Agentic AI design patterns

Mia Mohammad Imran

Announcements

- No paper review/in-class activities this week
- **Instead: think about the class project**
  - Single/2 person (max) team
  - Can be Research/Development project
  - **Draft one-page project overview and submit on canvas by Feb 28**

So far…

We have talked about

- Common Prompt patterns
- Applying RAG in Prompting
- Building an AI Agent

Question for you:

- Did you observe any difficulties?
- What are the common problem you faced?
- Did you get overwhelmed?

4 types of agentic AI design patterns

- **Reflection pattern**
- **Tool use pattern**
- Planning pattern
- Multi-agent pattern

Reflection pattern

- **Self-feedback mechanism**: The agent reviews its own outputs before finalizing its response or action.
- **Continuous improvement**: Allows agents to identify errors, refine methods, and improve over time, with adjustments in future interactions.

- Real-world example: GitHub Copilot refines its code through self-reflection:
  - **Initial response**: Generates code based on a prompt.
  - **Reflection process**: Reviews code for errors and inefficiencies.
  - **Self-iteration**: Refines logic and suggests optimizations based on feedback.

Reflection pattern

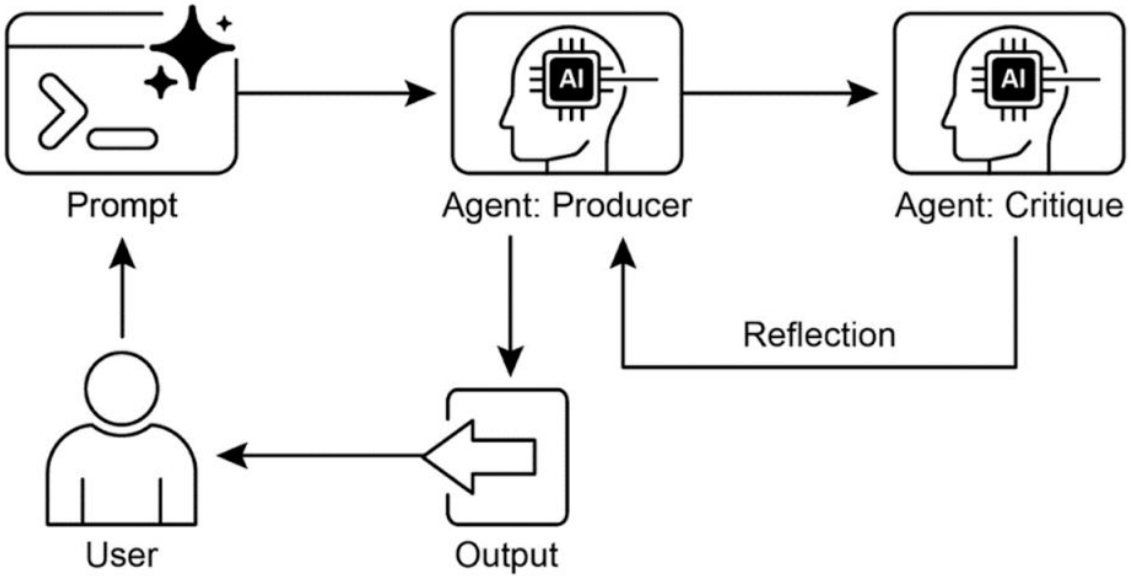


Fig. 4.2 Reflection design pattern, producer and critique agent

Reflection pattern: Steps

- **Execution**: The agent performs a task or generates an initial output.
- **Evaluation/Critique**: The agent analyzes the result for correctness, completeness, coherence, or instruction adherence.
- **Reflection/Refinement**: Based on the critique, the agent improves the output or adjusts the approach.
- **Iteration (Optional but common)**: The improved version is executed again, and the reflection cycle repeats until quality criteria are met.

Reflection pattern have two separate Logical Units

- **The Producer Agent**: primary responsibility is to perform the initial execution of the task. It focuses entirely on generating the content.
- **The Critic Agent**: evaluate the output generated by the Producer. It is given a different set of instructions, often a distinct persona (e.g., “You are a senior software engineer,” “You are a meticulous fact-checker”).

Code Generation and Debugging

Writing code, identifying errors, and fixing them

- **Use Case: An agent writing a Python function**
- **Reflection**:
  - Write initial code
  - run tests or static analysis
  - identify errors or inefficiencies, then modify the code based on the findings
- **Benefit**: Generates more robust and functional code

Complex Problem Solving

Evaluating intermediate steps or proposed solutions in multi-step reasoning tasks

- **Use Case: An agent solving a logic puzzle**
- **Reflection:**
  - Propose a step
  - evaluate if it leads closer to the solution or introduces contradictions
  - backtrack or choose a different step if needed
- **Benefit:** Improves the agent’s ability to navigate complex problem spaces.

Code Refactoring

Agent refactors a module.
Reflection stage asks:

- Did I preserve behavior?
- Did I modify public interfaces?
- Did I introduce performance regressions?
- Are edge cases handled?

Revised output follows.

Summarization and Information Synthesis

- Refining summaries for accuracy, completeness, and conciseness
- **Use Case**: An agent summarizing a long document
- **Reflection**:
  - Generate an initial summary
  - Compare it against key points in the original document
  - Refine the summary to include missing information or improve accuracy
- **Benefit**: Creates more accurate and comprehensive summaries

Tool use pattern

- **Tool use pattern**: Enhances LLMs by allowing dynamic interaction with external tools and resources.
- **MCP protocol**: Standardizes the tool use process, enabling agents to go beyond training data for real-world tasks.

Real-world example: Anytime you ask a question to ChatGPT and it crawls internet to get response.

Tool use pattern

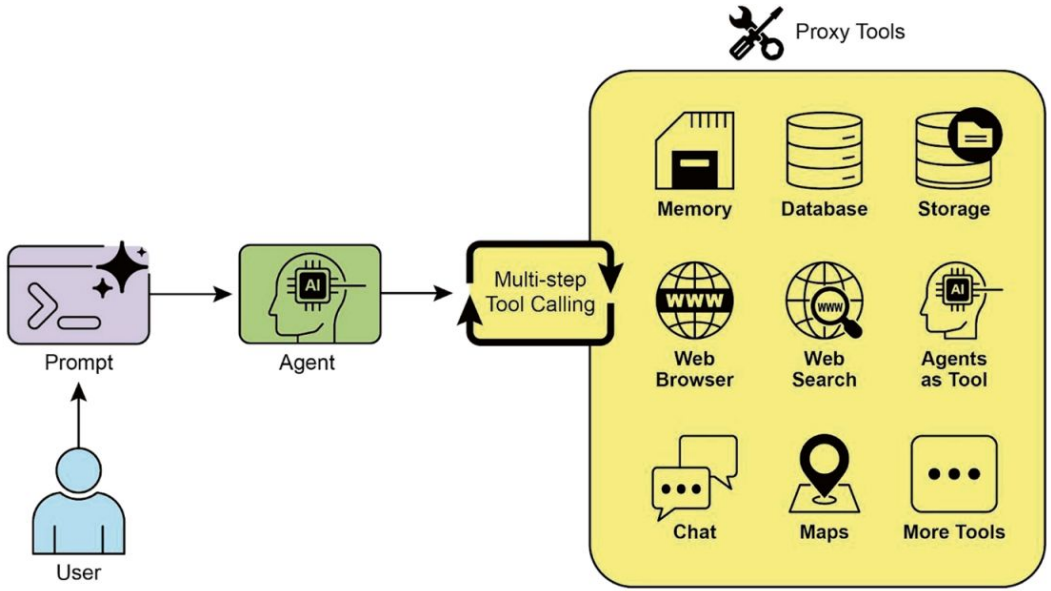


Fig. 5.1 Some examples of an Agent using Tools

Tool use pattern

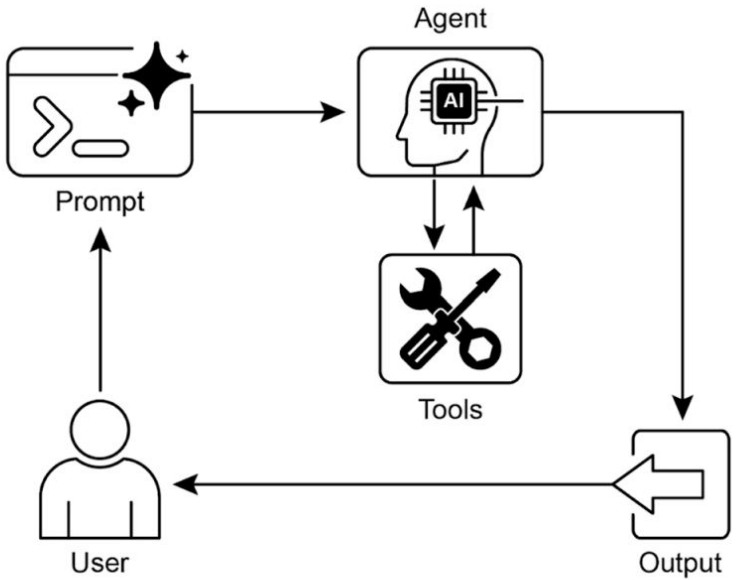


Fig. 5.2 Tool use design pattern

Tool use pattern: Steps

- **Tool Definition:** External functions are described to the LLM, including name, purpose, and parameters
- **LLM Decision:** The LLM determines whether a tool is needed to satisfy the user request
- **Function Call Generation:** If needed, the LLM produces a structured output specifying the tool name and arguments
- **Tool Execution:** The system executes the requested external function with the provided inputs
- **Observation/Result:** The tool's output is returned to the agent
- **LLM Processing (Optional but common):** The LLM uses the tool output to generate the final response or decide the next action

Information Retrieval from External Sources

Accessing real-time data or information which is not present in the LLM's training data

- **Use Case: Consider A Weather App**
- **Tool**: A weather API that takes a location and returns the current weather conditions
- **Agent Flow:**
  - User asks, "What's the weather in London?"
  - LLM identifies the need for the weather tool
    - calls the tool with "London"
    - tool returns data
  - LLM formats the data into a user-friendly response.

Interacting with Databases and APIs

Performing queries, updates, or other operations on structured data

- **Use Case: An e-commerce agent:**
- **Tools**: API calls to check product inventory, get order status, or process payments.
- **Agent Flow**:
  - User asks “Is product X in stock?”
  - LLM calls inventory API
    - tool returns stock count
  - LLM tells the user the stock status.

Performing Calculations and Data Analysis

Using external calculators, data analysis libraries, or statistical tools

- **Use Case: A financial agent**
- **Tools**: A calculator function, a stock market data API, a spreadsheet tool
- **Agent Flow**:
  - User asks “What’s the current price of AAPL and calculate the potential profit if I bought 100 shares at $150?”
  - LLM calls stock API
    - gets current price
  - Then calls calculator tool, gets result, formats response

Sending Communications

Sending emails, messages, or making API calls to external communication services

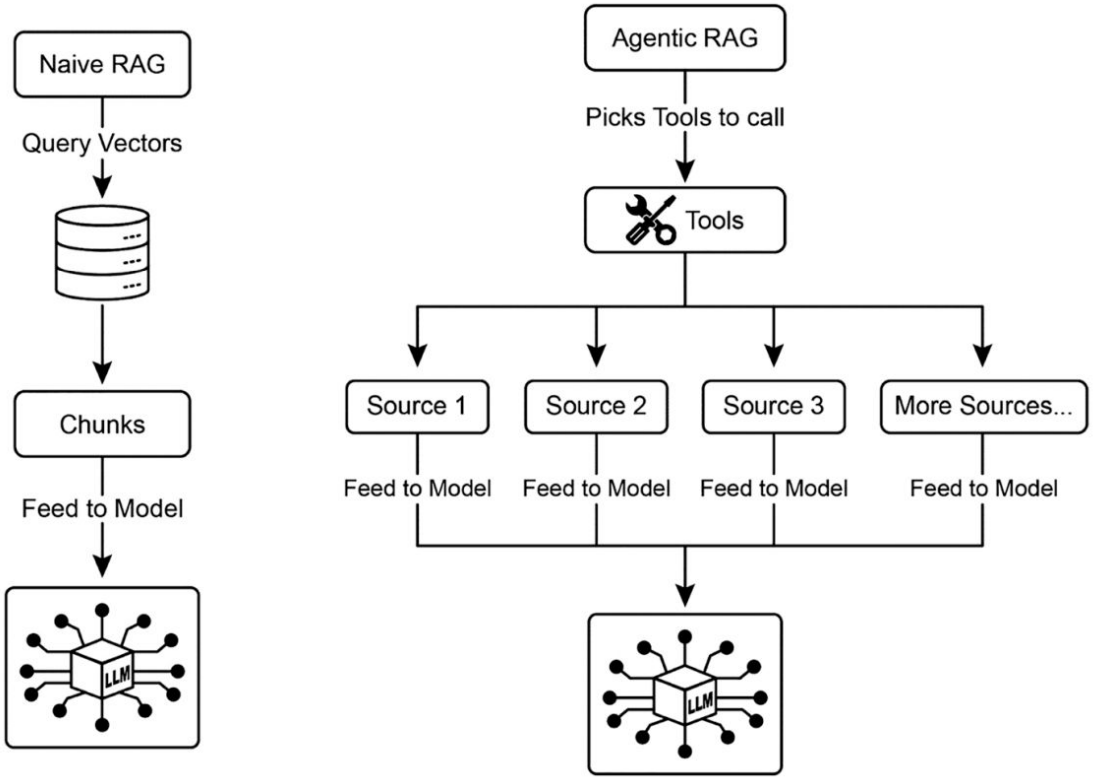
- **Use Case: A personal assistant agent**
- **Tool:** An email sending API
- **Agent Flow:**
  - User says “Send an email to John about the meeting tomorrow”
  - LLM calls an email tool with recipient, subject, and body extracted from the request

Executing Code

Running code snippets in a safe environment to perform specific tasks.

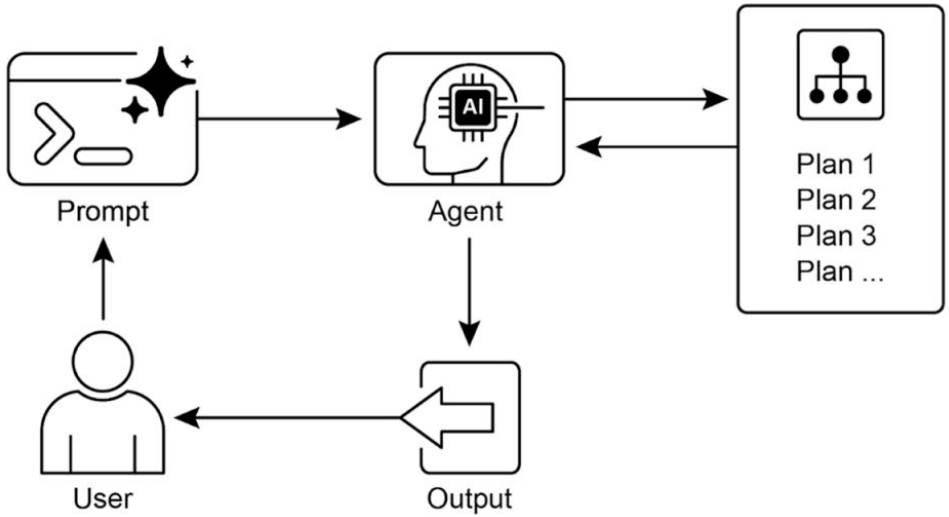
- **Use Case: A coding assistant agent**
- **Tool**: A code interpreter
- **Agent Flow**:
  - User provides a Python snippet and asks “What does this code do?”
  - LLM uses the interpreter tool to run the code and analyze its output

RAG and Tool Use Pattern



Planning pattern

- **Planning pattern**: Helps LLMs break down large tasks into smaller, manageable subtasks and organize them logically.
- **Execution**: Agents may plan tasks linearly or in parallel, depending on complexity.



Robotics and Autonomous Navigation

- Generates action sequences from initial state to goal state
- Optimizes for time, energy, or cost
- Respects environmental constraints such as obstacles or regulations

Structured Information Synthesis

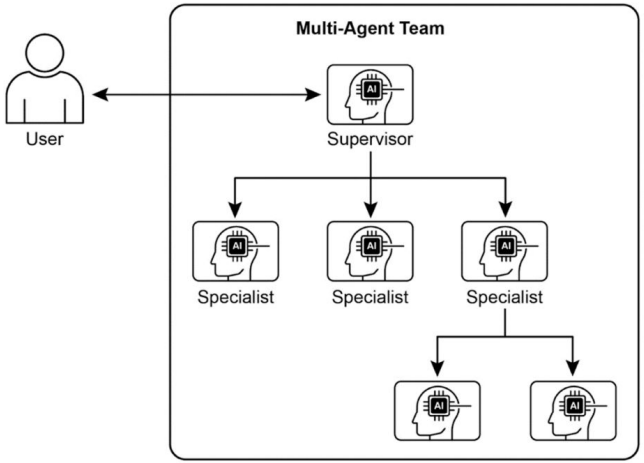
- Plans phases for complex outputs such as research reports
  - Information gathering
  - Summarization
  - Structuring
  - Iterative refinement

Procedural Task Automation

- Think about onboarding a Novice Software Developer in a project
  - Create system accounts
  - Assign training modules
  - Coordinate across departments
- Manages dependencies and invokes required tools

Multi-agent pattern

- **Multi-agent pattern**: Involves assigning distinct tasks to different agents, which can be prompted by a single LLM or multiple LLMs.
- **A2A communication**: Agents communicate using protocols like Google's A2A, which provide structured context and tools.



Complex Research & Analysis

- Research agent retrieves sources
- Summarization agent extracts key findings
- Trend agent identifies patterns
- Synthesis agent produces final report
- Mirrors human research teams

Software Development

- Requirements analyst agent
- Code generation agent
- Testing agent
- Documentation agent
- Sequential output passing and verification

Financial & Market Analysis

- Stock data retrieval agent
- News sentiment analysis agent
- Technical analysis agent
- Recommendation agent

Demo

https://github.com/imranraad07/CS-5001-AI-Augmented-SE/tree/master/Week_4

Summary

1. Reflection Pattern: Agent reviews and critiques its own output before finalizing
- Involves Producer and Critic logical roles
- Improves correctness, robustness, and reasoning quality
- Useful for debugging, refactoring, summarization, and multi-step reasoning

2. Tool Use Pattern: Agent dynamically invokes external tools, APIs, databases, or interpreters
- Flow: Define tool → LLM decides → Tool call → Execute → Process result
- Enables real-time data access, computation, code execution, and automation
- Foundation for RAG and practical AI systems

3. Planning Pattern: Breaks large tasks into smaller, structured subtasks
- Supports linear or parallel execution
- multi-step software engineering problems

4. Multi-Agent Pattern: Multiple specialized agents collaborate
- Supports structured agent-to-agent communication