Link to the notebook:

https://colab.research.google.com/drive/177eT9QJYes7_f9_Su1aP0Rja4f0ymL9Q?usp=sharing

The entire flow of this project:

1. Upload & Prepare the Document

- **What Happens:**
  - You upload a PDF (e.g., SRS_Document.pdf).
  - The system splits it into small chunks (like breaking a book into pages).
- **Why?**
  - Large documents can't be processed all at once. Chunks make searching easier.

---

2. Create a "Searchable Memory"

- **What Happens:**
  - Each text chunk is converted into numbers (called *embeddings*).
  - These numbers are stored in a smart index (like a book's index but for AI).
- **Why?**
  - Numbers help the AI quickly find relevant parts when you ask a question.

---

3. Load the LLM (Mistral-7B)

- **What Happens:**
  - A pre-trained AI model (Mistral-7B) is loaded onto Google Colab's GPU.
  - This model acts like a local ChatGPT (no internet needed).
- **Why?**
  - The model generates answers based on the document's context.

---

1. System Overview

Purpose

A self-contained question-answering system that:

- Processes local documents (PDF/DOC/TXT)
- Answers queries using Mistral-7B LLM (offline)
- Maintains full data privacy (no external APIs)

Key Components

Component	Technology Used
Document Loader	UnstructuredFileLoader
Text Processing	CharacterTextSplitter
Embeddings	GPT4AllEmbeddings
Vector Database	ChromaDB
LLM	Mistral-7B-Instruct (GGUF)

---

2. Installation Guide

Prerequisites

- Google Colab (Recommended) or Linux environment
- NVIDIA GPU (T4 or higher for optimal performance)

Step-by-Step Setup

1. Install dependencies

!pip install "unstructured[pdf]" pytesseract langchain-community gpt4all chromadb

!pip install -U llama-cpp-python langchain faiss-cpu sentence-transformers pypdf --force-reinstall

2. Download Mistral-7B model (4-bit quantized)

!wget https://huggingface.co/TheBloke/Mistral-7B-Instruct-v0.1-GGUF/resolve/main/mistral-7b-instruct-v0.1.Q4_K_M.gguf

---

3. Core Workflow

1. Document Ingestion

```
from google.colab import files
from langchain.document_loaders import UnstructuredFileLoader
from langchain.text_splitter import CharacterTextSplitter
uploaded = files.upload()

loader = UnstructuredFileLoader("SRS_Document.pdf")  # Replace with your filename
docs = loader.load()
text_splitter = CharacterTextSplitter(chunk_size=2000, chunk_overlap=50)
split_docs = text_splitter.split_documents(docs)
```

2. Vector Database Setup

```
from langchain.embeddings import GPT4AllEmbeddings
from langchain.vectorstores import Chroma

Create embeddings and store in ChromaDB
embedding = GPT4AllEmbeddings()
db = Chroma.from_documents(split_docs, embedding)
```

3. LLM Initialization

```
from gpt4all import GPT4All

Initialize Mistral-7B
llm = GPT4All(
    model_name="mistral-7b-instruct-v0.1.Q4_K_M.gguf",
    model_path="/content/",
    n_threads=8  # Adjust based on CPU cores
)
```

4. Query Processing

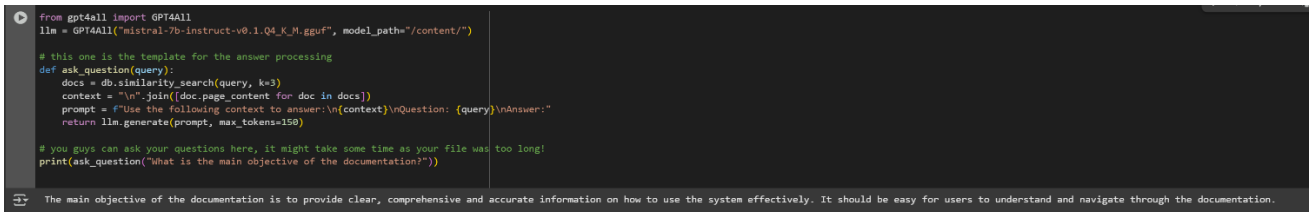
```
def ask_question(query):
    Retrieve relevant context
    docs = db.similarity_search(query, k=3)
    context = "\n".join([doc.page_content for doc in docs])

    Generate answer
    prompt = f"""Answer the question based ONLY on this context:
    {context}
    Question: {query}
    Answer:"""

    return llm.generate(prompt, max_tokens=150)

# Example usage
print(ask_question("What is the main objective of the documentation?"))
```

output:



4. Configuration Reference

Critical Parameters

Parameter	Recommended Value	Description
chunk_size	1500-2500	Balance between context and precision
chunk_overlap	50-100	Prevents context fragmentation
k (retrieval)	2-3	Number of document chunks to consider
max_tokens	100-200	Controls answer length

---

5. Performance Optimization

For Faster Processing

1. Reduce chunk_size to 1000 if memory-constrained
2. Use n_threads=4 (for 4-core CPUs) in GPT4All()
3. Pre-process documents to remove unnecessary images/tables

For Better Accuracy

1. Increase chunk_overlap to 200 for technical documents
2. Set k=4 in similarity_search() for broader context

OUTPUT: