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

July 7 2025

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 [@honnibal2020spacy]. 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 <-> 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`. 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 [@mikolov2013efficient]. 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 [@julia]. Many of these principles are introduced in [@bird2009natural];

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 (eg. spaCy [@honnibal2020spacy], Stanford CoreNLP [@manning2014stanford], and Gensim [@vrehuuvrek2010software]), 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 [@kaushal2020wordtokenizers] provide fast tokenisation primitives [@kaushal2020wordtokenizers], 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 [@bird2009natural].

Acknowledgements

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

References