# AI Conclave

Online

AIOT104

# Build generative AI applications with Amazon Bedrock and open source frameworks

**Vatsal Shah**
Principal Solutions Architect
AWS India

# Agenda

- Overview of Amazon Bedrock
- Key concepts in generative AI
- Popular open-source frameworks and their integration with Amazon Bedrock
- Hands-on demo: Building an intelligent travel assistant
- Do it yourself!

# Amazon Bedrock

The easiest way to build and scale generative AI applications with powerful tools and foundation models

Choice of leading FMs through a single API

Model customization

Retrieval Augmented Generation (RAG)

Agents that execute multistep tasks

Security, privacy, and data governance

# Strategies for implementation and their trade-offs

1

**Prompt engineering**

Using context instructions, examples and output indicators

2

**Retrieval Augmented Generation (RAG)**

Text generation based on specified corpus of data, to generate accurate responses with no hallucination

3

**Agentic systems**

Using LLMs to understand tasks, gather information, decide best path and execute actions

4

**Fine-tuning**

Give a model company or domain specific data, or instructions examples

5

**Re-training**

Give a model a different dataset, or building a model from scratch

**Increasing accuracy and complexity**

**Higher cost and latency**

# Amazon Bedrock Agents

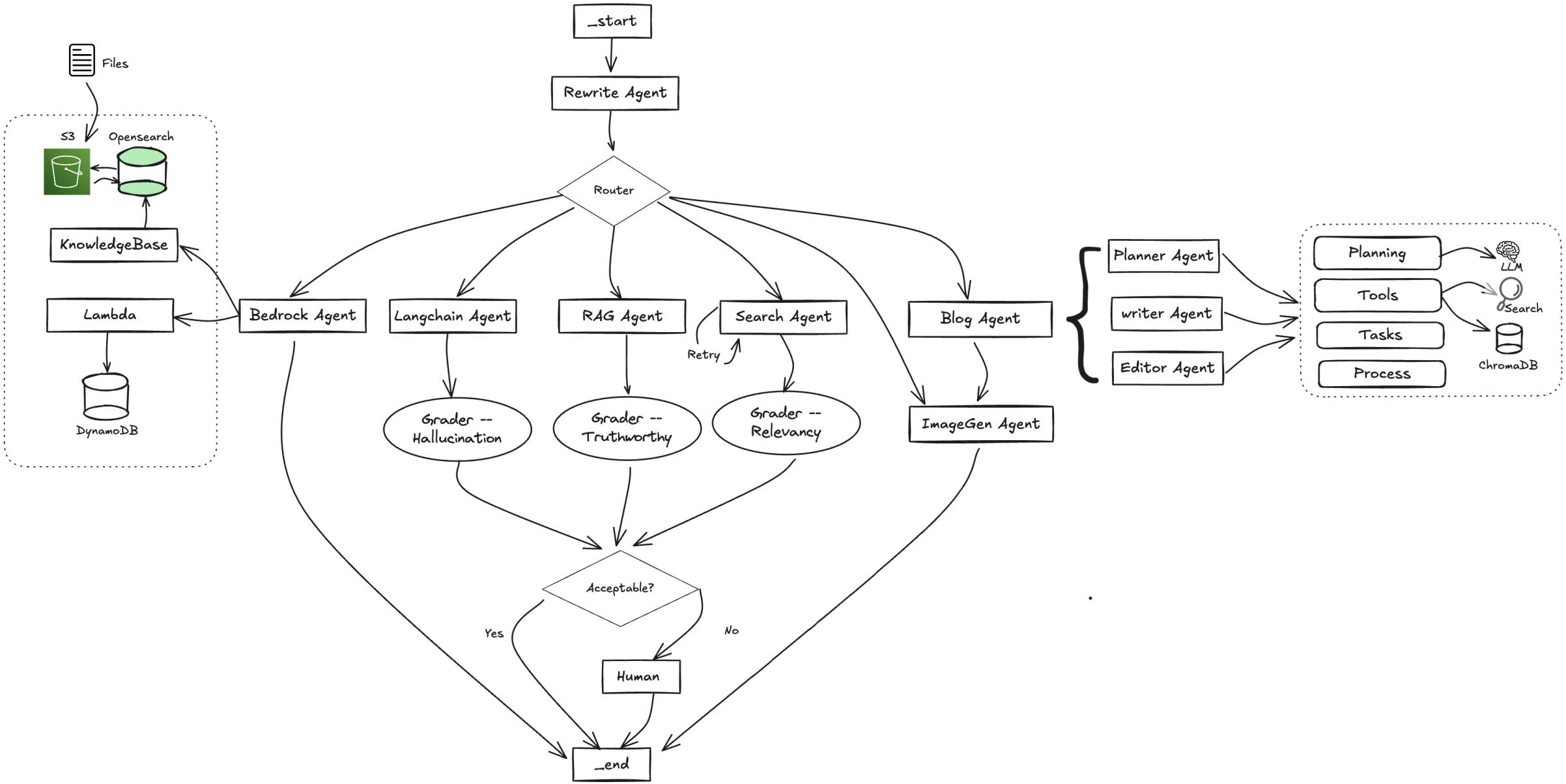
ENABLE GENERATIVE AI APPLICATIONS TO EXECUTE MULTISTEP TASKS USING COMPANY SYSTEMS AND DATA SOURCES


Breaks down and orchestrates tasks

Securely accesses and retrieves company data for RAG

Takes action by invoking API calls on your behalf

Chain-of-thought trace and ability to modify agent prompts


_start
Rewrite Agent
Router
Files
S3
Opensearch
KnowledgeBase
Lambda
DynamoDB
Bedrock Agent
Langchain Agent
RAG Agent
Search Agent
Retry
Blog Agent
Grader -- Hallucination
Grader -- Truthworthy
Grader -- Relevancy
ImageGen Agent
Acceptable?
Yes
No
Human
_end
Planner Agent
writer Agent
Editor Agent
Planning
Tools
Tasks
Process
LLM
Search
ChromaDB

# LangGraph

FEATURES

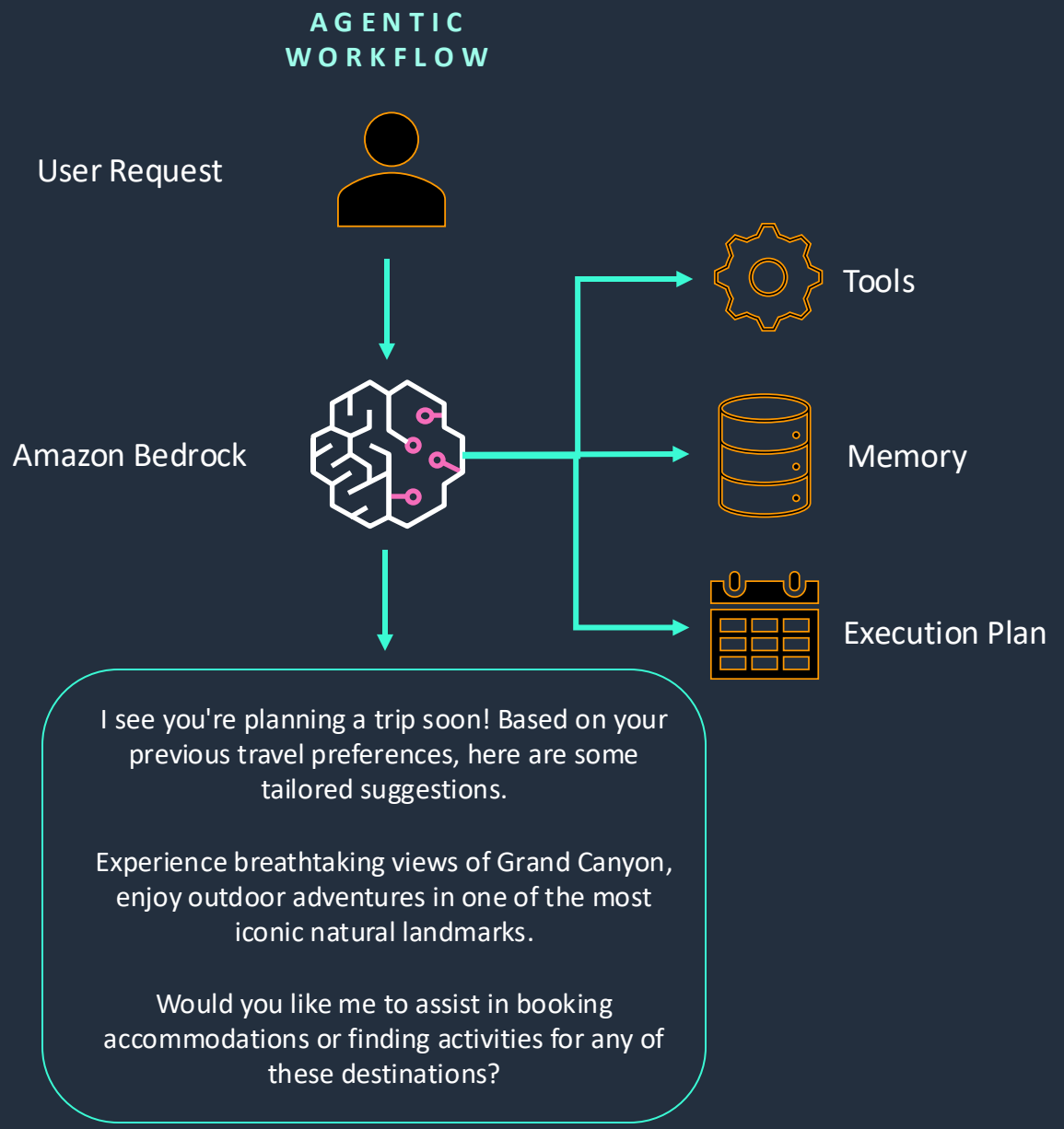

- **State**
  - A shared data structure that represents the current snapshot of your application
- **Node**
  - Python functions that encode the logic of your agents. They receive the current State as input, perform some computation
- **Edge**
  - Python functions that determine which Node to execute next based on the current State

- **Memory**
  - Allows users to store, retrieve , use and learn from feedback
  - Short-term memory has scope from within a single conversation
  - Long-term can be recalled at any time in any thread
- **Agentic Patterns**
  - Router
  - Tool calling agent
  - Multi-agent systems
  - Guardrails

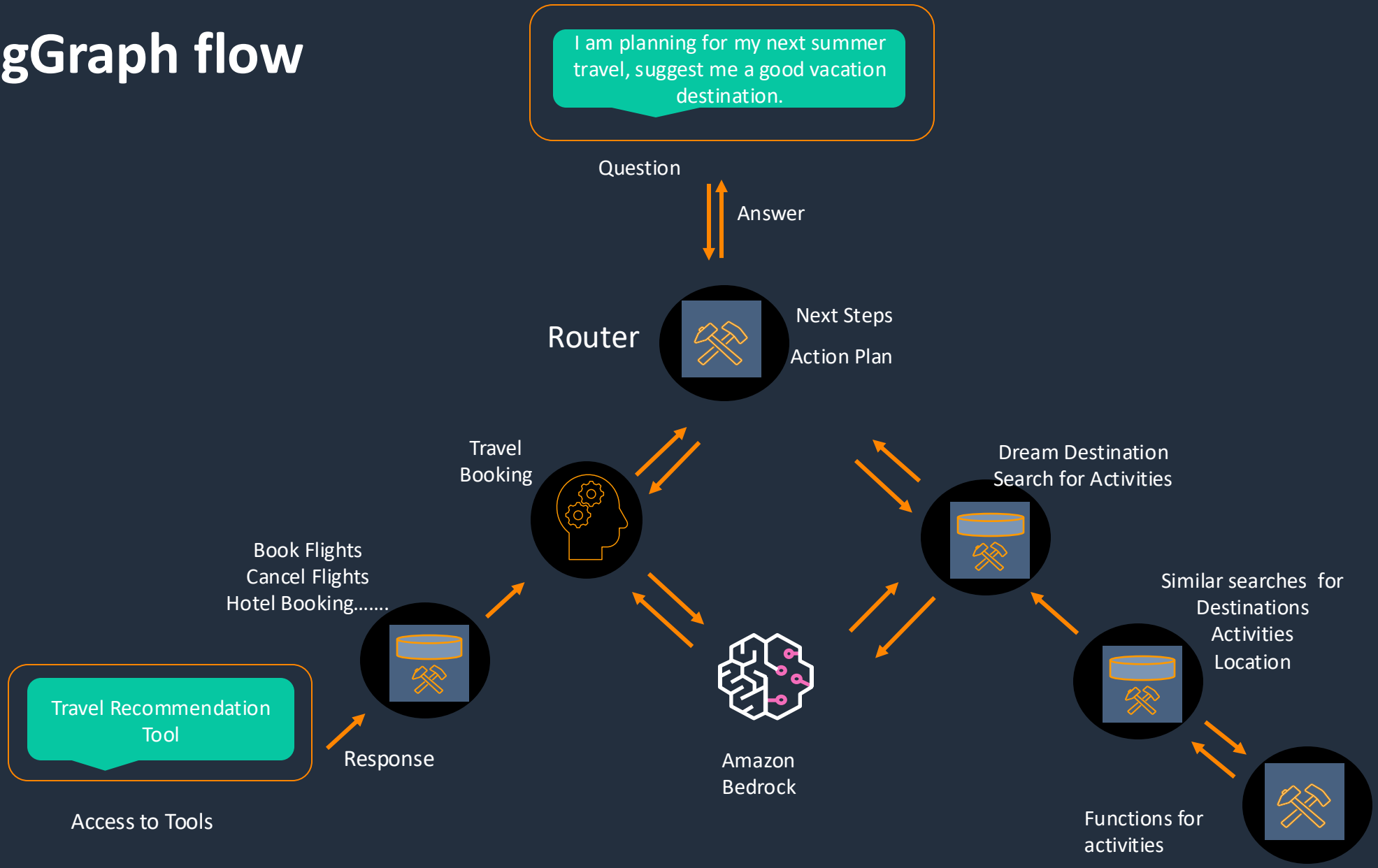
# Agentic Workflow use case

**TRAVEL PLAN**
**"PERSONALIZED EXPERIENCE"**

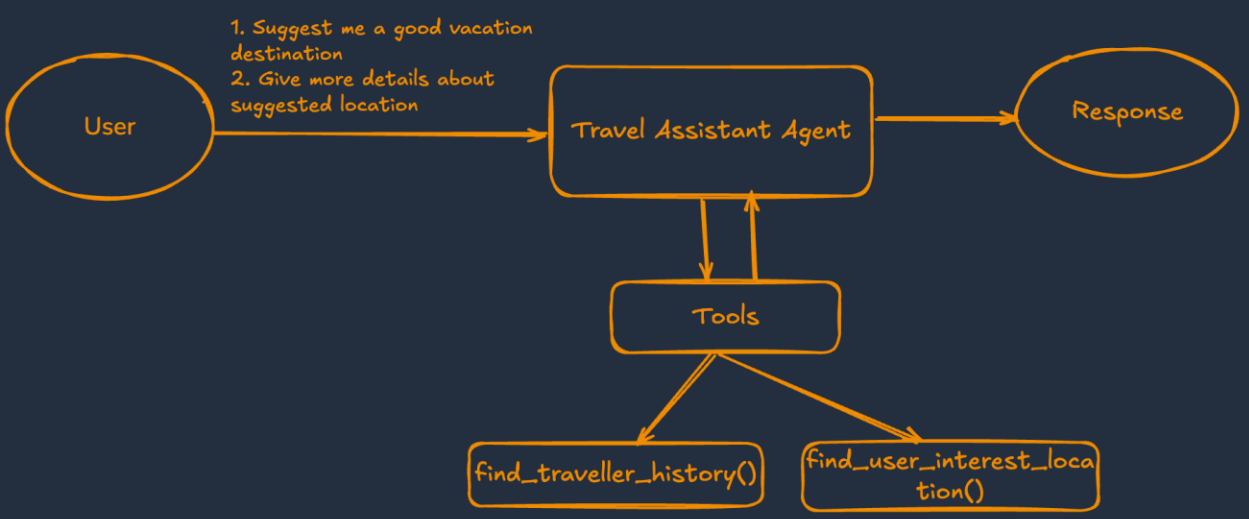
I am planning my next summer vacation with my family, would like to visit a national park in the US.
Can you suggest an end-to-end trip itinerary, activity plan and assist with flights and hotel booking?

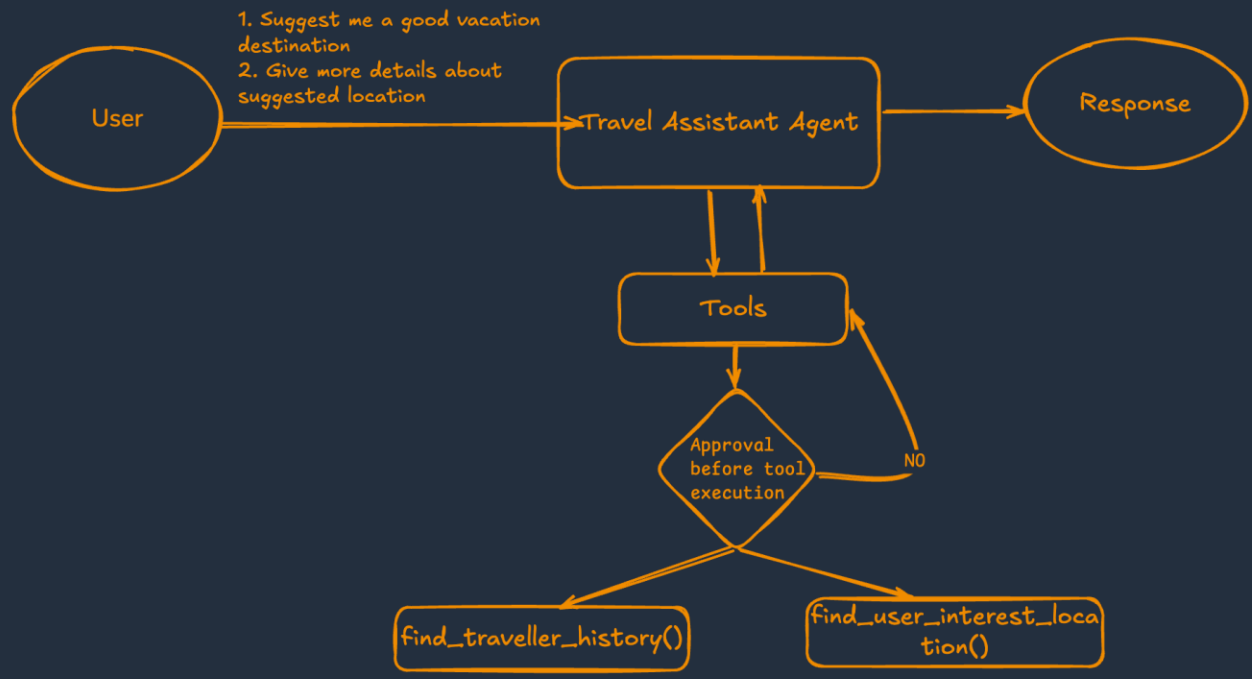


# LangGraph flow



# Tools



# Amazon Bedrock Converse API

```python
from langchain_aws import ChatBedrockConverse

llm = ChatBedrockConverse(
    model = "anthropic.claude-3-haiku-20240307-v1:0",
    temperature=0,
    max_tokens=None,
    client=bedrock_client,
    # other params...
)
```

# LangGraph nodes and edges

```python
def input_interests(state: PlannerState) -> PlannerState:
    user_message = state['user_message'] #input("Your input: ")
    #print(f"We are going to :: {user_message}:: for trip to {state['city']} based on your interests mentioned ir

    if not state.get('messages', None) : state['messages'] = []
    return {
        **state,
    }

def create_itinerary(state: PlannerState) -> PlannerState:
    response = llm.invoke(itinerary_prompt.format_messages(city=state['city'], user_message=state['user_message']
    print("\nFinal Itinerary:")
    print(response.content)
    return {
        **state,
        "messages": state['messages'] + [HumanMessage(content=state['user_message']), AIMessage(content=response.
        "itinerary": response.content
    }
```

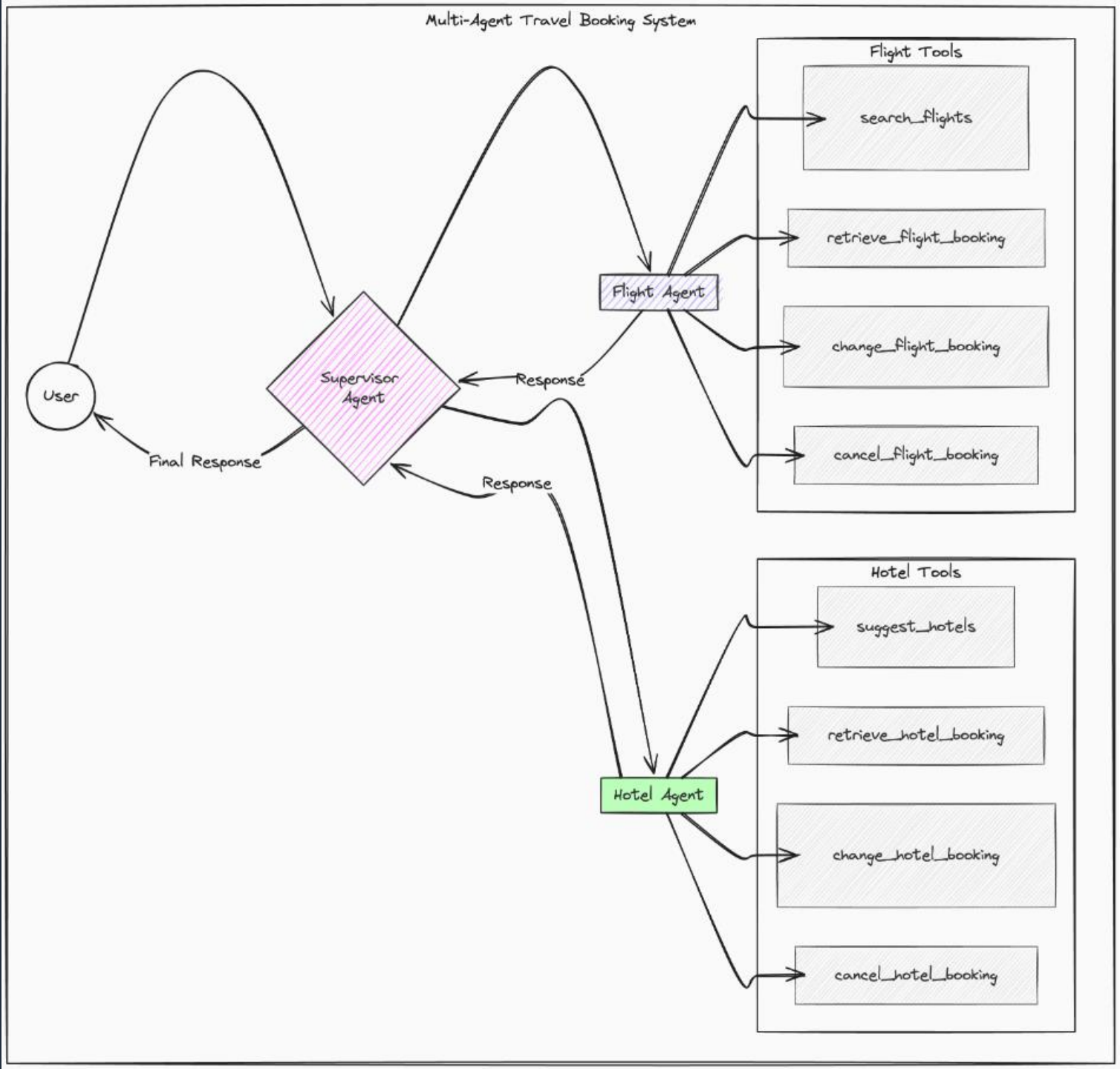
```python
workflow = StateGraph(PlannerState)

#workflow.add_node("input_city", input_city)
workflow.add_node("input_interests", input_interests)
workflow.add_node("create_itinerary", create_itinerary)

workflow.set_entry_point("input_interests")

#workflow.add_edge("input_city", "input_interests")
workflow.add_edge("input_interests", "create_itinerary")
workflow.add_edge("create_itinerary", END)

# The checkpointer lets the graph persist its state
# this is a complete memory for the entire graph.
memory = MemorySaver()
app = workflow.compile(checkpointer=memory)
```

# Creating a tool – Document search example

```python
from langchain_aws.embeddings.bedrock import BedrockEmbeddings
from langchain_community.vectorstores import FAISS
from langchain.retrievers import ParentDocumentRetriever
from langchain.text_splitter import RecursiveCharacterTextSplitter

# import faiss
from io import BytesIO
import pickle

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

child_splitter = RecursiveCharacterTextSplitter(
    separators=["\n", "\n\n"], chunk_size=2000, chunk_overlap=250
)

in_memory_store_file = "data/section_doc_store.pkl"
vector_store_file = "data/section_vector_store.pkl"

store = pickle.load(open(in_memory_store_file, "rb"))
vector_db_buff = BytesIO(pickle.load(open(vector_store_file, "rb")))
vector_db = FAISS.deserialize_from_bytes(
    serialized=vector_db_buff.read(),
    embeddings=embeddings_model,
    allow_dangerous_deserialization=True,
)

retriever = ParentDocumentRetriever(
    vectorstore=vector_db,
    docstore=store,
    child_splitter=child_splitter,
)
```

```python
from langchain.tools.retriever import create_retriever_tool

retriever_tool = create_retriever_tool(
    retriever,
    "search_user_interest",
    "Searches through multiple PDF documents containing city
)
```

# Multi agents using LangGraph



# Open source agent frameworks

| Criteria | LangGraph | Crew AI |
|---|---|---|
| Architecture | Graph-based architecture | Role-based architecture |
| Integrations and tool support | Part of LangChain framework, extensive tool support options | Built on top of LangChain, integrated with LangChain-based tools |
| Memory and Context Management | Long-term, short-term and contextual memory with time travel features for debugging | Supports wide range of memory options to maintain context across multiple agents |
| Other Features | Excels in stateful applications | Fit for collaborative AI teams |
| Ease Of Use | Provide granular control, has steep learning curve | User-friendly for beginners |
| Flexibility and Customization | Fine-grained control on workflows, fit for complex, stateful apps | Less flexible with abstracted customization |

# Ragas

SUPERCHARGE YOUR LLM APPLICATION EVALUATIONS 🚀


**Objective metrics:** Evaluate Gen AI applications using both **LLM-based** and **traditional metrics**

**Test data generation:** Automatically create comprehensive test datasets

**Seamless integrations:** Works flawlessly with **Amazon Bedrock** and popular generative AI frameworks like **LangChain**

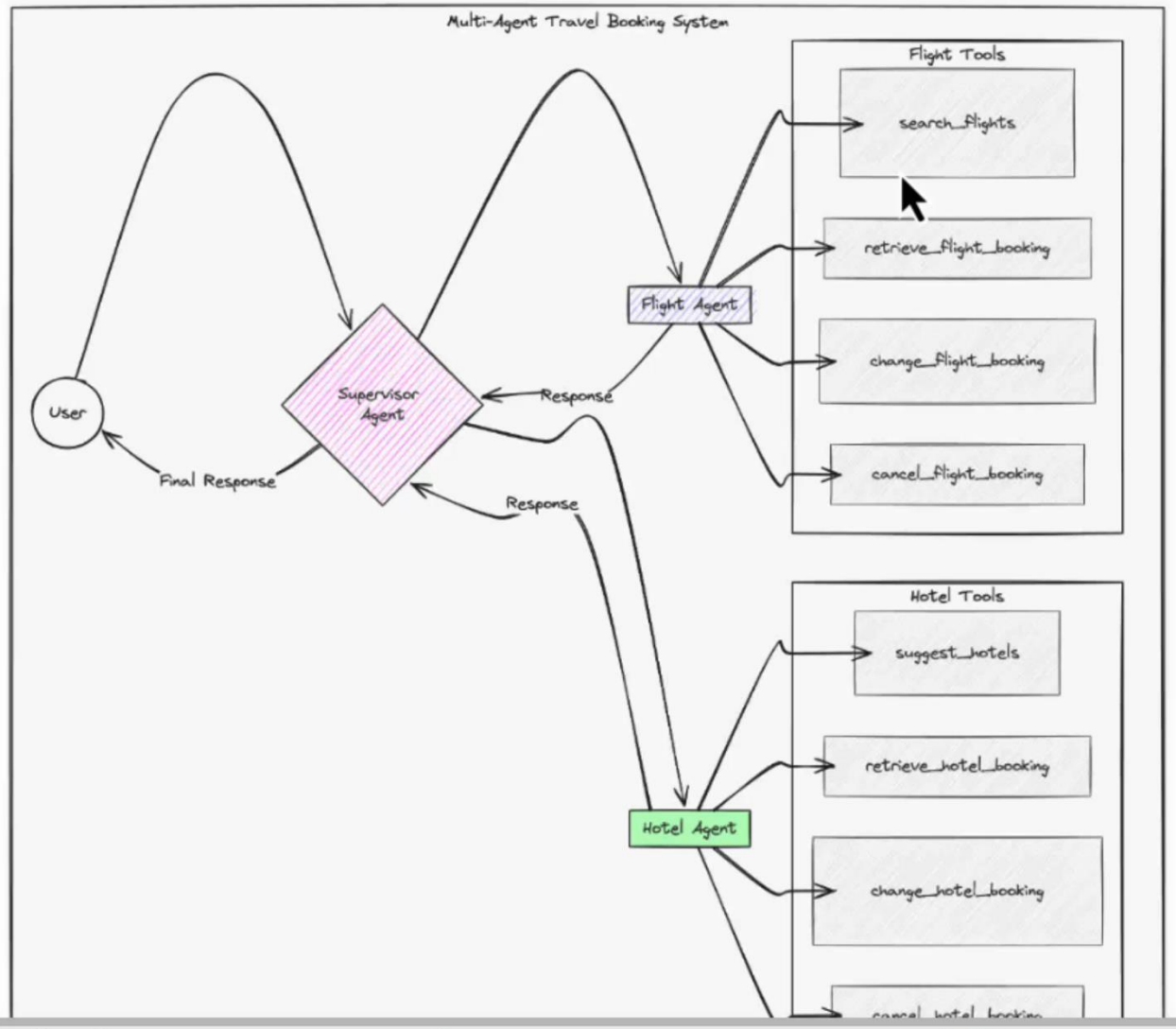
**Build feedback loops:** Leverage production data to **continually improve** your Gen AI applications

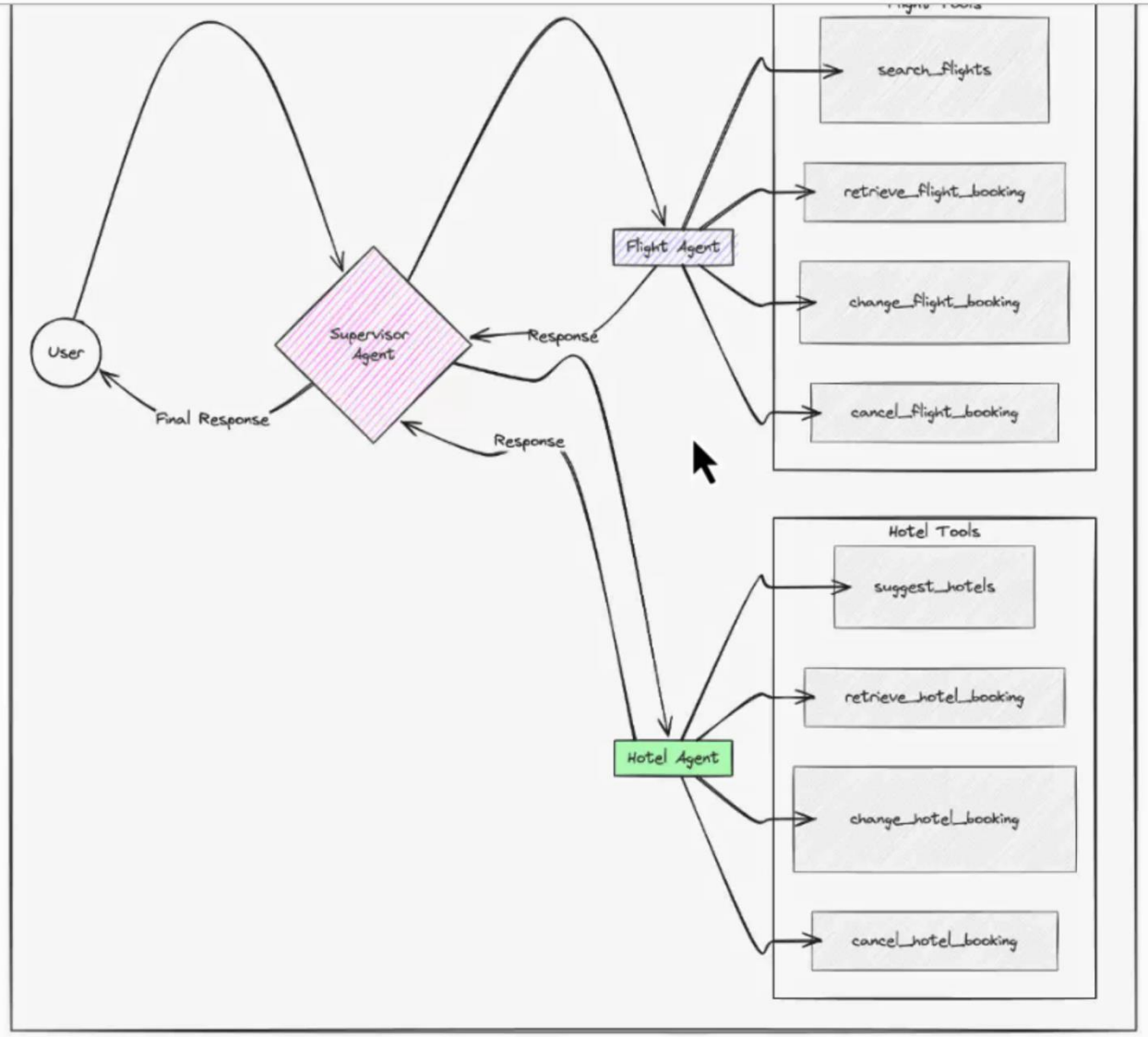
```
pip install ragas
```

# Demo – Let’s build

## Scenario

The below image shows the tools and the flow of data and Control as we progress with our `Travel Assistant bot`






## Flight agent

```
ret_messages = flight_agent.invoke({"messages": [("user", "Find flight to Amsterdam")]}, config)
ret_messages['messages'][-1].pretty_print()

#- un coment i you want to see the full orchesteration including the tool calling
#ret_messages
```

```
================================== Ai Message ==================================

The search results show several flight options from Nice to Amsterdam on January 10, 2025. The flights range in price from 212 to 378 euros and have a duration of 2 hou
et me know if you would like me to book one of these flights for you.
```

# Hotel Agent

Just like flight agent we need to create few tools, which can manage hotel bookings. We will use the same approach as we did with flight agents.

The Hotel Agent will be responsible for handling various hotel-related tasks, including:

1. Suggesting hotels based on city and check-in date
2. Retrieving hotel booking details
3. Modifying existing hotel bookings
4. Cancelling hotel reservations

These functionalities will be implemented as separate tools, similar to the Flight Agent. The Hotel Agent will use these tools to interact with a simulated hotel booking system.

## Suggest hotel tool

The `suggest_hotels` function is a tool designed to suggest hotels based on city and check-in date. It takes in a city name (e.g., "New York") and a check-in date (e.g., 2019-08-30) as input, and returns a list of suggested hotel names.

**Purpose**: This tool simulates a hotel booking system that suggests hotels based on city and check-in date.

**Note**: This function is designed for demonstration and testing purposes, using randomly generated data rather than real information from hotel booking system.

```
@tool
```

```
================================ Human Message =================================

cancel my hotel booking id 203
================================== Ai Message ==================================

[{'type': 'text', 'text': "Okay, let's cancel your hotel booking with ID 203."}, {'type': 'tool_use', 'name': 'cancel_hotel_booking', 'input': {'booking_id': 203}, 'id'
oluse_FCLq5c7HSxGmvQoU51lgVA'}]
Tool Calls:
  cancel_hotel_booking (tooluse_FCLq5c7HSxGmvQoU51lgVA)
 Call ID: tooluse_FCLq5c7HSxGmvQoU51lgVA
  Args:
    booking_id: 203
Do you approve of the above actions? Type 'y' to continue; otherwise, explain your requested changed.

 y
================================= Tool Message =================================
Name: cancel_hotel_booking

"No booking found with ID: 203 FINISHED"
================================== Ai Message ==================================

It looks like there is no booking found with ID 203. I was unable to cancel the booking, as it does not seem to exist in the system.
```

## Supervisor agent

Now its time to create supervisor agent that will be in charge of deciding which child agent to call based on the user input and based on the conversation history.

The Supervisor Agent is responsible for:

1. Analyzing the conversation history and user input
2. Deciding which child agent (flight_agent or hotel_agent) to call next
3. Determining when to finish the conversation

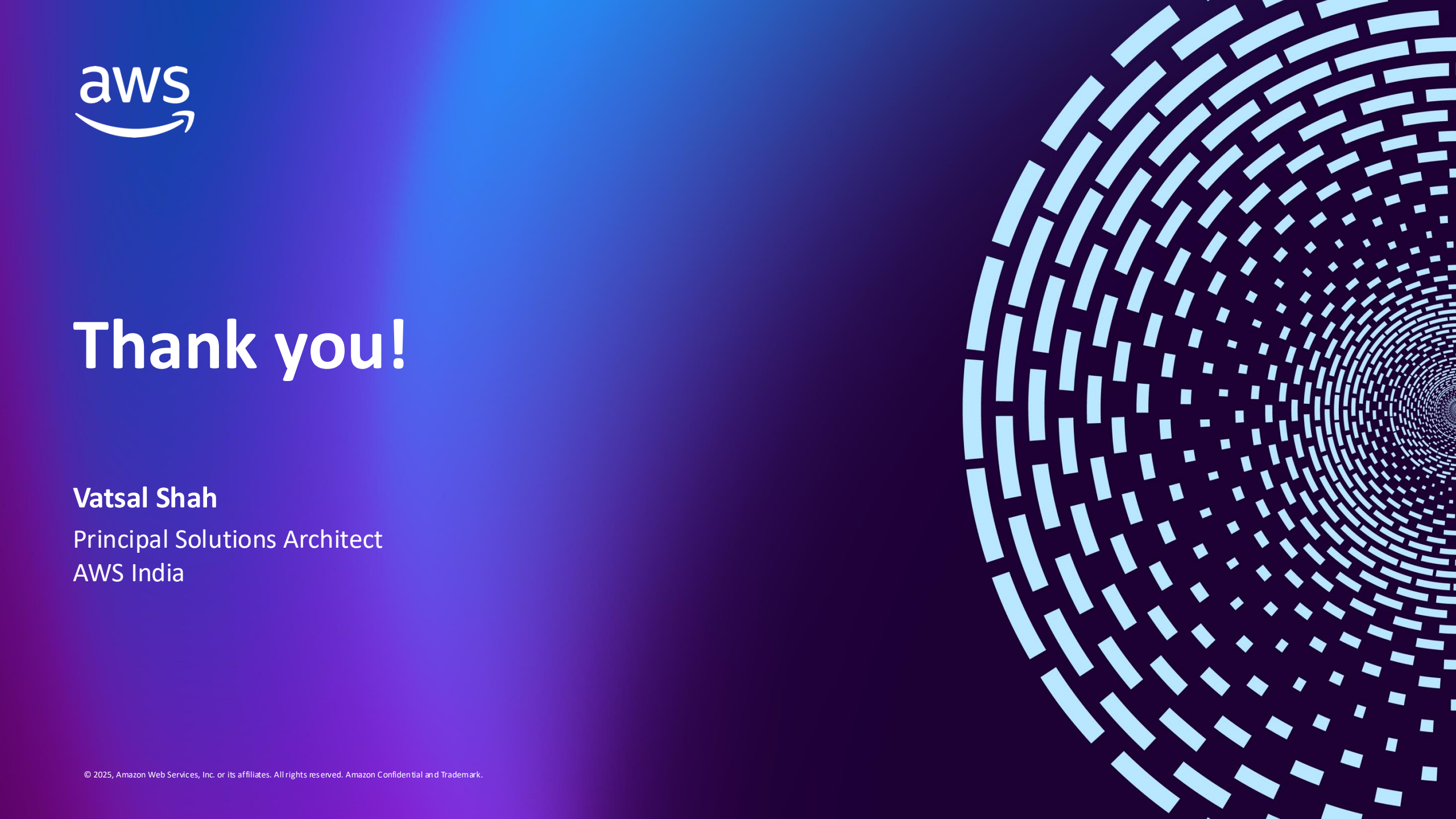
We will create this agent with LangChain runnable chain created using supervisor prompt. We need to get the `next_step` from the chain and we use `with_structured_output` to return next step.

The Supervisor Agent routes tasks and maintains the overall flow of the conversation between the user and child agents.

# Important resources

Workshop

Blog



# Thank you!

**Vatsal Shah**
Principal Solutions Architect
AWS India