Step–1 :- Install the following libraries using pip install.

```
pymilvus==2.0.1
flask-cors
numpy
flask
flask_restful
psycopg2-binary
sentence_transformers
```

Step-2 :- Download the docker-compose(wget https://raw.githubusercontent.com/milvus-io/milvus/master/deployments/docker/standalone/docker-compose.yml -O docker-compose.yml) and run command sudo docker-compose up -d.

```
--2022-08-18 12:53:53--  https://raw.githubusercontent.com/milvus-io/milvus/master/deployments/docker/standalone/docker-compose.yml
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.110.133, 185.199.111.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1323 (1.3K) [text/plain]
Saving to: ‘docker-compose.yml’

docker-compose.yml  100%[===================>]   1.29K  --.-KB/s    in 0s

2022-08-18 12:53:53 (45.2 MB/s) - ‘docker-compose.yml’ saved [1323/1323]

Creating network "milvus" with the default driver
Creating milvus-minio ...
Creating milvus-etcd  ...
Creating milvus-standalone ...
ting milvus-standalone ... done
```

Step-3 :- Start the Postgres Server by command(!docker run --name postgres0 -d -p 5438:5432 -e POSTGRES_HOST_AUTH_METHOD=trust postgres)

```
52a2815c315ae86bb40127458826a52cff95af81898da530d3e8067cc35bfe69
```

Step-4 :- See the docker logs.

```
! docker logs postgres0 --tail 6
✓ 0.3s                                                                                                    Python
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling parallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
2022-08-18 10:52:16.730 UTC [1] LOG:  starting PostgreSQL 14.5 (Debian 14.5-1.pgdg110+1) on x86_64-pc-linux-gnu, compiled by gcc (Debian 10.2.1-6) 10.2.1 20210110, 64-bit
2022-08-18 10:52:16.730 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2022-08-18 10:52:16.731 UTC [1] LOG:  listening on IPv6 address "::", port 5432
2022-08-18 10:52:16.799 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2022-08-18 10:52:16.867 UTC [61] LOG:  database system was shut down at 2022-08-18 10:52:16 UTC
2022-08-18 10:52:16.909 UTC [1] LOG:  database system is ready to accept connections
```

Step-5 :- Connect to Docker and postgres server.

```
from pymilvus import connections
import psycopg2
connections.connect(host='localhost', port='19530')
conn = psycopg2.connect(host='localhost', port='5438', user='postgres', password='postgres')
cursor = conn.cursor()
```

✓ 0.8s

Step-6 :- Create the collection by giving the name of the collection and dimension of the vector.

```
TABLE_NAME = "text_collection"
field_name = "example_field"
from pymilvus import Collection, CollectionSchema, FieldSchema, DataType
pk = FieldSchema(name="id", dtype=DataType.INT64, is_primary=True, auto_id=True)
field = FieldSchema(name=field_name, dtype=DataType.FLOAT_VECTOR, dim=768)
schema = CollectionSchema(fields=[pk,field], description="example collection")
collection = Collection(name=TABLE_NAME, schema=schema)
```

✓ 0.2s

Step-7 :- Set the index. This can be done before or after inserting the data. If done before, indexes will be made as data comes in and fills the data segments.

```
index_param = {
        "metric_type":"L2",
        "index_type":"IVF_SQ8",
        "params":{"nlist":1024}
    }
collection.create_index(field_name=field_name, index_params=index_param)
```

✓ 2.1s

```
Status(code=0, message='')
```

Step-8 :- Create the new table in PostgresSQL.

```
#Deleting previouslny stored table for clean run
drop_table = "DROP TABLE IF EXISTS " + TABLE_NAME
cursor.execute(drop_table)
conn.commit()

try:
    sql = "CREATE TABLE if not exists " + TABLE_NAME + " (ids bigint, title text, text text);"
    cursor.execute(sql)
    conn.commit()
    print("create postgres table successfully!")
except Exception as e:
    print("can't create a postgres table: ", e)
```

✓ 0.4s

```
create postgres table successfully!
```

s

Step-9 :- Generate the embeddings, in this we are using the sentence_transformer library to encode the sentence into vectors. This library uses the modified BERT model to generate the embeddings.

```
from sentence_transformers import SentenceTransformer
import pandas as pd
from sklearn.preprocessing import normalize

model = SentenceTransformer('paraphrase-mpnet-base-v2')
# Get questions and answers.
data = pd.read_csv('data/example.csv')
title_data = data['title'].tolist()
text_data = data['text'].tolist()

sentence_embeddings = model.encode(title_data)
sentence_embeddings = normalize(sentence_embeddings)
print(type(sentence_embeddings))
```

✓ 2.2s

```
<class 'numpy.ndarray'>
```

Step-10 :- Insert the vectors into Milvus.

```
em =list(sentence_embeddings)
mr = collection.insert([em])
ids = mr.primary_keys
dicts ={}
```

✓ 0.2s

Step-11 :- Insert the data(ID, Title and Text) into PostgreSQL.

```
import os

def record_temp_csv(fname, ids, title, text):
    with open(fname,'w') as f:
        for i in range(len(ids)):
            line = str(ids[i]) + "|" + title[i] + "|" + text[i] + "\n"
            f.write(line)

def copy_data_to_pg(table_name, fname, conn, cur):
    fname = os.path.join(os.getcwd(),fname)
    try:
        sql = "COPY " + table_name + " FROM STDIN DELIMITER '|' CSV HEADER"
        cursor.copy_expert(sql, open(fname, "r"))
        conn.commit()
        print("Inserted into Postgress Sucessfully!")
    except Exception as e:
        print("Copy Data into Postgress failed: ", e)

DATA_WITH_IDS = 'data/test.csv'

record_temp_csv(DATA_WITH_IDS, ids, title_data, text_data)
copy_data_to_pg(TABLE_NAME, DATA_WITH_IDS, conn, cursor)
```

✓ 0.1s

```
Inserted into Postgress Sucessfully!
```

Step-12 :- Now, we will generate the embedding we want to search and then will search for the similiar Embeddings in the Milvus.

```
search_params = {"metric_type": "L2", "params": {"nprobe": 10}}

query_vec = []

title = "Storm"

query_embeddings = []
embed = model.encode(title)
embed = embed.reshape(1,-1)
embed = normalize(embed)
query_embeddings = embed.tolist()

collection.load()
results = collection.search(query_embeddings, field_name, param=search_params, limit=25, expr=None)
```

✓ 0.7s

Step-13 :- Get the most closest Titles.

```
similar_titles = []

for result in results[0]:
    sql = "select title from " + TABLE_NAME + " where ids = " + str(result.id) + ";"
    cursor.execute(sql)
    rows=cursor.fetchall()
    if len(rows):
        similar_titles.append((rows[0][0], result.distance))
        print((rows[0][0], result.distance))
```

✓ 0.7s

```
('Politics an Afterthought Amid Hurricane  ', 1.1896356344223022)
('Hurricane Survivors Wait for Water, Gas', 1.4036846160888672)
('Explosions Echo Throughout Najaf', 1.444091558456421)
('News: Sluggish movement on power grid cyber security', 1.4572890996932983)
('Prediction Unit Helps Forecast Wildfires  ', 1.466585397720337)
('St. Louis Cardinals News', 1.520593523979187)
```

Step-14 :- We have prepared our Dataset.

milvus.ipynb | milvus_new.ipynb | text_search_engine.ipynb M | test.csv M | example.csv M

bootcamp > solutions > text_search_engine > data > test.csv

```
1  435415679799132173|Linux,Agile,C++,Shell Scripting,SQL,Scrum,GitLab,JIRA,Salesforce,Confluence,SalesForce,JAVA|
2  435415679799132174|HTML/CSS,MYSQL,Outsystems,Javascript|
3  435415679799132175|Python,R,Power BI,Data Analysis,Tableau,DialogFlow,Machine Learning,Deep Learning,Quantitative Analysis,Natural
4  435415679799132176|Javascript,NodeJs,ReactJS,Angular,AngularJs,React Native|
5  435415679799132177|NodeJS,React-Native,ReactJS|
6  435415679799132178|Lead Generation,Detailed Prospecting,Sales Pitch Development,Customer Relationship Management,Team management,Ti
7  435415679799132179|Python,Odoo Framework,JavaScript,HTML,MVC,CSS,Bootstarp,PostgreSQL,XML,GIT|
8  435415679799132180|React.js,vue.js,HTML/CSS,Bootstrap,Material UI,SCSS,Jquery,axios,ajax,flux,react hooks|
9  435415679799132181|React JS,Redux,HTML 5,CSS 3,Javascript|
10 435415679799132182|Blockchain,Ethereum,Java,Git,Rust,Golang,Substrate,Python,Stellar,Solidity,NodeJS,Postgres,React.js,Solana,Types
11 435415679799132183|Next Js with Type script,Redux Toolkit,Tailwind CSS,Electron Js,Style Components|
12 435415679799132184|JavaScript,HTML/CSS,Python,React js,Node js,MERN stack,Visual Studio|
13 435415679799132185|C,C++,Python,Metasploit,Nmap,Html5|
14 435415679799132186|HTML5,CSS3,JavaScript,React.JS,Redux|
15
```

Step-15 :- Now, get the similar skills. We have searched skills :-Python,odoo,C Language,Arduino,Html,CSS,Javascript

```
search_params = {"metric_type": "L2", "params": {"nprobe": 10}}

query_vec = []

title = "Python,odoo,C Language,Arduino,Html,CSS,Javascript"

query_embeddings = []
embed = model.encode(title)
embed = embed.reshape(1,-1)
embed = normalize(embed)
query_embeddings = embed.tolist()

collection.load()
results = collection.search(query_embeddings, field_name, param=search_params, limit=10, expr=None)
```

✓ 0.8s

Step-16 :- Get the closest titles.

```python
similar_titles = []

for result in results[0]:
    sql = "select title from " + TABLE_NAME + " where ids = " + str(result.id) + ";"
    cursor.execute(sql)
    rows=cursor.fetchall()
    if len(rows):
        similar_titles.append((rows[0][0], result.distance))
        print((rows[0][0], result.distance))
```

0.3s Python

```
('Python,Odoo Framework,JavaScript,HTML,MVC,CSS,Bootstarp,PostgreSQL,XML,GIT', 0.416193425655365)
('JavaScript,HTML/CSS,Python,React js,Node js,MERN stack,Visual Studio', 0.7450960874557495)
('React.js,vue.js,HTML/CSS,Bootstrap,Material UI,SCSS,Jquery,axios,ajax,flux,react hooks', 0.9022969007492065)
('C,C++,Python,Metasploit,Nmap,Html5', 0.9163162708282471)
('Blockchain,Ethereum,Java,Git,Rust,Golang,Substrate,Python,Stellar,Solidity,NodeJS,Postgres,React.js,Solana,Typescript,Redis,Next.js,Snowflakes,Nes
0.9717453718185425)
```