The Journal of Open Source Software

KeemenaPreprocessing.jl: Unicode-Robust Cleaning, Multi-Level Tokenisation and Streaming Offset Bundling for Julia NLP

Alexander V. Mantzaris [1]

1 Department of Statistics and Data Science, University of Central Florida (UCF), USA

DOI: 10.21105/joss.09348

Software

- Review
- Repository
- Archive

Editor: Owen Lockwood

Reviewers:

- @atantos
- @akki2825

Submitted: 07 July 2025
Published: 23 February 2026

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 (Honnibal et al., 2020).
Multiple segmentation levels can be recorded in one sweep (byte, character, word, sentence, paragraph, document), so downstream code can choose the granularity it needs without re-running preprocessing. 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 ↔ char ↔ 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`. For Julia workflows, the bundle can be saved or loaded with one function call using JLD2 (a Julia-native serialization format). This persistence mechanism is independent of any particular model: the bundle exposes the token-id stream, vocabulary, offsets, and alignment tables needed by embedding and language-model training code, including word2vec-style pipelines (Mikolov et al., 2013), and these same arrays can be exported to other interchange formats when interoperability is required. 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 design principles are present in classic NLP preprocessing practices (Bird et al., 2009).

Statement of Need

Modern NLP and language-modeling experiments depend on preprocessing that is reliable, reproducible, and auditable: changes in cleaning rules, tokenisation boundaries, or vocabulary

construction can change model behavior and evaluation. Some ecosystems provide full-featured NLP toolkits (e.g. spaCy (Honnibal et al., 2020), Stanford CoreNLP (Manning et al., 2014), and Gensim (Řehůřek & Sojka, 2010)), but these are primarily developed in and for Python/Java and are commonly used as end-to-end NLP pipelines rather than as a lightweight preprocessing step that produces a stable output type for downstream Julia modeling.

Within Julia, existing packages such as WordTokenizers.jl (Kaushal et al., 2020) provide fast tokenisation primitives, but many research workflows require additional infrastructure that is typically reimplemented per project: (i) a deterministic vocabulary and token-id representation, (ii) multi-level offsets and span traceability back to the raw text, and (iii) predictable memory behavior for corpora that cannot be loaded into RAM in one piece.

KeemenaPreprocessing.jl fills this gap by focusing narrowly on corpus preprocessing as an explicit, reproducible artifact-building stage. It is intended for researchers and practitioners who preprocess large corpora for training or evaluating ML/NLP models and who need stable alignment across tokenisation levels (byte/char/word/sentence/paragraph/document). It is *not* intended to be a general NLP toolkit (tagging, parsing, NER, etc), nor a collection of tokenizer implementations; instead, it emphasizes a stable data model, deterministic preprocessing, and loose interoperability via user-supplied callables.

Concretely, KeemenaPreprocessing provides:

- A streaming, two-pass preprocessing workflow that supports corpora larger than available RAM by processing fixed-size token chunks.
- Deterministic vocabulary construction with user-defined special tokens, producing stable token-id streams suitable for downstream modeling.
- Dense offset tables and cross-level alignment maps that preserve exact traceability between bytes, characters, and higher-level tokenisation units, enabling robust span alignment and evaluation.
- A compact `PreprocessBundle` interface that can be saved and loaded for long-running experiments while remaining a plain Julia object for direct use in numerical and modeling code.

These design choices support Julia-native modeling pipelines while keeping the preprocessing step transparent, testable, and reproducible—principles that underlie many established NLP workflows (Bird et al., 2009).

Acknowledgements

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

References

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98. https://doi.org/10.1137/141000671

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python: Analyzing text with the natural language toolkit*. "O'Reilly Media, Inc.". ISBN: 978-0-596-51649-9

Honnibal, M., Montani, I., Van Landeghem, S., & Boyd, A. (2020). *spaCy: Industrial-strength Natural Language Processing in Python*. https://doi.org/10.5281/zenodo.1212303

Kaushal, A., White, L., Innes, M., & Kumar, R. (2020). WordTokenizers.jl: Basic tools for tokenizing natural language in Julia. *Journal of Open Source Software*, *5*(46), 1956. https://doi.org/10.21105/joss.01956

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., & McClosky, D. (2014). The stanford CoreNLP natural language processing toolkit. *Proceedings of 52nd Annual*

Meeting of the Association for Computational Linguistics: System Demonstrations, 55–60. https://doi.org/10.3115/v1/P14-5010

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv Preprint arXiv:1301.3781*, *3781*. https://doi.org/10.48550/arXiv.1301.3781

Řehůřek, R., & Sojka, P. (2010). Software framework for topic modelling with large corpora. *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. https://doi.org/10.13140/2.1.2393.1847