept` was evaluated only on the peptide subset because of its limited domain support. Among the 867,922 peptide HELM entries, 90,095 ($\approx$ 10.38%) were successfully parsed, with most errors stemming from its inability to handle inline monomers. In contrast, `helmkit` achieved over 99% concordance across the full PubChem HELM dataset. All RNA-related failures were manually inspected, and every case was traced back to incorrect structures embedded in the in-line monomers, with the vast majority caused by a missing oxygen atom in diphosphate or triphosphate groups. Peptide discrepancies were also examined on a sampled basis, and all checked cases were confirmed to have originated from database errors, although the full set was too large to review exhaustively.

Table 3 Performance comparison on the CycPeptMPDB dataset

Tool	Total time (s)	s/peptide	Peptides/s	Speedup vs `pyPept`
`pyPept`	676.65	0.14400	6.94	1.0×
`helmkit` (Forced DB reload)	252.67	0.05383	18.58	2.7×
`helmkit`	2.07	0.00044	2267.63	327×
`helmkit` (parallel loading)	0.95	0.00020	4926.11	712×

Performance

Performance benchmarks were conducted using the CycPeptMPDB dataset, which comprises 7,298 cyclic peptides in HELM format. To enable the comparison, entries with monomer names incompatible with `pyPept` (e.g., containing whitespaces, hyphens, or parentheses) were excluded, yielding 4,694 peptides for benchmarking. We measured the total and average parsing time for each tool (Table 3):

`helmkit` outperforms `pyPept` by approximately 327× when the monomer library is loaded once per session. When forced to reload the library for every peptide, `helmkit` is still approximately 3× faster. If we use parallelized loading in `helmkit`, we achieve an additional 2× speedup, resulting in a total speedup of approximately 712× over `pyPept`.

Benchmarks were run on an Intel Core i7-4790 (4 cores, 8 threads, 3.6 GHz) with 31.1 GiB RAM and SSD storage, using Python 3.12.10 on Arch Linux (kernel 6.15.7). Key package versions:

- `polars 1.31.0` (CSV parsing and dataframe processing)
- `rdkit 2025.3.3` (target output format and structure processing)
- `pypept 1.0.0` (commit `ade9f5840691ad1f8fa22d13939a665c25175d5a`)
- `helmkit 0.7.0` (local development version)

Conclusion

We have introduced `helmkit`, a Python library that addresses fundamental limitations in existing HELM parsing tools through architectural innovation and comprehensive biomolecule support. The library's direct HELM-to-RDKit conversion pathway, which eliminates intermediate representations used by tools such as `pyPept`, achieves speedups exceeding 300× in standard configurations and over 700× with parallelization, enabling a throughput of nearly 5000 HELM entities per second on commodity hardware.

In addition to performance, `helmkit` extends functionality to nucleic acids, chemical linkers, and hybrid macromolecules, addressing a critical gap where existing tools are restricted to peptides. The implementation's robustness is demonstrated through successful processing of real-world datasets that challenge other parsers: `helmkit` achieves 100% concordance on CycPeptMPDB and $>99\%$ on PubChem's HELM entries, handling monomer names with special characters, inline SMILES definitions, and missing R-groups that cause failures in alternative tools.

Key design decisions contribute to both performance and usability. The single-dependency architecture (RDKit only) simplifies deployment, whereas session-level monomer library caching eliminates redundant I/O operations. Programmatic error handling via exceptions enables batch processing workflows that are impractical with tools that terminate the process on errors. These features, combined with support for arbitrary monomer naming conventions, make `helmkit` suitable for production cheminformatics pipelines that process diverse macromolecular datasets.

Validation across thousands of structures from public repositories confirms both correctness and scalability. The library's ability to process datasets like CycPeptMPDB that are partially incompatible with existing tools demonstrates its practical value for modern macromolecular informatics. As the field increasingly relies on hierarchical representations for complex therapeutics, `helmkit` provides a foundation for high-throughput analysis of peptides, oligonucleotides, and hybrid constructs within unified computational workflows.

Potential extensions to `helmkit` include atom-level resolution of hydrogen bonds based on standard pairing schemes, which would enable explicit representation of non-covalent interactions in the molecular graph. Support for more HELM notation features, integration with conformer generation tools, and export formats that preserve hierarchical information represent additional directions for expansion. The development of formalized benchmark suites and enhanced monomer curator tooling could facilitate broader community adoption and contributions to monomer libraries.

Acknowledgements

The authors acknowledge support from the PubChem team at NCBI, who assisted in compiling HELM and related biologic notation data used for testing in this work.

Author contributions

RA conceptualized the project, designed and implemented the helmkit library, conducted validation and performance testing, and wrote the original draft of the manuscript. GS and TM provided supervision, resources, and critical revisions of the manuscript. IZ contributed to project conceptualization, provided supervision, and reviewed the manuscript.

Funding

This research was conducted with the support of Pla de Doctorats Industrials del Departament de Recerca i Universitats de la Generalitat de Catalunya (grant number 2023-DI-00006).

Availability of data and materials

- Project name: helmkit
- Project home page: https://github.com/adaliaramon/helmkit
- Archived version: https://doi.org/10.5281/zenodo.17909787
- Operating system(s): Platform independent
- Programming language: Python
- Other requirements: Python 3.11 or higher
- License: MIT
- Any restrictions to use by non-academics: None

Declarations

Conflict of interest

The authors declare no conflict of interest.

Received: 12 December 2025 Accepted: 3 May 2026
Published online: 29 May 2026

References

1. Bennett CF, Swayze EE (2010) RNA targeting therapeutics: molecular mechanisms of antisense oligonucleotides as a therapeutic platform. Annu Rev Pharmacol Toxicol 50(1):259–293. https://doi.org/10.1146/annurev.pharmtox.010909.105654
2. Chen WL, Leland BA, Durant JL et al (2011) Self-contained sequence representation: bridging the gap between bioinformatics and cheminformatics. J Chem Inf Model 51(9):2186–2208. https://doi.org/10.1021/ci2001988
3. Craik DJ, Fairlie DP, Liras S et al (2012) The future of peptide?based drugs. Chem Biol Drug Design 81(1):136–147. https://doi.org/10.1111/cbdd.12055
4. Dalby A, Nourse JG, Hounshell WD et al (1992) Description of several chemical structure file formats used by computer programs developed at Molecular Design Limited. J Chem Inf Comput Sci 32(3):244–255. https://doi.org/10.1021/ci00007a012
5. Fosgerau K, Hoffmann T (2015) Peptide therapeutics: current status and future directions. Drug Discovery Today 20(1):122–128. https://doi.org/10.1016/j.drudis.2014.10.003
6. Heller SR, McNaught A, Pletnev I et al (2015) InChI, the IUPAC international chemical identifier. J Cheminform. https://doi.org/10.1186/s13321-015-0068-4
7. Kiessling LL, Gestwicki JE, Strong LE (2006) Synthetic multivalent ligands as probes of signal transduction. Angew Chem Int Ed 45(15):2348–2368. https://doi.org/10.1002/anie.200502794
8. Kim S, Chen J, Cheng T et al (2024) PubChem 2025 update. Nucleic Acids Res 53(D1):D1516–D1525. https://doi.org/10.1093/nar/gkae1059
9. Kunz RK, Rojnuckarin A, Schmidt CM et al (2023) Development of human-machine language interfaces for the visual analysis of complex biologics and RNA modalities and associated experimental data. AAPS Open. https://doi.org/10.1186/s41120-023-00073-w
10. Landrum G, Tosco P, Kelley B, et al. RDKit: Open-source cheminformatics [2025_03_3 (Q1 2025) release]. 2025. https://doi.org/10.5281/ZENODO.15605628,
11. Li J, Yanagisawa K, Sugita M et al (2023) CycPeptMPDB: a comprehensive database of membrane permeability of cyclic peptides. J Chem Inf Model 63(7):2240–2250. https://doi.org/10.1021/acs.jcim.2c01573
12. Lu K, Duan QP, Ma L et al (2009) Chemical strategies for the synthesis of peptide-oligonucleotide conjugates. Bioconjug Chem 21(2):187–202. https://doi.org/10.1021/bc900158s
13. National Center for Biotechnology Information. PubChem compound summary for CID 55287, 2-[2-[(1-carboxy-3-phenylpropyl)amino]propanoyl]pyrrolidine-1-carboxylic acid. 2025. https://pubchem.ncbi.nlm.nih.gov/compound/55287, retrieved November 12, 2025
14. Ochoa R, Brown JB, Fox T (2023) pyPept: a python library to generate atomistic 2D and 3D representations of peptides. J Cheminform. https://doi.org/10.1186/s13321-023-00748-2
15. Pearson WR, Lipman DJ (1988) Improved tools for biological sequence comparison. Proc Natl Acad Sci 85(8):2444–2448. https://doi.org/10.1073/pnas.85.8.2444
16. Pinato DJ, Black JRM, Ramaswami R et al (2016) Peptide receptor radionuclide therapy for metastatic paragangliomas. Med Oncol. https://doi.org/10.1007/s12032-016-0737-9
17. Pistoia Alliance. Pistoia Alliance HELM project. 2016. https://github.com/PistoiaHELM, accessed: 2025-12-09
18. Pistoia Alliance. HELM antibody editor (HAbE). 2018a. https://pistoiaalliance.atlassian.net/wiki/spaces/HELM/pages/2683306085/HELM+Antibody+Editor+HAbE, accessed: 2025-12-09
19. Pistoia Alliance. HELM Toolkit. 2018b. https://pistoiaalliance.atlassian.net/wiki/spaces/PUB/pages/14286877/HELM+Toolkit, accessed: 2025-12-09
20. Pistoia Alliance. HELM web-editor. 2018c. https://pistoiaalliance.atlassian.net/wiki/spaces/PUB/pages/35028994/HELM+Web-editor, accessed: 2025-12-09
21. Rékási Z, Szöke B, Nagy A et al (1993) Effect of luteinizing hormone-releasing hormone analogs containing cytotoxic radicals on the function of rat pituitary cells: tests in a long term superfusion system. Endocrinology 132(5):1991–2000. https://doi.org/10.1210/endo.132.5.8477650
22. Sharma VK, Rungta P, Prasad AK (2014) Nucleic acid therapeutics: basic concepts and recent developments. RSC Adv 4(32):16618. https://doi.org/10.1039/c3ra47841f
23. Srinivasarao M, Galliford CV, Low PS (2015) Principles in the design of ligand-targeted cancer therapeutics and imaging agents. Nat Rev Drug Discovery 14(3):203–219. https://doi.org/10.1038/nrd4519
24. Steinbeck C, Han Y, Kuhn S et al (2003) The chemistry development kit (CDK): An open-source Java library for chemo- and bioinformatics. J Chem Inf Comput Sci 43(2):493–500. https://doi.org/10.1021/ci025584y
25. Valeur E, Guéret SM, Adihou H et al (2017) New modalities for challenging targets in drug discovery. Angew Chem Int Ed 56(35):10294–10323. https://doi.org/10.1002/anie.201611914
26. Weininger D (1988) SMILES, a chemical language and information system. 1. introduction to methodology and encoding rules. J Chem Inf Comput Sci 28(1):31–36. https://doi.org/10.1021/ci00057a005
27. Yutaka Akiyama's Group. CycPeptMPDB ID: 964. 2023.
28. Zhang T, Li H, Xi H et al (2012) HELM: a hierarchical notation language for complex biomolecule structure representation. J Chem Inf Model 52(10):2796–2806. https://doi.org/10.1021/ci3001925

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.