RAG Workflow Overview

A Step-by-Step Guide to Implementing Retrieval-Augmented Generation (RAG)

Step 1: Set up Environment

- - Import required libraries
- - Load API key from .env file
- - Determine LLM version based on date
- - Example: GPT-3.5 Turbo or GPT-4

Step 2: Load Document and Create VectorDB

- - Use OpenAI Embeddings for vectorization
- - Set up Chroma as the vector database
- - Persist directory: `docs/chroma/`
- - Example: `Chroma` with OpenAI embeddings

Step 4: Create LLM

- - Use `ChatOpenAI` class from LangChain
- - Specify model name and temperature
- - Example: `llm = ChatOpenAI(model_name=llm_name, temperature=0)`
- - Verify LLM response: `llm.predict('Hello world!')`

Step 5: Conversational Retrieval Chain

- - Create a retriever from the vector database
- - Set up memory with `ConversationBufferMemory`
- - Combine LLM, retriever, and memory into a QA chain
- - Example: `ConversationalRetrievalChain.from_llm(...)`

Step 6: Build Chatbot on Documents

- - Use text loaders for PDF or text files
- - Split text with `CharacterTextSplitter` or similar tools
- - Create a chatbot class to manage conversations
- - Example: Class `cbfs` handles chat logic

Step 7: Speech Integration

- - Use libraries like `speech_recognition`, `pydub`, and `whisper`
- - Implement wake word detection for interactions
- - Transcribe audio inputs with Whisper models
- - Example: `transcribe_forever` and `reply` functions

Conclusion

- - RAG combines LLMs with document retrieval for effective QA
- - Steps include environment setup, vector DB creation, and chatbot development
- - Integrate speech for enhanced user interaction
- - Tools used: OpenAI, LangChain, Whisper, and others