




# KeemenaPreprocessing.jl — Unicode-Robust Cleaning, Multi-Level Tokenisation & Streaming Offset Bundling for Julia NLP

Alexander V. Mantzaris <sup>1</sup>

<sup>1</sup> Department of Statistics and Data Science, University of Central Florida (UCF), USA

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

## Software

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

Editor: 

Submitted: 07 July 2025

Published: unpublished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](https://creativecommons.org/licenses/by/4.0/)).

## Summary

KeemenaPreprocessing.jl begins where raw text first enters a research workflow, applying a carefully chosen set of cleaning operations that work well for most corpora yet remain fully customisable. By default the toolkit lower-cases characters, folds accents, removes control glyphs, normalises whitespace, and replaces URLs, e-mails, and numbers by sentinel tokens; each rule may be toggled individually through an optional `PreprocessConfiguration`, so users can disable lower-casing for case-sensitive tasks or preserve digits for OCR evaluation without rewriting the pipeline.

After cleaning, the same configuration drives tokenisation. Keemena ships byte-, character-, and word-level tokenisers and will seamlessly wrap a user-supplied function—allowing, for instance, a spaCy segmentation pass when language-specific heuristics are required (Honribal et al., 2020). Multiple tokenisers can operate in one sweep, so a single corpus pass can yield both sub-word pieces for a language model and whitespace tokens for classical bag-of-words features. Each token stream is accompanied by dense offset vectors: words are anchored to their byte and character positions, sentences and paragraphs are delimited explicitly, and a cross-alignment table keeps byte  $\leftrightarrow$  char  $\leftrightarrow$  word mappings exact. This design guarantees that every higher-level span can be traced unambiguously back to the source bytes, a property indispensable for annotation projection and reversible data augmentation.

All artefacts—clean strings, token-ids, offset vectors, vocabulary statistics, and alignment tables are consolidated into a single `PreprocessBundle`. The bundle can be saved or loaded with one function call using the JLD2 format, making it a drop-in dependency for downstream embedding or language-model pipelines inspired by word2vec (Mikolov, 2013). For modest datasets, the entire pipeline executes in a single statement; for web-scale corpora, KeemenaPreprocessing’s streaming mode processes fixed-size token chunks in constant memory while still accumulating global frequency tables. Thus, whether invoked with default settings for a quick experiment or finely tuned for production, KeemenaPreprocessing.jl offers a cohesive, Julia-native path from raw text to analysis-ready data (Bezanson et al., 2017). Many of these principles are introduced in (Bird et al., 2009);

## Statement of Need

Natural-language ML pipelines depend on reliable, reproducible preprocessing. Popular toolkits such as spaCy (Honribal et al., 2020), Stanford CoreNLP (Manning et al., 2014), and Gensim (Rehrek & Sojka, 2010) are Python or Java-centric, require heavyweight installations, and assume the full corpus fits in memory or on a local filesystem. While `WordTokenizers.jl` provides basic tokenisation for Julia (Kaushal et al., 2020), Julia users still lack an integrated, streaming pipeline that:

- 41     ▪ Scales beyond RAM through chunked streaming.
- 42     ▪ Tracks fine-grained offsets so models can mix sub-word and sentence-level features.
- 43     ▪ Lives entirely in Julia, avoiding Python/Java dependencies and enabling zero-copy interop
- 44         with Julia's numerical stack.
- 45     KeemenaPreprocessing.jl fills this gap, letting researchers preprocess billions of tokens on
- 46     commodity hardware while retaining compatibility with embedding or language-model training
- 47     workflows inspired by word2vec (Mikolov, 2013).

## 48     Acknowledgements

49     Thanks to the Julia community for their continued support of open-source scientific computing.

## 50     References

- 51     Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to
- 52         numerical computing. *SIAM Review*, 59(1), 65–98. <https://doi.org/10.1137/141000671>
- 53     Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with python: Analyzing*
- 54         *text with the natural language toolkit*. " O'Reilly Media, Inc."
- 55     Honnibal, M., Montani, I., Van Landeghem, S., Boyd, A., & others. (2020). *spaCy: Industrial-*
- 56         *strength natural language processing in python*.
- 57     Kaushal, A., White, L., Innes, M., & Kumar, R. (2020). WordTokenizers. JI: Basic tools for
- 58         tokenizing natural language in julia. *Journal of Open Source Software*, 5(46), 1956.
- 59     Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., & McClosky, D. (2014).
- 60         The stanford CoreNLP natural language processing toolkit. *Proceedings of 52nd Annual*
- 61         *Meeting of the Association for Computational Linguistics: System Demonstrations*, 55–60.
- 62     Mikolov, T. (2013). Efficient estimation of word representations in vector space. *arXiv Preprint*
- 63         *arXiv:1301.3781*, 3781.
- 64     Rehrek, R., & Sojka, P. (2010). *Software framework for topic modelling with large corpora*.