🧪 Hands-On Lab: Build a Basic LLM Pipeline Using OpenAI API with Prompt Templates

✅ Overview:

In this lab, you'll:

- Set up the OpenAI API
- Design a prompt template for a specific task
- Send structured prompts to an LLM
- Capture, parse, and display the output
- Wrap the logic into a reusable pipeline

We'll use **Python**, the **OpenAI API**, and a basic terminal interface (no front-end needed). By the end, you'll have a working local pipeline that can perform intelligent task-specific completions.

🔧 Step 1: Install Required Tools

✅ Prerequisites:

- Python 3.7+
- An OpenAI API key (Sign up and get API key here)

🖥️ Terminal Commands:

```
# Create a virtual environment (optional but recommended)
python -m venv llm-pipeline-env
source llm-pipeline-env/bin/activate  # macOS/Linux
llm-pipeline-env\Scripts\activate.bat  # Windows

# Install required packages
pip install openai python-dotenv
```

📁 Step 2: Set Up Your Environment Variables

Create a file named `.env` in your project folder:

```
touch .env
```

Add your API key inside `.env` :

```
OPENAI_API_KEY=your-api-key-here
```

This keeps your key secure and out of your codebase.

📜 Step 3: Create Your Prompt Template

Let's say our task is **"Summarize any article or paragraph into 3 bullet points."**

Create a prompt template with variable injection:

```
SUMMARY_PROMPT_TEMPLATE = """
You are a helpful assistant. Summarize the following text
into 3 concise bullet points:
---
{text}
---
Bullet Points:
"""
```

Templates help you **structure** the prompt for repeatable performance.

🤖 Step 4: Write the Core LLM Pipeline Code

Create a file called `llm_pipeline.py`

```
import os
import openai
from dotenv import load_dotenv

# Load API key
load_dotenv()
openai.api_key = os.getenv("OPENAI_API_KEY")

# Template with placeholder
SUMMARY_PROMPT_TEMPLATE = """
You are a helpful assistant. Summarize the following
text into 3 concise bullet points:
---
{text}
---
Bullet Points:
"""

def run_summary_pipeline(input_text):
    # Inject user input into the prompt
    prompt = SUMMARY_PROMPT_TEMPLATE.format(text=input_text)

    # Call OpenAI's completion endpoint (GPT-3.5 or GPT-4)
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",  # You can use "gpt-4" if you have access
        messages=[
            {"role": "system", "content": "You are a summarization assistant."},
            {"role": "user", "content": prompt}
```

```
        ],
        temperature=0.7,
        max_tokens=300
    )

    return response['choices'][0]['message']['content']

if __name__ == "__main__":
    print("Paste your text below to summarize:\n")
    user_input = input()
    output = run_summary_pipeline(user_input)
    print("\n🧠 Summary:\n", output)
```

▶️ Step 5: Run the Pipeline

In your terminal:

```
python llm_pipeline.py
```

Paste a sample paragraph like:

> “Artificial intelligence (AI) is transforming industries by automating tasks, analyzing data at scale, and improving decision-making. Generative AI, specifically, is unlocking new capabilities in content creation, such as text, images, and even music. As enterprises adopt these tools, it’s important to balance innovation with ethical considerations.”

Your output will look something like:

```
🧠 Summary:
- AI is revolutionizing industries through automation and data analysis.
- Generative AI enables creation of diverse content like text and images.
- Ethical concerns must be addressed as adoption grows.
```

🔁 Step 6: Extend with More Prompt Templates

Try other templates for:

- Email rewriting
- Extracting action items from meeting notes
- Converting natural language to SQL
- Classifying user intent

Example:

```
EMAIL_FIX_TEMPLATE = """
Revise the following email to sound more professional,
but keep it short and polite:
---
{text}
---
```

```
Rewritten Email:
"""
```

📦 Optional: Wrap in a Function for API Reuse

This allows the pipeline to be reused in agents, web apps, or microservices.

```
def generate_with_prompt(template, input_text,
role="assistant", temperature=0.5):
    prompt = template.format(text=input_text)
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": f"You are a helpful {role}."},
            {"role": "user", "content": prompt}
        ],
    temperature=temperature,
    max_tokens=500
)
return response['choices'][0]['message']['content']
```

✅ What You Learned

- How to set up the OpenAI API in Python
- How to use structured prompt templates
- How to inject user input into LLMs
- How to build a modular, testable LLM pipeline
- How to scale to more use cases with prompt abstraction
