Making AI Travel Agent

Overview

Introduction

In the rapidly evolving digital landscape, providing personalized recommendations has become crucial for enhancing user experiences. The Travel Itinerary Recommendation Chatbot leverages cutting-edge technologies like Retrieval-Augmented Generation (RAG), Large Language Models (LLMs), LangChain, and vector databases to offer tailored travel plans. This chatbot aims to interact with users to understand their preferences and generate custom itineraries based on pre-existing travel data.

Objective

The goal of this project is to develop a domain-specific application that combines the strengths of an LLM for understanding and processing natural language queries with the efficiency of a vector database for data storage and retrieval. The chatbot will provide personalized travel recommendations using RAG, ensuring that the suggestions are accurate and relevant to the user's stated preferences.

Key Requirements

User Interaction: The chatbot should allow users to input their preferences and needs through a conversational interface. RAG-Based Recommendations: Generate personalized travel recommendations based on user input. Intelligent Responses: Provide relevant information and suggestions based on existing data. Pre-existing Data: Recommendations should be based solely on the existing data stored in the vector database. Backend Integration: Utilize LangChain to manage interaction flow, with details stored in a vector database. Data Fetching: Retrieve the top K (K <= 3) data entries based on user descriptions using similarity search in the vector database. Mock Data: Use prompt engineering to generate mock data for the vector database.

Costs

This tutorial uses billable components of Google Cloud:

- Vertex AI

Learn about [Vertex AI pricing]() and use the [Pricing Calculator]() to generate a cost estimate based on your projected usage.

Objectives

This notebook provides a guide to building a questions answering system using multimodal retrieval augmented generation (RAG).

You will complete the following tasks:

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.

Getting Started

Install Vertex AI SDK and other required packages

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

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.5/3.5 MB 8.8 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 983.6/983.6 kB 18.8 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.3/12.3 MB 16.5 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.1/5.1 MB 23.2 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 73.0/73.0 kB 4.0 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 15.7/15.7 MB 17.2 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 362.4/362.4 kB 12.4 MB/s eta 0:00:00
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 kB 4.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 92.0/92.0 kB 4.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 318.2/318.2 kB 10.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.6/75.6 kB 5.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 141.1/141.1 kB 3.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.1/10.1 MB 40.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.4/62.4 kB 4.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.9/129.9 kB 8.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 126.5/126.5 kB 8.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.9/77.9 kB 4.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.3/58.3 kB 4.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 71.9/71.9 kB 4.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53.6/53.6 kB 2.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 307.7/307.7 kB 28.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 341.4/341.4 kB 31.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.4/3.4 MB 86.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 57.0 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 = "key-transformer-429201-m0"  # @param {type:"strin
LOCATION = "us-east1"  # @param {type:"string"}

# Initialize Vertex AI
import vertexai

vertexai.init(project=PROJECT_ID, location=LOCATION)
```

PROJECT_ID: " key-transformer-429201-m0 "

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 8.5 MB/s eta 0:00:00
```

```
Requirement already satisfied: PyYAML>=5.3 in /usr/local/lib/python3.10/dist-packages (from langchain_community) (6.0.1)
Requirement already satisfied: SQLAlchemy<3,>=1.4 in /usr/local/lib/python3.10/dist-packages (from langchain_community) (2.0
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in /usr/local/lib/python3.10/dist-packages (from langchain_community) (
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/python3.10/dist-packages (from langchain_community)
Requirement already satisfied: langchain-core<0.3.0,>=0.2.12 in /usr/local/lib/python3.10/dist-packages (from langchain_comm
Requirement already satisfied: langsmith<0.2.0,>=0.1.0 in /usr/local/lib/python3.10/dist-packages (from langchain_community)
Requirement already satisfied: numpy<2,>=1 in /usr/local/lib/python3.10/dist-packages (from langchain_community) (1.25.2)
Requirement already satisfied: requests<3,>=2 in /usr/local/lib/python3.10/dist-packages (from langchain_community) (2.31.0)
Requirement already satisfied: tenacity!=8.4.0,<9.0.0,>=8.1.0 in /usr/local/lib/python3.10/dist-packages (from langchain_com
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->lang
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langcha
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->lan
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->l
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->langch
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.
Collecting marshmallow<4.0.0,>=3.18.0 (from dataclasses-json<0.7,>=0.5.7->langchain_community)
  Downloading marshmallow-3.21.3-py3-none-any.whl (49 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.2/49.2 kB 4.7 MB/s eta 0:00:00
Collecting typing-inspect<1,>=0.4.0 (from dataclasses-json<0.7,>=0.5.7->langchain_community)
  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/local/lib/python3.10/dist-packages (from lango
Requirement already satisfied: pydantic<3,>=1 in /usr/local/lib/python3.10/dist-packages (from langchain<0.3.0,>=0.2.7->lang
Requirement already satisfied: jsonpatch<2.0,>=1.33 in /usr/local/lib/python3.10/dist-packages (from langchain-core<0.3.0,>=
Requirement already satisfied: packaging<25,>=23.2 in /usr/local/lib/python3.10/dist-packages (from langchain-core<0.3.0,>=0
Requirement already satisfied: orjson<4.0.0,>=3.9.14 in /usr/local/lib/python3.10/dist-packages (from langsmith<0.2.0,>=0.1.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->lan
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->langchain_commu
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->langchain
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->langchain
Requirement already satisfied: typing-extensions>=4.6.0 in /usr/local/lib/python3.10/dist-packages (from SQLAlchemy<3,>=1.4-
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-packages (from SQLAlchemy<3,>=1.4->langcha
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.10/dist-packages (from jsonpatch<2.0,>=1.33->lango
Requirement already satisfied: annotated-types>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1->lango
Requirement already satisfied: pydantic-core==2.20.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1->langch
Collecting mypy-extensions>=0.3.0 (from typing-inspect<1,>=0.4.0->dataclasses-json<0.7,>=0.5.7->langchain_community)
  Downloading mypy_extensions-1.0.0-py3-none-any.whl (4.7 kB)
Installing collected packages: mypy-extensions, marshmallow, typing-inspect, dataclasses-json, langchain_community
Successfully installed dataclasses-json-0.6.7 langchain_community-0.2.7 marshmallow-3.21.3 mypy-extensions-1.0.0 typing-insp
```

Importing all the libraries

```
import os


import time

import uuid
from datetime import datetime

import fitz
import gradio as gr
import pandas as pd


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


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 Text Embedding with models Gemini Vision Pro

```
multimodal_model = GenerativeModel("gemini-1.0-pro-vision")


text_embedding_model = TextEmbeddingModel.from_pretrained("textembedding-gecko@003")

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_23-2150696455.jpg

# 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 02:29:34--  https://www.hitachi.com/rev/archive/2023/r2023_04/pdf/04a02.pdf
Resolving www.hitachi.com (www.hitachi.com)... 99.84.160.124, 99.84.160.64, 99.84.160.122, ...
Connecting to www.hitachi.com (www.hitachi.com)|99.84.160.124|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1462074 (1.4M) [application/pdf]
Saving to: ‘04a02.pdf’

04a02.pdf           100%[===================>]   1.39M  5.31MB/s    in 0.3s

2024-07-12 02:29:34 (5.31 MB/s) - ‘04a02.pdf’ saved [1462074/1462074]

--2024-07-12 02:29:35--  https://img.freepik.com/free-vector/hand-drawn-no-data-illustration_23-2150696455.jpg
Resolving img.freepik.com (img.freepik.com)... 23.220.103.173, 23.220.103.170, 2600:1407:7400:1d::172e:1731, ...
Connecting to img.freepik.com (img.freepik.com)|23.220.103.173|: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 02:29:35 (289 KB/s) - ‘hand-drawn-no-data-illustration_23-2150696455.jpg’ saved [32694/32694]
```

Split PDF to images and extract data using Gemini Vision Pro

```
# Run the following code for each file
PDF_FILENAME = "035 Travel Sample Lesson.pdf"  # Replace with your filename


# To enhance resolution
zoom_x = 2.0  # horizontal zoom
zoom_y = 2.0  # vertical zoom
mat = fitz.Matrix(zoom_x, zoom_y)

doc = fitz.open(PDF_FILENAME)
for page in doc:
    pix = page.get_pixmap(matrix=mat)
    outpath = f"./Images/{PDF_FILENAME}_{page.number}.jpg"
    pix.save(outpath)


image_names = os.listdir(Image_Path)
Max_images = len(image_names)


page_source = []
page_content = []
page_id = []

p_id = 0
rest_count = 0

while p_id < Max_images:
    try:

        image_path = Image_Path + image_names[p_id]
```

```
        image = Image.load_from_file(image_path)


        prompt_text = "Extract all text content in the image"
        prompt_table = (
            "Detect table in this image. Extract content maintaining the structure"
        )


        contents = [image, prompt_text]
        response = multimodal_model.generate_content(contents)
        text_content = response.text


        contents = [image, prompt_table]
        response = multimodal_model.generate_content(contents)
        table_content = response.text


        print(f"processed image no: {p_id}")
        page_source.append(image_path)
        page_content.append(text_content + "\n" + table_content)
        page_id.append(p_id)
        p_id += 1

    except Exception as err:

        print(err)
        print("Taking Some Rest")
        time.sleep(1)
        rest_count += 1
        if rest_count == 5:
            rest_count = 0
            print(f"Cannot process image no: {image_path}")
            p_id += 1

df = pd.DataFrame(
    {"page_id": page_id, "page_source": page_source, "page_content": page_content}
)
df.head()
```

```
processed image no: 0
processed image no: 1
processed image no: 2
processed image no: 3
processed image no: 4
429 Quota exceeded for aiplatform.googleapis.com/generate_content_requests_per_minute_per_project_per_base_model with base r
Taking Some Rest
429 Quota exceeded for aiplatform.googleapis.com/generate_content_requests_per_minute_per_project_per_base_model with base r
Taking Some Rest
429 Quota exceeded for aiplatform.googleapis.com/generate_content_requests_per_minute_per_project_per_base_model with base r
Taking Some Rest
429 Quota exceeded for aiplatform.googleapis.com/generate_content_requests_per_minute_per_project_per_base_model with base r
Taking Some Rest
429 Quota exceeded for aiplatform.googleapis.com/generate_content_requests_per_minute_per_project_per_base_model with base r
Taking Some Rest
Cannot process image no: ./Images/035 Travel Sample Lesson.pdf_0.jpg
```

	page_id	page_source	page_content
0	0	./Images/035 Travel Sample Lesson.pdf_1.jpg	needs and wants from their customers. As a tr...
1	1	./Images/035 Travel Sample Lesson.pdf_4.jpg	Another characteristic of tour operators is t...
2	2	./Images/035 Travel Sample Lesson.pdf_2.jpg	Escorted Tours\nAn escorted tour can be defin...
3	3	./Images/035 Travel Sample Lesson.pdf_3.jpg	an air and land tour. The tour company will u...
4	4	./Images/blank.jpg	?\n?\nX\n | Header 1 | Header 2 |\n|---|---|\...

Next steps: Generate code with df | View recommended plots

Generate Text Embeddings

Using gecko

```
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	36a4d01c-032a-49cd-9758-0b47531d75ab	[0.028441298753023148, -0.03281616047024727, -...	./Images/035 Travel Sample Lesson.pdf_1.jpg	needs and wants from their customers. As a tra...
1	67091fe1-7812-4871-afec-555d1817c917	[0.025769606232643127, -0.056975144892930984, ...	./Images/035 Travel Sample Lesson.pdf_4.jpg	Another characteristic of tour operators is th...
2	715d6807-a929-45a6-9acf-4a3ba159b767	[0.03668646141886711, -0.06587941944599152, -0...	./Images/035 Travel Sample Lesson.pdf_2.jpg	Escorted Tours\nAn escorted tour can be define...
3	282884e5-506b-4e60-bca8-	[0.02981729619204998,	./Images/035 Travel Sample	an air and land tour. The tour

Next steps: Generate code with `embedding_df` View recommended plots

Creating Vertex AI: Vector Search

```
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

```
# save id and embedding as a json file
jsonl_string = embedding_df[["id", "embedding"]].to_json(orient="records", lines=True)
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":"36a4d01c-032a-49cd-9758-0b47531d75ab","embedding":[0.0284412988,-0.0328161605,-0.032952752,0.0020614653,0.0678828061,
{"id":"67091fe1-7812-4871-afec-555d1817c917","embedding":[0.0257696062,-0.0569751449,-0.045261275,0.0049706362,0.0683058426,
{"id":"715d6807-a929-45a6-9acf-4a3ba159b767","embedding":[0.0366864614,-0.0658794194,-0.0391372852,-0.0022027516,0.060445986
```


```
# 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://key-transformer-429201-m0-vector-search-bucket-ht-07120232/...
Copying file://data.json [Content-Type=application/json]...
-
Operation completed over 1 objects/50.5 KiB.
```

Create an Index

Create an [MatchingEngineIndex]

```
# 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 MatchingEngineIndex
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index:Create MatchingEngineIndex backing LRO: projects/21974038
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index:MatchingEngineIndex created. Resource name: projects/2197
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index:To use this MatchingEngineIndex in another session:
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index:index = aiplatform.MatchingEngineIndex('projects/21974038
```

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

```
# 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:Creating MatchingEngineIndexEndpoint
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index_endpoint:Create MatchingEngineIndexEndpoint backing LRO:
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index_endpoint:MatchingEngineIndexEndpoint created. Resource na
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index_endpoint:To use this MatchingEngineIndexEndpoint in anoth
INFO:google.cloud.aiplatform.matching_engine.matching_engine_index_endpoint:index_endpoint = aiplatform.MatchingEngineIndexE
<google.cloud.aiplatform.matching_engine.matching_engine_index_endpoint.MatchingEngineIndexEndpoint object at 0x7d9b8617f40
resource name: projects/21974038498/locations/us-east1/indexEndpoints/2316442301305454592
```

```
# 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_ID)
```

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.

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

    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 meaningful answer,
              False if the response indicates it could not find relevant information.

    This function works by presenting a formatted classification prompt to the LLM (`gemini_pro_model`).
    The prompt includes the original text and specific categories indicating whether sufficient information was available.
    The function analyzes the LLM's classification output to make the determination.
    """

    classification_prompt = f""" Classify the text as one of the following categories:
        -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 the provided question and context.

    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 context, designed to guide the language model's a
    """
    prompt = """
      Answer the question using the context below. Respond with only from the text provided
      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 generation (RAG).

    This function leverages a vector search system to find relevant text documents from a
    pre-indexed store of multimodal data. Then, it uses a large language model (LLM) to generate
    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 generation
                                     (for fetching image path).

    Raises:
        RuntimeError: If no valid answer could be generated within the specified search attempts.
    """

    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
        ]  # Extract text context from the relevant document

        prompt = get_prompt_text(
            query, context
        )  # Create a prompt using the question and context
        result = model.generate_content(prompt).text  # Generate an answer with the LLM

        if Test_LLM_Response(result):
            answer_found_flag = 1  # Exit loop when getting a valid response
        else:
            neighbor_index += (
                1  # Try the next retrieved document if the answer is unsatisfactory
            )

    if answer_found_flag == 1:
        page_source = embedding_df[
            embedding_df["id"] == response[0][neighbor_index].id
        ].page_source.values[
            0
        ]  # Extract image_path from the relevant document
    return result, page_source


query = (
    "what is the steps of Transformer Manufacturing Flow ?")

result, page_source = get_answer(query)
print(result)
```

```
I am sorry, but the context does not contain information about Transformer Manufacturing Flow, so I am unable to answer the
```

Ask Questions to the PDF using Gradio UI

this code creates a web-based frontend for your question-answering system, allowing users to easily enter queries and see the results along with relevant images.

```
import gradio as gr
from PIL import Image as PIL_Image

def gradio_query(query):
    print(query)

    # Retrieve the answer from your QA system
    result, image_path = get_answer(query)
    print("result here")
    print(result)

    try:
        # Attempt to fetch the source image reference
        image = PIL_Image.open(image_path)  # Open the reference image
    except:
        # Use a default image if the reference is not found
        image = PIL_Image.open("./Images/blank.jpg")

    return result, image  # Return both the text answer and the image
```

```
gr.close_all()  # Ensure a clean Gradio interface
with gr.Blocks() as demo:
    with gr.Row():
        with gr.Column():
            # Input / Output Components
            query = gr.Textbox(label="Query", info="Enter your query")
            btn_enter = gr.Button("Process")
            answer = gr.Textbox(label="Response", interactive=False)  # Use gr.Textbox for plain text response
            btn_clear = gr.Button("Clear")
        with gr.Column():
            image = gr.Image(label="Reference", visible=True)

    # Button Click Event
    btn_enter.click(fn=gradio_query, inputs=query, outputs=[answer, image])
    btn_clear.click(lambda: ("", None), inputs=None, outputs=[query, answer, image])

demo.launch(share=True, debug=True, inbrowser=True)  # Launch the Gradio app
```

```
Colab notebook detected. This cell will run indefinitely so that you can see err
Running on public URL: https://5d70bf6db47134eeb5.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades
```

What Are Tours and Vacation Packages?

Tours and vacation packages offer a wide range of options, from relaxing on a beach in the West Indies to trekking through the Himalayas.

Here's a breakdown of the two main types:

Escorted Tours: These involve traveling with a group led by a professional guide. They typically include transportation, accommodation, meals, and planned activities.

Independent Tours: These offer more flexibility, allowing you to explore on your own while benefiting from pre-arranged elements like flights and hotels. Some independent tours also include organized activities.

Clear

```
What Are Tours and Vacation Packages?
result here
## What Are Tours and Vacation Packages?

Tours and vacation packages offer a wide range of options, from relaxing on a be

Here's a breakdown of the two main types:

**Escorted Tours:** These involve traveling with a group led by a professional g

**Independent Tours:** These offer more flexibility, allowing you to explore on
```