Making Movie Mania

Overview

Building a Movie Making Chatbot Assistance System with RAG, LangChain, LLM, and Vector Database

Introduction

The goal of this project is to develop a movie recommendation system that leverages advanced machine learning techniques to provide personalized recommendations based on user inputs. The system integrates Retrieval-Augmented Generation (RAG) using a Large Language Model (LLM), LangChain, and a vector database to enhance the recommendation process.

Components and Technologies

Large Language Model (LLM): Used for generating natural language responses and enhancing the recommendation process. LangChain: Manages the flow of interactions between the user, LLM, and vector database. Vector Database: Stores embeddings of movie data for efficient retrieval based on semantic similarity. Examples include Pinecone, Weaviate, or FAISS. Streamlit: Provides an interactive web interface for users to input queries and receive recommendations.

Workflow

Data Collection and Preprocessing

Collect movie data including titles, genres, ratings, and descriptions. Clean and preprocess the data to ensure it is in a suitable format for generating embeddings. Embedding Generation

Pre-trained Embedding Model

Use a pre-trained embedding model to generate embeddings for movie titles and descriptions. Store these embeddings in a vector database for efficient retrieval. Vector Database Integration

Vector Database

Initialize and configure the vector database. Index the movie embeddings for fast similarity searches. LangChain Integration

LangChain

Set up LangChain to manage the interaction flow between the user inputs, LLM responses, and vector database retrievals. Define the prompts and response generation logic. Retrieval-Augmented Generation (RAG) Workflow

LLM

User inputs a query through the Streamlit interface. LangChain processes the query and retrieves relevant movie embeddings from the vector database. The LLM uses the retrieved embeddings to generate a natural language response recommending movies.

Streamlit Interface

Develop a user-friendly web interface for inputting queries and displaying recommendations. Implement input fields and submit buttons for user interaction.

Objectives

This notebook provides a guide to building a Adaptive Recommendation Chatbot using multimodal retrieval augmented generation (RAG) and Vector Database.

The tasks that this notebook would perform:

1. Extract data from documents containing both text and images using Gemini Vision Pro, and generate embeddings of the data, store it in vector store
2. Search the vector store with text queries to find similar text data
3. Using Text data as context, generate answer to the user query using Gemini Pro Model.

Begin with Vertex AI SDK Setup

Setting Up Vertex AI SDK and Essential Packages

```
!pip install --upgrade --quiet pymupdf langchain gradio google-cloud-aiplatform lan
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.5/3.5 MB 14.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 983.6/983.6 kB 13.0 MB/s eta 0:00:
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.3/12.3 MB 30.0 MB/s eta 0:00:0
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.1/5.1 MB 29.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 73.0/73.0 kB 2.8 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 15.7/15.7 MB 34.1 MB/s eta 0:00:0
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 362.4/362.4 kB 18.7 MB/s eta 0:00:
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 kB 1.8 MB/s eta 0:00:0
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 92.0/92.0 kB 3.7 MB/s eta 0:00:00
```

```
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 318.2/318.2 kB 8.6 MB/s eta 0:00:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.6/75.6 kB 4.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 141.1/141.1 kB 3.6 MB/s eta 0:00:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.1/10.1 MB 41.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.4/62.4 kB 4.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.9/129.9 kB 7.4 MB/s eta 0:00:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 126.5/126.5 kB 9.5 MB/s eta 0:00:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.9/77.9 kB 6.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 6.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 71.9/71.9 kB 6.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53.6/53.6 kB 768.9 kB/s eta 0:00:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 307.7/307.7 kB 34.0 MB/s eta 0:00:
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 341.4/341.4 kB 35.5 MB/s eta 0:00:
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.4/3.4 MB 89.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 61.0 MB/s eta 0:00:00
  Building wheel for ffmpy (setup.py) ... done
```

Restart runtime

To use the newly installed packages in this Jupyter runtime, you must restart the runtime. You can do this by running the cell below, which restarts the current kernel.

The restart might take a minute or longer. After its restarted, continue to the next step.

```
import IPython

app = IPython.Application.instance()
app.kernel.do_shutdown(True)
```

```
{'status': 'ok', 'restart': True}
```

⚠️ **Wait for the kernel to finish restarting before you continue.** ⚠️

Authenticate your notebook environment (Colab only)

If you are running this notebook on Google Colab, run the cell below to authenticate your environment.

This step is not required if you are using [Vertex AI Workbench](#).

```
import sys

# Additional authentication is required for Google Colab
if "google.colab" in sys.modules:
    # Authenticate user to Google Cloud
    from google.colab import auth

    auth.authenticate_user()
```

Define Google Cloud project information and initialize Vertex AI

To get started using Vertex AI, you must have an existing Google Cloud project and [enable the Vertex AI API](#).

Learn more about [setting up a project and a development environment](#).

```
# Define project information
PROJECT_ID = "project-llm-428915"  # @par
LOCATION = "us-east1"  # @param {type:"st

# Initialize Vertex AI
import vertexai

vertexai.init(project=PROJECT_ID, locatio
```

PROJECT_ID: " project-llm-42891 "

LOCATION: " us-east1 "

```
!pip install langchain_community
```

```
Collecting langchain_community
  Downloading langchain_community-0.2.7-py3-none-any.whl (2.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 16.7 MB/s eta 0:00:00
Requirement already satisfied: PyYAML>=5.3 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: SQLAlchemy<3,>=1.4 in /usr/local/lib/python3.10/c
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in /usr/local/lib/python3.1
Collecting dataclasses-json<0.7,>=0.5.7 (from langchain_community)
  Downloading dataclasses_json-0.6.7-py3-none-any.whl (28 kB)
Requirement already satisfied: langchain<0.3.0,>=0.2.7 in /usr/local/lib/python3
Requirement already satisfied: langchain-core<0.3.0,>=0.2.12 in /usr/local/lib/p
Requirement already satisfied: langsmith<0.2.0,>=0.1.0 in /usr/local/lib/python3
Requirement already satisfied: numpy<2,>=1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: requests<3,>=2 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: tenacity!=8.4.0,<9.0.0,>=8.1.0 in /usr/local/lib/
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dis
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3
Collecting marshmallow<4.0.0,>=3.18.0 (from dataclasses-json<0.7,>=0.5.7->langch
  Downloading marshmallow-3.21.3-py3-none-any.whl (49 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.2/49.2 kB 7.6 MB/s eta 0:00:00
```

```
Collecting typing-inspect<1,>=0.4.0 (from dataclasses-json<0.7,>=0.5.7->langchai
  Downloading typing_inspect-0.9.0-py3-none-any.whl (8.8 kB)
Requirement already satisfied: langchain-text-splitters<0.3.0,>=0.2.0 in /usr/lo
Requirement already satisfied: pydantic<3,>=1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: jsonpatch<2.0,>=1.33 in /usr/local/lib/python3.10
Requirement already satisfied: packaging<25,>=23.2 in /usr/local/lib/python3.10/
Requirement already satisfied: orjson<4.0.0,>=3.9.14 in /usr/local/lib/python3.1
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/d
Requirement already satisfied: typing-extensions>=4.6.0 in /usr/local/lib/python
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dis
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.10/dis
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.
Requirement already satisfied: pydantic-core==2.20.0 in /usr/local/lib/python3.1
Collecting mypy-extensions>=0.3.0 (from typing-inspect<1,>=0.4.0->dataclasses-js
  Downloading mypy_extensions-1.0.0-py3-none-any.whl (4.7 kB)
Installing collected packages: mypy-extensions, marshmallow, typing-inspect, dat
Successfully installed dataclasses-json-0.6.7 langchain_community-0.2.7 marshmal
```

Importing libraries

Let's start by importing the libraries that we will need for this tutorial

```
# File system operations and displaying images
import os

# Import utility functions for timing and file handling
import time

# Libraries for downloading files, data manipulation, and creating a user interface
import uuid
from datetime import datetime

import fitz
import gradio as gr
import pandas as pd

# Initialize Vertex AI libraries for working with generative models
from google.cloud import aiplatform
from PIL import Image as PIL_Image
from vertexai.generative_models import GenerativeModel, Image
from vertexai.language_models import TextEmbeddingModel

# Print Vertex AI SDK version
print(f"Vertex AI SDK version: {aiplatform.__version__}")

# Import LangChain components
import langchain

print(f"LangChain version: {langchain.__version__}")
from langchain.text_splitter import CharacterTextSplitter
from langchain_community.document_loaders import DataFrameLoader
```

```
Vertex AI SDK version: 1.59.0
LangChain version: 0.2.7
```

Initializing Gemini Vision Pro and Text Embedding models

```
# Loading Gemini Pro Vision Model
multimodal_model = GenerativeModel("gemini-1.0-pro-vision")

# Initializing embedding model
text_embedding_model = TextEmbeddingModel.from_pretrained("textembedding-gecko@003"

# Loading Gemini Pro Model
model = GenerativeModel("gemini-1.0-pro")
```

```
!wget https://www.hitachi.com/rev/archive/2023/r2023_04/pdf/04a02.pdf
!wget https://img.freepik.com/free-vector/hand-drawn-no-data-illustration_23-215069

# Create an "Images" directory if it doesn't exist
Image_Path = "./Images/"
if not os.path.exists(Image_Path):
    os.makedirs(Image_Path)

!mv hand-drawn-no-data-illustration_23-2150696455.jpg {Image_Path}/blank.jpg
```

```
--2024-07-12 02:29:23--  https://www.hitachi.com/rev/archive/2023/r2023_04/pdf/0
Resolving www.hitachi.com (www.hitachi.com)... 18.238.136.34, 18.238.136.7, 18.2
Connecting to www.hitachi.com (www.hitachi.com)|18.238.136.34|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1462074 (1.4M) [application/pdf]
Saving to: ‘04a02.pdf.1’

04a02.pdf.1         100%[===================>]   1.39M  2.89MB/s    in 0.5s

2024-07-12 02:29:24 (2.89 MB/s) - ‘04a02.pdf.1’ saved [1462074/1462074]

--2024-07-12 02:29:24--  https://img.freepik.com/free-vector/hand-drawn-no-data-
Resolving img.freepik.com (img.freepik.com)... 23.33.85.241, 23.33.85.240, 2600:
Connecting to img.freepik.com (img.freepik.com)|23.33.85.241|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 32694 (32K) [image/jpeg]
Saving to: ‘hand-drawn-no-data-illustration_23-2150696455.jpg’

hand-drawn-no-data- 100%[===================>]  31.93K  --.-KB/s    in 0.1s

2024-07-12 02:29:25 (325 KB/s) - ‘hand-drawn-no-data-illustration_23-2150696455.
```

Convert PDF to Images and Extract Data Using Gemini Vision Pro

This module processes a set of images, extracting text and tabular data using the multimodal model Gemini Vision Pro. It manages potential errors, stores the extracted information in a DataFrame, and saves the results to a CSV file.

```
# Run the following code for each file
PDF_FILENAME = "Making-Movies-Manual.pdf"  # Replace with the filename for making mo
```

```
          {
            "start_index": 1004,
            "end_index": 1163,
            "uri": "https://www.studocu.com/en-us/document/american-film-institu
          }
        ]
      }
    }
  ],
  "usage_metadata": {
    "prompt_token_count": 265,
    "total_token_count": 265
  }
}
```


```
Taking Some Rest
Cannot process image no: ./Images/Making-Movies-Manual.pdf_1.jpg
processed image no: 1
processed image no: 2
processed image no: 3
processed image no: 4
processed image no: 5
```

	page_id	page_source	page_content
0	1	./Images/Making-Movies-Manual.pdf_0.jpg	013322\nThe Film Foundation presents:\nMAKING...
1	2	./Images/blank.jpg	?\n?\n404\n I Column 1 I Column 2 I\nI :----:...
2	3	./Images/Making-Movies-Manual.pdf_4.jpg	Movies matter because they are more than imag...
3	4	./Images/Making-Movies-Manual.pdf_2.jpg	Introduction\nThis manual will help you make ...
4	5	./Images/Making-Movies-Manual.pdf_3.jpg	A Word from Your Sponsor\nDo you like going t...

Generate Text Embeddings

Leverage a powerful language model textembedding-gecko to generate rich text embeddings that helps us find relevant information from a dataset.

```python
def generate_text_embedding(text) -> list:
    """Text embedding with a Large Language Model."""
    embeddings = text_embedding_model.get_embeddings([text])
    vector = embeddings[0].values
    return vector


# Create a DataFrameLoader to prepare data for LangChain
loader = DataFrameLoader(df, page_content_column="page_content")

# Load documents from the 'page_content' column of your DataFrame
documents = loader.load()

# Log the number of documents loaded
print(f"# of documents loaded (pre-chunking) = {len(documents)}")

# Create a text splitter to divide documents into smaller chunks
text_splitter = CharacterTextSplitter(
    chunk_size=10000,  # Target size of approximately 10000 characters per chunk
    chunk_overlap=200,  # overlap between chunks
)

# Split the loaded documents
doc_splits = text_splitter.split_documents(documents)

# Add a 'chunk' ID to each document split's metadata for tracking
for idx, split in enumerate(doc_splits):
    split.metadata["chunk"] = idx

# Log the number of documents after splitting
print(f"# of documents = {len(doc_splits)}")

texts = [doc.page_content for doc in doc_splits]
text_embeddings_list = []
id_list = []
page_source_list = []
for doc in doc_splits:
    id = uuid.uuid4()
    text_embeddings_list.append(generate_text_embedding(doc.page_content))
    id_list.append(str(id))
    page_source_list.append(doc.metadata["page_source"])
    time.sleep(1)  # So that we don't run into Quota Issue

# Creating a dataframe of ID, embeddings, page_source and text
embedding_df = pd.DataFrame(
    {
        "id": id_list,
        "embedding": text_embeddings_list,
        "page_source": page_source_list,
        "text": texts,
    }
```

```
)
embedding_df.head()
```

```
# of documents loaded (pre-chunking) = 5
# of documents = 5
```

	id	embedding	page_source	text
0	65ab6184-de63-474f-ba54-0360e63b897f	[-0.038054924458265305, -0.04563278704881668, ...	./Images/Making-Movies-Manual.pdf_0.jpg	013322\nThe Film Foundation presents:\nMAKING\...
1	0fd663f0-a499-4545-a42d-79eda6043603	[0.04799053072929382, -0.08848366141319275, -0...	./Images/blank.jpg	?\n?\n404\n I Column 1 I Column 2 I\nI :----: ...
2	775bf47e-567d-4895-963e-e57113219feb	[-0.03016633912920952, 0.0068882484920322895, ...	./Images/Making-Movies-Manual.pdf_4.jpg	Movies matter because they are more than image...
3	e6f0eea2-7838-45e4-84f5-5aacd167da73	[-0.03584425523877144, -0.023852290585637093, ...	./Images/Making-Movies-Manual.pdf_2.jpg	Introduction\nThis manual will help you make a...

Creating Vertex AI: Vector Search

The code configures and deploys a vector search index on Google Cloud, making it ready to store and search through embeddings.

Embedding size : The number of values used to represent a piece of text in vector form. Larger dimensions mean a denser and potentially more expressive representation.

Dimensions vs. Latency

- Search: Higher-dimensional embeddings can make vector similarity searches slower, especially in large databases.
- Computation: Calculations with larger vectors generally take more time during model training and inference.

```
VECTOR_SEARCH_REGION = "us-central1"
VECTOR_SEARCH_INDEX_NAME = f"{PROJECT_ID}-vector-search-index-ht"
VECTOR_SEARCH_EMBEDDING_DIR = f"{PROJECT_ID}-vector-search-bucket-ht"
VECTOR_SEARCH_DIMENSIONS = 768
```

Save the embeddings in a JSON file

To load the embeddings to Vector Search, we need to save them in JSON files with JSONL format. See more information in the docs at [Input data format and structure](#).

First, export the `id` and `embedding` columns from the DataFrame in JSONL format, and save it.

Then, create a new Cloud Storage bucket and copy the file to it.

```
# save id and embedding as a json file
jsonl_string = embedding_df[["id", "embedding"]].to_json(orient="records", lines=Tru
with open("data.json", "w") as f:
    f.write(jsonl_string)

# show the first few lines of the json file
! head -n 3 data.json
```

```
{"id":"65ab6184-de63-474f-ba54-0360e63b897f","embedding":[-0.0380549245,-0.04563
{"id":"0fd663f0-a499-4545-a42d-79eda6043603","embedding":[0.0479905307,-0.088483
{"id":"775bf47e-567d-4895-963e-e57113219feb","embedding":[-0.0301663391,0.00688
```

```
# Generates a unique ID for session
UID = datetime.now().strftime("%m%d%H%M")

# Creates a GCS bucket
BUCKET_URI = f"gs://{VECTOR_SEARCH_EMBEDDING_DIR}-{UID}"
! gsutil mb -l $LOCATION -p {PROJECT_ID} {BUCKET_URI}
! gsutil cp data.json {BUCKET_URI}
```

```
Creating gs://project-llm-428915-vector-search-bucket-ht-07120233/...
Copying file://data.json [Content-Type=application/json]...
-
Operation completed over 1 objects/50.5 KiB.
```

Create an Index

Now it's ready to load the embeddings to Vector Search. Its APIs are available under the [aiplatform]() package of the SDK.

Create an [MatchingEngineIndex]() with its `create_tree_ah_index` function (Matching Engine is the previous name of Vector Search).

```
# create index
my_index = aiplatform.MatchingEngineIndex.create_tree_ah_index(
    display_name=f"{VECTOR_SEARCH_INDEX_NAME}",
    contents_delta_uri=BUCKET_URI,
    dimensions=768,
    approximate_neighbors_count=20,
    distance_measure_type="DOT_PRODUCT_DISTANCE",
)
```

```
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index:Creating Matc
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index:Create Matchi
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index:MatchingEngir
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index:To use this M
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index:index = aipla
```

By calling the `create_tree_ah_index` function, it starts building an Index. This will take under a few minutes if the dataset is small, otherwise about 50 minutes or more depending on the size of the dataset. You can check status of the index creation on [the Vector Search Console > INDEXES tab](#).

The parameters for creating index

- `contents_delta_uri` : The URI of Cloud Storage directory where you stored the embedding JSON files
- `dimensions` : Dimension size of each embedding. In this case, it is 768 as we are using the embeddings from the Text Embeddings API.
- `approximate_neighbors_count` : how many similar items we want to retrieve in typical cases
- `distance_measure_type` : what metrics to measure distance/similarity between embeddings. In this case it's `DOT_PRODUCT_DISTANCE`

See [the document](#) for more details on creating Index and the parameters.

Create Index Endpoint and deploy the Index

To use the Index, you need to create an [Index Endpoint](#). It works as a server instance accepting query requests for your Index.

```
# create IndexEndpoint
my_index_endpoint = aiplatform.MatchingEngineIndexEndpoint.create(
    display_name=f"{VECTOR_SEARCH_INDEX_NAME}",
    public_endpoint_enabled=True,
)
print(my_index_endpoint)
```

```
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index_endpoint:Crea
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index_endpoint:Crea
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index_endpoint:Matc
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index_endpoint:To u
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index_endpoint:inde
<google.cloud.aiplatform.matching_engine.matching_engine_index_endpoint.Matching
resource name: projects/34837578213/locations/us-east1/indexEndpoints/868565807
```

This tutorial utilizes a Public Endpoint and does not support Virtual Private Cloud (VPC). Unless you have a specific requirement for VPC, we recommend using a Public Endpoint. Despite the term "public" in its name, it does not imply open access to the public internet. Rather, it functions like other endpoints in Vertex AI services, which are secured by default through IAM. Without explicit IAM permissions, as we have previously established, no one can access the endpoint.

With the Index Endpoint, deploy the Index by specifying an unique deployed index ID.

```
# DEPLOYED_INDEX_NAME = VECTOR_SEARCH_INDEX_NAME.replace(
#     "-", "_"
# )  # Can't have - in deployment name, only alphanumeric and _ allowed
# DEPLOYED_INDEX_ID = f"{DEPLOYED_INDEX_NAME}_{UID}"
# # deploy the Index to the Index Endpoint
# my_index_endpoint.deploy_index(index=my_index, deployed_index_id=DEPLOYED_INDEX_II
```

If it is the first time to deploy an Index to an Index Endpoint, it will take around 25 minutes to automatically build and initiate the backend for it. After the first deployment, it will finish in seconds. To see the status of the index deployment, open the Vector Search Console > INDEX ENDPOINTS tab and click the Index Endpoint.

Ask Questions to the PDF

This code snippet establishes a question-answering (QA) system. It leverages a vector search engine to find relevant information from a dataset and then uses the 'gemini-pro' LLM model to generate and refine the final answer to a user's query.

```python
def Test_LLM_Response(txt):
    """
    Determines whether a given text response generated by an LLM indicates a lack

    Args:
        txt (str): The text response generated by the LLM.

    Returns:
        bool: True if the LLM's response suggests it was able to generate a meanin
              False if the response indicates it could not find relevant informati

    This function works by presenting a formatted classification prompt to the LLM
    The prompt includes the original text and specific categories indicating wheth
    The function analyzes the LLM's classification output to make the determinatio
    """

    classification_prompt = f""" Classify the text as one of the following categor
        -Information Present
        -Information Not Present
        Text=The provided context does not contain information.
        Category:Information Not Present
        Text=I cannot answer this question from the provided context.
        Category:Information Not Present
        Text:{txt}
        Category:"""
    classification_response = model.generate_content(classification_prompt).text

    if "Not Present" in classification_response:
        return False  # Indicates that the LLM couldn't provide an answer
    else:
        return True  # Suggests the LLM generated a meaningful response


def get_prompt_text(question, context):
    """
    Generates a formatted prompt string suitable for a language model, combining t

    Args:
        question (str): The user's original question.
        context (str): The relevant text to be used as context for the answer.

    Returns:
        str: A formatted prompt string with placeholders for the question and cont
    """
    prompt = """
      Answer the question using the context below. Respond with only from the text
      Question: {question}
      Context : {context}
      """.format(
        question=question, context=context
    )
```

```
    return prompt


def get_answer(query):
    """
    Retrieves an answer to a provided query using multimodal retrieval augmented g

    This function leverages a vector search system to find relevant text documents
    pre-indexed store of multimodal data. Then, it uses a large language model (LL
    an answer, using the retrieved documents as context.

    Args:
        query (str): The user's original query.

    Returns:
        dict: A dictionary containing the following keys:
            * 'result' (str): The LLM-generated answer.
            * 'neighbor_index' (int): The index of the most relevant document used
                                    (for fetching image path).

    Raises:
        RuntimeError: If no valid answer could be generated within the specified s
    """

    neighbor_index = 0  # Initialize index for tracking the most relevant document
    answer_found_flag = 0  # Flag to signal if an acceptable answer is found
    result = ""  # Initialize the answer string
    # Use a default image if the reference is not found
    page_source = "./Images/blank.jpg"  # Initialize the blank image
    query_embeddings = generate_text_embedding(
        query
    )  # Generate embeddings for the query

    response = my_index_endpoint.find_neighbors(
        deployed_index_id=DEPLOYED_INDEX_ID,
        queries=[query_embeddings],
        num_neighbors=5,
    )  # Retrieve up to 5 relevant documents from the vector store

    while answer_found_flag == 0 and neighbor_index < 4:
        context = embedding_df[
            embedding_df["id"] == response[0][neighbor_index].id
        ].text.values[
            0
```