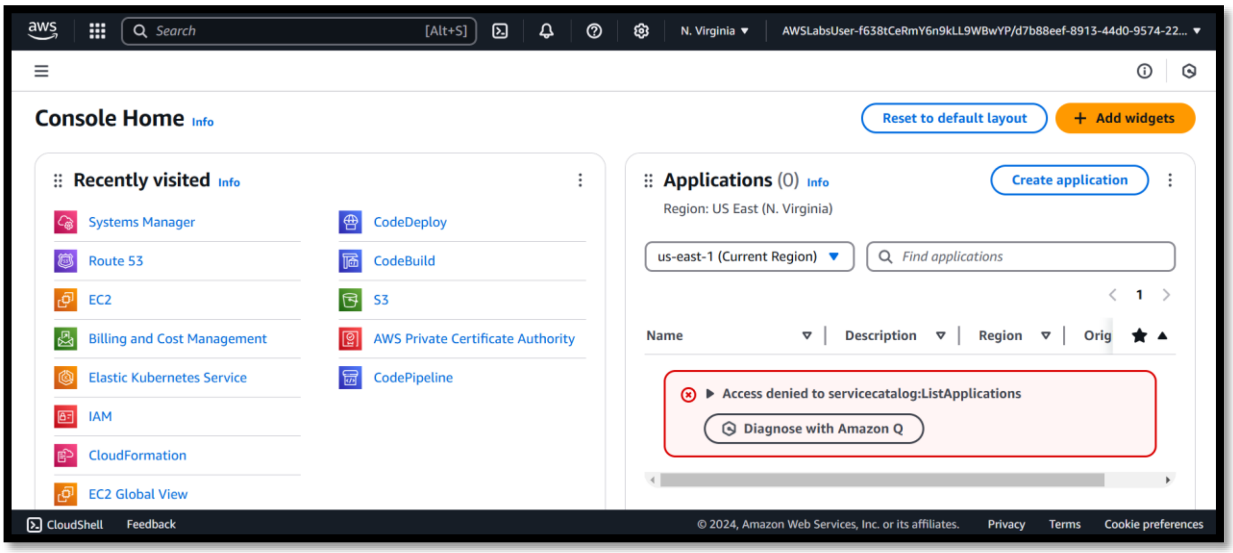
Objective: To build a chatbot using the Foundation Models (FMs) in Amazon Bedrock, specifically utilizing llama3-8b-instruct and titan-text-premier as the FMs for the chatbot development.

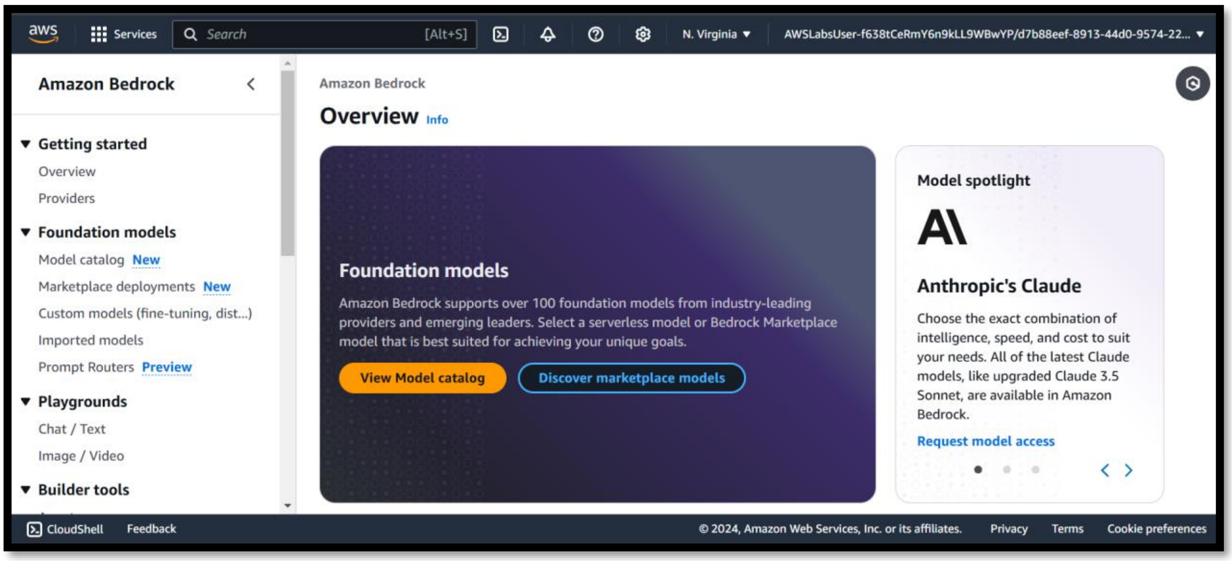
Task 0: Set up the environment

In this task, I registered the base models in the Amazon Bedrock console and launched an Amazon SageMaker Studio application to access my lab resources.



Access denied to servicecatalog:ListApplications

© 2024, Amazon Web Services, Inc. or its affiliates.

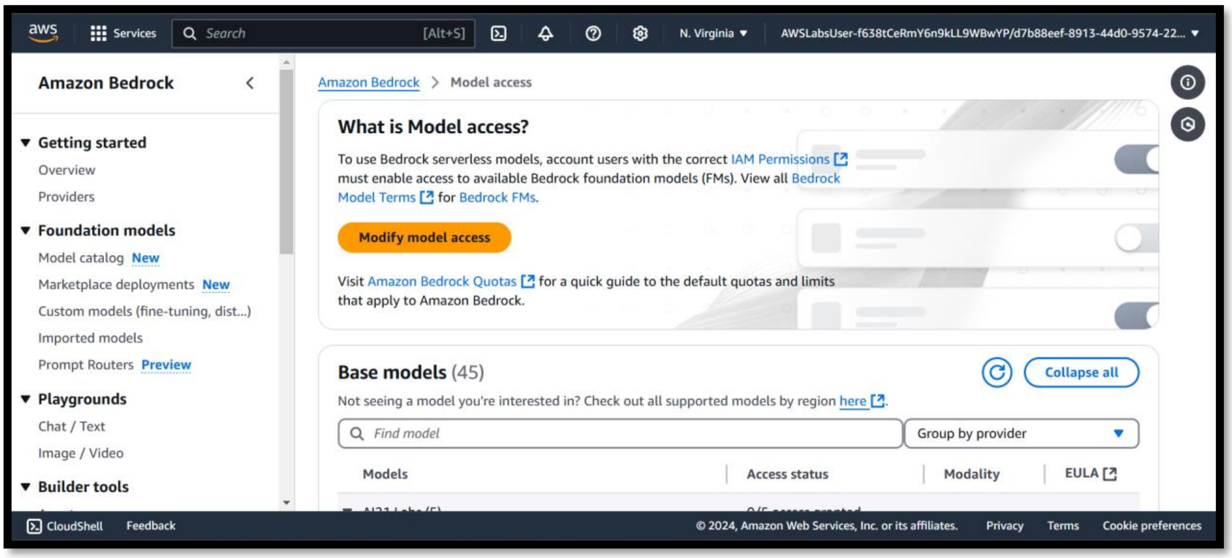


Amazon Bedrock supports over 100 foundation models from industry-leading providers and emerging leaders. Select a serverless model or Bedrock Marketplace model that is best suited for achieving your unique goals.

Choose the exact combination of intelligence, speed, and cost to suit your needs. All of the latest Claude models, like upgraded Claude 3.5 Sonnet, are available in Amazon Bedrock.

© 2024, Amazon Web Services, Inc. or its affiliates.

I reviewed the Access Status for each of the models. If the Access Status for one or more of the models was set to Available to request, I expanded this menu and followed the steps to enable access for them.



What is Model access?

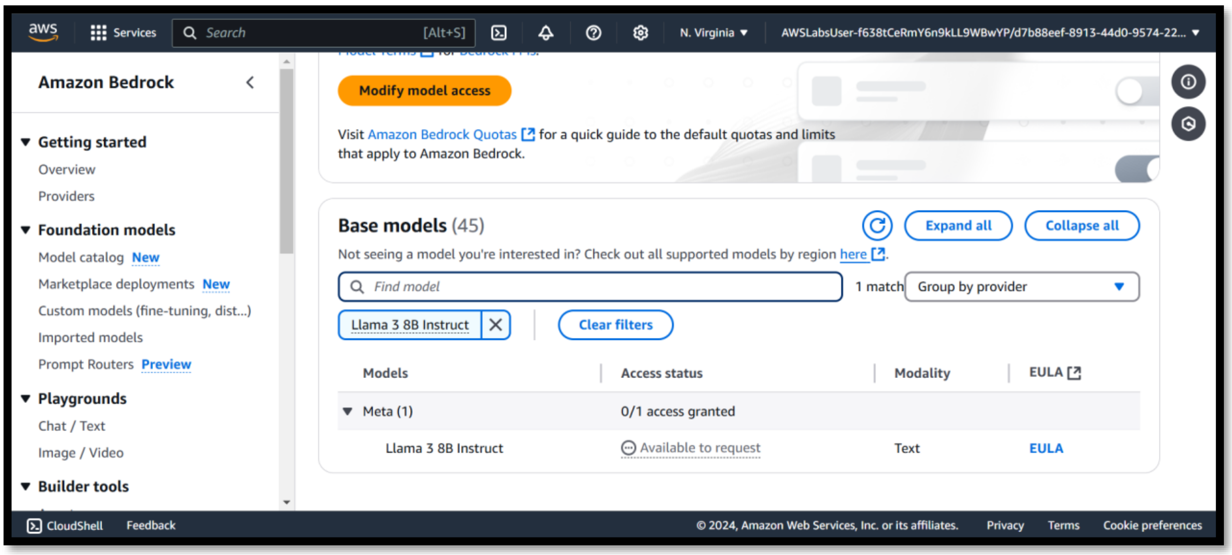
To use Bedrock serverless models, account users with the correct IAM Permissions must enable access to available Bedrock foundation models (FMs). View all Bedrock Model Terms for Bedrock FMs.

Visit Amazon Bedrock Quotas for a quick guide to the default quotas and limits that apply to Amazon Bedrock.

Base models (45)

Not seeing a model you're interested in? Check out all supported models by region here.

Models	Access status	Modality	EULA



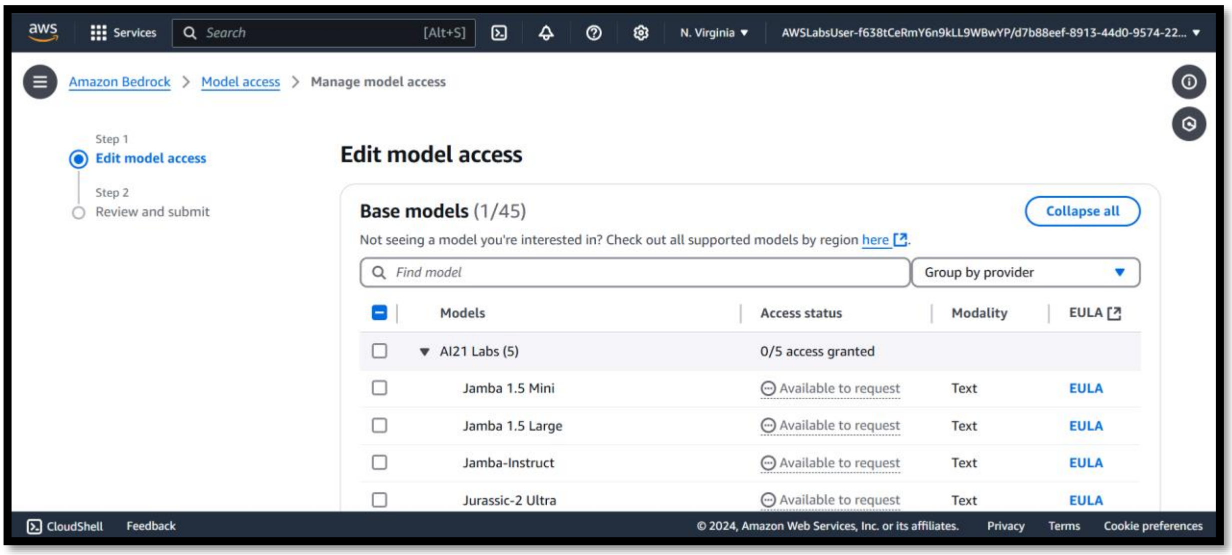
Visit Amazon Bedrock Quotas for a quick guide to the default quotas and limits that apply to Amazon Bedrock.

Base models (45)

Not seeing a model you're interested in? Check out all supported models by region here.

Models	Access status	Modality	EULA
Meta (1)	0/1 access granted		
Llama 3 8B Instruct	Available to request	Text	EULA

I chose Modify model access at the top of the screen.



Amazon Bedrock > Model access > Manage model access

Step 1
Edit model access

Step 2
Review and submit

Edit model access

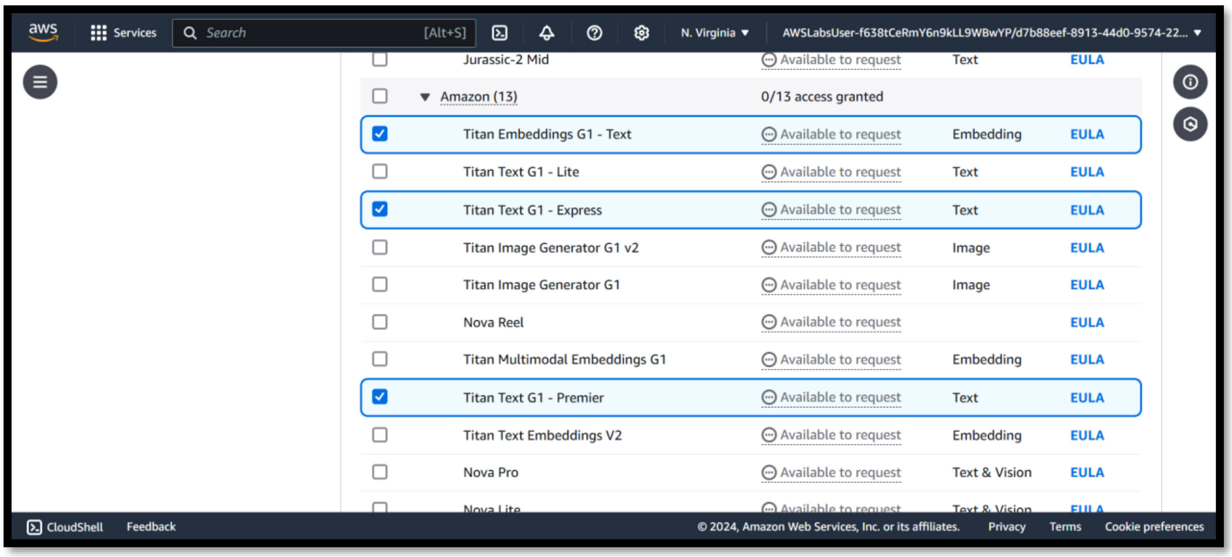
Base models (1/45)

Not seeing a model you're interested in? Check out all supported models by region here.

Group by provider

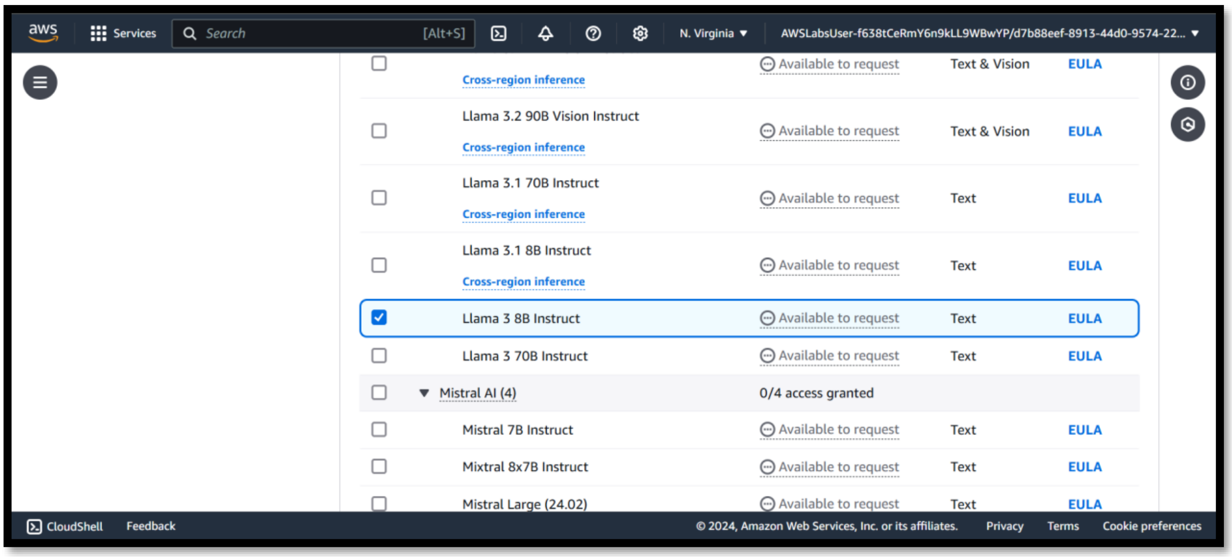
	Models	Access status	Modality	EULA
☐	▼ AI21 Labs (5)	0/5 access granted		
☐	Jamba 1.5 Mini	Available to request	Text	EULA
☐	Jamba 1.5 Large	Available to request	Text	EULA
☐	Jamba-Instruct	Available to request	Text	EULA
☐	Jurassic-2 Ultra	Available to request	Text	EULA

© 2024, Amazon Web Services, Inc. or its affiliates.



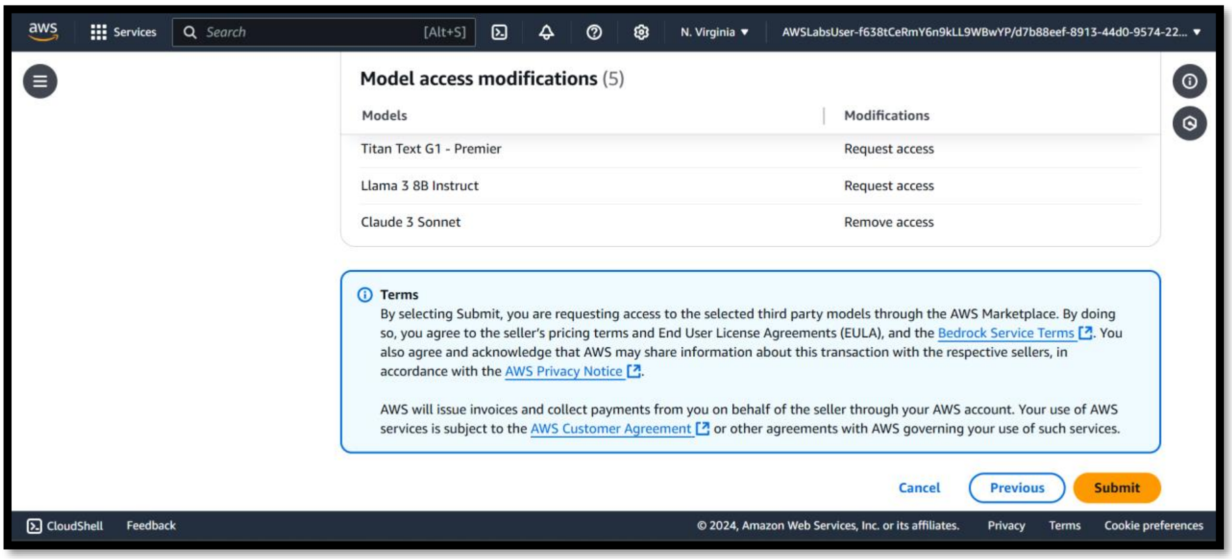
	Models	Access status	Modality	EULA
☐	Jurassic-2 Mid	Available to request	Text	EULA
☐	▼ Amazon (13)	0/13 access granted		
✓	Titan Embeddings G1 - Text	Available to request	Embedding	EULA
☐	Titan Text G1 - Lite	Available to request	Text	EULA
✓	Titan Text G1 - Express	Available to request	Text	EULA
☐	Titan Image Generator G1 v2	Available to request	Image	EULA
☐	Titan Image Generator G1	Available to request	Image	EULA
☐	Nova Reel	Available to request		EULA
☐	Titan Multimodal Embeddings G1	Available to request	Embedding	EULA
✓	Titan Text G1 - Premier	Available to request	Text	EULA
☐	Titan Text Embeddings V2	Available to request	Embedding	EULA
☐	Nova Pro	Available to request	Text & Vision	EULA
☐	Nova Lite	Available to request	Text & Vision	EULA

© 2024, Amazon Web Services, Inc. or its affiliates.



	Model	Access status	Modality	EULA
☐	Cross-region inference	Available to request	Text & Vision	EULA
☐	Llama 3.2 90B Vision Instruct Cross-region inference	Available to request	Text & Vision	EULA
☐	Llama 3.1 70B Instruct Cross-region inference	Available to request	Text	EULA
☐	Llama 3.1 8B Instruct Cross-region inference	Available to request	Text	EULA
☑	Llama 3 8B Instruct	Available to request	Text	EULA
☐	Llama 3 70B Instruct	Available to request	Text	EULA
☐	▼ Mistral AI (4)	0/4 access granted		
☐	Mistral 7B Instruct	Available to request	Text	EULA
☐	Mixtral 8x7B Instruct	Available to request	Text	EULA
☐	Mistral Large (24.02)	Available to request	Text	EULA

aws
Services
Search [Alt+S]
N. Virginia ▾
AWSLabsUser-f638tCeRmY6n9kLL9WBwYP/d7b88eef-8913-44d0-9574-22... ▾
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences



Model access modifications (5)

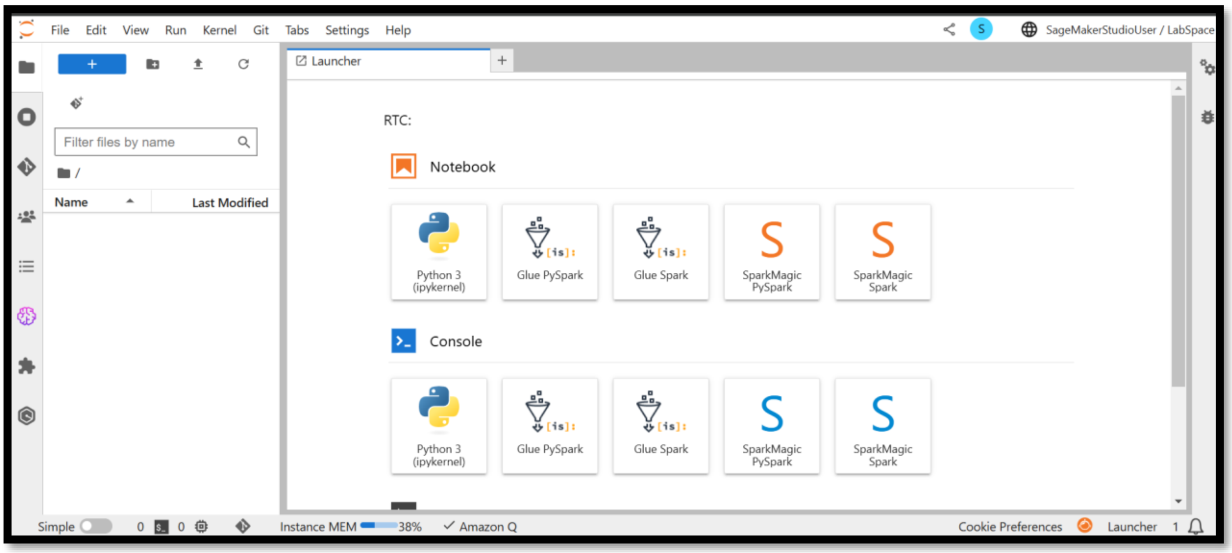
Models	Modifications
Titan Text G1 - Premier	Request access
Llama 3 8B Instruct	Request access
Claude 3 Sonnet	Remove access

Terms
By selecting Submit, you are requesting access to the selected third party models through the AWS Marketplace. By doing so, you agree to the seller's pricing terms and End User License Agreements (EULA), and the Bedrock Service Terms. You also agree and acknowledge that AWS may share information about this transaction with the respective sellers, in accordance with the AWS Privacy Notice.

AWS will issue invoices and collect payments from you on behalf of the seller through your AWS account. Your use of AWS services is subject to the AWS Customer Agreement or other agreements with AWS governing your use of such services.

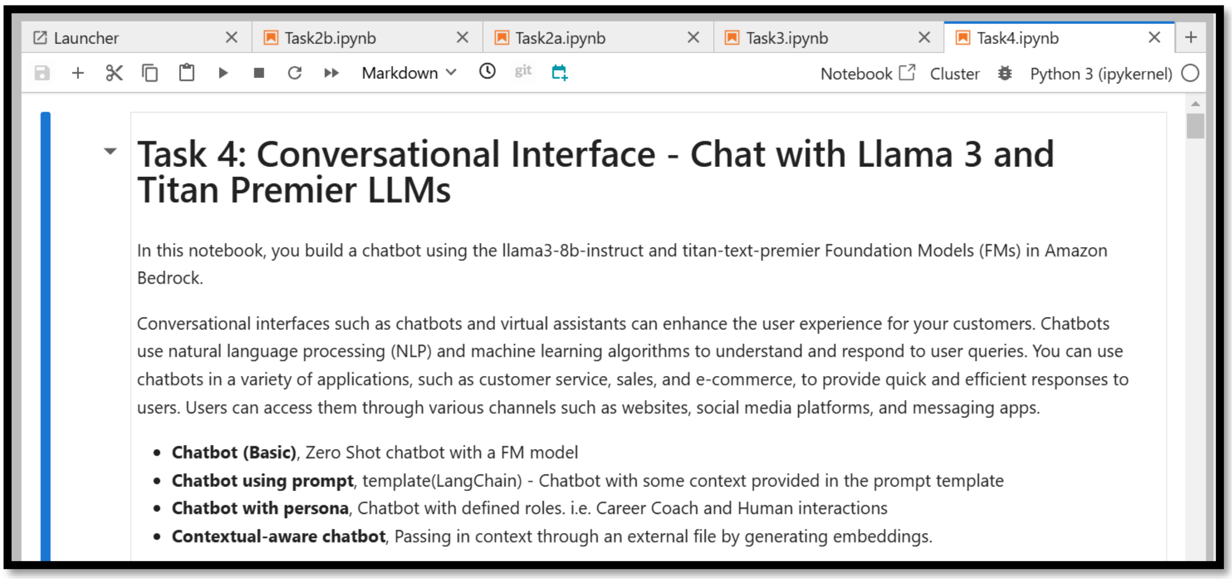
aws
Services
Search [Alt+S]
N. Virginia ▾
AWSLabsUser-f638tCeRmY6n9kLL9WBwYP/d7b88eef-8913-44d0-9574-22... ▾
Cancel
Previous
Submit
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

Launch an Amazon SageMaker Studio application



Task 1: Build a Chatbot

I built a chatbot using the Foundation Models (FMs) in Amazon Bedrock and used llama3-8b-instruct and titan-text-premier as my FMs for building the chatbots.

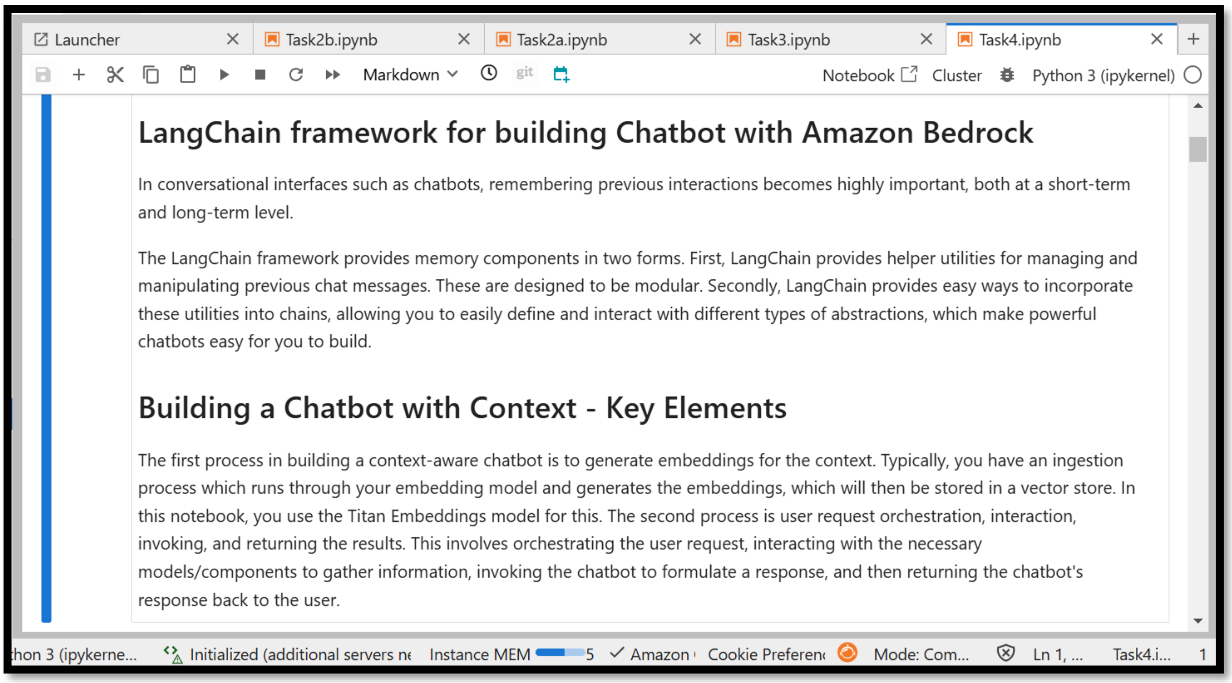


Task 4: Conversational Interface - Chat with Llama 3 and Titan Premier LLMs

In this notebook, you build a chatbot using the llama3-8b-instruct and titan-text-premier Foundation Models (FMs) in Amazon Bedrock.

Conversational interfaces such as chatbots and virtual assistants can enhance the user experience for your customers. Chatbots use natural language processing (NLP) and machine learning algorithms to understand and respond to user queries. You can use chatbots in a variety of applications, such as customer service, sales, and e-commerce, to provide quick and efficient responses to users. Users can access them through various channels such as websites, social media platforms, and messaging apps.

- **Chatbot (Basic)**, Zero Shot chatbot with a FM model
- **Chatbot using prompt**, template(LangChain) - Chatbot with some context provided in the prompt template
- **Chatbot with persona**, Chatbot with defined roles. i.e. Career Coach and Human interactions
- **Contextual-aware chatbot**, Passing in context through an external file by generating embeddings.



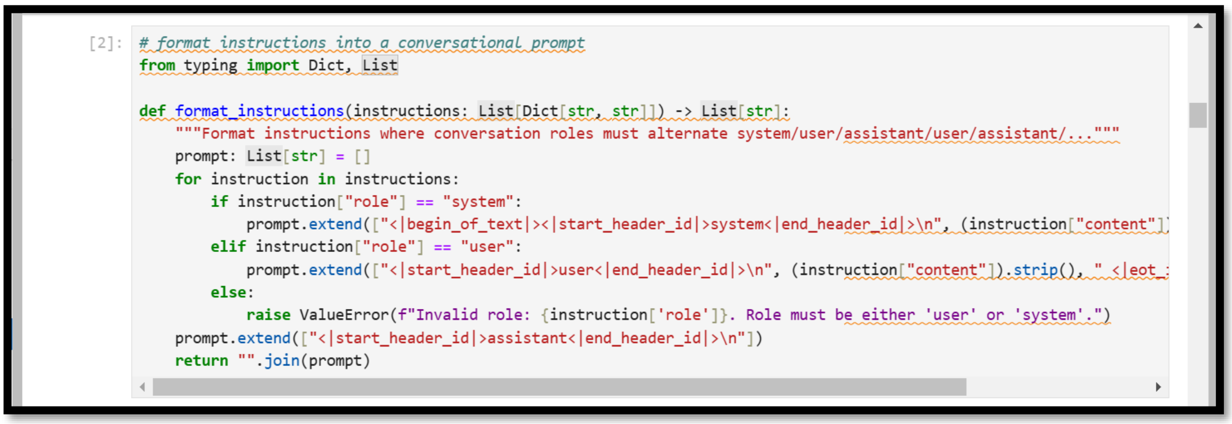
LangChain framework for building Chatbot with Amazon Bedrock

In conversational interfaces such as chatbots, remembering previous interactions becomes highly important, both at a short-term and long-term level.

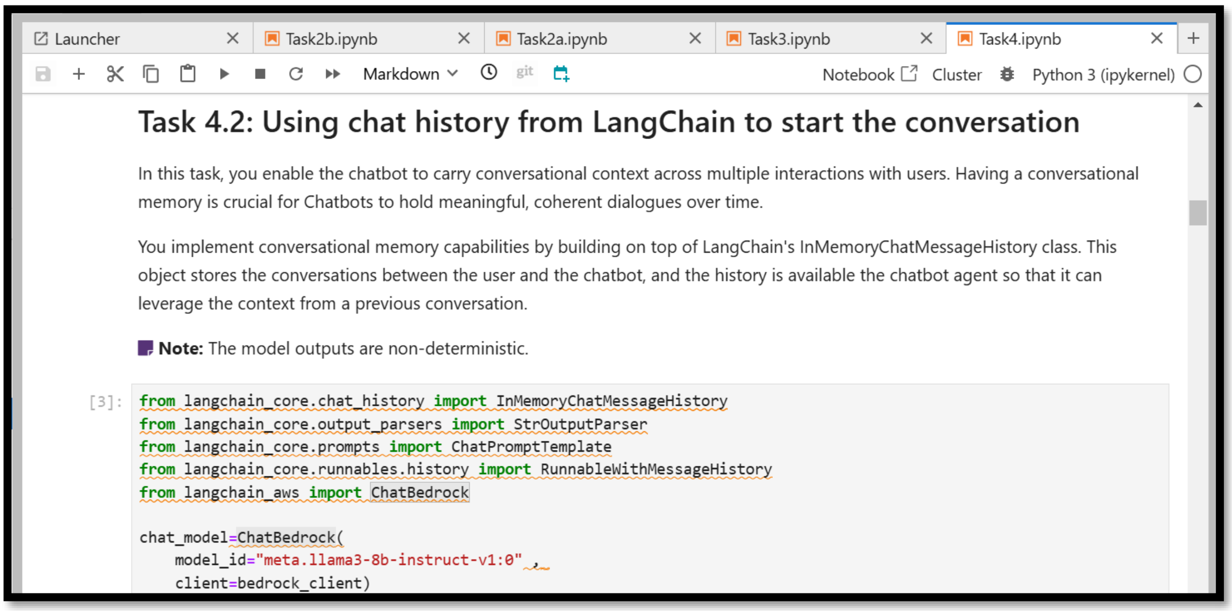
The LangChain framework provides memory components in two forms. First, LangChain provides helper utilities for managing and manipulating previous chat messages. These are designed to be modular. Secondly, LangChain provides easy ways to incorporate these utilities into chains, allowing you to easily define and interact with different types of abstractions, which make powerful chatbots easy for you to build.

Building a Chatbot with Context - Key Elements

The first process in building a context-aware chatbot is to generate embeddings for the context. Typically, you have an ingestion process which runs through your embedding model and generates the embeddings, which will then be stored in a vector store. In this notebook, you use the Titan Embeddings model for this. The second process is user request orchestration, interaction, invoking, and returning the results. This involves orchestrating the user request, interacting with the necessary models/components to gather information, invoking the chatbot to formulate a response, and then returning the chatbot's response back to the user.



```
[2]: # format instructions into a conversational prompt
from typing import Dict, List

def format_instructions(instructions: List[Dict[str, str]]) -> List[str]:
    """Format instructions where conversation roles must alternate system/user/assistant/user/assistant/..."""
    prompt: List[str] = []
    for instruction in instructions:
        if instruction["role"] == "system":
            prompt.extend(["<|begin_of_text|><|start_header_id|>system<|end_header_id|>\n", (instruction["content"]
        elif instruction["role"] == "user":
            prompt.extend(["<|start_header_id|>user<|end_header_id|>\n", (instruction["content"]).strip(), " <|eot_
        else:
            raise ValueError(f"Invalid role: {instruction['role']}. Role must be either 'user' or 'system'.")
    prompt.extend(["<|start_header_id|>assistant<|end_header_id|>\n"])
    return "".join(prompt)
```



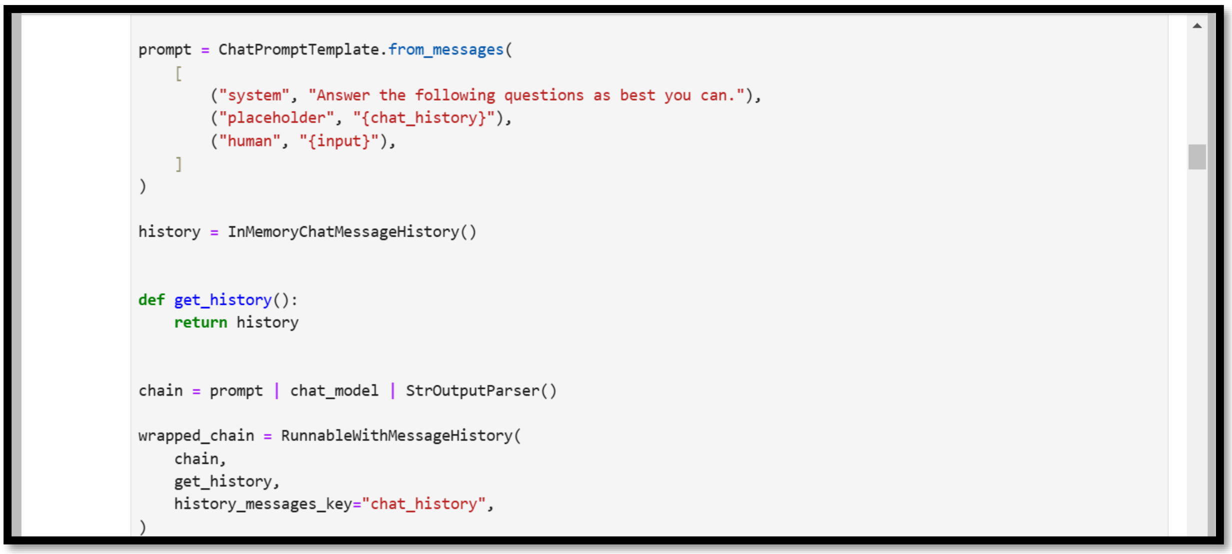
Task 4.2: Using chat history from LangChain to start the conversation

In this task, you enable the chatbot to carry conversational context across multiple interactions with users. Having a conversational memory is crucial for Chatbots to hold meaningful, coherent dialogues over time.

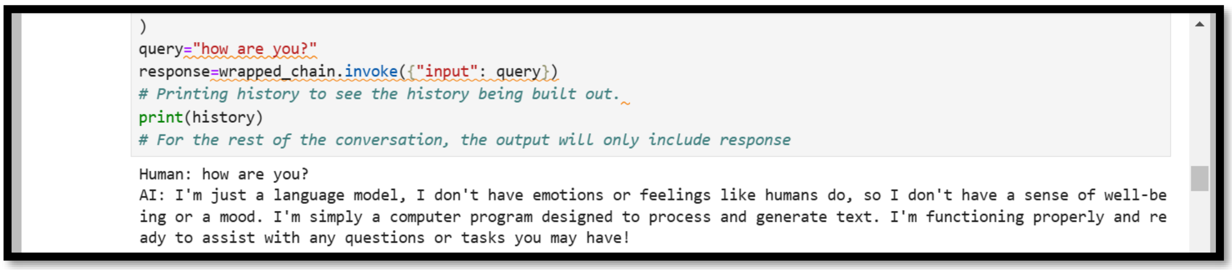
You implement conversational memory capabilities by building on top of LangChain's InMemoryChatMessageHistory class. This object stores the conversations between the user and the chatbot, and the history is available the chatbot agent so that it can leverage the context from a previous conversation.

Note: The model outputs are non-deterministic.

```
[3]: from langchain_core.chat_history import InMemoryChatMessageHistory
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables.history import RunnableWithMessageHistory
from langchain_aws import ChatBedrock

chat_model=ChatBedrock(
    model_id="meta.llama3-8b-instruct-v1:0" ,
    client=bedrock_client)
```

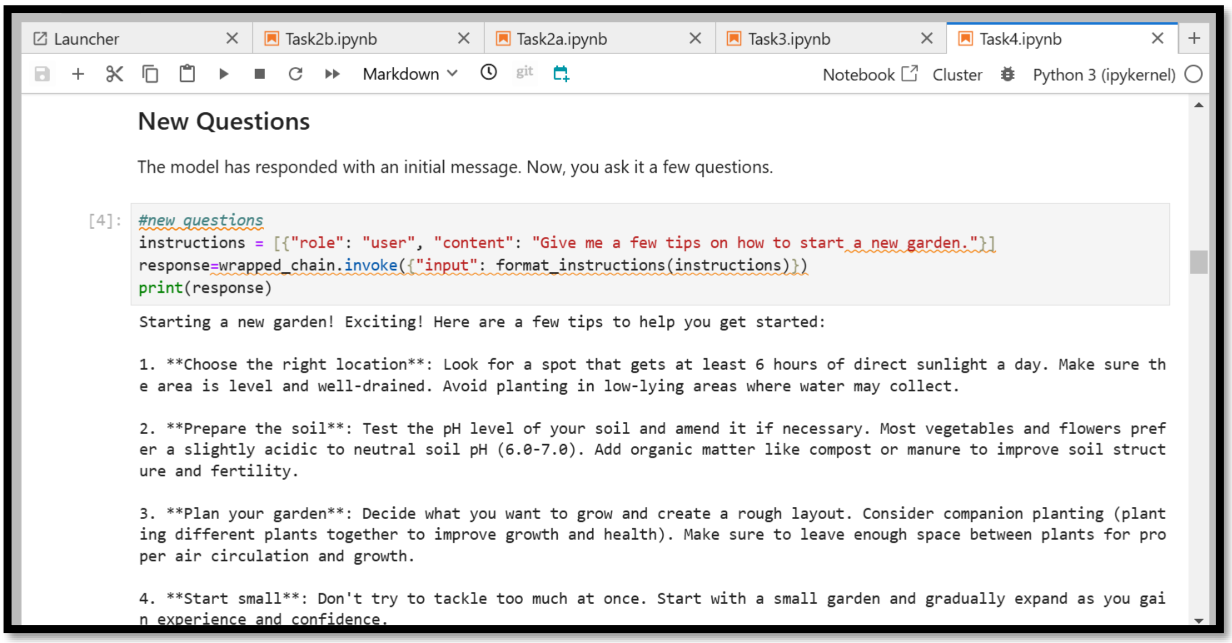


```
prompt = ChatPromptTemplate.from_messages(
    [
        ("system", "Answer the following questions as best you can."),
        ("placeholder", "{chat_history}"),
        ("human", "{input}"),
    ]
)

history = InMemoryChatMessageHistory()


def get_history():
    return history


chain = prompt | chat_model | StrOutputParser()

wrapped_chain = RunnableWithMessageHistory(
    chain,
    get_history,
    history_messages_key="chat_history",
)
```



```
)
query="how are you?"
response=wrapped_chain.invoke({"input": query})
# Printing history to see the history being built out.
print(history)
# For the rest of the conversation, the output will only include response
```

```
Human: how are you?
AI: I'm just a language model, I don't have emotions or feelings like humans do, so I don't have a sense of well-be
ing or a mood. I'm simply a computer program designed to process and generate text. I'm functioning properly and re
ady to assist with any questions or tasks you may have!
```

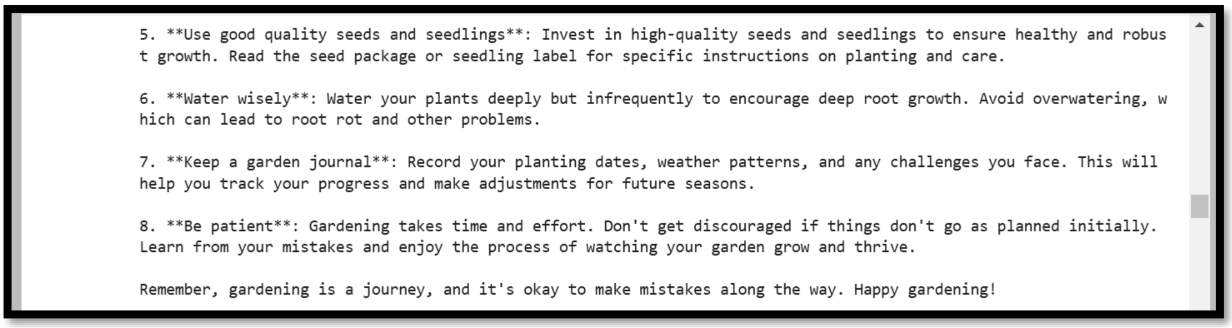


New Questions

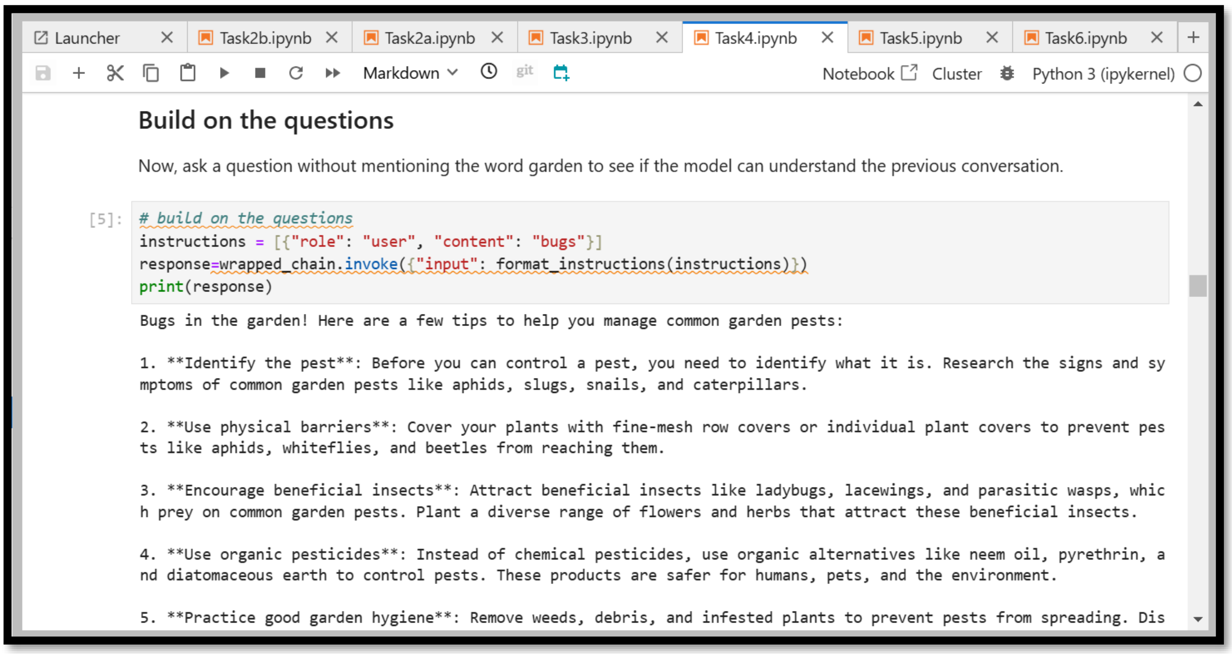
The model has responded with an initial message. Now, you ask it a few questions.

```
[4]: #new questions
instructions = [{"role": "user", "content": "Give me a few tips on how to start a new garden."}]
response=wrapped_chain.invoke({"input": format_instructions(instructions)})
print(response)
```

```
Starting a new garden! Exciting! Here are a few tips to help you get started:

1. **Choose the right location**: Look for a spot that gets at least 6 hours of direct sunlight a day. Make sure th
e area is level and well-drained. Avoid planting in low-lying areas where water may collect.

2. **Prepare the soil**: Test the pH level of your soil and amend it if necessary. Most vegetables and flowers pref
er a slightly acidic to neutral soil pH (6.0-7.0). Add organic matter like compost or manure to improve soil struct
ure and fertility.

3. **Plan your garden**: Decide what you want to grow and create a rough layout. Consider companion planting (plant
ing different plants together to improve growth and health). Make sure to leave enough space between plants for pro
per air circulation and growth.

4. **Start small**: Don't try to tackle too much at once. Start with a small garden and gradually expand as you gai
n experience and confidence.
```



```
5. **Use good quality seeds and seedlings**: Invest in high-quality seeds and seedlings to ensure healthy and robus
t growth. Read the seed package or seedling label for specific instructions on planting and care.

6. **Water wisely**: Water your plants deeply but infrequently to encourage deep root growth. Avoid overwatering, w
hich can lead to root rot and other problems.

7. **Keep a garden journal**: Record your planting dates, weather patterns, and any challenges you face. This will
help you track your progress and make adjustments for future seasons.

8. **Be patient**: Gardening takes time and effort. Don't get discouraged if things don't go as planned initially.
Learn from your mistakes and enjoy the process of watching your garden grow and thrive.

Remember, gardening is a journey, and it's okay to make mistakes along the way. Happy gardening!
```

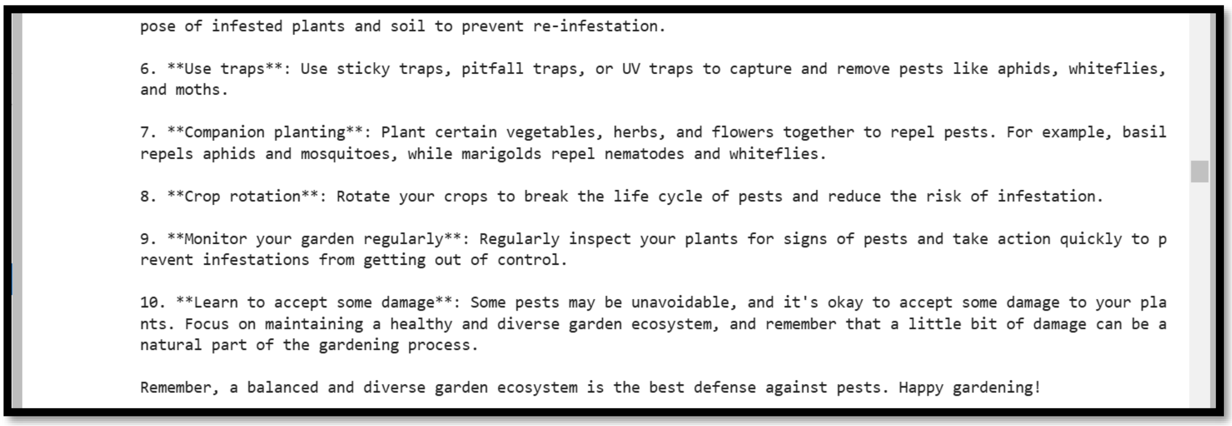


Build on the questions

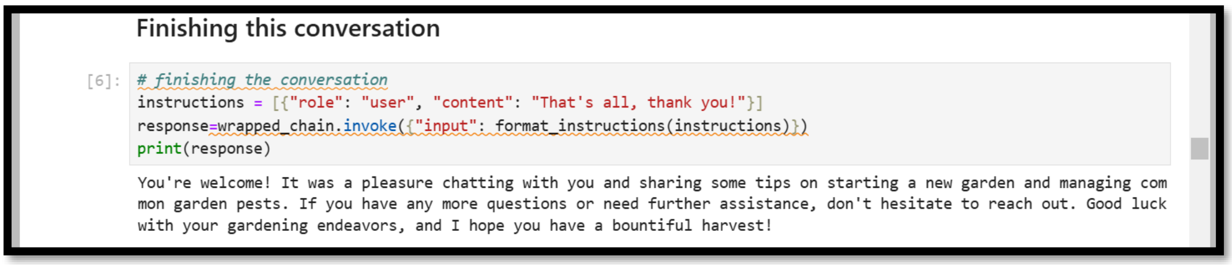
Now, ask a question without mentioning the word garden to see if the model can understand the previous conversation.

[5]:
```python
# build on the questions
instructions = [{"role": "user", "content": "bugs"}]
response=wrapped_chain.invoke({"input": format_instructions(instructions)})
print(response)
```

```
Bugs in the garden! Here are a few tips to help you manage common garden pests:

1. **Identify the pest**: Before you can control a pest, you need to identify what it is. Research the signs and sy
mptoms of common garden pests like aphids, slugs, snails, and caterpillars.

2. **Use physical barriers**: Cover your plants with fine-mesh row covers or individual plant covers to prevent pes
ts like aphids, whiteflies, and beetles from reaching them.

3. **Encourage beneficial insects**: Attract beneficial insects like ladybugs, lacewings, and parasitic wasps, whic
h prey on common garden pests. Plant a diverse range of flowers and herbs that attract these beneficial insects.

4. **Use organic pesticides**: Instead of chemical pesticides, use organic alternatives like neem oil, pyrethrin, a
nd diatomaceous earth to control pests. These products are safer for humans, pets, and the environment.

5. **Practice good garden hygiene**: Remove weeds, debris, and infested plants to prevent pests from spreading. Dis
```



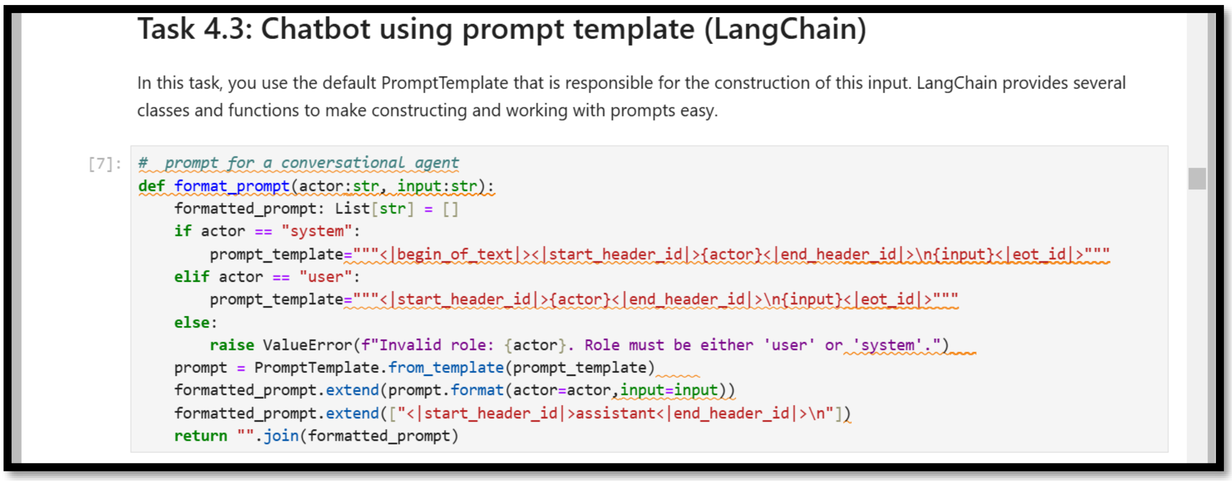
```
pose of infested plants and soil to prevent re-infestation.

6. **Use traps**: Use sticky traps, pitfall traps, or UV traps to capture and remove pests like aphids, whiteflies,
and moths.

7. **Companion planting**: Plant certain vegetables, herbs, and flowers together to repel pests. For example, basil
repels aphids and mosquitoes, while marigolds repel nematodes and whiteflies.

8. **Crop rotation**: Rotate your crops to break the life cycle of pests and reduce the risk of infestation.

9. **Monitor your garden regularly**: Regularly inspect your plants for signs of pests and take action quickly to p
revent infestations from getting out of control.

10. **Learn to accept some damage**: Some pests may be unavoidable, and it's okay to accept some damage to your pla
nts. Focus on maintaining a healthy and diverse garden ecosystem, and remember that a little bit of damage can be a
natural part of the gardening process.

Remember, a balanced and diverse garden ecosystem is the best defense against pests. Happy gardening!
```



Finishing this conversation

[6]:
```python
# finishing the conversation
instructions = [{"role": "user", "content": "That's all, thank you!"}]
response=wrapped_chain.invoke({"input": format_instructions(instructions)})
print(response)
```

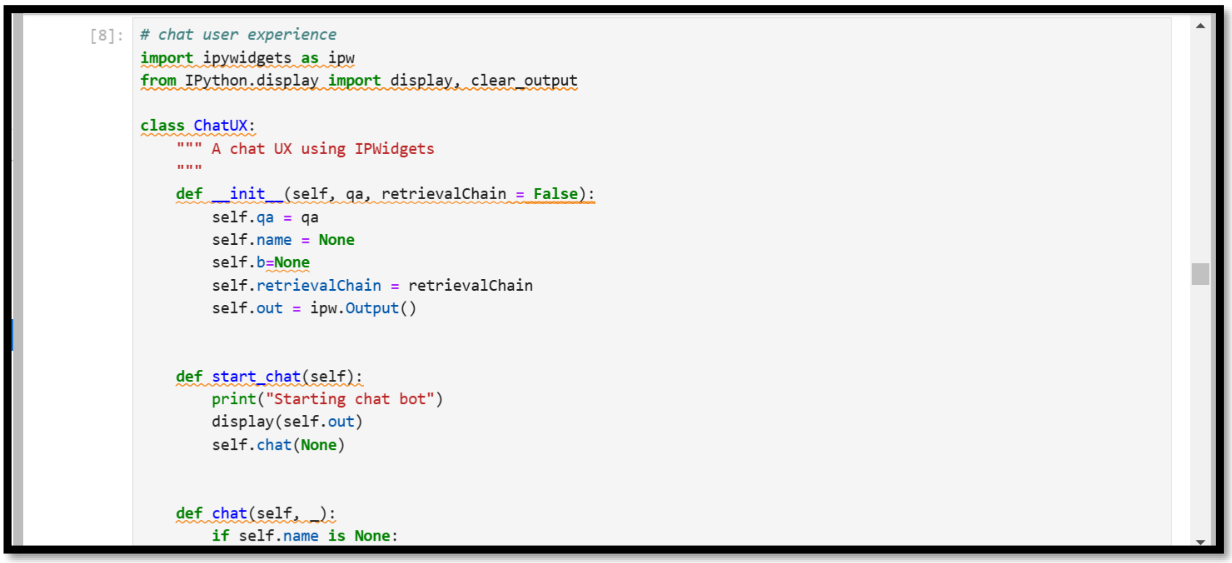
```
You're welcome! It was a pleasure chatting with you and sharing some tips on starting a new garden and managing com
mon garden pests. If you have any more questions or need further assistance, don't hesitate to reach out. Good luck
with your gardening endeavors, and I hope you have a bountiful harvest!
```



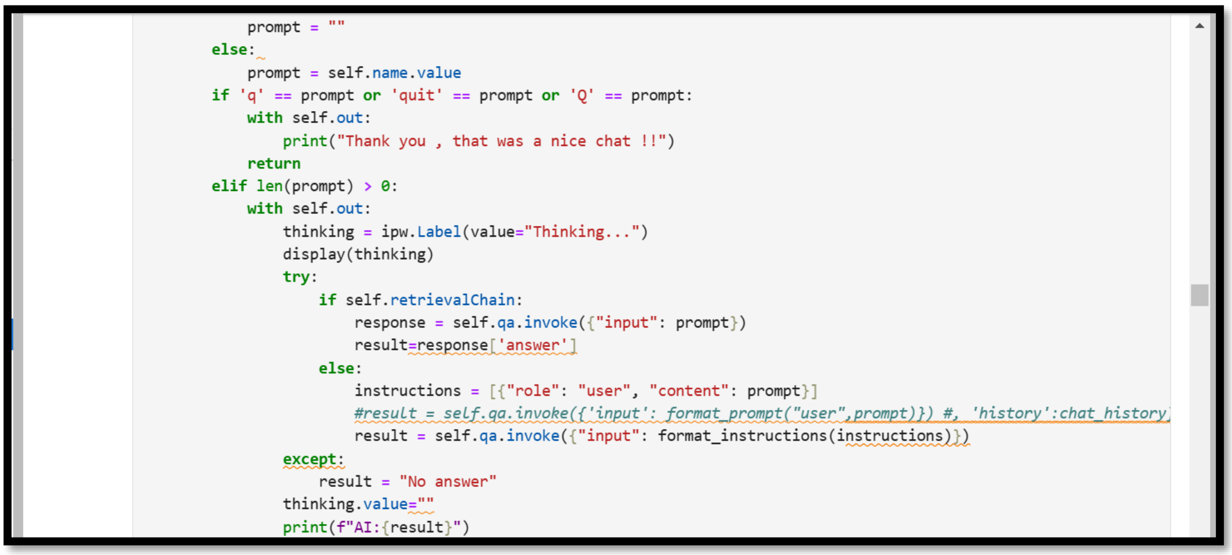
Task 4.3: Chatbot using prompt template (LangChain)

In this task, you use the default PromptTemplate that is responsible for the construction of this input. LangChain provides several classes and functions to make constructing and working with prompts easy.

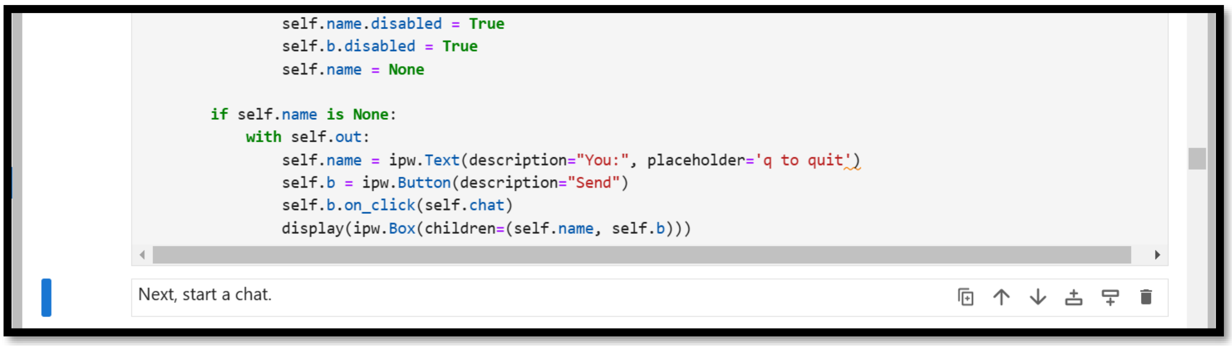
```
[7]: #  prompt  for  a  conversational  agent
def format_prompt(actor:str, input:str):
    formatted_prompt: List[str] = []
    if actor == "system":
        prompt_template="""<|begin_of_text|><|start_header_id|>{actor}<|end_header_id|>\n{input}<|eot_id|>"""
    elif actor == "user":
        prompt_template="""<|start_header_id|>{actor}<|end_header_id|>\n{input}<|eot_id|>"""
    else:
        raise ValueError(f"Invalid role: {actor}. Role must be either 'user' or 'system'.")
    prompt = PromptTemplate.from_template(prompt_template)
    formatted_prompt.extend(prompt.format(actor=actor,input=input))
    formatted_prompt.extend(["<|start_header_id|>assistant<|end_header_id|>\n"])
    return "".join(formatted_prompt)
```



```
[8]: # chat user experience
import ipywidgets as ipw
from IPython.display import display, clear_output

class ChatUX:
    """ A chat UX using IPWidgets
    """
    def __init__(self, qa, retrievalChain = False):
        self.qa = qa
        self.name = None
        self.b=None
        self.retrievalChain = retrievalChain
        self.out = ipw.Output()


    def start_chat(self):
        print("Starting chat bot")
        display(self.out)
        self.chat(None)


    def chat(self, _):
        if self.name is None:
```

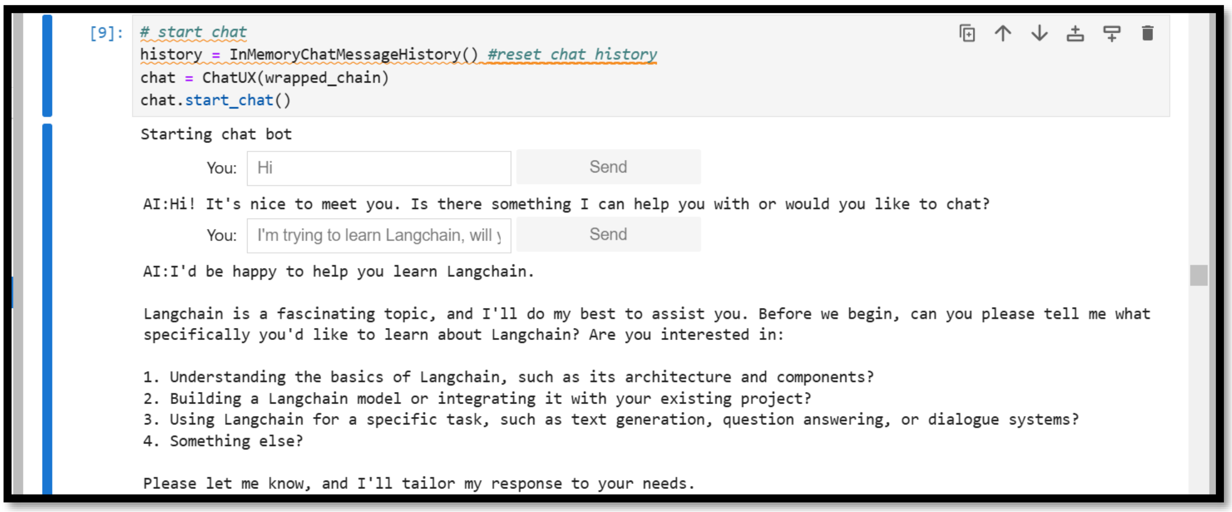


```
            prompt = ""
        else:
            prompt = self.name.value
        if 'q' == prompt or 'quit' == prompt or 'Q' == prompt:
            with self.out:
                print("Thank you , that was a nice chat !!")
            return
        elif len(prompt) > 0:
            with self.out:
                thinking = ipw.Label(value="Thinking...")
                display(thinking)
                try:
                    if self.retrievalChain:
                        response = self.qa.invoke({"input": prompt})
                        result=response['answer']
                    else:
                        instructions = [{"role": "user", "content": prompt}]
                        #result = self.qa.invoke({'input': format_prompt("user",prompt)}) #, 'history':chat_history
                        result = self.qa.invoke({"input": format_instructions(instructions)})
                except:
                    result = "No answer"
                thinking.value=""
                print(f"AI:{result}")
```



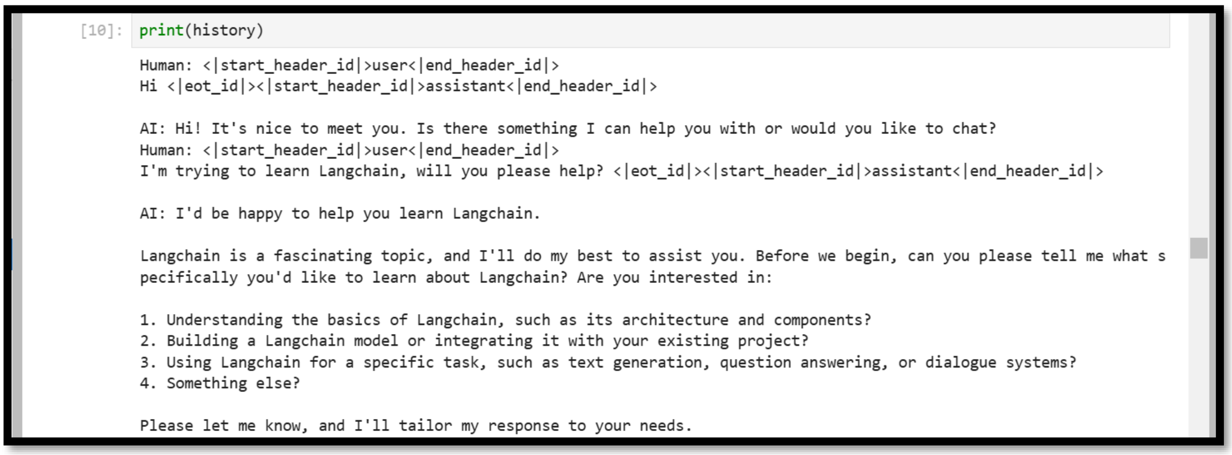
```
                self.name.disabled = True
                self.b.disabled = True
                self.name = None

        if self.name is None:
            with self.out:
                self.name = ipw.Text(description="You:", placeholder='q to quit')
                self.b = ipw.Button(description="Send")
                self.b.on_click(self.chat)
                display(ipw.Box(children=(self.name, self.b)))
```

Next, start a chat.



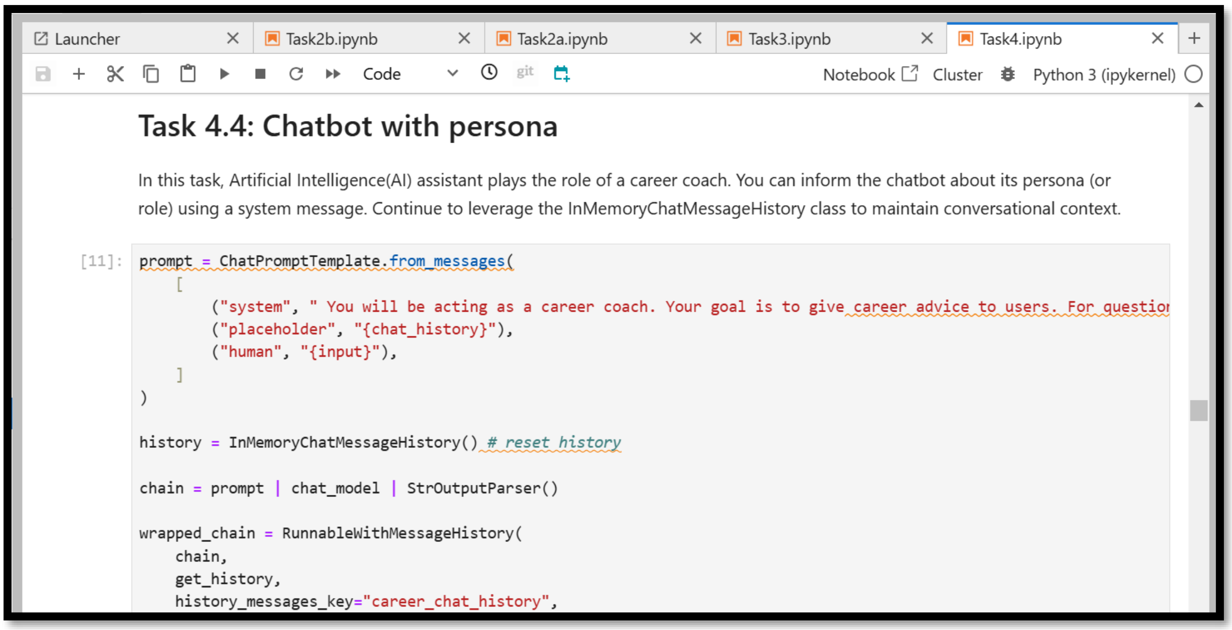
```
[9]: # start chat
     history = InMemoryChatMessageHistory() #reset chat history
     chat = ChatUX(wrapped_chain)
     chat.start_chat()
```

```
Starting chat bot
        You: Hi                                   Send
AI:Hi! It's nice to meet you. Is there something I can help you with or would you like to chat?
        You: I'm trying to learn Langchain, will y   Send
AI:I'd be happy to help you learn Langchain.

Langchain is a fascinating topic, and I'll do my best to assist you. Before we begin, can you please tell me what
specifically you'd like to learn about Langchain? Are you interested in:

1. Understanding the basics of Langchain, such as its architecture and components?
2. Building a Langchain model or integrating it with your existing project?
3. Using Langchain for a specific task, such as text generation, question answering, or dialogue systems?
4. Something else?

Please let me know, and I'll tailor my response to your needs.
```



```
[10]: print(history)
```

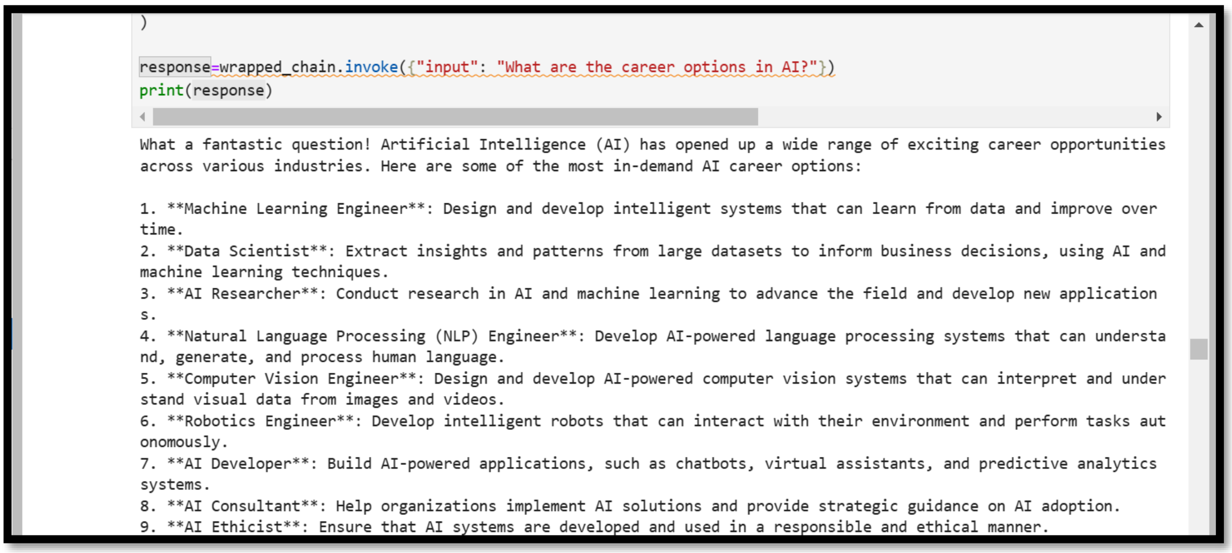
```
Human: <|start_header_id|>user<|end_header_id|>
Hi <|eot_id|><|start_header_id|>assistant<|end_header_id|>

AI: Hi! It's nice to meet you. Is there something I can help you with or would you like to chat?
Human: <|start_header_id|>user<|end_header_id|>
I'm trying to learn Langchain, will you please help? <|eot_id|><|start_header_id|>assistant<|end_header_id|>

AI: I'd be happy to help you learn Langchain.

Langchain is a fascinating topic, and I'll do my best to assist you. Before we begin, can you please tell me what s
pecifically you'd like to learn about Langchain? Are you interested in:

1. Understanding the basics of Langchain, such as its architecture and components?
2. Building a Langchain model or integrating it with your existing project?
3. Using Langchain for a specific task, such as text generation, question answering, or dialogue systems?
4. Something else?

Please let me know, and I'll tailor my response to your needs.
```



Task 4.4: Chatbot with persona

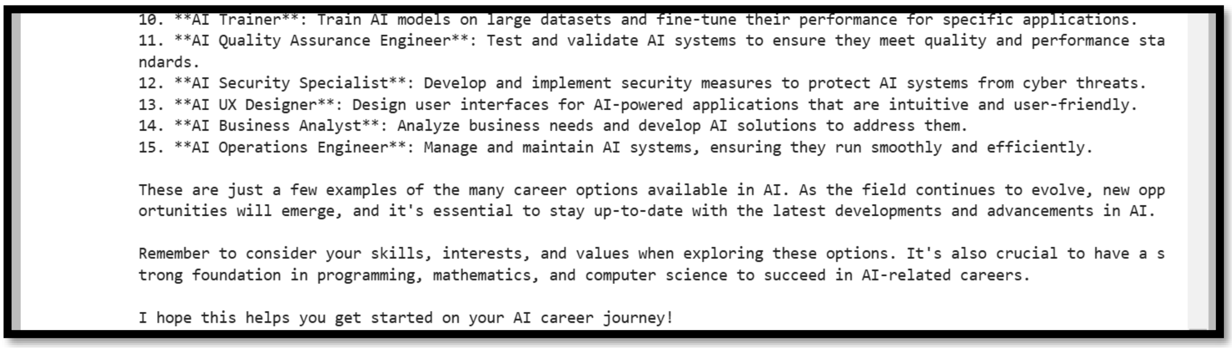
In this task, Artificial Intelligence(AI) assistant plays the role of a career coach. You can inform the chatbot about its persona (or role) using a system message. Continue to leverage the InMemoryChatMessageHistory class to maintain conversational context.

```
[11]: prompt = ChatPromptTemplate.from_messages(
          [
              ("system", " You will be acting as a career coach. Your goal is to give career advice to users. For questior
              ("placeholder", "{chat_history}"),
              ("human", "{input}"),
          ]
      )

      history = InMemoryChatMessageHistory() # reset history

      chain = prompt | chat_model | StrOutputParser()

      wrapped_chain = RunnableWithMessageHistory(
          chain,
          get_history,
          history_messages_key="career_chat_history",
```

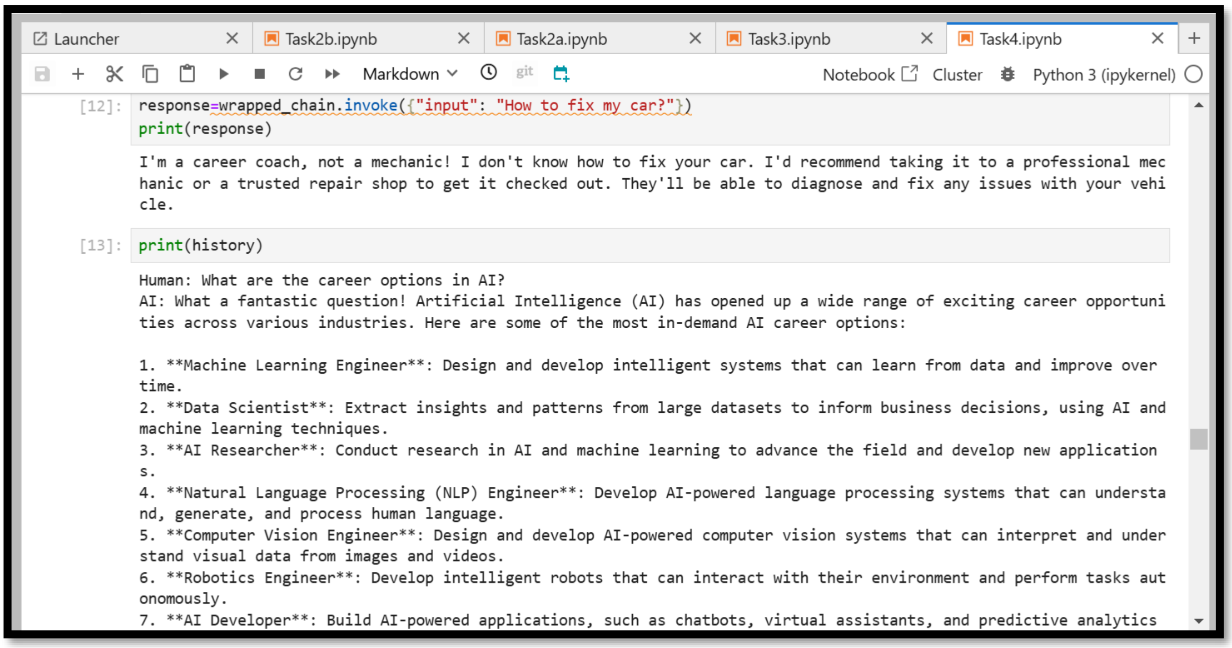


```
      )

      response=wrapped_chain.invoke({"input": "What are the career options in AI?"})
      print(response)
```

```
What a fantastic question! Artificial Intelligence (AI) has opened up a wide range of exciting career opportunities
across various industries. Here are some of the most in-demand AI career options:

1. **Machine Learning Engineer**: Design and develop intelligent systems that can learn from data and improve over
time.
2. **Data Scientist**: Extract insights and patterns from large datasets to inform business decisions, using AI and
machine learning techniques.
3. **AI Researcher**: Conduct research in AI and machine learning to advance the field and develop new application
s.
4. **Natural Language Processing (NLP) Engineer**: Develop AI-powered language processing systems that can understa
nd, generate, and process human language.
5. **Computer Vision Engineer**: Design and develop AI-powered computer vision systems that can interpret and under
stand visual data from images and videos.
6. **Robotics Engineer**: Develop intelligent robots that can interact with their environment and perform tasks aut
onomously.
7. **AI Developer**: Build AI-powered applications, such as chatbots, virtual assistants, and predictive analytics
systems.
8. **AI Consultant**: Help organizations implement AI solutions and provide strategic guidance on AI adoption.
9. **AI Ethicist**: Ensure that AI systems are developed and used in a responsible and ethical manner.
```



```
10. **AI Trainer**: Train AI models on large datasets and fine-tune their performance for specific applications.
11. **AI Quality Assurance Engineer**: Test and validate AI systems to ensure they meet quality and performance sta
ndards.
12. **AI Security Specialist**: Develop and implement security measures to protect AI systems from cyber threats.
13. **AI UX Designer**: Design user interfaces for AI-powered applications that are intuitive and user-friendly.
14. **AI Business Analyst**: Analyze business needs and develop AI solutions to address them.
15. **AI Operations Engineer**: Manage and maintain AI systems, ensuring they run smoothly and efficiently.

These are just a few examples of the many career options available in AI. As the field continues to evolve, new opp
ortunities will emerge, and it's essential to stay up-to-date with the latest developments and advancements in AI.

Remember to consider your skills, interests, and values when exploring these options. It's also crucial to have a s
trong foundation in programming, mathematics, and computer science to succeed in AI-related careers.

I hope this helps you get started on your AI career journey!
```

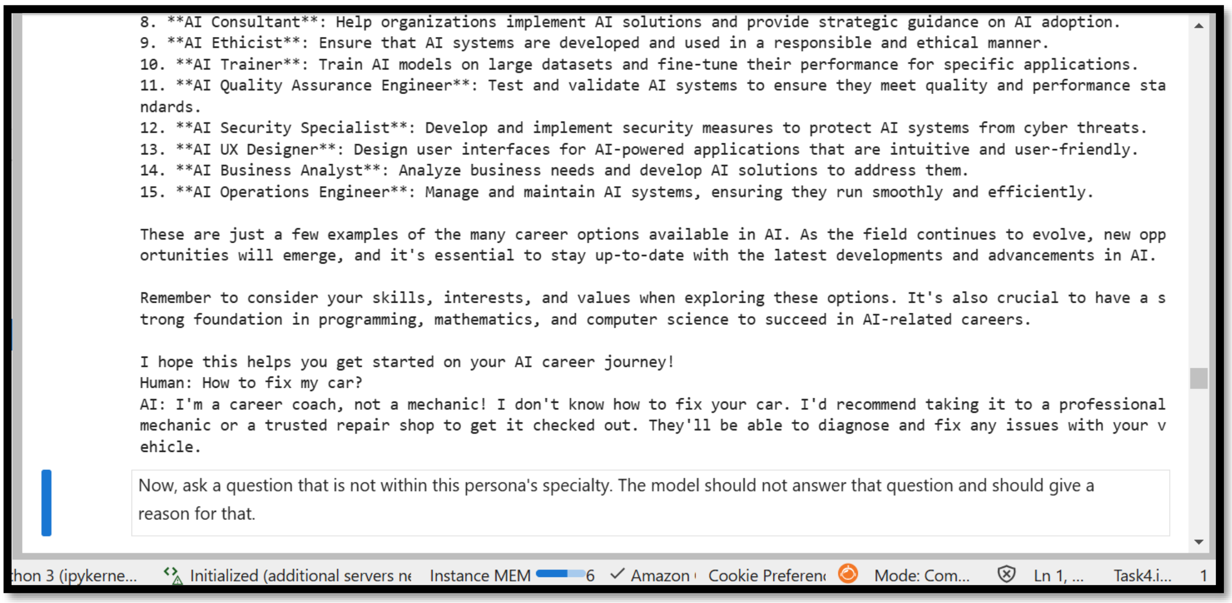


```
[12]: response=wrapped_chain.invoke({"input": "How to fix my car?"})
      print(response)
```

```
I'm a career coach, not a mechanic! I don't know how to fix your car. I'd recommend taking it to a professional mec
hanic or a trusted repair shop to get it checked out. They'll be able to diagnose and fix any issues with your vehi
cle.
```

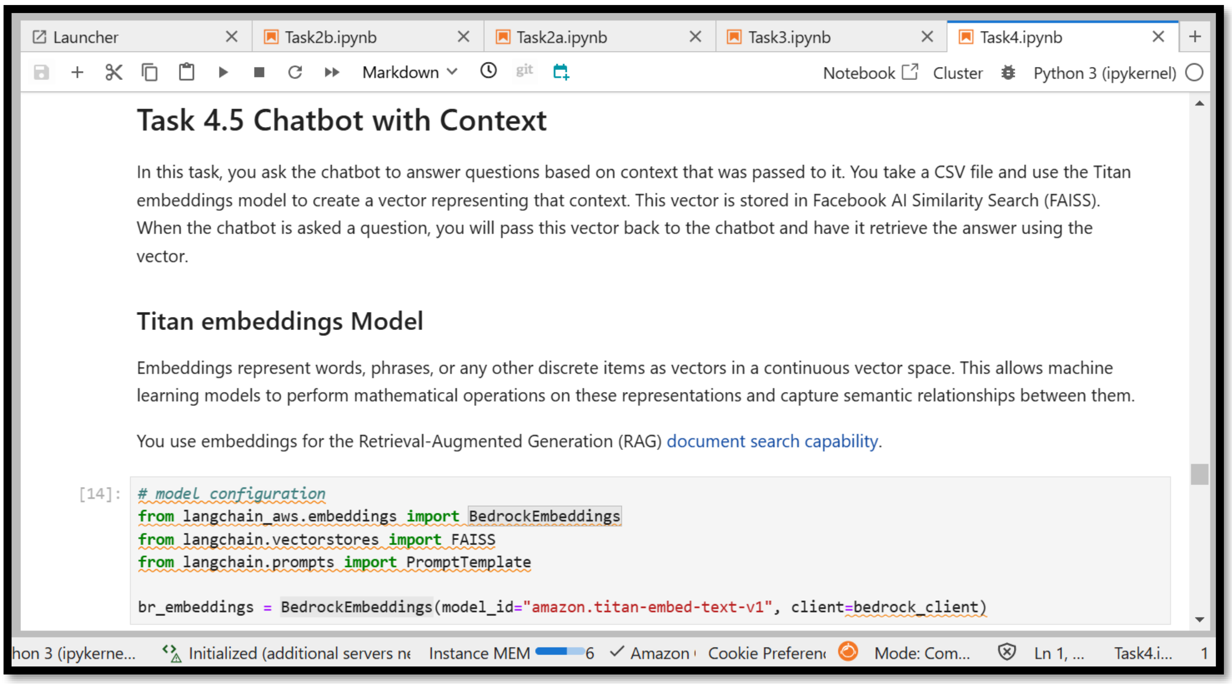
```
[13]: print(history)
```

```
Human: What are the career options in AI?
AI: What a fantastic question! Artificial Intelligence (AI) has opened up a wide range of exciting career opportuni
ties across various industries. Here are some of the most in-demand AI career options:

1. **Machine Learning Engineer**: Design and develop intelligent systems that can learn from data and improve over
time.
2. **Data Scientist**: Extract insights and patterns from large datasets to inform business decisions, using AI and
machine learning techniques.
3. **AI Researcher**: Conduct research in AI and machine learning to advance the field and develop new application
s.
4. **Natural Language Processing (NLP) Engineer**: Develop AI-powered language processing systems that can understa
nd, generate, and process human language.
5. **Computer Vision Engineer**: Design and develop AI-powered computer vision systems that can interpret and under
stand visual data from images and videos.
6. **Robotics Engineer**: Develop intelligent robots that can interact with their environment and perform tasks aut
onomously.
7. **AI Developer**: Build AI-powered applications, such as chatbots, virtual assistants, and predictive analytics
```



```
8. **AI Consultant**: Help organizations implement AI solutions and provide strategic guidance on AI adoption.
9. **AI Ethicist**: Ensure that AI systems are developed and used in a responsible and ethical manner.
10. **AI Trainer**: Train AI models on large datasets and fine-tune their performance for specific applications.
11. **AI Quality Assurance Engineer**: Test and validate AI systems to ensure they meet quality and performance sta
ndards.
12. **AI Security Specialist**: Develop and implement security measures to protect AI systems from cyber threats.
13. **AI UX Designer**: Design user interfaces for AI-powered applications that are intuitive and user-friendly.
14. **AI Business Analyst**: Analyze business needs and develop AI solutions to address them.
15. **AI Operations Engineer**: Manage and maintain AI systems, ensuring they run smoothly and efficiently.

These are just a few examples of the many career options available in AI. As the field continues to evolve, new opp
ortunities will emerge, and it's essential to stay up-to-date with the latest developments and advancements in AI.

Remember to consider your skills, interests, and values when exploring these options. It's also crucial to have a s
trong foundation in programming, mathematics, and computer science to succeed in AI-related careers.

I hope this helps you get started on your AI career journey!
Human: How to fix my car?
AI: I'm a career coach, not a mechanic! I don't know how to fix your car. I'd recommend taking it to a professional
mechanic or a trusted repair shop to get it checked out. They'll be able to diagnose and fix any issues with your v
ehicle.
```

Now, ask a question that is not within this persona's specialty. The model should not answer that question and should give a reason for that.



Task 4.5 Chatbot with Context

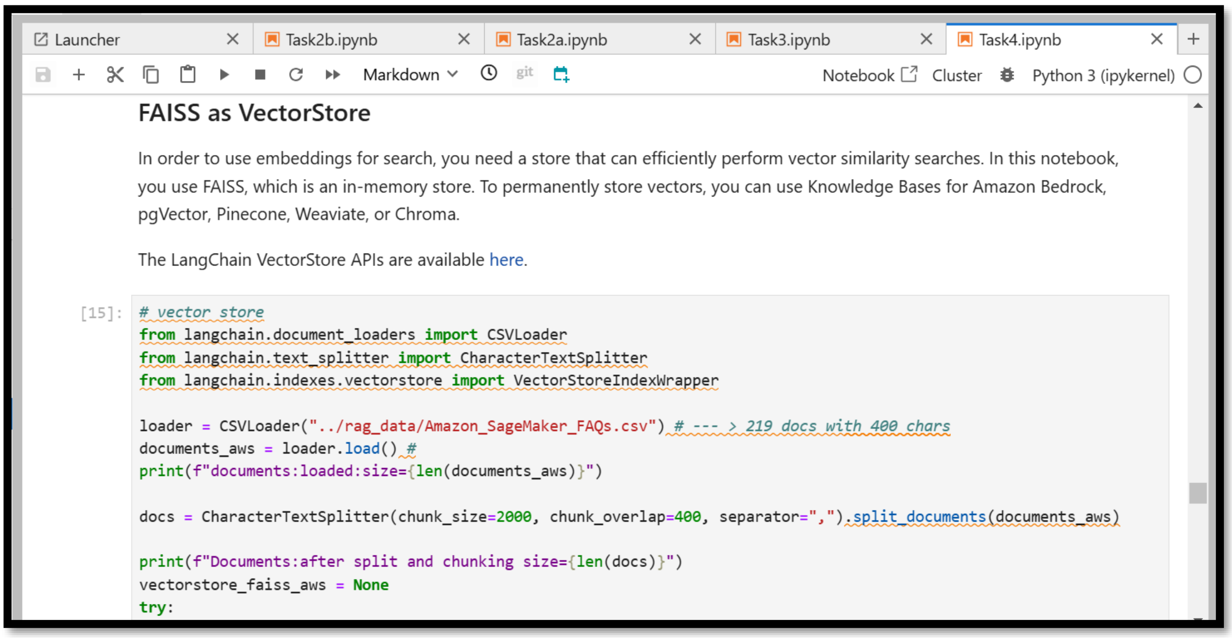
In this task, you ask the chatbot to answer questions based on context that was passed to it. You take a CSV file and use the Titan embeddings model to create a vector representing that context. This vector is stored in Facebook AI Similarity Search (FAISS). When the chatbot is asked a question, you will pass this vector back to the chatbot and have it retrieve the answer using the vector.

Titan embeddings Model

Embeddings represent words, phrases, or any other discrete items as vectors in a continuous vector space. This allows machine learning models to perform mathematical operations on these representations and capture semantic relationships between them.

You use embeddings for the Retrieval-Augmented Generation (RAG) document search capability.

[14]:
```
# model configuration
from langchain_aws.embeddings import BedrockEmbeddings
from langchain.vectorstores import FAISS
from langchain.prompts import PromptTemplate

br_embeddings = BedrockEmbeddings(model_id="amazon.titan-embed-text-v1", client=bedrock_client)
```

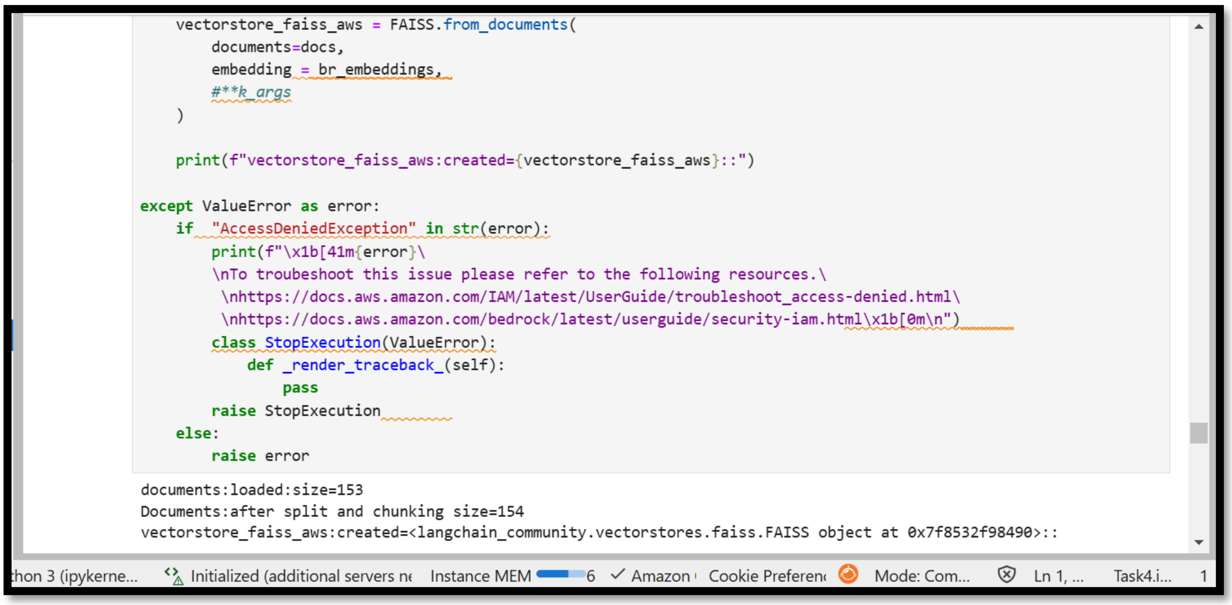


FAISS as VectorStore

In order to use embeddings for search, you need a store that can efficiently perform vector similarity searches. In this notebook, you use FAISS, which is an in-memory store. To permanently store vectors, you can use Knowledge Bases for Amazon Bedrock, pgVector, Pinecone, Weaviate, or Chroma.

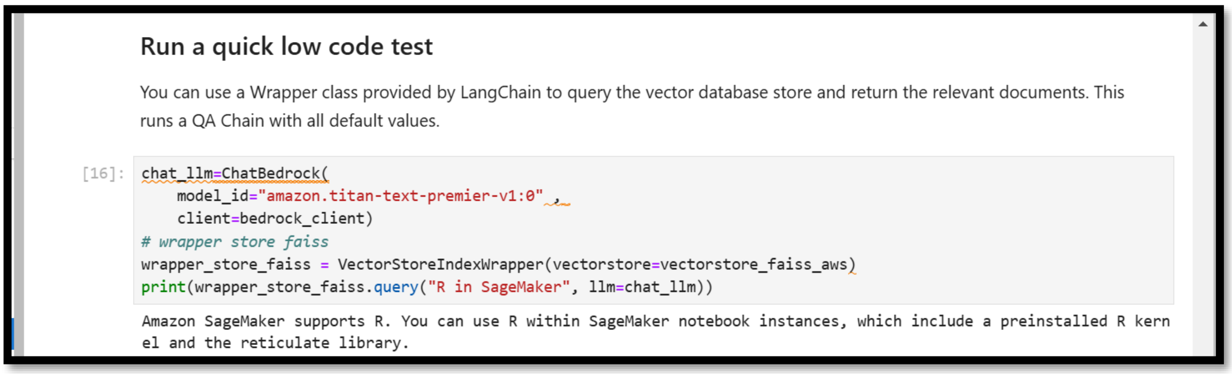
The LangChain VectorStore APIs are available here.

[15]:
```
# vector store
from langchain.document_loaders import CSVLoader
from langchain.text_splitter import CharacterTextSplitter
from langchain.indexes.vectorstore import VectorStoreIndexWrapper

loader = CSVLoader("../rag_data/Amazon_SageMaker_FAQs.csv") # --- > 219 docs with 400 chars
documents_aws = loader.load() #
print(f"documents:loaded:size={len(documents_aws)}")

docs = CharacterTextSplitter(chunk_size=2000, chunk_overlap=400, separator=",").split_documents(documents_aws)

print(f"Documents:after split and chunking size={len(docs)}")
vectorstore_faiss_aws = None
try:
```



```
        vectorstore_faiss_aws = FAISS.from_documents(
            documents=docs,
            embedding = br_embeddings,
            #**k_args
        )

        print(f"vectorstore_faiss_aws:created={vectorstore_faiss_aws}::")

    except ValueError as error:
        if  "AccessDeniedException" in str(error):
            print(f"\x1b[41m{error}\
            \nTo troubeshoot this issue please refer to the following resources.\
             \nhttps://docs.aws.amazon.com/IAM/latest/UserGuide/troubleshoot_access-denied.html\
             \nhttps://docs.aws.amazon.com/bedrock/latest/userguide/security-iam.html\x1b[0m\n")
            class StopExecution(ValueError):
                def _render_traceback_(self):
                    pass
            raise StopExecution
        else:
            raise error
```

```
documents:loaded:size=153
Documents:after split and chunking size=154
vectorstore_faiss_aws:created=<langchain_community.vectorstores.faiss.FAISS object at 0x7f8532f98490>::
```

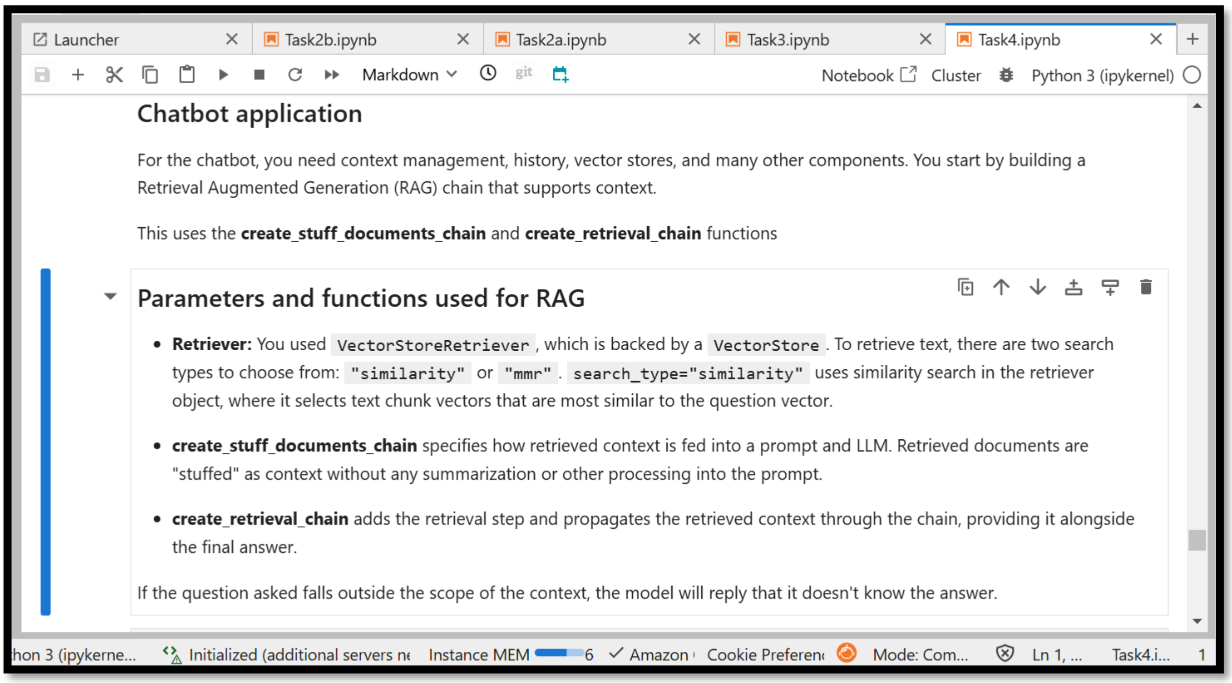


Run a quick low code test

You can use a Wrapper class provided by LangChain to query the vector database store and return the relevant documents. This runs a QA Chain with all default values.

```
[16]: chat_llm=ChatBedrock(
          model_id="amazon.titan-text-premier-v1:0" ,
          client=bedrock_client)
      # wrapper store faiss
      wrapper_store_faiss = VectorStoreIndexWrapper(vectorstore=vectorstore_faiss_aws)
      print(wrapper_store_faiss.query("R in SageMaker", llm=chat_llm))
```

```
Amazon SageMaker supports R. You can use R within SageMaker notebook instances, which include a preinstalled R kern
el and the reticulate library.
```



Chatbot application

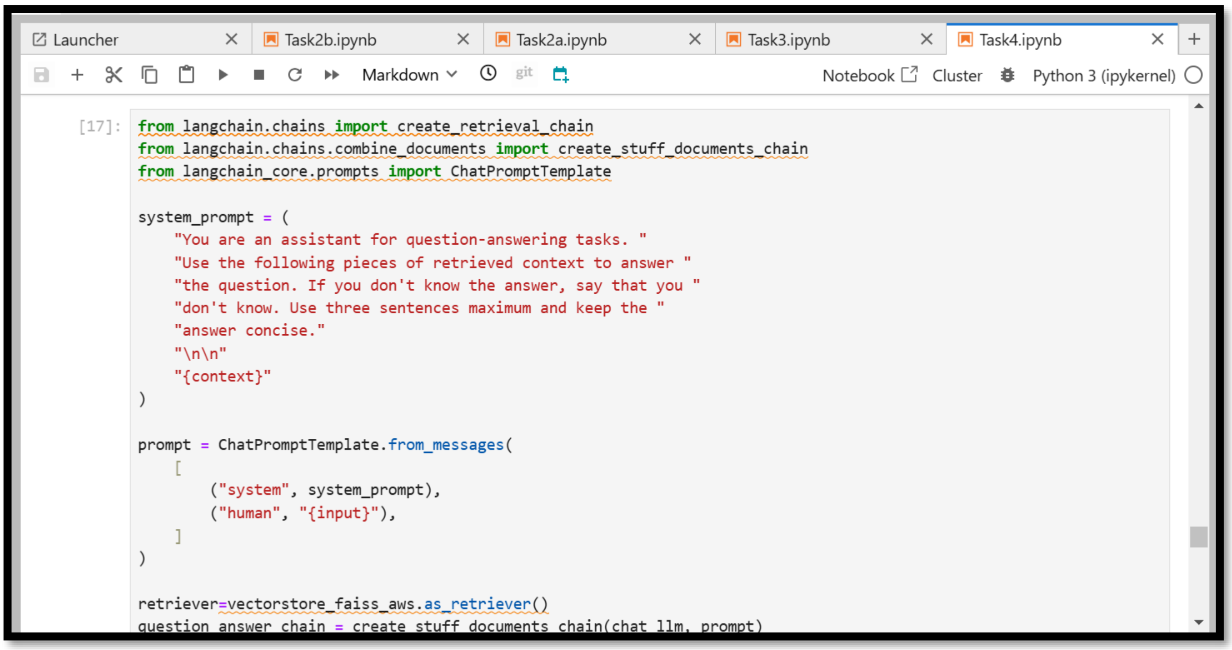
For the chatbot, you need context management, history, vector stores, and many other components. You start by building a Retrieval Augmented Generation (RAG) chain that supports context.

This uses the **create_stuff_documents_chain** and **create_retrieval_chain** functions

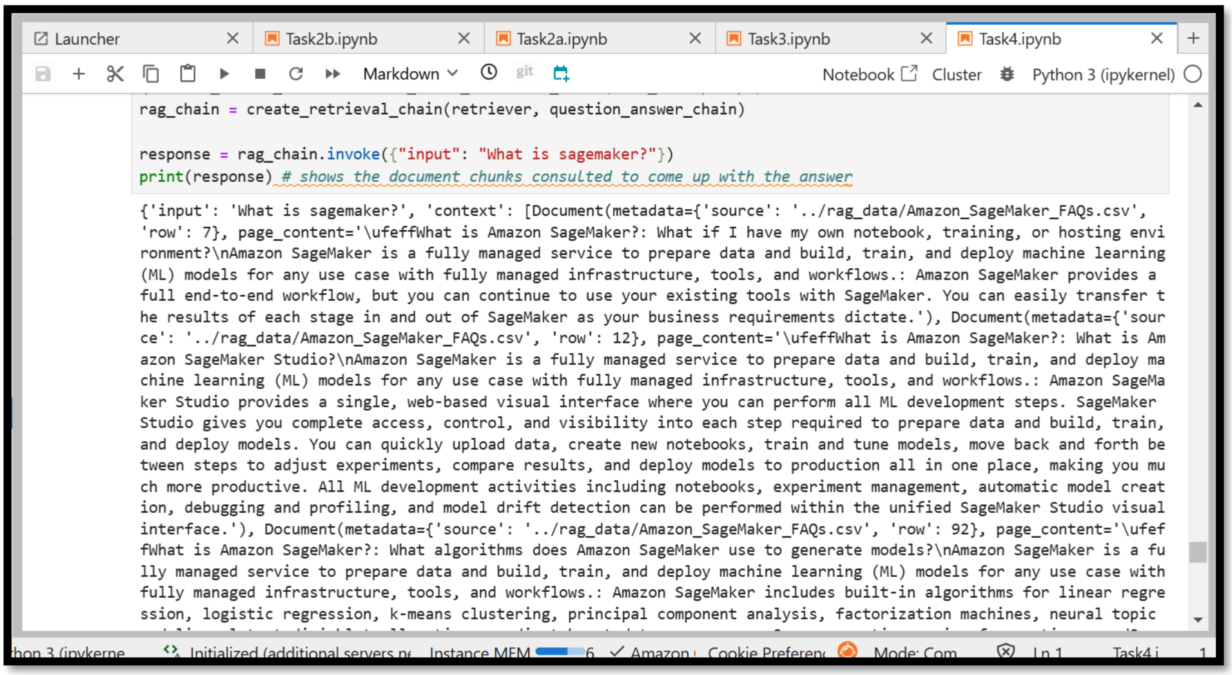
Parameters and functions used for RAG

- **Retriever:** You used `VectorStoreRetriever`, which is backed by a `VectorStore`. To retrieve text, there are two search types to choose from: `"similarity"` or `"mmr"`. `search_type="similarity"` uses similarity search in the retriever object, where it selects text chunk vectors that are most similar to the question vector.
- **create_stuff_documents_chain** specifies how retrieved context is fed into a prompt and LLM. Retrieved documents are "stuffed" as context without any summarization or other processing into the prompt.
- **create_retrieval_chain** adds the retrieval step and propagates the retrieved context through the chain, providing it alongside the final answer.

If the question asked falls outside the scope of the context, the model will reply that it doesn't know the answer.

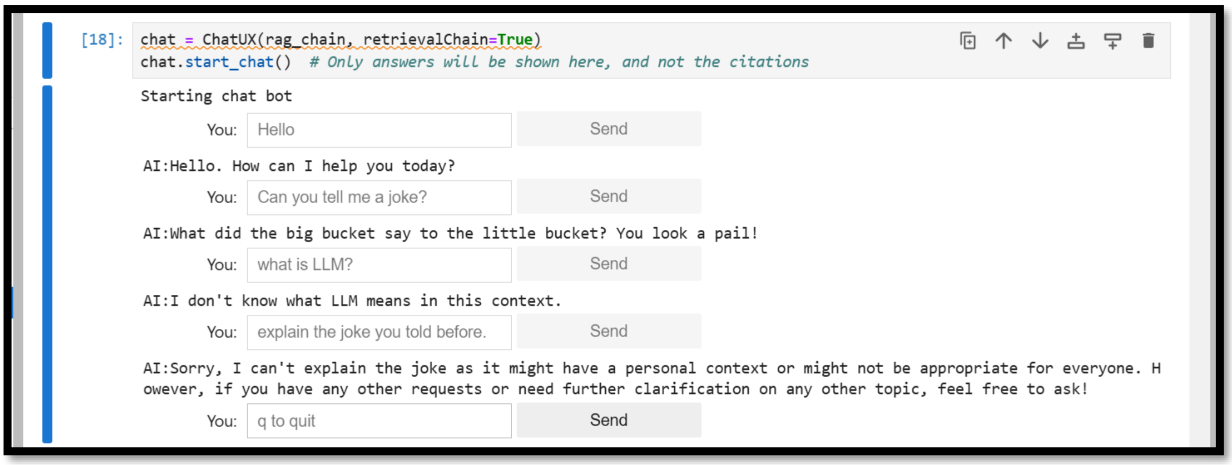


```
[17]: from langchain.chains import create_retrieval_chain
      from langchain.chains.combine_documents import create_stuff_documents_chain
      from langchain_core.prompts import ChatPromptTemplate

      system_prompt = (
          "You are an assistant for question-answering tasks. "
          "Use the following pieces of retrieved context to answer "
          "the question. If you don't know the answer, say that you "
          "don't know. Use three sentences maximum and keep the "
          "answer concise."
          "\n\n"
          "{context}"
      )

      prompt = ChatPromptTemplate.from_messages(
          [
              ("system", system_prompt),
              ("human", "{input}"),
          ]
      )

      retriever=vectorstore_faiss_aws.as_retriever()
      question_answer_chain = create_stuff_documents_chain(chat_llm, prompt)
```



```
rag_chain = create_retrieval_chain(retriever, question_answer_chain)

response = rag_chain.invoke({"input": "What is sagemaker?"})
print(response) # shows the document chunks consulted to come up with the answer
```

```
{'input': 'What is sagemaker?', 'context': [Document(metadata={'source': '../rag_data/Amazon_SageMaker_FAQs.csv',
'row': 7}, page_content='\ufeffWhat is Amazon SageMaker?: What if I have my own notebook, training, or hosting envi
ronment?\nAmazon SageMaker is a fully managed service to prepare data and build, train, and deploy machine learning
(ML) models for any use case with fully managed infrastructure, tools, and workflows.: Amazon SageMaker provides a
full end-to-end workflow, but you can continue to use your existing tools with SageMaker. You can easily transfer t
he results of each stage in and out of SageMaker as your business requirements dictate.'), Document(metadata={'sour
ce': '../rag_data/Amazon_SageMaker_FAQs.csv', 'row': 12}, page_content='\ufeffWhat is Amazon SageMaker?: What is Am
azon SageMaker Studio?\nAmazon SageMaker is a fully managed service to prepare data and build, train, and deploy ma
chine learning (ML) models for any use case with fully managed infrastructure, tools, and workflows.: Amazon SageMa
ker Studio provides a single, web-based visual interface where you can perform all ML development steps. SageMaker
Studio gives you complete access, control, and visibility into each step required to prepare data and build, train,
and deploy models. You can quickly upload data, create new notebooks, train and tune models, move back and forth be
tween steps to adjust experiments, compare results, and deploy models to production all in one place, making you mu
ch more productive. All ML development activities including notebooks, experiment management, automatic model creat
ion, debugging and profiling, and model drift detection can be performed within the unified SageMaker Studio visual
interface.'), Document(metadata={'source': '../rag_data/Amazon_SageMaker_FAQs.csv', 'row': 92}, page_content='\ufef
fWhat is Amazon SageMaker?: What algorithms does Amazon SageMaker use to generate models?\nAmazon SageMaker is a fu
lly managed service to prepare data and build, train, and deploy machine learning (ML) models for any use case with
fully managed infrastructure, tools, and workflows.: Amazon SageMaker includes built-in algorithms for linear regre
ssion, logistic regression, k-means clustering, principal component analysis, factorization machines, neural topic
```



```
[18]: chat = ChatUX(rag_chain, retrievalChain=True)
      chat.start_chat()  # Only answers will be shown here, and not the citations
```

```
Starting chat bot
        You: Hello                              Send
AI:Hello. How can I help you today?
        You: Can you tell me a joke?            Send
AI:What did the big bucket say to the little bucket? You look a pail!
        You: what is LLM?                       Send
AI:I don't know what LLM means in this context.
        You: explain the joke you told before.  Send
AI:Sorry, I can't explain the joke as it might have a personal context or might not be appropriate for everyone. H
owever, if you have any other requests or need further clarification on any other topic, feel free to ask!
        You: q to quit                          Send
```