```python
import os
from datetime import datetime
from typing import List, Optional

from dotenv import load_dotenv
from loguru import logger
from pydantic import BaseModel, Field
from swarm_models import OpenAIChat

from swarms import Agent
from swarms.prompts.finance_agent_sys_prompt import (
    FINANCIAL_AGENT_SYS_PROMPT,
)

load_dotenv()

# Get the OpenAI API key from the environment variable
api_key = os.getenv("OPENAI_API_KEY")

# Create an instance of the OpenAIChat class
model = OpenAIChat(
    openai_api_key=api_key,
    model_name="gpt-4o-mini",
    temperature=0.1,
    max_tokens=2000,
)
```

```
# Initialize the agent
agent = Agent(
    agent_name="Financial-Analysis-Agent",
    system_prompt=FINANCIAL_AGENT_SYS_PROMPT,
    llm=model,
    max_loops=1,
    autosave=True,
    dashboard=False,
    verbose=True,
    dynamic_temperature_enabled=True,
    saved_state_path="finance_agent.json",
    user_name="swarms_corp",
    retry_attempts=1,
    context_length=200000,
    return_step_meta=False,
    # output_type="json",
    output_type=str,
)


class ThoughtLog(BaseModel):
    """
    Pydantic model to log each thought generated by the agent.
    """
    thought: str

    timestamp: datetime = Field(default_factory=datetime.now)

    recursion_depth: int



class MemoryLog(BaseModel):

    """

    Pydantic model to log memory states during the agent's execution.

    """


    thoughts: List[ThoughtLog] = []

    final_result: Optional[str] = None

    completion_status: bool = False

    task: str



class RecursiveAgent(Agent):

    """

    An autonomous agent built on top of the Swarms Agent framework.

    Capable of recursively exploring tasks using a Tree of Thoughts mechanism.


    Attributes:

    - agent_name (str): The name of the agent.

    - system_prompt (str): The system prompt guiding the agent's behavior.

    - max_loops (int): The maximum depth for recursion in the Tree of Thoughts.

    - memory_limit (int): The maximum number of thought logs to store.
- memory (MemoryLog): Pydantic model to store thoughts and logs.
"""


def __init__(
    self,
    agent_name: str,
    system_prompt: str,
    max_loops: int,
    memory_limit: int = 5,
    agent: Agent = agent,
    *args,
    **kwargs,
) -> None:
    """
    Initialize the RecursiveAgent.


    :param agent_name: Name of the agent.
    :param system_prompt: The prompt guiding the agent's behavior.
    :param max_loops: The maximum number of recursive loops allowed.
    :param memory_limit: Maximum number of memory entries.
    :param kwargs: Additional arguments passed to the base Agent.
    """
    super().__init__(agent_name=agent_name, **kwargs)
    self.system_prompt = system_prompt
    self.max_loops = max_loops
    self.memory = MemoryLog(task="")
```

```
        self.memory_limit = memory_limit  # Max thoughts to store
        self.finished = False  # Task completion flag
        self.agent = agent(
            agent_name=agent_name,
            system_prompt=system_prompt,
            max_loops=max_loops,
        )
        logger.info(
            f"Initialized agent {self.agent_name} with recursion limit of {self.max_loops}"
        )

    def add_to_memory(
        self, thought: str, recursion_depth: int
    ) -> None:
        """
        Add a thought to the agent's memory using the Pydantic ThoughtLog model.

        :param thought: The thought generated by the agent.
        :param recursion_depth: The depth of the current recursion.
        """
        if len(self.memory.thoughts) >= self.memory_limit:
            logger.debug(
                "Memory limit reached, discarding the oldest memory entry."
            )
            self.memory.thoughts.pop(0)  # Maintain memory size
        thought_log = ThoughtLog(
            thought=thought, recursion_depth=recursion_depth
        )
        self.memory.thoughts.append(thought_log)
        logger.info(
            f"Added thought to memory at depth {recursion_depth}: {thought}"
        )

    def check_if_finished(self, current_thought: str) -> bool:
        """
        Check if the task is finished by evaluating the current thought.

        :param current_thought: The current thought or reasoning result.
        :return: True if task completion keywords are found, else False.
        """
        # Define criteria for task completion based on keywords
        completion_criteria = [
            "criteria met",
            "task completed",
            "done",
            "fully solved",
        ]
        if any(
            keyword in current_thought.lower()
            for keyword in completion_criteria
        ):
            self.finished = True
            self.memory.completion_status = True
            logger.info(
                f"Task completed with thought: {current_thought}"
            )
        return self.finished

    def run_tree_of_thoughts(
        self, task: str, current_depth: int = 0
    ) -> Optional[str]:
        """
        Recursively explore thought branches based on the Tree of Thoughts mechanism.

        :param task: The task or query to be reasoned upon.
        :param current_depth: The current recursion depth.
        :return: The final solution or message indicating task completion or failure.
        """
        logger.debug(f"Current recursion depth: {current_depth}")
        if current_depth >= self.max_loops:
            logger.warning(
                "Max recursion depth reached, task incomplete."
            )
            return "Max recursion depth reached, task incomplete."

        # Generate multiple possible thoughts/branches using Swarms logic
        response = self.generate_thoughts(task)
        thoughts = self.extract_thoughts(response)
        self.memory.task = task  # Log the task in memory

        # Store thoughts in memory
        for idx, thought in enumerate(thoughts):
            logger.info(
                f"Exploring thought {idx + 1}/{len(thoughts)}: {thought}"
            )
            self.add_to_memory(thought, current_depth)

            if self.check_if_finished(thought):
                self.memory.final_result = (
                    thought  # Log the final result
                )
                return f"Task completed with thought: {thought}"

            # Recursive exploration
            result = self.run_tree_of_thoughts(
                thought, current_depth + 1
            )

            if self.finished:
                return result

        return "Exploration done but no valid solution found."

    def generate_thoughts(self, task: str) -> str:
        """
        Generate thoughts for the task using the Swarms framework.

        :param task: The task or query to generate thoughts for.
        :return: A string representing multiple thought branches generated by Swarms logic.
        """
        logger.debug(f"Generating thoughts for task: {task}")
        response = self.agent.run(
            task
        )  # Assuming Swarms uses an LLM for thought generation
        return response

    def extract_thoughts(self, response: str) -> List[str]:
        """
        Extract individual thoughts/branches from the LLM's response.

        :param response: The response string containing multiple thoughts.
        :return: A list of extracted thoughts.
        """
        logger.debug(f"Extracting thoughts from response: {response}")
        return [
            thought.strip()
            for thought in response.split("\n")
            if thought
        ]
```

```
def reflect(self) -> str:
    """
    Reflect on the task and thoughts stored in memory, providing a summary of the process.
    The reflection will be generated by the LLM based on the stored thoughts.

    :return: Reflection output generated by the LLM.
    """
    logger.debug("Running reflection on the task.")

    # Compile all thoughts into a prompt for reflection
    thoughts_for_reflection = "\n".join(
        [
            f"Thought {i + 1}: {log.thought}"
            for i, log in enumerate(self.memory.thoughts)
        ]
    )
    reflection_prompt = (
        f"Reflect on the following task and thoughts:\n"
        f"Task: {self.memory.task}\n"
        f"Thoughts:\n{thoughts_for_reflection}\n"
        "What did we learn from this? How could this process be improved?"
    )

    # Use the agent's LLM to generate a reflection based on the memory
    reflection_response = self.agent.run(reflection_prompt)
    self.memory.final_result = reflection_response
```

```
        logger.info(f"Reflection generated: {reflection_response}")
        return reflection_response


# # Example usage of the RecursiveAgent

# if __name__ == "__main__":
#     # Example initialization and running
#     agent_name = "Autonomous-Financial-Agent"
#     system_prompt = "You are a highly intelligent agent designed to handle financial queries
efficiently."
#     max_loops = 1

#     # Initialize the agent using Swarms
#     agent = RecursiveAgent(
#         agent_name=agent_name,
#         system_prompt=system_prompt,
#         max_loops=max_loops
#     )

#     # Define the task for the agent
#     task = "How can I establish a ROTH IRA to buy stocks and get a tax break? What are the
criteria?"

#     # Run the tree of thoughts mechanism
#     result = agent.run_tree_of_thoughts(task)

#     logger.info(f"Final result: {result}")


#     # Perform reflection

#     reflection = agent.reflect()

#     logger.info(f"Reflection: {reflection}")
```