Agentic AI and MCP Demonstration using LangChain

This document contains a complete Python program that demonstrates the concepts of Agentic AI and Multi-Component Pipelines (MCP) using the LangChain framework. It shows how different components—Planner, Executor, Memory, and Reflector—work together to form an Agentic AI loop (Plan → Act → Observe → Reflect). The demonstration is ideal for classroom illustration of modern autonomous agent architectures.

```
from langchain.chat_models import ChatOpenAI
from langchain.memory import ConversationBufferMemory
from langchain.prompts import ChatPromptTemplate
from langchain.chains import LLMChain
import random, time

# ---------------------------
# 1■■ SETUP: MODEL + MEMORY
# ---------------------------
llm = ChatOpenAI(model="gpt-4o-mini", temperature=0.7)
memory = ConversationBufferMemory(memory_key="history")

# ---------------------------
# 2■■ PLANNER CHAIN
# ---------------------------
planner_prompt = ChatPromptTemplate.from_template("""
You are an intelligent planner.
Given the user goal: "{goal}"
Break it down into 3 specific, actionable steps.
""")

planner_chain = LLMChain(llm=llm, prompt=planner_prompt, memory=memory)

# ---------------------------
# 3■■ EXECUTOR (Simulated Tool)
# ---------------------------
def execute_step(step):
    print(f"\n■■ Executing step: {step}")
    time.sleep(1)
    outcomes = ["■ Success", "■■ Partial success", "■ Failure"]
    result = random.choice(outcomes)
    print(f"Result: {result}")
    return f"{step} → {result}"

# ---------------------------
# 4■■ REFLECTION CHAIN
# ---------------------------
reflection_prompt = ChatPromptTemplate.from_template("""
You are a reflective AI agent.
Given the recent task outcomes:
{history}
Reflect and suggest whether to retry, refine, or proceed.
""")

reflection_chain = LLMChain(llm=llm, prompt=reflection_prompt, memory=memory)

# ---------------------------
```

```
# 5■■ AGENTIC ORCHESTRATOR
# ---------------------------
def agentic_loop(goal):
    print("\n■ Starting Agentic AI Loop for goal:", goal)

    # PLAN
    plan = planner_chain.run(goal=goal)
    print("\n■ PLAN GENERATED:\n", plan)

    # EXECUTE
    steps = [s.strip() for s in plan.split("\n") if s.strip()]
    for step in steps:
        result = execute_step(step)
        memory.save_context({"step": step}, {"result": result})

    # REFLECT
    reflection = reflection_chain.run()
    print("\n■ REFLECTION RESULT:\n", reflection)

    # MEMORY DUMP
    print("\n■ MEMORY SUMMARY:")
    print(memory.load_memory_variables({})["history"])

# ---------------------------
# 6■■ RUN DEMO
# ---------------------------
if __name__ == "__main__":
    goal = input("■ Enter a goal for the agent: ")
    agentic_loop(goal)
```

■ Key Concepts Demonstrated - **Planner:** Uses LangChain's `LLMChain` to decompose a user goal into actionable steps. - **Executor:** Simulates real-world task execution (could be replaced by APIs or tools). - **Memory:** Maintains the context of previous runs using `ConversationBufferMemory`. - **Reflector:** Performs self-assessment of progress and suggests next actions. - **Agentic Orchestrator:** Manages the entire pipeline to complete the goal. This example demonstrates the foundational logic behind modern autonomous AI systems and provides students with an intuitive understanding of LangChain’s modular design.