PolyglotRAG System Architecture Design

This document outlines the architecture for the Multilingual PDF Q&A RAG System with Real-Time Translation, based on the clarified user requirements.

1. Overview

The system allows users to upload PDF documents, ask questions in various supported languages, and receive answers translated into their query language. It leverages Retrieval-Augmented Generation (RAG) with multilingual capabilities, real-time translation, and efficient language models.

2. Goals

- Support 10+ languages (including English, Hindi, Spanish, French, German, Arabic, Chinese, Japanese, Tamil, Bengali).
- Process user-uploaded PDFs (up to 50MB).
- Provide a web-based chat interface for interaction.
- Utilize open-source translation (NLLB-200) and efficient LLMs (Groq LLaMA 3/ Mistral).
- Employ multilingual embeddings (`intfloat/multilingual-e5-large`) and a hybrid vector store (Pinecone).
- Achieve fast response times (< 2 seconds for typical queries).
- Deliver the system as a Dockerized package.

3. Architecture Components

The system comprises the following key components:

1. **Frontend (Web Interface):**
2. **Technology:** Streamlit (for rapid prototyping and demo) or FastAPI backend with a separate JavaScript frontend (e.g., React, Vue) for a more complex UI.

3. **Responsibilities:** Handles user interactions, including PDF uploads, language selection (input/output), query input, and displaying streaming responses.
4. **Backend (API Server):**
5. **Technology:** FastAPI (chosen for performance, async capabilities, and scalability).
6. **Responsibilities:** Manages API endpoints for file uploads, queries, and potentially other integrations. Orchestrates the overall workflow.
    - **Key Endpoints:**
    - `/upload`: Accepts PDF files, triggers the ingestion pipeline.
    - `/query`: Receives user questions, source/target languages, initiates the RAG and translation process, streams results.
7. **PDF Processing Module:**
8. **Technology:** `poppler-utils` (specifically `pdftotext`) via shell commands.
9. **Responsibilities:** Extracts raw text content from uploaded PDF files. Includes basic error handling for corrupted or unreadable PDFs.
10. **Text Chunking Module:**
11. **Technology:** LangChain text splitters (e.g., `RecursiveCharacterTextSplitter`, potentially customized for multilingual context and semantic boundaries).
12. **Responsibilities:** Divides the extracted PDF text into smaller, manageable, and contextually relevant chunks suitable for embedding and retrieval.
13. **Translation Module:**
14. **Technology:** Hugging Face `transformers` library with the NLLB-200 model (local execution or dedicated API). Fallback to the core LLM's multilingual capabilities if NLLB proves too resource-intensive or slow.
15. **Responsibilities:** Translates user queries from their native language to English (the primary internal processing language). Translates the final generated English answer back to the user's query language.
16. **Embedding Module:**
17. **Technology:** `sentence-transformers` library with the `intfloat/multilingual-e5-large` model.

18. **Responsibilities:** Generates dense vector embeddings for text chunks (processed in English) and the translated user query.
19. **Vector Database Module:**
20. **Technology:** Pinecone.
21. **Responsibilities:** Stores text chunks, their corresponding embeddings, and relevant metadata (e.g., document source, chunk ID). Provides efficient hybrid search capabilities (BM25 keyword search + dense vector similarity search).
22. **Retrieval Module (RAG Core):**
23. **Technology:** LangChain / LangGraph.
24. **Responsibilities:** Orchestrates the retrieval process. Takes the translated English query embedding, performs hybrid search against the Pinecone index using an Ensemble Retriever approach, retrieves relevant chunks, and applies a reranker (e.g., Cohere Rerank or a cross-encoder model) to improve context relevance.
25. **Generation Module (LLM):**
26. **Technology:** Groq API integration (using LLaMA 3-8B or Mistral 7B).
27. **Responsibilities:** Receives the original query, the translated query, and the reranked, relevant context chunks. Synthesizes this information to generate a coherent and accurate answer in English.
28. **Orchestration Layer:**
29. **Technology:** LangGraph integrated within the FastAPI backend.
30. **Responsibilities:** Manages the complex flow of data and control between all modules for both PDF ingestion and Q&A processes. Handles state management and potential asynchronous operations (e.g., background PDF processing).

4. Data Flow

A. PDF Ingestion:

1. User uploads PDF via Frontend.
2. Backend receives PDF, stores it temporarily.
3. PDF Processing Module extracts text.
4. Text Chunking Module splits text into chunks.

5. Assumption: Since internal processing is English and embeddings are multilingual, we embed chunks directly without pre-translation unless initial tests show poor cross-lingual retrieval.
6. Embedding Module generates embeddings for each chunk using `multilingual-e5-large`.
7. Vector Database Module stores chunks, embeddings, and metadata in Pinecone.

B. Query Processing:

1. User submits a question in Language X via Frontend.
2. Backend receives the query and Language X.
3. Translation Module translates the query from Language X to English.
4. Embedding Module generates an embedding for the translated English query.
5. Retrieval Module performs hybrid search in Pinecone using the query embedding and keywords, retrieves relevant chunks, and reranks them.
6. Generation Module (Groq LLM) receives the original query (Language X), translated query (English), and retrieved chunks to generate an answer in English.
7. Translation Module translates the English answer back to Language X.
8. Backend streams the translated answer (Language X) to the Frontend.

5. Technology Stack Summary

- **Frontend:** Streamlit / FastAPI + JS Framework
- **Backend:** FastAPI (Python)
- **PDF Extraction:** `poppler-utils`
- **Translation:** NLLB-200 (`transformers`)
- **Embeddings:** `intfloat/multilingual-e5-large` (`sentence-transformers`)
- **Vector DB:** Pinecone
- **RAG/Orchestration:** LangChain / LangGraph (Python)
- **LLM:** Groq API (LLaMA 3 / Mistral)

- **Deployment:** Docker