1.Defining a set of agents with specialized capabilities and roles

A. Identify the Overall Objective:Determine the overarching goal or mission that the agents need to achieve. This helps in understanding what capabilities are required.
B. Determine Specialized Capabilities:List the distinct skills or capabilities needed to accomplish the objective.
Examples include:
Navigation: The ability to move through an environment efficiently.
Communication: The ability to send and receive messages with other agents or humans.Data Analysis: The capability to process and interpret data.
Resource Management: Skills in allocating and utilizing resources effectively.
Decision Making: The ability to make choices based on available information
C.Define Agent Roles:
Assign roles to agents based on the specialized capabilities identified. Each role should focus on one or a combination of these capabilities.
For instance:
Navigator: Specializes in pathfinding and movement.
Communicator: Handles interactions with other agents or humans.
Analyst: Processes and interprets data.
Manager: Allocates and manages resources.
Strategist: Makes high-level decisions based on situational awareness.
D.Specify Agent Interactions:Define how agents will interact with each other to achieve the overall objective. This includes:
Communication Protocols: How agents exchange information (e.g., direct messaging, broadcasting).
Coordination Mechanisms: How agents work together (e.g., task delegation, synchronization).
E.Design Agent Behaviors: Outline the behaviors and actions each type of agent will perform. This involves creating rules or algorithms that dictate:
Reactive Behaviors: How agents respond to changes in their environment.
Proactive Behaviors: How agents plan and act to achieve their goals.
F.Implement and Test:
Develop the agents using suitable programming languages and tools. Test the system to ensure agents perform their roles effectively and cooperate as intended.
Example Scenario: Autonomous Delivery System
Objective: Efficiently deliver packages within a city.
Specialized Capabilities:
Navigation: Finding and following the best routes.
Communication: Updating status and receiving instructions.
Package Handling: Picking up, transporting, and dropping off packages.
Obstacle Detection: Identifying and avoiding obstacles.
Agent Roles:
Delivery Bot: Specializes in navigation and package handling.
Control Center Agent: Manages communications and provides routing instructions.
Maintenance Bot: Focuses on monitoring and maintaining delivery bots.

Agent Interactions:
Delivery Bot ↔ Control Center Agent: Regular updates on location and status, receiving new delivery assignments.
Maintenance Bot ↔ Delivery Bot: Status checks and performing repairs as needed.
Behaviors:
Delivery Bot:
Reactive: Avoid obstacles detected in real-time.
Proactive: Follow the assigned route to deliver packages.
Control Center Agent:
Reactive: Update routes based on traffic information.
Proactive: Assign new delivery tasks to idle delivery bots.
Maintenance Bot:
Reactive: Respond to maintenance requests.
Proactive: Regularly check on delivery bots' status.

2.lDefine Interaction Behaviour Between Agents:

A.Interaction behavior between agents involves specifying what responses or actions should occur when one agent receives a message from another. This can be done using a set of rules or a state machine.

Example:

Agent A sends a query to Agent B.
Agent B checks the message type:If the message is a data request, Agent B retrieves the requested data and sends a response.If the message is a task delegation, Agent B acknowledges and starts the task.If the message is a status update request, Agent B sends its current status.

Sample pseudocode:

```
class Agent:
    def __init__(self, name):
        self.name = name

    def receive_message(self, message):
        if message.type == "data_request":
            return self.handle_data_request(message)
        elif message.type == "task_delegation":
            return self.handle_task_delegation(message)
        elif message.type == "status_update":
            return self.send_status_update()
        else:
            return self.unknown_message()
```

```
    def handle_data_request(self, message):
        # Retrieve and return data
        pass

    def handle_task_delegation(self, message):
        # Acknowledge and start the task
        pass

    def send_status_update(self):
        # Return status
        pass

    def unknown_message(self):
        # Handle unknown messages
        pass
```

B.Integrate Local LLM from Hugging FaceIntegrating a local LLM (Language Model) from Hugging Face involves setting up the model and using it to generate responses or perform tasks.Installation:pip install transformers
pip install torchExample Code:from transformers import AutoModelForCausalLM, AutoTokenizer

```
class LLM_Agent(Agent):
    def __init__(self, name, model_name):
        super().__init__(name)
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.model = AutoModelForCausalLM.from_pretrained(model_name)

    def generate_response(self, prompt):
        inputs = self.tokenizer(prompt, return_tensors="pt")
        outputs = self.model.generate(inputs['input_ids'])
        return self.tokenizer.decode(outputs[0], skip_special_tokens=True)

# Initialize an LLM agent
llm_agent = LLM_Agent("Assistant", "gpt-neo-2.7B")

# Example usage
prompt = "What is the capital of France?"
response = llm_agent.generate_response(prompt)
print(response)
```

C.Set Up User Proxy Agent and Assistant AgentUser Proxy Agent: Acts as an intermediary between the user and other agents.Assistant Agent: Performs tasks and generates responses using the local LLM.

```
class UserProxyAgent(Agent):
    def __init__(self, name, assistant_agent):
        super().__init__(name)
        self.assistant_agent = assistant_agent

    def forward_user_query(self, query):
        response = self.assistant_agent.generate_response(query)
        return response

# Create an assistant agent with LLM capabilities
assistant_agent = LLM_Agent("Assistant", "gpt-neo-2.7B")

# Create a user proxy agent that forwards queries to the assistant agent
user_proxy_agent = UserProxyAgent("UserProxy", assistant_agent)

# Example
user_query = "Explain the theory of relativity."
response = user_proxy_agent.forward_user_query(user_query)
print(response)
```

D.Map External Function Calls to Agent ChatAgents may need to perform specific tasks by calling external functions. Define these functions and integrate them into the agent's behavior.Example External Function:def fetch_weather_data(location):
    # Simulate fetching weather data for a location
    return f"The weather in {location} is sunny."

```
class FunctionalAgent(Agent):
    def __init__(self, name):
        super().__init__(name)

    def receive_message(self, message):
        if message.type == "weather_request":
            return self.handle_weather_request(message)
        else:
            return super().receive_message(message)

    def handle_weather_request(self, message):
        location = message.content['location']
        weather_info = fetch_weather_data(location)
        return weather_info

# Example
functional_agent = FunctionalAgent("WeatherAgent")
weather_message = {"type": "weather_request", "content": {"location": "New York"}}
```

```
response = functional_agent.receive_message(weather_message)
print(response)
```

E.Retrieve Agents with Task QA and Code as RAG:
Retrieval-Augmented Generation (RAG) involves combining retrieval of relevant documents with a generative model to provide accurate responses

Example:

```
from transformers import RagTokenizer, RagRetriever, RagSequenceForGeneration

class RAG_Agent(Agent):
    def __init__(self, name, retriever_model_name, generator_model_name):
        super().__init__(name)
        self.tokenizer = RagTokenizer.from_pretrained(retriever_model_name)
        self.retriever = RagRetriever.from_pretrained(retriever_model_name)
        self.model = RagSequenceForGeneration.from_pretrained(generator_model_name)

    def generate_response(self, prompt):
        inputs = self.tokenizer(prompt, return_tensors="pt")
        retrieved_docs = self.retriever(inputs['input_ids'])
        outputs = self.model.generate(input_ids=inputs['input_ids'],
context_input_ids=retrieved_docs['context_input_ids'])
        return self.tokenizer.batch_decode(outputs, skip_special_tokens=True)

# Initialize a RAG agent
rag_agent = RAG_Agent("RAG_Assistant", "facebook/rag-token-nq",
"facebook/rag-sequence-nq")

# Example
prompt = "Who developed the theory of relativity?"
response = rag_agent.generate_response(prompt)
print(response)
```