Package 'RAGFlowChainR'

February 22, 2026

Type Package

Title Retrieval-Augmented Generation (RAG) Workflows in R with Local and Web Search

Version 0.1.6

Maintainer Kwadwo Daddy Nyame Owusu Boakye <kwadwo.owusuboakye@outlook.com>

Description Enables Retrieval-Augmented Generation (RAG) workflows in R by combining local vector search using 'DuckDB' with optional web search via the 'Tavily' API. Supports 'OpenAI'- and 'Ollama'-compatible embedding models, full-text and 'HNSW' (Hierarchical Navigable Small World) indexing, and modular large language model (LLM) invocation. Designed for advanced question-answering, chat-based applications, and production-ready AI pipelines. This package is the R equivalent of the 'python' package 'RAGFlowChain' available at <https://pypi.org/project/RAGFlowChain/>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

URL https://github.com/knowusuboaky/RAGFlowChainR, https://knowusuboaky.github.io/RAGFlowChainR/

BugReports https://github.com/knowusuboaky/RAGFlowChainR/issues

Depends R (>= 4.1.0)

Imports DBI, duckdb (>= 0.10.0), httr, dplyr, pdftools, officer, rvest, xml2, curl,

Suggests testthat (>= 3.0.0), jsonlite, stringi, magrittr, roxygen2, knitr, rmarkdown, VectrixDB

Config/testthat/edition 3

NeedsCompilation no

Author Kwadwo Daddy Nyame Owusu Boakye [aut, cre]

Repository CRAN

Date/Publication 2026-02-22 09:50:02 UTC

Contents

`create_rag_chain` *create_rag_chain.R Overview*

Description

A refined implementation of a LangChain-style Retrieval-Augmented Generation (RAG) pipeline. Includes vector search across multiple backends, optional web search using the Tavily API, and a built-in chat message history.

This function powers ‘create_rag_chain()‘, the exported entry point for constructing a full RAG pipeline.

Features: - Context-aware reformulation of user queries - Semantic chunk retrieval using DuckDB, VectrixDB, Qdrant, Pinecone, Weaviate, or Elasticsearch - Optional real-time web search (Tavily) - Compatible with any LLM function (OpenAI, Claude, etc.)

Required Packages `install.packages(c("DBI", "duckdb", "httr", "jsonlite", "stringi", "dplyr"))`

Arguments

`llm`	A function that takes a prompt and returns a response (e.g. a call to OpenAI or Claude).
`vector_database_directory`	Path to the vector backend. For ‘method = "DuckDB"‘, pass a DuckDB database file path. For ‘method = "VectrixDB"‘, pass a VectrixDB collection path/root path or collection name. For ‘method = "Qdrant"‘, pass ‘"https://host:6333|collection_name"‘. For ‘method = "Pinecone"‘, pass ‘"https://index-host|namespace"‘ (namespace optional). For ‘method = "Weaviate"‘, pass ‘"https://weaviate-host|ClassName"‘. For ‘method = "Elasticsearch"‘, pass ‘"https://elastic-host:9200|index_name|vector_field"‘ (vector field optional).
`method`	Retrieval backend. One of ‘"DuckDB"‘, ‘"VectrixDB"‘, ‘"Qdrant"‘, ‘"Pinecone"‘, ‘"Weaviate"‘, or ‘"Elasticsearch"‘.
`embedding_function`	A function to embed text. Defaults to `embed_openai()`.
`system_prompt`	Optional prompt with placeholders `{chat_history}`, `{input}`, `{context}`.
`chat_history_prompt`	Prompt used to rephrase follow-up questions using prior conversation history.
`tavily_search`	Tavily API key (set to `NULL` to disable web search).
`embedding_dim`	Integer; embedding vector dimension. Defaults to `1536`.
`use_web_search`	Logical; whether to include web results from Tavily. Defaults to `TRUE`.

Details

Create a Retrieval-Augmented Generation (RAG) Chain

Creates a LangChain-style RAG chain using DuckDB for vector store operations, optional Tavily API for web search, and in-memory message history for conversational context.

Value

A list of utility functions:

- `invoke(text)` — Performs full context retrieval and LLM response
- `custom_invoke(text)` — Retrieves context only (no LLM call)
- `get_session_history()` — Returns complete conversation history
- `clear_history()` — Clears in-memory chat history
- `disconnect()` — Closes any open local backend connection

Note

Only `create_rag_chain()` is exported. Helper functions are internal.

Examples

```
## Not run:
rag_chain <- create_rag_chain(
  llm = call_llm,
  vector_database_directory = "tests/testthat/test-data/my_vectors.duckdb",
  method = "DuckDB",
  embedding_function = embed_openai(),
  use_web_search = FALSE
)

response <- rag_chain$invoke("Tell me about R")

## End(Not run)
```

---

embed_openai *Embed text with OpenAI*

Description

Helper for vector-store pipelines. If called without ‘x‘, this returns a closure that can be passed directly to ‘insert_vectors(embed_fun = ...)‘.

Initializes a DuckDB database connection for storing embedded documents, with optional support for the experimental ‘vss‘ extension.

Chunks long text rows, generates embeddings when needed, and inserts ‘(page_content, embedding)‘ rows into the ‘vectors‘ table.

Builds HNSW (‘vss‘) and/or full-text (‘fts‘) indexes on the ‘vectors‘ table.

Embeds ‘query_text‘, computes vector distance against stored embeddings, and returns the nearest matches.

Usage

```
embed_openai(
  x,
  model = "text-embedding-ada-002",
  base_url = "https://api.openai.com/v1",
  api_key = Sys.getenv("OPENAI_API_KEY"),
  batch_size = 20L,
  embedding_dim = 1536
)

create_vectorstore(
  db_path = ":memory:",
  overwrite = FALSE,
  embedding_dim = 1536,
  load_vss = identical(Sys.getenv("_R_CHECK_PACKAGE_NAME_"), "")
)

insert_vectors(
  con,
  df,
  embed_fun = embed_openai(),
  chunk_chars = 12000,
  embedding_dim = 1536
)

build_vector_index(store, type = c("vss", "fts"))

search_vectors(
  con,
  query_text,
  top_k = 5,
  embed_fun = embed_openai(),
  embedding_dim = 1536
)
```

Arguments

x	Character vector of texts, or a data frame with a ‘page_content‘ column.
model	OpenAI embedding model name.
base_url	Base URL for an OpenAI-compatible API.
api_key	API key; defaults to ‘Sys.getenv("OPENAI_API_KEY")‘.
batch_size	Batch size for embedding requests.

embedding_dim	Integer; the dimensionality of the vector embeddings to store.
db_path	Path to the DuckDB file. Use '":memory:"' to create an in-memory database.
overwrite	Logical; if 'TRUE', deletes any existing DuckDB file or table.
load_vss	Logical; whether to load the experimental 'vss' extension. This defaults to 'TRUE', but is forced to 'FALSE' during CRAN checks.
con	Active DuckDB DBI connection.
df	Data frame containing 'page_content' (or 'content') text.
embed_fun	Function used to convert text into numeric embeddings.
chunk_chars	Approximate max chunk size in bytes before splitting.
store	Active DuckDB DBI connection or vector-store handle.
type	Index types to build; any of '"vss"' and/or '"fts"'.
query_text	Query text to embed and search.
top_k	Number of nearest matches to return.

Details

This function is part of the vector-store utilities for:

- Embedding text via the OpenAI API
- Storing and chunking documents in DuckDB
- Building 'HNSW' and 'FTS' indexes
- Running nearest-neighbour search over vector embeddings

Core helpers like `embed_openai()`, `insert_vectors()`, `build_vector_index()`, and `search_vectors()` are also exported to support composable workflows.

Value

For character input, a numeric matrix of embeddings. For data-frame input, the same data frame with an added 'embedding' column. If 'x' is missing, a configured embedding function is returned.

A live DuckDB connection object. Be sure to manually disconnect with: `DBI::dbDisconnect(con, shutdown = TRUE)`

Examples

```
## Not run:
# Create vector store
con <- create_vectorstore("tests/testthat/test-data/my_vectors.duckdb", overwrite = TRUE)

# Assume response is output from fetch_data()
docs <- data.frame(head(response))

# Insert documents with embeddings
insert_vectors(
  con = con,
  df = docs,
```

```
  embed_fun = embed_openai(),
  chunk_chars = 12000
)

# Build vector + FTS indexes
build_vector_index(con, type = c("vss", "fts"))

# Perform vector search
response <- search_vectors(con, query_text = "Tell me about R?", top_k = 5)

## End(Not run)
```

`fetch_data` *Fetch data from local files and websites*

Description

Extracts content and metadata from local documents or websites. Supports:

- Local files: PDF, DOCX, PPTX, TXT, HTML
- Crawled websites: with optional breadth-first crawl depth

Arguments

`local_paths`	A character vector of file paths or directories to scan for documents.
`website_urls`	A character vector of website URLs to crawl and extract text from.
`crawl_depth`	Integer indicating BFS crawl depth; use `NULL` for unlimited depth.

Details

The returned data frame includes structured columns such as: `source`, `title`, `author`, `publishedDate`, `description`, `content`, `url`, and `source_type`.

Required Packages `install.packages(c("pdftools", "officer", "rvest", "xml2", "dplyr", "stringi", "curl", "httr", "jsonlite", "magrittr"))`

Value

A data frame with extracted metadata and content.

Note

Internal functions used include `read_local_file()`, `read_website_page()`, and `crawl_links_bfs()`.

Examples

```
## Not run:
local_files <- c("tests/testthat/test-data/sprint.pdf",
                 "tests/testthat/test-data/introduction.pptx",
                 "tests/testthat/test-data/overview.txt")
website_urls <- c("https://www.r-project.org")
crawl_depth <- 1

response <- fetch_data(
  local_paths = local_files,
  website_urls = website_urls,
  crawl_depth = crawl_depth
)

## End(Not run)
```

Index