Chapter_04

September 5, 2025

0.1 # Chapter 4 – Reflection and Introspection in Agents

Install dependencies

[ ]:
```
!pip install -U openai ipywidgets crewai pysqlite3-binary
```

0.2 # 1. Meta Reasoning - example

What is Meta-reassioning?

Meta-reasoning refers to the processes that monitor and control reasoning activities, allowing agents to reflect upon their own reasoning processes and make adjustments where appropriate. In the context of a reflective travel agent, meta-reasoning plays a crucial role in enabling the agent to continuously evaluate and refine its decision-making processes.

Let's take a look at a simple meta-reasoning approach without AI.

[2]:
```
import random
```

0.3 Simulated travel agent with meta-reasoning capabilities

- recommend_destination: The agent recommends a destination based on user preferences (budget, luxury, adventure) and internal weightings.
- get_user_feedback: The agent receives feedback on the recommendation (positive or negative).
- meta_reasoning: The agent adjusts its reasoning by updating the weights based on feedback, improving future recommendations.

[4]:
```
# Simulated travel agent with meta-reasoning capabilities
class ReflectiveTravelAgent:
    def __init__(self):
        # Initialize preference weights that determine how user preferences␣
 ↪influence recommendations
        self.preferences_weights = {
            "budget": 0.5,    # Weight for budget-related preferences
            "luxury": 0.3,    # Weight for luxury-related preferences
            "adventure": 0.2  # Weight for adventure-related preferences
        }
```

```
        self.user_feedback = []  # List to store user feedback for␣
↪meta-reasoning

    def recommend_destination(self, user_preferences):
        """
        Recommend a destination based on user preferences and internal␣
↪weightings.

        Args:
            user_preferences (dict): User's preferences with keys like␣
↪'budget', 'luxury', 'adventure'

        Returns:
            str: Recommended destination
        """
        # Calculate scores for each destination based on weighted user␣
↪preferences
        score = {
            "Paris": (self.preferences_weights["luxury"] *␣
↪user_preferences["luxury"] +
                      self.preferences_weights["adventure"] *␣
↪user_preferences["adventure"]),
            "Bangkok": (self.preferences_weights["budget"] *␣
↪user_preferences["budget"] +
                        self.preferences_weights["adventure"] *␣
↪user_preferences["adventure"]),
            "New York": (self.preferences_weights["luxury"] *␣
↪user_preferences["luxury"] +
                         self.preferences_weights["budget"] *␣
↪user_preferences["budget"])
        }
        # Select the destination with the highest calculated score
        recommendation = max(score, key=score.get)
        return recommendation

    def get_user_feedback(self, actual_experience):
        """
        Simulate receiving user feedback and trigger meta-reasoning to adjust␣
↪recommendations.

        Args:
            actual_experience (str): The destination the user experienced
        """
        # Simulate user feedback: 1 for positive, -1 for negative
        feedback = random.choice([1, -1])
```

```
        print(f"Feedback for {actual_experience}: {'Positive' if feedback == 1␣
↪else 'Negative'}")

        # Store the feedback for later analysis
        self.user_feedback.append((actual_experience, feedback))

        # Trigger meta-reasoning to adjust the agent's reasoning process based␣
↪on feedback
        self.meta_reasoning()

    def meta_reasoning(self):
        """
        Analyze collected feedback and adjust preference weights to improve␣
↪future recommendations.
        This simulates the agent reflecting on its reasoning process and making␣
↪adjustments.
        """
        for destination, feedback in self.user_feedback:
            if feedback == -1:  # Negative feedback indicates dissatisfaction
                # Reduce the weight of the main attribute associated with the␣
↪destination
                if destination == "Paris":
                    self.preferences_weights["luxury"] *= 0.9  # Decrease␣
↪luxury preference
                elif destination == "Bangkok":
                    self.preferences_weights["budget"] *= 0.9  # Decrease␣
↪budget preference
                elif destination == "New York":
                    self.preferences_weights["budget"] *= 0.9  # Decrease␣
↪budget preference
            elif feedback == 1:  # Positive feedback indicates satisfaction
                # Increase the weight of the main attribute associated with the␣
↪destination
                if destination == "Paris":
                    self.preferences_weights["luxury"] *= 1.1  # Increase␣
↪luxury preference
                elif destination == "Bangkok":
                    self.preferences_weights["budget"] *= 1.1  # Increase␣
↪budget preference
                elif destination == "New York":
                    self.preferences_weights["budget"] *= 1.1  # Increase␣
↪budget preference

        # Normalize weights to ensure they sum up to 1 for consistency
        total_weight = sum(self.preferences_weights.values())
        for key in self.preferences_weights:
```

```
            self.preferences_weights[key] /= total_weight

        # Display updated weights after meta-reasoning adjustments
        print(f"Updated weights: {self.preferences_weights}\n")
```

0.4 Simulation

- User Preferences: Defines the user's preferences for budget, luxury, and adventure.
- First Recommendation: The agent recommends a destination based on the initial weights and user preferences.
- User Feedback Simulation: Simulates the user providing feedback on the recommended destination.
- Second Recommendation: After adjusting the weights based on feedback, the agent makes a new recommendation that reflects the updated reasoning process.

[6]:
```
# Simulate agent usage
if __name__ == "__main__":
    agent = ReflectiveTravelAgent()

    # User's initial preferences
    user_preferences = {
        "budget": 0.8,      # High preference for budget-friendly options
        "luxury": 0.2,      # Low preference for luxury
        "adventure": 0.5    # Moderate preference for adventure activities
    }

    # First recommendation based on initial preferences and weights
    recommended = agent.recommend_destination(user_preferences)
    print(f"Recommended destination: {recommended}")

    # Simulate user experience and provide feedback
    agent.get_user_feedback(recommended)

    # Second recommendation after adjusting weights based on feedback
    recommended = agent.recommend_destination(user_preferences)
    print(f"Updated recommendation: {recommended}")
```

```
Recommended destination: Bangkok
Feedback for Bangkok: Negative
Updated weights: {'budget': 0.4736842105263158, 'luxury': 0.3157894736842105,
'adventure': 0.2105263157894737}

Updated recommendation: Bangkok
```

0.5 ## Meta-reasoning with AI

Now let's bring in AI to perform meta-reasoning with agents. In this case, we will use the `CrewAI` framework to create our meta-reasoning Agents with OpenAI LLMs. We will also emulate user feedback using AI just for demonstration purposes.

0.5.1 What is CrewAI?

CrewAI is a leading platform focused on enabling and orchestrating **multi-agent AI systems**. In this context, "multi-agent" refers to the use of multiple AI agents—each with specialized roles or capabilities—that can collaborate to solve complex tasks more efficiently than a single AI model working alone. CrewAI provides the infrastructure and tools needed to build, deploy, and manage these agentic systems at scale.

A recent partnership between Centrilogic and CrewAI highlights CrewAI's position as a top platform for multi-agent AI, aiming to accelerate the adoption of these technologies in enterprise environments.

How Can You Use CrewAI with AI Agents? Using CrewAI with AI agents typically involves:

1. **Defining Agent Roles:**
   You can create multiple AI agents, each with a specific function (e.g., data analysis, customer support, workflow automation).
2. **Orchestrating Collaboration:**
   CrewAI provides tools to coordinate how these agents interact, share information, and make collective decisions to achieve a common goal.
3. **Integration with Existing Systems:**
   CrewAI can be integrated into your IT infrastructure, allowing agents to access data, trigger workflows, or interact with users and other software.
4. **Scaling and Managing Agents:**
   The platform offers management features for monitoring agent performance, scaling up resources as needed, and ensuring security and compliance.

Example Use Cases: - **Enterprise Automation:** Automate complex business processes by assigning different agents to handle document processing, approvals, and notifications. - **Customer Service:** Deploy a team of agents where one handles FAQs, another escalates complex issues, and a third manages follow-ups. - **Data Analysis:** Use specialized agents for data collection, cleaning, analysis, and reporting, all coordinated through CrewAI.

Why Use CrewAI?

- **Efficiency:** Multi-agent systems can break down large tasks and work in parallel, speeding up results.
- **Specialization:** Each agent can be optimized for a specific task, improving accuracy and performance.

- **Scalability:** CrewAI helps manage and scale agentic systems as your needs grow.

First, let's make sure we initialize our OpenAI API key, and then let's define the "Crew" (with CrewAI) and the Agents.

[7]:
```
import getpass
import os

api_key = getpass.getpass(prompt="Enter OpenAI API Key: ")
os.environ["OPENAI_API_KEY"] = api_key
```

```
Enter OpenAI API Key:  ........
```

We will define **three tools** that our agents will use-

1. `recommend_destination`: This tool will use a set of base weights that prioritize budget, luxury, and adventure equally, and then use the user's preference weights to recommend a destination. Paris will emphasize luxury, NYC emphasizes luxury and adventure, whereas Bangkok emphasizes budget.
2. `update_weights_on_feedback`: This tool will update the internal base weights based on the user's feedback on the recommended destination. Positive feedback will tell the model that its recommendation is correct and it needs to update its internal base weights based on the given (arbitrary adjustment factor), or reduce the weights using the adjustment factor if the feedback is dissatisfied.
3. `feedback_emulator`: This tool will emulate a user providing "satisfied" or "dissatisfied" feedback to the AI agent's destination recommendation

[8]:
```
from crewai.tools import tool

# Tool 1
@tool("Recommend travel destination based on preferences.")
def recommend_destination(user_preferences: dict) -> str:
    """
    Recommend a destination based on user preferences and internal weightings.

    Args:
        user_preferences (dict): User's preferences with keys - 'budget',␣
 ↪'luxury', 'adventure'
                                 default user_preference weights 'budget' = 0.8,␣
 ↪'luxury' = 0.2, 'adventure' = 0.5
                                 user_preferences = {
                                                 "budget": 0.8,
                                                 "luxury": 0.4,
                                                 "adventure": 0.3
                                             }
    Returns:
        str: Recommended destination
    """
```

```
    internal_default_weights = {
            "budget": 0.33,    # Weight for budget-related preferences
            "luxury": 0.33,    # Weight for luxury-related preferences
            "adventure": 0.33  # Weight for adventure-related preferences
        }
    # Calculate weighted scores for each destination
    score = {
        "Paris": (
            internal_default_weights["luxury"] * user_preferences["luxury"] +  # Paris emphasizes luxury
            internal_default_weights["adventure"] * user_preferences["adventure"] +
            internal_default_weights["budget"] * user_preferences["budget"]
        ),
        "Bangkok": (
            internal_default_weights["budget"] * user_preferences["budget"] * 2 +  # Bangkok emphasizes budget
            internal_default_weights["luxury"] * user_preferences["luxury"] +
            internal_default_weights["adventure"] * user_preferences["adventure"]
        ),
        "New York": (
            internal_default_weights["luxury"] * user_preferences["luxury"] * 1.5 +  # NYC emphasizes luxury and adventure
            internal_default_weights["adventure"] * user_preferences["adventure"] * 1.5 +
            internal_default_weights["budget"] * user_preferences["budget"]
        )
    }

    # Select the destination with the highest calculated score
    recommendation = max(score, key=score.get)
    return recommendation

# Tool 2
@tool("Reasoning tool to adjust preference weights based on user feedback.")
def update_weights_on_feedback(destination: str, feedback: int, adjustment_factor: float) -> dict:
    """
    Analyze collected feedback and adjust internal preference weights based on user feedback for better future recommendations.

    Args:
        destination (str): The destination recommended ('New York', 'Bangkok' or 'Paris')
        feedback (int): Feedback score; 1 = Satisfied, -1 = dissatisfied
```

```
        adjustment_factor (int): The adjustment factor between 0 and 1 that␣
↪will be used to adjust the internal weights.
                                 Value will be used as (1 - adjustment_factor)␣
↪for dissatisfied feedback and (1 + adjustment_factor)
                                 for satisfied feedback.
    Returns:
        dict: Adjusted internal weights
    """
    internal_default_weights = {
        "budget": 0.33,    # Weight for budget-related preferences
        "luxury": 0.33,    # Weight for luxury-related preferences
        "adventure": 0.33  # Weight for adventure-related preferences
    }

    # Define primary and secondary characteristics for each destination
    destination_characteristics = {
        "Paris": {
            "primary": "luxury",
            "secondary": "adventure"
        },
        "Bangkok": {
            "primary": "budget",
            "secondary": "adventure"
        },
        "New York": {
            "primary": "luxury",
            "secondary": "adventure"
        }
    }

    # Get the characteristics for the given destination
    dest_chars = destination_characteristics.get(destination, {})
    primary_feature = dest_chars.get("primary")
    secondary_feature = dest_chars.get("secondary")

    # adjustment_factor = 0.2  # How much to adjust weights by

    if feedback == -1:  # Negative feedback
        # Decrease weights for the destination's characteristics
        if primary_feature:
            internal_default_weights[primary_feature] *= (1 - adjustment_factor)
        if secondary_feature:
            internal_default_weights[secondary_feature] *= (1 -␣
↪adjustment_factor/2)

    elif feedback == 1:  # Positive feedback
        # Increase weights for the destination's characteristics
```

```
        if primary_feature:
            internal_default_weights[primary_feature] *= (1 + adjustment_factor)
        if secondary_feature:
            internal_default_weights[secondary_feature] *= (1 +␣
↪adjustment_factor/2)

    # Normalize weights to ensure they sum up to 1
    total_weight = sum(internal_default_weights.values())
    for key in internal_default_weights:
        internal_default_weights[key] = round(internal_default_weights[key] /␣
↪total_weight, 2)

    # Ensure weights sum to exactly 1.0 after rounding
    adjustment = 1.0 - sum(internal_default_weights.values())
    if adjustment != 0:
        # Add any rounding difference to the largest weight
        max_key = max(internal_default_weights, key=internal_default_weights.
↪get)
        internal_default_weights[max_key] =␣
↪round(internal_default_weights[max_key] + adjustment, 2)

    return internal_default_weights

# Tool 3
@tool("User feedback emulator tool")
def feedback_emulator(destination: str) -> int:
    """
    Given a destination recommendation (such as 'New York' or 'Bangkok') this␣
↪tool will emulate to provide
    a user feedback as 1 (satisfied) or -1 (dissatisfied)
    """
    import random
    feedback = random.choice([-1, 1])
    return feedback
```

Once the tools are defined, we will declare **three** `CrewAI` Agents, each of which will use one of the tools above. The `meta_agent` is basically the agent that will perform meta-reasoning using the emulated user feedback and the previously recommended destination to update the internal weights using an `adjustment_factor`.

Note that here, the model assigns an adjustment factor dynamically to adjust the internal system weights (which is `{"budget": 0.33, "luxury": 0.33, "adventure": 0.33}` in the beginning), i.e., we are not hard-coding the adjustment factor. Although the nature of user feedback in this example is limited to "satisfied" or "dissatisfied" (1 or -1), feedback can be of various forms and may contain more details, in which case your AI Agent may adjust different values to the adjustment_factor. More contextual feedback with details will help the model perform better meta-reasoning on its previous responses.

[9]:
```python
from crewai import Agent, Task, Crew
from typing import Dict, Union
import random

# Utility functions
def process_recommendation_output(output: str) -> str:
    """Extract the clean destination string from the agent's output."""
    # Handle various ways the agent might format the destination
    for city in ["Paris", "Bangkok", "New York"]:
        if city.lower() in output.lower():
            return city
    return output.strip()

def process_feedback_output(output: Union[Dict, str]) -> int:
    """Extract the feedback value from the agent's output."""
    if isinstance(output, dict):
        return output.get('feedback', 0)
    try:
        # Try to parse as an integer if it's a string
        return int(output)
    except (ValueError, TypeError):
        return 0

def generate_random_preferences():
    # Generate 3 random numbers and normalize them
    values = [random.random() for _ in range(3)]
    total = sum(values)

    return {
        "budget": round(values[0]/total, 2),
        "luxury": round(values[1]/total, 2),
        "adventure": round(values[2]/total, 2)
    }

# Initial shared state for weights, preferences, and results
state = {
    "weights": {"budget": 0.33, "luxury": 0.33, "adventure": 0.33},
    "preferences": generate_random_preferences()
}

# Agents Definition
preference_agent = Agent(
    name="Preference Agent",
    role="Travel destination recommender",
    goal="Provide the best travel destination based on user preferences and␣
 ↪weights.",
    backstory="An AI travel expert adept at understanding user preferences.",
```

```
    verbose=True,
    llm='gpt-4o-mini',
    tools=[recommend_destination]
)

feedback_agent = Agent(
    name="Feedback Agent",
    role="Simulated feedback provider",
    goal="Provide simulated feedback for the recommended travel destination.",
    backstory="An AI that mimics user satisfaction or dissatisfaction for␣
 ↪travel recommendations.",
    verbose=True,
    llm='gpt-4o-mini',
    tools=[feedback_emulator]
)

meta_agent = Agent(
    name="Meta-Reasoning Agent",
    role="Preference weight adjuster",
    goal="Reflect on feedback and adjust the preference weights to improve␣
 ↪future recommendations.",
    backstory="An AI optimizer that learns from user experiences to fine-tune␣
 ↪recommendation preferences.",
    verbose=True,
    llm='gpt-4o-mini',
    tools=[update_weights_on_feedback]
)


# Tasks with data passing that shall be performed by the AI Agents
generate_recommendation = Task(
    name="Generate Recommendation",
    agent=preference_agent,
    description=(
        f"Use the recommend_destination tool with these preferences:␣
 ↪{state['preferences']}\n"
        "Return only the destination name as a simple string (Paris, Bangkok,␣
 ↪or New York)."
    ),
    expected_output="A destination name as a string",
    output_handler=process_recommendation_output
)

simulate_feedback = Task(
    name="Simulate User Feedback",
    agent=feedback_agent,
    description=(
```

```
        "Use the feedback_emulator tool with the destination from the previous␣
 ↪task.\n"
        "Instructions:\n"
        "1. Get the destination string from the previous task\n"
        "2. Pass it directly to the feedback_emulator tool\n"
        "3. Return the feedback value (1 or -1)\n\n"
        "IMPORTANT: Pass the destination as a plain string, not a dictionary."
    ),
    expected_output="An integer feedback value: 1 or -1",
    context=[generate_recommendation],
    output_handler=process_feedback_output
)

adjust_weights = Task(
    name="Adjust Weights Based on Feedback",
    agent=meta_agent,
    description=(
        "Use the update_weights_on_feedback tool with:\n"
        "1. destination: Get from first task's output (context[0])\n"
        "2. feedback: Get from second task's output (context[1])\n"
        "3. adjustment_factor: a number between 0 and 1 that will be used to␣
 ↪adjust internal weights based on feedback\n\n"
        "Ensure all inputs are in their correct types (string for destination,␣
 ↪integer for feedback)."
    ),
    expected_output="Updated weights as a dictionary",
    context=[generate_recommendation, simulate_feedback]
)

# Crew Definition
crew = Crew(
    agents=[preference_agent, feedback_agent, meta_agent],
    tasks=[generate_recommendation, simulate_feedback, adjust_weights],
    verbose=False
)

# Execute the workflow
result = crew.kickoff()
print("\nFinal Results:", result)
```

```
                    Agent Started
                                                                              ␣
 ↪
  Agent: Travel destination recommender                                       ␣
 ↪
                                                                              ␣
 ↪
```

Task: Use the recommend_destination tool with these preferences: {'budget': 0.33, 'luxury': 0.15,
'adventure': 0.52}

Return only the destination name as a simple string (Paris, Bangkok, or New York).

Agent Tool Execution

Agent: **Travel destination recommender**

Thought: you should always think about what to do

Using Tool: **Recommend travel destination based on preferences.**

Tool Input

"{\"user_preferences\": {\"budget\": 0.33, \"luxury\": 0.15, \"adventure\": 0.52}}"

Tool Output

New York

Agent Final Answer

Agent: **Travel destination recommender**

Final Answer:
New York

Agent Started

Agent: **Simulated feedback provider**

Task: Use the feedback_emulator tool with the destination from the previous task.
Instructions:
1. Get the destination string from the previous task
2. Pass it directly to the feedback_emulator tool
3. Return the feedback value (1 or -1)

IMPORTANT: Pass the destination as a plain string, not a dictionary.

Agent Tool Execution

Agent: **Simulated feedback provider**

Thought: **Thought: I should use the feedback emulator tool to get feedback for the destination New York.**

Using Tool: **User feedback emulator tool**

Tool Input

"{\"destination\": \"New York\"}"

Tool Output

-1

Agent Final Answer

Agent: **Simulated feedback provider**

Final Answer:

-1

Agent Started

Agent: **Preference weight adjuster**

Task: Use the update_weights_on_feedback tool with:

1. destination: Get from first task's output (context[0])

2. feedback: Get from second task's output (context[1])

3. adjustment_factor: a number between 0 and 1 that will be used to adjust internal weights based on feedback

Ensure all inputs are in their correct types (string for destination, integer for feedback).

Agent Tool Execution

Agent: **Preference weight adjuster**

Thought: Thought: I need to adjust the preference weights based on the provided feedback regarding New York,
which is a dissatisfaction (feedback score of -1).

Using Tool: **Reasoning tool to adjust preference weights based on user feedback.**

Tool Input

"{\"destination\": \"New York\", \"feedback\": -1, \"adjustment_factor\": 0.5}"

Tool Output

{'budget': 0.45, 'luxury': 0.22, 'adventure': 0.33}

Agent Final Answer

Agent: **Preference weight adjuster**

```
 Final Answer:
 {'budget': 0.45, 'luxury': 0.22, 'adventure': 0.33}
```

```
Final Results: {'budget': 0.45, 'luxury': 0.22, 'adventure': 0.33}
```

Figure 4.1 shows a visual understanding of this flow:

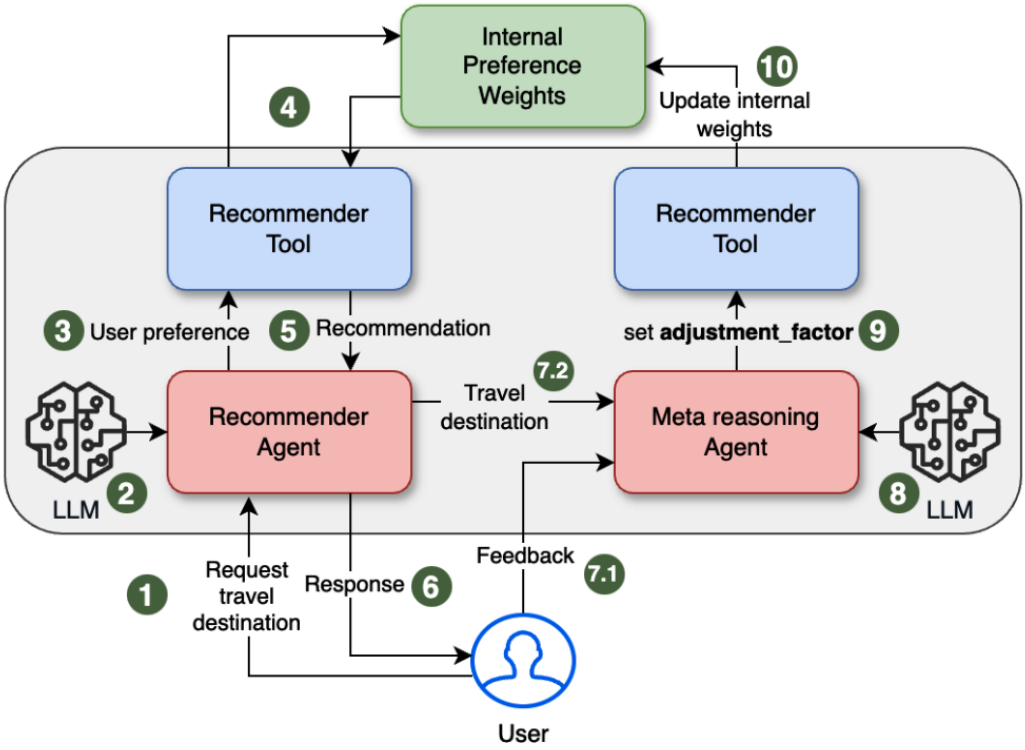

Figure 4.1 – Meta-reasoning with AI agents and CrewAI framework

1 2. Self-Explanation - example

1.0.1 What is Self-explanation?

Self-explanation is a process through which agents verbalize their reasoning processes, generating explanations for decisions reached. This technique serves several crucial purposes for reflective

agents, particularly in the context of our travel agent example, as discussed in the following sections.

Self-explanation serves two distinct purposes: ***enhancing transparency*** and ***facilitating learning***.

In this section, we will see how to implement transparency, learning and refinement, user engagement & collaboration.

1. **Self-Explanation transparency**: For each recommendation, the agent generates a detailed self-explanation. This explanation outlines the factors that led to the recommendation, such as proximity to popular attractions, budget-friendly options, or the presence of adventure activities. The purpose is to provide transparency into how the decision was made, helping the user understand the reasoning process.
2. **Learning and refinement**: The agent doesn't stop after making the recommendation. It actively reflects on user feedback (whether positive or negative). If the feedback is negative, it introspects on its decision-making process and adjusts the importance (weights) it assigns to user preferences for future recommendations. For instance, if a user dislikes a budget-friendly recommendation, the agent might reduce the emphasis it places on budget-related preferences.
3. **User Engagement**: The class also simulates a dialogue with the user. After giving the recommendation and the self-explanation, it collects feedback from the user, allowing for a more collaborative interaction. This feedback is then used to refine future recommendations, making the agent more adaptive and personalized.

[15]:
```
import getpass
import os

api_key = getpass.getpass(prompt="Enter OpenAI API Key: ")
os.environ["OPENAI_API_KEY"] = api_key
```

```
Enter OpenAI API Key:  ········
```

1.0.2 2.1 Transparency: Verbalizing Reasoning in Decisions

Lets use OpenAI SDK to see how a model can perform reasoning in the decisions it makes. Here, the agent generates explanations for its reasoning when recommending a travel itinerary. It uses GPT-4o-mini to generate self-explanations.

[16]:
```
import openai

# Mock data for the travel recommendation
user_preferences = {
    "location": "Paris",
    "budget": 200,
    "preferences": ["proximity to attractions", "user ratings"],
}

# Input reasoning factors for the GPT model
reasoning_prompt = f"""
```

```
You are an AI-powered travel assistant. Explain your reasoning behind a hotel␣
 ↪recommendation for a user traveling to {user_preferences['location']}.
Consider:
1. Proximity to popular attractions.
2. High ratings from similar travelers.
3. Competitive pricing within ${user_preferences['budget']} budget.
4. Preferences: {user_preferences['preferences']}.
Provide a clear, transparent self-explanation.
"""

response = openai.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {"role": "system", "content": "You are a reflective travel assistant."},
        {"role": "user", "content": reasoning_prompt},
    ]
)

# Print self-explanation
print("Agent Self-Explanation:")
print(response.choices[0].message.content)
```

```
Agent Self-Explanation:
When recommending a hotel for a user traveling to Paris, I will take into
account the preferences specified, namely proximity to attractions and high user
ratings, while also adhering to the budget of $200.

1. **Proximity to Popular Attractions**: Paris is filled with iconic sites such
as the Eiffel Tower, Louvre Museum, Notre-Dame Cathedral, and Montmartre. A good
choice would be a hotel located in areas such as the 1st (near the Louvre), the
7th (near the Eiffel Tower), or the 18th (Montmartre). This ensures that the
user can easily access these attractions, potentially minimizing travel time and
maximizing the overall experience.

2. **High Ratings from Similar Travelers**: I would search for hotels that have
received excellent ratings on travel platforms, typically around 4.0 out of 5.0
or better. This provides assurance that previous guests had positive
experiences, contributing to the reliability of the recommendation. Feedback
regarding cleanliness, service, and amenities would play a crucial role to
ensure that the user has a comfortable stay.

3. **Competitive Pricing within $200 Budget**: It's important to filter hotels
that fall within the user's financial constraints. The cost consideration
ensures that the user can enjoy their trip without overextending their budget. I
would look for hotels that not only meet the accommodation needs but also offer
good value for the price.
```

```
Given these considerations, a recommended hotel might be something like **Hôtel
le Relais Saint-Honoré** in the 1st arrondissement. This hotel typically offers
competitive pricing around the $200 mark, boasts high ratings from guests, and
is just a short walk from several major attractions such as the Louvre and the
Palais Royal.

To summarize, my recommendation for a hotel in Paris hinges on ensuring that it
is well-located for easy access to key sites, maintains high user ratings
reflecting positive traveler experiences, and stays within an affordable price
range that aligns with the user's budget. This holistic approach helps ensure a
fulfilling trip to Paris, emphasizing both convenience and guest satisfaction.
```

Answer:

Agent Self-Explanation: When recommending a hotel for a user traveling to Paris, I will take into account the preferences specified, namely proximity to attractions and high user ratings, while also adhering to the budget of $200.

1. **Proximity to Popular Attractions**: Paris is filled with iconic sites such as the Eiffel Tower, Louvre Museum, Notre-Dame Cathedral, and Montmartre. A good choice would be a hotel located in areas such as the 1st (near the Louvre), the 7th (near the Eiffel Tower), or the 18th (Montmartre). This ensures that the user can easily access these attractions, potentially minimizing travel time and maximizing the overall experience.
2. **High Ratings from Similar Travelers**: I would search for hotels that have received excellent ratings on travel platforms, typically around 4.0 out of 5.0 or better. This provides assurance that previous guests had positive experiences, contributing to the reliability of the recommendation. Feedback regarding cleanliness, service, and amenities would play a crucial role in ensuring that the user has a comfortable stay.
3. **Competitive Pricing within $200 Budget**: It's important to filter hotels that fall within the user's financial constraints. The cost consideration ensures that the user can enjoy their trip without overextending their budget. I would look for hotels that not only meet the accommodation needs but also offer good value for the price.

Given these considerations, a recommended hotel might be something like **Hôtel le Relais Saint-Honoré** in the 1st arrondissement. This hotel typically offers competitive pricing around the $200 mark, boasts high ratings from guests, and is just a short walk from several major attractions such as the Louvre and the Palais Royal.

To summarize, my recommendation for a hotel in Paris hinges on ensuring that it is well-located for easy access to key sites, maintains high user ratings reflecting positive traveler experiences, and stays within an affordable price range that aligns with the user's budget. This holistic approach helps ensure a fulfilling trip to Paris, emphasizing both convenience and guest satisfaction.

A conceptual flow of how an agentic system with self-explanation and transparency would look is shown in Figure 4.2:

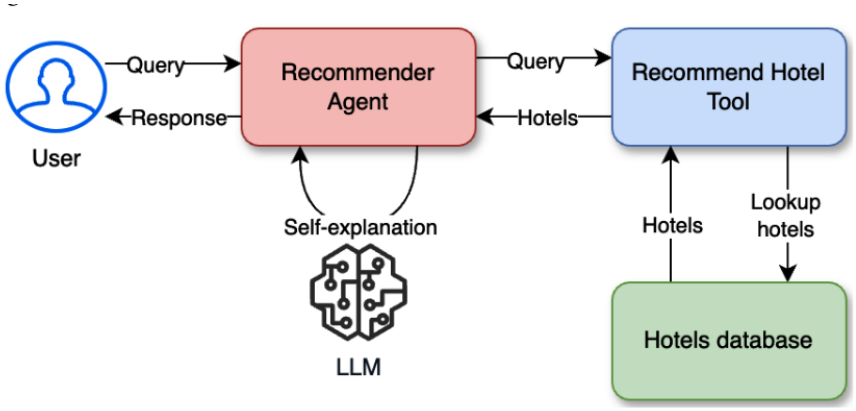

Figure 4.2 – Self-explanations transparency with AI agents

Using Crew AI Our previous example was pretty simple and didn't use Agents. But with an agentic system you may have agents actually lookup hotels appropriate to the user query using tools. Subsequently, a second agent may perform the self-explanation transparency on the response. Let's first define a tool that will respond back with mock hotel data based on price.

[18]:
```
from crewai.tools import tool

# Tool 1
@tool("Recommend hotels based on user query.")
def recommend_hotel(cost_per_night: int) -> str:
    """
    Returns hotels based on cost per night.

    Args:
        cost_per_night (int): User's preference for hotel room per night cost.

    """
    static_hotels = [
        {
            "hotel_name": "Le Royal Monceau Raffles",
            "price_per_night": 1200,
            "transportation_convenience": "convenient",
            "location": "8th arrondissement",
            "nearest_metro": "Charles de Gaulle-Étoile",
            "distance_from_metro": '1 km'
        },
        {
            "hotel_name": "Citadines Les Halles",
            "price_per_night": 250,
            "transportation_convenience": "convenient",
            "location": "1st arrondissement",
            "nearest_metro": "Les Halles",
            "distance_from_metro": '2.8 km'
```

```
        },
        {
            "hotel_name": "Ibis Paris Montmartre",
            "price_per_night": 120,
            "transportation_convenience": "moderate",
            "location": "18th arrondissement",
            "nearest_metro": "Place de Clichy",
            "distance_from_metro": '5 km'
        },
        {
            "hotel_name": "Four Seasons Hotel George V",
            "price_per_night": 1500,
            "transportation_convenience": "convenient",
            "location": "8th arrondissement",
            "nearest_metro": "George V",
            "distance_from_metro": '1 km'
        },
        {
            "hotel_name": "Hotel du Petit Moulin",
            "price_per_night": 300,
            "transportation_convenience": "moderate",
            "location": "3rd arrondissement",
            "nearest_metro": "Saint-Sébastien Froissart",
            "distance_from_metro": '1.9 km'
        }
    ]
    matching_hotels = [
        hotel for hotel in static_hotels
        if cost_per_night <= hotel["price_per_night"]
    ]
    return matching_hotels
```

Now we will perform the same transparency reasoning with a CrewAI Agent/Task combination.

1.0.3 Transparency

By generating self-explanations for its recommendations and decisions, the reflective travel agent can provide users with insights into its thought processes and decision-making rationale. This transparency fosters trust and confidence in the agent's capabilities, as users can better understand the reasoning behind the suggested itineraries, accommodations, or activities.

[21]:
```
from crewai import Agent, Task, Crew
from crewai.process import Process

travel_agent = Agent(
    role="Travel Advisor",
    goal="Provide hotel recommendations with transparent reasoning.",
    backstory="""
```

```
    An AI travel advisor specializing in personalized travel planning.
    You always explain the steps you take to arrive at a conclusion
    """,
    allow_delegation=False,
    llm='gpt-4o-mini',
    tools=[recommend_hotel]
)

recommendation_task = Task(
    name="Recommend hotel",
    description="""
    Recommend a hotel based on the user's query:
    '{query}'.
    """,
    agent=travel_agent,
    expected_output="The hotel recommendation and reasoning in the following␣
 ↪format\n\nHotel: [Your answer]\n\nPrice/night: [The price]\n\nReason:␣
 ↪[Detailed breakdown of your thought process]"
)

travel_crew = Crew(
    agents=[travel_agent],
    tasks=[recommendation_task],
    process=Process.sequential,
    verbose=False
)

travel_crew.kickoff(inputs={'query': "I am looking for a hotel in Paris under␣
 ↪$300 a night."})

# Retrieve and print the output
output = recommendation_task.output
print("Hotel Recommendation and Explanation:")
print(output)
```

```
Hotel Recommendation and Explanation:
Hotel: Hotel du Petit Moulin

Price/night: $300

Reason: After analyzing various hotel options in Paris under the budget of $300
per night, I found that the Hotel du Petit Moulin is the only viable option.
This hotel is located in the 3rd arrondissement, which is known for its vibrant
atmosphere. It has a moderate level of transportation convenience, with the
nearest metro station being Saint-Sébastien Froissart, roughly 1.9 km away.
Given its unique decor and central location, it offers a good balance of comfort
and accessibility within the user's budget.
```

Not only does our Agent can lookup hotels using the tool but it clearly explains why it gave the recommendation as `Reason`.

1.0.4 2.2 Learning and Refinement: Using Self-Explanation to Identify Gaps

While our self-explaining agent is great, the user may still not like the recommendation it gave. In which case, the user may express their dissatisfaction with the recommendation. This is where we need learning and refinement of the approach. In our case, we may extend the previous agent-based system to now also include a second agent that can take user feedback and re-strategize on its approach to recommend a hotel. Note that in this case, sequential execution or parallel execution of the agents may not be appropriate; thus, we need a hierarchical approach where a top-level agent can manage the two agents and then delegate tasks accordingly.

Let's define our **learning** and **refinement** agent that can take user feedback and its previous recommendation in context, and then complete the task by refining its strategy.

[33]:
```
from crewai import Agent, Task, Crew
from crewai.process import Process

# Agent Definition
reflective_travel_agent = Agent(
    role="Self-Improving Travel Advisor",
    goal="Refine hotel recommendations based on user feedback to your previous␣
 ↪recommendation to improve decision-making.",
    backstory="""
    A reflective AI travel advisor specializing in personalized travel planning␣
 ↪that learns from user feedback.
    When a user highlights an issue with a recommendation, it revisits its␣
 ↪reasoning,
    identifies overlooked factors, and updates its decision process accordingly.
    """,
    allow_delegation=False,
    llm='gpt-4o-mini',
    #tools=[recommend_hotel] # <-- This agent also uses the same tool to refine␣
 ↪its recommendation
)

# Task Definition
feedback_task = Task(
    description="""
    Based on your previous recommendation:
    '{recommendation}'

    Reflect on the user's feedback on the hotel recommendation:
    '{query}'

    - Identify any oversight in your previous reasoning process.
    - Update your reasoning process to include aspects that were missed.
```

```
    - Provide the refined steps that you will use to recommend hotels.
    """,
    expected_output="""
    A refined explanation that acknowledges the oversight, includes missed␣
 ↪factors,
    and provides revised steps to recommend hotels tailored to the user's␣
 ↪feedback.
    """,
    agent=reflective_travel_agent,
    context=[recommendation_task]
)

# Crew Orchastartion
travel_feedback_crew = Crew(
    agents=[reflective_travel_agent],
    tasks=[feedback_task],
    process=Process.sequential,
    verbose=False
)

# We will run the travel_crew from the previous example with user's query
response1 = travel_crew.kickoff(inputs={'query': "I am looking for a hotel in␣
 ↪Paris under $300 a night."})
print(response1)

# Adjusted code
response2 = travel_feedback_crew.kickoff(inputs={
    'recommendation': str(response1),  # Convert CrewOutput to string
    'query': "The hotel you recommended was too far from public transport. I␣
 ↪prefer locations closer to metro stations."
})

print(response2)
```

```
Hotel: Hotel du Petit Moulin

Price/night: $300

Reason: "Hotel du Petit Moulin is the only recommendation that fits the user's
budget of under $300 per night. It is located in the 3rd arrondissement, which
is a vibrant area known for its cafes and boutiques. The nearest metro station
is Saint-Sébastien Froissart, approximately 1.9 km away, providing moderate
transportation convenience. This hotel offers a unique charm, making it a
suitable choice for a pleasant stay in Paris."
I appreciate your feedback regarding the hotel's proximity to public transport,
as this is crucial for ensuring an enjoyable experience during your travels.
Upon reflection, I realize that my previous recommendation of Hotel du Petit
```

```
Moulin did not adequately consider the desire for closer access to metro
stations, particularly as the distance of approximately 1.9 km could pose an
inconvenience.

In addition to distance, I should also take into account the types of metro
stations available nearby, as some lines might provide more convenient access to
major attractions in Paris. Furthermore, a review of nearby amenities or
services, like restaurants or grocery stores, could enhance the overall
experience of your stay.

To better refine my process for future hotel recommendations, I will adopt the
following updated reasoning steps:

1. **User Preferences Assessment**: Begin by confirming the user's specific
preferences regarding proximity to transport.
2. **Transport Accessibility Evaluation**: Identify hotels within a defined
walking distance (e.g., less than 0.5 km) from metro stations, ensuring
excellent access to key transport routes.
3. **Budget Consideration**: Verify that hotel options still fit within the
user's budget constraints.
4. **Local Amenities**: Assess the vibrancy of the surrounding area, considering
cafes, restaurants, and shops, to enhance the overall travel experience.
5. **User Feedback Integration**: Actively seek feedback on my recommendations
and adjust my approach accordingly to refine future suggestions continually.

By incorporating these steps, I can ensure that my hotel recommendations provide
not only good pricing and local charm but also essential access to public
transportation in accordance with your preferences. Thank you for your
invaluable input, and I look forward to assisting you with future travel plans!
```

Figure 4.3 shows the high-level flow:

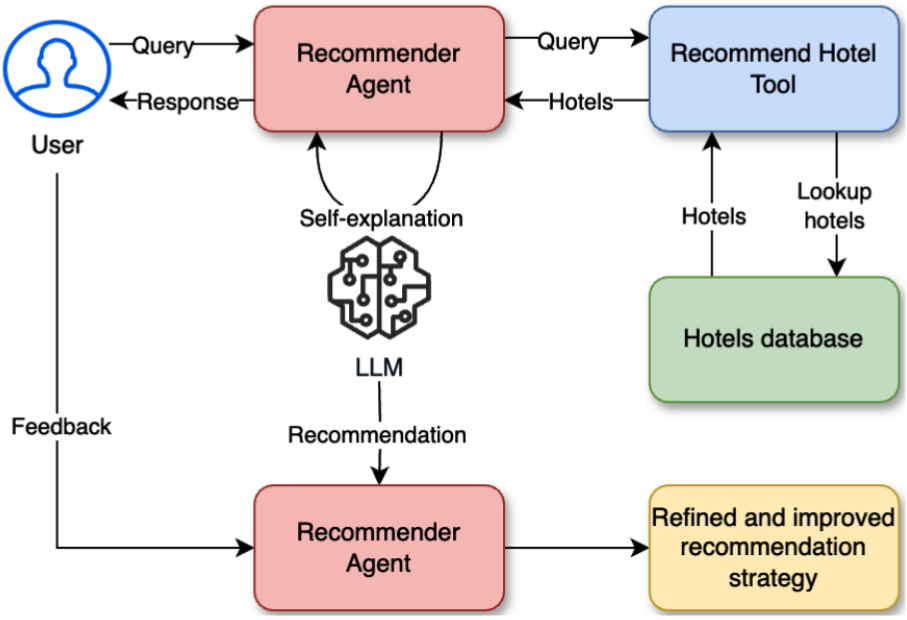

Figure 4.3 – Learning and refinement with AI agents

1.0.5 2.3. User Engagement and Collaboration: Enabling Interactive Explanations

In this example, the agent provides explanations for its decisions and engages users to refine suggestions interactively. Just like before, we can have an Agent/Task pair with CrewAI framework whose job is to interact with the users by asking clarifying questions about their preferences.

[34]:
```
from crewai import Agent, Task, Crew
from crewai.process import Process

# Step 1: Define the Collaborative Agent
collaborative_travel_agent = Agent(
    role="Collaborative AI Travel Assistant",
    goal="""
    Engage in an interactive dialogue with the user to clarify hotel␣
 ↪recommendations.
    Explain the reasoning for prioritizing certain factors and invite the user␣
 ↪to share their preferences.
    """,
    backstory="""
    An AI travel assistant that values user input and ensures recommendations␣
 ↪are well-aligned with user needs.
    It provides clear explanations for its decisions and encourages␣
 ↪collaborative planning.
    """
)

# Assemble and define the Task
```

```
interactive_task = Task(
    description="""
    Facilitate an interactive dialogue with the user.

    - Here is the initial recommendation: {initial_recommendation}
    - The user has asked: {user_query}

    Respond by:
    1. Explaining the reasoning behind prioritizing proximity to attractions.
    2. Inviting the user to clarify whether proximity to public transport is␣
 ↪more important.
    """,
    expected_output="""
    A clear and polite response explaining the reasoning and inviting the user␣
 ↪to share further input.
    """,
    agent=collaborative_travel_agent
)

# Step 3: Assemble the Crew
interactive_crew = Crew(
    agents=[collaborative_travel_agent],
    tasks=[interactive_task],
    process=Process.sequential,
    verbose=True
)


# Step 2: Define the Task for Clarification Dialogue

# Initial recommendation
initial_recommendation = "I recommend Hotel Lumière in Paris for its proximity␣
 ↪to the Eiffel Tower, high ratings, and budget-friendly price."
user_query = "Why did you prioritize proximity to attractions over public␣
 ↪transport access?"

# Step 4: Run the Crew and Output the Results
print("Starting Interactive Dialogue with User...\n")
result = interactive_crew.kickoff(inputs={"initial_recommendation":␣
 ↪initial_recommendation, "user_query": user_query})

print("Final Interactive Response:")
print(result)
```

```
Starting Interactive Dialogue with User…
```

```
                                Crew Execution Started
                                                                                                                    ␣
↪
 Crew Execution Started                                                                                             ␣
↪
 Name: crew                                                                                                         ␣
↪
 ID: 3e5d29cb-e73c-4b58-a3ea-ba90c73ace3c                                                                           ␣
↪
 Tool Args:                                                                                                         ␣
↪
                                                                                                                    ␣
↪
                                                                                                                    ␣
↪


                                  Agent Started
                                                                                                                    ␣
↪
 Agent: Collaborative AI Travel Assistant                                                                           ␣
↪
                                                                                                                    ␣
↪
 Task:                                                                                                              ␣
↪
     Facilitate an interactive dialogue with the user.                                                              ␣
↪
                                                                                                                    ␣
↪
     - Here is the initial recommendation: I recommend Hotel Lumière in Paris␣
↪for its proximity to the Eiffel
 Tower, high ratings, and budget-friendly price.                                                                    ␣
↪
     - The user has asked: Why did you prioritize proximity to attractions over␣
↪public transport access?
                                                                                                                    ␣
↪
     Respond by:                                                                                                    ␣
↪
     1. Explaining the reasoning behind prioritizing proximity to attractions.                                      ␣
↪
```

```
    2. Inviting the user to clarify whether proximity to public transport is␣
↪more important.
                                                                              ␣
↪
                                                                              ␣
↪
```

```
Output()
```

```
                          Agent Final Answer
                                                                              ␣
↪
 Agent: Collaborative AI Travel Assistant                                     ␣
↪
                                                                              ␣
↪
 Final Answer:                                                                ␣
↪
 Thank you for your question! I prioritized proximity to attractions like the␣
↪Eiffel Tower because it enhances
 convenience for sightseeing, allowing you to maximize your time exploring the␣
↪iconic landmarks of Paris.
 Being within walking distance can also lead to a more enjoyable and immersive␣
↪experience without the need for
 extensive travel time.                                                       ␣
↪
                                                                              ␣
↪
 However, I completely understand that some travelers place a high value on␣
↪access to public transport, as it
 can provide flexibility in exploring other areas of the city and may be␣
↪beneficial for longer trips outside
 the immediate vicinity.                                                      ␣
↪
                                                                              ␣
↪
 Could you share if proximity to public transport is a higher priority for your␣
↪travel needs, or are there
 specific attractions you're particularly keen on visiting? Your input will␣
↪help me refine my recommendations!
```

```
Task Completion

Task Completed
Name: dd082fa4-9690-48cb-a186-3b8628e5ac57
Agent: Collaborative AI Travel Assistant
Tool Args:



Crew Completion

Crew Execution Completed
Name: crew
ID: 3e5d29cb-e73c-4b58-a3ea-ba90c73ace3c
Tool Args:
Final Output: Thank you for your question! I prioritized proximity to attractions like the Eiffel Tower
because it enhances convenience for sightseeing, allowing you to maximize your time exploring the iconic
landmarks of Paris. Being within walking distance can also lead to a more enjoyable and immersive experience
```

```
 without the need for extensive travel time.
↪
↪
 However, I completely understand that some travelers place a high value on
↪access to public transport, as it
 can provide flexibility in exploring other areas of the city and may be
↪beneficial for longer trips outside
 the immediate vicinity.
↪
↪
 Could you share if proximity to public transport is a higher priority for your
↪travel needs, or are there
 specific attractions you're particularly keen on visiting? Your input will
↪help me refine my recommendations!
↪
↪
```

```
Final Interactive Response:
Thank you for your question! I prioritized proximity to attractions like the
Eiffel Tower because it enhances convenience for sightseeing, allowing you to
maximize your time exploring the iconic landmarks of Paris. Being within walking
distance can also lead to a more enjoyable and immersive experience without the
need for extensive travel time.

However, I completely understand that some travelers place a high value on
access to public transport, as it can provide flexibility in exploring other
areas of the city and may be beneficial for longer trips outside the immediate
vicinity.

Could you share if proximity to public transport is a higher priority for your
travel needs, or are there specific attractions you're particularly keen on
visiting? Your input will help me refine my recommendations!
```

2 3. Self Modeling - example

Self-modeling is a crucial aspect of reflective agents, allowing them to maintain an internal representation of their goals, beliefs, and knowledge. This self-model serves as a foundation for decision-making and reflection, enabling the agent to adapt and evolve in response to changing circumstances or newly acquired information. To clarify a bit further, the term modeling in this

context means the agent's initial environment and state. The agent (or group of agents) starts with some initial state with a specific environment, and as the agent learns more via human-machine interactions or via its task executions, it continues to update that internal state, thus changing its own environment within which it operates.

The `ReflectiveTravelAgentWithSelfModeling` class represents a sophisticated travel recommendation system that utilizes **self-modeling** to enhance its decision-making and adaptability.

2.0.1 1. Initialization:

- **Self-Model and Knowledge Base:** The agent starts with an internal model that includes its goals and a knowledge base.
  - **Goals:** Initially, the goals are set to provide personalized recommendations, optimize user satisfaction, and not prioritize eco-friendly options by default.
  - **Knowledge Base:** It contains information about various travel destinations, including their ratings, costs, luxury levels, and sustainability. This base also tracks user preferences.

2.0.2 2. Updating Goals:

- **Adapting to Preferences:** When new user preferences are provided, the agent can update its goals accordingly. For example, if the user prefers eco-friendly options, the agent will adjust its goals to prioritize recommending sustainable travel options. Similarly, if the user's budget changes, the agent will refocus on cost-effective recommendations.

2.0.3 3. Updating Knowledge Base:

- **Incorporating Feedback:** After receiving feedback from users, the agent updates its knowledge base. If the feedback is positive, the agent increases the rating of the recommended destination. If the feedback is negative, the rating is decreased. This helps the agent refine its recommendations based on real user experiences.

2.0.4 4. Making Recommendations:

- **Calculating Scores:** The agent evaluates each destination by calculating a score based on its rating and, if eco-friendly options are a goal, it adjusts the score by adding the sustainability rating.
- **Selecting the Best Destination:** The destination with the highest score is recommended to the user. This process ensures that the recommendation aligns with both user preferences and the agent's goals.

2.0.5 5. Engaging with the User:

- **Providing Recommendations:** The agent presents the recommended destination to the user and asks for feedback.
- **Feedback Handling:** The feedback (positive or negative) is used to update the knowledge base, which helps improve future recommendations.

Figure 4.4 gives a high-level overview of agent self-modeling as we further discuss the two components of internal state. Agents may have individual internal states that they independently

self-model within an agentic system, or they may have a shared internal state that they collaboratively self-model.

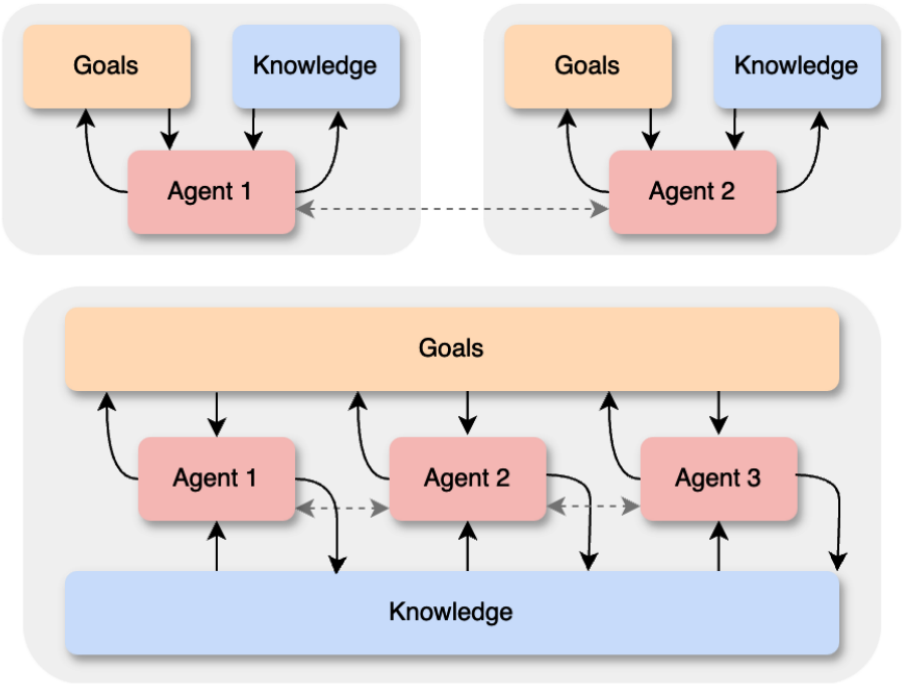

Figure 4.4 – Individual and shared internal states for self-modeling

[35]:
```
class ReflectiveTravelAgentWithSelfModeling:
    def __init__(self):
        # Initialize the agent with a self-model that includes goals and a␣
↪knowledge base
        self.self_model = {
            "goals": {
                "personalized_recommendations": True,
                "optimize_user_satisfaction": True,
                "eco_friendly_options": False  # Default: Not prioritizing␣
↪eco-friendly options
            },
            "knowledge_base": {
                "destinations": {
                    "Paris": {"rating": 4.8, "cost": 2000, "luxury": 0.9,␣
↪"sustainability": 0.3},
                    "Bangkok": {"rating": 4.5, "cost": 1500, "luxury": 0.7,␣
↪"sustainability": 0.6},
                    "Barcelona": {"rating": 4.7, "cost": 1800, "luxury": 0.8,␣
↪"sustainability": 0.7}
                },
                "user_preferences": {}
```

```
            }
        }

    def update_goals(self, new_preferences):
        """Update the agent's goals based on new user preferences."""
        if new_preferences.get("eco_friendly"):
            self.self_model["goals"]["eco_friendly_options"] = True
            print("Updated goal: Prioritize eco-friendly travel options.")
        if new_preferences.get("adjust_budget"):
            print("Updated goal: Adjust travel options based on new budget␣
↪constraints.")

    def update_knowledge_base(self, feedback):
        """Update the agent's knowledge base based on user feedback."""
        destination = feedback["destination"]
        if feedback["positive"]:
            # Increase rating for positive feedback
            self.
↪self_model["knowledge_base"]["destinations"][destination]["rating"] += 0.1
            print(f"Positive feedback received for {destination}; rating␣
↪increased.")
        else:
            # Decrease rating for negative feedback
            self.
↪self_model["knowledge_base"]["destinations"][destination]["rating"] -= 0.2
            print(f"Negative feedback received for {destination}; rating␣
↪decreased.")

    def recommend_destination(self, user_preferences):
        """Recommend a destination based on user preferences and the agent's␣
↪self-model."""
        # Store user preferences in the agent's self-model
        self.self_model["knowledge_base"]["user_preferences"] = user_preferences

        # Update agent's goals based on new preferences
        if user_preferences.get("eco_friendly"):
            self.update_goals(user_preferences)

        # Calculate scores for each destination
        best_destination = None
        highest_score = 0
        for destination, info in self.
↪self_model["knowledge_base"]["destinations"].items():
            score = info["rating"]
            if self.self_model["goals"]["eco_friendly_options"]:
```

```
                # Boost score for eco-friendly options if that goal is␣
↪prioritized
                score += info["sustainability"]

            # Update the best destination if the current score is higher
            if score > highest_score:
                best_destination = destination
                highest_score = score

        return best_destination

    def engage_with_user(self, destination):
        """Simulate user engagement by providing the recommendation and␣
↪receiving feedback."""
        print(f"Recommended destination: {destination}")
        # Simulate receiving user feedback (e.g., through input in a real␣
↪application)
        feedback = input(f"Did you like the recommendation of {destination}?␣
↪(yes/no): ").strip().lower()
        positive_feedback = feedback == "yes"
        return {"destination": destination, "positive": positive_feedback}
```

The provided code snippet is designed to simulate the usage of the `ReflectiveTravelAgentWithSelfModeling` class.

2.0.6 1. Creating an Instance of the Agent:

```
agent = ReflectiveTravelAgentWithSelfModeling()
```

- **Purpose:** Initializes a new instance of the `ReflectiveTravelAgentWithSelfModeling` class.
- **Outcome:** This instance represents a travel agent equipped with self-modeling capabilities, including goal management and a knowledge base.

2.0.7 2. Setting User Preferences:

```
user_preferences = {
    "budget": 0.6,            # Moderate budget constraint
    "luxury": 0.4,            # Moderate preference for luxury
    "adventure": 0.7,         # High preference for adventure
    "eco_friendly": True      # User prefers eco-friendly options
}
```

- **Purpose:** Defines a set of preferences provided by the user.
- **Outcome:** These preferences indicate that the user has a moderate budget, moderate luxury preferences, a high interest in adventure, and a strong preference for eco-friendly options.

2.0.8 3. Getting a Recommendation:

```
recommendation = agent.recommend_destination(user_preferences)
```

- **Purpose:** Requests a travel destination recommendation from the agent based on the provided user preferences.
- **Outcome:** The agent processes the preferences, updates its goals if necessary (e.g., prioritizing eco-friendly options), and selects the best destination to recommend.

2.0.9 4. Engaging with the User:

`feedback = agent.engage_with_user(recommendation)`

- **Purpose:** Simulates interaction with the user by presenting the recommendation and gathering feedback.
- **Outcome:** The user provides feedback on the recommended destination, which is used to evaluate the effectiveness of the recommendation.

2.0.10 5. Updating the Knowledge Base:

`agent.update_knowledge_base(feedback)`

- **Purpose:** Updates the agent's knowledge base with the feedback received from the user.
- **Outcome:** The agent adjusts its knowledge base by modifying ratings or other attributes based on whether the feedback was positive or negative. This update helps improve future recommendations by refining the agent's understanding of user preferences and destination qualities.

2.0.11 Summary:

In essence, this code snippet demonstrates how the `ReflectiveTravelAgentWithSelfModeling` class operates in a simulated environment. It initializes the agent, sets user preferences, obtains a recommendation, engages the user for feedback, and updates the agent's knowledge base based on that feedback. This simulation helps illustrate the agent's self-modeling capabilities and its ability to adapt and improve recommendations over time.

[36]:
```python
# Simulating agent usage
if __name__ == "__main__":
    # Create an instance of the reflective travel agent with self-modeling
    agent = ReflectiveTravelAgentWithSelfModeling()

    # Example user preferences including a focus on eco-friendly options
    user_preferences = {
        "budget": 0.6,            # Moderate budget constraint
        "luxury": 0.4,            # Moderate preference for luxury
        "adventure": 0.7,         # High preference for adventure
        "eco_friendly": True      # User prefers eco-friendly options
    }

    # Get the recommended destination based on user preferences
    recommendation = agent.recommend_destination(user_preferences)

    # Engage with the user to provide feedback on the recommendation
    feedback = agent.engage_with_user(recommendation)
```

```
    # Update the knowledge base with the user feedback
    agent.update_knowledge_base(feedback)
```

```
Updated goal: Prioritize eco-friendly travel options.
Recommended destination: Barcelona

Did you like the recommendation of Barcelona? (yes/no):  no

Negative feedback received for Barcelona; rating decreased.
```

[ ]: