```
from typing import Any, Dict, List, Optional


from langchain import hub
from langchain.agents import AgentExecutor, create_openai_tools_agent
from langchain_community.tools.tavily_search import (
    TavilySearchResults,
)
from langchain_openai import ChatOpenAI
from loguru import logger



class LangchainAgent:
    def __init__(
        self,
        tavily_api_key: str,
        llm_model: str = "gpt-3.5-turbo",
        temperature: float = 0.7,
        tavily_max_results: int = 1,
        prompt_hub_url: str = "hwchase17/openai-tools-agent",
        verbose: bool = True,
        log_file: Optional[str] = None,
        openai_api_key: Optional[str] = None,
    ) -> None:
        """
        Initializes the LangchainAgent with given tools and parameters.
        :param tavily_api_key: The API key for the Tavily search tool.
        :param llm_model: The OpenAI language model to be used (default: "gpt-3.5-turbo").
        :param temperature: Temperature for the language model (default: 0.7).
        :param tavily_max_results: Maximum results for the Tavily search (default: 1).
            :param prompt_hub_url: URL of the prompt hub to fetch the agent prompt (default:
"hwchase17/openai-tools-agent").
        :param verbose: If True, the agent will print detailed logs (default: True).
        :param log_file: Optional log file to store logs using Loguru.
        :param openai_api_key: Optional OpenAI API key for connecting to OpenAI services.
        """
        # Setup Loguru for logging
        if log_file:
            logger.add(log_file, rotation="500 MB")

        # Log initialization
        logger.info(
            "Initializing LangchainAgent with model: {}, temperature: {}",
            llm_model,
            temperature,
        )

        # Set up Tavily Search tool
        logger.info(
            "Setting up Tavily Search with max_results: {}",
            tavily_max_results,
        )
```

```
self.tavily_search = TavilySearchResults(
    api_key=tavily_api_key, max_results=tavily_max_results
)

# Tools list (can be expanded)
self.tools = [self.tavily_search]

# Initialize the LLM (OpenAI Chat model)
logger.info("Initializing OpenAI model: {}", llm_model)
self.llm = ChatOpenAI(
    model=llm_model,
    temperature=temperature,
    openai_api_key=openai_api_key,
)

# Fetch the prompt template from LangChain hub
logger.info(
    "Fetching prompt template from {}", prompt_hub_url
)
self.prompt = hub.pull(prompt_hub_url)

# Create the OpenAI Tools agent
logger.info(
    "Creating OpenAI Tools agent with fetched prompt and LLM."
)
self.agent = create_openai_tools_agent(
        self.llm, self.tools, self.prompt
    )

    # Create AgentExecutor with the agent and tools
    logger.info(
        "Setting up AgentExecutor with verbose: {}", verbose
    )
    self.agent_executor = AgentExecutor(
        agent=self.agent, tools=self.tools, verbose=verbose
    )

def run(
    self,
    task: str,
    chat_history: Optional[List[Dict[str, str]]] = None,
) -> str:
    """
    Run the LangchainAgent with a specific task.

    :param task: The task (query) for the agent to handle.
    :param chat_history: Optional previous chat history for context (default: None).
    :return: The result of the task.
    """
    logger.info("Running agent with task: {}", task)

    # Create input for agent execution
```

```
        input_data: Dict[str, Any] = {"input": task}
        if chat_history:
            logger.info("Passing chat history for context.")
            input_data["chat_history"] = chat_history

        # Invoke the agent
        logger.info("Invoking the agent executor.")
        result = self.agent_executor.invoke(input_data)

        # Log the result
        logger.info(
            "Task executed successfully. Result: {}", result["output"]
        )

        # Return the output from the agent
        # return result["output"]
        return result


# # Example usage:
# agent = LangchainAgent(
#     tavily_api_key="your_tavily_api_key",
#     llm_model="gpt-3.5-turbo",
#     temperature=0.5,
#     tavily_max_results=3,
#     prompt_hub_url="your-prompt-url",
#     verbose=True,

#     log_file="agent.log",

#     openai_api_key="your_openai_api_key"

# )

# result = agent.run("What is LangChain?")

# print(result)
```