Problem Explanation

Finding the right job can be time-consuming and overwhelming.
Job seekers often have to manually search multiple platforms like Google Jobs or LinkedIn, filter results based on location, role, and salary, and still may end up with irrelevant postings.

This project solves that problem by:

- Taking **natural language input** from the user (e.g., *"Data analyst job in Bengaluru with good salary package"*).
- Automatically retrieving relevant job listings from **Google Jobs** and **LinkedIn**.
- Using **LangChain + Gemini AI** to analyze and rank the results.
- Presenting the **top 3 best-matched jobs** with explanations and application links.

Use of RAG with LangChain + Gemini

This project uses **Retrieval-Augmented Generation (RAG)** to enhance the accuracy and relevance of job recommendations:

1. **Retrieval Step**
    - Job listings are fetched from **Google Jobs** or **LinkedIn** based on user queries.
    - The results are stored in a temporary **vector database** for semantic search.
2. **Augmentation Step**
    - The retrieved job descriptions are **embedded** and matched with the user's skills, experience, and preferences.
    - This ensures the AI only sees **relevant context** instead of scanning the entire internet.
3. **Generation Step**

- The **Gemini LLM** (via LangChain) takes the filtered, context-rich job data and generates:
    - A ranked list of jobs
    - Short explanations of why each job is a good fit
    - Direct application links

This approach **reduces hallucinations**, improves job-match accuracy, and ensures recommendations are **personalized and data-driven**.

Challenges Faced

Scraping LinkedIn job listings directly turned out to be nearly impossible due to strict anti-scraping measures and rate limits.
To overcome this, I used **SerpAPI** to scrape **Google Jobs** listings instead, which provided a more accessible and reliable data source.

Another challenge was integrating the **Streamlit UI** with the job retrieval backend.
Establishing a smooth connection between the data retrieval process (via SerpAPI) and the interactive interface in Streamlit required careful handling of API responses, parsing job data, and dynamically updating the UI — but I successfully implemented it. ✅

What I Learned

- **Working with RAG (Retrieval-Augmented Generation):**
  Learned how to integrate **LangChain** with **Gemini** to fetch relevant context and enhance job recommendation accuracy.
- **API Data Retrieval:**
  Understood how to use **SerpAPI** effectively for extracting job data from Google Jobs.
- **Data Parsing & Cleaning:**
  Gained experience in processing and cleaning scraped job data for better usability in recommendations.
- **UI & Backend Integration:**
  Learned how to connect a backend retrieval system with a **Streamlit** frontend for

real-time interaction.

- **Overcoming Scraping Challenges:**
 Discovered alternative strategies when direct scraping is restricted (LinkedIn ➡ Google Jobs via SerpAPI).

📜 Code Walkthrough – PDF Q&A with Google Gemini in Google Colab

---

1. Importing Required Libraries

python

CopyEdit

```
from google.colab import files, userdata

import PyPDF2

import faiss

import numpy as np

from sentence_transformers import SentenceTransformer

import os

import asyncio

from dotenv import load_dotenv

import google.generativeai as genai
```

- **google.colab** → For file uploads (`files.upload()`) and securely accessing secrets (`userdata.get()`).

- **PyPDF2** → Extract text from uploaded PDF files.
- **faiss** → Efficient similarity search for embeddings.
- **numpy** → Numerical operations (used with FAISS).
- **sentence_transformers** → Create semantic embeddings of text chunks.
- **dotenv** → Load `.env` variables if running locally.
- **google.generativeai** → Connect with **Google Gemini** models for Q&A.

---

2. Loading Environment Variables

python

CopyEdit

```
load_dotenv()
```

Loads API keys from a `.env` file (only needed if running locally).

---

3. Configuring Google API Key in Colab

python

CopyEdit

```
try:
    GOOGLE_API_KEY = userdata.get('GOOGLE_API_KEY')
    if not GOOGLE_API_KEY:
        raise ValueError("Google API Key not found in Colab Secrets.")
    genai.configure(api_key=GOOGLE_API_KEY)
    print("Google API key configured successfully from Colab
Secrets.")

except ValueError as e:

    ...

except Exception as e:

    ...
```

- **userdata.get('GOOGLE_API_KEY')** → Retrieves your API key stored in Colab's secret manager.
- **genai.configure()** → Configures the Gemini client.
- If not found, it throws an error with instructions.

---

4. Initializing SentenceTransformer Embedder

python

CopyEdit

```
embedder = SentenceTransformer("all-MiniLM-L6-v2")
```

- Loads a small, fast embedding model for converting text into vectors.

---

5. Async Loop Setup (Colab Compatibility)

python

CopyEdit

```
try:
    asyncio.get_running_loop()
except RuntimeError:
    asyncio.set_event_loop(asyncio.new_event_loop())
```

- Ensures an asyncio event loop is running (needed for SentenceTransformer in some environments).

---

6. PDF Processing Functions

Extract PDF Text

python

CopyEdit

```
def get_pdf_text(pdf_docs):
    text = ""
    for pdf in pdf_docs:
        pdf_reader = PyPDF2.PdfReader(pdf)
        for page in pdf_reader.pages:
            text += page.extract_text() or ""
    return text
```

- Reads each PDF and extracts text page by page.
- Uses `or ""` to avoid errors if text extraction fails for a page.

Split Text into Chunks

python

CopyEdit

```
def split_text(text, chunk_size=500, overlap=50):

    words = text.split()

    chunks = []

    for i in range(0, len(words), chunk_size - overlap):

        chunk = " ".join(words[i:i + chunk_size])

        chunks.append(chunk)

    return chunks
```

- Breaks text into **overlapping word chunks** for better retrieval accuracy.

Create FAISS Vector Store

python

CopyEdit

```
def create_vector_store(chunks):

    embeddings = embedder.encode(chunks)

    dim = embeddings.shape[1]

    index = faiss.IndexFlatL2(dim)

    index.add(np.array(embeddings).astype('float32'))

    return index, chunks
```

- Encodes chunks into embeddings.
- Stores them in a FAISS index for fast similarity search.

Search Chunks

python

CopyEdit

```
def search_chunks(index, chunks, query, top_k=3):
    query_embedding = embedder.encode([query])
    D, I = index.search(np.array(query_embedding).astype('float32'),
k=top_k)
    return [chunks[i] for i in I[0]]
```

- Finds the **top_k most relevant chunks** for a given query.

Generate Answer with Gemini

python

CopyEdit

```
def generate_answer(context, question):
    full_prompt = f"""Answer the question using only the information
provided in the context...
    Context:{context}
    Question:{question}
    Answer:
    """
    model = genai.GenerativeModel(model_name="gemini-2.5-flash")
    response = model.generate_content(full_prompt)
    return response.text.strip()
```

- Creates a structured prompt.
- Calls Gemini (`gemini-2.5-flash`) to generate an **answer grounded in the given context**.

---

7. File Upload in Colab

python

CopyEdit

```
uploaded_files = files.upload()

pdf_docs = [f for f in uploaded_files.keys() if f.endswith('.pdf')]
```

- Lets the user upload PDFs.
- Filters only `.pdf` files.

---

8. Processing Uploaded PDFs

python

CopyEdit

```
if pdf_docs:

    ...

    for pdf_path in pdf_docs:

        with open(pdf_path, 'rb') as f:

            ...
```

- Reads uploaded PDFs.
- Extracts text from each page.

---

9. Chunking, Indexing, and Question Answering

python

CopyEdit

```
chunks = split_text(raw_text)

index, chunk_list = create_vector_store(chunks)
```

- Splits extracted text into chunks.
- Creates a FAISS vector index.

python

CopyEdit

```
user_question = input("Enter your question about the PDF(s): ")
```

- Takes a question from the user.

python

CopyEdit

```
top_chunks = search_chunks(index, chunk_list, user_question)

context_text = " ".join(top_chunks)

answer = generate_answer(context_text, user_question)
```

- Retrieves top relevant chunks.
- Passes them to Gemini for an answer.

---

10. Output

python

CopyEdit

```
print("\nAnswer:")

print(answer)
```

- Displays the **final AI-generated answer**.