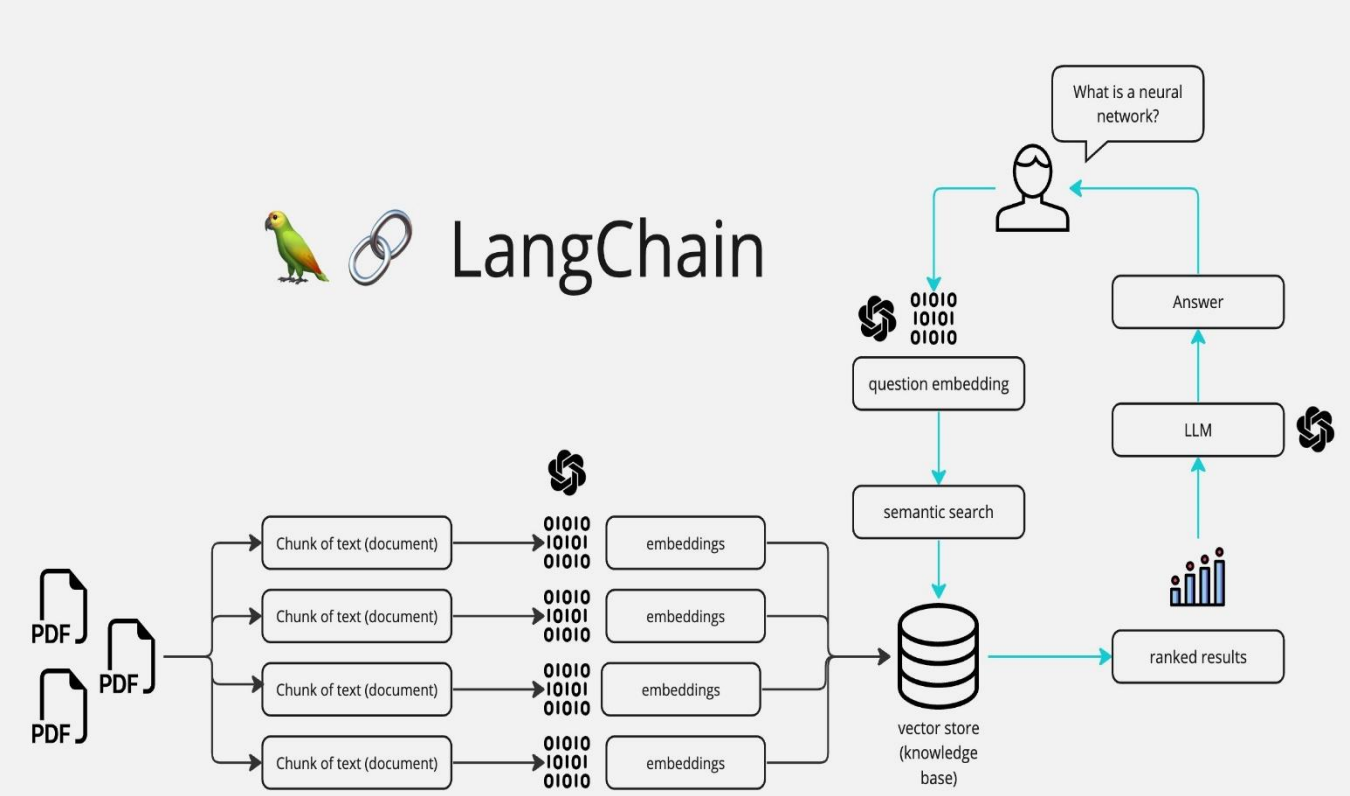
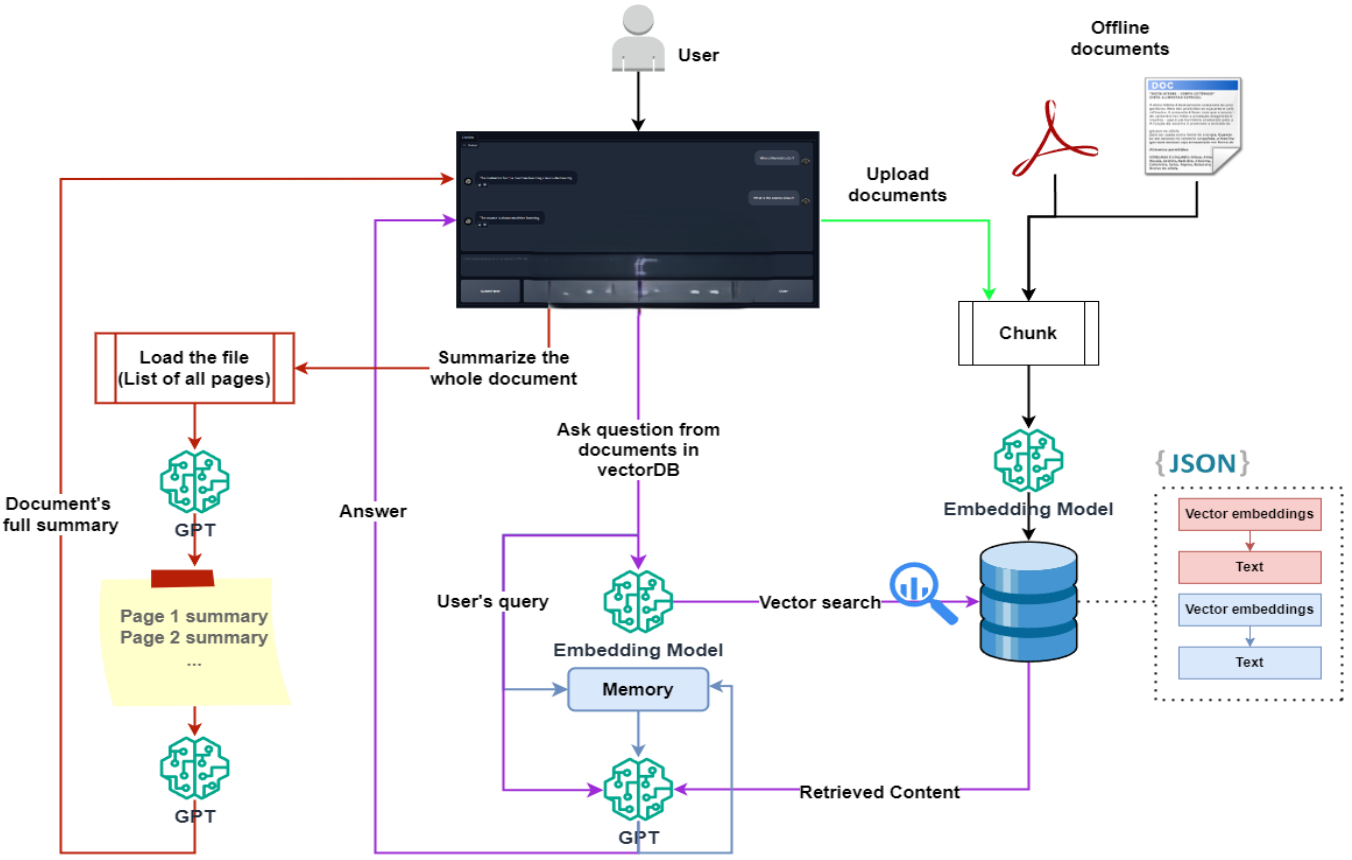
Diagram:


What is a neural network?
LangChain
Answer
question embedding
LLM
semantic search
PDF
PDF
PDF
Chunk of text (document)
Chunk of text (document)
Chunk of text (document)
Chunk of text (document)
embeddings
embeddings
embeddings
embeddings
vector store (knowledge base)
ranked results


User
Offline documents
Upload documents
Chunk
Load the file (List of all pages)
Summarize the whole document
Ask question from documents in vectorDB
Document's full summary
GPT
Answer
Embedding Model
{JSON}
Vector embeddings
Text
Vector embeddings
Text
Page 1 summary
Page 2 summary
...
User's query
Vector search
Embedding Model
Memory
GPT
Retrieved Content
GPT

High-Level Design (HLD)

1. System Overview

- This application enables users to upload PDF files, process their content using Google Generative AI embeddings, store the processed data in a vector store (FAISS), and interact with the documents via natural language queries.
- The system utilizes a Retrieval-Augmented Generation (RAG) approach to handle user questions, offering accurate and context-driven answers based on document content.

2. Key Components and Interactions

A. User Interface (UI) Layer

- **Streamlit UI**: Provides a user-friendly interface for file uploads, question input, and displaying responses.
- **Sidebar Menu**: Allows users to upload multiple PDFs and submit for processing.
- **Main Content Area**: Accepts user questions and displays AI-generated responses.

B. Application Logic Layer

- **PDF Text Extraction**: Extracts and concatenates text from each PDF page using PyPDF2.
- **Text Chunking**: Splits extracted text into manageable chunks for embedding, using LangChain's RecursiveCharacterTextSplitter.
- **Embedding Creation**: Generates embeddings from text chunks using Google Generative AI embeddings.
- **Vector Storage (FAISS)**: Stores embeddings in a FAISS index for similarity search, enabling efficient document retrieval.
- **Query Processing**: Uses a Retrieval-Augmented Generation (RAG) approach for handling user queries, with LangChain’s QA chain to generate responses.

C. Data Storage Layer

- **Vector Store (FAISS)**: Stores and retrieves document embeddings for similarity search, optimizing response relevance.

D. External Services

- **Google Generative AI**: Provides embeddings for document text and powers the RAG-based question-answering model.

Low-Level Design (LLD)

1. User Interface (UI) Layer

a. Sidebar Components

- **File Uploader (st.file_uploader)**: Allows users to upload multiple PDFs. When "Submit & Process" is clicked, files are passed to the backend for processing.
- **Submit Button (st.button)**: Initiates the PDF processing workflow.

b. Main Content Components

- **Question Input (st.text_input)**: Accepts user questions related to document content.
- **Display Response (st.write)**: Shows the AI-generated response after processing.

2. Application Logic Layer

a. PDF Text Extraction

- **Function**: get_pdf_text(pdf_docs)
- **Description**: Loops through each uploaded PDF, reads pages with PdfReader, and extracts text.
- **Error Handling**: Checks for missing or empty text extraction and alerts users if no content is retrieved.

b. Text Chunking

- **Function**: get_text_chunks(text)
- **Description**: Splits extracted text into chunks (10,000 characters with 1,000-character overlap) using RecursiveCharacterTextSplitter.
- **Purpose**: Manages large text size, ensuring efficient and accurate embedding generation.

c. Embedding Creation

- **Function**: get_vector_store(text_chunks)
- **Description**: Converts text chunks into embeddings with GoogleGenerativeAIEmbeddings, specifying the API key and embedding model.
- **Storage**: Saves embeddings in a FAISS vector store (faiss_index) for later retrieval.

d. Vector Storage (FAISS)

- **Vector Storage Creation**: Initializes and stores embeddings in a FAISS index.
- **Load Vector Store**: Loads FAISS index locally with the allow_dangerous_deserialization=True flag for query processing.

e. Query Processing

- **Conversational Chain Creation**:
    - **Function**: get_conversational_chain()
    - **Description**: Sets up the LangChain QA chain with a custom prompt, instructing the model to provide accurate answers or indicate if the answer is unavailable in the context.
- **User Query Handler**:
    - **Function**: user_input(user_question)
    - **Description**: Loads FAISS index, performs similarity search for relevant documents, and generates responses using the QA chain.

3. Data Storage Layer

- **FAISS Vector Store**:
    - **Purpose**: Holds embeddings for similarity search, enabling quick retrieval of relevant documents.
    - **Load and Save Functions**: Ensures efficient storage and retrieval across multiple user sessions.

4. External Services

- **Google Generative AI Embeddings**:
    - **Function**: GoogleGenerativeAIEmbeddings(api_key=api_key, model="models/embedding-001")
    - **Role**: Generates embeddings for document chunks to facilitate similarity search.
- **Google Generative AI for RAG Model**:
    - **Model**: ChatGoogleGenerativeAI(model="gemini-pro", temperature=0.3)
    - **Purpose**: Processes user questions and returns contextually accurate answers.

Detailed Workflow

1. **User Uploads PDF Files**:
    - User selects PDF files and clicks “Submit & Process”.
    - The system extracts text from PDFs using get_pdf_text.
2. **Text Chunking and Vector Store Creation**:

- Text is split into chunks with get_text_chunks and converted into embeddings with get_vector_store.
- Embeddings are stored in the FAISS index for future querying.

3. **User Enters a Question**:
   - User inputs a question, which triggers the user_input function.
   - FAISS index is loaded and queried using similarity search to retrieve relevant documents.
4. **Query Processing and Response Generation**:
   - Retrieved documents and user question are processed by the conversational chain (get_conversational_chain).
   - The response is generated and displayed to the user via Streamlit.

Conclusion

This HLD and LLD provide a comprehensive understanding of the application, including its major components, their roles, and interactions. The **high-level design** explains the general system architecture, while the **low-level design** offers specific details about each function and process, clarifying the step-by-step flow of the system.