**Extracting Answers from URL’s using LangChain and HuggingFace**

Import the necessary packages

import os  
import pickle  
import langchain  
from langchain\_community.embeddings import HuggingFaceEmbeddings  
from langchain import HuggingFaceHub  
from langchain.chains import RetrievalQA  
from langchain.text\_splitter import RecursiveCharacterTextSplitter  
from langchain.document\_loaders import UnstructuredURLLoader  
from langchain.vectorstores import FAISS

I intended to utilize the OpenAI API for my demonstration, but my free credit has been depleted. Therefore, I’m using the Hugging Face API instead.

Configure your Hugging Face API key as an environment variable to access the Hugging Face API.

#load Hugging Face api key  
os.environ['HUGGINGFACEHUB\_API\_TOKEN'] = "Your Hugging Face API Key"

Here am using pre-trained language model named “Mixtral-8x7B-Instruct-v0.1”. Which is a opensource Model available in Huggingface.

llm = HuggingFaceHub(  
 repo\_id="mistralai/Mixtral-8x7B-Instruct-v0.1",  
 model\_kwargs={"temperature":0.8, "max\_length":1000}  
)

* temperature: 0.8 — Controls the probability distribution of the model’s output.
* max\_length: 1000 — Limits the maximum length of generated text to 1000 tokens.

Now we can load our document , Here I am using News web article’s.

Below are the links:

'https://www.moneycontrol.com/news/business/tata-motors-mahindra-gain-certificates-for-production-linked-payouts-11281691.html',

'https://www.moneycontrol.com/news/business/tata-motors-launches-punch-icng-price-starts-at-rs-7-1-lakh-11098751.html'

A close-up of a computer screen

Description automatically generated

Once the data load from the web article , we need to break down the content into smaller manageable chunks of text.

Instead of using the entire content as a whole to create the prompt, we split it into smaller chunks. By doing so, we can locate the specific chunks that contain our input query, merge them with our query, and construct the final prompt more effectively.

By doing this, we can conserve our API credit by reducing the number of tokens in the input query sent to the LLM API.

A white rectangular object with black text

Description automatically generated

Once we’ve segmented the content into multiple chunks, we can transform them into vectors (embeddings) and save them into a vector database.

Various vector databases are accessible in the market, including **Chroma** and **Pinecone**. However, for this specific use case, I’m opting not to utilize any of these mentioned databases. Instead, for relatively smaller datasets, **FAISS** can be employed.

FAISS is not a traditional vector database like Chroma or Pinecone. Instead, it is a library specifically designed for efficient similarity search.While FAISS provides indexing capabilities similar to those of a database, it primarily focuses on the indexing and search aspects rather than providing a complete database management system.

A screenshot of a computer

Description automatically generated

Once FAISS vector index created from these embeddings, saving the index to a file, and then loading it back into memory if needed.

A screenshot of a computer

Description automatically generated

The above statement instantiates a retrieval-based question answering model, configuring it with a language model for generating answers, a retriever for retrieving relevant documents, and specifying that the source documents should be returned along with the answers.

After setting up the retrieval chain we can ask our query related to the document.

Here my question will be relate to that given article url’s . That is “what is the price of Tiago iCNG?”

A screenshot of a computer

Description automatically generated

References:

https://www.youtube.com/watch?v=d4yCWBGFCEs&t=8777s

https://medium.com/@jishnumohan481/extracting-answers-from-a-document-using-langchain-and-hugging-face-5e30a0c1fbac