RAG System - Detailed Version Control Documentation

Version 1.0: ragtest.py - Foundational PDF RAG System

- **Core Functionality:**
  - Establishes a command-line interface (CLI) for a Retrieval-Augmented Generation (RAG) system specifically designed for PDF documents.
  - The primary LLM interaction is configured to use the Groq API.
- **Key Components & Features:**
  - **RAGConfig Class:**
    - Manages system configuration loaded from an config.ini file.
    - Handles paths (pdf_dir, index_dir), model names (encoder_model, summarizer_model, llm_model), and processing parameters (chunk_size, overlap, k_retrieval, temperature).
    - Default LLM: meta-llama/llama-4-scout-17b-16e-instruct (via Groq).
    - Provides default values if config.ini is missing and can save a default configuration file.
    - Supports automatic device selection (cuda if available, else cpu) for local models.
  - **RAGSystem Class:**
    - **Initialization:**
      - Loads local models: SentenceTransformer for embeddings and T5ForConditionalGeneration/T5Tokenizer for summarization.
      - Creates necessary directories for data and indexes.
    - **Text Processing & Indexing:**
      - clean_text(): Performs basic text cleaning (removes "(cid:X)" artifacts, normalizes whitespace).
      - extract_text_and_tables(): Uses pdfplumber to extract raw text and detect tables from PDF pages. Tables are converted to string representations.
      - chunk_content(): Splits the extracted text and table strings into smaller, overlapping chunks based on chunk_size and overlap parameters.
      - process_pdfs():
        - Orchestrates the processing of all PDF files in the specified directory.
        - For each PDF, it attempts to load pre-existing chunks, embeddings (NumPy arrays), and FAISS indexes if they exist and their dimensions are compatible.
        - If cached artifacts are not found or are incompatible, it re-generates them:
          - Text extraction and chunking.
          - Sentence embedding generation for each chunk using the loaded SentenceTransformer model.

            - Construction of a FAISS IndexFlatL2 index from the embeddings.
        -
        - Saves chunks (JSON), embeddings (.npy), and FAISS indexes to disk.
        - Includes error handling for individual PDF processing, attempting to clean up intermediate files if a PDF fails.
    - **Querying & Generation:**
        - summarize_text(): Implements text summarization using the T5 model (though its usage in the main query pipeline is commented out in this version).
        - query_pdfs():
            - Encodes the user's query using the SentenceTransformer model.
            - Performs a similarity search using FAISS across all loaded PDF indexes.
            - Retrieves the top k_retrieval chunks from each PDF based on L2 distance (lower is better).
            - Returns a dictionary where keys are PDF filenames and values are lists of ranked result chunks.
        - aggregate_context():
            - Takes the query_pdfs results.
            - Constructs a single context string by concatenating the content of retrieved chunks.
            - Supports "simple" (concatenate all retrieved) and "top_k" (concatenate top K globally) strategies.
            - Manages context length to stay within an approximate token limit (max_context_tokens).
        - query_llm():
            - Takes the aggregated context and the original user query.
            - Formats a prompt instructing the LLM to answer based solely on the provided context.
            - Retrieves the GROQ_API_KEY from environment variables.
            - Uses langchain_groq.ChatGroq to send the request to the configured Groq model.
            - Includes basic retry logic for API calls.
            - Provides a fallback response if the LLM query fails or returns no answer.
        - run_query(): The main entry point for a user query. It orchestrates the sequence: query_pdfs -> aggregate_context -> query_llm.
- **CLI (main() function):**
    - Initializes RAGSystem.
    - Calls process_pdfs() to build/load indexes.

        - Enters an interactive loop, prompting the user for queries.
        - Calls rag.run_query() for each query.
        - Uses print_results() to display the LLM's answer and top retrieval results to the console.
        - Includes basic startup checks for GROQ_API_KEY.
    - **Logging:**
        - Configures logging to output messages to both a file (rag_system.log) and the console (StreamHandler).

Version 2.0: ragtest_c.py - Streamlit UI Integration

- **Core Theme:** Transitioned from a CLI application to a web-based graphical user interface (GUI) using Streamlit, enhancing usability and interactivity.
- **Key Changes:**
    - **Streamlit Integration:**
        - Added streamlit and time imports.
        - The main() function and print_results() were replaced by Streamlit's application flow.
        - UI elements like st.title, st.text_input, st.button, st.spinner, st.progress, st.expander, st.sidebar, st.error, st.warning, st.success, st.info are used for layout, input, feedback, and output.
    - **Caching for Performance:**
        - @st.cache_resource decorates load_rag_system() to cache the RAGSystem object (including loaded models), preventing re-initialization on every UI interaction.
        - @st.cache_data decorates process_documents() to cache the results of PDF processing (status and list of indexed files), avoiding reprocessing unless relevant files or code change.
    - **UI-Driven Workflow:**
        - The application first loads the RAGSystem.
        - It then calls process_documents(), which internally calls rag_sys.process_pdfs(). This function now includes a progress_callback to update the Streamlit UI with status messages and progress bars.
        - If processing is successful, the UI allows users to input queries.
        - Query results, including LLM answers and retrieval details, are displayed within expandable sections.
    - **RAGConfig Adaptations:**
        - Default llm_model in MODELS changed to llama3-8b-8192.
        - Automatic saving of default config.ini (_save_config) was commented out to prevent potential file permission issues in deployed Streamlit environments.
    - **RAGSystem Adaptations for UI Feedback:**
        - Model loading in __init__ is wrapped with st.spinner. Errors during model loading now use st.error and call st.stop().

        - extract_text_and_tables and chunk_content use st.progress for visual feedback on lengthy operations.
        - process_pdfs was significantly enhanced to use the progress_callback for detailed UI updates (current file, stage of processing, errors, warnings). It also disables the internal progress bar of the sentence transformer encoding to avoid redundancy.
        - query_pdfs and other methods now use st.warning or st.error for UI notifications.
    - **LLM Interaction (query_llm)**:
        - The system prompt was updated to include "You are called the SatSure LLM..." and specific instructions about answering based on proposals from a PDF document.
        - Enhanced retry logic for Groq API calls with exponential backoff.
    - **Return Structure (run_query):**
        - Now returns a structured dictionary containing query details, retrieval results, aggregated context, answers, status, and error messages, facilitating easier display in the Streamlit UI.

Version 3.0: ragtest_c2.py - Multi-Document Support (PDF, XLSX, CSV, PPTX)

- **Core Theme:** Extended document processing capabilities beyond PDFs to include Microsoft Excel (.xlsx), CSV (.csv), and PowerPoint (.pptx) files.
- **Key Changes:**
    - **Configuration (RAGConfig):**
        - PATHS: pdf_dir renamed to data_dir.
        - MODELS: summarizer_model (T5) commented out as it was largely unused. Default LLM model set to meta-llama/llama-4-scout-17b-16e-instruct.
        - PARAMETERS: Added max_context_tokens (default 4000) for LLM context window management and max_chars_per_element (default 1000) to truncate excessively long content from individual spreadsheet cells or PowerPoint shapes.
        - SUPPORTED_EXTENSIONS: New section to define processable file types (e.g., ".pdf, .xlsx, .csv, .pptx").
        - _ensure_defaults(): Added to ensure all default sections/options are present in loaded configs.
    - **RAGSystem - Content Extraction & Processing:**
        - **New Extractors:**
            - _extract_xlsx(): Uses pandas.ExcelFile to parse sheets, converts each sheet to a string representation for indexing.
            - _extract_csv(): Uses pandas.read_csv (with encoding detection) to parse CSVs, converts to string.
            - _extract_pptx(): Uses python-pptx to extract text from slide shapes and notes.

        - extract_content(): New dispatcher method; calls the appropriate _extract_* method based on file extension.
        - Metadata in extracted content blocks now includes file_type (e.g., "pdf", "xlsx") and more detailed source_info (e.g., sheet name, slide number).
        - process_pdfs() renamed to process_files(). It now iterates through files matching any of the supported_extensions.
        - pdf_chunks renamed to file_chunks to reflect generic file handling.
    - **RAGSystem - Querying & Context:**
        - query_pdfs() renamed to query_files().
        - aggregate_context(): Improved formatting of source information in the context header to dynamically include file type, sheet/slide details.
        - query_llm(): System prompt generalized to refer to "various source documents" and use source_file_name.
    - **Streamlit UI:**
        - UI text updated to reflect multi-document capabilities.
        - Document processing spinner now lists all supported extensions.
        - Retrieval details in the UI display enhanced source information (sheet, slide, etc.).
        - Sidebar now displays max_context_tokens.

Version 3.1: ragtest_c2-1.py - Enhanced PPTX Extraction & Global Context Aggregation

- **Core Theme:** Refined PowerPoint content processing by prepending titles and adopted a global strategy for context aggregation.
- **Key Changes:**
    - **PowerPoint Extraction (_extract_pptx):**
        - Introduced logic to identify a slide's title and prepend it to the content of the *subsequent* slide with a marker "[Context from Title on Previous Slide (slide_number-1)]: {title_text}".
        - This aims to create more contextually complete chunks for slides that follow a title slide.
        - Separator \n\n---\n\n used when combining prepended title with current slide content.
    - **Context Aggregation (aggregate_context):**
        - Shifted from per-file context aggregation to a global strategy.
        - All retrieved chunks from all files are now flattened into one list.
        - This list is sorted globally by relevance score.
        - The context for the LLM is built by taking the top chunks from this globally sorted list, up to k_retrieval (if "top_k" strategy) or the character limit.
        - Return structure changed to {"combined_context": str, "source_files": list}.
    - **LLM Interaction (query_llm, run_query):**
        - Adapted to handle the new single, combined context string and list of contributing source files.
        - run_query now produces a single answer instead of per-file answers.
    - **Streamlit UI:**

        - Conditional rendering checks (if "streamlit" in sys.modules:) added around Streamlit-specific UI calls (st.spinner, st.progress) to allow core logic to run independently.
        - "Re-process Documents" button added to clear Streamlit caches and force a full re-initialization and processing.
        - UI updated to display the single combined answer and to show the combined context sent to the LLM.

Version 3.2: ragtest_c2-2.py (Internal: ragtest_c2_merged.py) - Advanced PPTX Slide Merging Strategy

- **Core Theme:** Implemented a more sophisticated heuristic for merging "title-like" PowerPoint slides with their subsequent content slides.
- **Key Changes:**
    - **Configuration (RAGConfig):**
        - New PARAMETERS option: pptx_merge_threshold_words (default 50). This threshold helps determine if a slide with a title and minimal other text should be considered for merging.
    -
    - **PowerPoint Extraction (_extract_pptx):**
        - Revised merging logic:
            - Identifies slides with a designated title placeholder (slide.shapes.title).
            - Calculates other_text_word_count (non-title content) for the slide.
            - If a slide (pending_title_slide_data) was identified as "title-like" (had a title, other_words <= threshold), and the *current* slide has content, their contents are merged.
            - The merged block is tagged as slide_text_merged and includes source_info about the merged slides.
            - If the *current* slide itself is deemed "title-like", it becomes the new pending_title_slide_data to be potentially merged with the *next* slide.
            - Otherwise, pending slides and the current slide are processed independently.
        -
        - More detailed logging for the merge decisions.
        - Improved detection of body placeholders in PPTX shapes (MSO_SHAPE_TYPE.BODY, etc.).
    - **Context Aggregation (aggregate_context):**
        - The k_retrieval limit is now explicitly applied to the globally sorted flat_results before iterating to build the context string.
        - Source string formatting in context headers now correctly includes details for merged PPTX slides (slide_title, slide_content).

Version 3.3: ragtest_c2-3.py (Internal: ragtest_c3_query_adapt.py) - LLM-Powered Query Analysis (Classification & Decomposition)

- **Core Theme:** Introduced an initial layer of query understanding using the LLM to classify queries and decompose complex ones before retrieval.
- **Key Changes:**
  - **RAGSystem - Query Analysis Helpers:**
    - _get_llm(): Centralized LLM client (Groq) initialization and caching.
    - _call_llm_for_analysis(): Helper for internal LLM calls (classification, decomposition).
    - _classify_query(): Prompts the LLM to classify the user query as "Simple Retrieval" or "Complex/Reasoning (Text)".
    - _decompose_query(): For "Complex/Reasoning (Text)" queries, prompts the LLM to break them into simpler, factual sub-queries.
  - **RAGSystem - Modified run_query() Workflow:**
    - **Classify Query**: User query is first classified.
    - **Decompose (if Complex Text)**: If classified as "Complex/Reasoning (Text)", attempts decomposition into sub-queries.
    - **Retrieve**: query_files() is called for the original query or *each sub-query*. Results are merged and de-duplicated.
    - **Aggregate Context**: Uses the (potentially expanded) retrieval results.
    - **Query LLM**: The **original user query** (not sub-queries) is sent to the LLM with the aggregated context.
  - **Streamlit UI:**
    - "Query Analysis" section added to display query classification and any generated sub-queries.

Version 3.4: ragtest_c2-4.py (Internal: ragtest_c4_dataframe.py) - DataFrame Querying via LLM-Generated Pandas Code

- **Core Theme:** Added capability to query structured data (CSV/XLSX) by having the LLM generate and execute Pandas code.
- **Key Changes:**
  - **RAGConfig:**
    - Default LLM updated (e.g., to llama3-70b-8192).
    - New parameter: dataframe_query_confidence_threshold (default 0.8).
  - **RAGSystem - DataFrame Handling:**
    - self.dataframes: New dictionary to store loaded Pandas DataFrames.
    - process_files(): Now loads CSV/XLSX files into self.dataframes. Text representations are still created for these files for FAISS indexing (used for target DF identification).
  - **RAGSystem - Query Processing:**
    - _classify_query(): Updated to include a "Structured Query (DataFrame)" category and return a confidence score.

        - _generate_pandas_code(): New method; prompts LLM to generate Pandas code based on query and DataFrame columns.
        - run_query():
            - If query is classified as "Structured Query (DataFrame)" with sufficient confidence:
                1. **Target DataFrame ID**: Uses text RAG on the query to find the most relevant text document, then heuristically links to a loaded DataFrame (e.g., by similar base filename).
                2. **Code Generation**: If target DF found, calls _generate_pandas_code().
                3. **Code Execution**: If code generated, uses exec() to run it (output captured via redirect_stdout). **Note: exec() introduces security risks.**
                4. Result formatted into the answer.
            -
            - **Fallback**: If DataFrame path fails, falls back to the text RAG pipeline.
    - **Streamlit UI:**
        - Displays answer_source ("DataFrame" or "Text RAG").
        - "Query Analysis" section shows DataFrame target and an expander for executed Pandas code.
        - "Supporting Evidence" shows DataFrame preview if applicable.

Version 3.5: ragtest_c2-5.py - DataFrame Path Specialization & Result Framing

- **Core Theme:** Refined structured data handling: CSV/XLSX files are now *exclusively* for DataFrame querying (no text RAG over their content). DataFrame results are framed into natural language by an LLM.
- **Key Changes:**
    - **RAGSystem - Processing:**
        - extract_content(): No longer extracts text from CSV/XLSX for FAISS. These files are only loaded as DataFrames.
        - process_files(): Reflects this change; CSV/XLSX only contribute to self.dataframes.
    - **RAGSystem - Querying:**
        - _frame_dataframe_answer(): New method that prompts the LLM to convert the raw string output of Pandas code into a user-friendly natural language sentence.
        - run_query(): After successful Pandas code execution, calls _frame_dataframe_answer() to generate the final answer. answer_source becomes "DataFrame (Framed)".
    - **Bug Fix:** Corrected the exec() call in run_query to pass execution_globals as the globals argument.
    - **LLM Prompts:** Minor refinements in _generate_pandas_code for column name handling.

Version 3.6: ragtest_c2-6.py (Internal: ragtest_c2-4_sql.py) - SQLite Backend & DeepSeek for Metadata

- **Core Theme:** Replaced in-memory Pandas querying with a persistent SQLite database for structured data. Introduced DeepSeek API for generating column metadata (descriptions, types).
- **Key Changes:**
    - **RAGConfig:**
        - sqlite_db_path added (defaults to in-memory).
        - deepseek_model and deepseek_api_key_config added.
    - **RAGSystem - Database & Metadata:**
        - SQLite connection (self.db_conn, self.db_cursor) established in __init__.
        - self.table_metadata, self.file_to_table_map for tracking SQL tables.
        - _get_safe_table_name(), _table_exists(): DB utility methods.
        - _get_column_metadata_deepseek(): New method using DeepSeek API to infer column descriptions and SQL types from DataFrame samples. Includes fallback to _infer_sql_type().
        - _load_df_to_sql(): New method to load DataFrames into SQLite tables using df.to_sql(), storing generated metadata.
    - **RAGSystem - Processing:**
        - _extract_and_load_xlsx(), _extract_and_load_csv(): Now load data, get metadata via DeepSeek, and load into SQL tables.
    - **RAGSystem - SQL Query Path:**
        - _identify_target_sql_table(): New method; LLM chooses the most relevant SQL table based on query and schemas (with DeepSeek descriptions).
        - _generate_sql_query(): New method; LLM generates SQLite query for the target table.
        - _execute_sql_query(): Executes SQL, formats results.
        - _synthesize_answer_from_sql(): LLM frames a natural language answer from SQL results.
    - **run_query():**
        - If "Structured Query (SQL)" path is taken, it now uses the identify table -> generate SQL -> execute SQL -> synthesize answer pipeline. Pandas exec() path is removed.
        - Text RAG remains the fallback.
    - **Other**: _infer_sql_type() refined for better numeric type detection.

Version 3.7: ragtest_c2-7.py - Conversational Context (Chat History)

- **Core Theme:** Enabled multi-turn conversational capabilities by incorporating chat history into LLM interactions.
- **Key Changes:**
    - **RAGConfig:**
        - max_chat_history_turns parameter added.

        - LLM model for Groq updated to meta-llama/llama-4-scout-17b-16e-instruct.
    - **RAGSystem - Chat History Handling:**
        - _format_chat_history_for_llm(): Converts Streamlit's chat history format to Langchain HumanMessage/AIMessage objects, cleaning assistant message footers.
        - _refine_query_with_history(): New method; LLM rewrites the current user query to be standalone by incorporating context from previous turns. Uses ChatPromptTemplate with MessagesPlaceholder.
        - query_llm_with_history() (replaces query_llm for text RAG): Takes chat_history, uses ChatPromptTemplate with MessagesPlaceholder to provide conversational context to the LLM.
        - _synthesize_answer_from_sql_with_history() (replaces _synthesize_answer_from_sql): Also takes chat_history and uses ChatPromptTemplate for contextual SQL result synthesis.
    - **RAGSystem - run_query_with_history() (replaces run_query):**
        - Orchestrates the new flow:
            1. Refine current query using chat history.
            2. Classify refined query.
            3. Route to SQL path (using refined query) or Text RAG path (using refined query/sub-queries).
            4. All LLM calls for answer generation/synthesis now use history-aware methods.
    - **Streamlit UI:**
        - Transformed into a chat interface (st.chat_message, st.chat_input).
        - st.session_state.messages stores the conversation.
        - "New Chat" button clears history.
        - Analysis details displayed in an expander under the last assistant message.
        - Document processing UI moved to a sidebar expander.

Version 3.8 & 3.9: ragtest_c2-8.py & ragtest_c2-9.py - PDF Structural Awareness & UI Enhancements

- **Core Theme:** Introduced heuristics for understanding PDF document structure and significantly polished the Streamlit UI.
- **Key Changes (v3.8):**
    - **RAGSystem - PDF Processing (_extract_pdf):**
        - Added heuristics to identify potential_section tags (e.g., "title_page", "table_of_contents", "main_content", "introduction_overview", "closing_appendix") within PDFs based on page position and keyword matching.
        - Attempts to extract a potential_doc_title from the first page.
        - Tags (potential_section) are stored with extracted PDF content blocks.
        - Table metadata from PDFs now includes table_index_on_page.

- **RAGSystem - Chunking & Context:**
  - chunk_content(): Propagates the potential_section tag to individual chunks.
  - aggregate_context(): Includes the Section: {potential_section} information in the context header for PDF-derived chunks.
- **LLM Prompts:**
  - System prompts for all LLM interactions (classification, decomposition, SQL generation, answer synthesis, query refinement) updated with "SatSure LLM" branding and instructions specific to "SatSure project proposal documents."
  - Reinforced the use of the specific "I don't really know but my next versions will be able to answer this for sure!" fallback response.
- **Streamlit UI:**
  - **CSS Overhaul:** Extensive custom CSS added via st.markdown for a dark theme, custom fonts (Inter), SatSure-esque color scheme (blue primary, teal accent), improved chat message styling, expander appearance, scrollbars, and spinner color.
  - **JavaScript for Colors:** Included JS to dynamically set CSS RGB variables from hex/rgb codes for rgba() transparency.
  - **Document Management UI**:
    - Refactored display_processing_ui to use st.sidebar.expander for document management, improving layout.
    - More robust use of st.session_state for managing processing status messages and file lists within the sidebar expander.
  - **Chat Display**: Assistant message footers styled distinctly.
- **Configuration (RAGConfig)**: Minor adjustments to default PARAMETERS (chunk size, overlap, k_retrieval, max_context_tokens, temperature).
- *Version 3.9 (ragtest_c2-9.py) was identical to Version 3.8, indicating no code changes between them.*

Version 4.0: ragtest_c2-10.py (Final Version) - Switched to SambaNova LLM

- **Core Theme:** Replaced Groq (for general LLM tasks) and DeepSeek (for metadata) with SambaNova as the unified LLM provider.
- **Key Changes:**
  - **RAGConfig:**
    - Configuration file default changed to config1.ini.
    - MODELS section updated:
      - llm_model (Groq) and deepseek_model removed.
      - sambanova_model (default: "Meta-Llama-3.3-70B-Instruct") and sambanova_base_url added.
    - PARAMETERS section updated:
      - top_p parameter added for SambaNova (default 0.1).
      - max_output_tokens_sambanova added (default 1024).

        - API_KEYS section updated for sambanova_api_key_config (or SAMBANOVA_API_KEY env var).
    -
    - **RAGSystem - LLM Interaction:**
        - **Client Initialization (_get_llm_client)**: Now initializes an openai.OpenAI client configured with SambaNova API key and base URL. The langchain_groq import is removed.
        - **Message Formatting (_convert_langchain_messages_to_samba_format)**: New helper to convert Langchain SystemMessage, HumanMessage, AIMessage objects (still used for internal prompt templating) into the list of dictionaries format [{"role": ..., "content": ...}] expected by the OpenAI-compatible SambaNova API.
        - **LLM Calls (_call_llm_for_analysis, query_llm_with_history, _synthesize_answer_from_sql_with_history, _refine_query_with_history, _get_column_metadata_sambanova, _identify_target_sql_table, _generate_sql_query)**:
            - All methods that previously called Groq or DeepSeek are rewritten to use the SambaNova client (client.chat.completions.create(...)).
            - Prompts (often constructed using Langchain SystemMessage/HumanMessage internally for structure) are converted to the SambaNova API's expected message format.
            - Error handling updated for openai.APIError.
        -
        - _get_column_metadata_sambanova(): Replaces the DeepSeek version, now uses SambaNova to generate column metadata (descriptions and SQL types) for spreadsheets.
    - **Dependencies:** openai library is now essential. requests is noted as no longer needed for DeepSeek.
    - **Streamlit UI:**
        - Updated titles and captions to reflect "SambaNova".
        - Sidebar configuration display now shows SambaNova model details.
        - API key check in load_rag_system now targets SAMBANOVA_API_KEY.
    - **Minor Refinements:**
        - Improved duplicate column name handling in _extract_and_load_xlsx and _extract_and_load_csv.
        - More robust data type conversion in _load_df_to_sql before loading into SQLite.
        - Enhanced logic in _get_column_metadata_sambanova for parsing JSON from LLM and handling missing/case-insensitive column names.