

Google Cloud Skills Boost for Partners

Course > Orchestrate LLM solutions with LangChain >

[End Lab](#)

00:46:01

Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked.

[Learn more.](#)[Open Google Cloud console](#)

GCP Username

student-02-f705f82c66df



GCP Password

iX1PFTFKQFFS



GCP Project ID

qwiklabs-gcp-04-3ffe0f9!



Building AI-powered data-driven applications using pgvector,

[Previous](#)[Next >](#)

Lab 1 hour No cost Intermediate



This lab may incorporate AI tools to support your learning.

Lab instructions and tasks



Overview

Scenario

Setup and requirements

Task 1. Enable APIs

Task 2. Open Python notebook and install packages

Task 3. Create a Cloud SQL Instance and PostgreSQL database.

Task 4. Download and load the dataset in PostgreSQL

Task 5. Generate vector embeddings using a Text Embedding model

Task 6. Use pgvector to store the generated embeddings within PostgreSQL

< Previous

Next >

Task 8. LLMs and LangChain

Congratulations

Overview

This hands-on lab will show you how you can add vertex AI features to your own applications with just a few lines of code using pgvector, LangChain and LLMs on Google Cloud. This lab help you to understand, how you can add vertex AI features to your own applications with just a few lines of code using pgvector, LangChain and LLMs on Google Cloud.

In this lab, we will build together a sample Python application that will be able to understand and respond to human language queries about the relational data stored in your PostgreSQL database. In fact, we will further push the creative limits

< Previous

Next >

Scenario

This lab uses an e-commerce company that runs an online marketplace for buying and selling children toys. This company wants to add new generative AI experiences in their e-commerce applications for both buyers and sellers on their platform.

Dataset:

- The dataset for this notebook has been sampled and created from a larger public retail dataset available at [Kaggle](#). The sampled dataset used in this notebook has only about 800 toy products, while the public dataset has over 370,000 products in different categories.

The goals are:

< Previous

Next >

describe their needs in simple English text, along with regular filters (like price, etc.)

- (Usecase 2) For sellers: Add a new AI-powered content generation feature, where sellers will get auto-generated item description suggestions for new products that they want to add to the platform.

Objective

- How to use the [pgvector extension](#) to store and search vector embeddings in PostgreSQL.
- How to use the open-source [LangChain framework](#) to develop applications powered by large language models. LangChain makes it easier to develop and deploy applications against any LLM model in a vendor-agnostic manner.
- How to use the powerful features in [Google models made available through Vertex AI](#).

What you will do

1. Download the dataset and load it into a PostgreSQL table called `products`

< Previous

Next >

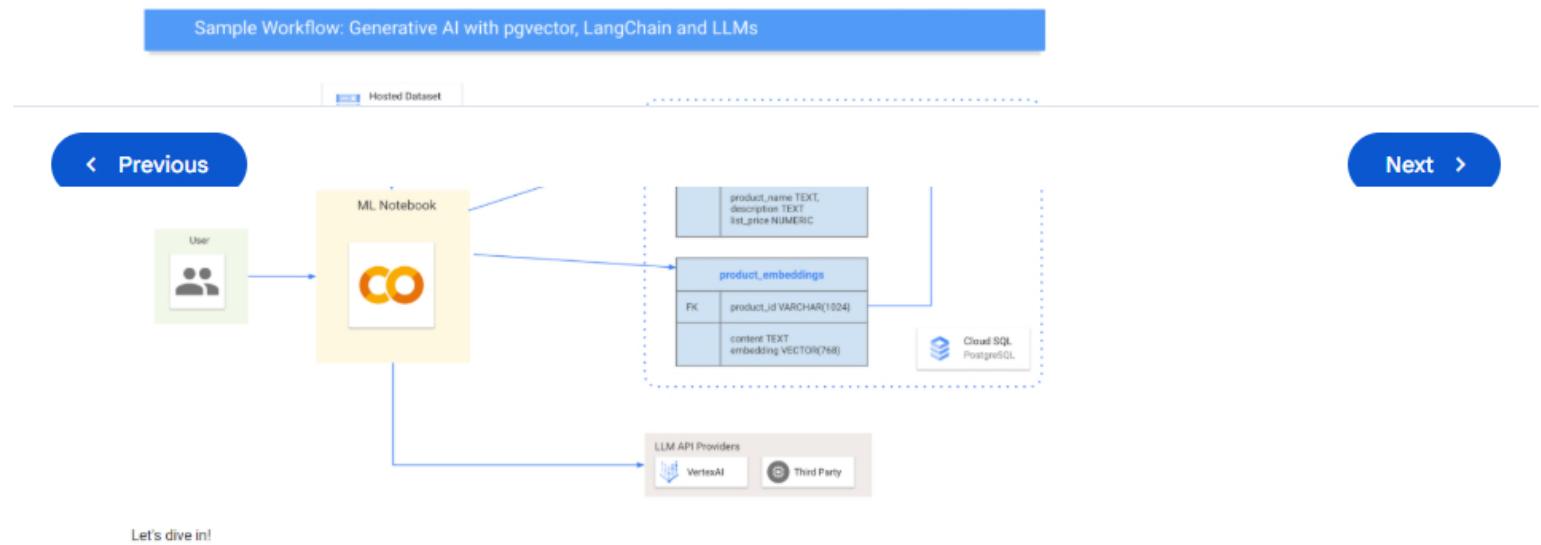
`list_price`.

2. Split the long `description` field values into smaller chunks and generate vector embeddings for each chunk. The vector embeddings are then stored in another PostgreSQL table called `product_embeddings` using the `pgvector` extension. The `product_embeddings` table has a foreign key referencing the `products` table.

3. For a given user query, generate its vector embeddings and use `pgvector` vector similarity search operators to find the closest matching products after applying the relevant SQL filters.

4. Once matching products and their descriptions are found, use the [MapReduceChain](#) from LangChain framework to generate a summarized high-quality context using an LLM model .

5. Finally, pass the context to an LLM prompt to answer the user query. The LLM model will return a well-formatted natural sounding English result back to the user.



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.

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.

< Previous

Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. [Learn more](#).

Username
google2727032_student@qwiklabs.net 

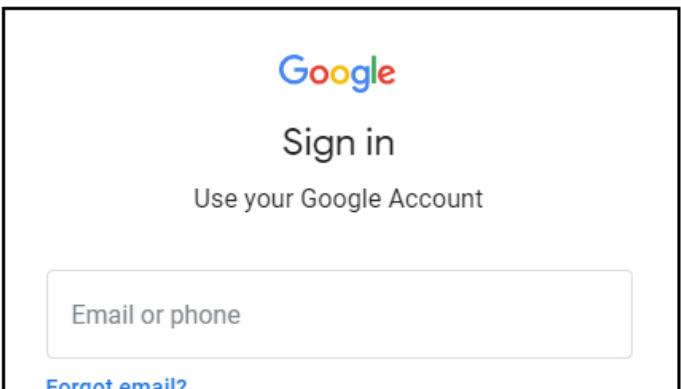
Password
k68CZXsxMZ 

GCP Project ID
qwiklabs-gcp-4fbfecac8667e457 

New to labs? [View our introductory video!](#)

Next >

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.



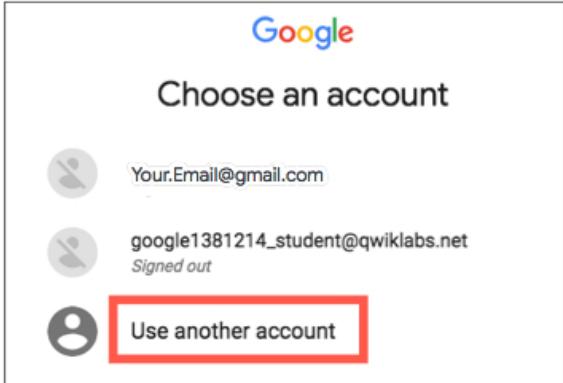
The image shows the Google Sign-in page. It features the Google logo at the top, followed by the "Sign in" button and the text "Use your Google Account". Below this is a large input field labeled "Email or phone". At the bottom of the input field, there is a link "Forgot email?".

< Previous

Next >

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

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



The image shows the "Choose an account" page. It displays two accounts: "Your.Email@gmail.com" and "google1381214_student@qwiklabs.net Signed out". Below the accounts is a button labeled "Use another account", which is highlighted with a red border.

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:

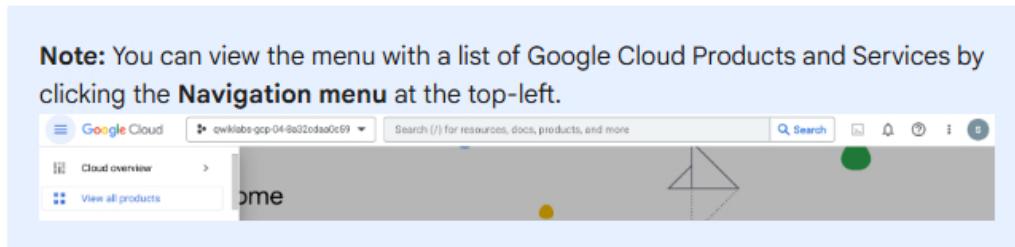
< Previous

Accept the terms and conditions.

Next >

- 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.



Task 1. Enable APIs

In this task, you need to enable the Dialogflow API.

< Previous

Next >

1. Type **Dialogflow API** into the top search bar of the Google Cloud console, choose the selected result as shown below, and continue.

A screenshot of a search results page for 'Dialogflow API'. The search bar at the top contains 'Dialogflow API'. Below the search bar is a 'DOCUMENTATION & TUTORIALS' section. It lists several items:

- Dialogflow API (Platform for BI, data applications, and embedded analytics...)
- Dialogflow Documentation (Workflow orchestration for serverless products and API...)
- APIs & references | Dialogflow ES (Platform for BI, data applications, and embedded analytics...)
- Quickstart: Interactions with the API | Dialogflow ES (Dialogflow responds with information about the matched...)
- API interactions | Dialogflow ES (When you use the API for interactions, your service interacts...)
- Dialogflow

- Dialogflow is a comprehensive platform for developing...
 - REST overview | Dialogflow ES
Workflow orchestration for serverless products and API...
 - APIs and reference | Dialogflow CX
Manage the full life cycle of APIs anywhere with visibility and...
 - Dialogflow API
It is used to build client libraries, IDE plugins, and other tools...
- [See more results for documentation and tutorials](#)

MARKETPLACE

DIALOGFLOW API

◀ Previous

Next ▶

-  Text AI Agent
Yosh.AI
-  Voice AI Agent
Yosh.AI
-  VoiceAI Connect
AudioCodes

2. Click the **Enable** button.

Task 2. Open Python notebook and install packages

1. In your Google Cloud project, navigate to **Vertex AI Workbench**. In the top search bar of the Google Cloud console, enter **Vertex AI Workbench**, and click on the first result.

◀ Previous

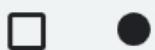
Next ▶

PRODUCTS & PAGES
 Workbench
Vertex AI

2. Click on **Instances** and then click on **Open JupyterLab**.

The JupyterLab will run in a new tab.

≡ Filter



Instance name ↑



[generative-ai-jupyterlab](#)

[Open JupyterLab](#)

3. On the **Launcher**, under **Notebook**, click on **Python 3** to open a new python notebook.

4. In the first cell, enter the following command to install the latest version of

< Previous

Next >

```
# Install dependencies.  
!pip install asyncio==3.4.3 asyncpg==0.27.0 cloud-sql-  
python-connector["asyncpg"]==1.2.3  
!pip install numpy==1.26.4 pandas  
!pip install pgvector==0.1.8  
!pip install langchain langchain_google_vertexai  
transformers  
!pip install -U langchain-google-vertexai langchain-  
community  
!pip install google-cloud-aiplatform  
!pip install shapely  
!pip install google-cloud-  
aiplatform[langchain,reasoningengine]  
!pip install langchain-text-splitters  
!pip install pydantic>=2.0
```

Output:

```
les>=0.2.1->google-auth<3.0dev,>=1.25.0->google-api-core[grpc]>=2.0.*,!=2.1.*,!=2.2.*,!=2.3.*,!=2.4.*,!=2.5.*,!=  
2.6.*,>=2.7.*,<3.0.0dev,>=1.32.0->google-cloud-aiplatform==1.26.0) (0.5.0)  
  Downloading google_cloud_aiplatform-1.26.0-py2.py3-none-any.whl (2.6 MB)  
    2.6/2.6 MB 25.7 MB/s eta 0:00:0000:0100:01  
Installing collected packages: google-cloud-aiplatform  
  Attempting uninstall: google-cloud-aiplatform  
    Found existing installation: google-cloud-aiplatform 1.36.0  
    Uninstalling google-cloud-aiplatform-1.36.0:  
      ...
```

< Previous

Next >

5. 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  at the top, followed by clicking **Restart** button.

```
# Automatically restart kernel after installation so that  
your environment can access the new packages.  
import IPython  
  
app = IPython.Application.instance()  
app.kernel.do_shutdown(True)
```

Output:

Kernel Restarting

The kernel for Untitled.ipynb appears to have died. It will restart automatically.



[Previous](#)

[Next >](#)

Click **OK** to move to the next step.

Task 3. Create a Cloud SQL Instance and PostgreSQL database.

1. Run the following code in the next cell, for declaring values prior to creating CloudSQL instance.

```
import os
import pandas as pd
import vertexai
from vertexai.language_models import TextEmbeddingModel
from vertexai.generative_models import GenerativeModel
from IPython.display import display, Markdown

from langchain_google_vertexai import VertexAIEMBEDDINGS
```

[Previous](#)

[Next >](#)

```
project_id = "quickstart-gcp-04-31-rev900e2"
database_password = "cymbal@123!"
region = "us-west1"
instance_name = "retail-ins"
database_name = "retail"
database_user = "retail-admin"

# Initialize Vertex AI SDK
vertexai.init(project=project_id, location=region)
```

2. Run the following code in the next notebook cell for creating a Cloud SQL instance and PostgreSQL database.

```
#@markdown Create and setup a Cloud SQL PostgreSQL
instance, if not done already.
cloudsql_instances = !cloud sql instances describe
```

```
!gcloud sql instances create {instance_name} --database-version=POSTGRES_15 \
--region={region} --cpu=1 --memory=4GB --root-password={root_password}
```

< Previous

Next >

```
# Create the database, if it does not exist.
out = !gcloud sql databases list --instance={instance_name}
--filter="NAME:{database_name}" --format="value(NAME)"
if ''.join(out) == database_name:
    print("Database %s already exists, skipping creation." % database_name)
else:
    !gcloud sql databases create {database_name} --instance={instance_name}

# Create the database user for accessing the database.
!gcloud sql users create {database_user} \
--instance={instance_name} \
--password={database_password}
```

Note: This code snippet may run for a few minutes.

Output:

```
Creating new Cloud SQL instance...
Creating Cloud SQL instance for POSTGRES_15...done.
Created [https://sqladmin.googleapis.com/sql/v1beta4/projects/qwiklabs-gcp-04-63807e05b641/instances/ins-001].
NAME  DATABASE_VERSION  LOCATION   TIER      PRIMARY_ADDRESS  PRIVATE_ADDRESS  STATUS
ins-001  POSTGRES_15    us-central1-b  db-custom-1-4096  34.42.33.53      -          RUNNABLE
Creating Cloud SQL database...done.
Created database [retail].
```

< Previous

Next >

Created user [retail-admin].

- Run the following code in the next notebook cell to verify that you are able to connect to the database.

```
# @markdown Verify that you are able to connect to the database. Executing this block should print the current PostgreSQL server version.
```

```
import asyncio
import asyncpg
from google.cloud.sql.connector import Connector

async def main():
    # get current running event loop to be used with
```

Connector

```
loop = asyncio.get_running_loop()
# initialize Connector object as async context manager
async with Connector(loop=loop) as connector:
    # create connection to Cloud SQL database
    conn: asyncpg.Connection = await
connector.connect_async(
    f"{project_id}:{region}:{instance_name}", #
    user=f'{database_user}',

    password=f'{database_password}',

    db=f'{database_name}'
    # ... additional database driver args
)

# query Cloud SQL database
results = await conn.fetch("SELECT version()")
print(results[0][ "version"])

# close asyncpg connection
await conn.close()

# Test connection with `asyncio`
await main() # type: ignore
```

< Previous

Next >

Output:

```
PostgreSQL 15.4 on x86_64-pc-linux-gnu, compiled by Debian clang version 12.0.1, 64-bit
```

< Previous

Next >

Task 4. Download and load the dataset in PostgreSQL

In this task, let's load the dataset from a web URL and store it in a pandas dataframe. Later save the Pandas dataframe in a PostgreSQL table.

1. Run the following code in the next notebook cell for loading the dataset from a web URL and storing it in a pandas dataframe.

```
# Load dataset from a web URL and store it in a pandas
dataframe.

import pandas as pd
import os
```



```
DATASET_URL =
"https://github.com/GoogleCloudPlatform/python-docs-
samples/raw/main/cloud-
sql/postgres/pgvector/data/retail_toy_dataset.csv"
df = pd.read_csv(DATASET_URL)
df = df.loc[:, ["product_id", "product_name",
"description", "list_price"]]
df
```

< Previous

Next >

Output:

	product_id	product_name	description	list_price
0	7e8697b5b7cdb5a40daf54caf1435cd5	Koplow Games Set of 2 D12 12-Sided Rock, Paper...	Rock, paper, scissors is a great way to resolv...	3.56
1	7de8b315b3cb91f3680eb5b88a20dce	12"-20" Schwinn Training Wheels	Turn any small bicycle into an instrument for ...	28.17
2	fb9535c103d7d717f0414b2b111cfaaa	Bicycle Pinochle Jumbo Index Playing Cards - 1...	Purchase includes 1 blue deck and 1 red deck. ...	6.49
3	c73ea622b3be6a3ffa3b0b5490e4929e	Step2 Woodland Adventure Playhouse & Slide	The Step2 Woodland Climber Adventure Playhouse...	499.99
4	dec7bd1f983887650715c6fafaa5b593	Step2 Naturally Playful Welcome Home Playhouse...	Children can play and explore in the Step2 Nat...	600.00
5	74a695e3675efc2aad11ed73c46db29b	Slip N Slide Triple Racer with Slide Boogies	Triple Racer Slip and Slide with Boogie Boards...	37.21
6	3ae5293b56c25f63b47cb8a89fb4813	Hydro Tools Digital Pool/Spa Thermometer	The solar-powered Swimline Floating Digital Th...	15.92
7	ed85bf829a36c67042503ffd9b6ab475	Full Bucket Swing With Coated Chain Toddler Sw...	Safe Kids&Children Full Bucket Swing With Coa...	102.26
8	55820fa53f0583cb637d5cb2b051d78c	Banzai Water Park Splash Zone	Dive into fun in your own backyard with the B...	397.82
9	0e26a9e92e4036bfaa60eb2040a8ec97	Polaris 39-310 5-Liter Zippered Super Bag for ...	Keep your pool water sparkling clean all seaso...	39.47

2. Save the pandas dataframe in a PostgreSQL table.

```
# Save the Pandas dataframe in a PostgreSQL table.
```

```
from google.cloud.sql.connector import Connector

async def main():
    loop = asyncio.get_running_loop()
    async with Connector(loop=loop) as connector:
        # Create connection to Cloud SQL database
        conn: asyncpg.Connection = await
connector.connect_async(
            f"{project_id}:{region}:{instance_name}", #
Cloud SQL instance connection name
            "asyncpg",
            user=f"{database_user}",
            password=f"{database_password}",
            db=f"{database_name}",
        )
        await conn.execute("DROP TABLE IF EXISTS products
```

< Previous

Next >

```
CASCADE")
    # Create the `products` table.
    await conn.execute(
        """CREATE TABLE products(
            product_id VARCHAR(1024)
PRIMARY KEY,
            product_name TEXT,
            description TEXT
```

< Previous

Next >

```
# Copy the dataframe to the `products` table.
tuples = list(df.itertuples(index=False))
await conn.copy_records_to_table(
    "products", records=tuples, columns=list(df),
timeout=10
)
await conn.close()

# Run the SQL commands now.
await main() # type: ignore
```

Task 5. Generate vector embeddings using a Text Embedding model

1. Split long product description text into smaller chunks

< Previous

Next >

The product descriptions can be much longer than what can fit into a single API request for generating the vector embedding.

- For example, the Vertex AI text embedding model accepts a maximum of 3,072 input tokens for a single API request.
- Use the `RecursiveCharacterTextSplitter` from the LangChain library to split the description into smaller chunks of 500 characters each.

Run the following code in the next notebook cell.

```
from langchain_text_splitters import
RecursiveCharacterTextSplitter
from langchain.schema import Document
```



```
from langchain.text_splitter import  
RecursiveCharacterTextSplitter  
  
text_splitter = RecursiveCharacterTextSplitter(  
    separators=[".", "\n"],  
    chunk_size=500,  
    chunk_overlap=0,  
    length_function=len
```

< Previous

Next >

```
max_documents = 47 # Reduced limit to further control API  
usage  
documents = []  
  
# Create Document objects with product_id as metadata  
for index, row in df.iterrows():  
    product_id = row["product_id"]  
    desc = row["description"]  
    documents.append(Document(page_content=desc, metadata={  
        "product_id": product_id}))  
  
# Use the text splitter on a subset of documents (e.g., 40-  
50)  
chunked = []  
docs =  
text_splitter.split_documents(documents[40:max_documents])  
  
# Collect split content along with product_id  
for doc in docs:  
    chunked.append({"product_id":  
        doc.metadata["product_id"], "content": doc.page_content})
```

2. Generate vector embedding for each chunk by calling an Embedding Generation service

< Previous

Next >

embeddings, which outputs a 768-dimensional vector for each chunk of text.

Run the following code in the next notebook cell.

```
from langchain_google_vertexai import VertexAIEMBEDDINGS  
from google.cloud import aiplatform  
import time  
  
embeddings_service = VertexAIEMBEDDINGS(model_name="text-  
embedding-005")  
  
# Helper function to retry failed API requests with  
exponential backoff.  
def retry_with_backoff(func, *args, retry_delay=10,  
backoff_factor=2.5, **kwargs): # Increased delay and  
backoff factor  
    max_attempts = 10
```

```

retries = 0
for i in range(max_attempts):
    try:
        return func(*args, **kwargs)
    except Exception as e:
        print(f"error: {e}")
        retries += 1

seconds... ,
time.sleep(wait)

# Reduced batch size for API calls to manage quota limits
batch_size = 3
for i in range(0, len(chunked), batch_size):
    request = [x["content"] for x in chunked[i : i + batch_size]]
    response =
retry_with_backoff(embeddings_service.embed_documents,
request)
    # Store the retrieved vector embeddings for each chunk
back.
    for x, e in zip(chunked[i : i + batch_size], response):
        x["embedding"] = e

# Store the generated embeddings in a pandas dataframe.
product_embeddings = pd.DataFrame(chunked)
product_embeddings.head()

```

< Previous

Next >

Note: The following code snippet may run for a few minutes.

Output:

< Previous

Next >

0	7e8697b5b7cdb5a40daf54caf1435cd5	resolv...		-0.022118045017123222, ...
1	7e8697b5b7cdb5a40daf54caf1435cd5	games, creating your own game, and friends and...		[-0.016362389549613, -0.052170153707265854, 0, ...]
2	7de8b315b3cb91f3680eb5b88a20dcee	Turn any small bicycle into an instrument for ...		[-0.029525158926844597, -0.023656629025936127, ...]
3	7de8b315b3cb91f3680eb5b88a20dcee	teel brackets stand up to heavy use. Customiza...		[0.0014654349070042372, -0.027971144765615463, ...]
4	7de8b315b3cb91f3680eb5b88a20dcee	es.com. Follow ride Schwinn on: Twitter, Faceb...		[-0.01432398147881031, -0.015937959775328636, ...]

Task 6. Use pgvector to store the generated embeddings within PostgreSQL

The `pgvector` extension introduces a new vector data type. The new vector data type allows you to directly save a vector embedding (represented as a NumPy array) through a simple `INSERT` statement in PostgreSQL!

1. Run the following code snippet to store the generated vector embeddings in

- PostgreSQL table

< Previous

Next >

```
# Store the generated vector embeddings in a PostgreSQL
# table.

# This code may run for a few minutes.

import asyncio
import asyncpg
from google.cloud.sql.connector import Connector
import numpy as np
from pgvector.asyncpg import register_vector

async def main():
    loop = asyncio.get_running_loop()
    async with Connector(loop=loop) as connector:
        # Create connection to Cloud SQL database.
        conn: asyncpg.Connection = await
connector.connect_async(
    f"{project_id}:{region}:{instance_name}", #
Cloud SQL instance connection name
    "asyncpg",
    user=f"{database_user}",
    password=f"{database_password}",
    db=f"{database_name}",
)

```



< Previous

Next >

```
await register_vector(conn)

        await conn.execute("DROP TABLE IF EXISTS
product_embeddings")
        # Create the `product_embeddings` table to store
vector embeddings.
        await conn.execute(
            """CREATE TABLE product_embeddings(
                product_id VARCHAR(1024)
NOT NULL REFERENCES products(product_id),
                content TEXT,
                embedding vector(768))"""
)
        # Store all the generated embeddings back into the
database.
        for index, row in product_embeddings.iterrows():
            await conn.execute(
                "INSERT INTO product_embeddings

```

```
(product_id, content, embedding) VALUES ($1, $2, $3)",  
    row["product_id"],  
    row["content"],  
    np.array(row["embedding"])),  
)  
  
await conn.close()
```

< Previous

Run this SQL command now
await main() # type: ignore

Next >

Task 7. Finding similar toy products using pgvector cosine search operator

1. Run the following code in the next notebook cell to search for similar toy products within a specified price range.

```
# @markdown Enter a short description of the toy to search  
for within a specified price range:  
toy = "playing card games" # @param {type:"string"}  
min_price = 25 # @param {type:"integer"}  
max_price = 100 # @param {type:"integer"}  
  
# Quick input validations.
```

< Previous

```
from langchain_google_vertexai import vertexaiitemembeddings  
from google.cloud import aiplatform  
  
aiplatform.init(project=f"{project_id}", location=f"  
{region}")  
  
embeddings_service = VertexAIEmbeddings(model_name="text-  
embedding-005")  
qe = embeddings_service.embed_query(toy)  
from pgvector.asyncpg import register_vector  
import asyncio  
import asyncpg  
from google.cloud.sql.connector import Connector  
  
matches = []  
  
async def main():  
    loop = asyncio.get_running_loop()
```

Next >

```
    async with Connector(loop=loop) as connector:
        # Create connection to Cloud SQL database.
        conn: asyncpg.Connection = await
connector.connect_async(
    f"{project_id}:{region}:{instance_name}", #
Cloud SQL instance connection name
    "asyncpg",
)
await register_vector(conn)
similarity_threshold = 0.1
num_matches = 50

# Find similar products to the query using the
cosine similarity search
# Over all vector embeddings. This new feature is
provided by `pgvector`.
results = await conn.fetch(
    """
        WITH vector_matches AS (
            SELECT product_id, 1 -
(embedding <=> $1) AS similarity
            FROM product_embeddings
            WHERE 1 - (embedding <=> $1) >
$2
            ORDER BY similarity DESC
            LIMIT $3
        )
        SELECT product_name,
list_price, description FROM products
            WHERE product_id IN (SELECT
product_id FROM vector_matches)
    """
)
```

< Previous

Next >

```
    qe,
    similarity_threshold,
    num_matches,
    min_price,
    max_price,
)

if len(results) == 0:
    raise Exception("Did not find any results.
Adjust the query parameters.")

for r in results:
    # Collect the description for all the matched
similar toy products.
    matches.append(
        {
            "product_name": r["product_name"],
            "description": r["description"],
            "list_price": r["list_price"]
        }
    )

```

< Previous

Next >

```
    "list_price": round(r["list_price"]),
    2),
)
)

await conn.close()
```

< Previous

Next >

```
# Show the results for similar products that matched the
# user query.
matches = pd.DataFrame(matches)
matches.head(5)
```

Output

	product_name	description	list_price
0	Purple Mini Poker Chips Plastic 7/8in Bulk App...	Purple Mini Poker Chips Plastic 7/8in Bulk App...	27.78
1	Arkham Horror Living Card Game: The Dream-Eat...	There is a hidden realm outside the world of ...	25.82
2	Zelda Uno Card Game Special Legend Rule Exclus...	Bonus Zelda Rules Exclusive Card Game Zelda Un...	29.56

Task 8. LLMs and LangChain

We have extracted the semantic knowledge of the dataset and made it searchable through pgvector and PostgreSQL. This task further shows how you can use this

< Previous

Next >

Building an AI-curated contextual hybrid search

Combine natural language query text with regular relational filters to create a powerful hybrid search.

Example: A grandparent wants to use the **AI-powered search interface** to find an educational toy for their grandkid that fits within their budget.

1. Generate the vector embedding for the user query

```
# @markdown Enter the user search query in a simple English
text. The price filters are shown separately here for demo
purposes. These filters may represent additional input from
your frontend application.
# Please fill in these values
```



```
# Please fill in these values.

user_query = "Do you have a beach toy set that teaches
numbers and letters to kids?" # @param {type:"string"}
min_price = 20 # @param {type:"integer"}
max_price = 100 # @param {type:"integer"}

# Quick input validations.
```

< Previous

Next >

```
qe = embeddings_service.embed_query(user_query)
```



2. Use pgvector to find similar products.

- The new `pgvector` similarity search operators provide powerful semantics to combine the vector search operation with regular query filters in a single SQL query.
- Using `pgvector`, you can now seamlessly integrate the power of relational databases with your vector search operations!

```
from pgvector.asyncpg import register_vector
import asyncio
import asyncpg
from google.cloud.sql.connector import Connector

matches = []

async def main():
```



< Previous

Next >

```
# Create connection to Cloud SQL database.
conn: asyncpg.Connection = await
connector.connect_async(
    f"{project_id}:{region}:{instance_name}", #
Cloud SQL instance connection name
    "asyncpg",
    user=f"{database_user}",
    password=f"{database_password}",
    db=f"{database_name}",
)

await register_vector(conn)
similarity_threshold = 0.5
num_matches = 10

# Find similar products to the query using cosine
similarity search
# over all vector embeddings. This new feature is
provided by `pgvector`.
results = await conn.fetch()
```

```

    results = await conn.execute(
        """
        WITH vector_matches AS (
            SELECT product_id, 1 -
        (embedding <=> $1) AS similarity
            FROM product_embeddings
            WHERE 1 - (embedding <=> $1) >
        """
    )

```

< Previous

Next >

```

        LIMIT $3
    )
    SELECT product_name,
list_price, description FROM products
    WHERE product_id IN (SELECT
product_id FROM vector_matches)
        AND list_price >= $4 AND
list_price <= $5
        """
        ,
qe,
similarity_threshold,
num_matches,
min_price,
max_price,
)
if len(results) == 0:
    raise Exception("Did not find any results.
Adjust the query parameters.")

for r in results:
    # Collect the description for all the matched
similar toy products.
    matches.append(
        f"""The name of the toy is
{r["product_name"]}.
        The price of the toy is

```

< Previous

Next >

```

        r["description"]).
    )
await conn.close()

# Run the SQL commands now.
await main() # type: ignore

# Show the results for similar products that matched the
user query.
matches

```

Output:

```

[('The name of the toy is Champion Sports Tournament Series Horseshoe Set.\nIts description is below:\n\nThe Champion Sports Deluxe Horseshoe Tournament Set features everything you need to set up a horseshoe tournament, including a set of rules to ensure games stay on track. The tough steel horseshoes are plated with chrome and brass for an attractive shine and extra durability, and the set includes two rugged chrome-plated steel stakes to take aim at. The horseshoe set comes with a carrying bag made from strong 1680 denier nylon fabric that can endure toting the sturdy heavy metal pieces around with ease. Two bronze powder coated steel horseshoes Two silver powder coated steel horseshoes Two silver powder coated solid steel stakes One durable weather-resistant 1680 Denier nylon carry bag with YKK zipper and embroidered logo. Two bronze powder coated steel horseshoes Two silver powder coated steel horseshoes One silver powder coated solid steel stakes One durable weather-resistant 1680 Denier nylon carry bag with YKK zipper and embroidered logo.', 'The price of the toy is $61.96.\n'),
 ('The name of the toy is KARMA'S PRODUCT Heavy Duty Glider Swing for Kids Fun Swings Seat.\nIts description is below:\n\nKARMA'S PRODUCT Heavy Duty Glider Swing for Kids Fun Swings Seat; Heavy Duty Glider Swings,play more fun! Use highly durable plastic ,very strong. Features footrests and handholds for added comfort and security Push with your feet; pull with your hands to get

```

This glider swing, riding waves in ocean safety, measures 44 inches long x 36 inches wide x 37 inches high. Weight is 17 pounds. It has a strong, features footrests and handholds for added comfort and security. Push with your feet; pull with your hands to get this glider swing flying! Meets US AB TM safety standards. Suitable for children 3 years up. Max weight is 55 pounds (25KG).
 The name of the toy is 11.5" Playing Cards Plastic Dealer Shoe, Clear & Black, Holds Up to 6 Decks by GHL.
 Its description is below:
 All of our products comply with international quality standards and are proudly appreciated in a variety of different markets throughout the world. 3700+ Features. Material: plastic. Holds up to 6 decks of card s. Game: -Poker, Accessory type: -Card Dispensers, keep Inval: -Yes, Dimensions: Overall Product Weight: ~2.5 Pounds. If you are interested in any of our products or would like to discuss a custom order, please feel free to contact us. Holds Up to 6 Decks Fun for the whole family! Smooth Rolling Action Heavy Plastic!
 The name of the toy is Melissa & Doug Jumbo ABC-123 Rug (58 x 79 inches - 36 Game Cards).
 Its description is below:
 Kids will have loads amounts of fun exploring and learning the alphabet, numbers, colors, and shapes in the Jumbo ABC-123 Rug. Theres more than 30 square feet of play space on the durable and soft rug to play matching games with the set of 36 double-sided game cards. There are whimsical pictures with each letter of the alphabet and numbers one through nine, and the border includes 12 shapes and shape names. Made from durable material that is incredibly soft to the touch for extra comfortable play, this multi-activity rug features a skid-proof backing and reinforced border binding. The rug measures approximately 58 inches x 79 inches. Melissa & Doug Jumbo ABC-123 Rug (58 x 79 inches - 36 Game Cards): Oversized activity rug with illustrated alphabet, numbers 1-9, shapes, and colors. Includes 36 double-sided game cards. Lots of room for multiple kids to play together. Soft, durable material, skid-resistant backing, and reinforced border binding. Area: 4.30' x 6.7' (58 x 79 in./147 x 200 cm).Oversized activit

< Previous

Next >

3. Use LangChain to summarize and generate a high-quality prompt to answer the user query

- After finding the similar products and their descriptions using `pgvector`, the next step is to use them for generating a prompt input for the LLM model.
- Since individual product descriptions can be very long, they may not fit within the specified input payload limit for an LLM model.
- The `MapReduceChain` from LangChain framework is used to generate and combine short summaries of similarly matched products.
- The combined summaries are then used to build a high-quality prompt for an input to the LLM model.

```
# Using LangChain for summarization and efficient context
building.
```

```
from langchain.chains.summarize import load_summarize_chain
from langchain.docstore.document import Document
from langchain_google_vertexai import VertexAI
```

< Previous

Next >

```
llm = VertexAI(temperature=0.7)

map_prompt_template = """
    You will be given a detailed description of a
toy product.
    This description is enclosed in triple
backticks (``').
    Using this description only, extract the name
of the toy,
    the price of the toy and its features.

    ``{text}``
SUMMARY:
````

map_prompt = PromptTemplate(template=map_prompt_template,
input_variables=["text"])

combine_prompt_template = """
 You will be given a detailed description
```

different toy products  
enclosed in triple backticks (```) and a  
question enclosed in  
double backticks(``).  
Select one toy that is most relevant to  
answer the question.

< Previous

Next >

question in as much detail as possible.  
You should only use the information in the  
description.  
Your answer should include the name of the  
toy, the price of the toy  
and its features. Your answer should be  
less than 200 words.  
Your answer should be in Markdown in a  
numbered list format.

Description:

```{text}```

Question:

``{user_query}``

Answer:

```

```
combine_prompt = PromptTemplate(
 template=combine_prompt_template, input_variables=
 ["text", "user_query"]
)
```

< Previous

Next >

```
chain_type=map_reduce, map_prompt=map_prompt,
combine_prompt=combine_prompt
)
output = chain.invoke(
 {
 "input_documents": docs,
 "user_query": user_query,
 }
)

#display(Markdown(answer))
Extract and display the output
answer = output.get('output_text', '')
display(Markdown(answer))
```

Output:

2. The rug is priced at \$54.99.
3. It comes with 36 double-sided game cards and is made of soft, durable material with a skid-proof backing and reinforced border binding.
4. It is suitable for ages 3 and up.

## Adding AI-powered creative content generation

[Previous](#)[Next](#)

from an initial prompt.

**Example:** A third-party seller on the retail platform wants to use the **AI-powered content generation** to create a detailed description of their new bicycle product.

```
@markdown Describe your a new product in just a few words:
Please fill in these values.
creative_prompt = "A bicycle with brand name 'Roadstar bike' for
kids that comes with training wheels and helmet." # @param
{type:"string"}

Quick input validations.
assert creative_prompt, "⚠️ Please input a valid input search
text"
```



1. Find an existing product description matching the initial prompt.

- Leverage the `pgvector` similarity search operator to find an existing product description that closely matches the new product specified in the initial prompt.

[Previous](#)[Next](#)

```
import asyncpg
from google.cloud.sql.connector import Connector

qe = embeddings_service.embed_query(creative_prompt)
qe_str = "[%s]" % (", ".join([str(x) for x in qe]))
matches = []

async def main():
 loop = asyncio.get_running_loop()
 async with Connector(loop=loop) as connector:
 # Create connection to Cloud SQL database.
 conn: asyncpg.Connection = await
 connector.connect_async(
 f"{project_id}:{region}:{instance_name}", #
 Cloud SQL instance connection name
 "asyncpg",
 user=f"{database_user}",
 password=f"{database_password}",
 database=database_name)
```

```
 database_name,
)

 await register_vector(conn)
 similarity_threshold = 0.5

 # Find similar products to the query using cosine
```

< Previous

Next >

```
provided by `pgvector`.

results = await conn.fetch(
 """
 WITH vector_matches AS (
 SELECT product_id, 1 -
 (embedding <=> $1) AS similarity
 FROM product_embeddings
 WHERE 1 - (embedding <=> $2) >
 $3
 ORDER BY similarity DESC
 LIMIT 1
)
 SELECT description FROM
 products
 WHERE product_id IN (SELECT
 product_id FROM vector_matches)
 """
 ,
 qe,
 qe,
 similarity_threshold,
)

for r in results:
 matches.append(r["description"])

await conn.close()
```

< Previous

Next >

```
Run the SQL commands now.

await main() # type: ignore

Show the matched product description.
matches
```

## Output:

This bike is all about turning little kids into cool little riders. With a durable steel frame, smooth rolling tires, long seat post and adjustable height handlebars, its the perfect balance bike for kids from 18 months to 5 years old. Durability meets style! Frame with silver coated tires (12" wheel) with lightweight alloy rim standard tires with inner tubes (all) bearings threaded headset (with 130 steering limit). Adjustable saddle and handlebar height 200-300g. Kids Balance/Bike Bicycle.]

2. Use the existing matched product description as the prompt context to generate new creative output from the LLM.

```
from IPython.display import display, Markdown
from langchain_google_vertexai import VertexAI
from langchain_core.prompts import PromptTemplate
from langchain_core.runnables import RunnableSequence
```



```
template = """
 You are given descriptions about some similar
kind of toys in the context.

 This context is enclosed in triple backticks
(```).
```

< Previous

Next >

```
the initial prompt. All the information from
the initial prompt must
be included. You are allowed to be as creative
as possible,
and describe the new toy in as much detail.
Your answer should be
less than 200 words.
```

Context:

```
```{context}```
```

Initial Prompt:

```
{creative_prompt}
```

Answer:

```
"""
```

```
prompt = PromptTemplate(
    template=template, input_variables=["context",
"creative_prompt"]
)
```

```
# Define the LLM
llm = VertexAI(temperature=0.7)
# Example `matches` list
```

< Previous

Next >

```
{"description": "This is a toy description 2."},
{}, # Missing `description`
"Invalid item" # Not a dictionary
]

# Construct the context by extracting valid descriptions
context = "\n".join(
    item["description"] for item in matches if
isinstance(item, dict) and "description" in item
)

# Use RunnableSequence for chaining
llm_chain = RunnableSequence(prompt | llm)
# Invoke the chain
answer = llm_chain.invoke({
    "context": context,
    "creative_prompt": creative_prompt,
})

# Display the answer in Markdown format
```

```
display(Markdown(answer))
```

Output:

The Roadstar Bike is the perfect first bicycle for young adventurers! Designed specifically for kids, this vibrant and sturdy bike features a durable frame and comes equipped with comfortable, adjustable handlebars and a front basket for storage. It's perfect for learning balance and coordination. Safety features include a reflective skirt and a bell. The Roadstar Bike includes a sturdy child seat and a rear rack for carrying gear.

Congratulations

Generative AI is a powerful paradigm shift in application development that lets you create novel applications to serve users in new ways - [from answering patients' complex medical questions](#) to [helping enterprises analyze cyberattacks](#). In this lab, we showed you just two examples of powerful features that you can create by combining LLMs and databases.

Manual Last Updated April 22, 2025

Lab Last Tested April 22, 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.

< Previous

Next >