Cheat Sheet: Agentic Frameworks and Design Patterns for Effective AI Systems

Estimated time: 20 minutes

---

1. Agentic AI Design Patterns

Agentic design patterns are reusable strategies for organizing multiple language model "agents" into structured workflows. These agents collaborate—each with specialized roles—to handle complex tasks that go beyond what a single prompt can manage. These patterns improve clarity, scalability, and control in LLM-powered applications. They are used to decompose complex problems into smaller, specialized tasks and add structure and memory to multi-step reasoning or decision flows.

1.1 Fundamental Components

- **Agent**: a specialized LLM prompt + logic unit
- **Orchestrator**: routes inputs, maintains state, aggregates outputs
- **Worker**: executes a single responsibility (for example, summarization, translation)
- **Router**: dispatches tasks based on intent or condition
- **Evaluator**: inspects and scores outputs for quality or correctness

These components embody the principle of separation of concerns. Decomposing your workflow into distinct agent roles makes it easier to debug, test, and reuse logic. It mirrors real-world software architecture by assigning specialized responsibilities to each part of the system.

1.2 Core Patterns

Pattern	Description	Use Cases
Orchestration	Central controller coordinates agents and tracks state	Document pipelines, decision trees, role-based workflows
Reflection	Evaluate and refine outputs with internal feedback loops	Quality improvement, self-correction, iterative generation

Pattern	Description	Use Cases
Sequential Coordination	Chain agents in a fixed order	Multi-step data pipelines (ingest → summarize → refine)
Intent-Based Routing	Dispatch inputs to agents based on user intent/class.	Multi-domain assistants (finance vs. weather vs. chat)
Parallel Execution	Fan-out tasks to multiple agents concurrently	Batch translations, multi-hypothesis generation
Prompt Chaining	Decompose a complex prompt into a sequence of simpler prompts	Complex content creation (outline → draft → edit)

> While "agentic frameworks" can include a variety of libraries and platforms (e.g., LangChain, AutoGen), this reading focuses exclusively on LangGraph as the agentic framework. All examples, patterns, and code snippets use LangGraph's APIs and conventions to illustrate core design patterns for building effective, multi-agent AI workflows.

---

2. LangGraph Workflows

2.1 Agents with Structured Output

First, we need chains of LLMs and prompts that configure agents to perform specific tasks. These agents will later be used in workers to perform certain tasks in the workflow. It's important that these agents have structured outputs so they return data in our desired format. This ensures that **LangGraph workflows** remain reliable and composable, especially when downstream agents depend on the outputs of earlier ones.

```
from pydantic import BaseModel, Field
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
# instantiate LLM
llm = ChatOpenAI(model="gpt-4o-mini")
# define output schema
class Output(BaseModel):
    name: str = Field(
        description="Name of the structured output"
    )
    field: str = Field(
        description="Field of the structured output"
    )
# create the LLM prompt template
```