RAG System Documentation

Version: 0.6

Table of Contents

---

1. Introduction

This document provides an in-depth, function-by-function description of the RAG system—a modular framework designed for efficient retrieval and generation of information from documents. Inspired by documentation styles such as the MENWIZ 1.2 Quick Tour, it begins with an overview of the general principles and key terminology used throughout the system:

- **Chunking:** The process of splitting long texts into smaller, manageable segments while preserving semantic integrity.
- **Embedding:** The transformation of text into high-dimensional vectors using pre-trained models (e.g., SentenceTransformer) to capture semantic meaning.
- **Indexing:** The creation and storage of vector indices (via FAISS, BM25, and TF-IDF) that allow for efficient retrieval.
- **Retrieval Strategies:** A combination of dense and sparse search methods that support various modes of querying (e.g., FAISS for dense retrieval, BM25 for token-based matching, hybrid approaches, and fusion strategies like Reciprocal Rank Fusion).

The system is built with a modular architecture that clearly separates concerns—from initialization and configuration to document management, embedding generation, and search. Each module is documented below with details on its functions, parameters, return values, and, where applicable, the allowed option values.

---

2. Module Documentation

2.1. init.py

Purpose:
Initializes the system environment and loads the key modules required for the RAG system.

Key Actions:

- **Environment Configuration:**
    - Checks if the environment variable TRANSFORMERS_CACHE exists; if it does, it removes it.
    - Dynamically sets HF_HOME (used for model storage) to a "models" directory relative to the current file location.
- **Module Imports:**
    - Imports core modules such as VectorStorage, DocumentManager, EmbeddingManager, SearchEngine, dynamic_chunk_text, compute_md5, and Config.
- **Initialization:**
    - Sets the system version (e.g., __version__ = "0.6").
    - Invokes Config.load() to load or initialize the system configuration.

Interface:
This module does not expose callable functions for external use; it serves as the entry point for configuration and initialization.

2.2. chunker.py

Purpose:
Provides functionality for splitting long texts into smaller, semantically coherent chunks.

Functions:

- **dynamic_chunk_text(text, chunk_size=100, overlap=20, min_chunks=3)**
  - **Parameters:**
    - text (str): The full text to be split.
    - chunk_size (int, default=100): Approximate number of words per chunk.
    - overlap (int, default=20): Number of overlapping words between consecutive chunks.
    - min_chunks (int, default=3): Minimum number of chunks desired; if not met, a fallback method is used.
  - **Behavior:**
    - Uses NLTK's sent_tokenize to break text into sentences.
    - Aggregates sentences until the approximate word count (chunk_size) is reached.
    - If fewer than min_chunks are produced, falls back to a word-based splitting strategy.
  - **Returns:**
    - A list of text chunks (each as a string).
- **fallback_chunk_by_words(words, chunk_size, overlap)**
  - **Parameters:**
    - words (list of str): The text split into words.
    - chunk_size (int): Number of words per chunk.
    - overlap (int): Overlap between chunks.
  - **Behavior:**
    - Splits the word list into overlapping segments.
  - **Returns:**
    - A list of chunks created solely based on word count.

2.3. config.py

Purpose:
Handles system configuration by storing, loading, and saving parameters necessary for operation.

Class: Config

- **Attributes:**
    - BASE_DIR: Directory for persistent stores.
    - EMBEDDING_MODEL_KEY: Key indicating the model to use (e.g., "MiniLM", "MPNet", "DistilRoBERTa").
    - CHUNK_SIZE, OVERLAP: Parameters used in text chunking.
    - CROSS_ENCODER_MODEL: Model name used for cross-encoding during re-ranking.
    - DEFAULT_TOP_K, DEFAULT_THRESHOLD, MIN_CHUNKS, DEFAULT_RETRIEVAL_MODE: Various parameters for search and retrieval strategies.
    - UI_LAST_STORE: Stores the last used storage location.
- **Allowed Options:**
    - For EMBEDDING_MODEL_KEY, the recognized values include:
        - **"MiniLM"** – maps to "all-MiniLM-L6-v2".
        - **"MPNet"** – maps to "all-mpnet-base-v2".
        - **"DistilRoBERTa"** – maps to "all-distilroberta-v1".
        - **“multilingual-MiniLM-L12-v2”** - maps to “paraphrase-multilingual-MiniLM-L12-v2
    - (Additional model keys may be used as long as they correspond to a valid SentenceTransformer model.)
- **Methods:**
    - **load(cls)**
        - **Behavior:**
            - Checks if a configuration file (rag_system.json) exists.

        - If found, loads parameters and updates class attributes.
        - If not, creates a new configuration file using default settings by calling save().
    - **Returns:**
        - None.
- **save(cls)**
    - **Behavior:**
        - Serializes current configuration attributes to a JSON file (rag_system.json).
    - **Returns:**
        - None.

2.4. document_manager.py

Purpose:
Manages the loading and storage of documents from various file formats, ensuring content normalization and duplicate detection.

Functions:

- **normalize_value(value)**
  - **Parameters:**
    - value (any): The value to be normalized.
  - **Behavior:**
    - Converts the value to a string (if not null), trims whitespace, and converts it to lowercase.
  - **Returns:**
    - A normalized string.
- **load_document(file_path)**
  - **Parameters:**
    - file_path (str): Path to the document file.
  - **Behavior:**
    - Determines the file type by extension.
    - Supports .txt, .md, .pdf, .docx, .xls, and .xlsx.
    - Uses appropriate libraries (e.g., PyPDF2 for PDFs, python-docx for DOCX) to extract text.
    - For Excel files, reads all sheets, cleans the data, and aggregates rows into chunks.
  - **Returns:**
    - The content of the document as a single string.
  - **Errors:**
    - Raises exceptions if the file format is unsupported or if decoding fails.

Class: DocumentManager

- **Attributes:**

- documents: A dictionary mapping document IDs to their content and metadata.
- document_hashes: Used to track MD5 hashes for duplicate detection.

- **Methods:**
  - **add_document(doc_id, content, metadata=None, file_path=None)**
    - **Parameters:**
      - doc_id (str): Unique identifier for the document.
      - content (str): The text content of the document.
      - metadata (dict, optional): Additional metadata such as source or type.
      - file_path (str, optional): Path of the file (used to infer metadata).
    - **Behavior:**
      - Adds the document to internal storage, automatically augmenting metadata if needed.
    - **Returns:**
      - None.
  - **delete_document(doc_id)**
    - **Parameters:**
      - doc_id (str): Identifier of the document to be removed.
    - **Behavior:**
      - Removes the document from internal storage.
    - **Returns:**
      - Boolean indicating success.
  - **list_documents()**
    - **Behavior:**
      - Retrieves a list of all document IDs currently managed.
    - **Returns:**
      - A list of document IDs.
  - **load_documents_from_directory(directory_path)**
    - **Parameters:**

        - directory_path (str): Directory to scan for documents.
    - **Behavior:**
        - Traverses the directory, loading and processing each file.
    - **Returns:**
        - Aggregated documents and metadata.

2.5. embedding_manager.py

Purpose:
Generates dense vector embeddings for text segments using SentenceTransformer.
For long texts, it leverages the chunker module to split the text before embedding.

Class: EmbeddingManager

- **Attributes:**
    - **EMBEDDER_OPTIONS**:
        - **Allowed keys and their mappings:**
            - **"MiniLM"** → "all-MiniLM-L6-v2": Lightweight and fast.
            - **"MPNet"** → "all-mpnet-base-v2": Offers robust semantic representations.
            - **"DistilRoBERTa"** → "all-distilroberta-v1": A balance between speed and performance.
            - **"multilingual-MiniLM-L12-v2"** → "paraphrase-multilingual-MiniLM-L12-v2": Supports multiple languages.[1]
    - model: The loaded SentenceTransformer model.
    - chunk_size, overlap: Parameters for text chunking.
- **Constructor: __init__(model_key="MiniLM", chunk_size=100, overlap=20)**
    - **Parameters:**
        - model_key (str): Recognized options are listed above. Alternatively, a full model name may be provided directly.[2]
        - chunk_size (int): Sets the maximum word count per chunk.
        - overlap (int): Sets the number of overlapping words between consecutive chunks.
    - **Behavior:**

[2] *Note: If a full model name (e.g., one of the mapped values) is provided directly to the EmbeddingManager, it will be used as is.*

        - Loads the appropriate embedding model based on model_key (using the mapping in EMBEDDER_OPTIONS).
        - Sets chunking parameters.

- **Methods:**
    - **embed_text(text, doc_name=None)**
        - **Parameters:**
            - text (str): The text to embed.
            - doc_name (str, optional): Source document name.
        - **Behavior:**
            - If the text starts with a specific marker (e.g., "[EXCEL]"), it processes each line as a separate chunk.
            - For texts with fewer than ~50 words, returns a single chunk.
            - Otherwise, uses dynamic_chunk_text to split the text.
            - Computes embeddings for each chunk.
            - Normalizes embeddings (L2 normalization) for cosine similarity.
            - If doc_name is provided, returns each chunk as a dictionary with its source.
        - **Returns:**
            - A tuple: (chunks, embeddings).
    - **embed_query(query)**
        - **Parameters:**
            - query (str): The query text.
        - **Behavior:**
            - Computes a normalized embedding vector for the query.
        - **Returns:**
            - A normalized embedding vector (1D numpy array).
    - **set_chunk_params(chunk_size, overlap)**
        - **Parameters:**
            - chunk_size (int): New chunk size.

        - overlap (int): New overlap count.
    - **Behavior:**
        - Updates chunking parameters at runtime.
    - **Returns:**
        - None.

2.6. search_engine.py

Purpose:
Implements multiple search strategies that combine dense (FAISS) and sparse (BM25, TF-IDF) retrieval methods, along with fusion techniques.

Class: SearchEngine

- **Constructor: __init__(vector_storage, embedding_manager, cross_encoder=None, weight_dense=0.4, weight_bm25=0.3, weight_tfidf=0.3)**
    - **Parameters:**
        - vector_storage: Instance managing vector indices.
        - embedding_manager: For generating embeddings.
        - cross_encoder (optional): Model for re-ranking search results.
        - weight_dense, weight_bm25, weight_tfidf: Weights for combining different retrieval signals.
    - **Behavior:**
        - Initializes the search engine with the specified components and weight parameters.
- **Allowed Search Strategy Options:**
    - **"faiss"** – Dense retrieval using the FAISS index.
    - **"bm25"** – Token-based sparse retrieval using BM25.
    - **"ibrido"** or **"hybrid"** – Combines FAISS and BM25 scores using predefined weights.
    - **"multi"** – Integrates FAISS, BM25, and TF-IDF representations with weighted fusion.
    - **"rrf"** – Uses Reciprocal Rank Fusion to combine rankings.
- **Allowed Retrieval Modes:**
    - **"chunk"** – Returns results at the text chunk level.
    - **"document"** – Aggregates results by document.
- **Methods:**

- **search_with_strategy(query, strategy="faiss", top_k=10, retrieval_mode="chunk", threshold=0.0, use_knee_detection=False, use_mmr=False, mmr_lambda=0.5, max_context_bytes=None)**
    - **Parameters:**
        - query (str): The search query.
        - strategy (str): Must be one of the allowed strategies ("faiss", "bm25", "ibrido"/"hybrid", "multi", "rrf").
        - top_k (int): Maximum number of results.
        - retrieval_mode (str): Either "chunk" or "document".
        - threshold (float): Minimum score threshold.
        - use_knee_detection (bool): If True, applies knee detection to determine a cutoff.
        - use_mmr (bool): If True, applies Maximal Marginal Relevance for result diversification.
        - mmr_lambda (float): Balancing parameter for MMR.
        - max_context_bytes (int, optional): Limit for the combined text context size.
    - **Behavior:**
        - Selects the appropriate search method based on strategy.
        - Aggregates and optionally re-ranks results using the specified options.
    - **Returns:**
        - A list of result dictionaries (with keys like doc_id, score, text, etc.).
- **search_faiss(query, top_k, retrieval_mode)**
    - **Behavior:**
        - Generates a dense query vector using the embedding manager.
        - Searches the FAISS index.
        - In "document" mode, aggregates chunk-level results.
        - Optionally re-ranks with a cross-encoder.
    - **Returns:**
        - A list of search result dictionaries.

- **search_bm25(query, top_k, retrieval_mode)**
  - **Behavior:**
    - Uses BM25 for token-based retrieval.
    - Aggregates results in "document" mode if necessary.
    - Optionally applies cross-encoder re-ranking.
  - **Returns:**
    - A list of result dictionaries.
- **search_hybrid(query, top_k, retrieval_mode)**
  - **Behavior:**
    - Combines FAISS (dense) and BM25 (sparse) scores using the weights weight_dense and weight_bm25.
    - Aggregates and optionally re-ranks results.
  - **Returns:**
    - A list of combined search results.
- **search_multi_representation(query, top_k, retrieval_mode)**
  - **Behavior:**
    - Merges dense, BM25, and TF-IDF signals.
    - Normalizes and fuses scores using a weighted sum.
    - Optionally applies cross-encoder re-ranking.
  - **Returns:**
    - A list of search results.
- **search_rrf(query, top_k, retrieval_mode)**
  - **Behavior:**
    - Implements Reciprocal Rank Fusion (RRF) by combining ranking lists from FAISS and BM25, optionally with TF-IDF.
  - **Returns:**
    - A fused list of search results.
- **Helper Function: detect_knee(scores)**
  - **Parameters:**

        - scores (list of float): An ordered list of scores.
    - **Behavior:**
        - Identifies the "knee" (largest drop) in the score distribution to determine a cutoff.
    - **Returns:**
        - A cutoff score (float).

2.7. utils.py

Purpose:
Contains utility functions for hashing, which are used for document integrity and duplicate detection.

Functions:

- **compute_md5(content)**
    - **Parameters:**
        - content (str): Input string.
    - **Behavior:**
        - Computes the MD5 hash of the given string using UTF-8 encoding.
    - **Returns:**
        - A hexadecimal MD5 hash string.
- **compute_file_md5(file_path)**
    - **Parameters:**
        - file_path (str): Path to the file.
    - **Behavior:**
        - Reads the file in binary mode and computes its MD5 hash.
    - **Returns:**
        - A hexadecimal MD5 hash string representing the file's content.

2.8. vector_storage.py

Purpose:
Manages the persistent storage of vector indices for efficient retrieval. This includes FAISS (for dense search), BM25 (for sparse, token-based search), and TF-IDF.

Class: VectorStorage

- **Constructor: __init__(embedding_dim, embedder="all-MiniLM-L6-v2", indices_present=["faiss", "bm25", "tfidf"], chunk_size=100, overlap=20)**
    - **Parameters:**

        - embedding_dim (int): Dimensionality of the embedding vectors (e.g., 384, 768).
        - embedder (str): Name of the embedding model used (allowed values are the full model names as mapped in EmbeddingManager such as "all-MiniLM-L6-v2", "all-mpnet-base-v2", etc.).
        - indices_present (list of str): Which indices to build and maintain. Allowed values include:
            - **"faiss"** – for dense vector search.
            - **"bm25"** – for BM25 token-based search.
            - **"tfidf"** – for sparse TF-IDF-based retrieval.
        - chunk_size, overlap (int): Parameters for chunking.
    - **Behavior:**
        - Initializes the FAISS index using inner product with L2 normalization.
        - Sets up structures for BM25 (corpus and document IDs) and TF-IDF (vectorizer and index).
        - Prepares internal storage for documents, chunk data, and MD5 signatures.
- **Methods:**
    - **set_chunk_params(chunk_size, overlap)**
        - **Behavior:**
            - Updates chunking parameters.
        - **Returns:**
            - None.
    - **add_vector(vector, doc_id, chunk_text, source=None)**
        - **Parameters:**
            - vector: The embedding vector (as a numpy array).
            - doc_id (str): Identifier of the source document.
            - chunk_text (str): Text of the chunk.
            - source (str, optional): Additional source information.
        - **Behavior:**

        - Adds the vector to the FAISS index.
        - Stores chunk metadata for retrieval.
    - **Returns:**
        - None.
- **remove_document(doc_id, embedding_manager)**
    - **Parameters:**
        - doc_id (str): Identifier of the document to remove.
        - embedding_manager: Instance used to rebuild indices.
    - **Behavior:**
        - Removes all chunks associated with the document.
        - Rebuilds indices to reflect the deletion.
    - **Returns:**
        - Boolean indicating success.
- **rebuild_all_indices(embedding_manager)**
    - **Behavior:**
        - Calls methods to rebuild FAISS, BM25, and TF-IDF indices.
    - **Returns:**
        - None.
- **rebuild_faiss_index(embedding_manager)**
    - **Behavior:**
        - Recomputes embeddings for all stored chunks and rebuilds the FAISS index.
    - **Returns:**
        - None.
- **rebuild_bm25_index()**
    - **Behavior:**
        - Updates the BM25 index incrementally by processing new chunks.
    - **Returns:**

        - None.
    - **search_bm25(query_text, top_k)**
        - **Parameters:**
            - query_text (str): The text query.
            - top_k (int): Maximum number of results.
        - **Behavior:**
            - Tokenizes the query and computes BM25 scores for each chunk.
            - Selects and returns the top matching chunks.
        - **Returns:**
            - A list of result dictionaries.
    - **search(query_vector, top_k)**
        - **Parameters:**
            - query_vector: The dense query vector.
            - top_k (int): Number of top results.
        - **Behavior:**
            - Uses FAISS to perform a nearest neighbor search.
        - **Returns:**
            - A list of matching chunks with associated metadata.
    - **rebuild_tfidf_index()**
        - **Behavior:**
            - Aggregates text for each document and rebuilds the TF-IDF index using Scikit-learn's TfidfVectorizer.
        - **Returns:**
            - None.
    - **get_tfidf_index()**
        - **Behavior:**
            - Returns the TF-IDF vectorizer, the TF-IDF index, and the list of document IDs.
        - **Returns:**

        - A tuple: (tfidf_vectorizer, tfidf_index, tfidf_doc_ids).
- **save(storage_dir, embedding_manager)**
    - **Parameters:**
        - storage_dir (str): Directory where indices and metadata will be saved.
        - embedding_manager: Used to rebuild the FAISS index before saving.
    - **Behavior:**
        - Rebuilds all indices and serializes them to disk.
        - Saves FAISS index, BM25 index, TF-IDF index, and metadata (using pickle).
    - **Returns:**
        - Paths to the saved files.
- **load(storage_dir)**
    - **Behavior:**
        - Class method that loads indices and metadata from the specified storage directory.
    - **Returns:**
        - An instance of VectorStorage with the loaded state.

---

3. Appendix

References & Additional Notes:

- **SentenceTransformers:** https://www.sbert.net/
- **FAISS:** https://faiss.ai/
- **BM25 Implementation:** Utilizes the rank_bm25 library for scoring based on token frequency.
- **TF-IDF:** Built using Scikit-learn's TfidfVectorizer for traditional sparse retrieval.
- **Design Considerations:**
  The system combines dense and sparse retrieval methods to provide robust performance across various document types and query scenarios. Its modular architecture facilitates easy updates and integration of new models or search strategies.