

Package ‘jiebaRS’

August 4, 2026

Title Chinese Text Segmentation, POS Tagging, and Keyword Extraction

Version 0.2.0

Description Provides fast Chinese text segmentation, keyword extraction via 'TF-IDF' and 'TextRank', and part-of-speech tagging, powered by a 'Rust' backend ('jieba-rs'). Supports custom dictionaries, user words, stop words, IDF files, and HMM models, with parallel batch processing of multiple strings. Serves as a modern, maintained replacement for the 'jiebaR' package.

License MIT + file LICENSE

URL <https://yousa-mirage.github.io/jiebaRS/>,
<https://github.com/Yousa-Mirage/jiebaRS>

BugReports <https://github.com/Yousa-Mirage/jiebaRS/issues>

Encoding UTF-8

Config/rextendr/version 0.5.0

SystemRequirements Cargo (Rust's package manager), rustc >= 1.65.0, xz

Depends R (>= 4.2)

Imports cli, rlang

Suggests pkgdown, rmarkdown, spelling, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Config/testthat/parallel true

Language en-US

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Hao Cheng [aut, cre, cph]

Maintainer Hao Cheng <Yousa-Mirage@foxmail.com>

Repository CRAN

Date/Publication 2026-08-04 16:20:13 UTC

Contents

count_ngrams	2
filter_segment	3
freq	4
get_idf	5
get_tuple	6
keywords	7
keywords_df	7
new_user_word	8
segment	9
segment_batch	10
tagging	11
tagging_batch	13
textrank	14
textrank_df	14
worker	15
Index	18

count_ngrams	<i>Count n-grams from segmented text</i>
--------------	--

Description

Count contiguous n-grams from a segmented character vector or from each element of a list of segmented character vectors.

This function is a drop-in replacement for `jiebaR::get_tuple()`, which is deprecated in `jiebaRS`. See Details for more information.

Usage

```
count_ngrams(  
  x,  
  ...,  
  n = 2,  
  sep = " ",  
  sort = TRUE,  
  format = c("data.frame", "vector")  
)
```

Arguments

- x A character vector of tokens or a list of character vectors.
- ... Must be empty. This enforces that optional arguments such as `n`, `sep`, `sort`, and `format` are supplied with explicit names.

n	A positive integer or integer vector giving the n-gram sizes to count. The default is 2. If n is a integer vector of length > 1, n-grams of all specified sizes will be counted.
sep	Separator inserted between tokens when constructing the n-gram label. The default is " ", a single space.
sort	Whether to sort results by descending frequency. The default is TRUE. If FALSE, results keep first-appearance order within each requested n.
format	Output format. "data.frame" returns a data frame with term, n, and count columns. "vector" returns a named integer vector using the n-gram terms as names.

Details

The original `jiebaR::get_tuple()` interface has several design problems:

1. Its n-gram extraction behavior does not match the most obvious reading of the argument name: `size = n` counts all contiguous n-grams from 2:n, not just the exact size n.
2. Its documentation says it accepts list input, but the original exported implementation does not reliably support lists.
3. It concatenates tokens without a separator, which makes tuple boundaries ambiguous.

`count_ngrams()` addresses these issues, providing more explicit and abundant parameters. In addition, this function is about **1.3x** to **2.0x** faster than `jiebaR::get_tuple()`.

Value

N-gram counts in the requested format.

See Also

[get_tuple\(\)](#)

Examples

```
count_ngrams(c("\u6211", "\u7231", "R"), n = 2)
count_ngrams(c("\u6211", "\u7231", "R"), n = 1:2, format = "data.frame")
count_ngrams(c("a", "b", "b", "b", "a"), n = 1, sort = FALSE)
count_ngrams(list(c("a", "b", "c"), c("a", "b")), n = 2)
```

filter_segment

Filter segmentation results

Description

Remove selected words from a segmented character vector or from each element of a list of segmented character vectors.

Usage

```
filter_segment(input, filter_words, keep_na = TRUE)
```

Arguments

- input A character vector or a list of character vectors.
- filter_words A character vector of words to remove.
- keep_na Whether to keep NA values in the returned result. The default TRUE matches jiebaR::filter_segment().

Details

This is a modern reimplementation of jiebaR::filter_segment() with the same core filtering behavior under the default settings.

In the reproducible benchmark, this version is about **110x** to **140x** faster than jiebaR::filter_segment() on the tested workloads.

Value

An object with the same shape as input, with matching words removed.

Examples

```
filter_segment(c("abc", "def", " ", "."), c("abc"))
filter_segment(c("a", NA, "b", "a"), c("b"), keep_na = FALSE)
input <- list(
  c("\u6211", "\u662f", "\u6d4b\u8bd5"),
  c("\u6d4b\u8bd5", "\u6587\u672c", "\u6211")
)
filter_segment(input, "\u6211")
```

freq	<i>The frequency of words</i>
------	-------------------------------

Description

This function returns the frequency of words.

Usage

```
freq(x, ..., sort = FALSE)
```

Arguments

- x A character vector of words.
- ... Must be empty. This enforces that optional arguments such as sort are supplied with explicit names.
- sort Whether to sort the result by descending frequency. The default FALSE keeps the first-appearance order.

Value

A data frame with char and freq columns.

Examples

```
freq(c("b", "a", "b", "c", "a"))
freq(c("b", "a", "b", "c", "a"), sort = TRUE)
```

get_idf	<i>Generate IDF dict</i>
---------	--------------------------

Description

Generate IDF dict from a list of documents.

Usage

```
get_idf(x, stop_word = NULL, stop_word_file = NULL, path = NULL)
```

Arguments

x	a list of character vectors. Each vector represents a document of already-segmented words.
stop_word	Optional character vector of stop words supplied directly.
stop_word_file	Optional file path containing one stop word per line.
path	Optional output file path. When NULL, a data frame is returned. Otherwise, the result is written to the file as word idf_value per line (the format expected by <code>worker(type = "keywords", idf = ...)</code>) and the path is returned invisibly.

Details

Input list contains multiple character vectors with words, and each vector represents a document.

Stop words will be removed from the result.

If path is not NULL, it will write the result to the path.

Value

A data frame with name and count columns, or a file path (invisibly) when path is supplied.

Examples

```
get_idf(list(c("abc", "def"), c("abc", " ")))
```

`get_tuple`*Compatibility wrapper for jiebaR::get_tuple()*

Description

`get_tuple()` is kept only for compatibility with jiebaR. New code should use `count_ngrams()` instead.

Usage

```
get_tuple(x, size = 2, dataframe = TRUE)
```

Arguments

<code>x</code>	A character vector of tokens or a list of character vectors.
<code>size</code>	A single integer ≥ 2 . The compatibility semantics count all contiguous n-grams from 2 up to size.
<code>dataframe</code>	Whether to return a data frame. If FALSE, a named integer vector is returned.

Details

This function is deprecated and should not be used in new code. It is provided only as a compatibility wrapper around `count_ngrams()` and replicates the behavior of `jiebaR::get_tuple()`.

Prefer `count_ngrams()` because the original `jiebaR::get_tuple()` interface has several design problems:

1. Its n-gram extraction behavior does not match the most obvious reading of the argument name: `size = n` counts all contiguous n-grams from 2:n, not just the exact size `n`.
2. Its documentation says it accepts list input, but the original exported implementation does not reliably support lists.
3. It concatenates tokens without a separator, which makes tuple boundaries ambiguous.

Value

If `dataframe = TRUE`, a data frame with name and count columns, sorted by descending count. Otherwise, a named integer vector.

See Also

[count_ngrams\(\)](#)

Examples

```
suppressWarnings(get_tuple(c("sd", "sd", "sd", "rd"), 2))
```

keywords	<i>Extract keywords from text</i>
----------	-----------------------------------

Description

Extract TF-IDF keywords from a single in-memory string with a keyword worker created by `worker()`. This is separate from `textrank()`, which uses TextRank weighting.

Usage

```
keywords(code, jiebar, ..., format = c("vector", "data.frame", "legacy"))
```

Arguments

code	A character to analyze.
jiebar	A <code>jieba_worker</code> object created with <code>worker(type = "keywords")</code> .
...	Must be empty. This enforces that optional arguments such as <code>format</code> are supplied with explicit names.
format	Output format. "vector" returns a named numeric vector, "data.frame" returns a data frame with <code>term</code> and <code>weight</code> columns, and "legacy" returns the old jiebaR style character vector with weights in <code>names()</code> . Default is "vector".

Value

Keyword results in the requested format.

keywords_df	<i>Extract keywords as a data frame</i>
-------------	---

Description

Convenience wrapper around `keywords()` that always returns a data frame with `term` and `weight` columns.

Usage

```
keywords_df(x, jiebar)
```

Arguments

x	A character to analyze.
jiebar	A <code>jieba_worker</code> object created with <code>worker(type = "keywords")</code> .

Value

A data frame with `term` and `weight` columns.

new_user_word	<i>Add user word</i>
---------------	----------------------

Description

Add one or more custom words to a jieba worker.

Usage

```
new_user_word(worker, words, tags = "n", freq = NULL)
```

```
add_word(worker, words, tags = "n", freq = NULL)
```

Arguments

worker	A jieba_worker object.
words	A non-empty string or a non-empty character vector of new words.
tags	A single tag or a character vector of tags. Defaults to "n" for each supplied word. NA values are allowed and will be interpreted as missing tags.
freq	Optional positive integer frequency or integer vector of frequencies. Defaults to NULL. NA values are allowed and will be interpreted as frequencies to infer from the current dictionary.

Value

NULL, invisibly. Called for its side effect of adding the supplied words to worker, thereby modifying the state used by subsequent operations with the same worker.

Examples

```
cutter <- worker()
segment("\u91cf\u5b50\u673a\u5668\u72d7", cutter)
new_user_word(cutter, "\u91cf\u5b50\u673a\u5668\u72d7", tags = "n", freq = 1000L)
segment("\u91cf\u5b50\u673a\u5668\u72d7", cutter)

cutter2 <- worker()
add_word(
  cutter2,
  c("\u8d85\u5bfc\u91cf\u5b50\u6bd4\u7279", "\u91cf\u5b50\u673a\u5668\u72d7"),
  tags = c(NA, "n"),
  freq = c(NA, 1000L)
)
segment("\u8d85\u5bfc\u91cf\u5b50\u6bd4\u7279", cutter2)
```

segment	<i>Segment text with a jieba worker</i>
---------	---

Description

Segment one or more strings with a jieba_worker created by `worker()`.

Usage

```
segment(
  code,
  jiebar,
  ...,
  mod = NULL,
  batch = c("list", "data.frame", "flatten")
)
```

Arguments

code	A character vector to segment.
jiebar	A jieba_worker object.
...	Must be empty. This enforces that optional arguments such as mod and batch are supplied with explicit names.
mod	Deprecated Compatibility argument retained from jiebaR. This argument no longer has any effect.
batch	Batch aggregation mode for multi-string input . Must be one of "list", "data.frame", or "flatten". The default is "list".

Details

For a single input string, `segment()` always returns a character vector of segmented tokens.

When the worker was created with `symbol = TRUE`, symbol-like characters are retained during pre-processing. For example, AK-47, N95, X-900, and 4.55 remain intact. With `symbol = FALSE` (the default), non-letter, non-number, and non-mark characters are replaced with spaces before segmentation, so the same input is returned as AK, 47, N95, X, 900, 4, and 55.

In the current release benchmarks on the bundled *Fortress Besieged* and *Dream of the Red Chamber* texts, `jiebaRS::segment()` is about **1.7x to 1.9x faster** than `jiebaR::segment()` when each novel is segmented as one long string. When the input is many short strings segmented in parallel, `jiebaRS::segment()` reaches about **7x to 12x speedup** over jiebaR.

For very long texts, splitting into about **32 to 128 chunks** before segmentation is recommended for good throughput.

For multiple input strings, the argument `batch` controls how the per-string token vectors are aggregated:

- "list": one character vector per input string.

- "data.frame": a data frame with doc_id and word columns.
- "flatten": all token vectors concatenated into one character vector.

When batch is omitted, jiebaRS returns list output for multi-string input.

The mod argument from jiebaR::segment() is retained only as a deprecated compatibility placeholder. In jiebaRS, segmentation behavior should be controlled by the worker type itself (for example, worker(type = "mix") or worker(type = "query")), not by mutating behavior at call time. When mod is supplied, jiebaRS warns and ignores it.

Value

Segmented tokens in the requested aggregation form.

Examples

```
seg <- worker()
text1 <- "\u5357\u4eac\u5e02\u957f\u6c5f\u5927\u6865"
text2 <- "\u8fd9\u662f\u4e00\u4e2a\u6d4b\u8bd5"
segment(text1, seg)
segment(c(text1, text2), seg, batch = "list")
segment(c(text1, text2), seg, batch = "data.frame")
```

segment_batch	<i>Segment a batch of strings</i>
---------------	-----------------------------------

Description

Convenience wrapper around `segment()` for multi-string input. When batch is omitted, `segment_batch()` will return **list** output by default.

Usage

```
segment_batch(texts, jiebar, ..., batch = c("list", "data.frame", "flatten"))
```

Arguments

texts	A character vector of strings to segment.
jiebar	A jieba_worker object.
...	Must be empty. This enforces that optional arguments such as batch are supplied with explicit names.
batch	Batch aggregation mode. Must be one of "list", "data.frame", or "flatten". The default is "list".

Details

`segment_batch()` is a convenience wrapper around `segment()` for explicit batch processing. It always treats texts as multi-string input. The returned object depends on batch:

- "list": one character vector per input string.
- "data.frame": a data frame with `doc_id` and `word` columns.
- "flatten": one concatenated character vector.

In the current release benchmarks on the bundled *Fortress Besieged* and *Dream of the Red Chamber* texts, batch segmentation reaches about **7x to 12x speedup** over the comparable jiebaR workflow on many-string inputs. For very long texts, splitting into about **32 to 128 chunks** before calling `segment_batch()` is recommended for good throughput.

Value

Segmented tokens in the requested aggregation form.

Examples

```
seg <- worker()
texts <- c("\u5357\u4eac\u5e02\u957f\u6c5f\u5927\u6865", "\u8fd9\u662f\u4e00\u4e2a\u6d4b\u8bd5")
segment_batch(texts, seg)
segment_batch(texts, seg, batch = "flatten")
```

tagging	<i>Tag text with a jiebaRS worker</i>
---------	---------------------------------------

Description

Tag one or more strings with a jieba_worker created by `worker()`.

Usage

```
tagging(
  code,
  jiebar,
  ...,
  format = c("vector", "data.frame", "legacy"),
  batch = c("list", "flatten")
)
```

Arguments

<code>code</code>	A non-empty character vector to tag.
<code>jiebar</code>	A jieba_worker object created with <code>worker(type = "tag")</code> .
<code>...</code>	Must be empty. This enforces that optional arguments such as <code>format</code> and <code>batch</code> are supplied with explicit names.

format	Output format for a single tagged string. Must be one of "vector", "data.frame", or "legacy".
batch	Aggregation mode for multi-string input. Must be one of "list" or "flatten".

Details

format controls the shape of each single-string tagging result:

- "vector": a named character vector with token names and tag values.
- "data.frame": a data frame with term and tag columns.
- "legacy": the old jiebaR layout with token values and tag names.

In the current release benchmarks on the bundled *Fortress Besieged* and *Dream of the Red Chamber* texts, `jiebaRS::tagging()` is about **1.6x to 1.8x faster** than `jiebaR::tagging()` when each novel is tagged as one long string. When the same content is split into many strings and processed in batch, `jiebaRS::tagging()` is about **2x to 5x faster** than `jiebaR`.

For very long texts, splitting before tagging is usually faster than sending one huge string. In the same release benchmarks, the best results appeared around **32 to 128 chunks**, while much finer splitting still helped but was no longer optimal.

When code contains multiple strings, batch controls how the per-string results are aggregated:

- "list": one single-string result per input string.
- "flatten": concatenate all results into one. The shape is decided by format: "vector"/"legacy" produce a named character vector, while "data.frame" produces a combined data frame with a doc_id column.

When batch is omitted, `jiebaRS` returns "vector" for single-string input and "list" for multi-string input.

Value

Tagging results in the requested format.

Examples

```

tagger <- worker(type = "tag")
text1 <- "\u8fd9\u662f\u4e00\u4e2a\u6d4b\u8bd5"
text2 <- "\u518d\u6765\u4e00\u6b21"
tagging(text1, tagger)
tagging(c(text1, text2), tagger)
tagging(c(text1, text2), tagger, format = "data.frame", batch = "flatten")

```

tagging_batch	<i>Tag a batch of strings</i>
---------------	-------------------------------

Description

Convenience wrapper around `tagging()` for multi-string input. When batch is not supplied, `tagging_batch()` always returns list output.

Usage

```
tagging_batch(
  texts,
  jiebar,
  ...,
  format = c("vector", "data.frame", "legacy"),
  batch = c("list", "flatten")
)
```

Arguments

<code>texts</code>	A non-empty character vector to tag.
<code>jiebar</code>	A <code>jieba_worker</code> object created with <code>worker(type = "tag")</code> .
<code>...</code>	Must be empty. This enforces that optional arguments such as <code>format</code> and <code>batch</code> are supplied with explicit names.
<code>format</code>	Output format for each single tagged result. Must be one of "vector", "data.frame", or "legacy".
<code>batch</code>	Aggregation mode. Must be one of "list" or "flatten".

Details

`tagging_batch()` is a convenience wrapper for explicit multi-string input. The returned object depends on both `format` and `batch`:

- `batch = "list"`: returns one single-string tagging result per input string.
- `batch = "flatten"`: concatenates all results into one. The shape is decided by `format`: "vector"/"legacy" produce a named character vector, while "data.frame" produces a combined data frame with a `doc_id` column.

In the current release benchmarks on the bundled *Fortress Besieged* and *Dream of the Red Chamber* texts, batch tagging is about **2x to 5x faster** than the comparable jiebaR workflow on many-string inputs. For very long texts, the best throughput was usually reached by splitting into about **32 to 128 chunks**, while much finer splitting still helped but was no longer optimal.

Value

Tagging results in the requested format.

Examples

```

tagger <- worker(type = "tag")
texts <- c("\u8fd9\u662f\u4e00\u4e2a\u6d4b\u8bd5", "\u518d\u6765\u4e00\u6b21")
tagging_batch(texts, tagger)
tagging_batch(texts, tagger, format = "legacy", batch = "flatten")

```

textrank	<i>Extract TextRank keywords from text</i>
----------	--

Description

Extract TextRank-ranked keywords from a single in-memory string with a TextRank worker created by `worker()`. This is separate from `keywords()`, which uses TF-IDF weighting.

Usage

```
textrank(code, jiebar, ..., format = c("vector", "data.frame", "legacy"))
```

Arguments

code	A character to analyze.
jiebar	A jieba_worker object created with <code>worker(type = "textrank")</code> .
...	Must be empty. This enforces that optional arguments such as <code>format</code> are supplied with explicit names.
format	Output format. "vector" returns a named numeric vector, "data.frame" returns a data frame with term and weight columns, and "legacy" returns a jiebaR-style character vector with weights in <code>names()</code> . Default is "vector".

Value

TextRank results in the requested format.

textrank_df	<i>Extract TextRank keywords as a data frame</i>
-------------	--

Description

Convenience wrapper around `textrank()` that always returns a data frame with term and weight columns.

Usage

```
textrank_df(x, jiebar)
```

Arguments

x	A character to analyze.
jiebar	A jieba_worker object created with worker(type = "textrank").

Value

A data frame with term and weight columns.

worker	<i>Initialize a jiebaRS worker</i>
--------	------------------------------------

Description

This function can initialize a jiebaRS worker. See Details for more information.

Usage

```
worker(
  type = c("mix", "mp", "hmm", "full", "query", "tag", "keywords", "textrank"),
  stop_word = NULL,
  stop_word_file = NULL,
  hmm = TRUE,
  topn = 5L,
  min_keyword_length = 2L,
  idf = NULL,
  dict = NULL,
  user = NULL,
  symbol = FALSE,
  bylines = FALSE
)
```

Arguments

type	Worker type. Supported values are "mix", "mp", "hmm", "full", "query", "tag", "keywords", and "textrank". Default is "mix".
stop_word	Optional character vector of stop words supplied directly.
stop_word_file	Optional file path containing one stop word per line.
hmm	Logical scalar or character scalar. If logical, controls whether to enable HMM fallback for unknown terms. If character, must be a path to a custom HMM model file compatible with jieba-rs's hmm.model format, and HMM fallback is enabled with that model. Default is TRUE.
topn	Integer. The number of terms returned by keywords and textrank workers. Default is 5.
min_keyword_length	Positive integer. The minimum number of Unicode scalar values in terms returned by keywords and textrank workers. Default is 2.

idf	Optional character scalar. A path to a custom IDF dictionary file for keywords workers. Each line should be word idf_value. When NULL, the embedded default IDF dictionary is used. Ignored by non-keyword workers. Default is NULL.
dict	Optional character scalar. A path to a custom main dictionary file that <i>replaces</i> the embedded dictionary. Each line should be word [freq] [tag] (whitespace-separated; freq defaults to 1, tag defaults to empty). When NULL, the embedded dictionary is used. Default is NULL.
user	Optional character vector containing one or more paths to user dictionary files whose entries are <i>appended</i> to the main dictionary in the supplied order. Supported line formats are word, word freq, word tag, and word freq tag. Omitted frequencies are inferred from the current dictionary. Default is NULL.
symbol	Logical. Whether to keep symbol-like tokens in the sentence. Default is FALSE.
bylines	Deprecated compatibility argument retained from jiebaR. jiebaRS no longer uses this value; control batch aggregation directly in specific functions.

Details

The qmax argument is not supported. Although jiebaR documented qmax for query workers, the value was never actually passed to the underlying segmentation call. Similarly, the jieba-rs backend implements search-mode segmentation without a configurable query threshold. To avoid user confusion, jiebaRS omits the qmax argument entirely rather than retaining a no-op parameter.

jieba-rs does not expose dedicated public implementations for mp or hmm workers. jiebaRS therefore maps mp to cut(..., false) and hmm to cut(..., true). This is a compatibility approximation rather than a byte-for-byte reimplement of jiebaR, and jiebaRS warns once per R session when either type is requested.

tag workers use jieba-rs tagging on top of the default mixed segmentation path, which is the closest public behavior to jiebaR.

stop_word and stop_word_file can be both supplied at once and then be merged together. Then they will be normalized.

In jiebaRS, hmm accepts either a logical scalar or a file path. A logical value controls whether the underlying jieba-rs segmentation/tagging pipeline may fall back to HMM for unknown terms. A character scalar is interpreted as a path to a custom HMM model file and enables HMM fallback with that model. The flag affects mix and query workers directly, tag workers through the underlying mixed tagging path, and keywords workers through TF-IDF keyword extraction. mp, hmm, and full workers ignore the runtime switch because their jieba-rs backends do not use this runtime switch.

dict and user load dictionary files at worker creation time. dict *replaces* the embedded main dictionary entirely; user accepts one or more file paths whose entries are appended, in the supplied order, to whatever main dictionary is in place (default or custom dict). Main dictionary entries use word [freq] [tag]; an omitted frequency defaults to 1. User dictionary entries may use word, word freq, word tag (the legacy two-column jiebaR format), or word freq tag. User frequencies must be positive integers; when omitted, a frequency is inferred from the current dictionary. Files must be UTF-8; a leading byte order mark (BOM) is allowed.

worker

17

Value

A jieba_worker S3 object.

Index

`add_word(new_user_word)`, [8](#)

`count_ngrams`, [2](#)

`count_ngrams()`, [6](#)

Deprecated, [9](#), [16](#)

`filter_segment`, [3](#)

`freq`, [4](#)

`get_idf`, [5](#)

`get_tuple`, [6](#)

`get_tuple()`, [3](#)

`keywords`, [7](#)

`keywords()`, [7](#), [14](#)

`keywords_df`, [7](#)

`new_user_word`, [8](#)

`segment`, [9](#)

`segment()`, [10](#), [11](#)

`segment_batch`, [10](#)

`tagging`, [11](#)

`tagging()`, [13](#)

`tagging_batch`, [13](#)

`textrank`, [14](#)

`textrank()`, [7](#), [14](#)

`textrank_df`, [14](#)

`worker`, [15](#)

`worker()`, [7](#), [9](#), [11](#), [14](#)