Build a Laptop Consultant Chatbot Using Azure OpenAI

1. Objective

- Develop a chatbot that interacts with users to understand their laptop requirements.
- Provide tailored laptop recommendations based on user inputs.
- Utilize Azure OpenAI's chat completion capabilities via the Azure OpenAI Python library.
- Operate the chatbot within a Jupyter Notebook environment.

2. Problem Statement

- Selecting the right laptop can be challenging due to the variety of options available. Users often need assistance identifying laptops that suit specific needs such as gaming, business, or general use.
- This exercise aims to build a conversational chatbot that gathers user requirements and suggests suitable laptops using Azure OpenAI.

3. Inputs / Shared Artifacts

- **Azure OpenAI Resource:**
    - API key, endpoint URL, and deployment name (to be set as environment variables).
- **ChromaDB:**
    - Used for storing and retrieving embedded information about laptops for recommendations.

4. Expected Outcome

- A functional chatbot running inside a Jupyter Notebook that:
    - Engages users in a conversation to understand their laptop needs.
    - Provides personalized laptop recommendations using Azure OpenAI chat completions.
    - Allows users to exit the chat gracefully.

5. Concepts Covered

- **Azure OpenAI API** integration via the official OpenAI Python library.
- **Secure environment variable management** in Python.
- **Building conversational agents** using prompt engineering.
- **Interactive chatbot loop** in a notebook environment.
- **ChromaDB integration** for storing and retrieving embeddings of product data.

6. Example: Step-by-Step Instructions with Code

```
# pip install openai chromadb

import chromadb
from openai import AzureOpenAI
import os

os.environ["AZURE_OPENAI_EMBEDDING_API_KEY"] = ""
os.environ["AZURE_OPENAI_EMBEDDING_ENDPOINT"] = ""
os.environ["AZURE_OPENAI_EMBED_MODEL"]= "text-embedding-3-small"

os.environ["AZURE_OPENAI_LLM_API_KEY"] = ""
os.environ["AZURE_OPENAI_LLM_ENDPOINT"] = ""
os.environ["AZURE_OPENAI_LLM_MODEL"]= "GPT-4o-mini"

# ---- EMBEDDING CONFIG ----
AZURE_OPENAI_EMBEDDING_API_KEY = os.getenv("AZURE_OPENAI_EMBEDDING_API_KEY")
AZURE_OPENAI_EMBEDDING_ENDPOINT =
os.getenv("AZURE_OPENAI_EMBEDDING_ENDPOINT")
AZURE_OPENAI_EMBED_MODEL = os.getenv("AZURE_OPENAI_EMBED_MODEL")

# ---- LLM CONFIG ----
AZURE_OPENAI_LLM_API_KEY = os.getenv("AZURE_OPENAI_LLM_API_KEY")
AZURE_OPENAI_LLM_ENDPOINT = os.getenv("AZURE_OPENAI_LLM_ENDPOINT")
AZURE_OPENAI_LLM_MODEL = os.getenv("AZURE_OPENAI_LLM_MODEL")

# ---- CLIENTS ----
```

```
embedding_client = AzureOpenAI(
    api_key=AZURE_OPENAI_EMBEDDING_API_KEY,
    azure_endpoint=AZURE_OPENAI_EMBEDDING_ENDPOINT,
    api_version="2023-05-15"
)

llm_client = AzureOpenAI(
    api_key=AZURE_OPENAI_LLM_API_KEY,
    azure_endpoint=AZURE_OPENAI_LLM_ENDPOINT,
    api_version="2023-05-15"
)




# ---- GET EMBEDDING ----

def get_embedding(text):
    response = embedding_client.embeddings.create(
        input=text,
        model=AZURE_OPENAI_EMBED_MODEL
    )
    return response.data[0].embedding



# ---- CALL LLM ----

def ask_llm(context, user_input):
    system_prompt = (
        "You are a helpful assistant specializing in laptop recommendations.
"
        "Use the provided context to recommend the best laptop(s) for the
user needs."
    )
    user_prompt = (
        f"User requirements: {user_input}\n\n"
        f"Context (top relevant laptops):\n{context}\n\n"
        "Based on the above, which laptop(s) would you recommend and why?"
    )
    messages = [

        {"role": "system", "content": system_prompt},
        {"role": "user", "content": user_prompt}

    ]
    response = llm_client.chat.completions.create(
        model=AZURE_OPENAI_LLM_MODEL,
        messages=messages
    )
    return response.choices[0].message.content
```

```
# ---- CHROMADB ----

chroma_client = chromadb.Client()
collection = chroma_client.create_collection(name="laptops")

# --------- Add Sample Laptops ---------

laptops = [

    {
        "id": "1",
        "name": "Gaming Beast Pro",
        "description": "A high-end gaming laptop with RTX 4080, 32GB RAM, and
1TB SSD. Perfect for hardcore gaming.",
        "tags": "gaming, high-performance, windows"
    },
    {
        "id": "2",
        "name": "Business Ultrabook X1",
        "description": "A lightweight business laptop with Intel i7, 16GB
RAM, and long battery life. Great for productivity.",
        "tags": "business, ultrabook, lightweight"
    },

    {
        "id": "3",
        "name": "Student Basic",
        "description": "Affordable laptop with 8GB RAM, 256GB SSD, and a
reliable battery. Ideal for students and general use.",
        "tags": "student, budget, general"
    },

]



# ---- ADD LAPTOPS TO CHROMADB ----

for laptop in laptops:
    embedding = get_embedding(laptop["description"])
    collection.add(
        embeddings=[embedding],
        documents=[laptop["description"]],
        ids=[laptop["id"]],
        metadatas=[{
            "name": laptop["name"],
            "tags": laptop["tags"]
        }],
    )



# ---- AUTOMATED MOCK INPUTS ----
```

```
user_queries = [
    "I want a lightweight laptop with long battery life for business trips.",
    "I need a laptop for gaming with the best graphics card available.",
    "Looking for a budget laptop suitable for student tasks and general
browsing."
]
```

```
def build_context(results, n_context=3):
    docs = results["documents"][0]
    metas = results["metadatas"][0]
    context_str = ""
    for doc, meta in zip(docs, metas):
        context_str += (
            f"Name: {meta['name']}\n"
            f"Description: {doc}\n"
            f"Tags: {meta['tags']}\n\n"
        )
    return context_str.strip()
```

```
# ---- MAIN RAG LOOP ----

for user_input in user_queries:
    print("="*60)
    print(f"User input: {user_input}")
    # Step 1: Retrieve relevant laptops via vector search
    query_embedding = get_embedding(user_input)
    results = collection.query(query_embeddings=[query_embedding],
n_results=3)
    # Step 2: Build context for LLM
    context = build_context(results)
    # Step 3: Get recommendation from LLM
    llm_output = ask_llm(context, user_input)
    print("\nLLM Recommendation:\n")
    print(llm_output)
    print("="*60 + "\n")
```

7. Final Submission Checklist

- Submit your Python script with:
    - Environment variable setup (with placeholders for API keys and endpoint).
    - Complete chatbot code as shown above.

- Automatic input by looping through the mock queries (3 queries) list, so there's no for input() at all.