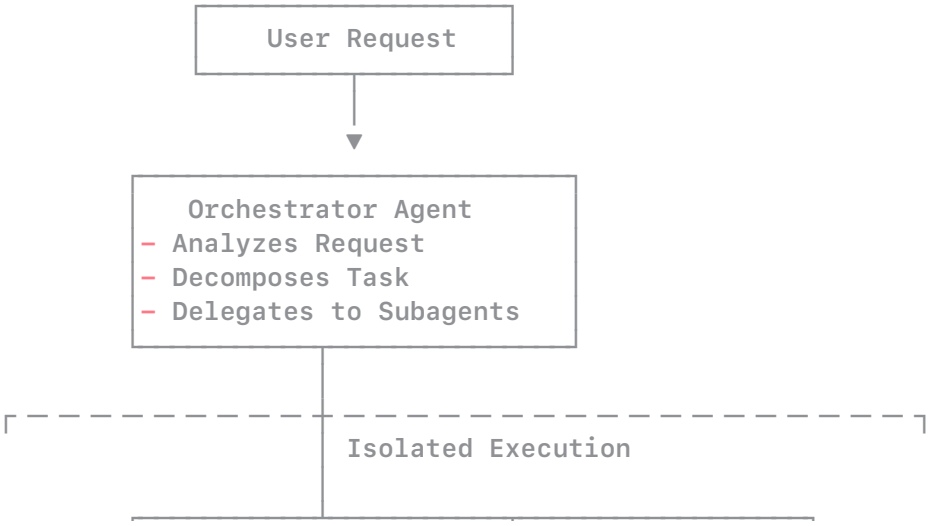
The Rise of Subagents

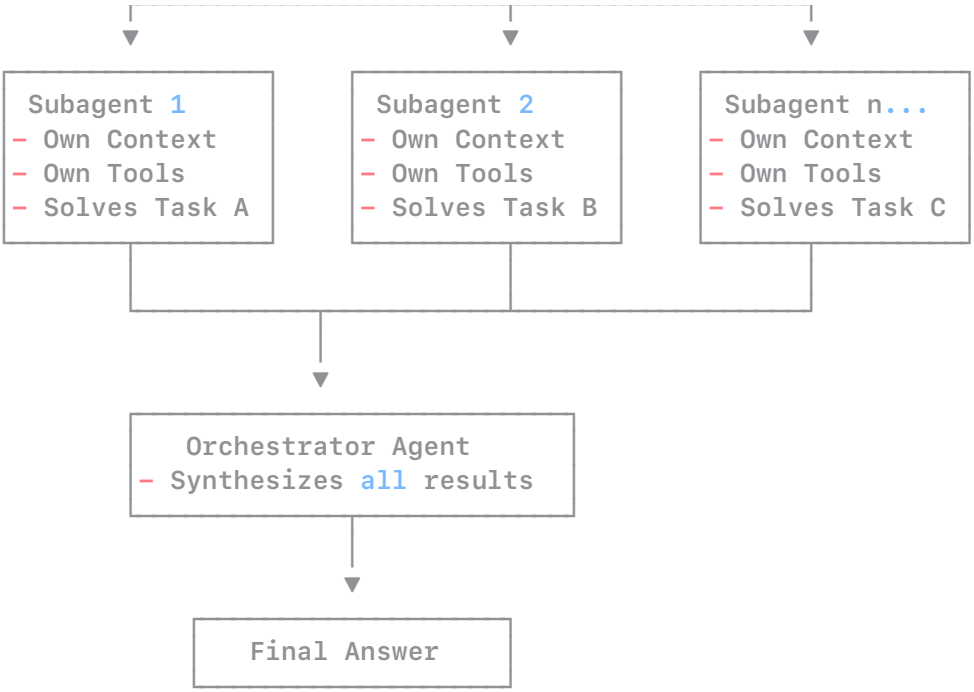
September 15, 2025 5 minute read

There is an increasing use of Subagents to reliably handle specific user goals. We can see this in tools like Claude Code and Poke.com. A subagent is a specialized Agent that is purpose-built for a single, well-defined task. Subagents can be explicit, where a user or model defines them for reuse or implicit and dynamically defined. This addresses a key limitation of monolithic AI agents context pollution. When a single, big and complex agent handles many tasks, its context window, number of tools, can become cluttered and less reliable.

What are Subagents?

Subagents are specialized AI agents. They are most of the time used in combination with an orchestrator, which delegates tasks to them. A subagent is just like a normal agent and has the same components. This includes a name for identification, a description of its capabilities, system instructions and a set of tools to interact with environments. It also has its own isolated context window.





Explicit, User-Defined Subagents

Explicit, User-defined Subagents are a permanent team of reusable specialists. Subagent can be defined in a static file or directly in code and the orchestrator uses them trough their name or when a prompt matches its **`description`**. Claude Code uses this method. The tools are manually listed in the definition, and the agent is stateless, meaning it starts fresh on every run.

Pros:

- You have full control over the agent's tools and behavior.
- The agent's actions are very predictable and secure.
- Reusable and easy to test in isolation

Cons:

- The system is rigid. You must update the definition file to add new tools.
- The orchestrator must manage all the state between steps.
- Hard to scale to hunderds of specialized task.

Here is a pseudo-code snippet of a definition file, inspired from Claude Code:

```
---
name: "Code-Reviewer"
description: "MUST BE USED for reviewing code against style guides and security practic
tools: ['file_read', 'search_code']
---
You are an expert security code reviewer. Your purpose is to analyze code and identify
```

Implicit, On-the-Fly Subagents

Implicit, On-the-Fly Subagents can be created temporary by an orchestrator to handle tasks as they come up. The orchestrator uses a tool (**`send_message_to_agent`**) to create and interact with the agent. The system dynamically assigns tools based on the user's natural language request from pre-defined pool which are needed to solve the task. Poke.com uses this method to created unlimited agents for specific user request. A key feature is that these agents can be stateful, e.g. keep context from previous runs when called with the same **`agent_name`**.

Here is a pseudo-code snippet for how the orchestrator calls the agent:

```
# The orchestrator calls this tool to create or reuse a subagent
send_message_to_agent(
    agent_name="q3_report_email_draft",
    description="An agent specializing in drafting internal company communications, spe
    message=f"""Draft an email to the marketing team about the Q3 report.

    TONE & STYLE:
    - Professional yet approachable and optimistic.
    - Be concise and data-driven. Avoid corporate jargon.
    - IMPORTANT: Analyze my last 5 sent emails to the 'marketing-team@' mailing list to

    RULES & CONSTRAINTS:
    - The subject line must be clear and engaging. Start with "Q3 Report Highlights:".
    - Use bullet points to present the key metrics for scannability.
    - Keep the total email length under 200 words.
    - End with a clear call to action: ask the team to review the full report and send
    """
)
```

- **Pros:**
  - The system is very flexible and requires no setup.
  - Tools are selected based on the unique user task.

    - Multi-step tasks are straightforward because the agent remembers its context.
- **Cons:**
    - The system is less predictable and you have less direct control.
    - It might fail if it misunderstands the task and assigns the wrong tools.
    - Harder to debug and reproduce failures since the agent's configuration is generated dynamically.

Conclusion

Both predefined and on-the-fly subagents will see increased usage. Dynamic agents might find more adoption in general B2C applications. Predefined agents are a great fit for more structured and repeatable enterprise workflows.

Context Engineering is everything. The key to success is giving the LLM the right information and tools at the right time. Using Subagents allows us to create a focused environment for the LLM with a clear, isolated context, specific system instructions, and a limited set of tools for the task. This improves performance and can reduce the cost of reaching the goal.

But even with a subagent architecture, reliability is still a challenge for agentic systems, breaking down a complex task into smaller subagent functions can make them easier to test and evaluate in isolation.

Models are improving very fast. Don't over-engineer a solution today that a simpler or better model can solve tomorrow.