

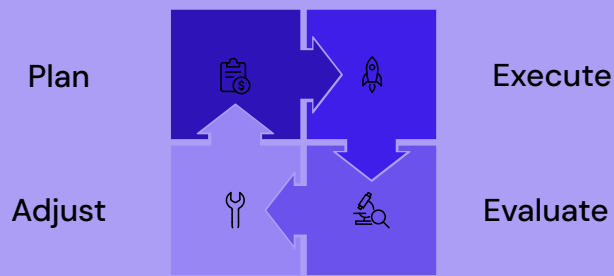
Agent Workflow Patterns Reference

A dense, practical reference for engineers building LLM-powered agent systems — covering architecture, patterns, and decision heuristics.

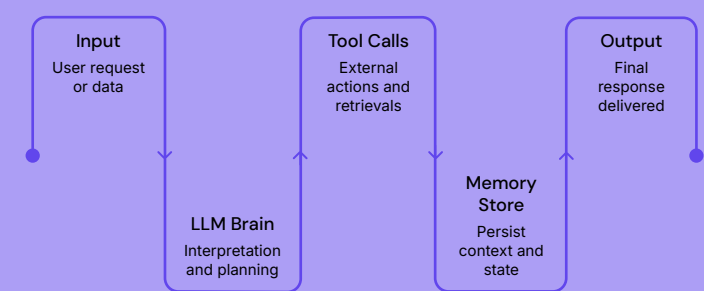
The Agent Pattern

Agent = LLM + Tools + Memory + Planning

Core Loop



Architecture Flow



Common Agent Patterns

	<h3>Router</h3> <p>Classify input → route to specialist</p> <p>Best for multi-domain systems where different inputs require different handling logic or downstream models.</p>
	<h3>Tool-User</h3> <p>Agent decides which tool to call</p> <p>Core pattern for agentic systems — the LLM selects from a registered toolset based on task context and intent.</p>
	<h3>Multi-Step Reasoner</h3> <p>Break complex task into sub-steps</p> <p>Enables chain-of-thought decomposition — the agent plans, executes, and evaluates each sub-task iteratively.</p>

✓ When to Use Agents

- **Multi-step tasks with branching logic**

When the workflow cannot be fully enumerated upfront and requires dynamic decision-making at each step.
- **Tasks requiring external tools**

When the LLM must call APIs, query databases, run code, or interact with external systems to complete the task.
- **When the path isn't known in advance**

When the sequence of operations depends on intermediate results — agents excel at adaptive, open-ended execution.

✗ When NOT to Use Agents

- Simple Q&A**

A single prompt is faster and dramatically cheaper. Agents add latency and token overhead with no benefit for straightforward retrieval or generation tasks.
- Deterministic workflows**

If the steps are fully known and fixed, use code — not an agent. Deterministic pipelines are more reliable, testable, and auditable.
- Latency-sensitive applications**

Agents are slow by design — each tool call and reasoning step adds round-trip time. Avoid for real-time or sub-second response requirements.

"Agents are powerful but expensive. Use them when the complexity justifies the cost."