Custom Embeddings with AI Applications

experimentLabschedule1 houruniversal\_currency\_altNo costshow\_chartIntermediate

infoThis lab may incorporate AI tools to support your learning.

**Overview**

This lab serves as a guide to utilizing text-embedding-004 in Vertex AI for obtaining text embeddings. It walks you through the process of converting these embeddings into the required format for Vertex AI Search. Additionally, the lab provides detailed instructions on creating a search application that utilizes custom embeddings, ensuring a seamless integration of advanced text-based search capabilities.

**Objective**

This lab showcases steps to:

* Retrieve text embeddings using the [Vertex AI API](https://cloud.google.com/vertex-ai/docs/generative-ai/embeddings/get-text-embeddings)
* Transform embeddings into the [format required by Vertex AI Search](https://cloud.google.com/generative-ai-app-builder/docs/prepare-data#unstructured)
* [Create a search application with custom embeddings](https://cloud.google.com/generative-ai-app-builder/docs/bring-embeddings)

**Setup and requirements**

**Before you click the Start Lab button**

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

**What you need**

To complete this lab, you need:

* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

**Note:** If you already have your own personal Google Cloud account or project, do not use it for this lab.

**Note:** If you are using a Pixelbook, open an Incognito window to run this lab.

**How to start your lab and sign in to the Google Cloud Console**

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.



2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



***Tip:*** Open the tabs in separate windows, side-by-side.

 If you see the **Choose an account** page, click **Use Another Account**.



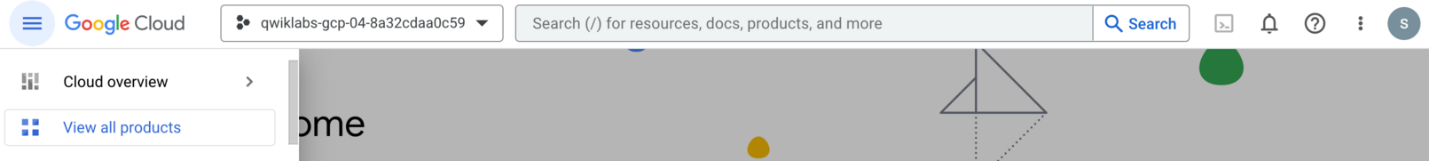
3. In the **Sign in** page, paste the username that you copied from the Connection Details panel. Then copy and paste the password.

***Important:*** You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

4. Click through the subsequent pages:

* Accept the terms and conditions.
* Do not add recovery options or two-factor authentication (because this is a temporary account).
* Do not sign up for free trials.

After a few moments, the Cloud Console opens in this tab.

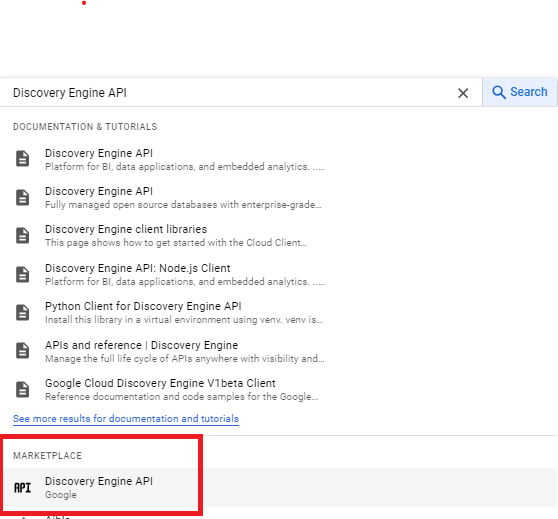
**Note:** You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left. 

**Task 1. Enable API**

In this section, let's enable the discovery engine API.

To enable the Discovery Engine API, follow these steps:

1. Type **Discovery Engine API** into the top search bar of the Google Cloud Console, choose the result as shown in the following image.

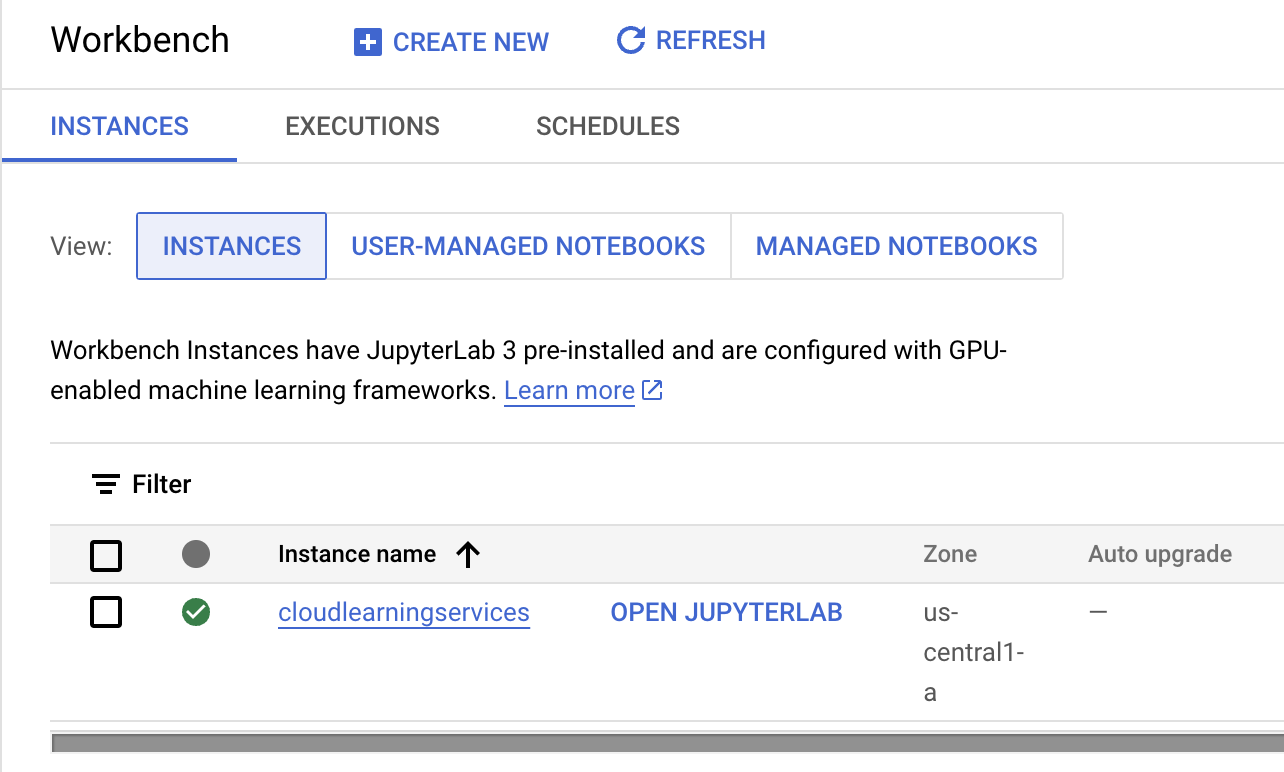


1. Click **Enable**.

**Task 2. Open Python Notebook and Install Packages**

1. In the Google Cloud Console, on the **Navigation menu**, click **Vertex AI > Workbench**.
2. Find the **cloudlearningservices** instance and click on the **Open JupyterLab** button.

The JupyterLab interface for your Workbench instance will open in a new browser tab.



1. On the **Launcher**, under **Notebook**, click on **Python 3** to open a new python notebook.
2. Install required packages by the running the following command in the first cell of the notebook. Either click the play play button at the top or click **SHIFT+ENTER** keys on your keyboard to execute the cell.

%pip install -q --upgrade --user google-cloud-aiplatform google-cloud-discoveryengine google-cloud-storage google-cloud-bigquery[pandas] google-cloud-bigquery-storage pandas ipywidgets

Copied!

%load\_ext google.cloud.bigquery

Copied!

1. To use the newly installed packages in this Jupyter runtime, it is recommended to restart the runtime. Restart the kernel by running the below code snippet or clicking the refresh button restart kernel at the top, followed by clicking **Restart** button.

# Automatically restart kernel after installs so that your environment can access the new packages

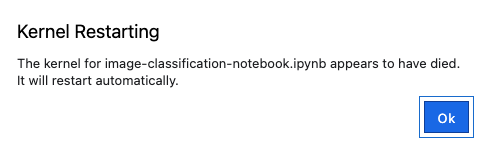
import IPython

app = IPython.Application.instance()

app.kernel.do\_shutdown(True)

Copied!

**Output**:



After the restart is complete, click **Ok** on the prompt to continute.

1. Run the following code snippet in the next cell. This code imports various python libraries and modules to perform tasks related to Google Cloud Platform (GCP) services, specifically related to Vertex AI, BigQuery, Cloud Storage, and Discovery Engine.

from typing import List

import requests

import subprocess

import time

from google.api\_core.client\_options import ClientOptions

from google.api\_core.exceptions import GoogleAPICallError

from google.cloud import bigquery

from google.cloud import discoveryengine\_v1alpha as discoveryengine

from google.cloud import storage

from tqdm import tqdm # to show a progress bar

import vertexai

from vertexai.language\_models import TextEmbeddingModel, TextEmbeddingInput

tqdm.pandas()

Copied!

1. Run the below code snippet in the next cell. This code sets up project information and initializes the Vertex AI SDK for a Google Cloud Platform (GCP) project.

# Define project information for Vertex AI

PROJECT\_ID = "qwiklabs-gcp-00-690e50226636" # @param {type:"string"}

LOCATION = "us-central1" # @param {type:"string"}

# Initialize Vertex AI SDK

vertexai.init(project=PROJECT\_ID, location=LOCATION)

Copied!

**Task 3. Create embeddings with Vertex AI**

Data Preparation

We will be using [the Stack Overflow public dataset](https://console.cloud.google.com/marketplace/product/stack-exchange/stack-overflow) hosted on BigQuery table bigquery-public-data.stackoverflow.posts\_questions. This is a very big dataset with 23 million rows that doesn't fit into memory. We are going to limit it to 500 rows for this lab.

In this task, we will:

* Fetch the data from BigQuery
* Concat the Title and Body, and create embeddings from the text.

1. Run the following code snippet in the next cell, that connects to Google BigQuery, executes a SQL query to retrieve information from the Stack Overflow dataset, loads the results into a Pandas DataFrame, and then performs some data manipulation.

bq\_client = bigquery.Client(project=PROJECT\_ID)

query = f"""

SELECT

DISTINCT

q.id,

q.title,

q.body,

q.answer\_count,

q.comment\_count,

q.creation\_date,

q.last\_activity\_date,

q.score,

q.tags,

q.view\_count

FROM

`bigquery-public-data.stackoverflow.posts\_questions` AS q

WHERE

q.score > 0

ORDER BY

q.view\_count DESC

LIMIT

500;

"""

# Load the BQ Table into a Pandas Dataframe

df = bq\_client.query(query).result().to\_dataframe()

# Convert ID to String

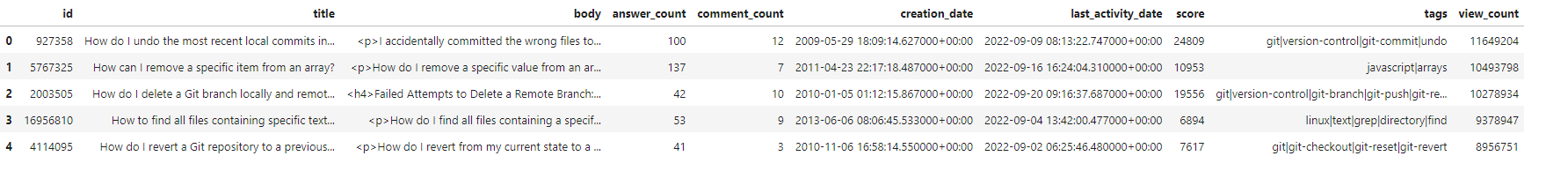
df["id"] = df["id"].apply(str)

# examine the data

df.head()

Copied!

**Output**:



Call the API to generate embeddings

With the Stack Overflow dataset, we will use the title and body columns (the question title and description) and generate embedding for it with Embeddings for Text API. The API is available under the vertexai package of the SDK.

From the package, import TextEmbeddingModel and get a model.

For more information, refer to:

* [Vertex AI: Get Text Embeddings](https://cloud.google.com/vertex-ai/docs/generative-ai/embeddings/get-text-embeddings)
* [Vertex AI: Model versions and lifecycle](https://cloud.google.com/vertex-ai/docs/generative-ai/learn/model-versioning)

1. Run the following code snippet in the next cell, that loads a pre-trained text embeddings model.

# Load the text embeddings model

model = TextEmbeddingModel.from\_pretrained("text-embedding-004")

Copied!

1. Run the following code snippet in the next cell, to define a python function named get\_embeddings\_wrapper that takes a list of texts and an optional batch size as input parameters. The function uses the previously loaded model (a text embeddings model) to obtain embeddings for the provided texts.

# Get embeddings for a list of texts

def get\_embeddings\_wrapper(texts, batch\_size: int = 50) -> List:

embs = []

for i in tqdm(range(0, len(texts), batch\_size)):

# Create embeddings optimized for document retrieval

# (supported in textembedding-gecko@002 and later)

result = model.get\_embeddings(

[

TextEmbeddingInput(text=text, task\_type="RETRIEVAL\_DOCUMENT")

for text in texts[i : i + batch\_size]

]

)

embs.extend([e.values for e in result])

return embs

Copied!

1. This code snippet modifies the previously loaded DataFrame (df) by combining the title and body columns into a new title\_body column. Then, it uses the get\_embeddings\_wrapper function to obtain text embeddings for each combined title and body, and the resulting embeddings are added as a new embedding column to the DataFrame. Finally, the first few rows of the updated DataFrame are displayed.

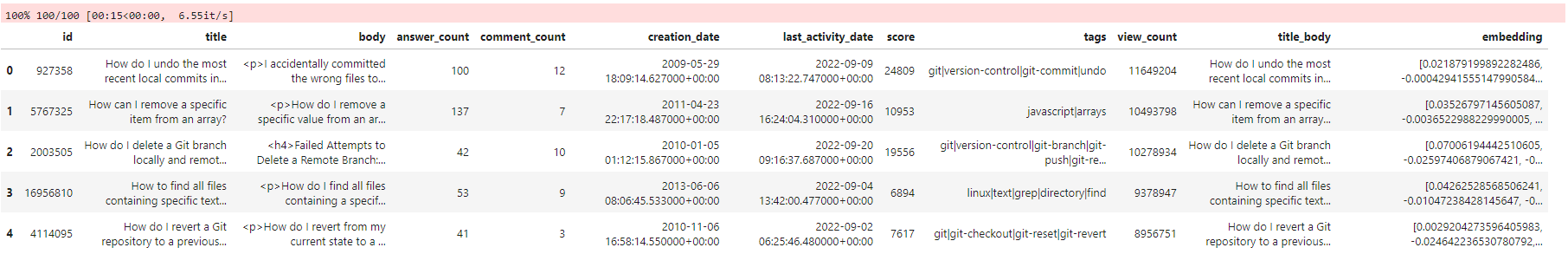
df["title\_body"] = df["title"] + "\n" + df["body"]

df = df.assign(embedding=get\_embeddings\_wrapper(df.title\_body))

df.head()

Copied!

**Output**:



**Task 4. Scrape HTML from Question Pages**

1. Run the following code snippet in the next cell to set up necessary project information.

BUCKET\_NAME = "qwiklabs-gcp-00-690e50226636"

BUCKET\_URI = "gs://qwiklabs-gcp-00-690e50226636"

REGION = "us-east4"

PROJECT\_ID = "qwiklabs-gcp-00-690e50226636"

Copied!

1. Run the following code snippet in the next cell, to create the Google Cloud Storage bucket.

! gsutil mb -l $REGION -p $PROJECT\_ID $BUCKET\_URI

Copied!

**Output**:

Bucket-Output

1. Create directories within the bucket.

%%bash

# Set your Google Cloud Storage bucket name

BUCKET\_NAME="gs://qwiklabs-gcp-00-690e50226636"

# Array of top-level directory names you want to create

TOP\_LEVEL\_DIRECTORIES=("embeddings-stackoverflow")

# Loop through the top-level array and create directories

for TOP\_LEVEL\_DIRECTORY in "${TOP\_LEVEL\_DIRECTORIES[@]}"; do

gsutil -m cp -r /dev/null "$BUCKET\_NAME/$TOP\_LEVEL\_DIRECTORY/"

# Array of subdirectory names you want to create inside the top-level directory

SUBDIRECTORIES=("html")

# Loop through the subdirectories array and create subdirectories inside the top-level directory

for SUBDIRECTORY in "${SUBDIRECTORIES[@]}"; do

gsutil -m cp -r /dev/null "$BUCKET\_NAME/$TOP\_LEVEL\_DIRECTORY/$SUBDIRECTORY/"

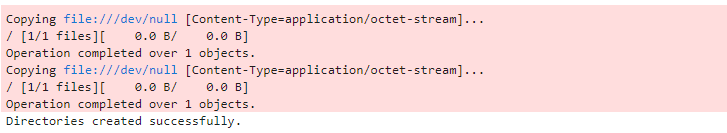
done

done

echo "Directories created successfully."

Copied!

**Output**:



1. Run the following code snippet in the next cell, to define a Python function named scrape\_question that performs the following tasks:

* It sends an HTTP GET request to a specified URL (question\_url) to scrape the content of a question page.
* If the request is successful (HTTP status code 200) and the response contains content, it uploads the HTML content of the question page to Google Cloud Storage (GCS).
* The GCS URI (Uniform Resource Identifier) of the uploaded HTML file is returned.

JSONL\_MIME\_TYPE = "application/jsonl"

HTML\_MIME\_TYPE = "text/html"

BUCKET\_NAME = "qwiklabs-gcp-00-690e50226636"

DIRECTORY = "embeddings-stackoverflow"

BLOB\_PREFIX = f"{DIRECTORY}/html/"

GCS\_URI\_PREFIX = f"gs://{BUCKET\_NAME}/{BLOB\_PREFIX}"

storage\_client = storage.Client()

bucket = storage\_client.bucket(BUCKET\_NAME)

def scrape\_question(question\_url: str) -> str:

response = requests.get(question\_url)

if response.status\_code != 200 or not response.content:

print(f"URL: {question\_url} Code: {response.status\_code}")

return None

print(f"Scraping {question\_url}")

link\_title = response.url.split("/")[-1] + ".html"

gcs\_uri = f"{GCS\_URI\_PREFIX}{link\_title}"

# Upload HTML to Google Cloud Storage

blob = bucket.blob(f"{BLOB\_PREFIX}{link\_title}")

blob.upload\_from\_string(response.content, content\_type=HTML\_MIME\_TYPE)

time.sleep(1)

return gcs\_uri

Copied!

1. The following code snippet has two main parts: it constructs URLs for Stack Overflow questions based on their IDs and then scrapes the HTML content from each of these URLs before uploading it to Google Cloud Storage (GCS). Run the following code snippet in the next cell.

# Get the published URL from the ID

QUESTION\_BASE\_URL = "https://stackoverflow.com/questions/"

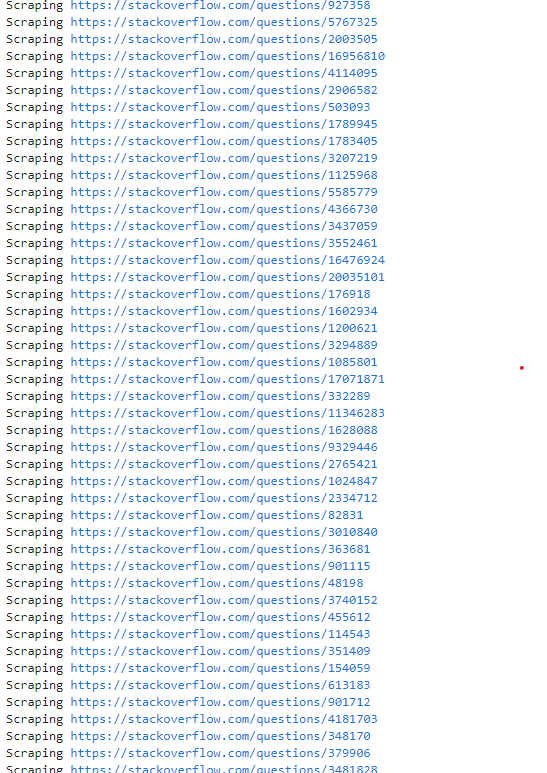
df["question\_url"] = df["id"].apply(lambda x: f"{QUESTION\_BASE\_URL}{x}")

# Scrape HTML from stackoverflow.com and upload to GCS

df["gcs\_uri"] = df["question\_url"].apply(scrape\_question)

Copied!

**Output**:



**Note:** Please wait for 10 minutes to finish the scrapping..

1. In the next cell, restructure the embeddings data to JSONL to follow the [Vertex AI Search format (Unstructured with Metadata)](https://cloud.google.com/generative-ai-app-builder/docs/prepare-data). This format is required to use custom embeddings.

EMBEDDINGS\_FIELD\_NAME = "embedding\_vector"

def format\_row(row):

return {

"id": row["id"],

"content": {"mimeType": HTML\_MIME\_TYPE, "uri": row["gcs\_uri"]},

"structData": {

EMBEDDINGS\_FIELD\_NAME: row["embedding"],

"title": row["title"],

"body": row["body"],

"question\_url": row["question\_url"],

"answer\_count": row["answer\_count"],

"creation\_date": row["creation\_date"],

"score": row["score"],

},

}

vais\_embeddings = (

df.apply(format\_row, axis=1)

.to\_json(orient="records", lines=True, force\_ascii=False)

.replace("\/", "/") # To prevent escaping the / characters

)

Copied!

1. In the next cell, upload the JSONL file to Google Cloud Storage.

jsonl\_filename = f"{DIRECTORY}/vais\_embeddings.jsonl"

embeddings\_file = f"gs://{BUCKET\_NAME}/{jsonl\_filename}"

blob = bucket.blob(jsonl\_filename)

blob.upload\_from\_string(vais\_embeddings, content\_type=JSONL\_MIME\_TYPE)

Copied!

**Task 5. Set up AI Applications**

1. In the next cell, set up client options for interacting with the Google Cloud Vertex AI Discovery Engine service. It specifies the API endpoint based on the provided DATA\_STORE\_LOCATION.

DATA\_STORE\_LOCATION = "global"

client\_options = (

ClientOptions(api\_endpoint=f"{DATA\_STORE\_LOCATION}-discoveryengine.googleapis.com")

if DATA\_STORE\_LOCATION != "global"

else None

)

Copied!

1. In the next cell, define several functions that interact with the Google Cloud Vertex AI Discovery Engine service. These functions are responsible for creating a data store, updating its schema, importing documents, and creating a search engine.

def create\_data\_store(

project\_id: str, location: str, data\_store\_name: str, data\_store\_id: str

):

# Create a client

client = discoveryengine.DataStoreServiceClient(client\_options=client\_options)

# Initialize request argument(s)

data\_store = discoveryengine.DataStore(

display\_name=data\_store\_name,

industry\_vertical="GENERIC",

content\_config="CONTENT\_REQUIRED",

solution\_types=["SOLUTION\_TYPE\_SEARCH"],

)

request = discoveryengine.CreateDataStoreRequest(

parent=discoveryengine.DataStoreServiceClient.collection\_path(

project\_id, location, "default\_collection"

),

data\_store=data\_store,

data\_store\_id=data\_store\_id,

)

operation = client.create\_data\_store(request=request)

try:

operation.result()

except GoogleAPICallError:

pass

def update\_schema(

project\_id: str,

location: str,

data\_store\_id: str,

):

client = discoveryengine.SchemaServiceClient(client\_options=client\_options)

schema = discoveryengine.Schema(

name=client.schema\_path(project\_id, location, data\_store\_id, "default\_schema"),

struct\_schema={

"$schema": "https://json-schema.org/draft/2020-12/schema",

"type": "object",

"properties": {

EMBEDDINGS\_FIELD\_NAME: {

"type": "array",

"keyPropertyMapping": "embedding\_vector",

"dimension": 768,

"items": {"type": "number"},

}

},

},

)

operation = client.update\_schema(

request=discoveryengine.UpdateSchemaRequest(schema=schema)

)

print("Waiting for operation to complete...")

response = operation.result()

# Handle the response

print(response)

def import\_documents(

project\_id: str,

location: str,

data\_store\_id: str,

gcs\_uri: str,

):

client = discoveryengine.DocumentServiceClient(client\_options=client\_options)

# The full resource name of the search engine branch.

# e.g. projects/{project}/locations/{location}/dataStores/{data\_store\_id}/branches/{branch}

parent = client.branch\_path(

project=project\_id,

location=location,

data\_store=data\_store\_id,

branch="default\_branch",

)

request = discoveryengine.ImportDocumentsRequest(

parent=parent,

gcs\_source=discoveryengine.GcsSource(input\_uris=[gcs\_uri]),

# Options: `FULL`, `INCREMENTAL`

reconciliation\_mode=discoveryengine.ImportDocumentsRequest.ReconciliationMode.FULL,

)

# Make the request

operation = client.import\_documents(request=request)

def create\_engine(

project\_id: str, location: str, data\_store\_name: str, data\_store\_id: str

):

client = discoveryengine.EngineServiceClient(client\_options=client\_options)

# Initialize request argument(s)

config = discoveryengine.Engine.SearchEngineConfig(

search\_tier="SEARCH\_TIER\_ENTERPRISE", search\_add\_ons=["SEARCH\_ADD\_ON\_LLM"]

)

engine = discoveryengine.Engine(

display\_name=data\_store\_name,

solution\_type="SOLUTION\_TYPE\_SEARCH",

industry\_vertical="GENERIC",

data\_store\_ids=[data\_store\_id],

search\_engine\_config=config,

)

request = discoveryengine.CreateEngineRequest(

parent=discoveryengine.DataStoreServiceClient.collection\_path(

project\_id, location, "default\_collection"

),

engine=engine,

engine\_id=engine.display\_name,

)

# Make the request

operation = client.create\_engine(request=request)

response = operation.result(timeout=90)

Copied!

1. In the next cell, set the project related variables.

DATA\_STORE\_NAME = "stackoverflow-embeddings"

DATA\_STORE\_ID = f"{DATA\_STORE\_NAME}-id"

Copied!

1. In the next cell, initialize and configure a search application in Google Cloud Vertex AI Discovery Engine, including creating a data store, updating its schema for embeddings, importing documents, and creating a search engine attached to the data store.

# Create a Data Store

create\_data\_store(PROJECT\_ID, DATA\_STORE\_LOCATION, DATA\_STORE\_NAME, DATA\_STORE\_ID)

# Update the Data Store Schema for embeddings

update\_schema(PROJECT\_ID, DATA\_STORE\_LOCATION, DATA\_STORE\_ID)

# Import the embeddings JSONL file

import\_documents(PROJECT\_ID, DATA\_STORE\_LOCATION, DATA\_STORE\_ID, embeddings\_file)

# Create a Search App and attach the Data Store

create\_engine(PROJECT\_ID, DATA\_STORE\_LOCATION, DATA\_STORE\_NAME, DATA\_STORE\_ID)

Copied!

**Output**:



Next, we need to set the embedding specification for the data store. We will set the same specifications for all search requests: 0.5 \* relevance\_score.

This is not supported in client libraries, so we will use the requests module to make a REST request Documentation: [Bring Embeddings](https://cloud.google.com/generative-ai-app-builder/docs/bring-embeddings#global)

1. Run the following code snippet in the next cell, that retrieves an access token using gcloud auth print-access-token, and then it sends a PATCH request to update the serving configuration of the search application in Google Cloud Vertex AI Discovery Engine. The request includes the embedding configuration and a ranking expression, and the server's response is printed.

access\_token = (

subprocess.check\_output(["gcloud", "auth", "print-access-token"])

.decode("utf-8")

.strip()

)

response = requests.patch(

url=f"https://discoveryengine.googleapis.com/v1alpha/projects/{PROJECT\_ID}/locations/{DATA\_STORE\_LOCATION}/collections/default\_collection/dataStores/{DATA\_STORE\_ID}/servingConfigs/default\_search?updateMask=embeddingConfig,rankingExpression",

headers={

"Authorization": f"Bearer {access\_token}",

"Content-Type": "application/json; charset=utf-8",

"X-Goog-User-Project": PROJECT\_ID,

},

json={

"name": f"projects/{PROJECT\_ID}/locations/{DATA\_STORE\_LOCATION}/collections/default\_collection/dataStores/{DATA\_STORE\_ID}/servingConfigs/default\_search",

"embeddingConfig": {"fieldPath": EMBEDDINGS\_FIELD\_NAME},

"ranking\_expression": "0.5 \* relevance\_score",

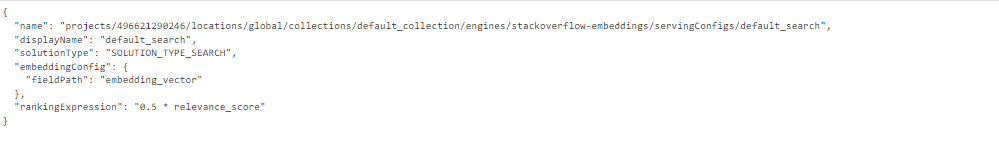
},

)

print(response.text)

Copied!

**Output**:



**Task 6. Test Search Application**

1. Run the following code snippet in the next cell, to define a function named search\_data\_store that performs a search operation on a Google Cloud Vertex AI Discovery Engine data store.

def search\_data\_store(

project\_id: str,

location: str,

data\_store\_id: str,

search\_query: str,

) -> List[discoveryengine.SearchResponse]:

# Create a client

client = discoveryengine.SearchServiceClient(client\_options=client\_options)

# The full resource name of the search engine serving config

# e.g. projects/{project\_id}/locations/{location}/dataStores/{data\_store\_id}/servingConfigs/{serving\_config\_id}

serving\_config = client.serving\_config\_path(

project=project\_id,

location=location,

data\_store=data\_store\_id,

serving\_config="default\_config",

)

# Optional: Configuration options for search

# Refer to the `ContentSearchSpec` reference for all supported fields:

# https://cloud.google.com/python/docs/reference/discoveryengine/latest/google.cloud.discoveryengine\_v1.types.SearchRequest.ContentSearchSpec

content\_search\_spec = discoveryengine.SearchRequest.ContentSearchSpec(

# For information about snippets, refer to:

# https://cloud.google.com/generative-ai-app-builder/docs/snippets

snippet\_spec=discoveryengine.SearchRequest.ContentSearchSpec.SnippetSpec(

return\_snippet=True

),

# For information about search summaries, refer to:

# https://cloud.google.com/generative-ai-app-builder/docs/get-search-summaries

summary\_spec=discoveryengine.SearchRequest.ContentSearchSpec.SummarySpec(

summary\_result\_count=5,

include\_citations=True,

ignore\_adversarial\_query=True,

ignore\_non\_summary\_seeking\_query=True,

),

)

# Refer to the `SearchRequest` reference for all supported fields:

# https://cloud.google.com/python/docs/reference/discoveryengine/latest/google.cloud.discoveryengine\_v1.types.SearchRequest

request = discoveryengine.SearchRequest(

serving\_config=serving\_config,

query=search\_query,

page\_size=10,

content\_search\_spec=content\_search\_spec,

query\_expansion\_spec=discoveryengine.SearchRequest.QueryExpansionSpec(

condition=discoveryengine.SearchRequest.QueryExpansionSpec.Condition.AUTO,

),

spell\_correction\_spec=discoveryengine.SearchRequest.SpellCorrectionSpec(

mode=discoveryengine.SearchRequest.SpellCorrectionSpec.Mode.AUTO

),

)

response = client.search(request)

return response

Copied!

1. Run the following code snippet in the next cell, to perform a search operation on a Google Cloud Vertex AI Discovery Engine data store using a specified search query and prints the summary text of the search response.

search\_query = "How do I create an array in Java?"

response = search\_data\_store(

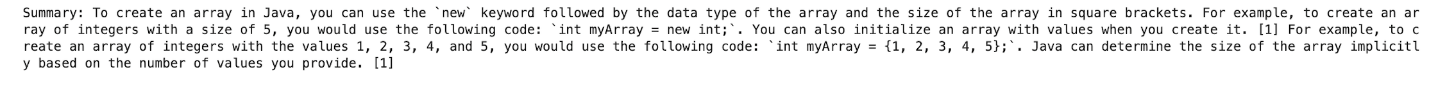
PROJECT\_ID, DATA\_STORE\_LOCATION, DATA\_STORE\_ID, search\_query

)

print(f"Summary: {response.summary.summary\_text}")

Copied!

**Output**:



**Note:** If you receive an error, then re-run the same cell after a 3-4 minutes.

If you see the message **No results could be found. Try rephrasing the search query.** in the output wait for few more minutes and re-run the command.

**Congratulations!**

At the end of this lab, you have gained an understanding of utilizing text-embedding-004 in Vertex AI to obtain text embeddings and converting them into the required format for Vertex AI Search. Additionally, you have learned how to create a search application that leverages custom embeddings, facilitating a seamless integration of advanced text-based search capabilities.

**Manual Last Updated March 06, 2025**

**Lab Last Tested March 06, 2025**

Copyright 2023 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are associated.

Top of Form

Bottom of Form