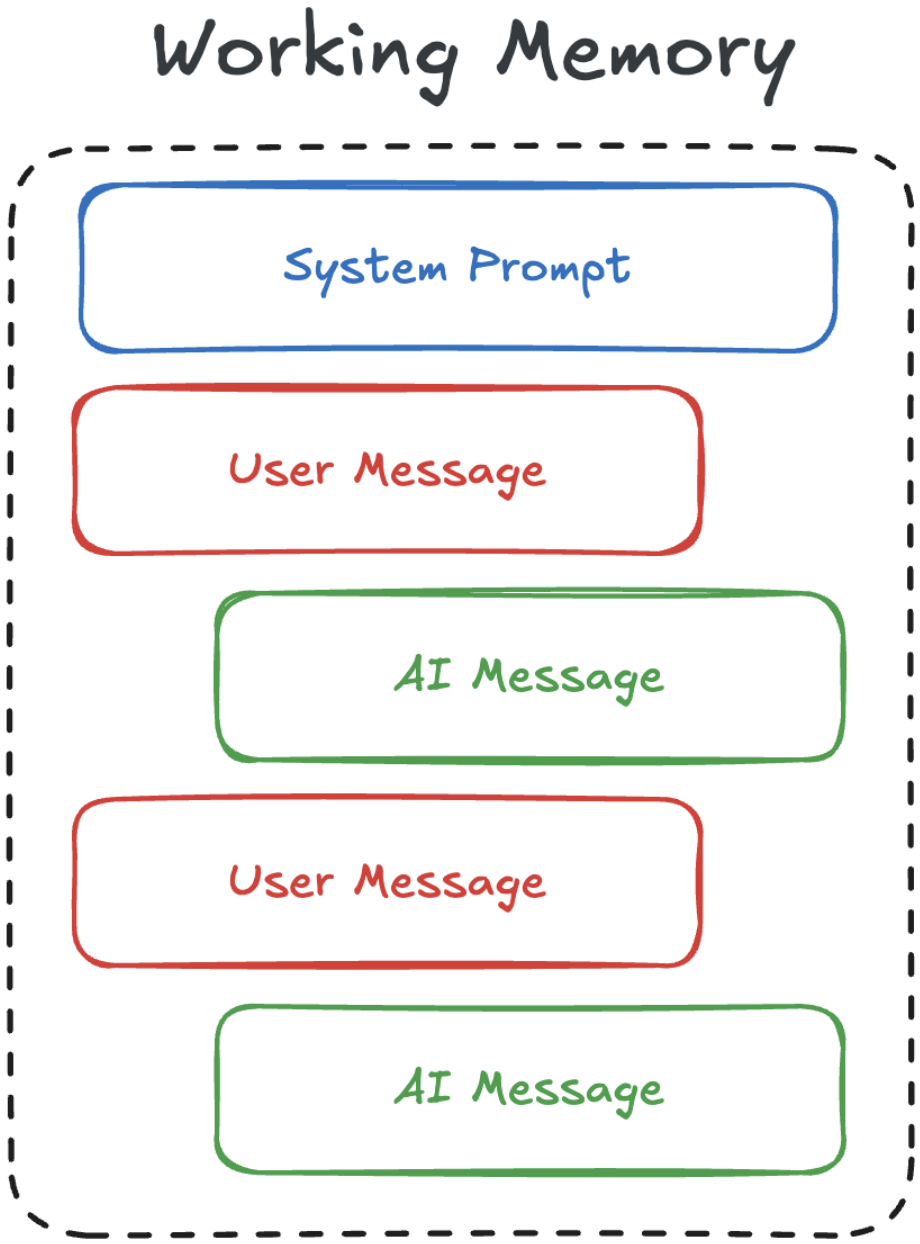
Memory :



1. Working Memory

Definition:

- **Human Context**: Short-term memory that holds a limited amount of information temporarily for immediate use or processing.
- **AI Context**: The immediate context of the ongoing conversation or task. It only remembers what is needed to respond or solve the current problem.

Key Features:

- Temporary and limited capacity.
- Constantly updated with new information.
- Used for real-time tasks and decision-making.

Examples:

- **Human**: Calculating a math problem mentally (e.g., multiplying 25 × 4) and forgetting the result after solving it.
- **AI**: Holding the last 10 lines of a conversation to generate the next response but forgetting older messages.

---

2. Episodic Memory

Definition:

- **Human Context**: Memory of specific events or experiences, including the context (time, place, emotions, etc.) in which they occurred.
- **AI Context**: A memory system that stores specific user interactions, conversations, or events to recall later.

Key Features:

- Personal and event-specific.
- Time-stamped and contextual.
- Helps recall past experiences to inform future decisions.

Examples:

- **Human**: Remembering a birthday celebration or your first day at school.

- **AI**: Recalling that a user discussed "machine learning" in a previous conversation and tailoring responses accordingly.

---

3. Semantic Memory

Definition:

- **Human Context**: Memory of facts, concepts, and general knowledge that is not tied to a specific event or experience.
- **AI Context**: A general knowledge base containing structured information to answer user queries.

Key Features:

- Focuses on general knowledge and facts.
- Independent of context or time.
- Helps provide accurate, factual information.

Examples:

- **Human**: Knowing that Paris is the capital of France or Python is a programming language.
- **AI**: Storing information like "Python is a high-level programming language" or "The Eiffel Tower is in Paris."

---

4. Procedural Memory

Definition:

- **Human Context**: Memory of how to perform tasks or actions, often learned through repetition.
- **AI Context**: Rules or patterns learned from feedback to perform tasks more efficiently in the future.

Key Features:

- Focuses on "how-to" knowledge.

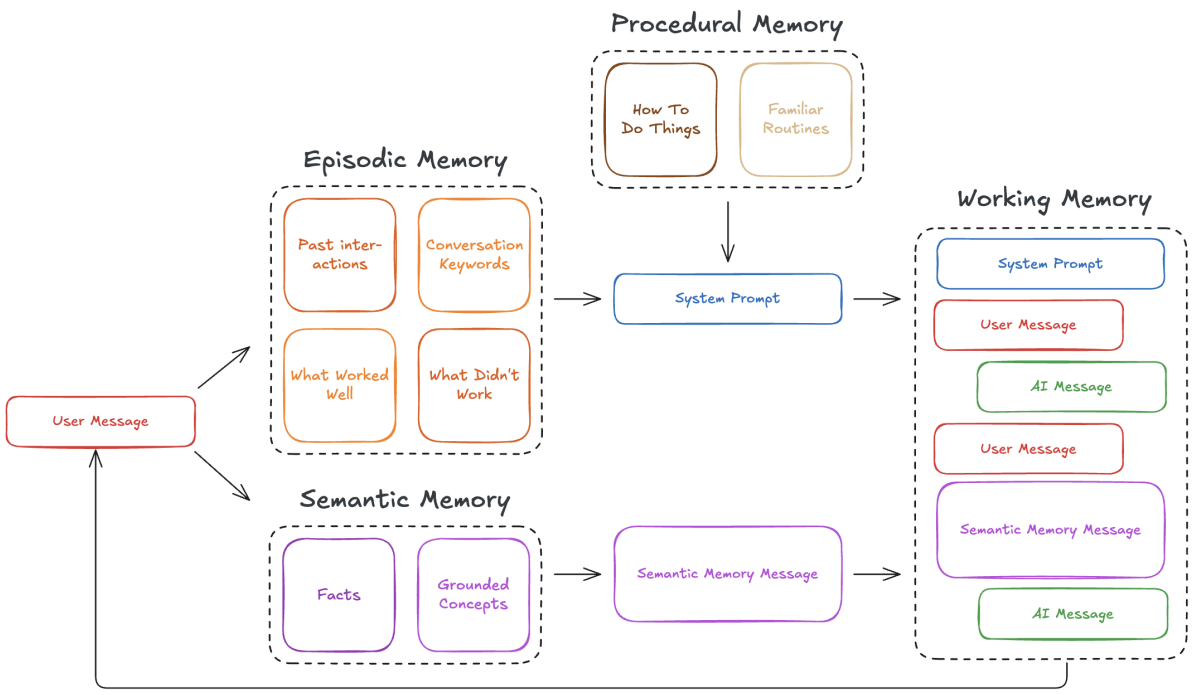
- Often operates unconsciously (in humans).
- Gained through experience and practice.

Examples:

- **Human**: Riding a bike, typing on a keyboard, or playing a musical instrument.
- **AI**: Remembering that a specific API call format works well for retrieving data or that users prefer concise answers.

Differences Between Memory Types

Memory Type	Human Context	AI Context	Focus	Duration
Working Memory	Temporary storage for immediate tasks or thinking.	Active conversation or task context.	Short-term, immediate use.	Very short (seconds).
Episodic Memory	Memory of specific events, tied to time and context.	Recall of past conversations or interactions.	Event-based, contextual.	Long-term.
Semantic Memory	General facts and knowledge about the world.	AI's knowledge base (e.g., factual info).	Fact-based, generalized.	Long-term.
Procedural Memory	Memory of how to perform actions or tasks.	Learning patterns, methods, or workflows.	Task-based, skill-oriented.	Long-term.



Visualizing with an AI Example

Scenario: A User Asks an AI Assistant Questions Over Time

1. **Working Memory**:
    - The AI holds the ongoing conversation context to craft relevant responses.
    - *E.g.*: User: "What is Python?" AI uses only this query to generate an answer.
2. **Episodic Memory**:
    - The AI remembers that the user asked about "Python" last week and stores their preferences for concise explanations.

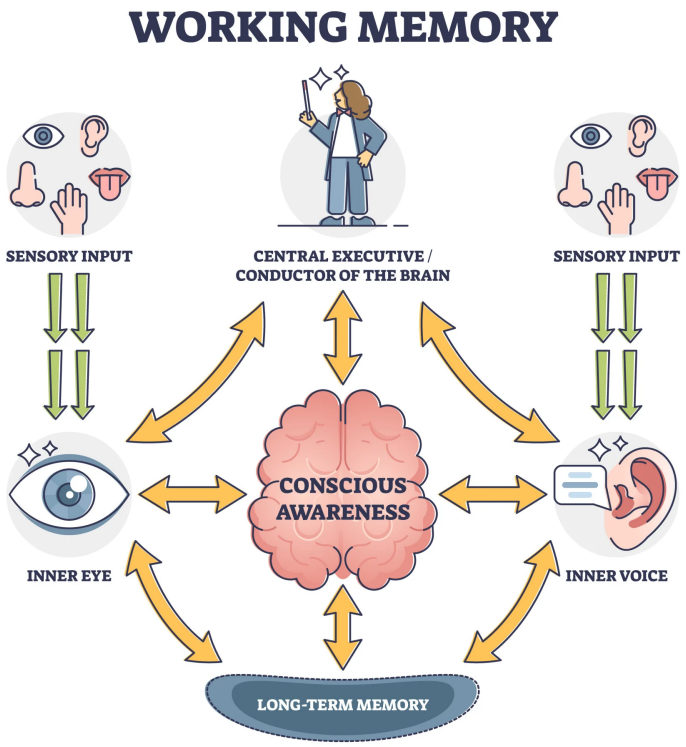
- *E.g.*: "Last week, you wanted a simple explanation of Python. Would you like more details now?"

3. **Semantic Memory**:
    - The AI pulls factual information from its knowledge base about Python.
    - *E.g.*: "Python is a high-level programming language used for web development, data science, and more."
4. **Procedural Memory**:
    - The AI adapts over time, learning that users prefer examples over technical definitions.
    - *E.g.*: "Python can be used for data analysis. For example, here's a simple script that analyzes sales data."

Summary

- **Working Memory** helps AI handle the **current task or conversation**.
- **Episodic Memory** helps AI remember **past interactions** for better personalization.
- **Semantic Memory** enables AI to provide **factual knowledge**.
- **Procedural Memory** allows AI to improve **skills and processes** over time based on feedback.

Working Memory



Working memory encompasses your active understanding and contextualization of immediate information requiring dynamic processing. For a chatbot, this represents the maintenance and manipulation of conversational context observed throughout real-time interactions.

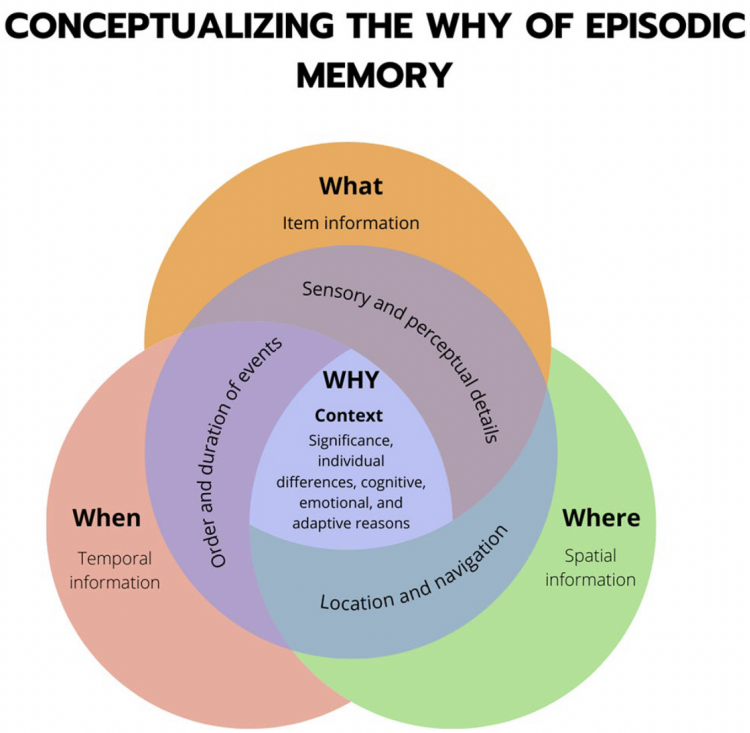
The type of information maintained in working memory consists of active messages and roles, current task/goal parameters, immediate state representations, and contextual processing requirements. This includes message history with associated metadata, conversation state vectors, goal hierarchies, and temporary computational results requiring immediate access.

```
chat_model . invoke([
    {
        "role": "system",
        "content": "You are a helpful AI Assistant.",
    },
    {
        "role": "user",
        "content": "Hello, how are you?",
    },
    {
```

```
        "role": "assistant",
        "content": "I'm doing well, thank you for asking.",
    },
    {
        "role": "user",
        "content": "Can you tell me a joke?",
    }
])
```

Remembering from working memory involves direct access to recent contextual data and action/result pairs. The system leverages immediate accessibility to maintain conversational coherence through continuous monitoring of the active message history, current state parameters, and ongoing computational processes. This direct access enables appropriate response generation grounded in the immediate conversational context.

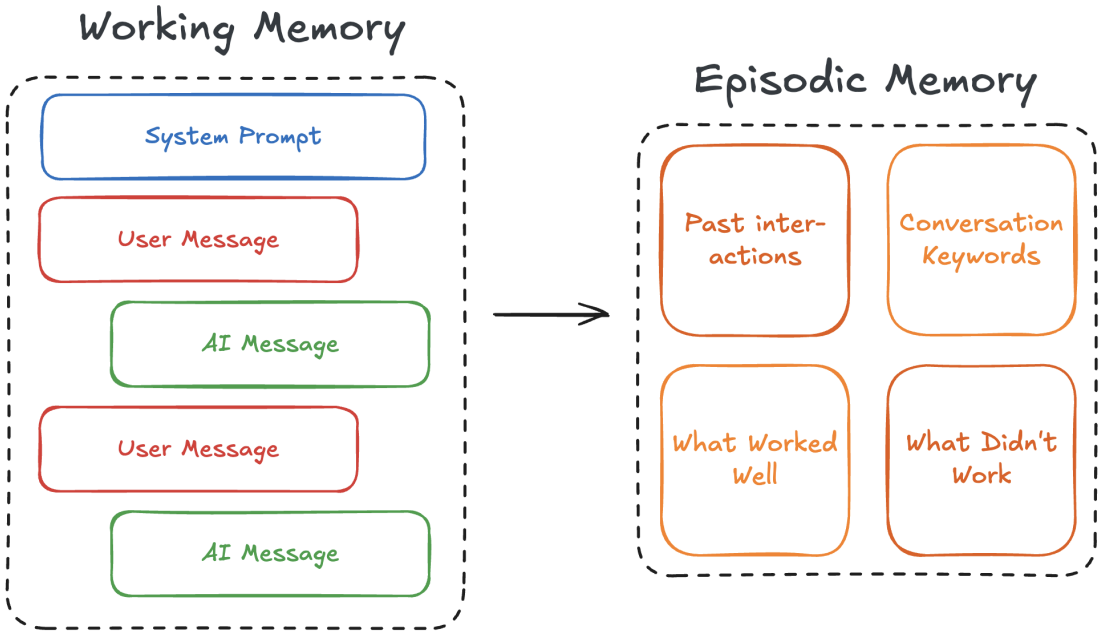
Episodic Memory



Episodic memory is a historical collection of prior experiences, or episodes. This can be both the literal recollection of how something happened and also any non-explicitly stated takeaways. When encountering a specific situation, you may recall similar related events that you've been in and their outcomes, which shape the way we approach new, comparable experiences.

For a chatbot, this includes both raw conversations it has participated in and the analytical understanding gained from those interactions. The act of remembering is implemented through dynamic few-shot prompting, automatically providing similar successful examples and instructions to better guide an LLM's response on subsequent similar queries.

But we don't just recall similar experiences - we also extract takeaways (or learning) from interactions. Learning in episodic memory happens through two processes: automatic storage of complete conversations, and generation of post-conversation analysis. The system stores full interaction sequences while implementing reflection protocols to identify what worked, what didn't, and what can be learned for future situations. This dual approach enables both specific recall and strategic learning for future conversations.



Episodic memory serves as the system's experiential foundation, allowing it to adapt its behavior based on accumulated conversation history while maintaining access to proven interaction patterns and their associated learnings. This creates a continuously improving system that learns not just from individual interactions, but from the patterns and insights derived across multiple conversations.

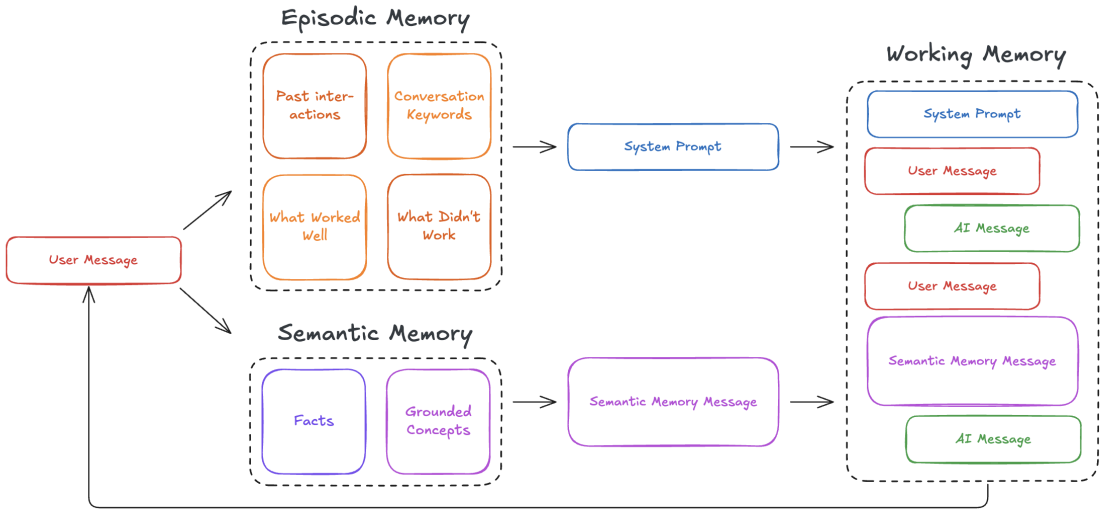
Let's implement this reflection, storage and retrieval:

Creating a Reflection Chai

This is where historical messages can be input, and episodic memories will be output. Given a message history, you will receive

```
{
    "context_tags": [
# 2-4 keywords that would help identify similar future conversations string,
# Use field-specific terms like "deep_learning", "methodology_question",
"results_interpretation" ... ],
    "conversation_summary": string,
# One sentence describing what the conversation accomplished "what_worked": string,
# Most effective approach or strategy used in this conversation "what_to_avoid": string
# Most important pitfall or ineffective approach to avoid }
```

3. Semantic Memory:



Semantic memory represents our structured knowledge of facts, concepts, and their relationships - essentially what we "know" rather than what we "remember experiencing." This type of memory allows us to understand and interact with the world by accessing our accumulated knowledge. For a chatbot, semantic memory would consist of its knowledge base and retrieval system, containing documentation, technical information, and general knowledge that can be accessed to provide accurate and informed responses.

The key difference from episodic memory is that semantic memory isn't tied to specific experiences or events - it's about understanding concepts and facts in an abstract way. In an AI system, this would be implemented through techniques like Retrieval Augmented Generation (RAG), where relevant information is dynamically pulled from a knowledge base to ground and inform responses.

4.

What is Procedural Memory?

Procedural memory is like a **"how-to" memory**. It doesn't focus on facts or information but rather on **how to perform tasks or solve problems**. In this case, we’re storing **strategies** or **approaches** that worked well or didn't work well.

For example:

- **What worked**: If a certain way of solving a problem was successful (like a specific approach to studying), you store that in **what_worked**.
- **What to avoid**: If a certain way of solving a problem didn’t work (like a strategy that led to confusion), you store that in **what_to_avoid**.

Example Breakdown

Let’s use a **study strategy** example:

1. **Initial Interaction**: A user asks the AI for a study strategy. The AI gives a strategy it knows, but it needs feedback to know if it was successful.
2. **Feedback Collection**: The AI asks the user if the strategy worked or failed. Based on the user's answer, the system updates its procedural memory.

Here's how it works step-by-step:

1. **User Input**: User asks, "What study strategy should I use?"
   - The AI gives a study plan.
2. **User Feedback**: User responds, "This worked for me!"
   - The AI stores this strategy in **what_worked**.
3. **User Input**: User asks, "What should I avoid during study?"
   - The AI checks if any previous strategies were bad (e.g., the user didn't like a strategy earlier).
4. **User Feedback**: User responds, "Avoid studying late at night!"
   - The AI stores this in **what_to_avoid**.

Revised Example Code with More Detailed Comments:

```
python
Copy code
from langchain_openai import ChatOpenAI
from langchain_core.messages import HumanMessage, SystemMessa
ge

# Initialize the language model (replace with your OpenAI API
key)
llm = ChatOpenAI(temperature=0.7, model="gpt-4")

# Define System Prompt (How the AI behaves in the conversatio
n)
system_prompt = SystemMessage("You are a helpful AI Assistan
t. Answer the User's queries with factual information and ins
tructions.")
```

```
# Storage for Procedural Memory: Keeps track of strategies th
at worked or failed
what_worked = set()  # Strategies or actions that worked
what_to_avoid = set()  # Strategies or actions that should be
avoided

def procedural_memory_update(what_worked, what_to_avoid):
    """ Updates procedural memory with strategies that worked
or should be avoided. """
    print("\n== Procedural Memory Update ==")

    # Example: Getting user feedback to update memory
    if 'good strategy' in input("Did this strategy work well?
(yes/no) ").lower():
        what_worked.add("used best study strategy")
    if 'bad strategy' in input("Did this strategy fail? (yes/
no) ").lower():
        what_to_avoid.add("avoid study at night")

    print("What Worked:", what_worked)
    print("What to Avoid:", what_to_avoid)

# Main Loop for Procedural Interaction
while True:
    # Get User's Message (Input from user)
    user_message = HumanMessage(input("\nUser: "))

    if user_message.content.lower() == "exit":
        # Update procedural memory and exit the program
        procedural_memory_update(what_worked, what_to_avoid)
        print("\n == Conversation Ended and Procedural Memory
Updated ==")
        break

    elif user_message.content.lower() == "exit_quiet":
```

```
        print("\n == Conversation Exited Quietly ==")
        break

    else:
        # Process user input and generate response
        print("\nAI: Let's work on your study strategy!")

        # Example: Check if the user asks for study advice
        if "study" in user_message.content.lower():
            if "used best study strategy" in what_worked:
                print("\nAI: Based on previous memory, let's
use the study strategy that worked for you!")
            else:
                print("\nAI: I have no prior memory of study
strategies. Let me suggest a new approach!")

        # Check if the user asks about what to avoid
        elif "avoid" in user_message.content.lower():
            if "avoid study at night" in what_to_avoid:
                print("\nAI: Based on previous memory, avoid
studying at night, as it didn't work for you.")
            else:
                print("\nAI: No memory of study mistakes rela
ted to this, let me suggest something to avoid.")

        # Ask the user for feedback to update procedural memo
ry
        procedural_memory_update(what_worked, what_to_avoid)

    # Optionally, store new procedural knowledge based on use
r input
    if "learn" in user_message.content.lower():
        print("\nAI Message: I've learned a new strategy!")

# To check the final procedural memory (for debug purposes)
print("\nFinal Procedural Memory:")
```

```
print("What Worked:", what_worked)
print("What to Avoid:", what_to_avoid)
```

Detailed Example Walkthrough:

1. **User's Initial Question**:
   - **User**: "What study strategy should I use?"
   - **AI**: Responds with a general study strategy, for example: "Start with small tasks and break your study into timed sessions."
   - AI then asks for feedback to see if this strategy worked.
2. **User Feedback**:
   - **User**: "This worked for me."
   - The AI will store this **study strategy** as something that worked by adding it to `what_worked`.
3. **Next User Question**:
   - **User**: "What should I avoid while studying?"
   - The AI checks its memory and realizes that **studying late at night** didn't work for the user in the past (stored in `what_to_avoid`).
4. **User Feedback on What to Avoid**:
   - **User**: "Avoid studying late at night."
   - The AI updates `what_to_avoid` with this information.
5. **End Conversation**:
   - **User**: "exit"
   - AI will print out the final memory of what worked and what to avoid, such as:
     - **What Worked**: {'used best study strategy'}
     - **What to Avoid**: {'avoid study at night'}

Sample Interaction:

```
vbnet
Copy code
User: What study strategy should I use?
AI: Let's work on your study strategy!
AI: I have no prior memory of study strategies. Let me sugges
t a new approach!
Did this strategy work well? (yes/no): yes

User: What should I avoid while studying?
AI: Based on previous memory, avoid studying at night, as it
didn't work for you.
Did this strategy fail? (yes/no): yes

User: exit
 == Conversation Ended and Procedural Memory Updated ==
Final Procedural Memory:
What Worked: {'used best study strategy'}
What to Avoid: {'avoid study at night'}
```

Conclusion:

This **procedural memory system** allows the AI to adapt and recall what strategies or actions worked well or should be avoided based on feedback from the user. This feedback loop helps the system refine its responses over time by storing strategies that the user finds helpful and avoiding those that were unhelpful.