# Lab: Using LangChain with MLflow

Welcome to this interactive tutorial designed to introduce you to LangChain and its integration with MLflow. This tutorial is structured as a notebook to provide a hands-on, practical learning experience with the simplest and most core features of LangChain.

### What You Will Learn

- **Understanding LangChain**: Get to know the basics of LangChain and how it is used in developing applications powered by language models.
- **Chains in LangChain**: Explore the concept of `chains` in LangChain, which are sequences of actions or operations orchestrated to perform complex tasks.
- **Integration with MLflow**: Learn how LangChain integrates with MLflow, a platform for managing the machine learning lifecycle, including logging, tracking, and deploying models.
- **Practical Application**: Apply your knowledge to build a LangChain chain that acts like a sous chef, focusing on the preparation steps of a recipe.

### Background on LangChain

LangChain is a Python-based framework that simplifies the development of applications using language models. It is designed to enhance context-awareness and reasoning in applications, allowing for more sophisticated and interactive functionalities.

### What is a Chain?

- **Chain Definition**: In LangChain, a `chain` refers to a series of interconnected components or steps designed to accomplish a specific task.
- **Chain Example**: In our tutorial, we'll create a chain that simulates a sous chef's role in preparing ingredients and tools for a recipe.

### Tutorial Overview

In this tutorial, you will:

1. **Set Up LangChain and MLflow**: Initialize and configure both LangChain and MLflow.
2. **Create a Sous Chef Chain**: Develop a LangChain chain that lists ingredients, describes preparation techniques, organizes ingredient staging, and details cooking implements preparation for a given recipe.
3. **Log and Load the Model**: Utilize MLflow to log the chain model and then load it for prediction.
4. **Run a Prediction**: Execute the chain to see how it would prepare a restaurant dish for a specific number of customers.

By the end of this tutorial, you will have a solid foundation in using LangChain with MLflow and an understanding of how to construct and manage chains for practical applications.

Let's dive in and explore the world of LangChain and MLflow!

### Prerequisites

In order to get started with this lab, we're going to need OPENAI_API_KEY.

To install the dependent packages simply run:

```
pip install "openai<1" tiktoken langchain
```

### API Key Security Overview

API keys, especially for SaaS Large Language Models (LLMs), are as sensitive as financial information due to their connection to billing.

**Essential Practices:**

- **Confidentiality**: Always keep API keys private.
- **Secure Storage**: Prefer environment variables or secure services.
- **Frequent Rotation**: Regularly update keys to avoid unauthorized access.

```
import os

from langchain.chains import LLMChain
from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate

import mlflow

import os
os.environ['OPENAI_API_KEY'] = "OPEN_AI_KEY_HERE"
```

## Configuring the OpenAI Completions Model in LangChain

In this section of the tutorial, we have configured the OpenAI model with specific parameters suitable for generating language completions. We're using a Completions model, not ChatCompletions, which means each request is independent, and the entire prompt needs to be included every time to generate a response.

### Understanding the Completions Model

- **Completions Model**: This model does not maintain contextual information across requests. It's ideal for tasks where each request is standalone and doesn't depend on past interactions. Offers flexibility for a variety of non-conversational applications.
- **No Contextual Memory**: The lack of memory of previous interactions means the model is best suited for one-off requests or scenarios where continuity of the conversation is not required.
- **Comparisons with the ChatCompletions Model Type**: Tailored for conversational AI, maintaining context across multiple exchanges for a continuous conversation. Suitable for chatbots or applications where dialogue history is crucial.

In this tutorial, we use the Completions model for its simplicity and effectiveness in handling individual, independent requests, aligning with our tutorial's focus on preparation steps before cooking.

```
llm = OpenAI(temperature=0.1, max_tokens=1000)
```

## Explanation of the Template Instruction for Sous Chef Simulation

In this part of the tutorial, we have crafted a detailed prompt template that simulates the role of a fine dining sous chef. This template is designed to guide the LangChain model in preparing for a dish, focusing exclusively on the mise-en-place process.

### Breakdown of the Template Instruction

- **Sous Chef Roleplay**: The prompt places the language model in the role of a sous chef, emphasizing meticulous preparation.
- **Task Outline**:

1. **List the Ingredients**: Instructs the model to itemize all necessary ingredients for a given dish.
2. **Preparation Techniques**: Asks the model to describe necessary techniques for ingredient preparation, such as cutting and processing.
3. **Ingredient Staging**: Requires the model to provide detailed staging instructions for each ingredient, considering the sequence and timing of use.
4. **Cooking Implements Preparation**: Guides the model to list and prepare all cooking tools required for the dish's preparation phase.

- **Scope Limitation**: The template is explicitly designed to stop at the preparation stage, avoiding the actual cooking process. It focuses on setting up everything needed for the chef to begin cooking.
- **Dynamic Inputs**: The template is adaptable to different recipes and customer counts, as indicated by placeholders `{recipe}` and `{customer_count}`.

This template instruction is a key component of the tutorial, demonstrating how to leverage LangChain declaring instructive prompts with parametrized features geared toward single-purpose completions-style applications.

```
template_instruction = (
    "Imagine you are a fine dining sous chef. Your task is to meticulously prepare for a dish, focusing on the mise-en-place process."
    "Given a recipe, your responsibilities are: "
    "1. List the Ingredients: Carefully itemize all ingredients required for the dish, ensuring every element is accounted for. "
    "2. Preparation Techniques: Describe the techniques and operations needed for preparing each ingredient. This includes cutting, "
    "processing, or any other form of preparation. Focus on the art of mise-en-place, ensuring everything is perfectly set up before cooking begins."
    "3. Ingredient Staging: Provide detailed instructions on how to stage and arrange each ingredient. Explain where each item should be placed for "
    "efficient access during the cooking process. Consider the timing and sequence of use for each ingredient. "
    "4. Cooking Implements Preparation: Enumerate all the cooking tools and implements needed for each phase of the dish's preparation. "
    "Detail any specific preparation these tools might need before the actual cooking starts and describe what pots, pans, dishes, and "
    "other tools will be needed for the final preparation."
    "Remember, your guidance stops at the preparation stage. Do not delve into the actual cooking process of the dish. "
    "Your goal is to set the stage flawlessly for the chef to execute the cooking seamlessly."
    "The recipe you are given is for: {recipe} for {customer_count} people. "
)
```

### Constructing the LangChain Chain

We start by setting up a `PromptTemplate` in LangChain, tailored to our sous chef scenario. The template is designed to dynamically accept inputs like the recipe name and customer count. Then, we initialize an `LLMChain` by combining our OpenAI language model with the prompt template, creating a chain that can simulate the sous chef's preparation process.

### Logging the Chain in MLflow

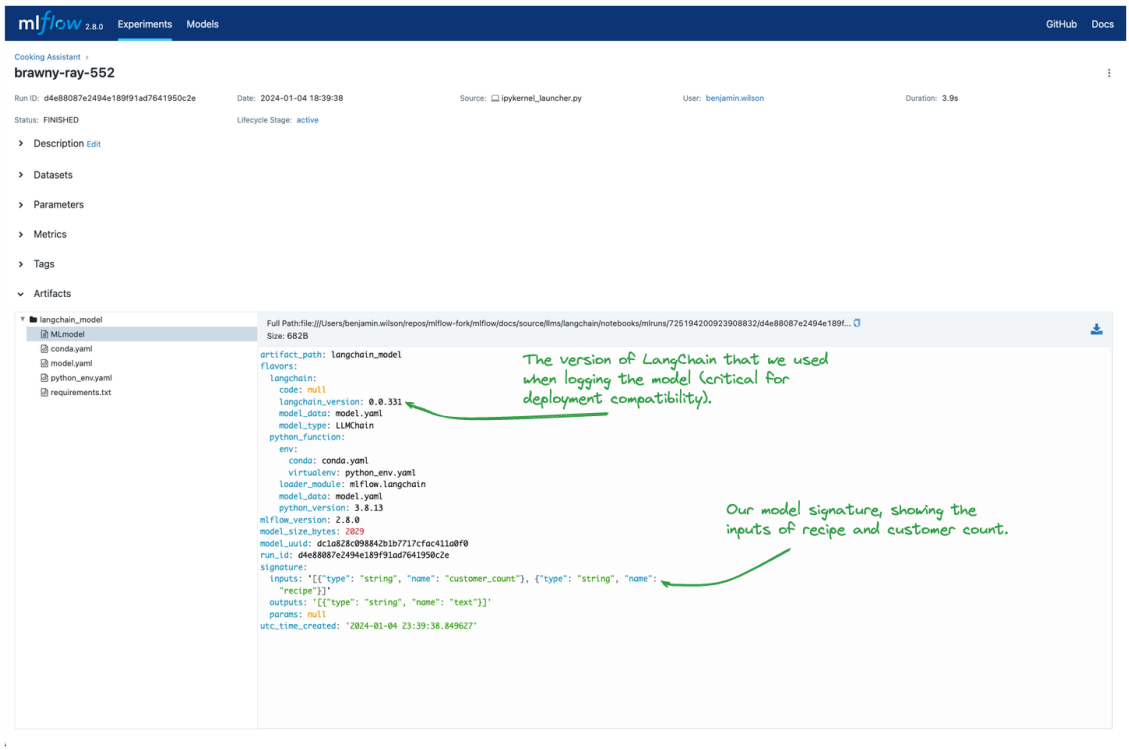
With the chain ready, we proceed to log it in MLflow. This is done within an MLflow run, which not only logs the chain model under a specified name but also tracks various details about the model. The logging process ensures that all aspects of the chain are recorded, allowing for efficient version control and future retrieval.

```
prompt = PromptTemplate(
    input_variables=["recipe", "customer_count"],
    template=template_instruction,
)
chain = LLMChain(llm=llm, prompt=prompt)

mlflow.set_experiment("Cooking Assistant")

with mlflow.start_run():
    model_info = mlflow.langchain.log_model(chain, "langchain_model")
```

If we navigate to the MLflow UI, we'll see our logged LangChain model.




## Loading the Model and Predicting with MLflow

In this part of our tutorial, we demonstrate the practical application of the logged LangChain model using MLflow. We load the model and run a prediction for a specific dish, showcasing the model's ability to assist in culinary preparation.

### Model Loading and Execution

After logging our LangChain chain with MLflow, we proceed to load the model using MLflow's `pyfunc.load_model` function. This step is crucial as it brings our previously logged model into an executable state.

We then input a specific recipe along with the customer count into our model. In this case, we use the recipe for "boeuf bourginon" and specify that it's for 12 customers. The model, acting as a sous chef, processes this information and generates detailed preparation instructions.

#### Output from the Model

The model's output provides a comprehensive guide on preparing "boeuf bourginon," covering several critical aspects:

- **Ingredients List**: A detailed enumeration of all necessary ingredients, quantified and tailored for the specified number of customers.
- **Preparation Techniques**: Step-by-step instructions on how to prepare each ingredient, following the principles of mise-en-place.
- **Ingredient Staging**: Guidance on how to organize and stage the ingredients, ensuring efficient access and use during the cooking process.
- **Cooking Implements Preparation**: Instructions on preparing the necessary cooking tools and implements, from pots and pans to bowls and colanders.

This example demonstrates the power and utility of combining LangChain and MLflow in a practical scenario. It highlights how such an integration can effectively translate complex requirements into actionable steps, aiding in tasks that require precision and careful planning.

```
loaded_model = mlflow.pyfunc.load_model(model_info.model_uri)

dish1 = loaded_model.predict({"recipe": "boeuf bourginon", "customer_count": "4"})

print(dish1[0])
```

```
dish2 = loaded_model.predict({"recipe": "Okonomiyaki", "customer_count": "12"})

print(dish2[0])
```

## Conclusion

In the final step of our tutorial, we execute another prediction using our LangChain model. This time, we explore the preparation for "Okonomiyaki," a Japanese dish, for 12 customers. This demonstrates the model's adaptability and versatility across various cuisines.

#### Additional Prediction with the Loaded Model

The model processes the input for "Okonomiyaki" and outputs detailed preparation steps. This includes listing the ingredients, explaining the preparation techniques, guiding ingredient staging, and detailing the required cooking implements, showcasing the model's capability to handle diverse recipes with precision.

#### What We've Learned

- **Model Versatility**: The tutorial highlighted the LangChain framework for assembling component parts of a basic LLM application, chaining a specific instructional prompt to a Completions-style LLM.
- **MLflow's Role in Model Management**: The integration of LangChain with MLflow demonstrated effective model lifecycle management, from creation and logging to prediction execution.

## Conclusion

This lab offered an insightful journey through creating, managing, and utilizing a LangChain model with MLflow for culinary preparation. It showcased the practical applications and adaptability of LangChain in complex scenarios. We

hope this experience has provided valuable knowledge and encourages you to further explore and innovate using LangChain and MLflow in your projects. Happy coding!