Tutorial: Creating a Retrieval-Augmented Generation (RAG) Pipeline with LLaMA-2

Introduction

This tutorial will guide you through the process of creating a Retrieval-Augmented Generation (RAG) pipeline. We will use LLaMA-2 for language generation, LangChain for managing the pipeline, and Pinecone for vector storage. The pipeline will enable you to convert PDF documents into text, chunk the text, store the chunks in a vector store, and query the documents to generate relevant responses.

Before you run the code make sure:

- Access to Google colab https://colab.research.google.com/ (sign in with your gmail account)
- Huggingface API key: Make account https://huggingface.co/
- Pinecone: Create account and get API key https://www.pinecone.io/Need
- LLM model to use: Can use Llama, request access on Huggingface
  - Go to https://huggingface.co/meta-llama, click on any Llama model you want to use and request access **Might need ~1hr to be approved**

Prerequisites

Before we start, ensure you have the following packages installed:

```
pip install pymupdf langchain sentence-transformers torch transformers
langchain-community bitsandbytes accelerate pinecone-client
```

Step 1: Import Libraries

First, we need to import the necessary libraries.

```
import pymupdf
from langchain.text_splitter import RecursiveCharacterTextSplitter
import os
import transformers
import bitsandbytes as bnb
import pinecone
from langchain.vectorstores import Pinecone
from langchain.chains import RetrievalQA
from langchain.llms import HuggingFacePipeline
```

Step 2: Define Functions to Chunk PDFs

We will define functions to load PDF documents, convert them to text, and chunk the text into smaller parts.

```
def chunk_doc(pdf_path):
    doc = pymupdf.open(pdf_path)
    metadata = {"author": doc.metadata["author"], "title":
doc.metadata["title"]}
    text = ""

    try:
        for page_num in range(len(doc)):
            page = doc.load_page(page_num)
            text += page.get_text()
        text_splitter = RecursiveCharacterTextSplitter(chunk_size=1024,
chunk_overlap=64)
        split_docs = text_splitter.split_text(text)
        return split_docs, metadata
    except Exception as e:
        print(f"error chunking file: {pdf_path}")
        return [], None

def load_folder(path):
    documents = []
    metadata = []
    for file in os.listdir(path):
        if file.endswith(".pdf"):
            pdf_path = os.path.join(path, file)
```

```
            print(f"loading file: {pdf_path}")
            chunks, doc_metadata = chunk_doc(pdf_path)
            if chunks:
                documents.append(chunks)
                metadata.append(doc_metadata)
    return documents, metadata
```

Step 3: Load PDF and Chunk Text

Use the functions defined above to load a PDF and chunk its text.

```
pdf_path = "/content/your_pdf_file.pdf"
chunks, metadata = chunk_doc(pdf_path)
print(chunks)
print(metadata)
```

Step 4: Initialize LLaMA-2 Model

Initialize the LLaMA-2 model for text generation.

```
model_id = "meta-llama/Llama-2-7b-chat-hf"
bnb_config = bnb.BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_compute_dtype=bfloat16
)

hf_auth = "your_huggingface_auth_token"
model_config = transformers.AutoConfig.from_pretrained(model_id,
use_auth_token=hf_auth)
model = transformers.AutoModelForCausalLM.from_pretrained(
    model_id,
    trust_remote_code=True,
    config=model_config,
    quantization_config=bnb_config,
    device_map='auto',
    use_auth_token=hf_auth
)
model.eval()
```

```
tokenizer = transformers.AutoTokenizer.from_pretrained(model_id,
use_auth_token=hf_auth)
generate_text = transformers.pipeline(
    model=model, tokenizer=tokenizer,
    return_full_text=False,
    task='text-generation',
    temperature=0.1,
    max_new_tokens=512,
    repetition_penalty=1.1
)
llm = HuggingFacePipeline(pipeline=generate_text)
```

Step 5: Initialize Pinecone and Vector Store

Initialize Pinecone and create a vector store to store the document chunks.

```
pinecone.init(api_key="your_pinecone_api_key", environment="us-west1-gcp")
index = pinecone.Index("your_index_name")
embed_model = transformers.AutoModel.from_pretrained("sentence-
transformers/all-mpnet-base-v2")
text_field = 'text'

vectorstore = Pinecone(index, embed_model.embed_query, text_field)
```

Step 6: Create RAG Pipeline

Create the RAG pipeline using the initialized components.

```
rag_pipeline = RetrievalQA.from_chain_type(
    llm=llm, chain_type='stuff',
    retriever=vectorstore.as_retriever(),
    return_source_documents=True
)
```

Step 7: Query the RAG Pipeline

Query the pipeline to get answers from the documents.

```
query = "Explain the process of electrodeposition"
answer = rag_pipeline(query)
print(answer["result"])
print(answer["source_documents"])
```

Conclusion

You have successfully created a Retrieval-Augmented Generation (RAG) pipeline using LLaMA-2, LangChain, and Pinecone. This pipeline allows you to process PDF documents, store their text chunks in a vector store, and query the documents to generate relevant responses.

Google Colab link: https://colab.research.google.com/drive/1rSChF2twDO14KGwUcQI4k__ra6L3GkDn#scrollTo=m9AnylSenC1W