



Question Answer Chatbot using Lang Chain Framework and Large Language Models (LLMs).

Text Analytics Project

Syed Muzammil Ahmed (25371) & Huzaifa Nafees (25378)

INTRODUCTION

With the rapid advancements in natural language processing and machine learning, chatbots have emerged as valuable tools for enhancing user interactions and providing instant responses to a wide range of inquiries. However, building an efficient and accurate QA chatbot remains a challenging task, especially when dealing with complex questions and diverse domains.

In this project, we are creating a question answer chat bot which will take any document, research paper, book or article as input and will answer any question related to that document. For this purpose, we are using lang chain as our framework along with different LLMs (Large language models). We are creating embeddings from four different embeddings models such as sentence transformers, cohere etc. Not only answering the question but our chatbot can also provide the summary of the document. The goal of the Chatbot QA project is to develop an AI-powered chatbot that can engage in dynamic and meaningful conversations, address user queries accurately, and provide insightful responses across various domains. With the combination of lang chain framework and LLMs we can make a robust chatbot that can accurately answer complex questions. In the end, we are also making user interface for chatbot using streamlit API.

MODELS AND FRAMEWORKS

Lang Chain Framework

Lang chain is the latest software development framework which can be used for application development using large language models. It's a framework that can be used for development of Question answer chatbots. Lang chain was launched in the last quarter of 2022 so it's quite a new framework and it contains utilities which make creation of LLMs web application less complex.

Embeddings Models

We are using different embedding models to create the embedding of the document and the question asked. For embedding creation, we are using different embedding models which are as follows:

- 1- Cohere Embedding (model: "embed-english-light-v2.0", size = 1024)
- 2- JINA Ai Embeddings (model: "ViT-H-14: laion2b-s32b-b79k", size = 1024)
- 3- Hugging Face Sentence Transformers ("sentence-transformers/all-mpnet-base-v2" size = 768)
- 4- Model Scape Embeddings ("damo/nlp_corom_sentence-embedding_english-base", size = 768)

Large Language Model

We are using different large language models to create the accurate chatbot. By combining the unique strengths and capabilities of different models, we aim to achieve a comprehensive and versatile chatbot that excels in understanding and answering user queries. LLMS which we are using are:

- 1- Cohere Base Light Model
- 2- Goose Ai (model: gpt-neo-20b)
- 3- Replicate (model: Dolly-v2-12b)
- 4- Hugging Face (model: santacoder)
- 5- Cohere Command Model.

We are implementing all these models on each embedding we use above to check out their performance through their similarity and relevancy of the answer result.

Pinecone Vector Space

Pinecone is a scalable vector database and similarity search service designed to handle high-dimensional vector data efficiently. It provides an infrastructure for storing, indexing, and searching vector representations, enabling fast and accurate similarity-based retrieval.

We are using pinecone vector space to store our document embeddings and then our model can access the vector space to check the similarity of the query asked with the created embeddings of documents.

The screenshot displays the Pinecone web interface for a project named 'text-analytics-project-embeddings'. On the left is a dark sidebar with navigation links: PROJECT, Default Project, Indexes, Collections, API Keys, Members, Docs, Support Center, and a user profile 'Syed Muzammil Ahm...'. The main area shows a table of indexes with one entry: 'text-analytics-project-embeddings' in the 'us-west4-gcp-free' environment, using 'cosine' metric and 'Starter' pod type, with 768 dimensions, 1 pod per replica, 1 replica, and 1 total pod. Below the table is an 'Index Info' section showing 'Total Vectors: 801' and a breakdown of 'Vectors without namespace: 801'.

Index Name	Environment	Metric	Pod Type	Dimensions	Pods per Replica	Replicas	Total Pods	Actions
text-analytics-project-embeddings text-analytics-project-embeddings-a9cfd29.svc.us-west4-gcp-free.pinecone.io ● Ready	us-west4-gcp-free	cosine	Starter	768	1	1	1	[Icons for copy, settings, delete]

Index Info

Total Vectors
801

Namespace	Number of Vectors
Vectors without namespace	801

METHODOLOGY

The process and the steps which we have followed for the creation of efficient chat bot are as follows:

- 1- Take any document, pdf, or webpage. Directory loader is a utility by length chain to load the data. It has a class unstructured data loader which is commonly used to load different types of data files such as text, PDF, or images etc.
- 2- Convert that document into small chunks, so rather than treating one document we will make chunks of the document and then semantically we could match those chunks and find out which is the most relevant chunk. Which can answer that question. So, this is the reason for making documents into chunks so that instead of taking whole document we split document into chunks which is computationally more feasible.

Learn chain also provides utility to create the chunks named as “text splitter”. This utility is used to split your text into chunks. It will recursively split documents into paragraphs then in line. We will define a parameter chunk size which sets how many characters to take in one chunk.

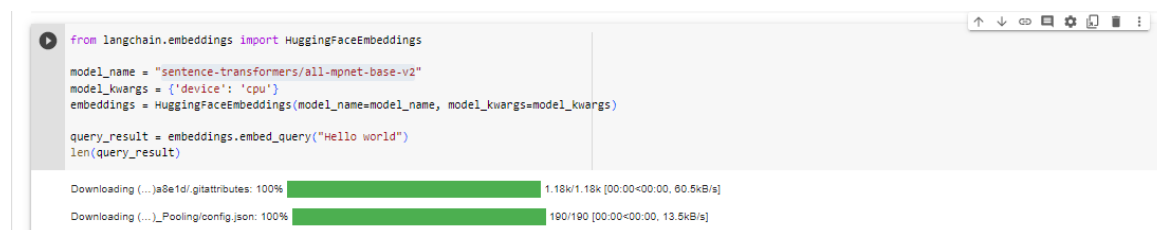
```
[ ] def split_documents(documents, chunk_size=1000, chunk_overlap=20):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=chunk_size, chunk_overlap=chunk_overlap)
    chunks = text_splitter.split_documents(documents)
    return chunks

chunks = split_documents(documents)
print(len(chunks))

237
```

- 3- Now we must generate embeddings of the chunks because we want to match two things, one is the document chunk and the query, or the question asked by the user.

Folding embeddings, lang chain also have utility such as “open AI embeddings”, “Cohere Embeddings: or “Hugging Face embeddings” etc. which will simply convert the chunks into the embedding in one step.



```
from langchain.embeddings import HuggingFaceEmbeddings

model_name = "sentence-transformers/all-mpnet-base-v2"
model_kwargs = {'device': 'cpu'}
embeddings = HuggingFaceEmbeddings(model_name=model_name, model_kwargs=model_kwargs)

query_result = embeddings.embed_query("Hello world")
len(query_result)
```

Downloading (...)ja8e1d/gitattributes: 100% 1.18k/1.18k [00:00<00:00, 60.5kB/s]

Downloading (...)Pooling/config.json: 100% 190/190 [00:00<00:00, 13.5kB/s]

- 4- Now we must store these embeddings in a vector space store. For this purpose, we are using pinecone as a vector database to store our embeddings. So now we must create an index of pinecone, index is something where we can store our document embeddings.

Lang chain framework also helped the utility to store embeddings into Pinecone.

```
[ ] import pinecone
    #from langchain.vectorstores import pinecone

    pinecone.init(
        api_key="1523033c-3666-40fc-a544-03600265eb80",
        environment="us-west4-gcp-free"
    )

    index_name = "text-analytics-project-embeddings"

    index = Pinecone.from_documents(chunks, embeddings, index_name=index_name)
```

- 5- Now we will create a function which will take query and return the similar chunks by calculating the similarity score.

In **Get_similar_doc** function “k” parameter tells us about how many chunks to return in output. If we do (score = True) then it will return similarity score as well.

- 6- After storing the data into vector space, our next step is to implement LLMs. For that Lang chain has a utility which supports implementation of large language models.

```
[ ] import os
    os.environ["HUGGINGFACEHUB_API_TOKEN"] = HUGGINGFACEHUB_API_TOKEN

[ ] from langchain.llms import HuggingFaceHub

    repo_id = "bigcode/santacoder"

    llm = HuggingFaceHub(repo_id=repo_id, model_kwargs={"temperature":1, "max_length":94})

[ ] chain = load_qa_chain(llm, chain_type="stuff")

    def get_answer(query):
        similar_docs = get_similar_docs(query)
        answer = chain.run(input_documents=similar_docs, question=query)
        return answer
```

- 7- Now for Question answering, we are using another utility of lang chain name as “lang chain QA” which will take our LLM model and the user query and return answer of that query.

```
[ ] chain = load_qa_chain(llm, chain_type="stuff")

    def get_answer(query):
        similar_docs = get_similar_docs(query)
        answer = chain.run(input_documents=similar_docs, question=query)
        return answer

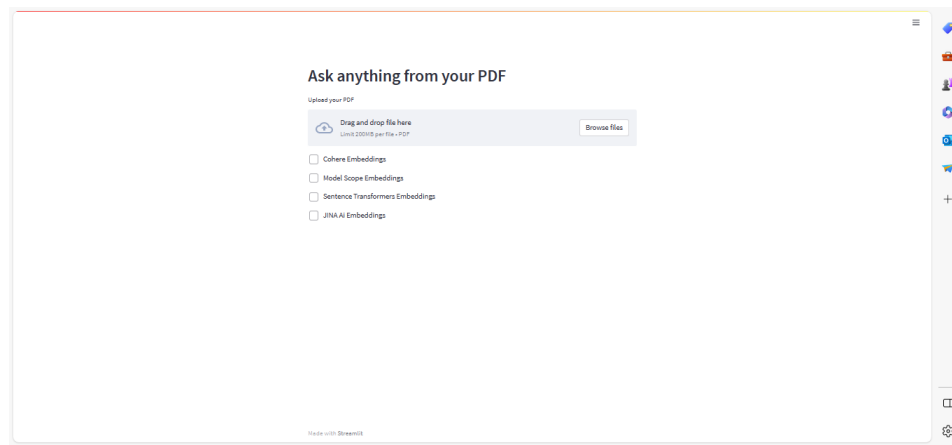
[ ] query = "What is the capital of Pakistan ?"
    answer = get_answer(query)
    print(answer)

    The capital of Pakistan is Islamabad.
```

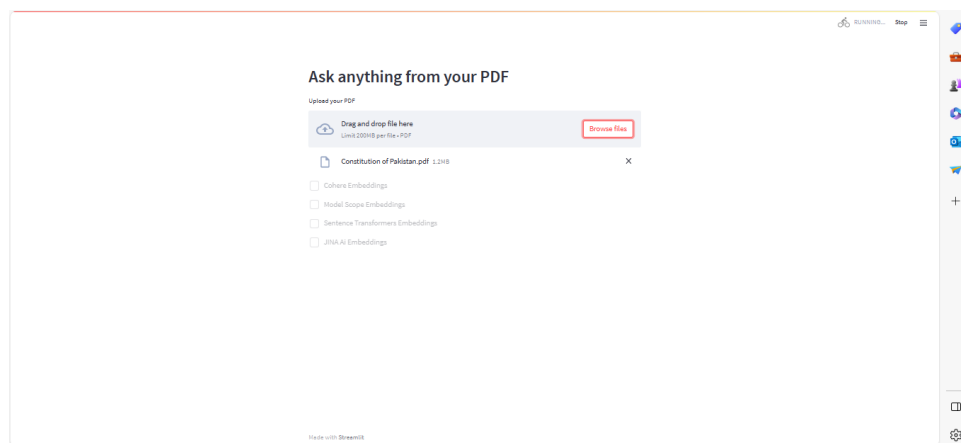
STREAM LIT API

We are also making User interface of our project using streamlit API. First, we will upload our PDF document on the app and then select the embeddings to be used. We have deployed the Command model of Cohere on our web app because it is best working among all and giving relatively correct answers to the query. We have also deployed our app Hugging face.

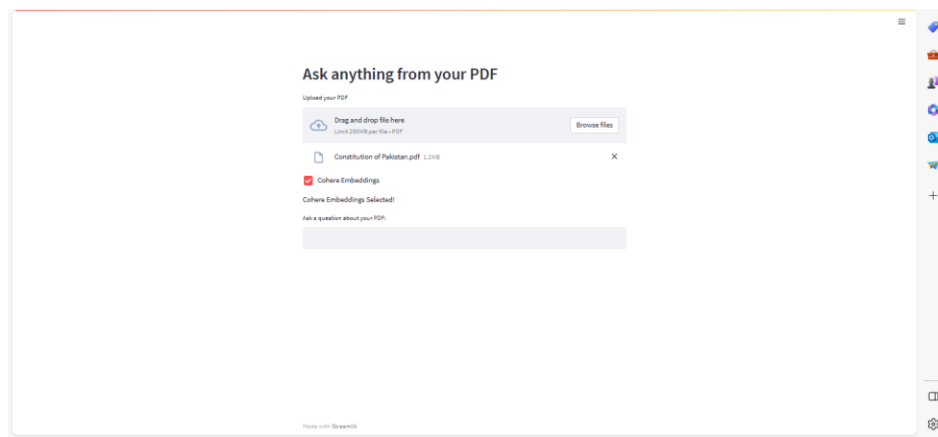
Step 1: Opening a Stream Lit App



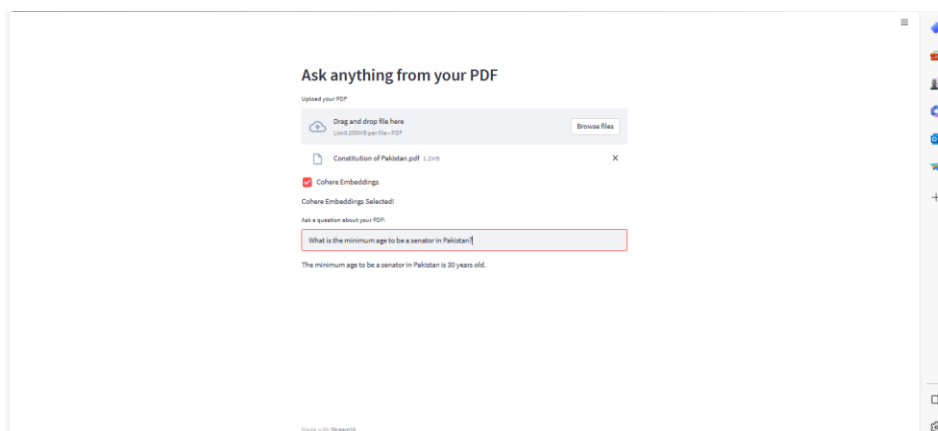
Step 2: Upload any document of your choice. We are using constitution of Pakistan in the below screenshot.



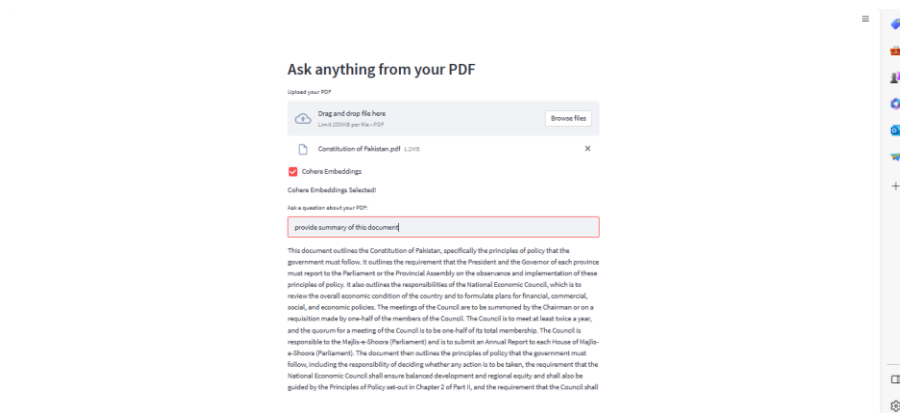
Step 3: Select embedding which you want to use.



Step 4: Ask any Question relevant to the document uploaded.



Our web app can also provide a summary of the uploaded document. Which can be seen below.



ANALYSIS AND OBSERVATIONS

We have applied different LL models but some of them are working fine and some of them are not performing accurate results such as Replicate.

Some models are returning on one word answer, so they are not compatible enough for long answers.

Many models are facing hallucination issues with their results. Which means they are generating text which is irrelevant and nonsensical to the query.

We have deployed, cohere model named “command” because it is best performing as compared to the other models.

CONCLUSION AND FUTURE WORK

In conclusion, this chatbot can help users to get instant responses which they want to search from their document or any other important paper. Users can also upload a whole book and ask anything about the uploaded book. It serves as a reliable and efficient support tool, enhancing the user experience and providing quick access to information.

We have explored different large language models and different embedding models which is great for our learning. For further enhancement, we can also use paid APIs such as Open Ai models and embeddings which can improve our project result further. Currently, we are using open-source models and embeddings.

