```python
# -*- coding: utf-8 -*-
"""Building Chatbot with RAG

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1EyAey7kSI-4SfM1xwVK5I1sZATcfHXE0
"""

!pip install -qU \
    langchain==0.0.292 \
    openai==0.28.0 \
    datasets==2.10.1 \
    pinecone-client==2.2.4 \
    tiktoken==0.5.1

"""BUILDING A CHATBOT WITH NO RAG"""

import os
from langchain.chat_models import ChatOpenAI

os.environ["OPENAI_API_KEY"] = ""

chat = ChatOpenAI(
    openai_api_key=os.environ["OPENAI_API_KEY"],
    model='gpt-3.5-turbo'
)

from langchain.schema import (
    SystemMessage,
    HumanMessage,
    AIMessage
)

messages = [
    SystemMessage(content="You are a helpful assistant."),
    HumanMessage(content="Hi AI, how are you today?"),
    AIMessage(content="I'm great thank you. How can I help you?"),
    HumanMessage(content="I'd like to understand string theory.")
]

res = chat(messages)

res

print(res.content)

messages.append(res)

# now create a new user prompt
prompt = HumanMessage(
    content="What is so special about Llama 2?"
)
# add to messages
messages.append(prompt)

# send to OpenAI
res = chat(messages)

print(res.content)



"""YOU SEE THE LLM DO NOT HAVE ANY INFO ABOUT THE RECENT LLAMA 2. They have no access to the external world and cannot work with the recent things. To solve th

BUILDING THE LLM USING RAG

IMPORTING THE DATASET
The dataset is about the some research papers on Llama 2. So with the help of this, our model will then be able to answer the questions regarding Llama 2.
"""

from datasets import load_dataset

dataset = load_dataset(
    "jamescalam/llama-2-arxiv-papers-chunked",
    split="train"
)

dataset

dataset[0]

"""BUILDING KNOWLEDGE BASE
For this we use the vector database.




"""

import pinecone

# get API key from app.pinecone.io and environment from console
pinecone.init(
    api_key='',
    environment='gcp-starter'
)

import time

index_name = 'llama-2-rag'

if index_name not in pinecone.list_indexes():
    pinecone.create_index(
        index_name,
        dimension=1536,
        metric='cosine'
    )
    # wait for index to finish initialization
    while not pinecone.describe_index(index_name).status['ready']:
        time.sleep(1)

index = pinecone.Index(index_name)

index.describe_index_stats()

from langchain.embeddings.openai import OpenAIEmbeddings

embed_model = OpenAIEmbeddings(model="text-embedding-ada-002")

texts = [
    'this is the first chunk of text',
    'then another second chunk of text is here'
]

res = embed_model.embed_documents(texts)
len(res), len(res[0])

from tqdm.auto import tqdm  # for progress bar

data = dataset.to_pandas()  # this makes it easier to iterate over the dataset

batch_size = 100

for i in tqdm(range(0, len(data), batch_size)):
    i_end = min(len(data), i+batch_size)
    # get batch of data
    batch = data.iloc[i:i_end]
    # generate unique ids for each chunk
    ids = [f"{x['doi']}-{x['chunk-id']}" for i, x in batch.iterrows()]
    # get text to embed
    texts = [x['chunk'] for _, x in batch.iterrows()]
    # embed text
    embeds = embed_model.embed_documents(texts)
    # get metadata to store in Pinecone
    metadata = [
        {'text': x['chunk'],
         'source': x['source'],
         'title': x['title']} for i, x in batch.iterrows()
    ]
    # add to Pinecone
    index.upsert(vectors=zip(ids, embeds, metadata))

index.describe_index_stats()

"""USing RAG"""

from langchain.vectorstores import Pinecone

text_field = "text"  # the metadata field that contains our text

# initialize the vector store object
vectorstore = Pinecone(
    index, embed_model.embed_query, text_field
)

query = "What is so special about Llama 2?"

vectorstore.similarity_search(query, k=3)

def augment_prompt(query: str):
    # get top 3 results from knowledge base
    results = vectorstore.similarity_search(query, k=3)
    # get the text from the results
    source_knowledge = "\n".join([x.page_content for x in results])
    # feed into an augmented prompt
    augmented_prompt = f"""Using the contexts below, answer the query.

    Contexts:
    {source_knowledge}

    Query: {query}"""
    return augmented_prompt

print(augment_prompt(query))

# create a new user prompt
prompt = HumanMessage(
    content=augment_prompt(query)
)
# add to messages
messages.append(prompt)

res = chat(messages)

print(res.content)

prompt = HumanMessage(
    content="what safety measures were used in the development of llama 2?"
)

res = chat(messages + [prompt])
print(res.content)

prompt = HumanMessage(
    content=augment_prompt(
        "what safety measures were used in the development of llama 2?"
    )
)

res = chat(messages + [prompt])
print(res.content)
```