

Orchestrating multiple agents

Orchestration refers to the flow of agents in your app. Which agents run, in what order, and how do they decide what happens next? There are two main ways to orchestrate agents:

1. Allowing the LLM to make decisions: this uses the intelligence of an LLM to plan, reason, and decide on what steps to take based on that.
2. Orchestrating via code: determining the flow of agents via your code.

You can mix and match these patterns. Each has their own tradeoffs, described below.

Orchestrating via LLM

An agent is an LLM equipped with instructions, tools and handoffs. This means that given an open-ended task, the LLM can autonomously plan how it will tackle the task, using tools to take actions and acquire data, and using handoffs to delegate tasks to sub-agents. For example, a research agent could be equipped with tools like:

- Web search to find information online
- File search and retrieval to search through proprietary data and connections
- Computer use to take actions on a computer
- Code execution to do data analysis
- Handoffs to specialized agents that are great at planning, report writing and more.

This pattern is great when the task is open-ended and you want to rely on the intelligence of an LLM. The most important tactics here are:

1. Invest in good prompts. Make it clear what tools are available, how to use them, and what parameters it must operate within.
2. Monitor your app and iterate on it. See where things go wrong, and iterate on your prompts.
3. Allow the agent to introspect and improve. For example, run it in a loop, and let it critique itself; or, provide error messages and let it improve.
4. Have specialized agents that excel in one task, rather than having a general purpose agent that is expected to be good at anything.
5. Invest in [evals](#). This lets you train your agents to improve and get better at tasks.

Orchestrating via code

While orchestrating via LLM is powerful, orchestrating via code makes tasks more deterministic and predictable, in terms of speed, cost and performance. Common patterns here are:

- Using [structured outputs](#) to generate well formed data that you can inspect with your code. For example, you might ask an agent to classify the task into a few categories, and then pick the next agent based on the category.
- Chaining multiple agents by transforming the output of one into the input of the next. You can decompose a task like writing a blog post into a series of steps - do research, write an outline, write the blog post, critique it, and then improve it.
- Running the agent that performs the task in a `while` loop with an agent that evaluates and provides feedback, until the evaluator says the output passes certain criteria.
- Running multiple agents in parallel, e.g. via Python primitives like `asyncio.gather`. This is useful for speed when you have multiple tasks that don't depend on each other.

We have a number of examples in [examples/agent_patterns](#).