```
"""
$ pip install swarms

- Add docs into the database
- Use better llm
- use better prompts [System and SOPs]
- Use a open source model like Command R
- Better SOPS ++ System Prompts
-
"""

from swarms import Agent
from swarm_models import OpenAIChat
from swarms_memory import ChromaDB
from swarms.tools.prebuilt.bing_api import fetch_web_articles_bing_api
import os
from dotenv import load_dotenv

load_dotenv()

# Let's create a text file with the provided prompt.

research_system_prompt = """
Research Agent LLM Prompt: Summarizing Sources and Content

Objective:
```

Your task is to summarize the provided sources and the content within those sources. The goal is to create concise, accurate, and informative summaries that capture the key points of the original content.

Instructions:

1. Identify Key Information:
   - Extract the most important information from each source. Focus on key facts, main ideas, significant arguments, and critical data.

2. Summarize Clearly and Concisely:
   - Use clear and straightforward language. Avoid unnecessary details and keep the summary concise.
   - Ensure that the summary is coherent and easy to understand.

3. Preserve Original Meaning:
   - While summarizing, maintain the original meaning and intent of the content. Do not omit essential information that changes the context or understanding.

4. Include Relevant Details:
   - Mention the source title, author, publication date, and any other relevant details that provide context.

5. Structure:
   - Begin with a brief introduction to the source.
   - Follow with a summary of the main content.

```
   - Conclude with any significant conclusions or implications presented in the source.

"""



def movers_agent_system_prompt():
    return """
    The Movers Agent is responsible for providing users with fixed-cost estimates for moving services

    based on the distance between their current location and destination, and the number of rooms in their home.
    Additionally, the agent allows users to attempt negotiation for better deals using the Retell API.

    Responsibilities:
    - Provide fixed-cost estimates based on distance and room size.
    - Allow users to attempt negotiation for better deals using the Retell API.

    Details:
     - Fixed Costs: Predefined costs for each of the 10 moving companies, with variations based on distance and number of rooms.
    - Distance Calculation: Use a fixed formula to estimate distances and costs.
     - Room Size: Standard sizes based on the number of rooms will be used to determine the base cost.
    - Negotiation: Users can click a "negotiate" button to initiate negotiation via Retell API.

    Tools and Resources Used:
```

```
    - Google Maps API: For calculating distances between the current location and destination.
    - Retell API: For simulating negotiation conversations.
    - Streamlit: For displaying estimates and handling user interactions.

    Example Workflow:
    1. User inputs their current location, destination, and number of rooms.
    2. The agent calculates the distance and estimates the cost using predefined rates.
    3. Displays the estimates from 10 different moving companies.
     4. Users can click "negotiate" to simulate negotiation via Retell API, adjusting the price within a predefined range.
    """


# Example usage


# Initialize
memory = ChromaDB(
    output_dir="research_base",
    n_results=2,
)

llm = OpenAIChat(
    temperature=0.2,
    max_tokens=3500,
    openai_api_key=os.getenv("OPENAI_API_KEY"),
)



# Initialize the agent
agent = Agent(
    agent_name="Research Agent",
    system_prompt=research_system_prompt,
    llm=llm,
    max_loops="auto",
    autosave=True,
    dashboard=False,
    interactive=True,
    # long_term_memory=memory,
    tools=[fetch_web_articles_bing_api],
)



# # Initialize the agent
# agent = Agent(
#     agent_name="Movers Agent",
#     system_prompt=movers_agent_system_prompt(),
#     llm=llm,
#     max_loops=1,
#     autosave=True,
#     dashboard=False,
#     interactive=True,
#     # long_term_memory=memory,
#     # tools=[fetch_web_articles_bing_api],
# )



def perplexity_agent(task: str = None, *args, **kwargs):
    """
    This function takes a task as input and uses the Bing API to fetch web articles related to the task.
    It then combines the task and the fetched articles as prompts and runs them through an agent.
    The agent generates a response based on the prompts and returns it.

    Args:
        task (str): The task for which web articles need to be fetched.

    Returns:
        str: The response generated by the agent.
    """
    out = fetch_web_articles_bing_api(
        task,
    )

    # Sources
    sources = [task, out]
    sources_prompts = "".join(sources)

    # Run a question
    agent_response = agent.run(sources_prompts)

    return agent_response



out = perplexity_agent(

     "What are the indian food restaurant names in standford university avenue? What are their cost ratios"

)

print(out)
```