Answer IQ: Retrieval-Augmented Knowledge Search System

1. Abstract

RAG-Gemini is an intelligent knowledge base assistant built using Retrieval-Augmented Generation (RAG) principles. The system enables users to upload unstructured data in the form of PDF or text documents and interact with the data through natural language queries. It integrates document retrieval using semantic search and response generation using transformer-based language models, resulting in precise, context-aware answers.
This project demonstrates the integration of Natural Language Processing (NLP), vector embeddings, and transformer-based generation in a unified Streamlit web application.

2. Introduction

Modern enterprises handle vast volumes of textual data that are often unstructured and difficult to query effectively. Traditional keyword-based search methods fail to capture semantic meaning, leading to inefficient information retrieval.
RAG-Gemini addresses this challenge by combining information retrieval with generative reasoning. It retrieves the most relevant document chunks based on vector similarity and generates concise or detailed answers using a transformer language model.
The project serves as a foundation for building intelligent, domain-specific knowledge assistants capable of functioning entirely offline without relying on external APIs.

3. Objectives

The primary objectives of the project are:

1. To enable users to upload and process multiple PDF or text documents for knowledge base creation.
2. To generate vector embeddings for document content using transformer-based embedding models.
3. To perform semantic retrieval of relevant information from documents using FAISS (Facebook AI Similarity Search).
4. To generate contextually relevant responses to user queries using a lightweight local language model.
5. To support both text-based and voice-based input for user interaction.
6. To provide a Streamlit-based interface that is user-friendly and efficient for real-time information exploration.

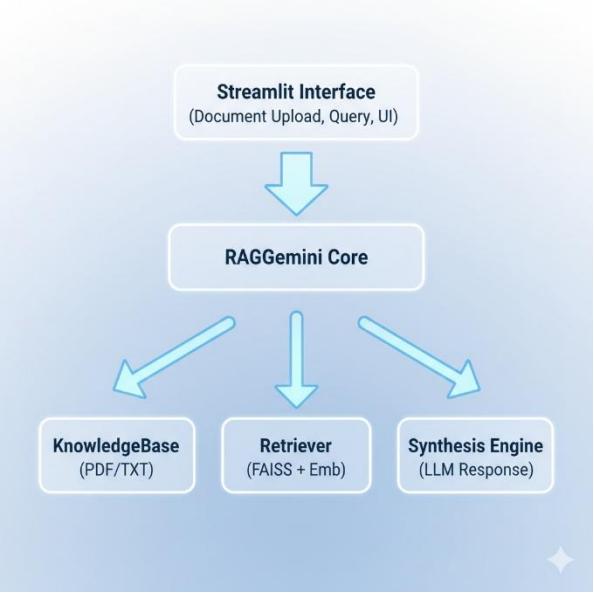
4. System Overview

The system architecture follows the Retrieval-Augmented Generation (RAG) paradigm, consisting of three core components:

1. **Knowledge Base Construction:** Responsible for document ingestion, text segmentation, and embedding generation.

2. **Retriever Module:** Utilizes FAISS to perform similarity searches over document embeddings.
3. **Synthesis Engine:** Employs a transformer model to generate natural language responses from retrieved context.

5. Architecture Diagram



6. Technologies Used

Component	Technology/Library
Frontend	Streamlit
Backend	Python
Document Loading	LangChain Document Loaders (PyPDFLoader, TextLoader)
Text Chunking	RecursiveCharacterTextSplitter
Vector Database	FAISS (Facebook AI Similarity Search)
Embeddings	Sentence-Transformers (all-MiniLM-L6-v2)
Large Language Model (LLM)	DistilGPT2
Voice Transcription	Whisper (Tiny model)
Data Storage	JSON (Local persistence)
Dependencies	torch, transformers, langchain, faiss-cpu, sentence-transformers, streamlit, openai-whisper

7. Implementation Details

7.1 Document Upload and Processing

Documents are uploaded via the Streamlit interface. The system supports PDF and text files. LangChain loaders read and parse content, which is then segmented into overlapping chunks (400 characters with 80-character overlap) to enhance semantic embedding quality.

7.2 Embedding and Vector Storage

Each chunk is converted into a dense vector using the sentence-transformers/all-MiniLM-L6-v2 model. These embeddings are stored in FAISS for efficient similarity-based retrieval. This step forms the foundation of the knowledge base.

7.3 Query Processing

When a user submits a query:

1. The query is converted into a vector using the same embedding model.
2. The retriever searches the FAISS database for the top-k semantically similar chunks (k=4 by default).
3. The retrieved context is passed to the language model for synthesis.

7.4 Answer Generation

The synthesis module uses a lightweight model (distilgpt2) to generate answers based on the provided context. The system supports both concise and detailed response modes depending on the user's preference.

7.5 Voice Querying

Users can upload an audio file in formats such as .wav, .mp3, or .flac. The system employs Whisper (Tiny model) to transcribe audio into text, which is then processed through the standard retrieval and generation pipeline.

7.6 Query History

Every interaction is logged in a local JSON file containing the timestamp, query, and generated response. The interface provides a "History" section to review recent interactions.

8. User Interface Design

The Streamlit interface is organized into four main tabs:

Tab	Function	Description
Build KB	Upload documents and build the knowledge base.	Handles embedding and FAISS index creation.
Ask	Text-based Q&A interface.	Retrieves context and generates LLM responses.

Tab	Function	Description
Voice	Voice-based interaction.	Transcribes audio and retrieves information.
History	Displays query logs.	Allows review of recent queries and answers.

9. Execution Guide

Step 1: Install Dependencies

pip install streamlit torch transformers langchain langchain-community faiss-cpu sentence-transformers openai-whisper

Step 2: Run the Application

streamlit run KBSA-unthinkable.py

Step 3: Using the Application

1. Navigate to the “Build KB” tab and upload the desired documents.
2. Click “Build Knowledge Base” to process and embed the documents.
3. Proceed to the “Ask” tab to submit queries and receive context-aware answers.
4. Optionally, use the “Voice” tab to upload an audio file for transcription-based querying.
5. Review past interactions in the “History” tab.

10. Evaluation

The system achieves fast response times due to FAISS’s efficient nearest-neighbor search. Using the all-MiniLM-L6-v2 embedding model provides a balance between performance and computational cost. The distilgpt2 model, though small, produces coherent and contextually relevant answers for general-purpose document sets.
For enhanced accuracy, larger LLMs or domain-specific embeddings can be integrated in future versions.

11. Limitations

1. The base model (distilgpt2) may produce limited contextual depth for complex queries.
2. The current FAISS implementation operates locally and may require adaptation for large-scale deployments.
3. Whisper transcription accuracy depends on audio quality.
4. The system does not include real-time conversational memory beyond recent query history.

12. Future Enhancements

- Integration of advanced language models such as Mistral, LLaMA 3, or Gemma.
- Support for scalable vector stores such as Pinecone or ChromaDB.
- Implementation of long-term memory through LangChain conversation buffers.
- Inclusion of multilingual support for both document ingestion and querying.
- Docker-based deployment on cloud platforms for enterprise use.

13. Conclusion

RAG-Gemini demonstrates the capability of combining information retrieval and generative AI into a cohesive knowledge exploration framework. By leveraging open-source models and local computation, it provides a cost-effective and privacy-preserving solution for building interactive document-based assistants. The system serves as a robust foundation for enterprise knowledge management, educational applications, and AI-driven research tools.