Frequently Asked Questions: Agentic Design Patterns

What is an "agentic design pattern"? An agentic design pattern is a reusable, high-level solution to a common problem encountered when building intelligent, autonomous systems (agents). These patterns provide a structured framework for designing agent behaviors, much like software design patterns do for traditional programming. They help developers build more robust, predictable, and effective AI agents.

What is the main goal of this guide? The guide aims to provide a practical, hands-on introduction to designing and building agentic systems. It moves beyond theoretical discussions to offer concrete architectural blueprints that developers can use to create agents capable of complex, goal-oriented behavior in a reliable way.

Who is the intended audience for this guide? This guide is written for AI developers, software engineers, and system architects who are building applications with large language models (LLMs) and other AI components. It is for those who want to move from simple prompt-response interactions to creating sophisticated, autonomous agents.

4. What are some of the key agentic patterns discussed? Based on the table of contents, the guide covers several key patterns, including:

- **Reflection:** The ability of an agent to critique its own actions and outputs to improve performance.
- **Planning:** The process of breaking down a complex goal into smaller, manageable steps or tasks.
- **Tool Use:** The pattern of an agent utilizing external tools (like code interpreters, search engines, or other APIs) to acquire information or perform actions it cannot do on its own.
- **Multi-Agent Collaboration:** The architecture for having multiple specialized agents work together to solve a problem, often involving a "leader" or "orchestrator" agent.
- **Human-in-the-Loop:** The integration of human oversight and intervention, allowing for feedback, correction, and approval of an agent's actions.

Why is "planning" an important pattern? Planning is crucial because it allows an agent to tackle complex, multi-step tasks that cannot be solved with a single action. By creating a plan, the agent can maintain a coherent strategy, track its progress, and handle errors or unexpected obstacles in a structured manner. This prevents the agent from getting "stuck" or deviating from the user's ultimate goal.

What is the difference between a "tool" and a "skill" for an agent? While the terms are often used interchangeably, a "tool" generally refers to an external resource the agent can call upon (e.g., a weather API, a calculator). A "skill" is a more integrated capability that the agent has learned, often combining tool use with internal reasoning to perform a specific function (e.g., the skill of "booking a flight" might involve using calendar and airline APIs).

How does the "Reflection" pattern improve an agent's performance? Reflection acts as a form of self-correction. After generating a response or completing a task, the agent can be prompted to review its work, check for errors, assess its quality against certain criteria, or consider alternative approaches. This iterative refinement process helps the agent produce more accurate, relevant, and high-quality results.

What is the core idea of the Reflection pattern? The Reflection pattern gives an agent the ability to step back and critique its own work. Instead of producing a final output in one go, the agent generates a draft and then "reflects" on it, identifying flaws, missing information, or areas for improvement. This self-correction process is key to enhancing the quality and accuracy of its responses.

Why is simple "prompt chaining" not enough for high-quality output? Simple prompt chaining (where the output of one prompt becomes the input for the next) is often too basic. The model might just rephrase its previous output without genuinely improving it. A true Reflection pattern requires a more structured critique, prompting the agent to analyze its work against specific standards, check for logical errors, or verify facts.

What are the two main types of reflection mentioned in this chapter? The chapter discusses two primary forms of reflection:

- **"Check your work" Reflection:** This is a basic form where the agent is simply asked to review and fix its previous output. It's a good starting point for catching simple errors.
- **"Internal Critic" Reflection:** This is a more advanced form where a separate, "critic" agent (or a dedicated prompt) is used to evaluate the output of the "worker" agent. This critic can be given specific criteria to look for, leading to more rigorous and targeted improvements.

How does reflection help in reducing "hallucinations"? By prompting an agent to review its work, especially by comparing its statements against a known source or by checking its own reasoning steps, the Reflection pattern can significantly reduce the likelihood of hallucinations (making up facts). The agent is forced to be more grounded in the provided context and less likely to generate unsupported information.

Can the Reflection pattern be applied more than once? Yes, reflection can be an iterative process. An agent can be made to reflect on its work multiple times, with each loop refining the output further. This is particularly useful for complex tasks where the first or second attempt may still contain subtle errors or could be substantially improved.

What is the Planning pattern in the context of AI agents? The Planning pattern involves enabling an agent to break down a complex, high-level goal into a sequence of smaller, actionable steps. Instead of trying to solve a big problem at once, the agent first creates a "plan" and then executes each step in the plan, which is a much more reliable approach.

Why is planning necessary for complex tasks? LLMs can struggle with tasks that require multiple steps or dependencies. Without a plan, an agent might lose track of the overall

objective, miss crucial steps, or fail to handle the output of one step as the input for the next. A plan provides a clear roadmap, ensuring all requirements of the original request are met in a logical order.

What is a common way to implement the Planning pattern? A common implementation is to have the agent first generate a list of steps in a structured format (like a JSON array or a numbered list). The system can then iterate through this list, executing each step one by one and feeding the result back to the agent to inform the next action.

How does the agent handle errors or changes during execution? A robust planning pattern allows for dynamic adjustments. If a step fails or the situation changes, the agent can be prompted to "re-plan" from the current state. It can analyze the error, modify the remaining steps, or even add new ones to overcome the obstacle.

Does the user see the plan? This is a design choice. In many cases, showing the plan to the user first for approval is a great practice. This aligns with the "Human-in-the-Loop" pattern, giving the user transparency and control over the agent's proposed actions before they are executed.

What does the "Tool Use" pattern entail? The Tool Use pattern allows an agent to extend its capabilities by interacting with external software or APIs. Since an LLM's knowledge is static and it can't perform real-world actions on its own, tools give it access to live information (e.g., Google Search), proprietary data (e.g., a company's database), or the ability to perform actions (e.g., send an email, book a meeting).

How does an agent decide which tool to use? The agent is typically given a list of available tools along with descriptions of what each tool does and what parameters it requires. When faced with a request it can't handle with its internal knowledge, the agent's reasoning ability allows it to select the most appropriate tool from the list to accomplish the task.

What is the "ReAct" (Reason and Act) framework mentioned in this context? ReAct is a popular framework that integrates reasoning and acting. The agent follows a loop of **Thought** (reasoning about what it needs to do), **Action** (deciding which tool to use and with what inputs), and **Observation** (seeing the result from the tool). This loop continues until it has gathered enough information to fulfill the user's request.

What are some challenges in implementing tool use? Key challenges include:

- **Error Handling:** Tools can fail, return unexpected data, or time out. The agent needs to be able to recognize these errors and decide whether to try again, use a different tool, or ask the user for help.
- **Security:** Giving an agent access to tools, especially those that perform actions, has security implications. It's crucial to have safeguards, permissions, and often human approval for sensitive operations.
- **Prompting:** The agent must be prompted effectively to generate correctly formatted tool calls (e.g., the right function name and parameters).

What is the Human-in-the-Loop (HITL) pattern? HITL is a pattern that integrates human oversight and interaction into the agent's workflow. Instead of being fully autonomous, the agent pauses at critical junctures to ask for human feedback, approval, clarification, or direction.

Why is HITL important for agentic systems? It's crucial for several reasons:

- **Safety and Control:** For high-stakes tasks (e.g., financial transactions, sending official communications), HITL ensures a human verifies the agent's proposed actions before they are executed.
- **Improving Quality:** Humans can provide corrections or nuanced feedback that the agent can use to improve its performance, especially in subjective or ambiguous tasks.
- **Building Trust:** Users are more likely to trust and adopt an AI system that they can guide and supervise.

At what points in a workflow should you include a human? Common points for human intervention include:

- **Plan Approval:** Before executing a multi-step plan.
- **Tool Use Confirmation:** Before using a tool that has real-world consequences or costs money.
- **Ambiguity Resolution:** When the agent is unsure how to proceed or needs more information from the user.
- **Final Output Review:** Before delivering the final result to the end-user or system.

Isn't constant human intervention inefficient? It can be, which is why the key is to find the right balance. HITL should be implemented at critical checkpoints, not for every single action. The goal is to build a collaborative partnership between the human and the agent, where the agent handles the bulk of the work and the human provides strategic guidance.

What is the Multi-Agent Collaboration pattern? This pattern involves creating a system composed of multiple specialized agents that work together to achieve a common goal. Instead of one "generalist" agent trying to do everything, you create a team of "specialist" agents, each with a specific role or expertise.

What are the benefits of a multi-agent system?

- **Modularity and Specialization:** Each agent can be fine-tuned and prompted for its specific task (e.g., a "researcher" agent, a "writer" agent, a "code" agent), leading to higher quality results.
- **Reduced Complexity:** Breaking a complex workflow down into specialized roles makes the overall system easier to design, debug, and maintain.
- **Simulated Brainstorming:** Different agents can offer different perspectives on a problem, leading to more creative and robust solutions, similar to how a human team works.

What is a common architecture for multi-agent systems? A common architecture involves an **Orchestrator Agent** (sometimes called a "manager" or "conductor"). The orchestrator understands the overall goal, breaks it down, and delegates sub-tasks to the appropriate specialist agents. It then collects the results from the specialists and synthesizes them into a final output.

How do the agents communicate with each other? Communication is often managed by the orchestrator. For example, the orchestrator might pass the output of the "researcher" agent to the "writer" agent as context. A shared "scratchpad" or message bus where agents can post their findings is another common communication method.

Why is evaluating an agent more difficult than evaluating a traditional software program? Traditional software has deterministic outputs (the same input always produces the same output). Agents, especially those using LLMs, are non-deterministic and their performance can be subjective. Evaluating them requires assessing the *quality* and *relevance* of their output, not just whether it's technically "correct."

What are some common methods for evaluating agent performance? The guide suggests a few methods:

- **Outcome-based Evaluation:** Did the agent successfully achieve the final goal? For example, if the task was "book a flight," was a flight actually booked correctly? This is the most important measure.
- **Process-based Evaluation:** Was the agent's *process* efficient and logical? Did it use the right tools? Did it follow a sensible plan? This helps debug why an agent might be failing.
- **Human Evaluation:** Having humans score the agent's performance on a scale (e.g., 1-5) based on criteria like helpfulness, accuracy, and coherence. This is crucial for user-facing applications.

What is an "agent trajectory"? An agent trajectory is the complete log of an agent's steps while performing a task. It includes all its thoughts, actions (tool calls), and observations. Analyzing these trajectories is a key part of debugging and understanding agent behavior.

How can you create reliable tests for a non-deterministic system? While you can't guarantee the exact wording of an agent's output, you can create tests that check for key elements. For example, you can write a test that verifies if the agent's final response *contains* specific information or if it successfully called a certain tool with the right parameters. This is often done using mock tools in a dedicated testing environment.

How is prompting an agent different from a simple ChatGPT prompt? Prompting an agent involves creating a detailed "system prompt" or constitution that acts as its operating instructions. This goes beyond a single user query; it defines the agent's role, its available tools, the patterns it should follow (like ReAct or Planning), its constraints, and its personality.

What are the key components of a good system prompt for an agent? A strong system prompt typically includes:

- **Role and Goal:** Clearly define who the agent is and what its primary purpose is.
- **Tool Definitions:** A list of available tools, their descriptions, and how to use them (e.g., in a specific function-calling format).
- **Constraints and Rules:** Explicit instructions on what the agent *should not* do (e.g., "Do not use tools without approval," "Do not provide financial advice").
- **Process Instructions:** Guidance on which patterns to use. For example, "First, create a plan. Then, execute the plan step-by-step."
- **Example Trajectories:** Providing a few examples of successful "thought-action-observation" loops can significantly improve the agent's reliability.

What is "prompt leakage"? Prompt leakage occurs when parts of the system prompt (like tool definitions or internal instructions) are inadvertently revealed in the agent's final response to the user. This can be confusing for the user and expose underlying implementation details. Techniques like using separate prompts for reasoning and for generating the final answer can help prevent this.

What are some future trends in agentic systems? The guide points towards a future with:

- **More Autonomous Agents:** Agents that require less human intervention and can learn and adapt on their own.
- **Highly Specialized Agents:** An ecosystem of agents that can be hired or subscribed to for specific tasks (e.g., a travel agent, a research agent).
- **Better Tools and Platforms:** The development of more sophisticated frameworks and platforms that make it easier to build, test, and deploy robust multi-agent systems.