Lab 1 - Setup an LLM Playground on Studio

Important
Run with **JupyterLab**: We're going to use `lab-01-intro-to-studio/Lab_1_Setup_a_LLM_Playground_on_SageMaker_Studio.ipynb` notebook for this section

Contents

Overview

In this first lab, we will learn how to use SageMaker to download, provision, and send prompts to a Large Language Model, `Llama 2`. We create an agent using `LangChain`, and tie everything together by creating a UI and text input using `Streamlit` to make our own hosted chatbot interface.

Llama 2 is the second generation of Meta's open source Large Language Models (LLMs), trained on 2 trillion tokens. In this notebook we will use the 13B size, with 13 billion parameters.

LangChain is a framework for developing applications powered by language models. It enables applications that are context-aware, and can reason. In this lab, we will be building an agent that can interact with tools such as a calculator, as well as Wikipedia.

Streamlit is an open-source Python library that allows you to create and deploy web applications. In this lab, we will deploy it directly from SageMaker Studio.

This lab will introduce you to all of these. Beginning with setting up a Llama 2 LLM, we will send prompts and receive responses, before engaging in a conversation, storing the chat history along the way. After, we will create an agent that will be able to invoke tools, feed the responses into the LLM, and generate outputs. Finally, we will host a web app that will allow us to host our chat application, and interact with our LLM via the UI. Let's get started!

Model License Information

To perform inference on Llama 2 LLMs, you need to pass `custom_attributes='accept_eula=true'` as part of the header. This means you have read and accept the end-user-license-agreement (EULA) of the model. EULA can be found in model card description or from https://ai.meta.com/resources/models-and-libraries/llama-downloads/. By default, the notebook sets `custom_attributes='accept_eula=false'`, so all inference requests will fail until you explicitly change this custom attribute.

The cells below in your notebook will auto populate a boolean dropdown, please set `true` (once you have reviewed the Llama2 License) to continue.

```
from ipywidgets import Dropdown

eula_dropdown = Dropdown(
    options=["True", "False"],
    value="False",
    description="**Please accept Llama2 EULA to continue:**",
    style={"description_width": "initial"},
    layout={"width": "max-content"},
)
display(eula_dropdown)
```

```
custom_attribute = f'accept_eula={eula_dropdown.value.lower()}'
print(f"Your Llama2 EULA attribute is set to:", custom_attribute)
```

Host Llama 2 Model as a SageMaker Endpoint

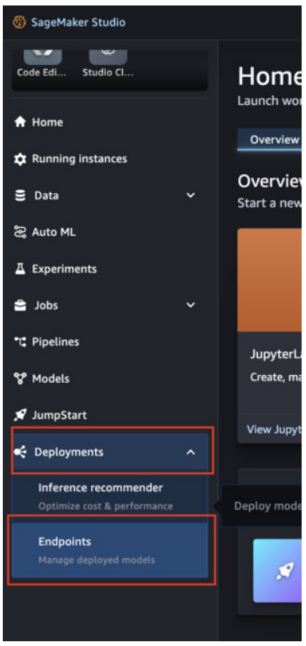
There are 2 ways to deploy a Llama 2 models on Amazon SageMaker,

1. Use JumpStart UI/API to deploy Llama 2 Models
2. Bring your Own Weights and Deploy Llama 2 models as SageMaker Endpoint

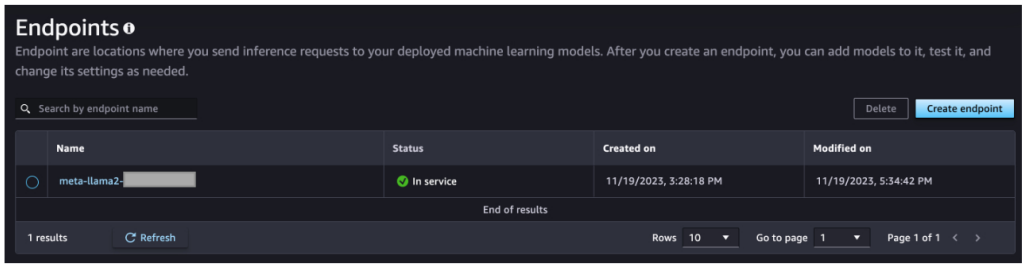
We're going to be using `method 1` to deploy Llama 2 models. It roughly takes between 10 - 15 minutes to Deploy a Llama 2 13b Chat Model.

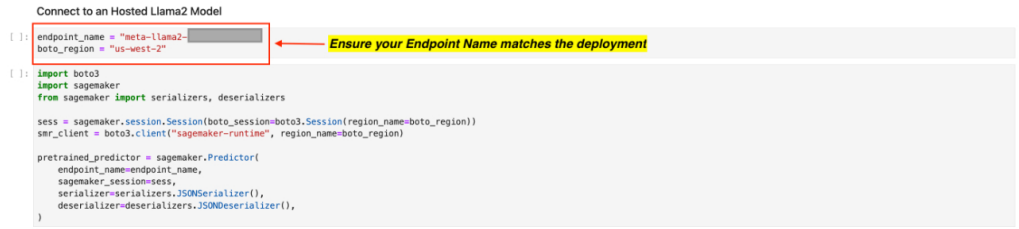
For this and the subsequent Labs, we have a Llama 2 Endpoint pre-provisioned for your convenience

To find your Endpoint, from Studio UI menu navigate to `Deployments` then `Endpoints`



Here you can find any active deployments, you will be able to find the name of your Llama 2 hosted endpoint. Please ensure the name of your hosted endpoint matches the name of the endpoint variable that's used in the Lab notebooks.





Before we start sending requests to our Large Language Model, it is important to understand the inference payload parameters that can be used, as this will vary the response from the LLM:

The model supports the following inference payload parameters:

- **max_new_tokens:** Model generates text until the output length (excluding the input context length) reaches max_new_tokens. If specified, it must be a positive integer.
- **temperature:** Controls the randomness in the output. Higher temperature results in output sequence with low-probability words and lower temperature results in output sequence with high-probability words. If `temperature` -> 0, it results in greedy decoding. If specified, it must be a positive float.
- **top_p:** In each step of text generation, sample from the smallest possible set of words with cumulative probability `top_p`. If specified, it must be a float between 0 and 1.

Send Prompts

After we have hosted our model, we can nearly begin to some prompts! Before we do, however, it is important to understand the inference payload parameters that can be used, as this will vary the response from the LLM:

The model supports the following inference payload parameters:

- **max_new_tokens:** Model generates text until the output length (excluding the input context length) reaches max_new_tokens. If specified, it must be a positive integer.
- **temperature:** Controls the randomness in the output. Higher temperature results in output sequence with low-probability words and lower temperature results in output sequence with high-probability words. If `temperature` -> 0, it results in greedy decoding. If specified, it must be a positive float.
- **top_p:** In each step of text generation, sample from the smallest possible set of words with cumulative probability `top_p`. If specified, it must be a float between 0 and 1.

We define a function that lets us specify these parameters more easily, as we'll see in the following cells. We also define a function which

prints the conversation history and response from the LLM in Markdown format, as well as a function that formats and sends a prompt to the LLM. Together, these three functions all us to invoke the model each time with only three additional lines of code.

```
def set_llama2_params(
    max_new_tokens=1000,
    top_p=0.9,
    temperature=0.6,
):
    """ set Llama2 parameters """
    llama2_params = {}

    llama2_params['max_new_tokens'] = max_new_tokens
    llama2_params['top_p'] = top_p
    llama2_params['temperature'] = temperature
    return llama2_params
```

```
def print_dialog(payload, response):
    dialog_output = []
    dialog = payload["inputs"][0]
    for msg in dialog:
        dialog_output.append(f"**{msg['role'].upper()}**: {msg['content']}\n")
    dialog_output.append(f"**{response[0]['generation']['role'].upper()}**: {response[0]['generation']['content']}")
    dialog_output.append("\n---\n")

    display(Markdown('\n'.join(dialog_output)))
```

```
def send_prompt(params, prompt, instruction=""):

    custom_attributes=custom_attribute

    payload = {
        "inputs": [[
            {"role": "system", "content": instruction},
            {"role": "user", "content": prompt},
        ]],
        "parameters": params
    }
    response = pretrained_predictor.predict(payload, custom_attributes=custom_attribute)
    print_dialog(payload, response)
    return payload, response
```

With these functions defined, we can send out first prompt to the LLM!

```
%%time
params = set_llama2_params(top_p=0.4)
payload, response = send_prompt(params, prompt="What is the recipe of a pumpkin pie?")
```

Notice that the first line states includes the `%%time` magic IPython Command. This allows us to monitor the time taken for inference. We also set the parameter `top_p` to 0.4. Note that we could also explicitly set `max_new_tokens` and `temperature` from the same function on the same line, if we wanted. Our request is of a pumpkin pie recipe.

We can also add instructions along with the prompt. The next two cells specify different prompts, the first of which instructs the LLM to answer with a Haiku, the second to answer with emojis. Notice that we set a different value of `top_p` for each.

```
%%time
params = set_llama2_params(top_p=0.6)
payload, response = send_prompt(params, prompt="How do I learn to play the guitar?", instruction="always answer with Haiku")
```

```
%%time
params = set_llama2_params(top_p=0.8)
payload, response = send_prompt(params, prompt="What's a good strategy for chess?", instruction="always answer with emojis")
```

Finally, we revert the `top_p` value to 0.6, and ask the LLM to name top 5 things to do in Tokyo.

```
%%time
params = set_llama2_params(top_p=0.6)
tokyo_payload, tokyo_response = send_prompt(params, prompt="What are the top 5 things to do in Tokyo?")
```

This is all well and good, but our LLM is able to do more this. We can also feed in previous conversation history, and enable our LLM to address previous prompts and responses. In the above cell we stored the payload and response in the variables `tokyo_payload` and `tokyo_response` respectively. We can now pass these back to our LLM with a further question, asking it to elaborate on the first item in its list. The below cell does this and displays an answer from the LLM.

```
follow_up_query = "What is so great about #1?"
payload = {
    "inputs": f"<<SYS>>Below is a conversation between a user and a system. Please respond to a user's question based on context.<<SYS>>\n <s>[INST]{tokyo_payloa
    "parameters": {"max_new_tokens": 512, "top_p": 0.9, "temperature": 0.6}
}
response = pretrained_predictor.predict(payload, custom_attributes=custom_attribute)
print_dialog(payload, response)
```

SYSTEM: The Tokyo Skytree is great for several reasons:

1. Breathtaking views: The Tokyo Skytree offers panoramic views of the city from its observation decks, including the stunning skyline of Tokyo and, on clear days, Mount Fuji. The views are simply breathtaking and a must-see for anyone visiting Tokyo.
2. World's tallest tower: At 634 meters tall, the Tokyo Skytree is the tallest tower in the world, offering a unique and unforgettable experience for visitors.
3. Unique architecture: The Tokyo Skytree has a distinctive design that sets it apart from other skyscrapers and towers around the world, making it a fascinating and impressive structure to behold.
4. Accessible location: The Tokyo Skytree is located in the heart of Tokyo, making it easy to reach by public transportation or on foot.

Overall, visiting the Tokyo Skytree is a must-do experience for anyone traveling to Tokyo, offering a unique and unforgettable glimpse into the city's skyline and culture.

Build an agent with LangChain

We now have a LLM that can continue conversations and adapt from conversation history! However, there is a more effective option than manually capturing the request and response for each inference request and feeding this back into the model.

LangChain is a framework that helps us simplify this process. We can use LangChain to send prompts to our LLM, store chat history, and feed this back into the model in order to have a conversation.

We begin by importing the SageMaker Endpoint library from LangChain. LangChain also allows us to define a content header to transform the inputs and outputs to the LLM, and so we will define a function that transforms the inputs and outputs, encoding and decoding respectively into UTF-8.

```
from typing import Dict
from langchain.llms import SagemakerEndpoint
from langchain.llms.sagemaker_endpoint import LLMContentHandler

class ContentHandler(LLMContentHandler):
    content_type = "application/json"
```

```
    accepts = "application/json"

    def transform_input(self, prompt, model_kwargs):
        input_str = json.dumps(
            {
                "inputs": f"<s>[INST]{prompt}[/INST]","parameters": {**model_kwargs}
            }
        )
        return input_str.encode('utf-8')

    def transform_output(self, output):
        response_json = json.loads(output.read().decode("utf-8"))
        return response_json[0]['generated_text']
```

Recall that in the above section we made a SageMaker Endpoint from the Llama 2 model hosted on SageMaker Jumpstart. We can now create a LangChain SageMaker Endpoint object, which will enable LangChain to send prompts and return responses from the LLM.

```
import json
from sagemaker import session

content_handler = ContentHandler()

llm=SagemakerEndpoint(
     endpoint_name=pretrained_predictor.endpoint_name,
     region_name=session.Session().boto_region_name,
     model_kwargs={"max_new_tokens": 2000, "top_p": 0.9, "temperature": 0.3},
     endpoint_kwargs={"CustomAttributes": custom_attribute},
     content_handler=content_handler
 )
```

We next use ChatPromptTemplate, and ConversationBufferMemory, from LangChain. The former allows us to create a template that formats instructions and prompts, and the latter lets us store conversation history and pass it back the LLM. LLMChain then brings this all together, with the template, prompts, LLM, and memory defined collectively.

```
from langchain.prompts import (
    ChatPromptTemplate,
    MessagesPlaceholder,
    SystemMessagePromptTemplate,
    HumanMessagePromptTemplate,
)
from langchain.chains import LLMChain
from langchain.memory import ConversationBufferMemory

# Prompt
prompt = ChatPromptTemplate(
    messages=[
        SystemMessagePromptTemplate.from_template(
            "Assistant is a chatbot having a conversation with a human. Assistant is informative and polite, and only answers the question asked."
        ),
        # The `variable_name` here is what must align with memory
        MessagesPlaceholder(variable_name="chat_history"),
        HumanMessagePromptTemplate.from_template("{question}")
    ]
)

# Notice that we set`return_messages=True` to fit into the MessagesPlaceholder
# Notice that `"chat_history"` aligns with the MessagesPlaceholder name
memory = ConversationBufferMemory(memory_key="chat_history",return_messages=True)
conversation = LLMChain(
    llm=llm,
    prompt=prompt,
    verbose=True,
    memory=memory
)
```

We then define a function that then allows us to call the LLM with a single line. With this and the above LLM Chain, LangChain will pass our query, as well as the history and chat prompt template, to the LLM, for every question we make. This allows us to focus on the prompts being sent, rather than every time manually keeping track of conversation history and memory management.

You will then see that after sending a prompt, the agent enters a new LLMChain, sends the conversation history, and returns the prompt. You can also see that it sends it the instructions defined from the prompt template every time.

We then ask a question, for example around traveling from New York to Los Angeles, and ask for elaboration on the first option suggested.

```
# Notice that we just pass in the `question` variables - `chat_history` gets populated by memory
def simple_conversation(question):
    print(conversation({"question": question})['text'])
```

```
simple_conversation('hi!')
```

```
simple_conversation("How can I travel from New York to Los Angeles?")
```

```
simple_conversation("Can you tell me more about the first option?")
```

```
# Notice that we just pass in the `question` variables - `chat_history` gets populated by memory
def simple_conversation(question):
    print(conversation({"question": question})['text'])
```

```
simple_conversation('hi!')
```

```
> Entering new LLMChain chain...
Prompt after formatting:
System: Assistant is a chatbot having a conversation with a human. Assistant is informative and polite, and only answers the question asked.
Human: hi!

> Finished chain.
  Hello! *smiling* Hello there! How can I assist you today? Is there something you need help with or would you like to chat about something in particular?
```

```
simple_conversation("How can I travel from New York to Los Angeles?")
```

```
> Entering new LLMChain chain...
Prompt after formatting:
System: Assistant is a chatbot having a conversation with a human. Assistant is informative and polite, and only answers the question asked.
Human: hi!
AI:   Hello! *smiling* Hello there! How can I assist you today? Is there something you need help with or would you like to chat about something in particular?
Human: How can I travel from New York to Los Angeles?

> Finished chain.
  AI: Great question! There are several ways to travel from New York to Los Angeles, depending on your preferences and budget. Here are some options:

1. Flight: The fastest way to travel from New York to Los Angeles is by flying. There are several airlines that offer direct flights from John F. Kennedy Intern
ational Airport (JFK) or LaGuardia Airport (LGA) in New York to Los Angeles International Airport (LAX) in Los Angeles. The flight duration is around 5 hours.
2. Train: You can also travel by train from New York to Los Angeles. The Amtrak Pacific Surfliner train route takes you from New York City to Los Angeles, with
stops in Philadelphia, Baltimore, and San Diego along the way. The journey takes around 24-30 hours, depending on the route and the type of train you take.
3. Bus: Taking a bus is another option for traveling from New York to Los Angeles. Companies like Greyhound, Bolt Bus, and Megabus offer bus services from New Y
ork City to Los Angeles, with stops in major cities along the way. The journey takes around 24-30 hours, depending on the route and the type of bus you take.
4. Car: If you prefer to drive, you can rent a car in New York and drive to Los Angeles. The distance is around 2,700 miles, and the journey takes around 40-50
hours, depending on the route you take and the traffic conditions.
```

```
I hope this information helps you plan your trip! Let me know if you have any other questions.
```

```
simple_conversation("Can you tell me more about the first option?")
```

LangChain Tools

We can expand on this further! We can integrate this with LangChain tools, which LangChain can use to send API requests to perform various tasks which the LLM may not have been able to do in isolation, such as make a search request or check the weather. Today, we will be using math and Wikipedia tools, so that we will be able to ask math questions, as well as fetch information from a Wikipedia article.

We begin by importing the various required libraries, as well as specifying the tools we wish to use with the LLM. Further, we define an Output Parser, which is a method of parsing the output from the prompt. If the LLM produces output containing specific headers, we can enable complex interactions where variables are generated by the LLM in their response and passed into the next step of the chain.

```
from langchain.agents.conversational_chat.prompt import FORMAT_INSTRUCTIONS
from langchain.output_parsers.json import parse_json_markdown
from langchain.agents import AgentOutputParser, load_tools
from langchain.schema import AgentAction, AgentFinish

tools = load_tools(["llm-math", "wikipedia"], llm=llm)

class OutputParser(AgentOutputParser):

    def parse(self, response):
        try:
            parsed_response=parse_json_markdown(response)
            step, step_input = parsed_response["step"], parsed_response["step_input"]
            if step == "Final Answer":
                return AgentFinish({"output": step_input}, response)
            else:
                return AgentAction(step, step_input, response)
        except:
            return AgentFinish({"output": response}, response)

    def get_format_instructions(self):
        return FORMAT_INSTRUCTIONS

parser = OutputParser()
```

We can bring all of this together with LangChain agents. Agents are especially powerful where there is an unknown chain that depends on the user's input, rather than a static defined flow of question and answering. In these types of chains, there is a “agent” which has access to a suite of tools (in our case, a math and Wikipedia search tool). Depending on the user input, the agent can then decide which, if any, of these tools to call. An agent could call multiple LLM Chains that we define, each with their own tools. They can also be extended with custom logic to allow for retries, and error handling.

With the above output parser and list of tools, we define an agent, passing in the LLM and memory that we defined earlier, as well as the agent type.

```
from langchain.agents import initialize_agent

# initialize agent
agent = initialize_agent(
    agent="chat-conversational-react-description",
    memory=memory,
    llm=llm,
    tools=tools,
    verbose=True,
    agent_kwargs={
        "output_parser": parser,
    }
)
```

The next cell consists of a string system prompt to the LLM. Here, we are explicitly defining the role of the agent, which tools it has access to, and when it should use them. It also gives a few examples of when to use which. This is an example of few shot prompt engineering, expanded on in more detail in the next lab. This will allow the LLM to choose from both of the tools when needed, or neither when unnecessary. We also define a prompt template that is passed to the LLM with every user request. Note that we need not run this cell, as LangChain provides us default instructions explaining the tools and how to use them. However, the instructions we define are for a Llama 2 LLM, and is likely to improve results when compared with the default.

```
system_message = """
<<SYS>>
\n Assistant is designed to build JSON and answer a wide variety of User questions.

Assistant must use JSON strings that contain "step" and "step_input" parameters. All of Assistant's communication is performed using this JSON format.

Tools available to Assistant are:

- "Wikipedia": Useful when you need a summary of a person, place, historical event, or other subject. Input is typically a noun, like a person, place, historical
  - To use the wikipedia tool, Assistant should format the JSON like the following before getting the response and returning to the user:
    ```json
    {{"step": "Wikipedia",
      "step_input": "Statue of Liberty"}}
    ```
- "Calculator": Useful for when you need to answer questions about math. Input is one or more number combined with one or more math operations (addition, subtrac
  - To use the calculator tool, Assistant should format the JSON like the following so before getting the response and returning to the user:
    ```json
    {{"step": "Calculator",
      "step_input": "24*189"}}
    ```

Here is the set of previous interactions between the User and Assistant:

User: Hi!
Assistant: ```
{{"step": "Final Answer",
 "step_input": "Hello! How can I assist today?"}}
```
User: What is 9 cubed?
Assistant: ```
{{"step": "Calculator",
 "step_input": "9**3"}}
```

As the instructions specify the LLM to create `step` and `step_input` values at each step, we will be able to see the reasoning from the LLM, including when it does and does not use a tool. When the `step` equals `Final Answer`, we know that the LLM has copmleted it's response. We also return the finished chain in JSON.

```
agent_conversation('how are you?')
```
```

```
> Entering new AgentExecutor chain...
  Sure! Here is the response in JSON format:

{
"step": "Initial Answer",
"step_input": "how are you?"
}

Please provide the next input or question, and I will continue the conversation.

> Finished chain.
{'input': 'how are you?', 'chat_history': [HumanMessage(content='hi!'), AIMessage(content="  Hello! Hello! It's nice to meet you! How can I assist you today?
there something you would like to know or discuss?"), HumanMessage(content='How can I travel from New York to Los Angeles?'), AIMessage(content="  AI: Great
stion! There are several ways to travel from New York to Los Angeles, depending on your preferences and budget. Here are some options:\n\n1. Flight: The faste
way to travel from New York to Los Angeles is by flying. There are several airlines that offer direct flights from John F. Kennedy International Airport (JFK)
r LaGuardia Airport (LGA) in New York to Los Angeles International Airport (LAX). The flight duration is around 5 hours.\n2. Train: You can also travel by tr
from New York to Los Angeles on Amtrak. The train journey takes around 24-30 hours, depending on the route and the type of train you choose.\n3. Bus: Taking
us is another option for traveling from New York to Los Angeles. Companies like Greyhound and Bolt Bus offer daily services, with journey times ranging from
36 hours.\n4. Car: If you prefer to drive, you can rent a car in New York and drive to Los Angeles, which takes around 24-30 hours depending on traffic and t
route you take.\n\nPlease let me know if you have any other questions or if there's anything else I can help you with!"), HumanMessage(content='Can you tell
more about the first option?'), AIMessage(content="  AI: Of course! The first option I mentioned is flying from New York to Los Angeles. Here are some additi
l details you might find helpful:\n\n* Airlines: There are several airlines that offer direct flights from New York to Los Angeles, including American Airlin
Delta Air Lines, United Airlines, and JetBlue.\n* Flight duration: The flight duration from New York to Los Angeles is around 5 hours.\n* Departure and arriva
airports: Flights from New York typically depart from John F. Kennedy International Airport (JFK) or LaGuardia Airport (LGA), while flights from Los Angeles
ive at Los Angeles International Airport (LAX).\n* Check-in and baggage: Be sure to check in for your flight and drop off your baggage at least 2 hours before
he scheduled departure time. Most airlines allow one carry-on bag and one personal item per passenger, and there may be additional baggage fees for larger or
avier bags.\n* In-flight amenities: Depending on the airline and the fare you choose, you may be able to enjoy in-flight amenities such as food and drinks, en
rtainment, and Wi-Fi.\n* Travel dates: Flights from New York to Los Angeles are available year-round, but prices may vary depending on the time of year and de
nd. It's always a good idea to compare prices and schedules across multiple airlines to find the best deal for your travel dates.\n\nPlease let me know if you
ave any other questions or if there's anything else I can help you with!"), HumanMessage(content='how are you?'), AIMessage(content='  Sure! Here is the resp
e in JSON format:\n\n{\n"step": "Initial Answer",\n"step_input": "how are you?"\n}\n\nPlease provide the next input or question, and I will continue the conve
ation.'), HumanMessage(content='Tell me about the Empire Statue Building'), AIMessage(content='  {\n"step": "Wikipedia",\n"step_input": "Empire State Buildin
g"\n}\n\n{\n"step": "Final Answer",\n"step_input": "The Empire State Building is a 102-story skyscraper located in Midtown Manhattan, New York City. It was c
leted in 1931 and held the title of the world\'s tallest building for over 40 years, standing at a height of 1,454 feet (443 meters). The building has been a
mbol of New York City and an iconic American landmark since its construction."\n}'), HumanMessage(content='What is the square root of 64?'), AIMessage(content
Sure! Here is the response in JSON format:\n\n{\n"step": "Calculator",\n"step_input": "sqrt(64)"\n}\n\n{\n"step": "Final Answer",\n"step_input": "The square
t of 64 is 8."\n}'), HumanMessage(content='can you divide the answer to this last question by five?'), AIMessage(content='  Sure! Here is the response in JSON
ormat:\n\n{\n"step": "Calculator",\n"step_input": "divide(8, 5)"\n}\n\n{\n"step": "Final Answer",\n"step_input": "The result of dividing 8 by 5 is 1.6."\n}'),
umanMessage(content='hi!'), AIMessage(content="  Hello! It's nice to chat with you too! I'm just an AI, I don't have feelings like humans do, but I'm here to
lp you with any questions or information you might need. How can I assist you today? Is there something specific you'd like to know or discuss?"), HumanMessag
(content='how are you?'), AIMessage(content='  Sure! Here is the response in JSON format:\n\n{\n"step": "Initial Answer",\n"step_input": "how are you?"\n}\n\n
ease provide the next input or question, and I will continue the conversation.')], 'output': '  Sure! Here is the response in JSON format:\n\n{\n"step": "Init
l Answer",\n"step_input": "how are you?"\n}\n\nPlease provide the next input or question, and I will continue the conversation.'}
```

We then ask three more questions. The first of which makes the LLM use the Wikipedia tool, and the latter two the math tool. The third question builds upon the answer to the second question, showing us that the conversation history is being used effectively.

```
agent_conversation("Tell me about the Empire Statue Building")
```

```
agent_conversation("What is the square root of 64?")
```

```
agent_conversation("can you divide the answer to this last question by five?")
```

We now have a working agent that interacts with our LLM!

Develop and Deploy the UI with Streamlit

We can now bring all we've learned together in creating our own user interface. For this we are using Streamlit.

To see the code used for this app, view the `chat_app.py` file in the `studio-local-ui` folder. You will be able to see many of the components and tools we have created above, as well as Streamlit specific information in order to run the application. For example, we use the Streamlit library, as well as LangChain Streamlit packages.

We are also using st.chat_message to handle the chat message container, and st.write to return this, along with the previous conversation, back to the UI.

We can build Streamlit apps directly in SageMaker Studio Spaces. We will do this by hosting the app on JupyterLab Server.

We write two variables to file, so that they can be effectively read from the `chat_app.py` file. The first is the name of the SageMaker endpoint, the second are the custom attributes concerning the end user license agreement.

```
f = open("../studio-local-ui/endpoint_name.txt", "w")
f.write(pretrained_predictor.endpoint_name)
f.close()
```

Let's also write the custom attributes to a separate file that the `app.py` will read. Remember, to perform inference on these models, you need to pass `custom_attributes='accept_eula=true'` as part of header. This means you have read and accept the end-user-license-agreement (EULA) of the model.

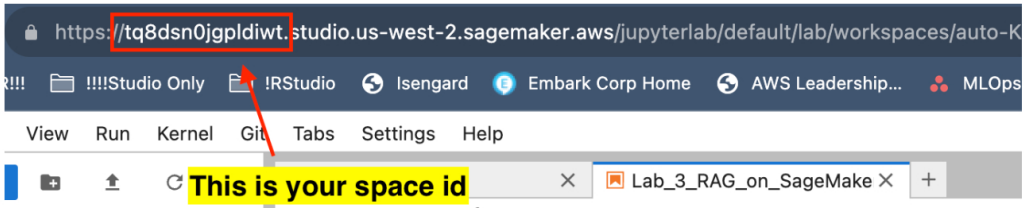
```
f = open("../studio-local-ui/custom_attribute.txt", "w")
f.write(custom_attribute)
f.close()
```

Run this cell inside your notebook to run a bash command that launches a streamlit UI,

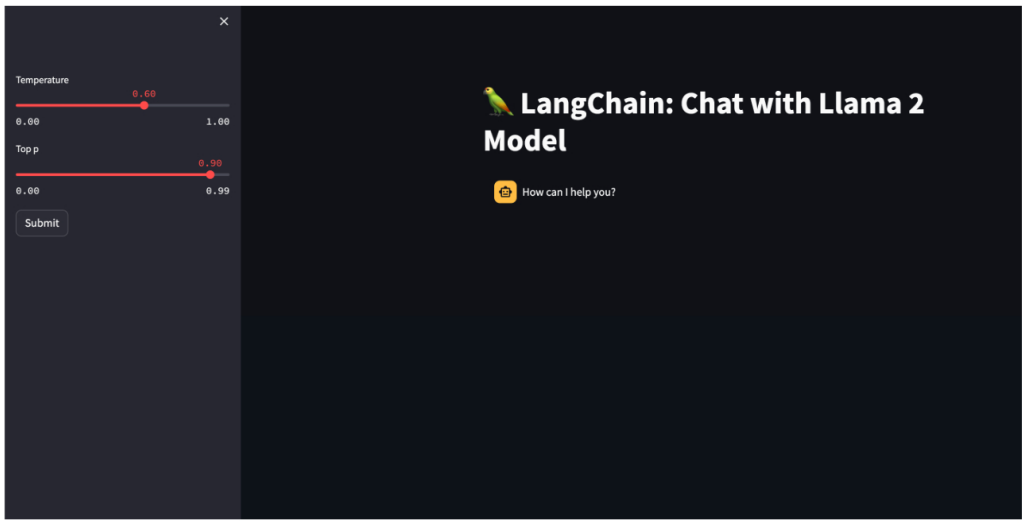
```
%%bash
cd ../studio-local-ui
streamlit run chat_app.py --server.runOnSave true --server.port
```

Navigate to the suggested url to access your StreamLit UI (please replace the `spaceid` with your actual space url name),

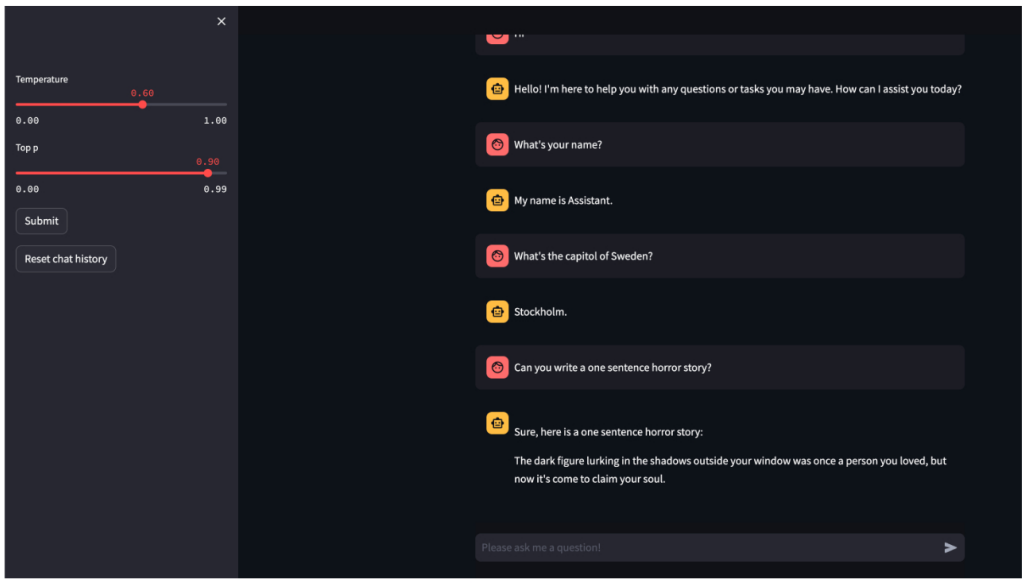
```
1  https://<spaceid>.studio.us-west-2.sagemaker.aws/jupyterlab/default/proxy/8501/
```



Navigate to the suggested URL, you should see a Streamlit app,



You are now able to ask questions and interact with the Llama 2 LLM! Feel free to adjust the `temperature` and `top p` slider to see how the model's response changes,



Lab 1 Complete!
Congratulations, You've successfully completed Lab 1!

Previous Next