

Peptacular: A ProForma 2.1 compliant Python package for amino acid sequence analysis

Patrick T. Garrett¹, Titus Jung¹, and John R. Yates III¹

¹ The Scripps Research Institute, United States ¶ Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#)
- [Repository](#)
- [Archive](#)

Editor: [Open Journals](#)

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

Summary

Mass spectrometry-based proteomics depends on computational methods to identify and characterize (amino acid) AA sequences. These sequences, ranging from short peptides to complete proteins, exhibit substantial chemical complexity due to PTMs, variable charge states, neutral losses, and isotopic patterns (Angel et al., 2012; Smith & Kelleher, 2013). **Peptacular** is a Python library designed to handle this complexity. The library provides functionality for modifying, calculating mass, m/z, isotopic distributions, physicochemical properties, enzymatic digestion, and fragmentation of AA sequences. Built around the standardized ProForma 2.1 notation, it supports all non-cross-linked ProForma features.

Statement of Need

Historically, the proteomics field has lacked standardization for representing AA sequences. Individual software tools have implemented proprietary notations, which has created barriers to data integration and reanalysis across platforms. ProForma notation (LeDuc et al., 2018) was developed to address this challenge by providing a unified representation system. However, adoption has remained limited, partly due to insufficient support in widely-used computational tools and libraries. **Peptacular** was specifically designed to accelerate ProForma adoption by offering a comprehensive and accessible API with clear documentation.

Additionally, with the continued advance of mass spectrometers, modern proteomics experiments routinely identify tens of thousands of AA sequences. These results are typically exported as tabular files and are frequently processed using Python, particularly with data manipulation libraries such as pandas (Team, 2025) and polars (Vink et al., 2025). To support this workflow, **Peptacular**'s functional API allows you to directly manipulate these tabular data structures and automatically parallelizes operations, making large-scale sequence analysis both fast and straightforward (See example 2).

State of the Field

Several Python packages provide AA sequence analysis capabilities, though each has limitations as a general-purpose library for ProForma notation. **Pyteomics** (Goloborodko et al., 2013) recently added partial ProForma support while maintaining its legacy notation system, requiring format conversions that are not universally supported. **BioPython** (Cock et al., 2009) offers limited support for AA sequences and has no ProForma compatibility; its broad scope includes DNA and RNA sequences. **RustyMS** (Schulte et al., n.d.) delivers comprehensive ProForma parsing through its Python bindings but focuses primarily on parsing and offers limited prebuilt functionality for working with sequences. **PyOpenMS** (Röst et al., 2013) provides an extensive mass spectrometry toolkit but lacks ProForma compatibility.

Peptacular was designed to address these limitations as a general-purpose, ProForma 2.1-compliant AA sequence library. Developing Peptacular from scratch rather than extending existing tools offered three key advantages. First, ProForma notation serves as the foundation rather than a retrofitted addition, enabling complete feature coverage and a cleaner API. Second, the library targets AA sequence analysis specifically, avoiding the complexity that arises from supporting a broader scope. Third, the pure Python implementation ensures compatibility with modern Python versions, including free-threaded builds with the Global Interpreter Lock (GIL) disabled, which will become increasingly relevant as Python's parallelization capabilities continue to improve.

Software Design

Peptacular provides two primary APIs: a functional API and an object-oriented API. The object-oriented API employs a factory pattern to modify Proforma Annotation objects, enabling precise control over annotations. The functional API provides functions that operate directly on serialized sequences and annotations, with automatic parallelization for batch processing.

The built-in parallelization supports three execution backends: sequential, threaded, and process-based (default). Sequential execution provides single-threaded processing for small batches where parallelization overhead is detrimental. Thread-based parallelization currently offers limited benefits due to the GIL but will improve as free-threaded Python builds become standard (PEP 703). Process-based parallelization is the default and most widely supported method. Process and thread spawning mechanisms (fork, spawn, forkserver) are globally configurable, and worker processes are cached to eliminate startup overhead.

Three performance optimizations enable efficient large-scale processing. First, lazy evaluation keeps modifications in serialized form until required. Second, aggressive caching exploits the fact that proteomics datasets typically contain a small number of repeated modifications. Third, the specific objects used to store modifications within the annotations objects are only initialized when needed, reducing memory footprint and accelerating object creation.

Modification reference data, including masses, compositions, and identifiers from Unimod (Creasy & Cottrell, 2004), PSI-MOD (Hupo-Psi, n.d.-a), RESID (RESID Database [PIR - Protein Information Resource], n.d.), XLMOD (Hupo-Psi, n.d.-b), and GNOme (Glycan-Glycan-Data, n.d.), are provided by the companion package Tacular (P. Garrett, 2026). This package embeds the data directly within itself as Python modules rather than storing them as external files. Only valid modifications are included in the embedded data; a modification is considered valid if it possesses at least one of the following properties: average mass, monoisotopic mass, or chemical formula. This design eliminates file I/O overhead during the parsing of supported ontologies.

The package includes full type annotations with a py.typed marker, which enables static type checking and provides IDE autocomplete, inline documentation, and compile-time error detection. Test coverage exceeds 70%, with continuous integration implemented through GitHub Actions. The only dependency is the companion package: tacular.

Research impact statement

Since its initial release, Peptacular has demonstrated measurable adoption. The package has accumulated over 33k downloads from PyPI (as of 07 February 2026), with sustained weekly download rates exceeding 200 installations. It has been listed as one of three Python packages to support ProForma notation by the PSI group. Additionally, Peptacular was used to generate figures within a textbook chapter (P. T. Garrett et al., 2025).

Example Usage

Object-Based API

```
import peptacular as pt

# Parse a sequence into a ProFormaAnnotation
peptide: pt.ProFormaAnnotation = pt.parse("PEM[Oxidation]TIDE")

# Calculate mass and m/z
mass: float = peptide.mass() # 849.342
mz: float = peptide.mz(charge=2) # 425.678

# Factory pattern
print(peptide.set_charge(2).set_peptide_name("Peptacular").serialize())
# (>Peptacular)PEM[Oxidation]TIDE/2
```

Functional-Based API

```
import peptacular as pt

peptides = ['[Acetyl]-PEPTIDES', '<C13>ARE', 'SICK/2']

# Calculate mass and m/z for all peptides
masses: list[float] = pt.mass(peptides) # [928.4026, 374.1914, 451.2454]
mzs: list[float] = pt.mz(peptides, charge=2) # [465.2086, 188.103, 225.6227]
```

Pandas-Functional API

```
import peptacular as pt
import pandas as pd

df = pd.DataFrame(
    {
        "seq": ["PEM[Oxidation]TIDE", "ACDEFGHIK", "M[Phospho]NOPQR"],
    }
)

df["mass"] = df["seq"].apply(pt.mass)
```

Figures

Table 1: Proforma 2.1 Compliance

?	Feature	Example	§ [Support]
Y	Amino acids (+UO)	AAHCFKUOT	6.1 [B]
Y	Unimod names	PEM[Oxidation]AT	6.2.1 [B]
Y	PSI-MOD names	PEM[monohydroxylated residue]AT	6.2.1 [B]
Y	Unimod numbers	PEM[UNIMOD:35]AT	6.2.2 [B]
Y	PSI-MOD numbers	PEM[MOD:00425]AT	6.2.2 [B]
Y	Delta masses	PEM[+15.995]AT	6.2.3 [B]
Y	N-terminal modifications	[Carbamyl]-QPEPTIDE	6.3 [B]
Y	C-terminal modifications	PEPTIDEG-[Methyl]	6.3 [B]
Y	Labile modifications	{Glycan:Hex}EM[U:Oxidation]EV	6.4 [B]

?	Feature	Example	§ [Support]
Y	Multiple modifications	MPGNW[Oxidation][Carboxymethyl]PESQE	6.5 [B]
Y	Information tag	ELV[INFO:AnyString]IS	6.6 [B]
Y	Ambiguous amino acids	BZJX	7.1 [2]
Y	Prefixed delta masses	PEM[U:+15.995]AT	7.2 [2]
Y	Mass gap	PEX[+147.035]AT	7.3 [2]
Y	Formulas	PEM[Formula:0]AT, PEM[Formula:[1701]]AT	7.4 [2]
Y	Mass with interpretation	PEM[+15.995\ Oxidation]AT	7.5 [2]
Y	Unknown mod position	[Oxidation]?PEMAT	7.6.1 [2]
Y	Set of positions	PEP[Oxidation#1]M[#1]AT	7.6.2 [2]
Y	Range of positions	PRT(ESFRMS)[+19.0523]ISK	7.6.3 [2]
Y	Position scores	PEP[Oxidation#1(0.95)]M[#1(0.05)]AT	7.6.4 [2]
Y	Range position scores	(PEP)[Oxidation#1(0.95)]M[#1(0.05)]AT	7.6.5 [2]
Y	Amino acid ambiguity	(?VCH)AT	7.7 [2]
Y	Modification prefixes	PEPM[U:Oxidation]AS[M:0-phospho-L-serine]	7.8 [2]
Y	RESID modifications	EM[R:L-methionine sulfone]EM[RESID:AA0581]	8.1 [T]
Y	Names	(>Heavy chain)EVQLVESG	8.2 [T]
Y	XL-MOD modifications	EVTK[X:Aryl azide]LEK[XLMOD:00114]SEFD	9.1 [X]
N	Cross-linkers (intrachain)	EVTK[X:Aryl azide#XL1]LEK[#XL1]SEFD	9.2.1 [X]
N	Cross-linkers (interchain)	EVTK[X:Aryl azide#XL1]L//EK[#XL1]SEFD	9.2.2 [X]
N	Branches	ED[MOD:00093#BRANCH]//D[#BRANCH]ATR	9.3 [X]
Y	GNO modifications	NEEYN[GNO:G59626AS]K	10.1 [G]
Y	Glycan compositions	NEEYN[Glycan:Hex5HexNAc4NeuAc1]K	10.2 [G]
Y	Charged formulas	SEQUEN[Formula:Zn1:z+2]CE	11.1 [A]
Y	Controlling placement	PTI(MERMERME)[+32\ Position:E]PTIDE	11.2 [A]
Y	Global isotope	<13C>CARBON	11.3.1 [A]
Y	Fixed modifications	<[Oxidation]@M>ATPEMILTCMGCLK	11.3.2 [A]
Y	Chimeric spectra	NEEYN+SEQUEN	11.4 [A]
Y	Charges	SEQUEN/2, SEQUEN/[Na:z+1,H:z+1]	11.5 [A]
Y	Ion notation	SEQUEN-[b-type-ion]	11.6 [A]

Table 1 presents the level of ProForma support implemented in Peptacular. The package currently supports all ProForma 2.1 features for linear peptides. Cross-linked peptides (both inter- and intrachain) and branched structures are not currently supported. Ion notation is also not supported at the sequence level; however, the package provides extensive fragmentation support through either API. Support levels are designated as follows: [B] - Base ProForma support, [2] - ProForma 2, [T] - Top down, [X] - Cross linking, [G] - Glycan, [A] - Advanced.

AI usage disclosure

Generative AI models were employed to support the development of this software package. Specifically, Claude Code, Cursor, and GitHub Copilot were utilized for code generation, test development, debugging assistance, and documentation preparation. Additionally, Type.ai was used to assist in manuscript preparation. All AI-generated content was subsequently reviewed and verified for accuracy.

Availability

Peptacular is distributed through PyPI (<https://pypi.org/project/peptacular/>) and available as open-source software on GitHub (<https://github.com/tacular-omics/peptacular>).

This work was supported by the National Institutes of Health under grants R01 AG077046 (Analysis of protein interactions in neurodegenerative disease), R01 MH132570 (Brain-wide mapping of neuronal inhibition by novel inverse activity markers), R01 MH100175 (Proteogenetics of Autism Spectrum Disorders), R01 HL165168 (The CFTR Interactome), and U01 AG088679 (Understanding Gene-Environment Interactions in Brain Aging and Alzheimer's Disease (AD) and AD-Related Dementias (ADRD)).

Röst, H. L., Schmitt, U., Aebersold, R., & Malmström, L. (2013). pyOpenMS: A Python-based interface to the OpenMS mass-spectrometry algorithm library. *PROTEOMICS*, 14(1), 74–77. <https://doi.org/10.1002/pmic.201300246>

- 148 Schulte, D., Gabriels, R., & Heerdink, A. (n.d.). *mzcore* (Version 0.11.0). <https://github.com/rusteomics/mzcore>
- 149
- 150 Smith, L. M., & Kelleher, N. L. (2013). Proteoform: a single term describing protein complexity.
- 151 *Nature Methods*, 10(3), 186–187. <https://doi.org/10.1038/nmeth.2369>
- 152 Team, P. D. (2025). *pandas-dev/pandas: Pandas*. *Zenodo (CERN European Organization for*
- 153 *Nuclear Research)*. <https://doi.org/10.5281/zenodo.17992932>
- 154 Vink, R., De Gooijer, S., Beedie, A., Burghoorn, G., Nameexhaustion, Peters, O., Gorelli,
- 155 M. E., Reswqa, Van Zundert, J., Marshall, Hulselmans, G., Grinstead, C., Denecker, K.,
- 156 Manley, L., chieIP, Turner-Trauring, I., Valtar, K., Mitchell, L., Sprenkels, A., ... Magarick,
- 157 J. (2025). *pola-rs/polars: Python Polars 1.36.1*. *Zenodo (CERN European Organization*
- 158 *for Nuclear Research)*. <https://doi.org/10.5281/zenodo.17873635>

DRAFT