**📄 Functional Specification Document – RAG Model for Oracle Document Intelligence**

* **Author:** AI Product Consultant
* **Version:** 1.1
* **Last Updated:** 2025-07-19

**Change Log**

* **v1.1 (2025-07-19):**
  + Added **FR-9**, a new functional requirement for secondary section-level vector summarization.
  + Added a dedicated table for hardcoded system parameters in Section 4.
  + Moved Folder Structure and Helper Function reference to Appendix A for clarity.
  + Updated System Architecture and Test Scenarios to reflect the new embedding strategy.
  + Refined language for clarity and conciseness.
* **v1.2 (2025-07-19):**
  + Updated **FR-1** and **FR-2** to detail the specific mechanisms for document de-duplication (SHA256 hashing) and pipeline state tracking via the documents.status field in the database.
  + Added **FR-10** to formally specify the system's intelligent text structuring capabilities, including font-based heading detection and page enrichment.
  + Enhanced the **System Components** description in Section 3 to reflect the detailed multi-pass extraction process.
  + Updated **Appendix A** to specify the hashing algorithm.
* **v1.3 (2025-07-19):**
  + Replaced the generic Dual-Level Content Processing component description with a more detailed explanation of the generate\_summaries.py script's two primary functions: sentence-aware chunking and context-driven summarization.
  + Added **FR-11** to formally specify the idempotent processing logic, ensuring that only unprocessed sections are chunked and summarized.
  + Added **FR-12** to detail the precise algorithm for token-aware, overlapping chunking.
  + Added MODEL\_NAME to the Hardcoded System Parameters table.
  + Added a specific test case (**UC-5**) to validate the chunking and overlap logic.
* **v1.4 (2025-07-19):**
  + Clarified that vector embeddings are generated **exclusively from the LLM-generated summaries**, not the raw chunk text, to improve semantic search performance. This impacts **FR-3** and the **System Components** description.
  + Added a clarification note to the **Data Flow Diagram** to make this distinction explicit.
* **v1.5 (2025-07-19):**
  + Updated the **System Components** description for Embedding & Storage to detail the creation of three distinct artifacts per model: a FAISS index, a BM25 index, and a model-specific metadata database.
  + Added **FR-13** to specify the "clean rebuild" strategy, where existing artifacts for a model are deleted before new ones are generated.
  + Added **FR-14** to formalize the creation of both semantic (FAISS) and lexical (BM25) indexes.
  + Added **FR-15** to specify the two-way data linkage between the central project DB and the model-specific metadata DBs.
  + Added **UC-6** to provide a test case for the final embedded status update.
* **v1.6 (2025-07-19):**
  + Updated the **System Components** description for Query & Retrieval to detail the multi-step "thinking" pipeline.
  + Added **FR-17** to specify the Reciprocal Rank Fusion (RRF) algorithm for hybrid search.
  + Added **FR-18** to formalize the two-step LLM call process: first for reasoning ("thought process") and second for the final persona-based answer.
  + Added **FR-19** to specify the rich context formatting used for the final LLM prompt.
  + Added MODEL\_NAME\_RETRIEVER to the Hardcoded System Parameters table.
  + Refined UC-3 to directly reflect the observed runtime error and its cause.
* **v1.7 (2025-07-19):**
  + Added **System Component** description for the Web Interface (ui\_app.py).
  + Added **FR-20** to specify the user-facing administrative functions for document processing and embedding.
  + Added **FR-21** to formalize the requirement for LLM-powered query rewriting to maintain conversational context, replacing the previous FR-8.
  + Added **UC-7** to provide a test case for the conversational memory feature.
  + Finalized the FSD as all code has been reviewed.
* **v1.8 (2025-07-19):**
  + **Enhanced Ingestion:** Added **FR-22** to incorporate an OCR fallback mechanism for scanned or image-based PDFs, significantly improving ingestion robustness.
  + **Refined Prompt Engineering:** The descriptions for Contextual Summarization (Section 3), FR-18 (Two-Step Thinking), and FR-21 (Conversational Rewriting) have been refined with expert-recommended, role-based instructions to improve LLM performance and reliability.
  + **Advanced Retrieval Strategy:** Added **FR-23** to specify a more efficient two-step hierarchical search, leveraging the tiered vector strategy. This resolves Open Question #1.
  + **Improved Scalability:** Replaced the simplistic FR-13 (Clean Rebuild) with the superior **FR-24** (Granular Document Updates) to ensure the system can be maintained efficiently at scale. This resolves Open Question #3.
  + **Added Configurability:** Added a new STRICT\_ENSEMBLE\_MODE parameter to the Hardcoded System Parameters table to make the fail-fast behavior on missing artifacts configurable. This resolves Open Question #2.
* **v1.9 (2025-07-20):**
  + Aligned all component descriptions and file paths with the major code refactoring, including the move to a centralized utils package and config.py file.
  + Added **FR-25** to formalize the new LLM-powered query expansion feature.
  + Updated **FR-23** and the System Architecture Diagram to reflect that the hierarchical search is performed using the expanded query set.
  + Updated the **Hardcoded System Parameters** table to include new search parameters and reflect variable name changes from config.py.
  + Added **NFR-5** to document the new offline model caching capability for improved resilience.
  + Identified a critical defect (**DEF-01**) where the UI calls outdated backend scripts.

**1. Executive Summary**

This document outlines the functional specifications for the **Oracle Document Intelligence RAG System**, an advanced information retrieval platform designed to ingest, process, and provide intelligent answers from complex PDF documents. The system allows users (Consultants, Developers, End-Users) to ask natural language questions and receive accurate, persona-tailored, and source-cited answers.

The primary business motivation is to dramatically reduce manual research time within dense technical documentation. By implementing a sophisticated pipeline involving **dual-level embeddings**, hybrid retrieval (semantic + lexical), and LLM-driven generation, the system ensures both high-relevance and broad contextual understanding, directly addressing complex user queries.

**2. Business Requirements**

* **BR-1: Reduce Manual Search Time:** Eliminate the need for users to manually read through lengthy PDF documents.
* **BR-2: Improve Answer Accuracy & Relevance:** Provide contextually precise answers by combining specific chunk-level details with broader section-level context.
* **BR-3: Centralize & Enhance Knowledge:** Create a single, multi-faceted, and queryable source of truth from unstructured Oracle PDFs.
* **BR-4: Empower User Self-Service:** Enable users of varying expertise to get immediate answers without relying on subject matter experts.

**Stakeholder Goals:**

* **Automation:** Automate the Q&A process for technical support and consulting.
* **Efficiency:** Accelerate project timelines and task completion for all user personas.
* **Confidence:** Increase user trust through transparent, citable answers.

**Business KPIs:**

* **First-Response Accuracy:** >90% of answers correctly address the user's query, validated by reranker scores and user feedback.
* **Retrieval Latency:** End-to-end response time < 15 seconds for a standard query on recommended hardware.
* **Citation Relevance:** >95% of cited sources are directly relevant to the generated answer.

**3. System Architecture Overview**

The system employs a multi-stage RAG pipeline featuring a dual-embedding strategy for comprehensive information retrieval.

**Data Flow Diagram:**

Generated mermaid

graph TD

A[1. PDF Ingestion (.pdf)] --> B(2. Extraction & Structuring);

B --> C{3. Central Database (SQLite)};

subgraph "Embedding Generation"

C -- Raw Text Sections --> D(4a. Section-Level Summarization);

D -- LLM Call (Ollama) --> E{Section Summary};

E --> F(5a. Secondary Vector Generation);

C -- Raw Text Sections --> G(4b. Chunking);

G --> H{Primary Text Chunks};

H -- LLM Call (Ollama) --> I{Chunk Summaries};

I --> J(5b. Primary Vector Generation);

F & J -- SentenceTransformer Models --> K(6. Vector & Lexical Storage);

K -- FAISS, BM25, SQLite --> L;

end

subgraph "Query Time"

M[User Query] --> M\_EXP(7. Query Expansion);

M\_EXP -- LLM Call --> M\_SET{Expanded Query Set};

M\_SET --> N(8. Hybrid Search);

N -- Ensemble Models --> K;

K -- Candidate Chunks --> O(9. Reranking);

M -- Original Query --> O;

O -- CrossEncoder --> P{Top-K Context};

P --> Q(10. LLM Answer Generation);

M -- Persona --> Q;

Q -- Ollama/Llama3 --> R[11. Final Answer + Sources];

end

* **Note on Data Flow:** The vector embedding process (5b. Primary Vector Generation) specifically uses the **Chunk Summaries (I)** as its source material. The raw **Primary Text Chunks (H)** are not embedded; they are retrieved after a successful vector search to be used as final context for the answer-generation LLM.

**System Components:**

* **Ingestion & Structuring (extractor\_for\_pdf.py):** This component orchestrates the initial processing of PDFs in a three-pass approach:
  + **Font Analysis:** It first analyzes the entire document to programmatically identify the font sizes that constitute body text versus headings. This is achieved by profiling the character count for each font size within the main content area of the pages.
  + **Content Extraction & Hierarchy Building:** It then iterates through the document, extracting all text blocks. Using the font analysis from the first pass, it builds a hierarchical section\_id (e.g., Heading 1|Sub-heading 1.1) for each paragraph. This is managed using a heading\_stack where encountering a new heading (e.g., an H2) automatically clears any lower-level headings (e.g., H3, H4) from the stack, ensuring the hierarchy is always correctly maintained. Simultaneously, it extracts page-level enrichments including headers, footers, hyperlinks (with anchor text), and tables.
  + **Grouping & Storage:** Finally, it groups all text from the same section on the same page, merges it with the page-level enrichments, and saves the final structured data into the central SQLite database, setting the document's initial status to 'extracted'.
  + **OCR Fallback:** During the extraction process, the system will detect pages that are image-based and lack selectable text. For such pages, it will automatically trigger an OCR engine to extract the text before proceeding with the standard structuring and hierarchy-building logic (per **FR-22**).
* **Content Processing & Summarization (process\_and\_summarize.py):** This component is responsible for transforming large text sections into embeddable units. It operates by:
  1. **Identifying Work:** First, it queries the database to find all sections belonging to documents with the status 'extracted' that do not yet have corresponding chunks. This ensures the process is idempotent and only new content is processed.
  2. **Chunking:** For each unprocessed section, it applies a sentence-aware, token-based chunking algorithm. This algorithm splits the text into sentences, groups them into chunks that respect maximum and minimum token limits, and uses a sliding window to create a sentence-based overlap between consecutive chunks, preserving contextual flow.
  3. **Contextual Summarization:** For each generated chunk, it constructs a detailed, role-based prompt for the LLM (Ollama). This prompt instructs the LLM to act as a **"technical indexing expert"**. It includes the parent section\_header, the chunk\_text, and an explicit instruction to create a dense summary optimized for semantic search by future technical queries. The goal is to capture all key entities, concepts, and technical terms from the chunk. Crucially, this generated summary is the text that will be used for creating the vector embedding in the next stage.
  4. **Updating Document Status:** After all sections for a given document are processed, the document's status in the database is updated to 'chunked\_and\_summarized'.
* **Embedding & Storage (create\_vector\_store.py):** This script transforms the processed text summaries into searchable indexes. It iterates through each config.EMBEDDING\_MODEL\_LIST and performs the following for each one:
  1. **Clean Rebuild:** It first deletes any pre-existing index files (.faiss, .pkl, .db) for the specific model being processed. This ensures a clean, from-scratch build and prevents data inconsistencies.
  2. **Data Fetching:** It queries the central database to retrieve all chunk summaries that need to be embedded.
  3. **Vector & Index Generation:** Using the specified SentenceTransformer model, it generates vector embeddings for all summaries in batches. These embeddings are stored in two separate index types:
     + A **FAISS IndexIDMap** for efficient semantic similarity search.
     + A **BM25Okapi** index for keyword-based lexical search, created by tokenizing the summary text.
  4. **Decoupled Metadata Storage:** A separate, model-specific SQLite database (e.g., oracle\_metadata\_all-MiniLM-L6-v2.db) is created to store a copy of all relevant metadata (document name, page number, full summary text, etc.), mapping it to the vector\_id generated by FAISS. This decouples the query-time metadata from the main project database for faster lookups.
  5. **Linkage and Finalization:** The generated vector\_id is written back to the central chunks table for cross-referencing. Finally, once all chunks for a document have been successfully embedded by all configured models, the document's status in the central documents table is updated to 'embedded'.
* **Query & Retrieval (query\_handler.py):** This is the core execution engine that processes a user query and generates a final answer. It operates in several distinct stages:
  + **Model Loading:** On startup, it pre-loads all required models (Retriever, Reranker) from a local cache directory defined in config.py for offline resilience. It connects to the FAISS, BM25, and metadata database files for all configured embedding models.
  + **Query Expansion:** The initial user query is sent to an LLM to generate a set of semantically similar but varied questions (**FR-25**).
  + **Hierarchical Search:** The system executes its two-step hybrid search (**FR-23**) for *each* of the expanded queries to gather a comprehensive set of candidate sections and chunks.
  + **Result Fusion:** The ranked lists of candidate chunks from all sources are fused into a single list using the **Reciprocal Rank Fusion (RRF)** algorithm to produce a unified relevance score.
  + **Contextual Reranking:** The top candidates from the fused list are passed to a more powerful CrossEncoder model, which reranks them based on their direct relevance to the user's query.
  + **Rich Context Formatting:** The top 'k' chunks after reranking are retrieved with their full metadata. **To preserve the high-relevance order determined by the reranker**, an order map is created from the reranked list. The final rows of metadata are then explicitly sorted according to this map before being formatted into a rich context block for the LLM, including document name, section, page number, summary text, and any associated hyperlinks or table data.
  + **Two-Step LLM Generation:** The system uses a "thinking" pipeline with two sequential LLM calls:
    - **Thought Process:** The first call prompts the LLM to analyze the query and the formatted context, laying out a step-by-step reasoning plan.
    - **Final Answer:** The second call takes this "thought process" as its *only* input and, using a persona-specific prompt, generates the final, clean answer for the user.
* **Web Interface (ui\_app.py):** A Streamlit-based web application that serves as the primary user interface for the RAG system. It provides two core functionalities:
  + **Administrative Dashboard:** A sidebar panel allows users to upload new PDF documents and trigger the back-end processing pipelines (extractor\_for\_pdf.py and create\_vector\_store.py) directly from the UI. It displays the standard output and error streams from these scripts to provide real-time feedback on the ingestion process.
  + **Conversational Chat:** A main panel provides a chat interface for interacting with the RAG system. To handle multi-turn conversations, the UI uses an LLM to condense the chat history and the latest user query into a new, self-contained question before sending it to the core RAG pipeline (query\_handler.py). This ensures that the retrieval engine always receives the full context of the user's intent.

**4. Functional Requirements**

* **FR-1: Document Ingestion:** The system must accept PDF files placed in the /data source directory. Before processing, it must calculate a **SHA256 hash** of the file's content to check against the database, preventing the ingestion of duplicate documents. Upon successful processing, the source PDF file is moved to the /data/processed/ directory to serve as an archive and prevent reprocessing.
* **FR-2: Data Persistence:** All extracted and generated data is stored in a central SQLite database (project\_data.db) with a clearly defined schema. The system's pipeline progress for each document is tracked via a status field in the documents table, which transitions through states: 'extracted', 'chunked\_and\_summarized', and 'embedded'. The schema includes separate tables for documents, sections (which stores the large, structured text blocks with their enrichments), and chunks (which stores the smaller text units and their corresponding summaries).
* **FR-3:** Summarization-Based Embedding: The system must implement a summarization-first embedding strategy. For each text chunk, a dense summary must be generated by an LLM using the chunk's content and its section header for context. This generated summary, not the raw chunk text, shall serve as the source text for the vector embedding. This strategy is designed to increase the conceptual density of the vectors and improve the performance of semantic retrieval.
* **FR-4: Ensemble Retrieval:** An --ensemble flag enables retrieval from multiple embedding models simultaneously.
* **FR-5: Contextual Reranking:** A CrossEncoder model must rerank all retrieved candidates to select the most relevant context for the final answer.
* **FR-6: Persona-Based Generation:** The system must support Consultant, Developer, and User personas, each with a unique prompt template to control the answer's tone and style. The web UI will map these user-friendly names to the corresponding script arguments (e.g., "Consultant" -> "consultant\_answer").
* **FR-7: Sourced Answers:** All generated answers must include citations listing the source document and page number.
* **<IGNORE OVERRIDDEN BY FR20 & FR21>FR-8: Conversational Memory:** The UI (ui\_app.py) must condense chat history into a standalone question for the RAG pipeline to maintain conversational context**<IGNORE OVERRIDDEN BY FR20 & FR21>**
* **FR-9: Hierarchical Two-Tiered Vector Generation:** The system will implement a two-tiered embedding strategy to handle both broad and specific queries. This is the **sole** method for creating secondary vectors.
  + **Tier 1 (Primary Vectors):** For each granular chunk, a single vector will be generated based on its LLM-generated summary. This tier is optimized for retrieving specific facts and detailed information.
  + **Tier 2 (Secondary Vectors):** For each high-level document section, a single vector will be generated based on an LLM-generated summary of that section's entire raw\_text and its section\_header. This tier is optimized for retrieving thematic and topic-level information.
  + **Design Rationale:** This hierarchical approach is explicitly chosen over generating multiple vector types (e.g., from raw text vs. summary) for the same chunk. It provides a cleaner, more efficient, and less redundant vector space, reducing system complexity and improving the distinction between broad and specific search intents.
* **FR-10: Intelligent Text Structuring:** The system must not rely on fixed rules for identifying document structure. It must programmatically analyze each PDF to differentiate headings from paragraph text based on font size distribution. It must also extract and associate key page-level enrichments with the core text, including headers, footers, hyperlinks, and tables, which can later be used to provide richer context.
* **FR-11: Idempotent Content Processing:** The summarization pipeline stage must be idempotent. The system must identify sections to be processed by querying for documents with a status of 'extracted' and confirming that no corresponding entries exist for that section\_id in the chunks table. This prevents reprocessing of already chunked content.
* **FR-12: Sentence-Aware Overlapping Chunking:** The system must use a specific chunking algorithm that first splits text into sentences. It shall then create chunks based on token limits (CHUNK\_SIZE\_TOKENS, CHUNK\_MIN\_TOKENS) while maintaining a configurable overlap of one or more full sentences (CHUNK\_OVERLAP\_SENTENCES) between adjacent chunks. This ensures that context is not abruptly lost at chunk boundaries. Trailing sentences that do not meet the minimum token count should be appended to the last valid chunk.
* **<DELETED REFER REASON>FR-13: Clean Rebuild of Indexes:** The embedding pipeline must follow a "clean rebuild" methodology. When the rag\_embed\_store.py script is executed for a given model, it must first delete all existing artifact files (.faiss, .pkl, .db) associated with that model before generating new ones. This ensures that the indexes are always perfectly synchronized with the content in the central database and prevents stale data. **<DELETED REFER REASON>  
  <**Reason: he existing requirement FR-13: Clean Rebuild of Indexes is functional for a small-scale prototype but is not a viable strategy for a production system, as it requires re-processing the entire dataset for a single document update. The expert recommendation for FR-24: Granular Document Update and Re-embedding is a best-practice solution that is not comparable in terms of efficiency and scalability. Therefore, FR-13 is superseded and will be replaced to align the FSD with a production-ready architecture. **>**
* **FR-14: Dual Index Creation (Semantic and Lexical):** For each embedding model, the system must generate and persist two distinct types of search indexes from the chunk summaries:
  + **Semantic Index:** A **FAISS (IndexIDMap)** index must be created from the dense vector embeddings for similarity search.
  + **Lexical Index:** A **BM25Okapi** index must be created from the tokenized text of the summaries to enable keyword-based search.
* **FR-15: Decoupled Metadata and Two-Way Linkage:** The system must maintain two separate but linked database systems.
  + A central **Project Database** (project\_data.db) tracks the overall pipeline state.
  + For each embedding model, a separate **Metadata Database** (e.g., oracle\_metadata\_model-name.db) must be created to store a snapshot of the chunk metadata required at query time.
  + This linkage must be two-way: the vector\_id from the FAISS index must be stored in the central chunks table, and the chunk\_id from the central database must be stored in the model-specific metadata database. This design optimizes query-time performance by preventing joins to the main database during retrieval.
* **FR-16: Holistic Document Embedding Status:** The final status of a source document shall only be updated to 'embedded' when **all of its constituent chunks have been successfully embedded by all configured models**. This state integrity must be enforced by a final SQL query that verifies that the vector\_id columns for every model (e.g., vector\_id\_minilm, vector\_id\_bge) are NOT NULL for all chunks associated with that document before the status is updated.
* **FR-17: Reciprocal Rank Fusion (RRF) for Hybrid Search:** The system must fuse the ranked lists from the semantic (FAISS) and lexical (BM25) searches using the Reciprocal Rank Fusion algorithm. Each candidate chunk's score in the final list will be calculated based on its rank in the initial lists, using the formula score = 1 / (k + rank), where k is a configurable constant. In --ensemble mode, results from all models will be added to the same fused list before sorting.
* **FR-18: Two-Step "Thinking" LLM Pipeline:** Answer generation must follow a two-step process to improve quality and traceability.
  + The system will first make an LLM call with a thought\_process prompt. This prompt instructs the LLM to act as a meticulous fact-checker. It will receive the user query and retrieved context, and be directed to: a) Lay out a step-by-step reasoning plan, b) Critically assess whether the provided context directly answers the user's query, and c) Extract all relevant facts and source citations from the context without inventing new information. The output of this step is the LLM's explicit reasoning.
  + A second LLM call will then be made. This call provides only the output from the first step (the thought process) along with a persona-specific prompt. This final prompt instructs the LLM to synthesize the provided facts into a coherent answer, adhere strictly to the given persona, and format the citations correctly. This step is forbidden from using any information not present in the "thought process" input.
* **FR-19: Rich Context Formatting for Generation:** When preparing the context for the final LLM prompt, the system must format each of the top 'k' reranked chunks to include not only the summary text but also all available metadata, including Document Name, Section, Page Number, Hyperlinks, Tables, and Header/Footer text, to give the LLM maximum context.
* **FR-20: User-Facing Pipeline Administration:** The system must provide a graphical user interface (GUI) that allows non-technical users to manage the document pipeline. This interface must include controls to:
  + Upload new PDF documents.
  + Trigger the document extraction and structuring process (extractor\_for\_pdf.py).
  + Trigger the embedding and storage process (create\_vector\_store.py).
  + Display process status and logs directly in the UI.
* **FR-21: LLM-Powered Conversational Rewriting:** To maintain context in a multi-turn conversation, the system's chat interface must not send follow-up questions directly to the RAG pipeline. Instead, it must first make a preliminary LLM call to a "query rewriting expert" prompt. This prompt will provide the chat history and the latest user question, and the LLM will be instructed to:
  + Resolve any coreferences (e.g., "it", "that", "them") by replacing them with the specific entities from the chat history.
  + Synthesize the user's latest question with relevant context from previous turns to form a complete, standalone question.
  + Preserve the key technical terms and intent of the original user question.  
    This rewritten question is then passed to the RAG query engine
* **FR-22: OCR Fallback for Image-Based PDFs:** During the extraction phase, the system must detect if a page contains primarily image-based content with no extractable text. If such a page is detected, the system shall automatically invoke an OCR process to extract the text from the page image before proceeding with the structuring and hierarchy-building logic.
* **FR-23: Hierarchical Search Strategy:** The query process shall leverage the two-tiered vector structure in a sequential manner. First, the *entire set of expanded query vectors* (**FR-25**) will be used to search against the secondary (section-level) vectors to identify the top N most relevant document sections the top N most relevant document sections. In the second step, the hybrid semantic (FAISS) and lexical (BM25) search for *primary (chunk-level)* candidates will be filtered to run exclusively on chunks belonging to the pre-identified relevant sections. This focuses the high-granularity search on the most promising areas of the knowledge base/.
* **FR-24: Granular Document Update and Re-embedding:** The system must support efficient updates when a source PDF is modified.
  + Upon detecting a new version of a document (identified by a different SHA256 hash for the same filename), the system shall first delete all existing data associated with the old document version from the documents, sections, and chunks tables.
  + The FAISS and BM25 indexes shall be updated by removing the vector and document IDs associated with the old document.
  + The new document version will then be processed through the standard ingestion pipeline (extracted, chunked\_and\_summarized, embedded), and the new vectors and metadata will be added to the existing indexes without requiring a full rebuild of all other documents.
* **FR-25: LLM-Powered Query Expansion:** Before initiating a search, the system must use a preliminary LLM call to expand the user's initial question into a set of 3-5 alternative queries. These variants should explore different phrasings and potential sub-topics related to the original question. The subsequent hierarchical search shall be performed using this entire set of expanded queries to maximize the retrieval of relevant sections and chunks.

**Hardcoded System Parameters**

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | File | Value | Description |
| LLM\_MODEL | config.py | 'llama3' | The specific LLM model to be called via the Ollama endpoint for summarization and generation tasks. |
| PRIMARY\_RETRIEVER\_MODEL | config.py | 'all-MiniLM-L6-v2' | The primary retriever model used to encode the user's query. Must be in the EMBEDDING\_MODELS\_LIST. |
| EMBEDDING\_MODELS\_LIST | config.py | ['all-MiniLM-L6-v2', 'BAAI/bge-small-en-v1.5'] | List of all embedding models to be used for document embedding and ensemble retrieval. |
| RERANKER\_MODEL | config.py | 'cross-encoder/ms-marco-MiniLM-L-6-v2' | The cross-encoder model for reranking. |
| CHUNK\_SIZE\_TOKENS | process\_and\_summarize.py | 384 | Maximum token size for a text chunk. |
| CHUNK\_MIN\_TOKENS | process\_and\_summarize.py | 50 | Minimum token size to be considered a valid chunk. |
| CHUNK\_OVERLAP\_SENTENCES | process\_and\_summarize.py | 1 | Number of sentences to overlap between chunks. |
| HEADER\_Y\_RATIO | extractor\_for\_pdf.py | 0.08 | Top 8% of the page is considered the header. |
| FOOTER\_Y\_RATIO | extractor\_for\_pdf.py | 0.915 | Bottom 8.5% of the page is considered the footer. |
| RRF\_K | config.py | 60 | Constant used in Reciprocal Rank Fusion formula. |
| HIERARCHICAL\_SEARCH\_TOP\_N\_SECTIONS | config.py | 5 | Number of sections to find in the first pass of a hierarchical search. |
| HIERARCHICAL\_SEARCH\_TOP\_N\_CHUNKS | config.py | 10 | Number of chunks to retrieve in the second pass for reranking. |
| STRICT\_ENSEMBLE\_MODE | config.py | True | If True, the system will exit if any configured model's artifacts are missing. If False, it will log a warning and proceed. |
| OLLAMA\_URL | config.py | http://localhost:11434/api/generate | Endpoint for the local LLM server. |

**<OLD MODEL IGNORE>**

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | File | Value | Description |
| **MODEL\_NAME** | **generate\_summaries.py** | **'llama3'** | **The specific LLM model to be called via the Ollama endpoint for summarization tasks.** |
| **MODEL\_NAME\_RETRIEVER** | **rag\_query\_cli\_thinking.py** | **'all-MiniLM-L6-v2'** | **The specific retriever model used to encode the user's query.** |
| EMBEDDING\_MODELS | rag\_embed\_store.py | ['all-MiniLM-L6-v2', 'BAAI/bge-small-en-v1.5'] | List of embedding models to use. |
| MODEL\_NAME\_RERANKER | rag\_query\_cli\_thinking.py | 'cross-encoder/ms-marco-MiniLM-L-6-v2' | The cross-encoder model for reranking. |
| CHUNK\_SIZE\_TOKENS | generate\_summaries.py | 384 | Maximum token size for a text chunk. |
| CHUNK\_MIN\_TOKENS | generate\_summaries.py | 50 | Minimum token size to be considered a valid chunk. |
| CHUNK\_OVERLAP\_SENTENCES | generate\_summaries.py | 1 | Number of sentences to overlap between chunks. |
| HEADER\_Y\_RATIO | extract\_pdf\_v6\_7.py | 0.08 | Top 8% of the page is considered the header. |
| STRICT\_ENSEMBLE\_MODE | **rag\_query\_cli\_thinking.py** | **True** | If True, the system will exit with an error if any configured ensemble model's artifacts are missing. If False, it will log a warning and proceed with the available models. |
| FOOTER\_Y\_RATIO | extract\_pdf\_v6\_7.py | 0.915 | Bottom 8.5% of the page is considered the footer. |
| RRF Constant (k) | rag\_query\_cli\_thinking.py | 60 | Constant used in Reciprocal Rank Fusion formula. |
| OLLAMA\_URL | rag\_query\_cli\_thinking.py | http://localhost:11434/api/generate | Endpoint for the local LLM server. |

**<OLD MODEL IGNORE>**

**5. Input/Output Specifications**

**Inputs:**

* **Documents:** .pdf files with a discernible structure.
* **Query Parameters (CLI/UI):**
  + --query: (string) The user's question.
  + --persona: (string) consultant\_answer, developer\_answer, user\_answer. Note: The UI maps these to user-friendly names (Consultant, Developer, User).
  + --ensemble: (flag) Enable multi-model retrieval.
  + --top\_k: (int) Max chunks for final context.

**Outputs:**

* **Formatted Response:**
  + answer: The final LLM-generated text.
  + citations: A list of source documents and page numbers.
  + context\_chunks: (Optional) The full text of chunks used for the answer.
  + logging\_info: Intermediate steps like "thought process" and scores.

**6. Extensibility / Integration Points**

* **Adding Embedding Models:** Add the model name to the EMBEDDING\_MODELS list in rag\_embed\_store.py. The pipeline handles the rest.
* **Custom Chunking:** The chunk\_text function in generate\_summaries.py can be replaced with an alternative chunking strategy.
* **Adding Personas:** Create a new [persona\_name]\_answer.txt file in the /prompts folder and add it as a choice in the UI/CLI.
* **API Integration:** The core query logic can be wrapped in a FastAPI or Flask server to provide an API endpoint for other applications.

**7. Non-Functional Requirements**

* **Performance:** < 15s end-to-end response time on hardware with a dedicated GPU (e.g., NVIDIA RTX 3060 or better).
* **Scalability:** FAISS is capable of handling millions of vectors. The primary scaling constraint is the local Ollama LLM's inference speed.
* **Resource Requirements:** Minimum 16GB RAM; 8GB+ VRAM on a dedicated GPU is strongly recommended.
* **Security & Privacy:** The entire pipeline is designed to run locally. No data is sent to external or cloud-based APIs, ensuring complete data privacy.
* **NFR-5: Offline Resilience:** The system must improve resilience to network outages by caching all required transformer models locally. On startup, the query engine will load models from the local\_models\_cache/ directory, avoiding the need to download them from the internet after the initial run.

**8. Test Scenarios / Use Cases**

* **UC-1: Standard Query:**
  + **Input:** python -m scripts.query\_handler --query "install Smart View"
  + **Expected Output:** A coherent answer with cited sources from the correct document.
* **UC-2: Full Ingestion Pipeline:**
  + **Input:** A new PDF in the /data folder.
  + **Expected Output:** The PDF is processed through all scripts (extract, summarize, embed), and the final artifacts are created in the output/ and embeddings/ directories.
* **UC-3: Negative Case (Missing Ensemble Artifacts):**
  + **Input:** Run a query with the --ensemble flag when the artifact files (e.g., oracle\_index\_BAAI-bge-small-en-v1.5.faiss) for one of the configured models do not exist.
  + **Observed Behavior:** The script fails on startup with a RuntimeError: could not open ... for reading: No such file or directory.
  + **Expected Behavior:** This is the correct behavior under the current fail-fast design. It confirms that the system requires all artifacts to be present for ensemble mode to function. This highlights the importance of the decision in Open Question #2.
* **UC-4: Secondary Vector Retrieval Test:**
  + **Input:** A broad query like --query "What is the main purpose of accounting hub reporting?"
  + **Expected Output:** The retrieval logs should show high scores for section-level summaries, demonstrating that the secondary vectors successfully matched the broad topic even if no specific chunk did.
* **UC-5: Chunking Algorithm Validation:**
  + **Input:** A text section containing 10 sentences where the 5th sentence causes the token count to exceed CHUNK\_SIZE\_TOKENS.
  + **Expected Output:** The system should create at least two chunks. The first chunk should contain the first 4 or 5 sentences. The second chunk must begin with the last sentence from the first chunk (respecting CHUNK\_OVERLAP\_SENTENCES = 1), followed by the subsequent sentences.
* **UC-6: Document Status Finalization Integrity:**
  + **Scenario:** A document has been processed, and the rag\_embed\_store.py script is run. However, the script is interrupted after successfully embedding for the all-MiniLM-L6-v2 model but *before* processing the BAAI/bge-small-en-v1.5 model.
  + **Expected Behavior:** A SELECT status FROM documents... query must show that the document's status remains 'chunked\_and\_summarized'. Only after the script is re-run and completes successfully for *all* models should the status change to 'embedded'.
* **UC-7: Conversational Context Handling:**
  + **Scenario:**
    - User asks: "How do I install Smart View?" The system provides an answer.
    - User asks a follow-up question: "What are the prerequisites for it?"
  + **Expected Behavior:**
    - The UI should show the condensed, standalone question that was sent to the RAG pipeline (e.g., "What are the prerequisites for installing Smart View?").
    - The final answer provided by the system should correctly describe the prerequisites for Smart View, demonstrating that the conversational context was successfully understood and passed to the retrieval engine.

**9. Open Questions / Clarifications**

1. **Retrieval Weighting:** How should the RRF algorithm weigh results from primary (chunk) vectors versus secondary (section) vectors? Does one type get priority? (TBD - Needs clarification from project owner).

**Recommendation:**  
Instead of simply weighting RRF scores, implement a dynamic, two-step retrieval strategy that leverages your hierarchical vectors. First, use the broad, section-level vectors to identify the most relevant *sections* of the documents. Then, perform the detailed, chunk-level hybrid search *only within those top-ranked sections*.

**Reasoning:**  
This approach is more efficient and often more accurate than fusing two disparate sets of vectors (section-level and chunk-level) in a single step. It mimics how a human expert would research: first find the right chapter (section), then look for the specific detail (chunk). It directly uses your two-tiered vector strategy (FR-9) in a more structured way.

* **[RESOLVED] Retrieval Weighting:** This is addressed by **FR-23: Hierarchical Search Strategy**. The system will use a two-step filtering approach rather than direct weighting in a single fusion step.

1. **Error Handling for Missing Models:** Should the system fall back to using only available models if an ensemble model's files are missing, instead of halting?

**Recommendation:**  
Make the "fail-fast" behavior configurable. This provides flexibility during development and testing without sacrificing production stability.

**[RESOLVED] Error Handling for Missing Models:** This is now a configurable behavior via the STRICT\_ENSEMBLE\_MODE parameter. The default remains to fail fast for production stability, but this can be toggled for development.

* **Suggested New Hardcoded System Parameter:**

|  |  |  |  |
| --- | --- | --- | --- |
| Parameter | File | Value | Description |
| STRICT\_ENSEMBLE\_MODE | rag\_query\_cli\_thinking.py | True | If True, the system will exit with an error if any configured ensemble model's artifacts are missing. If False, it will log a warning and proceed with the available models. |

1. **Document Updates:** What is the strategy for updating embeddings when a source PDF is updated? Does it require a full rebuild?

**Recommendation:**  
Implement a granular update strategy instead of relying on a full rebuild. This is a critical best practice for maintaining large-scale RAG systems efficiently.

**Reasoning:**  
A "clean rebuild" is simple but does not scale. For large document sets, rebuilding all embeddings for a single updated file is computationally expensive and leads to system downtime. A more sophisticated approach is to manage updates at the document level.

* **[RESOLVED] Document Updates:** This is addressed by **FR-24: Granular Document Update and Re-embedding**, which specifies a scalable, document-level update process instead of a full corpus rebuild.

**10. Glossary and Acronyms**

|  |  |
| --- | --- |
| Term | Description |
| **RAG** | Retrieval-Augmented Generation. |
| **FAISS** | Facebook AI Similarity Search library for vector search. |
| **BM25** | Keyword-based scoring algorithm for lexical search. |
| **Embedding** | A dense vector representation of text. |
| **Reranker** | A Cross-Encoder model that re-scores retrieved chunks for relevance. |
| **Persona** | A role that dictates the tone and style of the final LLM answer. |
| **Ollama** | A tool for running open-source LLMs locally. |
| **LLM** | Large Language Model (e.g., Llama3). |

**Appendix A: Project Structure and Helper Reference**

**Folder Structure**

Generated code

oracle\_rag\_project/

├── data/ # Raw PDF input files

├── data/processed/ # Archived PDFs after processing

├── embeddings/ # FAISS/BM25 indexes + per-model metadata DBs

├── local\_models\_cache/ # Cached transformer models for offline use

├── output/ # Central project database (project\_data.db)

├── prompts/ # Prompt templates for personas and tasks

├── scripts/

│ ├── config.py # Central configuration for paths and models

│ ├── extractor\_for\_pdf.py # PDF to Structured Text

│ ├── process\_and\_summarize.py # Text to Chunks & Summaries

│ ├── create\_vector\_store.py # Chunks to Embeddings

│ ├── query\_handler.py # Query Engine CLI

│ ├── ui\_app.py # Streamlit UI

│ └── utils/

│ ├── database\_utils.py # Manages central DB schema and connections

│ ├── llm\_utils.py # Manages calls to the Ollama LLM

│ ├── utils.py # General utilities (text cleaning, logging, etc.)

│ └── vector\_store\_utils.py # Manages vector index updates

**Key Utility Modules (scripts/utils/)**

* **database\_utils.py:** initialize\_database(), get\_db\_connection(), calculate\_file\_hash() (uses SHA256).
* **utils.py:** get\_logger(), clean\_text(), classify\_paragraph\_font\_sizes(), extract\_page\_enrichments().
* **llm\_utils.py:** llama3\_call() provides a standardized, reusable function for interacting with the Ollama API.
* **vector\_store\_utils.py:** remove\_document\_vectors() provides the core logic for granularly deleting vectors from all model indexes, supporting **FR-24**.