AI Document Search and Question Answering Project

# 1. Introduction

This document provides detailed steps and information about the AI Document Search and Question Answering project. The project uses LangChain, FAISS, and Groq to create a document search and question-answering system. It allows users to upload documents, split them into manageable chunks, and perform semantic searches based on user queries. The project also integrates with Groq’s language model to answer questions related to the content of the documents.

# 2. Project Description

The AI Document Search and Question Answering project is designed to provide an efficient way to search and retrieve information from a large corpus of documents. By leveraging embeddings and similarity search, the project can find relevant sections of text based on a user's query and then provide a detailed answer using a pre-trained language model from Groq.

# 3. Prerequisites

Ensure that your system meets the following requirements:  
- Python 3.x  
- Google Colab or a local Python environment  
- Groq API key

# 4. Setup Instructions

## Cloning the Repository

1. Clone the repository using the following command:

```bash  
git clone https://github.com/your-username/AI-Document-Search-and-Question-Answering.git  
```  
2. Navigate to the project directory:  
```bash  
cd AI-Document-Search-and-Question-Answering  
```

## Installing Dependencies

3. Install the required dependencies:

```bash  
pip install -q langchain langchain\_core langchain\_community sentence\_transformers faiss-cpu unstructured chromadb Cython tiktoken unstructured[local-inference] langchain\_groq  
```

## API Key Setup

4. Obtain an API key from Groq. Set up the API key in your environment by adding the following code to your script:  
```python  
import getpass  
import os  
  
if 'GROQ\_API\_KEY' not in os.environ:  
 os.environ['GROQ\_API\_KEY'] = getpass.getpass('Provide your GROQ API TOKEN')  
```

# 5. Running the Project

1. \*\*Load and Process Documents:\*\*  
 Use the TextLoader to load your document into the system.  
 ```python  
 from langchain.document\_loaders import TextLoader  
 loader = TextLoader('/path/to/your/document.txt')  
 documents = loader.load()  
 ```

2. \*\*Wrap Text for Readability:\*\*  
 Ensure the loaded text is readable by wrapping it at a specified width.  
 ```python  
 import textwrap  
  
 def wrap\_text\_preserve\_newlines(text, width=110):  
 lines = text.split('\n')  
 wrapped\_lines = [textwrap.fill(line, width=width) for line in lines]  
 wrapped\_text = '\n'.join(wrapped\_lines)  
 return wrapped\_text  
  
 print(wrap\_text\_preserve\_newlines(str(documents[0])))  
 ```

3. \*\*Split Documents into Chunks:\*\*  
 Split the documents into smaller chunks to optimize the search and answer retrieval process.  
 ```python  
 from langchain.text\_splitter import CharacterTextSplitter  
 text\_splitter = CharacterTextSplitter(chunk\_size=1000, chunk\_overlap=0)  
 docs = text\_splitter.split\_documents(documents)  
 ```

4. \*\*Create a Vector Store with FAISS:\*\*  
 Use embeddings to convert text chunks into vectors and store them with FAISS for fast retrieval.  
 ```python  
 from langchain.embeddings import HuggingFaceEmbeddings  
 from langchain.vectorstores import FAISS  
  
 embeddings = HuggingFaceEmbeddings()  
 db = FAISS.from\_documents(docs, embeddings)  
 ```

5. \*\*Perform a Similarity Search:\*\*  
 Search the vector store for chunks that are semantically similar to a user's query.  
 ```python  
 query = 'explain about indian premier league'  
 docs = db.similarity\_search(query)  
 print(wrap\_text\_preserve\_newlines(str(docs[0].page\_content)))  
 ```

6. \*\*Question Answering with Groq:\*\*  
 Use Groq's language model to generate answers based on the search results.  
 ```python  
 import langchain\_groq  
 from langchain\_groq import ChatGroq  
 from langchain.chains.question\_answering import load\_qa\_chain  
  
 GROQ\_LLM = ChatGroq(api\_key=os.getenv('GROQ\_API\_KEY'), model='gemma2-9b-it')  
 chain = load\_qa\_chain(GROQ\_LLM, chain\_type='stuff')  
  
 query = 'HOW IS INDIAN ECONOMY'  
 docs = db.similarity\_search(query)  
 chain.run(input\_documents=docs, question=query)  
 ```

7. \*\*Run Additional Queries:\*\*  
 Perform additional searches and question-answering operations as needed.  
 ```python  
 query = 'explain about indian geography?'  
 docs = db.similarity\_search(query)  
 chain.run(input\_documents=docs, question=query)  
 ```

# 6. Customization

You can customize the following aspects of the project:  
- \*\*Text Splitting:\*\* Adjust the `chunk\_size` and `chunk\_overlap` parameters in `CharacterTextSplitter` to optimize document processing for different text lengths and types.  
- \*\*Groq Model:\*\* Experiment with different Groq models by specifying the `model` parameter in `ChatGroq`.  
- \*\*Vector Store:\*\* Use alternative vector stores if needed by replacing FAISS with another implementation that suits your needs.

# 7. Conclusion

This project demonstrates how to build an efficient document search and question-answering system using modern AI tools. By integrating LangChain, FAISS, and Groq, the system provides a robust solution for extracting and querying information from large text corpora.