

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Published: unpublished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 (Glygen-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).

84 Example Usage

85 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
```

86 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]
```

87 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)
```

88 Mathematics

89 Peptacular calculates the molecular mass and isotopic patterns for AA sequences. This section
90 presents the mathematical framework that underlies these calculations.

91 Base Mass

92 The base mass M_{base} of a AA sequence containing modifications is calculated as the sum of
93 all constituent molecular components:

$$94 \quad M_{base} = \sum_{i=1}^n m_{AA_i} + M_N + M_C + M_S + M_I + M_R + M_U + \mathbb{1}_{precursor} \cdot M_L$$

95 where:

- 96 ▪ n is the sequence length
- 97 ▪ m_{AA_i} is the mass of amino acid at position i

- M_N is the total mass of N-terminal modifications
 - M_C is the total mass of C-terminal modifications
 - M_S is the total mass of static/fixed modifications (Applied to sequence)
 - M_I is the total mass of position-specific modifications
 - M_R is the total mass of modifications within defined sequence intervals
 - M_U is the total mass of modifications with unknown positions
 - M_L is the total mass of labile modifications
 - $\mathbb{1}_{\text{precursor}}$ is an indicator function: 1 for precursor ions, 0 for fragment ions
- Labile modifications are included only for precursor and neutral ion types.

Neutral Mass

The neutral mass M_{neutral} is calculated by combining the base mass with ion-type adjustments, isotope modifications, neutral deltas.

$$M_{\text{neutral}} = M_{\text{base}} + M_{\text{ion}} + M_{\text{isotope}} + M_{\text{ndelta}}$$

where:

- M_{base} is the peptide base mass from the previous section
- M_{ion} is the ion-type-specific mass offset
- M_{isotope} is the mass shift from a specific isotopic species
- M_{ndelta} is the mass change from neutral losses/gains

Mass-to-charge Ratio

The mass-to-charge (m/z) ratio is calculated by incorporating charge carriers and electron mass corrections to the neutral mass:

$$\frac{m}{z} = \frac{M_{\text{neutral}} + M_{\text{adduct}} - z \cdot m_e}{z}$$

where:

- M_{neutral} is the neutral fragment mass
- M_{adduct} is the total mass of charge carriers
- z is the total charge state
- $m_e = 0.0005485799$ Da (electron mass)

Isotopic Distribution

The isotopic distribution of a peptide is determined by convolving the isotopic patterns of all constituent elements. For a peptide with elemental composition $\{E_1 : n_1, E_2 : n_2, \dots, E_k : n_k\}$, the isotopic distribution is:

$$P(\text{total}) = P(E_1)^{n_1} \otimes P(E_2)^{n_2} \otimes \dots \otimes P(E_k)^{n_k}$$

where $P(E_i)$ is the natural isotopic distribution of element E_i , n_i is the count of that element, and \otimes represents the convolution operation.

The computational complexity of isotopic distribution calculations scales with the number of isotopic peaks retained during the convolution process. To balance accuracy with computational efficiency, Peptacular implements several parameters to limit the number of isotopic peaks propagated through convolution operations.

Averagine Model

When the exact elemental composition is unknown, the averagine model provides an estimate of composition based on molecular mass using empirically-derived atomic ratios. The averagine values used in Peptacular were calculated by determining the cumulative count of all atoms

present in the entire human proteome (all reviewed proteins from UP000005640, downloaded from UniProt on February 2, 2026). This cumulative count was then normalized by the total monoisotopic mass of the proteome, yielding an atoms-per-dalton ratio for each element.

The composition is calculated as:

$$n_E = r_E \cdot M_{\text{neutral}} + n_{E,\text{ion}}$$

where:

- n_E is the estimated count of element E
- r_E is the averagine ratio (atoms per dalton) for element E
- M_{neutral} is the neutral peptide mass
- $n_{E,\text{ion}}$ is the elemental contribution from the ion type

The averagine ratios (atoms/Da) derived from the human proteome are:

- C: 0.044179
- H: 0.069749
- N: 0.012344
- O: 0.013352
- S: 0.000400

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]
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]

Table with 4 columns: Feature, Example, and Support. It lists various ProForma 2.1 features like Cross-linkers, Branches, GNO modifications, Glycan compositions, Charged formulas, Controlling placement, Global isotope, Fixed modifications, Chimeric spectra, Charges, and Ion notation, along with their corresponding example syntax and support levels.

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.

Figure 1: Parallelization Performance - GIL Enabled vs GIL Disabled (Python 3.14t)

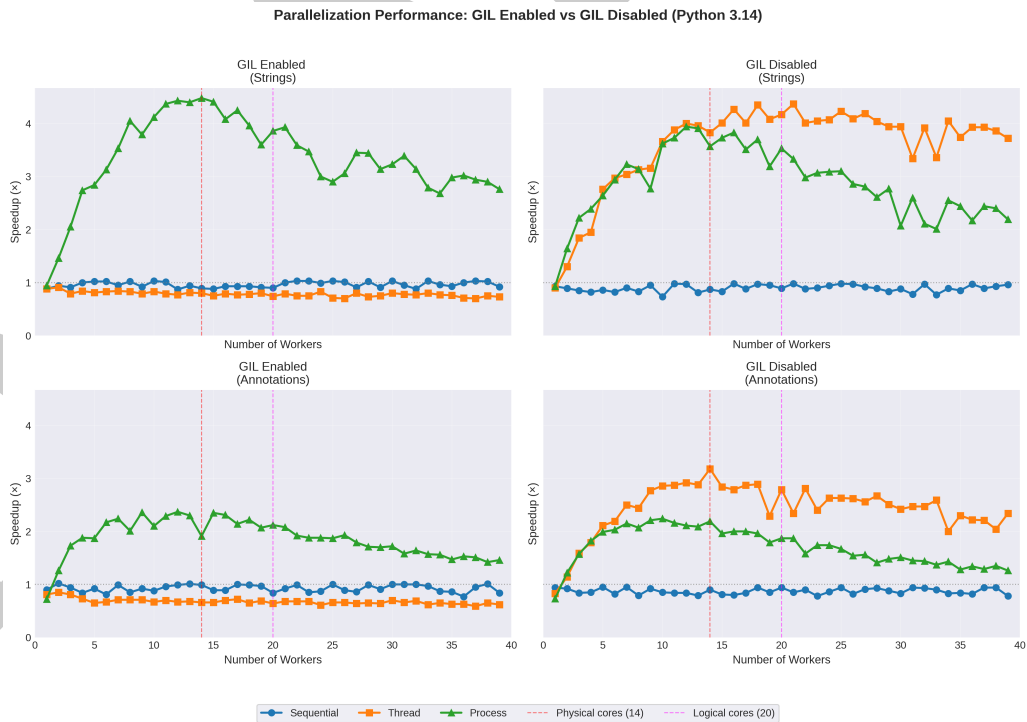


Figure 1: This figure presents a parallelization performance comparison for mass calculations of 10,000 randomly generated modified peptides with lengths ranging from 10 to 30 amino acids. The benchmark evaluates serialized annotations (strings) and annotation objects across different parallelization methods and varying numbers of workers in both GIL-enabled and GIL-disabled configurations. Single-worker sequential-based execution serves as the baseline for speedup calculations (0.336s ± 0.011s for serialized strings; 0.178s ± 0.004s for annotation objects). The benchmark was conducted on an Intel i7-12700H processor (14 cores, 20 threads) with 64GB RAM using Python 3.14t.

165 AI usage disclosure

166 Generative AI models were employed to support the development of this software package.
 167 Specifically, Claude Sonnet 4.5, Gemini 2.0 Pro, and GitHub Copilot's autocomplete extension
 168 were utilized for code generation, test development, debugging assistance, and documentation
 169 preparation. These tools were accessed through the Copilot extension in Visual Studio Code.
 170 Additionally, Type.ai was used to assist in manuscript preparation. All AI-generated content
 171 was subsequently reviewed and verified for accuracy.

172 Availability

173 Peptacular is distributed through PyPI (<https://pypi.org/project/peptacular/>) and available
 174 as open-source software on GitHub (<https://github.com/tacular-omics/peptacular>).
 175 Documentation is accessible at <https://peptacular.readthedocs.io>. The software is released
 176 under the MIT license.

177 Acknowledgements

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

184 Angel, T. E., Aryal, U. K., Hengel, S. M., Baker, E. S., Kelly, R. T., Robinson, E. W., &
 185 Smith, R. D. (2012). Mass spectrometry-based proteomics: existing capabilities and future
 186 directions. *Chemical Society Reviews*, 41(10), 3912. <https://doi.org/10.1039/c2cs15331a>

187 Cock, P. J. A., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., Friedberg,
 188 I., Hamelryck, T., Kauff, F., Wilczynski, B., & De Hoon, M. J. L. (2009). Biopython:
 189 freely available Python tools for computational molecular biology and bioinformatics.
 190 *Bioinformatics*, 25(11), 1422–1423. <https://doi.org/10.1093/bioinformatics/btp163>

191 Creasy, D. M., & Cottrell, J. S. (2004). Unimod: Protein modifications for mass spectrometry.
 192 *PROTEOMICS*, 4(6), 1534–1536. <https://doi.org/10.1002/pmic.200300744>

193 Garrett, P. (2026). *tacular-omics/tacular: v1.0.1 - Zenodo + Docs* (Version v1.0.1). Zenodo.
 194 <https://doi.org/10.5281/zenodo.18475557>

195 Garrett, P. T., Turner, N. P., Nakorchevsky, A., Pankow, S., & Yates, J. R. (2025). *Mass*
 196 *spectrometry-based proteomics*. <https://doi.org/10.1016/b978-0-323-99507-8.00013-5>

197 Glygen-Glycan-Data. (n.d.). *GitHub - glygen-glycan-data/GNOME: GNOME - Glycan Naming*
 198 *and Subsumption Ontology*. <https://github.com/glygen-glycan-data/GNOME>

199 Goloborodko, A. A., Levitsky, L. I., Ivanov, M. V., & Gorshkov, M. V. (2013). Pyteomics—A
 200 Python framework for exploratory data analysis and rapid software prototyping in proteomics.
 201 *Journal of the American Society for Mass Spectrometry*, 24(2), 301–304. <https://doi.org/10.1007/s13361-012-0516-6>

203 Hupo-Psi. (n.d.-a). *GitHub - HUPO-PSI/psi-ms-CV: HUPO-PSI mass spectrometry CV*.
 204 <https://github.com/HUPO-PSI/psi-ms-CV>

205 Hupo-Psi. (n.d.-b). *GitHub - HUPO-PSI/xlmod-CV: Repo for the XLMOD ontology for*
 206 *chemical cross linkers*. <https://github.com/HUPO-PSI/xlmod-CV>

207 LeDuc, R. D., Schwämmle, V., Shortreed, M. R., Cesnik, A. J., Solntsev, S. K., Shaw, J.

- 208 B., Martin, M. J., Vizcaino, J. A., Alpi, E., Danis, P., Kelleher, N. L., Smith, L. M.,
209 Ge, Y., Agar, J. N., Chamot-Rooke, J., Loo, J. A., Pasa-Tolic, L., & Tsybin, Y. O.
210 (2018). ProForma: a standard proteoform notation. *Journal of Proteome Research*, 17(3),
211 1321–1325. <https://doi.org/10.1021/acs.jproteome.7b00851>
- 212 *RESID Database [PIR - Protein Information Resource]*. (n.d.). [https://proteininformationresource.](https://proteininformationresource.org/resid/)
213 [org/resid/](https://proteininformationresource.org/resid/)
- 214 Röst, H. L., Schmitt, U., Aebersold, R., & Malmström, L. (2013). pyOpenMS: A Python-based
215 interface to the OpenMS mass-spectrometry algorithm library. *PROTEOMICS*, 14(1),
216 74–77. <https://doi.org/10.1002/pmic.201300246>
- 217 Schulte, D., Gabriels, R., & Heerdink, A. (n.d.). *mzcore* (Version 0.11.0). [https://github.](https://github.com/rusteomics/mzcore)
218 [com/rusteomics/mzcore](https://github.com/rusteomics/mzcore)
- 219 Smith, L. M., & Kelleher, N. L. (2013). Proteoform: a single term describing protein complexity.
220 *Nature Methods*, 10(3), 186–187. <https://doi.org/10.1038/nmeth.2369>
- 221 Team, P. D. (2025). pandas-dev/pandas: Pandas. *Zenodo (CERN European Organization for*
222 *Nuclear Research)*. <https://doi.org/10.5281/zenodo.17992932>
- 223 Vink, R., De Gooijer, S., Beedie, A., Burghoorn, G., Nameexhaustion, Peters, O., Gorelli,
224 M. E., Reswqa, Van Zundert, J., Marshall, Hulselmans, G., Grinstead, C., Denecker, K.,
225 Manley, L., chieIP, Turner-Trauring, I., Valtar, K., Mitchell, L., Sprenkels, A., ... Magarick,
226 J. (2025). pola-rs/polars: Python Polars 1.36.1. *Zenodo (CERN European Organization*
227 *for Nuclear Research)*. <https://doi.org/10.5281/zenodo.17873635>