# Lesson 2 Demo 2

# LangGraph Prompt Chaining for Automated Product Descriptions

**Objective:** To showcase prompt chaining with LangGraph to generate structured product descriptions

Creating high-quality product descriptions manually is time-consuming and inconsistent. Businesses need a streamlined, automated way to generate compelling and structured product descriptions that highlight key features, benefits, and marketing messages. Traditional methods lack scalability, efficiency, and coherence across different descriptions.

**Prerequisites:**

1. Create a virtual environment
2. Activate the virtual environment
3. Install the libraries in `requirements.txt`

**Steps to be followed:**

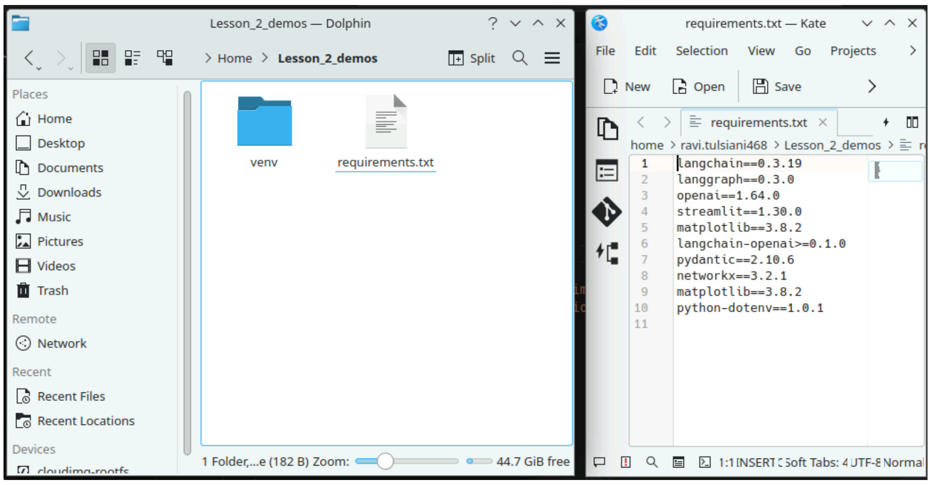
1. Set up the environment
2. Import the required libraries
3. Define graph state
4. Generate a basic product description
5. Add features and benefits
6. Create a marketing message
7. Finalize the product description
8. Build the LangGraph workflow
9. Visualize the workflow
10. Create the Streamlit UI
11. Run the webapp

## Step 1: Set up the environment

1.1 Open command prompt and go to the "**Lesson_2_demos**" folder (which we created in Demo_1) using the command given below:
**mkdir Lesson_2_demos** (not needed if the folder is already created in Demo1)
**cd Lesson_2_demos**

1.2 After this, activate the virtual environment using the command below:
**python3 -m venv venv** (not needed if the virtual env. is already created in Demo1)
**source venv/bin/activate**

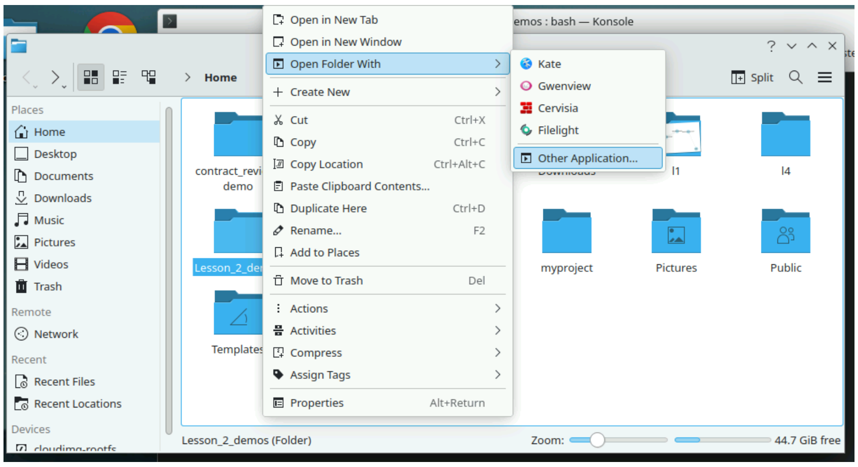
1.3 Now, create a `requirements.txt` file inside the folder with required libraries (not needed if already done in Demo1):



```
langchain==0.3.19
langgraph==0.3.0
openai==1.64.0
streamlit==1.30.0
matplotlib==3.8.2
langchain-openai>=0.1.0
pydantic==2.10.6
networkx==3.2.1
matplotlib==3.8.2
python-dotenv==1.0.1
```

1.4 Install all the required libraries using the command below:
**pip install -r requirements.txt** (not needed if already done in Demo1)

1.5 Now, open the folder using VS Code editor:



1.6 After this, open a new Python file using the "**New File**" option and name it as "**Demo2**".

## Step 2: Import the required libraries

2.1 Import necessary libraries to build the UI, process user input, and interact with the LLM.

2.2 Build the client with the required parameters and their respective values.

```
import openai
from typing_extensions import TypedDict
from langgraph.graph import StateGraph, START, END
from dotenv import load_dotenv
import os
from dataclasses import asdict, dataclass
import matplotlib.pyplot as plt
import networkx as nx
import streamlit as st

client = openai.AzureOpenAI(

api_key="2ABecnfxzhRg4M5D6pBKiqxXVhmGB2WvQ0aYKkbTCPsj0JLKsZPfJQQJ99BDAC77bzfXJ3w3AAA
BACOGi3sC",
    api_version="2023-12-01-preview",
    #azure_endpoint="https://openai-api-management-gw.azure-api.net/"
    azure_endpoint="https://openai-api-management-gw.azure-api.net/deployments/gpt-
4o-mini/chat/completions?api-version=2023-12-01-preview"
)
```

## Step 3: Define graph state

3.1 The first step is to define a structured state to hold data throughout the workflow. The State class is created using `TypedDict` to store fields such as `product_name`, `basic_description`, `features_benefits`, `marketing_message`, and `final_description`.

3.2 This structure ensures that each step can access and modify relevant data during the execution of the LangGraph workflow.

```
# Graph state definition
# @dataclass
class State(TypedDict):
    product_name: str
    basic_description: str
    features_benefits: str
    marketing_message: str
    final_description: str
```

## Step 4: Generate a basic product description

4.1 The first step is to define a structured state to hold data throughout the workflow. The State class is created using TypedDict to store fields such as product_name, basic_description, features_benefits, marketing_message, and final_description.

4.2 This structure ensures that each step can access and modify relevant data during the execution of the LangGraph workflow.

```
# Generate a basic product description
def generate_basic_description(state):
    """Generate a basic description for the product."""
    response = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {"role": "system", "content": "You are a helpful assistant that generates 
brief product descriptions."},
            {"role": "user", "content": f"Write a brief description of a product 
named '{state['product_name']}'."}
        ]
    )
    basic_description = response.choices[0].message.content
    return {"basic_description": basic_description}
```

## Step 5: Add features and benefits

5.1 The add_features_benefits function enhances the basic product description by requesting GPT to list key features and benefits. Using the previously generated description, it prompts the model to extract important details that highlight the product’s unique selling points.

5.2 The response is stored in the features_benefits field, adding more depth to the content.

```
# Add key features and benefits to the product description
def add_features_benefits(state: State):
    """Add features and benefits to the product description."""
    response = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[{"role": "user", "content": f"List key features and benefits of 
the product: {state['basic_description']}"}]
    )
    features_benefits = response.choices[0].message.content
    return {"features_benefits": features_benefits}
```

## Step 6: Create a marketing message

6.1 In this step, create_marketing_message crafts a compelling marketing message based on the product’s features. It takes the features_benefits field and prompts GPT to create an engaging, persuasive message that resonates with potential customers.

6.2 The output is stored in marketing_message, ensuring a smooth transition to the final description.

```
# Create a compelling marketing message based on the product's features
def create_marketing_message(state: State):
    """Create a marketing message for the product."""
    response = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[{"role": "user", "content": f"Create a compelling marketing
message for the product: {state['features_benefits']}"}]
    )
    marketing_message = response.choices[0].message.content
    return {"marketing_message": marketing_message}
```

## Step 7: Finalize the product description

7.1 The polish_final_description function refines the product description by incorporating the marketing message. It requests GPT to blend all previous elements into a polished, well-structured final output.

7.2 The result is stored in final_description and ready for use in marketing materials.

```
# Final polish and completion of the product description
def polish_final_description(state: State):
    """Polish and finalize the product description."""
    response = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[{"role": "user", "content": f"Polish and finalize the product
description, incorporating the marketing message: {state['marketing_message']}"}]
    )
    final_description = response.choices[0].message.content
    return {"final_description": final_description}
```

## Step 8: Build the LangGraph workflow

8.1 The build_workflow function constructs the prompt chaining workflow using LangGraph. It defines each step as a node and establishes connections between them to ensure sequential execution.

8.2 The workflow starts with generate_basic_description, progresses through the steps, and ends with polish_final_description, producing a fully structured product description.

```
# Function to build the workflow (Separate from Streamlit logic)
def build_workflow():
    """Build and compile the workflow steps using LangGraph."""
    # Build the workflow for generating the product description
    workflow = StateGraph(State)

    # Add nodes to the workflow (steps in the process)
    workflow.add_node("generate_basic_description", generate_basic_description)  #
Step 1: Basic Description
    workflow.add_node("add_features_benefits", add_features_benefits)  # Step 2:
Features and Benefits
    workflow.add_node("create_marketing_message", create_marketing_message)  # Step
3: Marketing Message
    workflow.add_node("polish_final_description", polish_final_description)  # Step
4: Final Description



    # Add edges to connect the nodes (steps in order)
    workflow.add_edge(START, "generate_basic_description")
    workflow.add_edge("generate_basic_description", "add_features_benefits")
    workflow.add_edge("add_features_benefits", "create_marketing_message")
    workflow.add_edge("create_marketing_message", "polish_final_description")
    workflow.add_edge("polish_final_description", END)


    # Compile the workflow into a chain of actions
    chain = workflow.compile()

    return chain
```

## Step 9: Visualize the workflow

9.1 The visualize_workflow function generates a visual representation of the workflow using NetworkX and Matplotlib. It creates a directed graph that illustrates the step-by-step process of transforming a product name into a fully developed marketing description.

9.2 The visualization helps in understanding the logical flow and dependencies of each stage.

```
# Function to visualize the workflow (saved as an image)
def visualize_workflow():
    """Visualize and save the workflow as an image."""

    graph = nx.DiGraph()
    edges = [
    ("START", "generate_basic_description"),
    ("generate_basic_description", "add_features_benefits"),
    ("add_features_benefits", "create_marketing_message"),
    ("create_marketing_message", "polish_final_description")]

    graph.add_edges_from(edges)

    plt.figure(figsize=(8, 5))
    nx.draw(graph, with_labels=True, node_color='lightblue', edge_color='gray',
node_size=2000, font_size=10, font_weight='bold')
    plt.savefig("workflow.png")
```

## Step 10: Create the Streamlit UI

11.1 The Streamlit app collects user input, initializes a `State` object with the product name, and runs the compiled LangGraph workflow. Each step updates the state progressively, generating a structured product description.

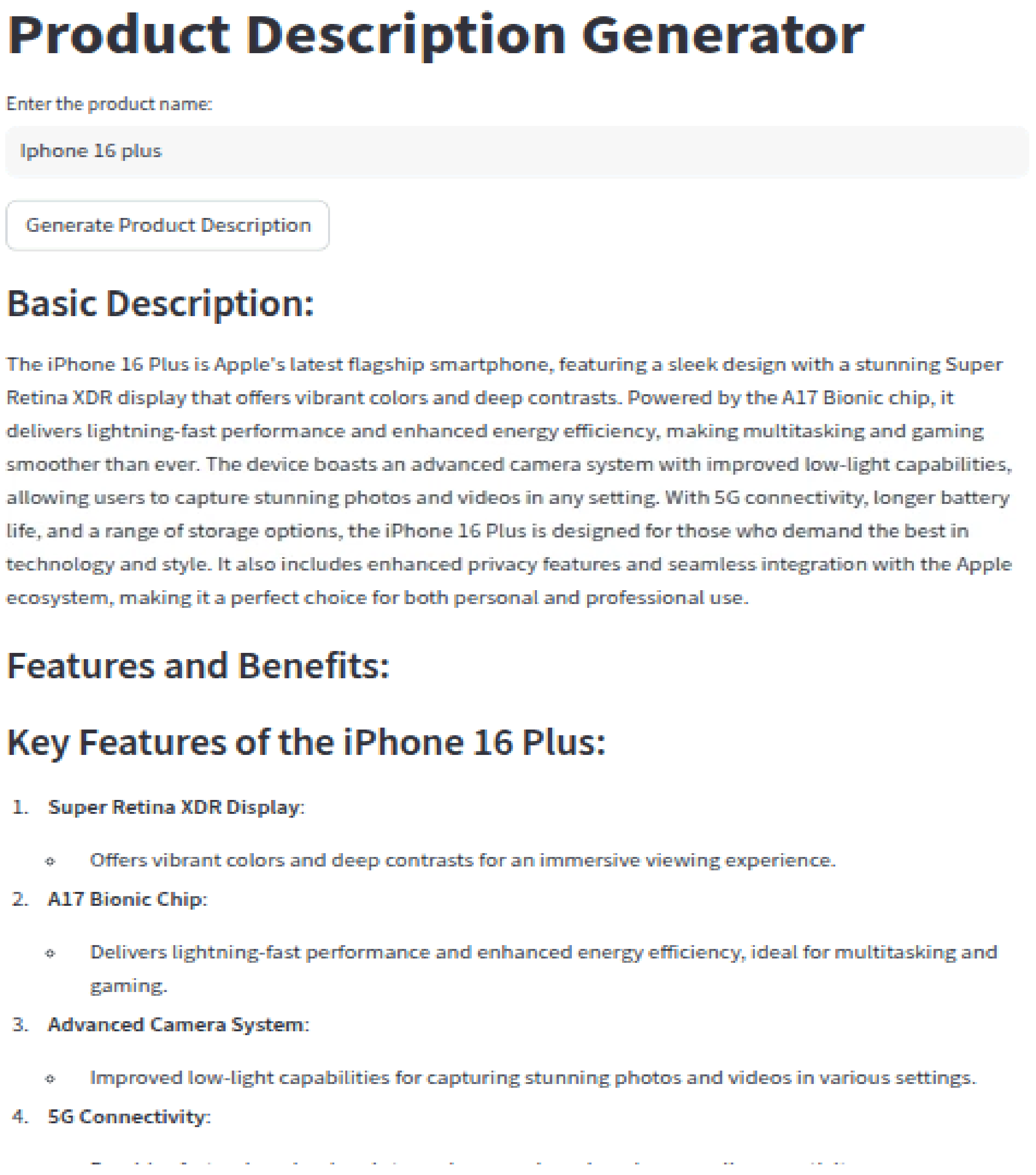
11.2 The results, including the basic description, features, marketing message, and final version, are displayed, along with a workflow visualization.

```
# Main Streamlit function
def run_streamlit_app():
    """Handles the entire app logic: input, workflow, and output."""
    # Title for the app
    st.title("Product Description Generator")

    # Step 1: Take product name as input from the user
    product_name = st.text_input("Enter the product name:")# "Smart Water Bottle"

    # Step 2: Button to generate product description
    if st.button("Generate Product Description"):
        # Create the initial state with product name and empty fields for
description steps
        state = State(product_name=product_name, basic_description="",
features_benefits="", marketing_message="", final_description="")

        # Build and run the workflow
        chain = build_workflow()

        # Run the workflow and get the results
        result = chain.invoke(state)

        # Display the results in Streamlit
        st.subheader("Basic Description:")
        st.write(result["basic_description"])

        st.subheader("Features and Benefits:")
        st.write(result["features_benefits"])

        st.subheader("Marketing Message:")
        st.write(result["marketing_message"])

        st.subheader("Final Description:")
        st.write(result["final_description"])

        # Step 3: Visualize the workflow and save it as an image
        visualize_workflow()
        st.image("workflow.png", caption="Product Description Workflow")

if __name__ == "__main__":
    run_streamlit_app()
```

### Step 11: Run the webapp

11.1 Save the file and then run the streamlit webapp from command prompt using the command given below:
**streamlit run Demo2.py**

**Output:**



# Product Description Generator

Enter the product name:

Iphone 16 plus

Generate Product Description

## Basic Description:

The iPhone 16 Plus is Apple's latest flagship smartphone, featuring a sleek design with a stunning Super Retina XDR display that offers vibrant colors and deep contrasts. Powered by the A17 Bionic chip, it delivers lightning-fast performance and enhanced energy efficiency, making multitasking and gaming smoother than ever. The device boasts an advanced camera system with improved low-light capabilities, allowing users to capture stunning photos and videos in any setting. With 5G connectivity, longer battery life, and a range of storage options, the iPhone 16 Plus is designed for those who demand the best in technology and style. It also includes enhanced privacy features and seamless integration with the Apple ecosystem, making it a perfect choice for both personal and professional use.

## Features and Benefits:

## Key Features of the iPhone 16 Plus:

1. **Super Retina XDR Display**:
   - Offers vibrant colors and deep contrasts for an immersive viewing experience.
2. **A17 Bionic Chip**:
   - Delivers lightning-fast performance and enhanced energy efficiency, ideal for multitasking and gaming.
3. **Advanced Camera System**:
   - Improved low-light capabilities for capturing stunning photos and videos in various settings.
4. **5G Connectivity**:

By following the above-mentioned steps, you have successfully highlighted the power of LangGraph in automating structured product description generation through prompt chaining. Breaking content into logical steps and refining it progressively ensures consistency, accuracy, and scalability. This approach streamlines AI-driven content creation, making it more efficient and adaptable for various business needs.