Retrieval-Augmented Generation (RAG) System Documentation

May 7, 2025

Abstract

The Retrieval-Augmented Generation (RAG) System is a comprehensive solution designed to process and query structured and unstructured data, including documents (PDF, DOCX), Excel files, and web content. By integrating advanced retrieval mechanisms, natural language processing, and a user-friendly interface, the system enables efficient information retrieval and response generation. This document provides an in-depth overview of the system's architecture, features, installation, usage, and technical details, serving as a guide for stakeholders and developers.

Contents

1 Introduction

The Retrieval-Augmented Generation (RAG) System is an innovative application that leverages large language models (LLMs), vector-based retrieval, and SQL query translation to provide accurate and context-aware responses to user queries. The system supports three primary data sources: documents, Excel-based SQL tables, and web content, making it versatile for various use cases such as research, data analysis, and information retrieval.

1.1 Purpose

This document aims to provide a comprehensive guide to the RAG System, detailing its functionality, setup, usage, and technical architecture. It serves as a reference for project managers, developers, and end-users to understand and utilize the system effectively. The documentation is structured to support both technical and non-technical stakeholders, with detailed explanations of system components and straightforward usage instructions.

1.2 Scope

The documentation covers:

- **System overview and key features**: A detailed exploration of the system's capabilities and distinctive elements
- **Installation and configuration instructions**: Step-by-step guide to setting up the system locally or in a production environment
- **User guide for interacting with the system**: Comprehensive instructions for day-to-day usage and operations
- **Technical architecture and component details**: In-depth explanation of system design, data flow, and component interactions
- **API endpoints and dependencies**: Thorough documentation of all available API endpoints and external dependencies
- **Maintenance and troubleshooting guidelines**: Practical advice for system upkeep and problem resolution

This documentation assumes a basic understanding of data processing systems and natural language applications but provides sufficient detail for users at all technical levels.

2 System Overview

The RAG System integrates multiple components to process and query data efficiently. It combines document indexing, Excel-to-SQL conversion, and web search capabilities, orchestrated through a FastAPI backend and a Streamlit frontend. The system's design prioritizes accuracy, speed, and user experience, making complex information retrieval tasks accessible through natural language queries.

2.1 Key Features

- **Document Processing:**
  The system supports ingestion of PDF and DOCX documents. It extracts textual content, generates embeddings using HuggingFace models, and stores them in a FAISS vector index. To improve retrieval quality, it employs a hybrid approach combining semantic search with BM25-based reranking.
- **Excel to SQL Conversion:**
  Excel files are parsed and transformed into structured SQLite databases. The system enables users to query these datasets using natural language, which is converted into SQL queries under the hood. This allows non-technical users to extract insights without knowing SQL.
- **Web Search Integration:**
  For queries requiring up-to-date or external information, the system uses the Tavily API to perform web searches. The retrieved content is processed, summarized, and optionally incorporated into the response generation pipeline, ensuring relevance and timeliness.
- **Response Caching:**
  Frequently repeated queries are cached at the response level to reduce latency and API load. The caching mechanism ensures quicker turnaround for common or repeated questions while maintaining up-to-date results for new queries.
- **Evaluation Framework:**
  Integrated with the Opik evaluation toolkit, the system measures the quality of responses using metrics such as hallucination detection and context precision. These evaluations help maintain high standards of accuracy, reliability, and trustworthiness.
- **Interactive User Interface:**
  The Streamlit frontend provides an intuitive user experience for uploading documents, managing table schemas, issuing queries, and viewing results. It bridges the gap between powerful backend capabilities and user-friendly access.

2.2 Use Cases

- **Research Teams:**
  Ideal for organizations and academic groups that need to extract, synthesize, and cross-reference information from complex technical documents quickly and efficiently.
- **Data Analysts:**
  Enables analysts to query Excel datasets in natural language, turning data exploration into a conversational and accessible process.
- **General Users:**
  Supports real-time web-based information retrieval, making it suitable for journalists, students, or professionals looking for quick and contextual answers without manual searching.

3 Installation and Setup

This section guides you through setting up the RAG system on your local machine. The setup process includes installing necessary dependencies, configuring environment variables, and running the backend and frontend components.

3.1 Prerequisites

Before you begin, ensure that the following tools and accounts are available:

- Python 3.8 or higher – Required for running the backend and core logic.
- Git – To clone the repository.
- Virtual Environment – Recommended to isolate project dependencies.
- API Keys – You will need active API keys for the following services:
    - Tavily (Web search)
    - Groq (LLM API)
    - Opik (Response evaluation)

3.2 Installation Steps

1. Clone the Repository
    1.1. git clone https://github.com/harsha-joshi02/SDS-RAG.git
    1.2. cd SDS-RAG
2. Set Up a Virtual Environment
    2.1. For macOS/Linux:
        2.1.1. python3 -m venv venv
        2.1.2. source venv/bin/activate
    2.2. For Windows:
        2.2.1. python -m venv venv
        2.2.2. venv\Scripts\activate
3. Install Project Dependencies
    3.1. pip install -r requirements.txt
4. Configure Environment Variables (Create a .env file in the root directory of the project and add your API credentials)
    4.1. TAVILY_API_KEY = your-tavily-api-key
    4.2. GROQ_API_KEY = your-groq-api-key
    4.3. OPIK_API_KEY = your-opik-api-key
    4.4. API_URL = http://localhost:8000

5. Verify Configuration Settings
    5.1. Ensure that config.yaml is properly set up with paths and model configuration details.

3.3 Running the System

You can start the system using the provided Makefile commands:

1. Optional Cleanup: Clean any temporary files or logs
    1.1. make cleanup
2. Start Backend Server: Launches the FastAPI backend
    2.1. make run_backend
3. Start Frontend Interface: Launches the Streamlit interface
    3.1. make run_frontend
4. Start both Backend and Frontend: Runs the entire system in one step
    4.1. make all

4 User Guide

4.1 Accessing the System

Navigate to http://localhost:8501 in a web browser to access the Streamlit frontend after starting the system.

4.2 Uploading Files

- **Documents**: Upload PDF or DOCX files via the “Upload” tab. Files are saved in the data directory and indexed for querying.
- **Excel Files**: Upload .xlsx or .xls files. Each sheet is converted to a SQLite table. Assign a schema name to group tables.

4.3 Querying the System

- **Document Query**: Select a document and ask questions. Responses include citations from retrieved chunks.
- **SQL Query**: Choose a schema and input natural language queries. The system translates these to SQL and executes them.
- **Web Search**: Select “Web Search” for queries answered using web content. Responses include source URLs.
- **Evaluation Metrics**: Toggle “Show Evaluation Metrics” to view hallucination and context precision scores.

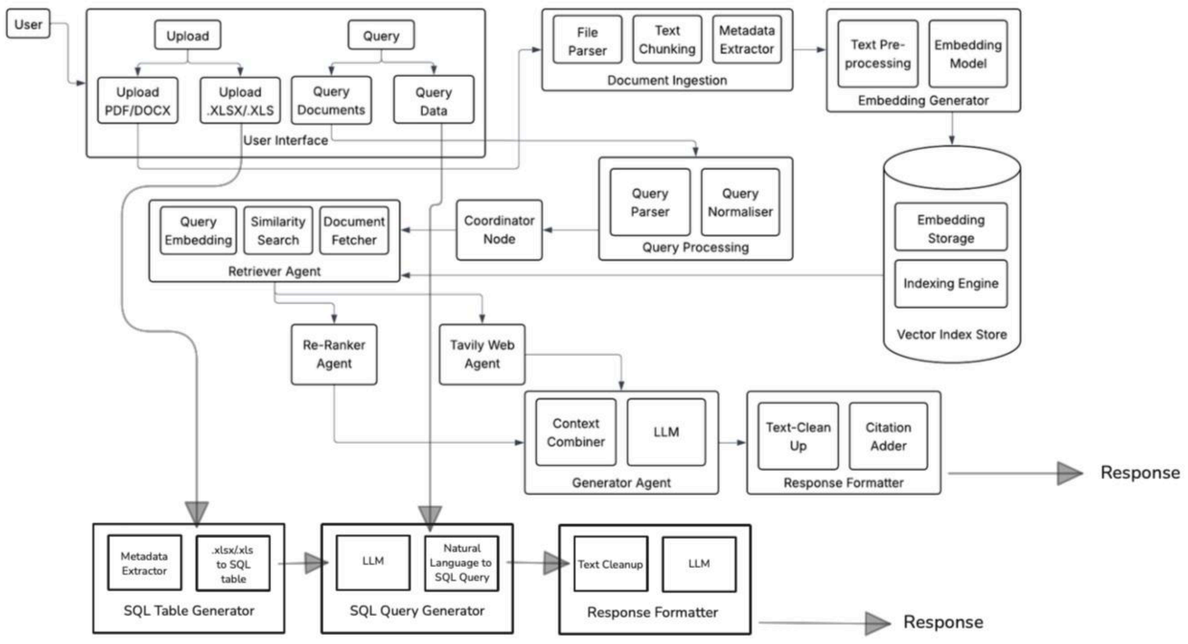
4.4 Viewing Results

- Answers are displayed with citations (for documents) or web sources.
- Metrics are shown in an expandable section if enabled.
- Chat history is maintained separately for each query type.

5 Technical Architecture

The RAG System is built using a modular architecture, with components handling specific tasks such as retrieval, query processing, and response formatting.

5.1 Architecture



5.2 Components

- **RAG System** **(rag.py)**:
  Handles document indexing using FAISS and HuggingFace embeddings, retrieves relevant chunks, and generates responses using the Groq LLM.
- **Excel Processor** **(excel_processor.py)**:
  Converts Excel sheets into SQLite tables and translates natural language queries into executable SQL.
- **Web Search Agent** **(web_search.py)**:
  Integrates with the Tavily API to fetch real-time web content and uses Groq LLM to process and summarize it.
- **Cache** **(cache.py)**:
  Implements a caching mechanism for document, SQL, and web queries with a configurable time-to-live (TTL) to improve performance.
- **Evaluation System** **(evaluation.py)**:
  Uses Opik and Sentence-Transformer to compute evaluation metrics such as hallucination and context precision for generated response

- **Frontend (frontend.py)**:
  Provides a Streamlit-based user interface for uploading files, querying data, and visualizing results.
- **Backend (main.py)**:
  Implements a FastAPI server that handles all API requests and orchestrates interactions between components.

5.3 Workflow

1. **Document Query**:
   The system checks the query against the cache. If not cached, it retrieves relevant chunks using FAISS, reranks them with BM25, and generates a response using Groq LLM.
2. **SQL Query**:
   Natural language queries are translated to SQL using Groq, executed against the corresponding SQLite tables, and the results are formatted with LLM output.
3. **Web Search**:
   Tavily API fetches relevant web data. The content is processed and summarized by Groq LLM, then stored in cache for future queries.
4. **Evaluation**:
   Responses are evaluated using Opik for hallucination and context precision. The results are saved in the evaluations directory for review.

5.4 API Endpoints

- **POST** /upload-sds/ – Upload documents for indexing.
- **POST** /upload-excel/ – Upload and process Excel files into SQLite tables.
- **POST** /set-schema/ – Assign a schema name to a processed table.
- **GET** /excel-tables/ – Retrieve metadata about stored tables.
- **POST** /query/ – Query indexed documents via natural language.
- **POST** /sql-query/ – Query structured Excel data using natural language.
- **POST** /web-search/ – Perform real-time web searches and return results.

6 Dependencies

The RAG system leverages several core Python libraries and tools to manage document processing, natural language querying, data retrieval, and evaluation. Below are the primary dependencies:

- fastapi – High-performance web framework used to build the backend API.
- streamlit – Interactive frontend framework for building intuitive web interfaces.
- langchain – Framework for orchestrating large language model (LLM) workflows.
- groq – Interface for accessing Groq’s high-speed LLM APIs.
- tavily-python – Client library to access Tavily’s real-time web search API.
- faiss-cpu – Vector store library used for efficient similarity search and indexing.
- pandas – Data processing library, primarily used for reading and manipulating Excel data.
- opik – Evaluation framework that provides hallucination detection and context precision metrics.
- sentence-transformers – Used for generating and comparing text embeddings for retrieval and evaluation.

A full list of dependencies is available in the requirements.txt file in the project root directory

7 Maintaining and Troubleshooting

This section provides guidance on routine system maintenance and how to resolve common issues that may arise during usage.

7.1 Maintenance

1. Cleanup: Use the following command to remove temporary files, evaluation logs, and SQLite databases-
    a. `make cleanup`
2. Logs: Refer to log files for diagnostics and error tracking-
    a. `rag_system.log` – Backend and document processing log.
    b. `frontend.log` – Logs related to user interactions and frontend behaviour.
3. Configuration Updates: Modify config.yaml to update settings such as-
    a. Model names and parameters (e.g., LLM model, embedding model)
    b. Upload directory paths
    c. Retrieval and reranking settings

7.2 Common Issues

- API Key Errors:
    - Ensure all required API keys (Tavily, Groq, Opik) are correctly set in the `.env` file.
    - Keys must be valid and active. Restart the server after updating keys.
- FAISS Index Errors:
    - If retrieval fails or returns incorrect results, try deleting the `faiss_index/` directory.
    - Re-upload documents to rebuild the index from scratch.
- Excel Processing Errors:
    - Ensure the Excel file is in .xlsx format.
    - Check that the sheet structure is consistent, with clear column names and tabular data.
- Web Search Failures:
    - Verify that your Tavily API key is active and has remaining quota.
    - Ensure the machine has a stable internet connection.

8 Conclusion

The RAG System offers a powerful, end-to-end framework for retrieving and generating responses from a wide range of data sources, including documents, structured Excel datasets, and real-time web content. By combining vector search (FAISS), hybrid reranking (BM25), natural language-to-SQL conversion, and LLM-powered generation through Groq, it delivers high-quality answers that are contextually accurate and timely.

Its modular architecture ensures that each component—from document indexing to evaluation—is independently maintainable and extensible. This design allows users to adapt the system to their specific workflows, whether they are part of a research team analyzing technical documents, a business analyst querying spreadsheets, or a general user seeking real-time information.

The interactive Streamlit frontend and the FastAPI backend make the system both developer-friendly and accessible to non-technical users. Additionally, integrated caching and evaluation features further enhance reliability and performance, ensuring responses are both fast and accurate.

8.1 Future Directions

To make the system even more versatile and intelligent, potential enhancements may include:

- **Support for Additional File Formats:** Expanding compatibility to include HTML, JSON, Markdown, and scanned image PDFs via OCR.
- **Advanced Evaluation Metrics:** Incorporating factual consistency checks, confidence scoring, and user feedback loops for continuous learning.
- **Multi-LLM Support:** Allowing dynamic routing between different LLMs (e.g., OpenAI, Claude, Gemini) based on use case, cost, or performance requirements.
- **Multi-modal Capabilities:** Extending support to handle images, charts, and audio inputs for richer interactions.
- **Fine-Tuning & Custom Training:** Enabling domain-specific adaptations of LLM behavior and retrieval strategies.

In summary, the RAG System is a flexible, scalable, and forward-thinking solution for intelligent data querying and information synthesis. It lays a solid foundation for future innovation and can evolve alongside advances in language models, retrieval algorithms, and user interface technologies.