3b_Openai

November 12, 2024

OPENAI LLM setup

Loading packages, libraries and secrets into notebook

[1]:
```
# Importing the required libraries
from langchain_openai import OpenAI
from langchain_openai import ChatOpenAI
from langchain.chains import RetrievalQA
import gradio as gr
from gradio.themes.base import Base
from langchain_core.output_parsers import StrOutputParser
from langchain.prompts import ChatPromptTemplate
from langchain_core.runnables import RunnableParallel, RunnablePassthrough
import os
from dotenv import load_dotenv
```

[2]:
```
# Accessing the secrets from the environment variables
load_dotenv()
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
```

Chain setup

[ ]:
```
query = "SELECT T1.fname ,  T1.age FROM student AS T1 JOIN has_pet AS T2 ON T1.
 ↪stuid  =  T2.stuid JOIN pets AS T3 ON T3.petid  =  T2.petid WHERE T3.pettype␣
 ↪ =  'dog' AND T1.stuid NOT IN (SELECT T1.stuid FROM student AS T1 JOIN␣
 ↪has_pet AS T2 ON T1.stuid  =  T2.stuid JOIN pets AS T3 ON T3.petid  =  T2.
 ↪petid WHERE T3.pettype  =  'cat')"
output_length = len(query.split())*3 # word count of SQL query multiplied by␣
 ↪four

# Model and parsing setup
model = ChatOpenAI(api_key=OPENAI_API_KEY, model="gpt-4o-mini", temperature=0)
parser = StrOutputParser()

# Define prompt template
template = """
Provide first a natural language Translation followed by an Explanation of the␣
 ↪SQL Query. Go through it step by step and output the result in simple and␣
 ↪concise language. Keep the output in line with the Length number.
```

```
Query: {query}

Lenght: {output_length}
"""

prompt = ChatPromptTemplate.from_template(template)

# Chain setup
chain_3b = (
    {"query": RunnablePassthrough(), "output_length" : RunnablePassthrough()}
    | prompt
    | model
    | parser
)

# Execute the chain with the logging retriever
chain_3b.invoke(query)
```

Chat interface setup

Change cell type below to Python, when running only this script. Markdown format for testing.

[ ]:
```
# Define the chain_invoke function
def chain_3b_invoke(query):
    # Execute the chain with the logging retriever
    result = chain_3b.invoke(query)
    # Return the result
    return result

# Create a web interface for the app, using Gradio
with gr.Blocks(theme=Base(), title="Question Answering App using Vector Search␣
 ↪+ RAG") as demo:
    gr.Markdown(
        """
        # Question Answering App using Atlas Vector Search + RAG Architecture
        """)
    textbox = gr.Textbox(label="Enter your SQL statement:")
    with gr.Row():
        button = gr.Button("Submit", variant="primary")
    output = gr.Textbox(lines=1, max_lines=30, label="Natural language␣
 ↪translation and explanation:")

# Call chain_invoke function upon clicking the Submit button
    button.click(chain_3b_invoke, textbox, outputs=output)

demo.launch()
```