Package: ragnar (via r-universe)

May 24, 2026

Title Retrieval-Augmented Generation (RAG) Workflows

Version 0.3.0.9000

Description Provides tools for implementing Retrieval-Augmented Generation (RAG) workflows with Large Language Models (LLM). Includes functions for document processing, text chunking, embedding generation, storage management, and content retrieval. Supports various document types and embedding providers ('Ollama', 'OpenAI'), with 'DuckDB' as the default storage backend. Integrates with the 'ellmer' package to equip chat objects with retrieval capabilities. Designed to offer both sensible defaults and customization options with transparent access to intermediate outputs. For a review of retrieval-augmented generation methods, see Gao et al. (2023) ``Retrieval-Augmented Generation for Large Language Models: A Survey'' <doi:10.48550/arXiv.2312.10997>.

License MIT + file LICENSE

URL https://ragnar.tidyverse.org/, https://github.com/tidyverse/ragnar

BugReports https://github.com/tidyverse/ragnar/issues

Depends R (>= 4.3.0)

Imports blob, cli, commonmark, curl, DBI, mirai (>= 2.6.0), dbplyr, dplyr, duckdb (>= 1.3.1), glue, httr2, jsonlite, methods, reticulate (>= 1.42.0), rlang (>= 1.1.0), rvest, S7, stringi, tidyr, vctrs, withr, xml2

Suggests connectcreds, ellmer (>= 0.3.0), gargle, knitr, lifecycle, mcptools (>= 0.2.0), pandoc, paws.common, rmarkdown, shiny, stringr, testthat (>= 3.0.0), tibble, jose, openssl

VignetteBuilder knitr

Config/Needs/website tidyverse/tidytemplate, rmarkdown

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/pak/sysreqs cmake libicu-dev libpng-dev libxml2-dev libssl-dev python3 xz-utils

Repository https://tidyverse.r-universe.dev

Date/Publication 2026-04-09 17:53:13 UTC

RemoteUrl https://github.com/tidyverse/ragnar

RemoteRef HEAD

RemoteSha 64acffa1ec78c32d8ceff54b5d340bf55d7a968c

Contents

`chunks_deoverlap` *Merge overlapping chunks in retrieved results*

Description

Groups and merges overlapping text chunks from the same origin in the retrieval results.

Usage

```
chunks_deoverlap(store, chunks)
```

Arguments

store	A RagnarStore object. Must have @version == 2.
chunks	A tibble of retrieved chunks, such as the output of ragnar_retrieve().

Details

When multiple retrieved chunks from the same origin have overlapping character ranges, this function combines them into a single non-overlapping region.

Value

A tibble of de-overlapped chunks.

---

embed_azure_openai *Uses Azure AI Foundry to create embeddings*

Description

Uses Azure AI Foundry to create embeddings

Usage

```
embed_azure_openai(
  x,
  endpoint = get_envvar("AZURE_OPENAI_ENDPOINT"),
  api_key = get_envvar("AZURE_OPENAI_API_KEY"),
  api_version = "2023-05-15",
  model,
  batch_size = 20L,
  api_args = list()
)
```

Arguments

x	x can be: • A character vector, in which case a matrix of embeddings is returned. • A data frame with a column named text, in which case the dataframe is returned with an additional column named embedding. • Missing or NULL, in which case a function is returned that can be called to get embeddings. This is a convenient way to partial in additional arguments like model, and is the most convenient way to produce a function that can be passed to the embed argument of ragnar_store_create().

endpoint	The Azure AI Foundry endpoint URL. A URI in the form of `https://<project>.cognitiveservices.a` Defaults to the value of the `AZURE_OPENAI_ENDPOINT` environment variable. This URL is appended with `/openai/deployments/{model}/embeddings`. Where `model` is the deployment name of the model.
api_key	resolved using env var `OPENAI_API_KEY`
api_version	The API version to use. Defaults to `2023-05-15`.
model	The deployment name of the model to use for generating embeddings.
batch_size	split `x` into batches when embedding. Integer, limit of strings to include in a single request.
api_args	A list of additional arguments to pass to the API request body.

Value

If `x` is a character vector, then a numeric matrix is returned, where `nrow = length(x)` and `ncol = <model-embedding-size>`. If `x` is a data.frame, then a new `embedding` matrix "column" is added, containing the matrix described in the previous sentence.

A matrix of embeddings with 1 row per input string, or a dataframe with an 'embedding' column.

`embed_bedrock` *Embed text using a Bedrock model*

Description

Embed text using a Bedrock model

Usage

```
embed_bedrock(x, model, profile = "", api_args = list())
```

Arguments

x	x can be: • A character vector, in which case a matrix of embeddings is returned. • A data frame with a column named `text`, in which case the dataframe is returned with an additional column named `embedding`. • Missing or `NULL`, in which case a function is returned that can be called to get embeddings. This is a convenient way to partial in additional arguments like `model`, and is the most convenient way to produce a function that can be passed to the `embed` argument of `ragnar_store_create()`.
model	Currently only Cohere.ai and Amazon Titan models are supported. There are no guardarails for the kind of model that is used, but the model must be available in the AWS region specified by the profile. You may look for available models in the Bedrock Model Catalog
profile	AWS profile to use. It's passed to paws.common::locate_credentials to locate AWS credentials.

`api_args` Additional arguments to pass to the Bedrock API. Depending on the `model`, you might be able to provide different parameters. Check the documentation for the model you are using in the Bedrock user guide.

Value

If `x` is missing returns a function that can be called to get embeddings. If `x` is not missing, a matrix of embeddings with 1 row per input string, or a dataframe with an 'embedding' column.

See Also

`embed_ollama()`

`embed_databricks` *Embed text using a Databricks model*

Description

`embed_databricks()` gets embeddings for text using a model hosted in a Databricks workspace. It relies on the **ellmer** package for managing Databricks credentials. See `ellmer::chat_databricks` for more on supported modes of authentication.

Usage

```
embed_databricks(
  x,
  workspace = databricks_workspace(),
  model = "databricks-bge-large-en",
  batch_size = 512L
)
```

Arguments

`x` x can be:

- A character vector, in which case a matrix of embeddings is returned.
- A data frame with a column named `text`, in which case the dataframe is returned with an additional column named `embedding`.
- Missing or `NULL`, in which case a function is returned that can be called to get embeddings. This is a convenient way to partial in additional arguments like `model`, and is the most convenient way to produce a function that can be passed to the `embed` argument of `ragnar_store_create()`.

`workspace` The URL of a Databricks workspace, e.g. `"https://example.cloud.databricks.com"`. Will use the value of the environment variable `DATABRICKS_HOST`, if set.

`model` The name of a text embedding model.

`batch_size` split `x` into batches when embedding. Integer, limit of strings to include in a single request.

`embed_google_gemini` *Embed using Google Vertex API platform*

Description

Embed using Google Vertex API platform

Usage

```
embed_google_gemini(
  x,
  model = "gemini-embedding-001",
  base_url = "https://generativelanguage.googleapis.com/v1beta",
  api_key = get_envvar("GEMINI_API_KEY"),
  dims = NULL,
  task_type = "RETRIEVAL_QUERY",
  batch_size = 20L
)

embed_google_vertex(
  x,
  model,
  location,
  project_id,
  task_type = "RETRIEVAL_QUERY"
)
```

Arguments

`x`	x can be: • A character vector, in which case a matrix of embeddings is returned. • A data frame with a column named `text`, in which case the dataframe is returned with an additional column named `embedding`. • Missing or `NULL`, in which case a function is returned that can be called to get embeddings. This is a convenient way to partial in additional arguments like `model`, and is the most convenient way to produce a function that can be passed to the `embed` argument of `ragnar_store_create()`.
`model`	Character specifying the embedding model. See supported models in Text embeddings API
`base_url`	string, url where the service is available.
`api_key`	resolved using env var `GEMINI_API_KEY`
`dims`	An integer, can be used to truncate the embedding to a specific size.
`task_type`	Used to convey intended downstream application to help the model produce better embeddings. If left blank, the default used is `"RETRIEVAL_QUERY"`. • `"RETRIEVAL_QUERY"`

- "RETRIEVAL_DOCUMENT"
- "SEMANTIC_SIMILARITY"
- "CLASSIFICATION"
- "CLUSTERING"
- "QUESTION_ANSWERING"
- "FACT_VERIFICATION"
- "CODE_RETRIEVAL_QUERY" For more information about task types, see Choose an embeddings task type.

batch_size — split x into batches when embedding. Integer, limit of strings to include in a single request.

location — Location, e.g. us-east1, me-central1, africa-south1 or global.

project_id — Project ID.

Functions

- embed_google_gemini(): Use the Gemini API to create embeddings.

Examples

```
embed_google_gemini("hello world")

## Not run:
embed_google_vertex(
 "hello world",
 model="gemini-embedding-001",
 project = "<your-project-id>",
 location = "us-central1"
)

## End(Not run)
```

---

embed_ollama *Embed Text*

Description

Embed Text

Usage

```
embed_ollama(
  x,
  base_url = "http://localhost:11434",
  model = "embeddinggemma:300m",
  batch_size = 10L
)
```

```
embed_openai(
  x,
  model = "text-embedding-3-small",
  base_url = "https://api.openai.com/v1",
  api_key = get_envvar("OPENAI_API_KEY"),
  dims = NULL,
  user = get_user(),
  batch_size = 20L
)

embed_lm_studio(
  x,
  model,
  base_url = "http://localhost:1234/v1",
  api_key = "lm-studio",
  dims = NULL,
  user = get_user(),
  batch_size = 20L
)
```

Arguments

x	x can be: • A character vector, in which case a matrix of embeddings is returned. • A data frame with a column named `text`, in which case the dataframe is returned with an additional column named `embedding`. • Missing or `NULL`, in which case a function is returned that can be called to get embeddings. This is a convenient way to partial in additional arguments like `model`, and is the most convenient way to produce a function that can be passed to the `embed` argument of `ragnar_store_create()`.
base_url	string, url where the service is available.
model	string; model name
batch_size	split `x` into batches when embedding. Integer, limit of strings to include in a single request.
api_key	resolved using env var `OPENAI_API_KEY`
dims	An integer, can be used to truncate the embedding to a specific size.
user	User name passed via the API.

Value

If `x` is a character vector, then a numeric matrix is returned, where `nrow = length(x)` and `ncol = <model-embedding-size>`. If `x` is a data.frame, then a new `embedding` matrix "column" is added, containing the matrix described in the previous sentence.

A matrix of embeddings with 1 row per input string, or a dataframe with an 'embedding' column.

Functions

- `embed_lm_studio()`: Embed Text using LMStudio. Indentical to `embed_openai()` but with suitable defaults for LMStudio.

Examples

```
text <- c("a chunk of text", "another chunk of text", "one more chunk of text")
## Not run:
text |>
  embed_ollama() |>
  str()

text |>
  embed_openai() |>
  str()

## End(Not run)
```

`embed_snowflake` *Generate embeddings using Snowflake*

Description

Uses the Cortex API `EMBED` functions to generate embeddings.

Usage

```
embed_snowflake(
  x,
  account = snowflake_account(),
  credentials = NULL,
  model = "snowflake-arctic-embed-m-v1.5",
  api_args = list(),
  batch_size = 512L
)
```

Arguments

x	x can be: • A character vector, in which case a matrix of embeddings is returned. • A data frame with a column named `text`, in which case the dataframe is returned with an additional column named `embedding`. • Missing or `NULL`, in which case a function is returned that can be called to get embeddings. This is a convenient way to partial in additional arguments like `model`, and is the most convenient way to produce a function that can be passed to the `embed` argument of `ragnar_store_create()`.

account	A Snowflake account identifier, e.g. "testorg-test_account". Defaults to the value of the SNOWFLAKE_ACCOUNT environment variable.
credentials	A list of authentication headers to pass into httr2::req_headers(), a function that returns them when called, or NULL, the default, to use ambient credentials.
model	string; model name
api_args	Named list of arbitrary extra arguments appended to the body of every chat API call. Combined with the body object generated by ellmer with modifyList().
batch_size	split x into batches when embedding. Integer, limit of strings to include in a single request.

Details

See complete documentation.

Authentication

- a *Programmatic Access Token* (PAT) defined via the SNOWFLAKE_PAT environment variable.
- A static OAuth token defined via the SNOWFLAKE_TOKEN environment variable.
- Key-pair authentication credentials defined via the SNOWFLAKE_USER and SNOWFLAKE_PRIVATE_KEY (which can be a PEM-encoded private key or a path to one) environment variables.
- Posit Workbench-managed Snowflake credentials for the corresponding account.
- Viewer-based credentials on Posit Connect. Requires the connectcreds package.

---

markdown_chunk *Chunk a Markdown document*

Description

markdown_chunk() splits a single Markdown string into shorter optionally overlapping chunks while nudging cut points to the nearest sensible boundary (heading, paragraph, sentence, line, word, or character). It returns a tibble recording the character ranges, headings context, and text for each chunk.

Usage

```
markdown_chunk(
  md,
  target_size = 1600L,
  target_overlap = 0.5,
  ...,
  max_snap_dist = target_size * (1 - target_overlap)/3,
  segment_by_heading_levels = integer(),
  context = TRUE,
  text = TRUE
)
```

Arguments

md	A MarkdownDocument, or a length-one character vector containing Markdown.
target_size	Integer. Target chunk size in characters. Default: 1600 (≈ 400 tokens, or 1 page of text). Actual chunk size may differ from the target by up to 2 * max_snap_dist. When set to NULL, NA or Inf and used with segment_by_heading_levels, chunk size is unbounded and each chunk corresponds to a segment.
target_overlap	Numeric in [0, 1). Fraction of desired overlap between successive chunks. Default: 0.5. Even when 0, some overlap can occur because the last chunk is anchored to the document end.
...	These dots are for future extensions and must be empty.
max_snap_dist	Integer. Furthest distance (in characters) a cut point may move to reach a semantic boundary. Defaults to one third of the stride size between target chunk starts. Chunks that end up on identical boundaries are merged.
segment_by_heading_levels	Integer vector with possible values 1:6. Headings at these levels are treated as segment boundaries; chunking is performed independently for each segment. No chunk will overlap a segment boundary, and any future deoverlapping will not combine segments. Each segment will have a chunk that starts at the segment start and a chunk that ends at the segment end (these may be the same chunk or overlap substantially if the segment is short). Default: disabled.
context	Logical. Add a context column containing the Markdown headings in scope at each chunk start. Default: TRUE.
text	Logical. If TRUE, include a text column with the chunk contents. Default: TRUE.

Value

A MarkdownDocumentChunks object, which is a tibble (data.frame) with with columns start end, and optionally context and text. It also has a @document property, which is the input md document (potentially normalized and converted to a MarkdownDocument).

See Also

ragnar_chunks_view() to interactively inspect the output of markdown_chunk(). See also MarkdownDocumentChunks() and MarkdownDocument(), where the input and return value of markdown_chunk() are described more fully.

Examples

```
md <- "
# Title

## Section 1

Some text that is long enough to be chunked.

A second paragraph to make the text even longer.
```

```
## Section 2

More text here.

### Section 2.1

Some text under a level three heading.

#### Section 2.1.1

Some text under a level four heading.

## Section 3

Even more text here.
"

markdown_chunk(md, target_size = 40)
markdown_chunk(md, target_size = 40, target_overlap = 0)
markdown_chunk(md, target_size = NA, segment_by_heading_levels = c(1, 2))
markdown_chunk(md, target_size = 40, max_snap_dist = 100)
```

`MarkdownDocument` *Markdown documents*

Description

`MarkdownDocument` represents a complete Markdown document stored as a single character string. The constructor normalizes `text` by collapsing lines and ensuring UTF-8 encoding, so downstream code can rely on a consistent format.

`read_as_markdown()` is the recommended way to create a `MarkdownDocument`. The constructor itself is exported only so advanced users can construct one by other means when needed.

Arguments

`text`	[string] Markdown text.
`origin`	[string] Optional source path or URL. Defaults to the `"origin"` attribute of `text`, if present, otherwise `NULL`.

Value

An S7 object that inherits from `MarkdownDocument`, which is a length 1 string of markdown text with an `@origin` property.

Examples

```
md <- MarkdownDocument(
  "# Title\n\nSome text.",
  origin = "example.md"
)
md
```

`MarkdownDocumentChunks` *Markdown documents chunks*

Description

`MarkdownDocumentChunks` stores information about candidate chunks in a Markdown document. It is a tibble with three required columns:

- `start`, `end` — integers. These are character positions (1-based, inclusive) in the source `MarkdownDocument`, so that `substring(md, start, end)` yields the chunk text. Ranges can overlap.
- `context` — character. A general-purpose field for adding context to a chunk. This column is combined with `text` to augment chunk content when generating embeddings with `ragnar_store_insert()`, and is also returned by `ragnar_retrieve()`. Keep in mind that when chunks are deoverlapped (in `ragnar_retrieve()` or `chunks_deoverlap()`), only the context value from the first chunk is kept. `markdown_chunk()` by default populates this column with all the markdown headings that are in-scope at the chunk start position.

Additional columns can be included.

The original document is available via the `@document` property.

For normal use, chunk a Markdown document with `markdown_chunk()`; the class constructor itself is exported only so advanced users can generate or tweak chunks by other means.

Arguments

`chunks`	A data frame containing `start`, `end`, and `context` columns, and optionally other columns.
`document`	A `MarkdownDocument`.

Value

An S7 object that inherits from `MarkdownDocumentChunks`, which is also a `tibble`.

See Also

`MarkdownDocument()`

Examples

```
doc_text <- "# A\n\nB\n\n## C\n\nD" # can be readLines() output, etc.
doc <- MarkdownDocument(doc_text, origin = "some/where")
chunk_positions <- tibble::tibble(
  start = c(1L, 9L),
  end = c(8L, 15L),
  context = c("", "# A"),
  text = substring(doc, start, end)
)
chunks <- MarkdownDocumentChunks(chunk_positions, doc)
identical(chunks@document, doc)
```

`mcp_serve_store` *Serve a Ragnar store over MCP*

Description

Launches an MCP server (via `mcptools::mcp_server()`) that exposes a retrieval tool backed by a Ragnar store. This lets MCP-enabled clients (e.g., Codex CLI, Claude Code) call into your store to retrieve relevant excerpts.

Usage

```
mcp_serve_store(
  store,
  store_description = "the knowledge store",
  ...,
  name = NULL,
  title = NULL,
  extra_tools = NULL
)
```

Arguments

`store`	A `RagnarStore` object or a file path to a Ragnar DuckDB store. If a character path is supplied, it is opened with `ragnar_store_connect()`.
`store_description`	Optional string used in the tool description presented to clients.
`...`	arguments passed on to `ragnar_retrieve()`.
`name, title`	Optional tool function name and title. By default, `store@name` and `store@title` will be used if present. The tool `name` must be a valid R function name and should be unique with the tools registered with the ellmer::Chat object. `title` is used for user-friendly display.
`extra_tools`	Optional additional tools (list of `ellmer::tool()` objects) to serve alongside the retrieval tool.

Details

To use this function with Codex CLI, add something like this to `~/.codex/config.toml`

```
[mcp_servers.quartohelp]
command = "Rscript"
args = [
  "-e",
  "ragnar::mcp_serve_store('/path/to/ragnar.store', top_k=10)"
]
```

You can confirm the agent can search the ragnar store by inspecting the output from the `/mcp` command, or by asking it "What tools do you have available?".

Value

This function blocks the current R process by running an MCP server. It is intended for non-interactive use. Called primarily for side-effects.

`ragnar_chunks_view` *View chunks with the store inspector*

Description

Visualize chunks read by `ragnar_read()` for quick inspection. Helpful for inspecting the results of chunking and reading while iterating on the ingestion pipeline.

Usage

```
ragnar_chunks_view(chunks)
```

Arguments

`chunks`	A data frame containing a few chunks.

`ragnar_find_links` *Find links on a page*

Description

Find links on a page

Usage

```
ragnar_find_links(
  x,
  depth = 0L,
  children_only = FALSE,
  progress = TRUE,
  ...,
  url_filter = identity,
  validate = FALSE
)
```

Arguments

x	URL, HTML file path, or XML document. For Markdown, convert to HTML using `commonmark::markdown_html()` first.
depth	Integer specifying how many levels deep to crawl for links. When `depth > 0`, the function will follow child links (links with `x` as a prefix) and collect links from those pages as well.
children_only	Logical or string. If `TRUE`, returns only child links (those having `x` as a prefix). If `FALSE`, returns all links found on the page. Note that regardless of this setting, only child links are followed when `depth > 0`.
progress	Logical, draw a progress bar if `depth > 0`.
...	Currently unused. Must be empty.
url_filter	A function that takes a character vector of URL's and may subset them to return a smaller list. This can be useful for filtering out URL's by rules different than `children_only` which only checks the prefix.
validate	Default is `FALSE`. If `TRUE` sends a `HEAD` request for each link and removes those that are not accessible. Requests are sent in parallel using `httr2::req_perform_parallel()`.

Value

A character vector of links on the page.

Examples

```
## Not run:
ragnar_find_links("https://r4ds.hadley.nz/base-R.html")
ragnar_find_links("https://ellmer.tidyverse.org/")
ragnar_find_links(
  paste0("https://github.com/Snowflake-Labs/sfquickstarts/",
         "tree/master/site/sfguides/src/build_a_custom_model_for_anomaly_detection"),
  children_only = "https://github.com/Snowflake-Labs/sfquickstarts",
  depth = 1
)

## End(Not run)
```

`ragnar_register_tool_retrieve` *Register a 'retrieve' tool with ellmer*

Description

Register a 'retrieve' tool with ellmer

Usage

```
ragnar_register_tool_retrieve(
  chat,
  store,
  store_description = "the knowledge store",
  ...,
  name = NULL,
  title = NULL
)
```

Arguments

`chat`	a `ellmer:::Chat` object.
`store`	a string of a store location, or a `RagnarStore` object.
`store_description`	Optional string, used for composing the tool description.
`...`	arguments passed on to `ragnar_retrieve()`.
`name, title`	Optional tool function name and title. By default, `store@name` and `store@title` will be used if present. The tool `name` must be a valid R function name and should be unique with the tools registered with the ellmer::Chat object. `title` is used for user-friendly display.

Value

`chat`, invisibly.

Examples

```
system_prompt <- stringr::str_squish("
  You are an expert assistant in R programming.
  When responding, you first quote relevant material from books or documentation,
  provide links to the sources, and then add your own context and interpretation.
")
chat <- ellmer::chat_openai(system_prompt, model = "gpt-4.1")

store <- ragnar_store_connect("r4ds.ragnar.duckdb")
ragnar_register_tool_retrieve(chat, store)
chat$chat("How can I subset a dataframe?")
```

ragnar_retrieve *Retrieve chunks from a* RagnarStore

Description

Combines both vss and bm25 search and returns the union of chunks retrieved by both methods.

Usage

```
ragnar_retrieve(store, text, top_k = 3L, ..., deoverlap = TRUE)
```

Arguments

store	A RagnarStore object returned by ragnar_store_connect() or ragnar_store_create().
text	Character. Query string to match.
top_k	Integer. Number of nearest entries to find per method.
...	Additional arguments passed to the lower-level retrieval functions.
deoverlap	Logical. If TRUE (default) and store@version == 2, overlapping chunks are merged with chunks_deoverlap().

Value

A tibble of retrieved chunks. Each row represents a chunk and always contains a text column.

Note

The results are not re-ranked after identifying the unique values.

See Also

Other ragnar_retrieve: ragnar_retrieve_bm25(), ragnar_retrieve_vss(), ragnar_retrieve_vss_and_bm25()

Examples

```
## Build a small store with categories
store <- ragnar_store_create(
  embed = \(x) ragnar::embed_openai(x, model = "text-embedding-3-small"),
  extra_cols = data.frame(category = character()),
  version = 1 # store text chunks directly
)

ragnar_store_insert(
  store,
  data.frame(
    category = c(rep("pets", 3), rep("dessert", 3)),
    text     = c("playful puppy", "sleepy kitten", "curious hamster",
                 "chocolate cake", "strawberry tart", "vanilla ice cream")
  )
```

```
)
ragnar_store_build_index(store)

# Top 3 chunks without filtering
ragnar_retrieve(store, "sweet")

# Combine filter with similarity search
ragnar_retrieve(store, "sweet", filter = category == "dessert")
```

`ragnar_retrieve_bm25` *Retrieves chunks using the BM25 score*

Description

BM25 refers to Okapi Best Matching 25. See doi:10.1561/1500000019 for more information.

Usage

```
ragnar_retrieve_bm25(
  store,
  text,
  top_k = 3L,
  ...,
  k = 1.2,
  b = 0.75,
  conjunctive = FALSE,
  filter
)
```

Arguments

`store`	A `RagnarStore` object returned by `ragnar_store_connect()` or `ragnar_store_create()`.
`text`	String, the text to search for.
`top_k`	Integer. Number of nearest entries to find per method.
`...`	Additional arguments passed to the lower-level retrieval functions.
`k, b`	k_1 and b parameters in the Okapi BM25 retrieval method.
`conjunctive`	Whether to make the query conjunctive i.e., all terms in the query string must be present in order for a chunk to be retrieved.
`filter`	Optional. A filter expression evaluated with `dplyr::filter()`.

Value

A tibble ordered by descending BM25 `metric_value` (higher is more relevant), with a `metric_name` column set to "bm25".

See Also

Other ragnar_retrieve: `ragnar_retrieve()`, `ragnar_retrieve_vss()`, `ragnar_retrieve_vss_and_bm25()`

`ragnar_retrieve_vss` *Vector Similarity Search Retrieval*

Description

Computes a similarity measure between the query and the document embeddings and uses this similarity to rank and retrieve document chunks.

Usage

```
ragnar_retrieve_vss(
  store,
  query,
  top_k = 3L,
  ...,
  method = "cosine_distance",
  query_vector = store@embed(query),
  filter
)
```

Arguments

`store`	A `RagnarStore` object returned by `ragnar_store_connect()` or `ragnar_store_create()`.
`query`	Character. The query string to embed and use for similarity search.
`top_k`	Integer. Maximum number of document chunks to retrieve. Defaults to 3.
`...`	Additional arguments passed to methods.
`method`	Character. Similarity method to use: `"cosine_distance"`, `"euclidean_distance"`, or `"negative_inner_product"`. Defaults to `"cosine_distance"`.
`query_vector`	Numeric vector. The embedding for `query`. Defaults to `store@embed(query)`.
`filter`	Optional. A filter expression evaluated with `dplyr::filter()`.

Details

Supported methods:

- **cosine_distance** – cosine of the angle between two vectors.
- **euclidean_distance** – L2 distance between vectors.
- **negative_inner_product** – negative sum of element-wise products.

If `filter` is supplied, the function first performs the similarity search, then applies the filter in an outer SQL query. It uses the HNSW index when possible and falls back to a sequential scan for large result sets or filtered queries.

Value

A `tibble` with the top_k retrieved chunks, ordered by `metric_value`.

Note

The results are not re-ranked after identifying the unique values.

See Also

Other ragnar_retrieve: `ragnar_retrieve()`, `ragnar_retrieve_bm25()`, `ragnar_retrieve_vss_and_bm25()`

Examples

```
## Build a small store with categories
store <- ragnar_store_create(
  embed = \(x) ragnar::embed_openai(x, model = "text-embedding-3-small"),
  extra_cols = data.frame(category = character()),
  version = 1 # store text chunks directly
)

ragnar_store_insert(
  store,
  data.frame(
    category = c(rep("pets", 3), rep("dessert", 3)),
    text     = c("playful puppy", "sleepy kitten", "curious hamster",
                 "chocolate cake", "strawberry tart", "vanilla ice cream")
  )
)
ragnar_store_build_index(store)

# Top 3 chunks without filtering
ragnar_retrieve(store, "sweet")

# Combine filter with similarity search
ragnar_retrieve(store, "sweet", filter = category == "dessert")
```

---

ragnar_store_atlas *Visualize a store using Embedding Atlas*

Description

Visualize a store using Embedding Atlas

Usage

```
ragnar_store_atlas(
  store,
  ...,
  host = "localhost",
  port = 3030,
  launch.browser = interactive()
)
```

Arguments

store	A RagnarStore object to inspect.
...	Passed to shiny::runApp().
host	Host to run the Embedding Atlas server on.
port	Port to run the Embedding Atlas server on.
launch.browser	Whether to launch the browser automatically.

Note

This function requires the embedding-atlas Python package. Make sure you have it installed in your reticulate Python environment. It also uses arrow to transfer data from the DuckDB store to Python.

Examples

```
## Not run:
# Connect or create a store
store <- ragnar_store_connect(':memory:')
# Launch the Embedding Atlas app
ragnar_store_atlas(store)

## End(Not run)
```

---

ragnar_store_build_index

Build a Ragnar Store index

Description

A search index must be built before calling ragnar_retrieve(). If additional entries are added to the store with ragnar_store_insert(), ragnar_store_build_index() must be called again to rebuild the index.

Usage

```
ragnar_store_build_index(store, type = c("vss", "fts"))
```

Arguments

store	a RagnarStore object
type	The retrieval search type to build an index for.

Value

store, invisibly.

---

ragnar_store_create *Create and connect to a vector store*

Description

Create and connect to a vector store

Usage

```
ragnar_store_create(
  location = ":memory:",
  embed = embed_ollama(),
  ...,
  embedding_size = ncol(embed("foo")),
  overwrite = FALSE,
  extra_cols = NULL,
  name = NULL,
  title = NULL,
  version = 2
)

ragnar_store_connect(location, ..., read_only = TRUE)
```

Arguments

location	filepath, or :memory:. Location can also be a database name specified with md:dbname, in this case the database will be created in MotherDuck after a connection is established.
embed	A function that is called with a character vector and returns a matrix of embeddings. Note this function will be serialized and then deserialized in new R sessions, so it cannot reference to any objects in the global or parent environments. Make sure to namespace all function calls with ::. If additional R objects must be available in the function, you can optionally supply a carrier::crate() with packaged data. It can also be NULL for stores that don't need to embed their texts, for example, if only using FTS algorithms such as ragnar_retrieve_bm25().

`...`	unused; must be empty.
`embedding_size`	integer
`overwrite`	logical, what to do if `location` already exists
`extra_cols`	A zero row data frame used to specify additional columns that should be added to the store. Such columns can be used for adding additional context when retrieving. See the examples for more information. `vctrs::vec_cast()` is used to consistently perform type checks and casts when inserting with `ragnar_store_insert()`.
`name`	A unique name for the store. Must match the `^[a-zA-Z0-9_-]+$` regex. Used by `ragnar_register_tool_retrieve()` for registering tools.
`title`	A title for the store, used by `ragnar_register_tool_retrieve()` when the store is registered with an ellmer::Chat object.
`version`	integer. The version of the store to create. See details.
`read_only`	logical, whether the returned connection can be used to modify the store.

Details

Store versions:

Version 2 – documents with chunk ranges (default)

With `version = 2`, ragnar stores each document once and records the start and end positions of its chunks. This provides strong support for overlapping chunk ranges with de-overlapping at retrieval, and generally allows retrieving arbitrary ranges from source documents, but does not support modifying chunks directly before insertion. Chunks can be augmented via the `context` field and with additional fields passed to `extra_cols`. The easiest way to prepare `chunks` for `version = 2` is with `read_as_markdown()` and `markdown_chunk()`.

Version 1 – flat chunks

With `version = 1`, ragnar keeps all chunks in a single table. This lets you easily modify chunk text before insertion. However, dynamic rechunking (de-overlapping) or extracting arbitrary ranges from source documents is not supported, since the original full documents are no longer available. Chunks can be augmented by modifying the chunk text directly (e.g., with `glue()`). Additionally, if you intend to call `ragnar_store_update()`, it is your responsibility to provide `rlang::hash(original_full_document)` with each chunk. The easiest way to prepare `chunks` for `version = 1` is with `ragnar_read()` and `ragnar_chunk()`.

Value

a `RagnarStore` object

Examples

```
# A store with a dummy embedding
store <- ragnar_store_create(
  embed = \(x) matrix(stats::runif(10), nrow = length(x), ncol = 10),
  version = 1
)
ragnar_store_insert(store, data.frame(text = "hello"))

# A store with a schema. When inserting into this store, users need to
```

```
# provide an `area` column.
store <- ragnar_store_create(
  embed = \(x) matrix(stats::runif(10), nrow = length(x), ncol = 10),
  extra_cols = data.frame(area = character()),
  version = 1
)
ragnar_store_insert(store, data.frame(text = "hello", area = "rag"))

# If you already have a data.frame with chunks that will be inserted into
# the store, you can quickly create a suitable store with `vec_ptype()`:
chunks <- data.frame(text = letters, area = "rag")
store <- ragnar_store_create(
  embed = \(x) matrix(stats::runif(10), nrow = length(x), ncol = 10),
  extra_cols = vctrs::vec_ptype(chunks),
  version = 1
)
ragnar_store_insert(store, chunks)

# version = 2 (the default) has support for deoverlapping
store <- ragnar_store_create(
  # if embed = NULL, then only bm25 search is used (not vss)
  embed = NULL
)
doc <- MarkdownDocument(
  paste0(letters, collapse = ""),
  origin = "/some/where"
)
chunks <- markdown_chunk(doc, target_size = 3, target_overlap = 2 / 3)
chunks$context <- substring(chunks$text, 1, 1)
chunks
ragnar_store_insert(store, chunks)
ragnar_store_build_index(store)

ragnar_retrieve(store, "abc bcd xyz", deoverlap = FALSE)
ragnar_retrieve(store, "abc bcd xyz", deoverlap = TRUE)
```

`ragnar_store_ingest` *Concurrently ingest documents into a Ragnar store*

Description

`ragnar_store_ingest()` distributes document preparation work over multiple processes using mirai. Each worker calls `prepare` on a single path and returns the resulting chunks (and any warnings) to the main process, which then writes them to the store.

Usage

```
ragnar_store_ingest(
  store,
```

```
  paths,
  prepare = function(path) markdown_chunk(read_as_markdown(path)),
  n_workers = NULL,
  progress = TRUE,
  build_index = TRUE
)
```

Arguments

store	A RagnarStore. Currently only version 2 stores are supported.
paths	Character vector of file paths or URLs to ingest.
prepare	Function that converts a single path into a MarkdownDocumentChunks object. It is called with an argument path and should return the prepared chunks (with or without an embedding column).
n_workers	Number of worker processes to use. Defaults to the smaller of length(paths) and parallel::detectCores() (with a minimum of 1).
progress	Logical; if TRUE, show a CLI progress bar.
build_index	Logical; whether to call ragnar_store_build_index() after ingestion.

Value

store, invisibly.

---

ragnar_store_insert *Inserts or updates chunks in a* RagnarStore

---

Description

Inserts or updates chunks in a RagnarStore

Usage

```
ragnar_store_insert(store, chunks)

ragnar_store_update(store, chunks)
```

Arguments

store	a RagnarStore object
chunks	Content to insert or update. The precise input structure depends on store@version. See Details.

Details

Store Version 2

`chunks` must be `MarkdownDocumentChunks` object.

Store Version 1

`chunks` must be a data frame containing `origin`, `hash`, and `text` columns. We first filter out chunks for which `origin` and `hash` are already in the store. If an `origin` is in the store, but with a different `hash`, we replace all of its chunks with the new chunks. Otherwise, a regular insert is performed.

This can help avoid needing to compute embeddings for chunks that are already in the store.

Value

`store`, invisibly.

`ragnar_store_inspect` *Launch the Ragnar Store Inspector*

Description

Launches a Shiny app for interactively browsing a Ragnar store, previewing document chunks, and testing search behavior.

Usage

```
ragnar_store_inspect(store, ...)
```

Arguments

`store`	A `RagnarStore` object to inspect.
`...`	Passed to `shiny::runApp()`.

Details

The Store Inspector is a Shiny app for exploring a `RagnarStore`. Use it to quickly see what was ingested and preview search results for different queries. Type a query in the search bar and choose BM25 or VSS. The list of documents on the left updates, and clicking a row shows its text and metadata on the right. You can drag the divider to resize the document list and preview area.

The preview area shows the chunk content. You can view it as rendered Markdown or switch to "Raw Text" to see the stored text (long lines are wrapped). Metadata is shown above the text in YAML format, including any extra fields stored with the chunk.

Value

`NULL` (invisibly).

Keyboard Shortcuts

Context	*Shortcut*	*Action*
Global	`/`, `Esc`	Focus search; clear it
Documents list	`ArrowUp`/`ArrowDown`, `j`/`k`	Move selection
Vertical Divider	`ArrowLeft`/`ArrowRight` (`+Shift`), `g`/`Home`	Resize; reset

`read_as_markdown` *Convert files to Markdown*

Description

Convert files to Markdown

Usage

```
read_as_markdown(
  path,
  ...,
  origin = path,
  html_extract_selectors = c("main"),
  html_zap_selectors = c("nav"),
  youtube_transcript_formatter = NULL
)
```

Arguments

`path`	[string] A filepath or URL. Accepts a wide variety of file types, including plain text (markdown), PDF, PowerPoint, Word, Excel, images (EXIF metadata and OCR), audio (EXIF metadata and speech transcription), HTML, text-based formats (CSV, JSON, XML), ZIP files (iterates over contents), YouTube URLs, and EPUBs.
`...`	Passed on to `MarkItDown.convert()`.
`origin`	The value to use for the `@origin` property of the returned `MarkdownDocument`.
`html_extract_selectors`	Character vector of CSS selectors. If a match for a selector is found in the document, only the matched node's contents are converted. Unmatched extract selectors have no effect.
`html_zap_selectors`	Character vector of CSS selectors. Elements matching these selectors will be excluded ("zapped") from the HTML document before conversion to markdown. This is useful for removing navigation bars, sidebars, headers, footers, or other unwanted elements. By default, navigation elements (`nav`) are excluded.

youtube_transcript_formatter
: A function used to customize how YouTube transcript data is converted to markdown. It receives a tibble/data.frame with columns `text` (chr), `start` (dbl, seconds), and `duration` (dbl, seconds), along with a `"youtube_metadata"` attribute, a named list containing elements `language`, `language_code`, `video_id`, and `is_generated`. The formatter must return a single string; by default it behaves like `\(transcript) paste0(transcript$text, collapse = " ")`. Provide a custom formatter to include timestamps or links (see examples).

Details

Converting HTML:

When converting HTML, you might want to omit certain elements, like sidebars, headers, footers, etc. You can pass CSS selector strings to either extract nodes or exclude nodes during conversion.

The easiest way to make selectors is to use SelectorGadget: https://rvest.tidyverse.org/articles/selectorgadget.html

You can also right-click on a page and select "Inspect Element" in a browser to better understand an HTML page's structure.

For comprehensive or advanced usage of CSS selectors, consult https://www.crummy.com/software/BeautifulSoup/bs4/doc/#css-selectors-through-the-css-property and https://facelessuser.github.io/soupsieve/selectors/

Value

A MarkdownDocument object, which is a single string of Markdown with an `@origin` property.

Examples

```
## Not run:
# Convert HTML
md <- read_as_markdown("https://r4ds.hadley.nz/base-R.html")
md

cat_head <- \(md, n = 10) writeLines(head(strsplit(md, "\n")[[1L]], n))
cat_head(md)

## Using selector strings

# By default, this output includes the sidebar and other navigational elements
url <- "https://duckdb.org/code_of_conduct"
read_as_markdown(url) |> cat_head(15)

# To extract just the main content, use a selector
read_as_markdown(url, html_extract_selectors = "#main_content_wrap") |>
  cat_head()

# Alternative approach: zap unwanted nodes
read_as_markdown(
  url,
  html_zap_selectors = c(
    "header",          # name
```

```
    ".sidenavigation", # class
    ".searchoverlay",  # class
    "#sidebar"         # ID
  )
) |> cat_head()

# Quarto example
read_as_markdown(
  "https://quarto.org/docs/computations/python.html",
  html_extract_selectors = "main",
  html_zap_selectors = c(
    "#quarto-sidebar",
    "#quarto-margin-sidebar",
    "header",
    "footer",
    "nav"
  )
) |> cat_head()

## Convert PDF
pdf <- file.path(R.home("doc"), "NEWS.pdf")
read_as_markdown(pdf) |> cat_head(15)
## Alternative:
# pdftools::pdf_text(pdf) |> cat_head()

# Convert images to markdown descriptions using OpenAI
jpg <- file.path(R.home("doc"), "html", "logo.jpg")
if (Sys.getenv("OPENAI_API_KEY") != "") {
  # if (xfun::is_macos()) system("brew install ffmpeg")
  reticulate::py_require("openai")
  llm_client <- reticulate::import("openai")$OpenAI()
  read_as_markdown(jpg, llm_client = llm_client, llm_model = "gpt-4.1-mini") |>
    writeLines()
  # # Description:
  # The image displays the logo of the R programming language. It features a
  # large, stylized capital letter "R" in blue, positioned prominently in the
  # center. Surrounding the "R" is a gray oval shape that is open on the right
  # side, creating a dynamic and modern appearance. The R logo is commonly
  # associated with statistical computing, data analysis, and graphical
  # representation in various scientific and professional fields.
}

# Alternative approach to image conversion:
if (
  Sys.getenv("OPENAI_API_KEY") != "" &&
    rlang::is_installed("ellmer") &&
    rlang::is_installed("magick")
) {
  chat <- ellmer::chat_openai(echo = TRUE)
  chat$chat("Describe this image", ellmer::content_image_file(jpg))
}

# YouTube transcripts
```

```
## read_as_markdown() fetches transcripts for YouTube links
cat_head(read_as_markdown("https://youtu.be/GELhdezYmP0"))

## The default transcript omits timestamps. Supply a custom
## `youtube_transcript_formatter` to control the output. This example formats
## the transcript with timestamped YouTube links.

format_youtube_timestamp <- function(time) {
  h <- time %/% 3600
  time <- time %% 3600
  m <- time %/% 60
  time <- time %% 60
  s <- floor(time)
  out <- paste0(h, "h", m, "m", s, "s")
  out <- sub("^0h", "", out)
  out <- sub("^0m", "", out)
  out
}

format_transcript_with_timestamps <-
  function(data, min_timestamp_stride_seconds = 30, links = FALSE) {
    ts <- format_youtube_timestamp(data$start)
    if (links) {
      video_id <- attr(data, "youtube_metadata")$video_id
      ts <- sprintf("\n<https://youtu.be/%s?t=%s>\n", video_id, ts)
    } else {
      ts <- sprintf("\n[%s] ", ts)
    }

    if (!is.null(min_timestamp_stride_seconds)) {
      show <- c(TRUE, as.logical(diff(x %/% min_timestamp_stride_seconds)))
      ts[!show] <- ""
    }

    paste0(ts, data$text, sep = "", collapse = "\n")
  }


read_as_markdown(
  "https://www.youtube.com/watch?v=GELhdezYmP0",
  youtube_transcript_formatter = \(data) {
    format_transcript_with_timestamps(data, links = TRUE)
  }
) |>
  cat_head(n = 60)

## End(Not run)
```

Index