Package ‘RAGFlowChainR’

April 24, 2025

Type Package

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

Version 0.1.1

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

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

Config/testthat/edition 3

NeedsCompilation no

Author Kwadwo Daddy Nyame Owusu Boakye [aut, cre]

Repository CRAN

Date/Publication 2025-04-24 11:40:27 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 using DuckDB, optional web search using the Tavily API, and a built-in chat message history.

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

Features: - Context-aware reformulation of user questions based on chat history - Retrieval of relevant chunks via semantic search - Optional real-time web search using Tavily (if API key is set) - Works with any LLM function (e.g., OpenAI, Claude)

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

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.

Usage

```
create_rag_chain(
  llm,
  vector_database_directory,
  method = "DuckDB",
  embedding_function = NULL,
  system_prompt = NULL,
  chat_history_prompt = NULL,
  tavily_search = NULL,
  embedding_dim = 1536,
  use_web_search = TRUE
)
```

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 DuckDB database file.
`method`	Currently only "DuckDB" is supported.

`embedding_function`	A function for embedding text. Defaults to ‘embed_openai()‘.
`system_prompt`	Optional prompt with placeholders `{chat_history}`, `{input}`, `{context}`
`chat_history_prompt`	Prompt used to rephrase user questions based on prior context.
`tavily_search`	API key for Tavily (or NULL to disable web search).
`embedding_dim`	Dimensionality of embedding vectors (default 1536).
`use_web_search`	Logical, whether to include web results from Tavily (default TRUE).

Value

A list of utility functions:

- `invoke(text)` — Performs full context retrieval + LLM response
- `custom_invoke(text)` — Retrieves context only, no LLM response
- `get_session_history()` — Returns full chat history
- `clear_history()` — Clears the chat memory
- `disconnect()` — Closes DuckDB connection

Note

Only ‘create_rag_chain()‘ is exported.

`create_vectorstore` *create_vectorstore.R — Vector-store utilities*

Description

Tools to • embed text with the OpenAI API • create a DuckDB-backed vector store (optionally with the ‘vss‘ extension) • insert documents with embeddings (handles chunking) • build HNSW/FTS indexes and run nearest-neighbour search

Usage

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

Arguments

`db_path`	Path to the DuckDB file (\"‘:memory:‘\" for RAM).
`overwrite`	If ‘TRUE‘, delete any existing file / table.
`embedding_dim`	Dimension of the embeddings stored.
`load_vss`	Try to load the experimental ‘vss‘ extension? Defaults to ‘TRUE‘ except during CRAN checks where it is forced ‘FALSE‘.

Details

Only ‘create_vectorstore()‘ is exported; all other helpers are internal.

Value

A live ‘duckdb_connection‘. Disconnect manually with ‘DBI::dbDisconnect(con, shutdown = TRUE)‘.

`data_fetcher` *data_fetcher.R Overview*

Description

Provides the ‘fetch_data()‘ function, which extracts and structures content from:

- Local files (PDF, DOCX, PPTX, TXT, HTML)
- Crawled websites (with optional BFS crawl depth)

The returned data frame includes metadata columns like ‘title‘, ‘author‘, ‘publishedDate‘, and the main extracted ‘content‘.

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

Note

Only ‘fetch_data()‘ is exported. Internal functions include ‘read_local_file()‘, ‘read_website_page()‘, and ‘crawl_links_bfs()‘.

fetch_data *Fetch Data from Local Files and Websites*

Description

Extracts content and metadata from local documents or websites. Supports PDF, DOCX, PPTX, TXT, HTML files and performs BFS web crawling up to the specified depth.

Usage

```
fetch_data(local_paths = NULL, website_urls = NULL, crawl_depth = NULL)
```

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; set to NULL for infinite crawl.

Value

A data frame with the following columns: source, title, author, publishedDate, description, content, url, source_type.

Index