gemini_langchain_rag

August 4, 2024

[1]:
```
%%capture
!pip -q install langchain_experimental langchain_core
!pip -q install google-generativeai==0.3.1
!pip -q install google-ai-generativelanguage==0.4.0
!pip -q install langchain-google-genai
!pip -q install wikipedia
!pip -q install docarray
!pip -q install --upgrade protobuf google.protobuf
```

[1]:
```
# Before proceeding with the next cells restart the kernel by clicking the␣
 ↪refresh icon on the top toolbar.
```

[2]:
```
import os
import google.generativeai as genai

from IPython.display import display
from IPython.display import Markdown
from google.protobuf.empty_pb2 import Empty

key_name = !gcloud services api-keys list --filter="gemini-api-key"␣
 ↪--format="value(name)"
key_name = key_name[0]

api_key = !gcloud services api-keys get-key-string $key_name␣
 ↪--location="us-central1" --format="value(keyString)"
api_key = api_key[0]

os.environ["GOOGLE_API_KEY"] = api_key

genai.configure(api_key=os.environ["GOOGLE_API_KEY"])
```

[3]:
```
models = [m for m in genai.list_models()]
models
```

[3]:
```
[Model(name='models/chat-bison-001',
       base_model_id='',
       version='001',
       display_name='PaLM 2 Chat (Legacy)',
```

```
       description='A legacy text-only model optimized for chat conversations',
       input_token_limit=4096,
       output_token_limit=1024,
       supported_generation_methods=['generateMessage', 'countMessageTokens'],
       temperature=0.25,
       max_temperature=None,
       top_p=0.95,
       top_k=40),
 Model(name='models/text-bison-001',
       base_model_id='',
       version='001',
       display_name='PaLM 2 (Legacy)',
       description='A legacy model that understands text and generates text as
an output',
       input_token_limit=8196,
       output_token_limit=1024,
       supported_generation_methods=['generateText', 'countTextTokens',
'createTunedTextModel'],
       temperature=0.7,
       max_temperature=None,
       top_p=0.95,
       top_k=40),
 Model(name='models/embedding-gecko-001',
       base_model_id='',
       version='001',
       display_name='Embedding Gecko',
       description='Obtain a distributed representation of a text.',
       input_token_limit=1024,
       output_token_limit=1,
       supported_generation_methods=['embedText', 'countTextTokens'],
       temperature=None,
       max_temperature=None,
       top_p=None,
       top_k=None),
 Model(name='models/gemini-1.0-pro-latest',
       base_model_id='',
       version='001',
       display_name='Gemini 1.0 Pro Latest',
       description=('The best model for scaling across a wide range of tasks.
This is the latest '
                    'model.'),
       input_token_limit=30720,
       output_token_limit=2048,
       supported_generation_methods=['generateContent', 'countTokens'],
       temperature=0.9,
       max_temperature=None,
       top_p=1.0,
```

```
       top_k=None),
 Model(name='models/gemini-1.0-pro',
       base_model_id='',
       version='001',
       display_name='Gemini 1.0 Pro',
       description='The best model for scaling across a wide range of tasks',
       input_token_limit=30720,
       output_token_limit=2048,
       supported_generation_methods=['generateContent', 'countTokens'],
       temperature=0.9,
       max_temperature=None,
       top_p=1.0,
       top_k=None),
 Model(name='models/gemini-pro',
       base_model_id='',
       version='001',
       display_name='Gemini 1.0 Pro',
       description='The best model for scaling across a wide range of tasks',
       input_token_limit=30720,
       output_token_limit=2048,
       supported_generation_methods=['generateContent', 'countTokens'],
       temperature=0.9,
       max_temperature=None,
       top_p=1.0,
       top_k=None),
 Model(name='models/gemini-1.0-pro-001',
       base_model_id='',
       version='001',
       display_name='Gemini 1.0 Pro 001 (Tuning)',
       description=('The best model for scaling across a wide range of tasks.
This is a stable '
                    'model that supports tuning.'),
       input_token_limit=30720,
       output_token_limit=2048,
       supported_generation_methods=['generateContent', 'countTokens',
'createTunedModel'],
       temperature=0.9,
       max_temperature=None,
       top_p=1.0,
       top_k=None),
 Model(name='models/gemini-1.0-pro-vision-latest',
       base_model_id='',
       version='001',
       display_name='Gemini 1.0 Pro Vision',
       description='The best image understanding model to handle a broad range
of applications',
       input_token_limit=12288,
```

```
       output_token_limit=4096,
       supported_generation_methods=['generateContent', 'countTokens'],
       temperature=0.4,
       max_temperature=None,
       top_p=1.0,
       top_k=32),
 Model(name='models/gemini-pro-vision',
       base_model_id='',
       version='001',
       display_name='Gemini 1.0 Pro Vision',
       description='The best image understanding model to handle a broad range
of applications',
       input_token_limit=12288,
       output_token_limit=4096,
       supported_generation_methods=['generateContent', 'countTokens'],
       temperature=0.4,
       max_temperature=None,
       top_p=1.0,
       top_k=32),
 Model(name='models/gemini-1.5-pro-latest',
       base_model_id='',
       version='001',
       display_name='Gemini 1.5 Pro Latest',
       description='Mid-size multimodal model that supports up to 2 million
tokens',
       input_token_limit=2097152,
       output_token_limit=8192,
       supported_generation_methods=['generateContent', 'countTokens'],
       temperature=1.0,
       max_temperature=2.0,
       top_p=0.95,
       top_k=64),
 Model(name='models/gemini-1.5-pro-001',
       base_model_id='',
       version='001',
       display_name='Gemini 1.5 Pro 001',
       description='Mid-size multimodal model that supports up to 2 million
tokens',
       input_token_limit=2097152,
       output_token_limit=8192,
       supported_generation_methods=['generateContent', 'countTokens',
'createCachedContent'],
       temperature=1.0,
       max_temperature=2.0,
       top_p=0.95,
       top_k=64),
 Model(name='models/gemini-1.5-pro',
```

```
       base_model_id='',
       version='001',
       display_name='Gemini 1.5 Pro',
       description='Mid-size multimodal model that supports up to 2 million
tokens',
       input_token_limit=2097152,
       output_token_limit=8192,
       supported_generation_methods=['generateContent', 'countTokens'],
       temperature=1.0,
       max_temperature=2.0,
       top_p=0.95,
       top_k=64),
 Model(name='models/gemini-1.5-pro-exp-0801',
       base_model_id='',
       version='exp-0801',
       display_name='Gemini 1.5 Pro Experimental 0801',
       description='Mid-size multimodal model that supports up to 2 million
tokens',
       input_token_limit=2097152,
       output_token_limit=8192,
       supported_generation_methods=['generateContent', 'countTokens'],
       temperature=1.0,
       max_temperature=2.0,
       top_p=0.95,
       top_k=64),
 Model(name='models/gemini-1.5-flash-latest',
       base_model_id='',
       version='001',
       display_name='Gemini 1.5 Flash Latest',
       description='Fast and versatile multimodal model for scaling across
diverse tasks',
       input_token_limit=1048576,
       output_token_limit=8192,
       supported_generation_methods=['generateContent', 'countTokens'],
       temperature=1.0,
       max_temperature=2.0,
       top_p=0.95,
       top_k=64),
 Model(name='models/gemini-1.5-flash-001',
       base_model_id='',
       version='001',
       display_name='Gemini 1.5 Flash 001',
       description='Fast and versatile multimodal model for scaling across
diverse tasks',
       input_token_limit=1048576,
       output_token_limit=8192,
       supported_generation_methods=['generateContent', 'countTokens',
```

```
'createCachedContent'],
       temperature=1.0,
       max_temperature=2.0,
       top_p=0.95,
       top_k=64),
 Model(name='models/gemini-1.5-flash',
       base_model_id='',
       version='001',
       display_name='Gemini 1.5 Flash',
       description='Fast and versatile multimodal model for scaling across
diverse tasks',
       input_token_limit=1048576,
       output_token_limit=8192,
       supported_generation_methods=['generateContent', 'countTokens'],
       temperature=1.0,
       max_temperature=2.0,
       top_p=0.95,
       top_k=64),
 Model(name='models/embedding-001',
       base_model_id='',
       version='001',
       display_name='Embedding 001',
       description='Obtain a distributed representation of a text.',
       input_token_limit=2048,
       output_token_limit=1,
       supported_generation_methods=['embedContent'],
       temperature=None,
       max_temperature=None,
       top_p=None,
       top_k=None),
 Model(name='models/text-embedding-004',
       base_model_id='',
       version='004',
       display_name='Text Embedding 004',
       description='Obtain a distributed representation of a text.',
       input_token_limit=2048,
       output_token_limit=1,
       supported_generation_methods=['embedContent'],
       temperature=None,
       max_temperature=None,
       top_p=None,
       top_k=None),
 Model(name='models/aqa',
       base_model_id='',
       version='001',
       display_name='Model that performs Attributed Question Answering.',
       description=('Model trained to return answers to questions that are
```

```
grounded in provided '
                     'sources, along with estimating answerable probability.'),
       input_token_limit=7168,
       output_token_limit=1024,
       supported_generation_methods=['generateAnswer'],
       temperature=0.2,
       max_temperature=None,
       top_p=1.0,
       top_k=40)]
```

1 Using Gemini directly with Python SDK

[4]:
```
# generate text
prompt = 'Who are you and what can you do?'

model = genai.GenerativeModel('gemini-pro')

response = model.generate_content(prompt)

Markdown(response.candidates[0].content.parts[0].text)
```

[4]:

I am Gemini, a multimodal AI model, developed by Google. I am designed to provide information and assist with a wide range of language-based tasks, such as answering questions, generating text, translating languages, and writing different kinds of creative content.

Here are some of my capabilities:

Information Retrieval: * Answering factual questions on a diverse range of topics * Providing summaries of complex information * Translating text between over 100 languages

Text Generation: * Generating creative text, such as stories, poems, and songs * Writing different types of content, including articles, blog posts, and marketing copy * Summarizing long pieces of text

Language Understanding: * Understanding the meaning and context of text * Identifying and extracting key information * Recognizing and responding to different types of user intent

Communication: * Engaging in natural language conversations * Answering follow-up questions * Providing clear and concise explanations

Other Capabilities: * Code generation * Math problem solving * Fact-checking * Sentiment analysis

I am constantly learning and improving, and my capabilities are expanding all the time. I am not yet able to perform all tasks perfectly, but I am always striving to provide the best possible assistance.

Please let me know if you have any other questions.

2 Using Gemini with LangChain

2.1 Basic LLM Chain

[5]:
```
from langchain_core.messages import HumanMessage
from langchain_google_genai import ChatGoogleGenerativeAI

llm = ChatGoogleGenerativeAI(model="gemini-pro",
                             temperature=0.7)


result = llm.invoke("What is a LLM?")

Markdown(result.content)
```

[5]:
LLM stands for Large Language Model. It is a type of artificial intelligence (AI) that is trained on a massive dataset of text and code. LLMs are able to understand and generate human-like text, translate languages, write different kinds of creative content, and even write different kinds of code. They are also able to answer questions and perform other tasks that require a deep understanding of language.

LLMs are still under development, but they have already shown great promise for a wide range of applications, including customer service, education, and entertainment. As LLMs continue to improve, they are likely to play an increasingly important role in our lives.

Here are some of the things that LLMs are capable of: * **Understanding and generating human-like text:** LLMs can understand the meaning of text and generate new text that is fluent, coherent, and informative. This makes them ideal for tasks such as writing articles, stories, and marketing copy. * **Translating languages:** LLMs can translate text between different languages with a high degree of accuracy. This makes them ideal for tasks such as customer service, travel, and education. * **Writing different kinds of creative content:** LLMs can write different kinds of creative content, such as poems, songs, and screenplays. This makes them ideal for tasks such as entertainment, education, and marketing. * **Answering questions:** LLMs can answer questions on a wide range of topics, including history, science, and current events. This makes them ideal for tasks such as customer service, education, and research. * **Performing other tasks that require a deep understanding of language:** LLMs can perform a variety of other tasks that require a deep understanding of language, such as summarizing text, identifying sentiment, and detecting plagiarism. This makes them ideal for tasks such as research, data analysis, and content moderation.

LLMs are a powerful tool that can be used for a wide range of applications. As they continue to improve, they are likely to play an increasingly important role in our lives.

[6]:
```
for chunk in llm.stream("Write a haiku about LLMs."):
    print(chunk.content)
    print("---")
```

```
Words dance in the void,
AI's boundless language stream,
Thoughts
```

```
---
 take flight anew.
---
```

2.2 Basic Multi Chain

[7]:
```
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_google_genai import GoogleGenerativeAIEmbeddings
from langchain.prompts import ChatPromptTemplate
from langchain.schema.output_parser import StrOutputParser


model = ChatGoogleGenerativeAI(model="gemini-pro", temperature=0.7)
```

[8]:
```
prompt = ChatPromptTemplate.from_template(
    "tell me a short joke about {topic}"
)

output_parser = StrOutputParser()
```

[9]:
```
chain = prompt | model | output_parser
```

[10]:
```
chain.invoke({"topic": "machine learning"})
```

[10]:
```
'Why did the machine learning algorithm get a speeding ticket?\n\nBecause it was
caught accelerating too fast.'
```

2.3 A more complicated Chain - RAG

[ ]:
```
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_google_genai import GoogleGenerativeAIEmbeddings
from langchain.vectorstores import DocArrayInMemorySearch
```

[ ]:
```
model = ChatGoogleGenerativeAI(model="gemini-pro",
                               temperature=0.7)
```

[ ]:
```
embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001")
```

[ ]:
```
from langchain.document_loaders import WikipediaLoader

# use Wikipedia loader to create some docs to use..
docs = WikipediaLoader(query="Machine Learning", load_max_docs=10).load()
docs += WikipediaLoader(query="Deep Learning", load_max_docs=10).load()
docs += WikipediaLoader(query="Neural Networks", load_max_docs=10).load()

# Take a look at a single document
docs[0]
```

[ ]:
```
vectorstore = DocArrayInMemorySearch.from_documents(
    docs,
    embedding=embeddings # passing in the model to embed documents..
)

retriever = vectorstore.as_retriever()
```

[ ]:
```
retriever.get_relevant_documents("what is machine learning?")
```

[ ]:
```
retriever.get_relevant_documents("what is gemini pro?")
```

[ ]:
```
template = """Answer the question a a full sentence, based only on the␣
 ↪following context:
{context}

Return you answer in three back ticks

Question: {question}
"""
prompt = ChatPromptTemplate.from_template(template)
```

[ ]:
```
from langchain.schema.runnable import RunnableMap
```

[ ]:
```
retriever.get_relevant_documents("What is a graident boosted tree?")
```

[ ]:
```
chain = RunnableMap({
    "context": lambda x: retriever.get_relevant_documents(x["question"]),
    "question": lambda x: x["question"]
}) | prompt | model | output_parser
```

[ ]:
```
chain.invoke({"question": "What is machine learning?"})
```

[ ]:
```
chain.invoke({"question": "When was the transformer invented?"})
```