Labs Data & AI Innovation Day

Lab 4: Build a Python chatbot

This document describes how to build a Python chatbot to query the Azure Cosmos DB for NoSQL in natural language

# Pre-requisites

Ensure that you have the following software installed on your system before proceeding with the lab:

* Visual Studio Code: A cross-platform code editor that supports Python development. You can download it from <https://code.visualstudio.com/>
* Python 3.10.11: The latest version of the Python programming language. You can download it from <https://www.python.org/downloads/release/python-31011/>

Note: If you are using a different version of Python, make sure that it is compatible with the libraries and packages used in this lab.

* Azure OpenAI account registered in the Azure subscription used for this lab
* An existing Python virtual environment, as described in Lab1

# Create a simple chatbot

In this exercice, we will create a chatbot that can answer simple questions

* Create a “Lab4” folder in the “Labs” foler
* Open Visual Studio Code
* In the “lab4” folder, create a file call simple\_chatbot.py
* Copy/paste the import statements

import streamlit as st

import os, time

from openai import AzureOpenAI

from dotenv import load\_dotenv

* Copy this code to load the content of the .env file

def init\_env():

    st.set\_page\_config(page\_title="CosmicWorks Chatbot", page\_icon="🛒")

    st.title("🛒 CosmicWorks Chatbot")

    load\_dotenv("..\.env")

    os.environ["OPENAI\_API\_TYPE"] = "azure"

    os.environ["OPENAI\_API\_VERSION"] = os.getenv("AZURE\_OPENAI\_API\_VERSION")

    os.environ["AZURE\_OPENAI\_ENDPOINT"] = os.getenv("AZURE\_OPENAI\_ENDPOINT")

    os.environ["AZURE\_OPENAI\_API\_KEY"] = os.getenv("AZURE\_OPENAI\_API\_KEY")

    os.environ["AZURE\_OPENAI\_EMBEDDING\_MODEL"] = os.getenv("AZURE\_OPENAI\_EMBEDDING\_MODEL")

if \_\_name\_\_ == "\_\_main\_\_":

    init\_env()

Save the app.py file

* Open a command prompt
* Navigate to the “Lab4” folder
* Activate the virtual environment with ”../.venv/scripts/activate”
* Run the application with streamlit run simple\_chatbot.py

A black text on a white background

Description automatically generated

Let’s now add code to our application to run simple queries

Add a new function called simple\_question()

def simple\_question():

    """ Ask a simple question """

    openai\_client = AzureOpenAI(

        api\_key = os.getenv("AZURE\_OPENAI\_API\_KEY"),

        api\_version = os.getenv("AZURE\_OPENAI\_API\_VERSION"),

        azure\_endpoint =os.getenv("AZURE\_OPENAI\_ENDPOINT")

    )

    question = st.text\_input("How can I help you?")

    if st.button("Submit"):

        user\_prompt = f"""

You are a chatbot, having a friendly conversation with a human

- Answer in markdown format

USER QUESTION:

{question}

ANSWER:

"""

        response, completion\_time = get\_completion(

            openai\_client,

            os.getenv("AZURE\_OPENAI\_CHAT\_MODEL"),

            user\_prompt)

        response, completion\_time

Modify the application entry point:

if \_\_name\_\_ == "\_\_main\_\_":

    init\_env()

    simple\_question()

Run the application again with streamlit run simple\_chatbot.py

In the textbox, enter “What is the capital of France?” and submit the query

You should get this answer:

A screenshot of a computer

Description automatically generated

Complete code:

import streamlit as st

import os, time

from openai import AzureOpenAI

from dotenv import load\_dotenv

def init\_env():

    st.set\_page\_config(page\_title="CosmicWorks Chatbot", page\_icon="🛒")

    st.title("🛒 CosmicWorks Chatbot")

    load\_dotenv("..\.env")

    os.environ["OPENAI\_API\_TYPE"] = "azure"

    os.environ["OPENAI\_API\_VERSION"] = os.getenv("AZURE\_OPENAI\_API\_VERSION")

    os.environ["AZURE\_OPENAI\_ENDPOINT"] = os.getenv("AZURE\_OPENAI\_ENDPOINT")

    os.environ["AZURE\_OPENAI\_API\_KEY"] = os.getenv("AZURE\_OPENAI\_API\_KEY")

    os.environ["AZURE\_OPENAI\_EMBEDDING\_MODEL"] = os.getenv("AZURE\_OPENAI\_EMBEDDING\_MODEL")

def get\_completion(openai\_client, model, prompt: str):

    start\_time = time.time()

    response = openai\_client.chat.completions.create(

        model = model,

        messages = [{"role": "user", "content": prompt}]

    )

    end\_time = time.time()

    elapsed\_time = end\_time - start\_time

    return response.choices[0].message.content, elapsed\_time

def simple\_question():

    """ Ask a simple question """

    openai\_client = AzureOpenAI(

        api\_key = os.getenv("AZURE\_OPENAI\_API\_KEY"),

        api\_version = os.getenv("AZURE\_OPENAI\_API\_VERSION"),

        azure\_endpoint =os.getenv("AZURE\_OPENAI\_ENDPOINT")

    )

    question = st.text\_input("How can I help you?")

    if st.button("Submit"):

        user\_prompt = f"""

You are a chatbot, having a friendly conversation with a human

- Answer in markdown format

USER QUESTION:

{question}

ANSWER:

"""

        response, completion\_time = get\_completion(

            openai\_client,

            os.getenv("AZURE\_OPENAI\_CHAT\_MODEL"),

            user\_prompt)

        response, completion\_time

if \_\_name\_\_ == "\_\_main\_\_":

    init\_env()

    simple\_question()

# Ground the chatbot on your data (RAG)

In this exercice, we will create a chatbot that can answer questions on the CosmicWorks inventory, stored in the Azure Cosmos DB for NoSQL database.

* In the “lab4” folder, create a file call rag\_chatbot.py
* Copy/paste the import statements (note the additional import statements for Cosmos DB and tenacity)

import streamlit as st

import os, time

from azure.cosmos import CosmosClient

from tenacity import retry, wait\_random\_exponential, stop\_after\_attempt

from openai import AzureOpenAI

from dotenv import load\_dotenv

* From the previous exercise, copy the init\_env() and get\_completion() code to load the environment variables

def init\_env():

    st.set\_page\_config(page\_title="CosmicWorks Chatbot", page\_icon="🛒")

    st.title("🛒 CosmicWorks Chatbot")

    load\_dotenv("..\.env")

    os.environ["OPENAI\_API\_TYPE"] = "azure"

    os.environ["OPENAI\_API\_VERSION"] = os.getenv("AZURE\_OPENAI\_API\_VERSION")

    os.environ["AZURE\_OPENAI\_ENDPOINT"] = os.getenv("AZURE\_OPENAI\_ENDPOINT")

    os.environ["AZURE\_OPENAI\_API\_KEY"] = os.getenv("AZURE\_OPENAI\_API\_KEY")

    os.environ["AZURE\_OPENAI\_EMBEDDING\_MODEL"] = os.getenv("AZURE\_OPENAI\_EMBEDDING\_MODEL")

def get\_completion(openai\_client, model, prompt: str):

    start\_time = time.time()

    response = openai\_client.chat.completions.create(

        model = model,

        messages = [{"role": "user", "content": prompt}]

    )

    end\_time = time.time()

    elapsed\_time = end\_time - start\_time

    return response.choices[0].message.content, elapsed\_time

* Copy this code to init\_cosmos()

def init\_cosmos():

    COSMOS\_DB\_ENDPOINT = os.getenv('AZURE\_COSMOSDB\_NOSQL\_ENDPOINT')

    COSMOS\_DB\_KEY = os.getenv('AZURE\_COSMOSDB\_NOSQL\_KEY')

    DATABASE\_NAME = os.getenv('AZURE\_COSMOSDB\_NOSQL\_DATABASE\_NAME')

    CONTAINER\_NAME = os.getenv('AZURE\_COSMOSDB\_NOSQL\_CONTAINER\_NAME')

    client = CosmosClient(COSMOS\_DB\_ENDPOINT, COSMOS\_DB\_KEY)

    database = client.get\_database\_client(DATABASE\_NAME)

    products\_container = database.get\_container\_client(CONTAINER\_NAME)

    return client, database, products\_container

* Create a function to create embeddings from a string

@retry(wait=wait\_random\_exponential(min=1, max=20), stop=stop\_after\_attempt(10))

def generate\_embeddings(openai\_client, text):

    """

    Generates embeddings for a given text using the OpenAI API v1.x

    """

    return openai\_client.embeddings.create(

        input = text,

        model= os.getenv("AZURE\_OPENAI\_EMBEDDING\_MODEL")

    ).data[0].embedding

* Create a method to send a prompt to Azure OpenAI and generate a response

def get\_completion(openai\_client, model, prompt: str):

    start\_time = time.time()

    response = openai\_client.chat.completions.create(

        model = model,

        messages = [{"role": "user", "content": prompt}]

    )

    end\_time = time.time()

    elapsed\_time = end\_time - start\_time

    return response.choices[0].message.content, elapsed\_time

* Create a method to query the Cosmos DB database to get a list of documents similar to the user’s question

def get\_similar\_docs(openai\_client, container, query\_text, limit=5):

    """

        Get similar documents from Cosmos DB for NoSQL

        input:

            container: name of the container

            query\_text: user question

            limit: max number of documents to return

        output:

            documents: json documents similar to the user question

            elapsed\_time

    """

*# vectorize the question*

    query\_vector = generate\_embeddings(openai\_client, query\_text)

*# find product keys of products that match the question*

    query = f"""

        SELECT TOP {limit}

            VALUE c.productKey

        FROM c

        WHERE c.type = 'vector'

        ORDER BY VectorDistance(c.embedding, {query\_vector})

    """

    start\_time = time.time()

    results = container.query\_items(

        query=query,

        parameters=None,

        enable\_cross\_partition\_query=True

    )

*# get products from list of id*

    id\_list = [id for id in results]

    id\_list\_str = ', '.join([f"'{id}'" for id in id\_list])

    query = f"""

        SELECT \* FROM c

        WHERE c.type = 'product' AND c.id IN ({id\_list\_str})

    """

    results = container.query\_items(

        query=query,

        enable\_cross\_partition\_query=True

    )

    products = []

    for product in results:

        products.append(product)

    end\_time = time.time()

    elapsed\_time = end\_time - start\_time

    return products, elapsed\_time

* And finally, create the main() entry point for the application

def main():

    """ Use vector search in Cosmos DB for NoSQL to answer the question """

    top\_k = 10

    questions = [

        "List the categories of bikes that you have",

        "Can you list all types of mountain bikes?",

        "Can you provide more details on the Mountain-100?",

        "What is your most expensive bike?",

        "Do you have any helmets?"

    ]

    questions

    question = st.text\_input("How can I help you?")

*# init Cosmos DB and Azure OpenAI*

    client, database, container = init\_cosmos()

    openai\_client = AzureOpenAI(

        api\_key = os.getenv("AZURE\_OPENAI\_API\_KEY"),

        api\_version = os.getenv("AZURE\_OPENAI\_API\_VERSION"),

        azure\_endpoint =os.getenv("AZURE\_OPENAI\_ENDPOINT")

    )

    if st.button("Submit"):

        with st.spinner("Please wait.."):

*# get similar docs from Cosmos DB*

            docs, elapsed\_time = get\_similar\_docs(openai\_client, container, question, top\_k)

*# pass docs in the context and generate response using Azure OpenAI*

            user\_prompt = f"""

    Using the following CONTEXT, answer the user's question as best as possible.

    - Answer in English

    - Answer in markdown format

    CONTEXT:

    {docs}

    USER QUESTION:

    {question}

    ANSWER:

    """

            response, completion\_time = get\_completion(openai\_client, os.getenv("AZURE\_OPENAI\_CHAT\_MODEL"), user\_prompt)

            st.write(response, unsafe\_allow\_html = True)

            st.write("Elapsed time:", completion\_time)

if \_\_name\_\_ == "\_\_main\_\_":

    init\_env()

    main()

* Start the application with streamlit run rag\_chatbot.py

Here are some questions that you can ask:

List the categories of bikes that you have

Can you list all types of mountain bikes?

Can you provide more details on the Mountain-100?

What is your most expensive bike?

Do you have any helmets?

A screenshot of a computer

Description automatically generated

You can expand the dropdown to view the history of messages

Full code:

import streamlit as st

import os, time

from azure.cosmos import CosmosClient

from tenacity import retry, wait\_random\_exponential, stop\_after\_attempt

from openai import AzureOpenAI

from dotenv import load\_dotenv

def init\_env():

    st.set\_page\_config(page\_title="CosmicWorks Chatbot", page\_icon="🛒")

    st.title("🛒 CosmicWorks Chatbot")

    load\_dotenv("..\.env")

    os.environ["OPENAI\_API\_TYPE"] = "azure"

    os.environ["OPENAI\_API\_VERSION"] = os.getenv("AZURE\_OPENAI\_API\_VERSION")

    os.environ["AZURE\_OPENAI\_ENDPOINT"] = os.getenv("AZURE\_OPENAI\_ENDPOINT")

    os.environ["AZURE\_OPENAI\_API\_KEY"] = os.getenv("AZURE\_OPENAI\_API\_KEY")

    os.environ["AZURE\_OPENAI\_EMBEDDING\_MODEL"] = os.getenv("AZURE\_OPENAI\_EMBEDDING\_MODEL")

def get\_completion(openai\_client, model, prompt: str):

    start\_time = time.time()

    response = openai\_client.chat.completions.create(

        model = model,

        messages = [{"role": "user", "content": prompt}]

    )

    end\_time = time.time()

    elapsed\_time = end\_time - start\_time

    return response.choices[0].message.content, elapsed\_time

def init\_cosmos():

    COSMOS\_DB\_ENDPOINT = os.getenv('AZURE\_COSMOSDB\_NOSQL\_ENDPOINT')

    COSMOS\_DB\_KEY = os.getenv('AZURE\_COSMOSDB\_NOSQL\_KEY')

    DATABASE\_NAME = os.getenv('AZURE\_COSMOSDB\_NOSQL\_DATABASE\_NAME')

    CONTAINER\_NAME = os.getenv('AZURE\_COSMOSDB\_NOSQL\_CONTAINER\_NAME')

    client = CosmosClient(COSMOS\_DB\_ENDPOINT, COSMOS\_DB\_KEY)

    database = client.get\_database\_client(DATABASE\_NAME)

    products\_container = database.get\_container\_client(CONTAINER\_NAME)

    return client, database, products\_container

@retry(wait=wait\_random\_exponential(min=1, max=20), stop=stop\_after\_attempt(10))

def generate\_embeddings(openai\_client, text):

    """

    Generates embeddings for a given text using the OpenAI API v1.x

    """

    return openai\_client.embeddings.create(

        input = text,

        model= os.getenv("AZURE\_OPENAI\_EMBEDDING\_MODEL")

    ).data[0].embedding

def get\_completion(openai\_client, model, prompt: str):

    start\_time = time.time()

    response = openai\_client.chat.completions.create(

        model = model,

        messages = [{"role": "user", "content": prompt}]

    )

    end\_time = time.time()

    elapsed\_time = end\_time - start\_time

    return response.choices[0].message.content, elapsed\_time

def get\_similar\_docs(openai\_client, container, query\_text, limit=5):

    """

        Get similar documents from Cosmos DB for NoSQL

        input:

            container: name of the container

            query\_text: user question

            limit: max number of documents to return

        output:

            documents: json documents similar to the user question

            elapsed\_time

    """

*# vectorize the question*

    query\_vector = generate\_embeddings(openai\_client, query\_text)

*# find product keys of products that match the question*

    query = f"""

        SELECT TOP {limit}

            VALUE c.productKey

        FROM c

        WHERE c.type = 'vector'

        ORDER BY VectorDistance(c.embedding, {query\_vector})

    """

    start\_time = time.time()

    results = container.query\_items(

        query=query,

        parameters=None,

        enable\_cross\_partition\_query=True

    )

*# get products from list of id*

    id\_list = [id for id in results]

    id\_list\_str = ', '.join([f"'{id}'" for id in id\_list])

    query = f"""

        SELECT \* FROM c

        WHERE c.type = 'product' AND c.id IN ({id\_list\_str})

    """

    results = container.query\_items(

        query=query,

        enable\_cross\_partition\_query=True

    )

    products = []

    for product in results:

        products.append(product)

    end\_time = time.time()

    elapsed\_time = end\_time - start\_time

    return products, elapsed\_time

def main():

    """ Use vector search in Cosmos DB for NoSQL to answer the question """

    top\_k = 10

    questions = [

        "List the categories of bikes that you have",

        "Can you list all types of mountain bikes?",

        "Can you provide more details on the Mountain-100?",

        "What is your most expensive bike?",

        "Do you have any helmets?"

    ]

    questions

    question = st.text\_input("How can I help you?")

*# init Cosmos DB and Azure OpenAI*

    client, database, container = init\_cosmos()

    openai\_client = AzureOpenAI(

        api\_key = os.getenv("AZURE\_OPENAI\_API\_KEY"),

        api\_version = os.getenv("AZURE\_OPENAI\_API\_VERSION"),

        azure\_endpoint =os.getenv("AZURE\_OPENAI\_ENDPOINT")

    )

    if st.button("Submit"):

        with st.spinner("Please wait.."):

*# get similar docs from Cosmos DB*

            docs, elapsed\_time = get\_similar\_docs(openai\_client, container, question, top\_k)

*# pass docs in the context and generate response using Azure OpenAI*

            user\_prompt = f"""

    Using the following CONTEXT, answer the user's question as best as possible.

    - Answer in English

    - Answer in markdown format

    CONTEXT:

    {docs}

    USER QUESTION:

    {question}

    ANSWER:

    """

            response, completion\_time = get\_completion(openai\_client, os.getenv("AZURE\_OPENAI\_CHAT\_MODEL"), user\_prompt)

            st.write(response, unsafe\_allow\_html = True)

            st.write("Elapsed time:", completion\_time)

if \_\_name\_\_ == "\_\_main\_\_":

    init\_env()

    main()