Topic : **RAG Model For Wikipedia Question Answering**

Group : Yashvi Shah (60018210013) Department : AI/DS

Achintya Shah (60018220037)

Hrushik Mehta (60018220042)

Dhruvi Doshi (60018220147)

1. Introduction

This project focuses on leveraging RAG (Retrieval-Augmented Generation) models for enhancing question-answering capabilities on Wikipedia data. RAG models combine retrieval-based and generation-based methods to improve accuracy and relevance in answering questions from a large corpus of text.

2. Problem Statement

The objective of this project is to develop a system that can effectively answer questions using Wikipedia data by employing a RAG model. The aim is to improve the accuracy and relevance of answers by utilizing the strengths of both retrieval-based and generation-based approaches, thereby minimizing errors and enhancing the user experience.

3. Implementation Details

3.1 RAG Model

The RAG (Retrieval-Augmented Generation) model combines two key components:

Retrieval Component: This component retrieves relevant documents or passages from a large corpus (in this case, Wikipedia) based on the input query. The retrieval is typically done using dense retrieval techniques like DPR (Dense Passage Retrieval).

Generation Component: This component uses a generative model, such as a transformer-based model like BART or GPT-3, to generate a coherent and contextually relevant answer based on the retrieved documents.

By integrating these components, RAG models can provide more accurate and comprehensive answers compared to traditional question-answering systems.

3.2 Dense Passage Retrieval (DPR)

Dense Passage Retrieval is a key part of the retrieval component. It uses dense vector representations to match the input query with relevant passages from the corpus. This is achieved through the following steps:

Embedding Queries and Passages: Both queries and passages are embedded into dense vectors using neural networks. The embeddings capture semantic similarities between the text segments.

Retrieving Relevant Passages: The system retrieves passages whose embeddings are closest to the query embedding, thus ensuring relevance and accuracy.

3.3 Generative Model

The generative model synthesizes an answer based on the retrieved passages. It involves:

Contextual Understanding: The model processes the retrieved passages to understand the context and generate a relevant answer.

Sequence Generation: The generative model outputs a coherent and contextually appropriate answer by leveraging the information from the retrieved passages.

The combination of dense retrieval and generative modeling ensures that the answers are not only relevant but also well-formed and contextually appropriate.

4. Application and Improvement for Wikipedia Question Answering

The methods presented in this project can be highly valuable for Wikipedia question answering, particularly in:

Enhanced Answer Accuracy: By combining retrieval and generation, the system can provide more accurate answers by leveraging relevant passages from Wikipedia.

Improved Relevance: The dual approach ensures that answers are contextually relevant and comprehensive.

Efficient Information Processing: The integration of DPR and generative models allows for efficient processing of large volumes of information, making it suitable for extensive knowledge bases like Wikipedia.

Through this approach, the project aims to create a robust and effective system for answering questions using Wikipedia data, providing users with accurate and relevant information.

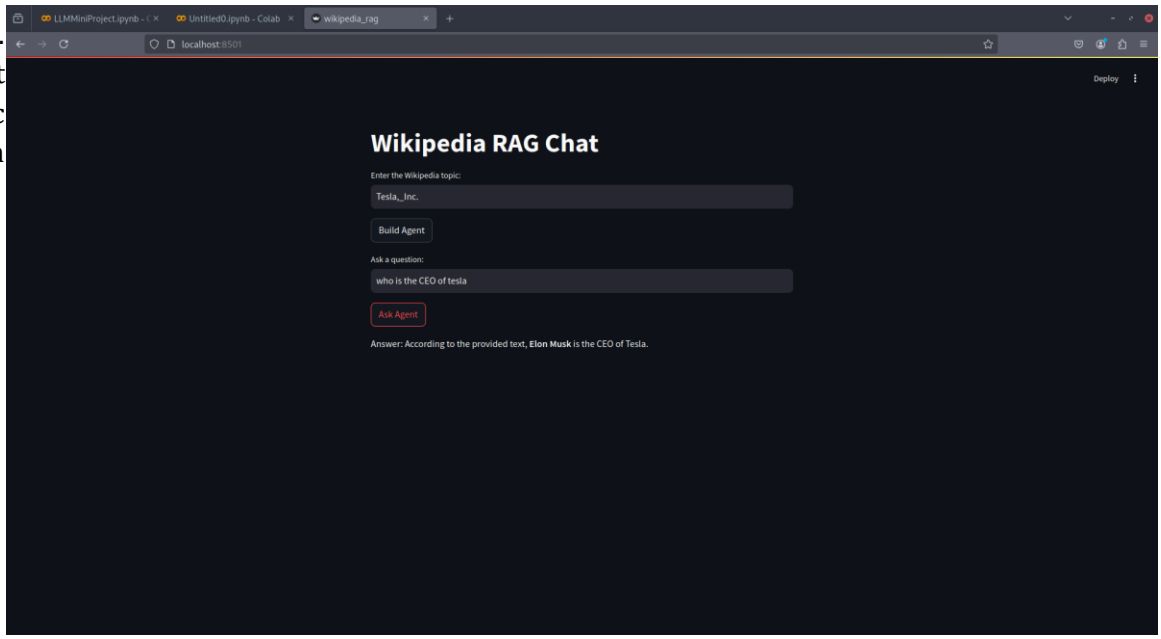
5. Code

```
# -*- coding: utf-8 -*-
"""Wikipedia RAG.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1JQVwCAltNtHTluESq5Rr07CXSbwUhg5j
"""

# !pip install requests beautifulsoup4 sentence-transformers transformers
langchain_google_genai langchain langchain_community faiss_cpu rank_bm25 -qqq

import os
import re
import requests
from bs4 import BeautifulSoup
from getpass import getpass
from dotenv import load_dotenv
import logging

from langchain_google_genai import GoogleGenerativeAI, GoogleGenerativeAIEmbeddings
from langchain_community.retrievers import BM25Retriever
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import FAISS
from langchain.agents.agent_toolkits import create_conversational_retrieval_agent
from langchain.tools.retriever import create_retriever_tool
from langchain.memory import ConversationBufferMemory
from langchain.chains import ConversationChain
from langchain_community.retrievers import WikipediaRetriever

load_dotenv()

os.environ["GOOGLE_API_KEY"] = os.environ["GEMINI_API_KEY"]

def scrape(article_title):
    base_url = "https://en.wikipedia.org/wiki/"
    url = base_url + article_title
    response = requests.get(url)
    response.raise_for_status()
    soup = BeautifulSoup(response.content, 'html.parser')
    content_div = soup.find('div', id='mw-content-text')
    return clean_text(clean_text(content_div.text))
```

```
def clean_text(text):
   text = re.sub(r'\[\d+\]', '', text)
   text = re.sub(r'\((.*?citation needed.*?)\)', '', text, flags=re.IGNORECASE)
   text = text.replace('\n', ' ')
   text = re.sub(r'\s+', ' ', text)
   return text.strip()

def retrieve(query):
   wikipedia_text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000,
chunk_overlap=50, length_function=len)
   article_text = scrape(query)
   texts = wikipedia_text_splitter.split_text(article_text)
   embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001",
task_type="RETRIEVAL_DOCUMENT")
   vectorstore = FAISS.from_texts(texts, embeddings)
   retriever = BM25Retriever.from_texts(texts=texts, vectorstore=vectorstore, k=10)
   return retriever

def retriever_tool(retriever):
   return create_retriever_tool(retriever, "Wikipedia", "Searches and returns information
from Wikipedia based on the topic provided.")

def build(query):
   retriever = retrieve(query)
   if not retriever:
      logging.error("Failed to retrieve information from Wikipedia.")
      return None
   model = GoogleGenerativeAI(model="gemini-1.5-flash", temperature=0, maxRetries = 5)

   agent = create_conversational_retrieval_agent(
      llm=model,
      tools=[retriever_tool(retriever)],
      verbose=False
   )
   return agent

def ask_agent(question):
   if agent is None:
      return "No relevant data found for the question."

   retrieved_info = agent.tools[0].run(question)
```

```
    prompt = f"You are a helpful AI assistant. You have access to the following information from Wikipedia: \n\n{retrieved_info}\n\nPlease answer the following question based *only* on the information provided above. Do not use any external knowledge or information beyond what is provided in the Wikipedia text. \n{question}\n\nAnswer:"
    response = agent.invoke({"input": prompt})
    final_output = response['output']

    if final_output == "":
        return "No relevant information available."

    return final_output


agent = build("Tesla,_Inc.")
print("Question: Summarize the wikipedia page")
print(ask_agent("Summarize the wikipedia page"))
print()
print("Question: What is bye in German?")
print(ask_agent("What is bye in German?"))
print()


import streamlit as st
from wikipedia_rag import build, ask_agent

from streamlit.web import cli as stcli
from streamlit import runtime
import sys

def main():
    # Initialize session state
    if 'agent' not in st.session_state:
        st.session_state.agent = None

    # Title
    st.title("Wikipedia RAG Chat")

    # Input for the topic
    topic = st.text_input("Enter the Wikipedia topic:", "")

    # Button to build the agent
    if st.button("Build Agent") and topic:
        st.session_state.agent = build(topic)
```

```
        if st.session_state.agent:
            st.success("Agent built successfully!")
        else:
            st.error("Failed to build the agent.")

    # Input for the question
    question = st.text_input("Ask a question:", "")

    if st.button("Ask Agent") and question:
        if st.session_state.agent:
            answer = ask_agent(question)
            st.write("Answer:", answer)
        else:
            st.error("Please build the agent first by entering a topic and clicking 'Build Agent'.")

if __name__ == "__main__":
    if runtime.exists():
        main()
    else:
        sys.argv = ["streamlit", "run", sys.argv[0]]
        sys.exit(stcli.main())
```



6. Conclusion

This project demonstrates the effectiveness of using RAG (Retrieval-Augmented Generation) models for Wikipedia question answering. By combining retrieval-based and generative models, we can achieve more accurate and relevant answers. The system efficiently processes large volumes of information, ensuring privacy and improving the user experience. This approach highlights the potential of RAG models in transforming question-answering systems, providing valuable insights and enhancing information retrieval.