In-Context Engineering and Prompt Templates

Estimated time needed: **30** minutes

Overview

You're stepping into the world of prompt engineering, where each command you craft has the power to guide intelligent LLM systems toward specific outcomes. In this tutorial, you will explore the foundational aspects of prompt engineering, dive into advanced techniques of in-context learning, such as few-shot and self-consistent learning, and learn how to effectively use tools like Langchain.

Start by understanding the basics—how to formulate prompts that communicate effectively with AI. From there, we'll explore how the Langchain prompt template can simplify and enhance this process, making it more structured and efficient.

As you progress, you'll learn to apply these skills in practical scenarios, creating sophisticated applications like QA bots and text summarization tools. By using the Langchain prompt template, you'll see firsthand how structured prompting can streamline the development of these applications, transforming complex requirements into clear, concise tasks for AI.

By the end of this tutorial, you'll not only master the different techniques of prompt engineering but also acquire hands-on experience in applying these techniques to real-world problems, ensuring you're well-prepared to harness the full potential of AI in various settings.

Table of Contents

1. Objectives
2. Setup
    A. Installing required libraries
    B. Importing required libraries
    C. Setup LLM
3. Prompt engineering
    A. First basic prompt
    B. Zero-shot prompt
    C. One-shot prompt
    D. Few-shot prompt
    E. Chain-of-thought (CoT) prompting
    F. Self-consistency
4. Applications
    A. Prompt template
    B. Text summarization
    C. Question answering
    D. Text classification
    E. Code generation
    F. Role playing

Exercises

1. Exercise 1: Change parameters for the LLM
2. Exercise 2: Observe how LLM thinks
3. Exercise 3: Revise the text classification agent to one-shot learning

Objectives

After completing this lab, you will be able to:

- **Understand the basics of prompt engineering**: Gain a solid foundation in how to effectively communicate with LLM using prompts, setting the stage for more advanced techniques.

- **Master advanced prompt techniques**: Learn and apply advanced prompt engineering techniques such as few-shot and self-consistent learning to optimize the LLM's response.

- **Utilize LangChain prompt template**: Become proficient in using LangChain's prompt template to structure and optimize your interactions with LLM.

- **Develop practical LLM agents**: Acquire the skills to create and implement agents such as QA bots and text summarization using the Langchain prompt template, translating theoretical knowledge into practical solutions.

Setup

For this lab, you will be using the following libraries:

- `ibm-watsonx-ai` for using LLMs from IBM's watsonx.ai.
- `langchain` for using langchain's different chain and prompt functions.
- `langchain-ibm` provides integration between langchain and ibm-watsonx-ai.

Installing required libraries

The following required libraries are **not** preinstalled in the Skills Network Labs environment. **You must run the following cell** to install them:

Note: The version has been pinned here to specify the version. It's recommended that you do this as well. Even though the library will be updated in the future, the library could still support this lab work.

This might take approximately 1-2 minutes.

`%%capture` has been used to capture the installation, you won't see the output process. But once the installation is done, you will see a number beside the cell.

Note : After installing please ensure that you restart the kernel and execute the subsequent cells.

In [2]:
```
%%capture
!pip install --user "ibm-watsonx-ai==0.2.6"
!pip install --user "langchain==0.1.16"
!pip install --user "langchain-ibm==0.1.4"
```

Importing required libraries

It is recommended that you import all required libraries in one place (here):

In [1]:
```
# You can also use this section to suppress warnings generated by your code:
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
warnings.filterwarnings('ignore')

from ibm_watsonx_ai.foundation_models import Model
from ibm_watsonx_ai.metanames import GenTextParamsMetaNames as GenParams
from ibm_watsonx_ai.foundation_models.utils.enums import ModelTypes
from ibm_watson_machine_learning.foundation_models.extensions.langchain import Wats
```

```
from langchain_core.prompts import PromptTemplate
from langchain.chains import LLMChain
```

Setup LLM

In this section, you will build an LLM model from IBM watsonx.ai.

The following code initializes a Mixtral model on IBM's watsonx.ai platform and wraps it into a function that could allow repeat use.

Some key parameters are explained here:

- `model_id` specifies which model you want to use. There are various model options available; refer to the Foundation Models documentation for more options. In this tutorial, you'll use the `mixtral-8x7b-instruct-v01` model.
- `parameters` define the model's configuration. Set five commonly used parameters for this tutorial. To explore additional commonly used parameters, you can run the code `GenParams().get_example_values()` to see. If no custom parameters are passed to the function, the model will use `default_params`.
- `credentials` and `project_id` are necessary parameters to successfully run LLMs from watsonx.ai. (Keep `credentials` and `project_id` as they are now so that you do not need to create your own keys to run models. This supports you in running the model inside this lab environment. However, if you want to run the model locally, refer to this tutorial for creating your own keys.
- `Model()` is used to wrap the parameters for the model and then call it with `WatsonxLLM()`.

Run the following code, you will initialize a LLM model.

In [2]:
```
def llm_model(prompt_txt, params=None):
    model_id = 'mistralai/mixtral-8x7b-instruct-v01'

    default_params = {
        "max_new_tokens": 256,
        "min_new_tokens": 0,
        "temperature": 0.5,
        "top_p": 0.2,
        "top_k": 1
    }

    if params:
        default_params.update(params)

    parameters = {
        GenParams.MAX_NEW_TOKENS: default_params["max_new_tokens"],  # this control
        GenParams.MIN_NEW_TOKENS: default_params["min_new_tokens"], # this controls
        GenParams.TEMPERATURE: default_params["temperature"], # this randomness or
        GenParams.TOP_P: default_params["top_p"],
        GenParams.TOP_K: default_params["top_k"]
```

```
    }

    credentials = {
        "url": "https://us-south.ml.cloud.ibm.com"
    }

    project_id = "skills-network"

    model = Model(
        model_id=model_id,
        params=parameters,
        credentials=credentials,
        project_id=project_id
    )

    mixtral_llm = WatsonxLLM(model=model)
    response  = mixtral_llm.invoke(prompt_txt)
    return response
```

Let's run the following code to see some other commonly used parameters and their default value.

In [3]:
```
GenParams().get_example_values()
```

Out[3]:
```
{'decoding_method': 'sample',
 'length_penalty': {'decay_factor': 2.5, 'start_index': 5},
 'temperature': 0.5,
 'top_p': 0.2,
 'top_k': 1,
 'random_seed': 33,
 'repetition_penalty': 2,
 'min_new_tokens': 50,
 'max_new_tokens': 200,
 'stop_sequences': ['fail'],
 ' time_limit': 600000,
 'truncate_input_tokens': 200,
 'return_options': {'input_text': True,
  'generated_tokens': True,
  'input_tokens': True,
  'token_logprobs': True,
  'token_ranks': False,
  'top_n_tokens': False}}
```

Prompt engineering

First basic prompt

In this example, let's introduce a basic prompt that utilizes specific parameters to guide the language model's response. You'll then define a simple prompt and retrieve the model's response,

The prompt used is "The wind is". Let the model generate itself.

In [4]:
```
params = {
    "max_new_tokens": 128,
    "min_new_tokens": 10,
    "temperature": 0.5,
    "top_p": 0.2,
    "top_k": 1
}

prompt = "The wind is"

response = llm_model(prompt, params)
print(f"prompt: {prompt}\n")
print(f"response : {response}\n")
```

```
prompt: The wind is

response :  howling outside, and the rain is coming down in sheets. It’s the perfect
weather for a cozy night in with a good book. But what to read? If you’re looking fo
r something to take your mind off the storm, here are five great books to curl up wi
th on a rainy night.

1. The Secret History by Donna Tartt

This gripping novel tells the story of a group of classics students at an elite New
England college who become embroiled in a murder conspiracy. With its richly drawn c
haracters and atmospheric setting, The Secret History is the perfect
```

As you can see from the response, the model continues generating content following the initial prompt, "The wind is". You might notice that the response appears truncated or incomplete. This is because you have set the `max_new_tokens,` which restricts the number of tokens the model can generate.

Try to adjust the parameters and observe the difference in the response.

Zero-shot prompt

Here is an example of a zero-shot prompt.

Zero-shot learning is crucial for testing a model's ability to apply its pre-trained knowledge to new, unseen tasks without additional training. This capability is valuable for gauging the model's generalization skills.

In this example, let's demonstrate a zero-shot learning scenario using a prompt that asks the model to classify a statement without any prior specific training on similar tasks. The prompt

requests the model to assess the truthfulness of the statement: "The Eiffel Tower is located in Berlin.". After defining the prompt, you'll execute it with default parameters and print the response.

This approach helps you understand how well the model can handle direct questions based on its underlying knowledge and reasoning abilities.

Try running the prompt to see the model's capacity to correctly analyze and respond to factual inaccuracies.

In [5]:
```
prompt = """Classify the following statement as true or false:
            'The Eiffel Tower is located in Berlin.'

            Answer:
"""
response = llm_model(prompt, params)
print(f"prompt: {prompt}\n")
print(f"response : {response}\n")
```

```
prompt: Classify the following statement as true or false:
            'The Eiffel Tower is located in Berlin.'

            Answer:


response :             False. The Eiffel Tower is located in Paris, France, not Berl
in, Germany.
```

The model responds with the 'False' answer, which is correct. It also gives the reason for it.

One-shot prompt

Here is a one-shot learning example where the model is given a single example to help guide its translation from English to French.

The prompt provides a sample translation pairing, "How is the weather today?" translated to "Comment est le temps aujourd'hui?" This example serves as a guide for the model to understand the task context and desired format. The model is then tasked with translating a new sentence, "Where is the nearest supermarket?" without further guidance.

In [6]:
```
params = {
    "max_new_tokens": 20,
    "temperature": 0.1,
}

prompt = """Here is an example of translating a sentence from English to French:

            English: “How is the weather today?”
            French: “Comment est le temps aujourd'hui?”
```

```
            Now, translate the following sentence from English to French:

            English: “Where is the nearest supermarket?”

"""
response = llm_model(prompt, params)
print(f"prompt: {prompt}\n")
print(f"response : {response}\n")
```

```
prompt: Here is an example of translating a sentence from English to French:

            English: “How is the weather today?”
            French: “Comment est le temps aujourd'hui?”

            Now, translate the following sentence from English to French:

            English: “Where is the nearest supermarket?”



response :              French: “Où est le supermarché le plus proche?”
```

The model's response shows how it applies the structure and context provided by the initial example to translate the new sentence.

Consider experimenting with different sentences or adjusting the parameters to see how these changes impact the model's translations.

Few-shot prompt

Here is an example of few-shot learning by classifying emotions from text statements.

Let's provide the model with three examples, each labeled with an appropriate emotion—joy, frustration, and sadness—to establish a pattern or guideline on how to categorize emotions in statements.

After presenting these examples, let's challenge the model with a new statement: "That movie was so scary I had to cover my eyes." The task for the model is to classify the emotion expressed in this new statement based on the learning from the provided examples.

In [7]:
```
 #parameters  `max_new_tokens` to 10, which constrains the model to generate brief

params = {
    "max_new_tokens": 10,
}

prompt = """Here are few examples of classifying emotions in statements:

            Statement: 'I just won my first marathon!'
```

```
            Emotion: Joy

            Statement: 'I can't believe I lost my keys again.'
            Emotion: Frustration

            Statement: 'My best friend is moving to another country.'
            Emotion: Sadness

            Now, classify the emotion in the following statement:
            Statement: 'That movie was so scary I had to cover my eyes.’


"""
response = llm_model(prompt, params)
print(f"prompt: {prompt}\n")
print(f"response : {response}\n")
```

```
prompt: Here are few examples of classifying emotions in statements:

           Statement: 'I just won my first marathon!'
           Emotion: Joy

           Statement: 'I can't believe I lost my keys again.'
           Emotion: Frustration

           Statement: 'My best friend is moving to another country.'
           Emotion: Sadness

           Now, classify the emotion in the following statement:
           Statement: 'That movie was so scary I had to cover my eyes.’



response : Answer: Fear. The person is express
```

The parameters are set with `max_new_tokens` to 10, which constrains the model to generate brief responses, focusing on the essential output without elaboration.

The model's response demonstrates its ability to use the provided few examples to understand and classify the emotion of the new statement effectively following the same pattern in examples.

Chain-of-thought (CoT) prompting

Here is an example of the Chain-of-Thought (CoT) prompting technique, designed to guide the model through a sequence of reasoning steps to solve a problem. In this example, the problem is a simple arithmetic question: “A store had 22 apples. They sold 15 apples today and received a new delivery of 8 apples. How many apples are there now?”

The CoT technique involves structuring the prompt by instructing the model to “Break down each step of your calculation.” This encourages the model to include explicit reasoning steps, mimicking human-like problem-solving processes.

In [8]:
```
params = {
    "max_new_tokens": 512,
    "temperature": 0.5,
}

prompt = """Consider the problem: 'A store had 22 apples. They sold 15 apples today
            How many apples are there now?’

            Break down each step of your calculation

"""
response = llm_model(prompt, params)
print(f"prompt: {prompt}\n")
print(f"response : {response}\n")
```

```
prompt: Consider the problem: 'A store had 22 apples. They sold 15 apples today and
got a new delivery of 8 apples.
            How many apples are there now?’

            Break down each step of your calculation



response :             Step 1: Add the number of apples the store had at the beginni
ng of the day to the number of apples they got in the new delivery.

            22 apples (initial amount) + 8 apples (new delivery) = 30 apples

            Step 2: Subtract the number of apples the store sold today from the tota
l number of apples after the new delivery.

            30 apples (total after new delivery) - 15 apples (sold) = 15 apples

            Answer: \boxed{15}.
```

From the response of the model, you can see the prompt directs the model to:

1. Add the initial number of apples to the apples received in the new delivery.
2. Subtract the number of apples sold from the sum obtained in the first step.

By breaking down the problem into specific steps, the model is better able to understand the sequence of operations required to arrive at the correct answer.

Self-consistency

This example demonstrates the self-consistency technique in reasoning through multiple calculations for a single problem. The problem posed is: “When I was 6, my sister was half

my age. Now I am 70, what age is my sister?"

The prompt instructs, "Provide three independent calculations and explanations, then determine the most consistent result." This encourages the model to engage in critical thinking and consistency checking, which are vital for complex decision-making processes.

In [9]:
```
params = {
    "max_new_tokens": 512,
}

prompt = """When I was 6, my sister was half of my age. Now I am 70, what age is my

            Provide three independent calculations and explanations, then determine

"""
response = llm_model(prompt, params)
print(f"prompt: {prompt}\n")
print(f"response : {response}\n")
```

prompt: When I was 6, my sister was half of my age. Now I am 70, what age is my sister?

Provide three independent calculations and explanations, then determine the most consistent result.

response : Calculation 1:
When my sister was half my age, I was twice as old as she was. Let's call her current age "S" and my current age "M". At that time, M = 2S. Now, M = 70, so we can solve for S:

70 = 2S
S = 35

Calculation 2:
We could also consider this a rate problem. The difference in our ages has always been a constant factor, even though the actual ages have changed. When I was 6, my sister was 3 (half my age). The ratio between our ages then was 6:3 or 2:1. That ratio still holds true today, so we can set up a proportion:

6/3 = 70/S

Solving for S gives us:

S = 35

Calculation 3:
Another way to approach this is by considering how much older I am than my sister now compared to when I was 6. I am 70 - 35 = 35 years older than my sister now. When I was 6, I was only 6 - 3 = 3 years older than my sister. The difference in our ages increased by 32 years (35 - 3) over this time period. If we divide the current age difference by the increase in the age difference, we get the fraction of the total increase that has occurred since I was 6:

35 / 32 ≈ 1.09375

Multiplying my sister's age at the time (3) by this fraction gives us her current age:

S ≈ 3 * 1.09375 ≈ 3.28125

Since my sister's age must be a whole number, rounding up gives us S = 36.

Ranking of calculations from best to worst:
1. Calculation 1: This answer is most likely correct because it uses simple algebra based on the information provided.
2. Calculation 2: This answer is also likely correct because it uses proportional reasoning, which is a valid mathematical concept.
3. Calculation 3: This answer is least likely correct because it involves rounding and approximation, which may introduce

The model's response shows that it provides three different calculations and explanations. Each calculation attempts to derive the sister's age using different logical approaches.

Self-consistency can help identify the most accurate and reliable answer in scenarios where multiple plausible solutions exist.

Applications

In this section, you will show you how to use the prompt template from Langchain to create more structured and reproducible prompts. You will also learn to create some applications based on the prompt template.

Prompt template

Prompt template is a key concept in langchain, it helps to translate user input and parameters into instructions for a language model. This can be used to guide a model's response, helping it understand the context and generate relevant and coherent language-based output.

To use the prompt template, you need to initialize a LLM first.

You can still use the `mixtral-8x7b-instruct-v01` from watsonx.ai.

In [10]:
```
model_id = 'mistralai/mixtral-8x7b-instruct-v01'

parameters = {
    GenParams.MAX_NEW_TOKENS: 256,  # this controls the maximum number of tokens in
    GenParams.TEMPERATURE: 0.5, # this randomness or creativity of the model's resp
}

credentials = {
    "url": "https://us-south.ml.cloud.ibm.com"
}

project_id = "skills-network"

model = Model(
    model_id=model_id,
    params=parameters,
    credentials=credentials,
    project_id=project_id
)

mixtral_llm = WatsonxLLM(model=model)
mixtral_llm
```

Out[10]:
```
WatsonxLLM(model=<ibm_watsonx_ai.foundation_models.model.Model object at 0x7fd5b85
9aea0>)
```

Use the `PromptTemplate` to create a template for a string-based prompt. In this template, you'll define two parameters: `adjective` and `content`. These parameters allow for the

reuse of the prompt across different situations. For instance, to adapt the prompt to various contexts, simply pass the relevant values to these parameters.

In [11]:
```
template = """Tell me a {adjective} joke about {content}.
"""
prompt = PromptTemplate.from_template(template)
prompt
```

Out[11]:
```
PromptTemplate(input_variables=['adjective', 'content'], template='Tell me a {adje
ctive} joke about {content}.\n')
```

Now, let's take a look at how the prompt has been formatted.

In [12]:
```
prompt.format(adjective="funny", content="chickens")
```

Out[12]:
```
'Tell me a funny joke about chickens.\n'
```

From the response, you can see that the prompt is formatted according to the specified context.

The following code will wrap the formatted prompt into the LLMChain, and then invoke the prompt to get the response from the LLM.

In [13]:
```
llm_chain = LLMChain(prompt=prompt, llm=mixtral_llm)
response = llm_chain.invoke(input = {"adjective": "funny", "content": "chickens"})
print(response["text"])
```

```
Why did the chicken cross the playground? To get to the other slide.
```

From the response, you can see the LLM came up with a funny joke about chickens.

To use this prompt in another context, simply replace the variables accordingly

In [14]:
```
response = llm_chain.invoke(input = {"adjective": "sad", "content": "fish"})
print(response["text"])
```

```
I just found out my favorite fish died... I'm koi.

Hey, it's a start.

(via reddit)

- The best part about getting a new cat is that they always come with a built-in vac
uum cleaner.

(via reddit)

- A man was in court being tried for murder.

The judge asked, "Did you kill the victim?"

The defendant replied, "No, I just hit him with my car."

The judge asked, "Did you intend to kill him?"

The defendant replied, "No, I just wanted to get him out of the road."

The judge asked, "But didn't you realize that a car could kill someone?"

The defendant replied, "I thought a car could only injure someone."

The judge asked, "What happened next?"

The defendant replied, "I pulled over and called 911."

The judge asked, "What did you say?"

The defendant replied, "I told the operator that I had injured someone with my car."

The judge asked, "Did the operator ask if the victim
```

In the following sections, you will learn how to create agents capable of completing various tasks using prompt templates.

Text summarization

Here is a text summarization agent designed to help summarize the content you provide to the LLM.

You can store the content to be summarized in a variable, allowing for repeated use of the prompt.

In [15]:
```
content = """
        The rapid advancement of technology in the 21st century has transformed var
        Innovations such as artificial intelligence, machine learning, and the Inte
        For instance, AI-powered diagnostic tools are improving the accuracy and sp
        Moreover, online learning platforms are making education more accessible to
        These technological developments are not only enhancing productivity but al
"""
```

```
template = """Summarize the {content} in one sentence.
"""
prompt = PromptTemplate.from_template(template)

llm_chain = LLMChain(prompt=prompt, llm=mixtral_llm)
response = llm_chain.invoke(input = {"content": content})
print(response["text"])
```

```
The rapid advancement of technology in the 21st century has transformed various indu
stries, including healthcare, education, and transportation, by enhancing productivi
ty and contributing to a more interconnected and informed society through innovation
s such as artificial intelligence, machine learning, and the Internet of Things.
```

Question answering

Here is a Q&A agent.

This agent enables the LLM to learn from the provided content and answer questions based on what it has learned. Occasionally, if the LLM does not have sufficient information, it might generate a speculative answer. To manage this, you'll specifically instruct it to respond with "Unsure about the answer" if it is uncertain about the correct response.

In [16]:
```
content = """
        The solar system consists of the Sun, eight planets, their moons, dwarf pla
        The inner planets—Mercury, Venus, Earth, and Mars—are rocky and solid.
        The outer planets—Jupiter, Saturn, Uranus, and Neptune—are much larger and
"""

question = "Which planets in the solar system are rocky and solid?"

template = """
            Answer the {question} based on the {content}.
            Respond "Unsure about answer" if not sure about the answer.

            Answer:

"""
prompt = PromptTemplate.from_template(template)
output_key = "answer"

llm_chain = LLMChain(prompt=prompt, llm=mixtral_llm, output_key=output_key)
response = llm_chain.invoke(input = {"question":question ,"content": content})
print(response["answer"])
```

```
The inner planets—Mercury, Venus, Earth, and Mars—are rocky and solid.
```

Text classification

Here is a text classification agent designed to categorize text into predefined categories. This example employs zero-shot learning, where the agent classifies text without prior exposure to related examples.

Can you revise it to the one-shot learning or few-shot learning in the exercises?

In [17]:
```
text = """
        The concert last night was an exhilarating experience with outstanding perf
"""

categories = "Entertainment, Food and Dining, Technology, Literature, Music."

template = """
            Classify the {text} into one of the {categories}.

            Category:

"""
prompt = PromptTemplate.from_template(template)
output_key = "category"

llm_chain = LLMChain(prompt=prompt, llm=mixtral_llm, output_key=output_key)
response = llm_chain.invoke(input = {"text":text ,"categories": categories})
print(response["category"])
```

Music

Code generation

Here is an example of an SQL code generation agent. This agent is designed to generate SQL queries based on given descriptions. It interprets the requirements from your input and translates them into executable SQL code.

In [18]:
```
description = """
        Retrieve the names and email addresses of all customers from the 'customers
        The table 'purchases' contains a column 'purchase_date'
"""

template = """
            Generate an SQL query based on the {description}

            SQL Query:

"""
prompt = PromptTemplate.from_template(template)
output_key = "query"

llm_chain = LLMChain(prompt=prompt, llm=mixtral_llm, output_key=output_key)
response = llm_chain.invoke(input = {"description":description})
print(response["query"])
```

```
SELECT customers.name, customers.email
FROM customers
INNER JOIN purchases
ON customers.customer_id = purchases.customer_id
WHERE purchases.purchase_date >= (CURRENT_DATE - INTERVAL '30 day');
```

Role playing

You can also configure the LLM to assume specific roles as defined by us, enabling it to follow predetermined rules and behave like a task-oriented chatbot.

For example, the code below configures the LLM to act as a game master. In this role, the LLM answers questions about games while maintaining an engaging and immersive tone, enhancing the user experience.

Run the following code to create the prompt template and create a LLMChian to wrap the prompt.

In [19]:
```
role = """
        game master
"""

tone = "engaging and immersive"

template = """
            You are an expert {role}. I have this question {question}. I would like

            Answer:

"""
prompt = PromptTemplate.from_template(template)
output_key = "answer"

llm_chain = LLMChain(prompt=prompt, llm=mixtral_llm, output_key=output_key)
```

The following code will create a game master chatbot that takes your questions as input and provides responses from the model.

Run the code below to launch the bot.

You can test the bot by asking the question, "Who are you?" The bot will respond with "I am a game master," indicating it has assumed the role that you predefined.

The function is written within a while loop, allowing continuous interaction. To exit the loop and terminate the conversation, type "quit," "exit," or "bye" into the input box.

In [20]:
```
while True:
    query = input("Question: ")

    if query.lower() in ["quit","exit","bye"]:
        print("Answer: Goodbye!")
        break

    response = llm_chain.invoke(input = {"role": role, "question": query, "tone": t

    print("Answer: ", response["answer"])
```

```
Answer:              I am the enigmatic and all-knowing game master, a being of pure
intellect and boundless creativity. I weave the tapestry of your adventures, breathi
ng life into the worlds you explore and the characters you encounter. I am here to g
uide you through thrilling challenges and thought-provoking puzzles, ensuring that e
very moment is filled with excitement and intrigue. So, are you ready to embark on a
n unforgettable journey?
```

```
Answer:              Of course! I'm here to help make your gaming experience the bes
t it can be. Let's dive into your question and explore the depths of the game world
together! What's your question today?
Answer:              Welcome, dear adventurer, to the mystical realm of Eldoria. A l
and where magic and mystery intertwine, and where heroes are forged in the heat of b
attle. Eldoria is a land of towering mountains, vast forests, and ancient ruins, fil
led with mythical creatures and powerful artifacts. It is a place where every decisi
on you make can change the course of history, and where the fate of the realm rests
in your hands. So, gather your courage, sharpen your skills, and prepare to embark o
n an epic journey through the enchanting world of Eldoria. What quest would you like
to undertake first, brave adventurer?
Answer: Goodbye!
```

Great! You finish the lab. Now let's take some exercises.

Exercises

Exercise 1: Change parameters for the LLM

Experiment with changing the parameters of the LLM to observe how different settings impact the responses. Adjusting parameters such as `max_new_tokens`, `temperature`, or `top_p` can significantly alter the behavior of the model. Try different configurations to see how each variation influences the output.

In [ ]:
```
# TODO
```

▶ Click here for Solution

Exercise 2: Observe how LLM thinks

You can set `verbose=True` in the `LLMChain()` to observe the thought process of the LLM, gaining insights into how it formulates its responses. Can you make it any agent you created before to observe it?

In [ ]:
```
# TODO
```

▶ Click here for Solution

Exercise 3: Revise the text classification agent to one-shot learning

You were using zero-shot learning when you created the text classification agent. Can you revise it to use one-shot learning?

In [ ]:
```
# TODO
```

▶ Click here for Solution

Authors

Kang Wang

Kang Wang is a Data Scientist in IBM. He is also a PhD Candidate in the University of Waterloo.

© Copyright IBM Corporation. All rights reserved.