How Sub-Agents Work in Claude Code: A Complete Guide

9 min read · 3 days ago

Kinjal Radadiya Follow

Claude Code has revolutionized AI-assisted development with its command-line tool for agentic coding. But one of its most powerful and often overlooked features is **sub-agents,** specialized AI assistants that can handle specific types of tasks with dedicated expertise. Think of them as assembling a team of specialized developers, where each expert focuses on what they do best.

In this comprehensive guide, we'll explore how sub-agents work in Claude Code, their architecture, practical use cases, and how you can leverage them to supercharge your development workflow.

What Are Sub-Agents?

Sub-agents are specialized AI assistants within Claude Code that operate independently to handle specific types of tasks. Unlike having a single general-purpose AI assistant trying to do everything, sub-agents enable you to create a team of specialists — each with their own expertise, tools, and context.

Key Characteristics of Sub-Agents

Independent Context Windows

Each sub-agent operates within its own isolated context space, preventing cross-contamination between different tasks and keeping the primary conversation focused on high-level objectives.

Custom System Prompts
Every sub-agent has a customized system prompt that guides its behavior, defining its personality, expertise areas, and specific workflows.

Selective Tool Access
You can control which tools each sub-agent can use. For example, a code reviewer might only have Read and Grep access, while a developer agent has Read, Write, Edit, and Bash capabilities.

Automatic Invocation
Claude Code can automatically delegate tasks to the appropriate sub-agent based on the context, or you can explicitly invoke specific agents by name.

. . .

How Sub-Agents Work: The Architecture

Sub-agents operate on a delegation model within Claude Code. Here's how the system works:

1. **Task Recognition**
   When you interact with Claude Code, the main agent analyzes your request to determine if a specialized sub-agent would be better suited for the task.

2. **Agent Selection**
   Based on the task requirements, Claude Code either:
   1. Automatically selects an appropriate sub-agent based on its description and capabilities
   2. Accepts your explicit request to use a specific sub-agent by name

3. **Isolated Execution**
   The sub-agent:
   1. Receives the delegated task
   2. Operates within its own context window (preventing main conversation pollution)
   3. Uses only its assigned tools
   4. Follows its custom system prompt

4. **Result Aggregation**
   After completing its task, the sub-agent returns only the relevant information to the main agent, which then integrates it into the primary conversation.

5. **No Nested Delegation**
   Sub-agents are not allowed to create other sub-agents, ensuring a single-level structure that prevents infinite nesting and keeps the execution hierarchy simple and manageable.

6. **Resumable Subagents**
   These are *persistent* sub-agents that can **pause and resume** their own conversation later.
   How it works:
   1. Each sub-agent gets a unique `agentId`.
   2. Its full conversation history is saved to a transcript file (`agent-{agentId}.jsonl`)
   3. Later, you can *resume* it by providing the same `agentId`.
   4. When resumed, it remembers all previous context — so it can continue where it left off.
   Use case:
   Perfect for **long-running or multi-step research** tasks that you might want to revisit — for example, a "Market Analysis Agent" that keeps gathering data over multiple sessions.

. . .

Built-in Sub-Agents

Claude Code includes specialized built-in sub-agents that activate automatically:

Plan Sub-Agent

When Claude runs in **plan mode** (also called **non-execution mode**), it's not directly running code or making changes.
Instead, it uses a **special helper called the Plan sub-agent.**
This **Plan sub-agent's job** is to:
1. Look through your project or codebase,
2. Gather relevant information,
3. And create a **clear plan** of what needs to be done before anything is executed.

Example scenario:

```
User: [In plan mode] Help me refactor the authentication module
Claude: Let me research your authentication implementation first...
[Internally invokes Plan sub-agent to explore auth-related files]
[Plan sub-agent searches codebase and returns findings]
Claude: Based on my research, here's my proposed plan...
```

. . .

Creating Custom Sub-Agents

One of the most powerful features of Claude Code is the ability to create your own custom sub-agents tailored to your specific workflow needs.

Configuration Structure

```
---
name: agent-name
description: When this agent should be invoked
tools: Read, Write, Edit, Bash, Glob, Grep
---

You are a [role description and expertise areas]...

[Agent-specific checklists, patterns, and guidelines]...

## Communication Protocol
Inter-agent communication specifications...

## Development Workflow
Structured implementation phases...
```

Creating a Sub-Agent

Step 1: Run the command

```
/agents
```

Step 2: Choose scope

Project-level: Located in `.claude/agents/` within your project
User-level: Available across all your projects

Step 3: Define the agent

Name: A unique identifier (e.g., `code-reviewer`, `security-auditor`)
Description: When Claude should invoke this agent
Tools: Which Claude Code tools the agent can access
System prompt: Detailed instructions defining the agent's role, behavior, and workflow

. . .

Tool Access Levels

Different agent types need different levels of access:

Read-only agents (reviewers, auditors)

```
tools: Read, Grep, Glob
```

Research agents (analysts, researchers)

```
tools: Read, Grep, Glob, WebFetch, WebSearch
```

Code writers (developers, engineers)

```
tools: Read, Write, Edit, Bash, Glob, Grep
```

Documentation agents (writers, documenters)

```
tools: Read, Write, Edit, Glob, Grep, WebFetch, WebSearch
```

. . .

Practical Use Cases

1. Code Review Agent
   Create a meticulous code reviewer that follows strict quality standards:

```
---
name: code-reviewer
description: Review code for quality, security, and best practices
tools: Read, Grep, Glob
---

You are a senior code reviewer with 15+ years of experience.

Your responsibilities:
- Analyze code for security vulnerabilities
- Check adherence to coding standards
- Identify potential performance issues
- Suggest improvements and optimizations
- Verify proper error handling

Review checklist:
1. Security: Check for SQL injection, XSS, CSRF vulnerabilities
2. Performance: Identify N+1 queries, inefficient algorithms
3. Style: Verify consistent naming, proper commenting
4. Testing: Ensure adequate test coverage
5. Documentation: Check for clear, accurate documentation
```

2. Test Automation Agent

Build an agent specialized in creating and running tests:

```
---
name: test-automator
description: Design and execute comprehensive test suites
tools: Read, Write, Edit, Bash, Glob, Grep
---

You are a QA automation engineer specializing in comprehensive testing.

Your workflow:
1. Analyze the feature requirements
2. Design test cases covering:
   - Happy path scenarios
   - Edge cases
   - Error conditions
   - Integration points
3. Write automated tests using appropriate frameworks
4. Execute tests and validate results
5. Report findings with clear pass/fail status

Focus areas:
- Unit tests for individual functions
- Integration tests for component interactions
- End-to-end tests for user workflows
- Performance tests for critical paths
```

3. Security Auditor Agent

Create a security-focused agent for vulnerability scanning:

```
---
name: security-auditor
description: Perform security analysis and vulnerability detection
tools: Read, Grep, Glob
---

You are a security specialist focused on identifying vulnerabilities.

Security audit checklist:
- Authentication and authorization flaws
- Input validation and sanitization
- Sensitive data exposure
- Security misconfigurations
- Vulnerable dependencies
```

```
- Insecure cryptographic practices
- API security issues

Provide detailed findings with:
- Severity level (Critical, High, Medium, Low)
- Location in codebase
- Description of the vulnerability
- Recommended remediation
- References to security standards (OWASP, CWE)
```

4. Database Architect Agent
Design database schemas and optimize queries:

```
---
name: database-architect
description: Design and optimize database structures
tools: Read, Write, Edit, Bash, Glob, Grep
---

You are a database architect with expertise in relational and NoSQL databases.

Your responsibilities:
- Design normalized database schemas
- Create efficient indexes
- Optimize query performance
- Plan migration strategies
- Ensure data integrity
- Implement proper relationships and constraints

Best practices:
- Follow normalization principles
- Use appropriate data types
- Create meaningful index strategies
- Document schema decisions
- Consider scalability requirements
```

. . .

Parallel Processing with Sub-Agents

One of the most powerful capabilities of sub-agents is parallel processing. You can launch multiple sub-agents simultaneously to tackle different parts of a problem:

```
Explore the codebase using 4 tasks in parallel.
Each agent should explore different directories.

● Task(Explore backend structure)
● Task(Explore frontend components)
● Task(Explore database migrations)
● Task(Explore test suites)
```

The parallelism level appears to be capped at 10 concurrent agents, but you can queue more tasks — Claude Code will batch them and execute subsequent groups once earlier batches complete.

Practical Parallel Use Cases

Codebase Analysis

```
Use 3 sub-agents to analyze these files:
1. Security analysis of auth.ts
2. Performance review of cache system
3. Type checking of utils.ts
```

Full-Stack Feature Development

```
Coordinate 7+ agents:
backend-architect → database-architect → frontend-developer →
test-automator → security-auditor → deployment-engineer →
observability-engineer
```

. . .

Benefits of Using Sub-Agents

1. Context Efficiency
Sub-agents help preserve the main context by only returning relevant information

rather than their entire conversation history. This is ideal for tasks that require sifting through large amounts of information where most won't be useful.

2. Task Specialization
Each sub-agent can be optimized for specific types of work, resulting in higher quality outputs. A security-focused agent will catch vulnerabilities that a general-purpose agent might miss.

3. Workflow Consistency
When sub-agents are shared across a team, everyone uses the same approach for common tasks, ensuring consistent quality and methodology.

4. Security Control
Tool access can be restricted based on the agent type. Read-only agents can analyze code without risk of modification, while privileged agents can make changes only when appropriate.

5. Scalability
As projects grow, you can add specialized sub-agents for new domains without overwhelming a single agent with too many responsibilities.

6. Reusability
Well-designed sub-agents can be used across multiple projects, saving setup time and leveraging refined expertise.

. . .

Best Practices for Sub-Agent Success

1. **Assign Clear, Non-Overlapping Roles**
   Each sub-agent should have a distinct responsibility. Overlapping duties create confusion and reduce efficiency.
   ❌ **Bad:** One agent handling both testing and code review
   ✅ **Good:** Separate test-automator and code-reviewer agents

2. **Minimize Tool Access**
   Give each agent only the tools it needs. This principle of least privilege reduces risk and keeps agents focused.

3. **Write Detailed System Prompts**
   The more specific your system prompt, the better the agent performs. Include:
   1. Role definition and expertise areas
   2. Step-by-step workflows
   3. Checklists and guidelines
   4. Expected output format
   5. Communication protocols

4. **Use Descriptive Names**
   Agent names should clearly indicate their purpose:
   python-backend-developer
   security-vulnerability-scanner
   typescript-code-reviewer

5. **Optimize Descriptions for Automatic Invocation**
   The description field determines when Claude automatically delegates to your agent. Make it clear and specific:
   ❌ **Bad:** "Helps with code"
   ✅ **Good:** "Reviews Python code for security vulnerabilities, PEP 8 compliance, and performance issues"

6. **Start Small and Iterate**
   Begin with one or two specialized agents, use them on real projects, and refine based on actual performance. Build your agent ecosystem gradually.

7. **Document Your Agents**
   Keep a team wiki or README documenting:
   1. What each agent does
   2. When to use it
   3. Examples of good prompts
   4. Known limitations

8. **Version Control Your Agents**
   Store agent configuration files in version control to track changes, share with team members, and maintain consistency across environments.

. . .

Challenges and Limitations
While sub-agents are powerful, they come with trade-offs:

Context Amnesia:
Sub-agents start with a clean slate each time they're invoked. They don't remember previous interactions, which can add latency as they gather necessary context. It doesn't automatically remember past work unless you specifically *resume it*.

Debugging Complexity
When issues occur, tracing them across multiple agents can be difficult. You're essentially debugging a **distributed reasoning system** instead of a single continuous conversation.

Setup Overhead
Creating and maintaining specialized agents requires upfront investment. For simple, one-off tasks, it may be overkill.

Performance Considerations
Launching sub-agents adds latency compared to handling everything in the main conversation. Use them when the benefits outweigh the performance cost.

Management Burden
As the number of agents grows, you'll need continuous **maintenance, documentation, and coordination** to keep them consistent and effective.

. . .

When to Use Sub-Agents vs. Main Agent

Use Sub-Agents When:
1. You need specialized expertise repeatedly across projects
2. Domain-specific work requires deep focus (security, performance, UX)
3. Role-specific tool access is important
4. You want team standardization
5. Tasks involve processing large amounts of information
6. You prefer automatic delegation based on task type

Use Main Agent When:

1. Tasks are simple and one-off
2. You want explicit control over every step
3. Work requires continuous context from the conversation
4. Speed is more important than specialization
5. You're exploring or prototyping

. . .

Quick Start: Your First Sub-Agent

Ready to create your first sub-agent? Here's a simple example to get started:

```
# Run the agents command in Claude Code
/agents

# Choose "Create new agent"
# Select scope: project-level or user-level

# Configuration:
name: code-formatter
description: Format code according to project style guidelines
tools: Read, Write, Edit

# System prompt:
You are a code formatting specialist. Your job is to:
1. Read the provided code files
2. Apply consistent formatting according to the project's style guide
3. Fix indentation, spacing, and line breaks
4. Organize imports and remove unused ones
5. Return neatly formatted code

Always preserve functionality while improving readability.
```

Once created, Claude will automatically invoke your formatter agent when you ask to format code, or you can explicitly call it:

```
Have the code-formatter agent clean up my Python files in the src/ directory
```

Welcome to the world of multi-agent development! 🚀

Follow

Written by Kinjal Radadiya

2 followers · 2 following

No responses yet

Bgerby

What are your thoughts?

More from Kinjal Radadiya

Kinjal Radadiya

Git Commands Guide with Examples (Beginner to Pro)

Git is one of the most important tools for developers. It helps you track changes, collaborate, and manage versions of your project code...

Sep 21 •••

See all from Kinjal Radadiya

Recommended from Medium

Joe Njenga

6 Google AI Coding Tools Making Developers Faster(And Cutting Budgets by 50%)

Gemini 3.0 is about to shake the AI coding space, and if you are not using one of these Google AI coding tools, you are behind or losing...

4d ago 133 2

In Level Up Coding by Fareed Khan

Building a Training Architecture for Self-Improving AI Agents

RL Algorithms, Policy Modeling, Distributed Training and more.

4d ago 719 14

Daniel Avila

Running Claude Code Agents in Docker Containers for Complete Isolation

Running AI-generated code directly on your machine can be risky.

4d ago 40

In Artificial Intelligence in Plain English by Somendradev

Modern Developer’s Toolbox: The 2025 Edition

The developer world never stops evolving. One year you're writing monolithic codebases, the next you're deploying microservices with AI...

Nov 1 47 2

Jannis

I Discovered Glow—and My Terminal Has Never Looked So Good

How a simple Markdown reader turned my terminal into a publishing studio

4d ago 202

In Coding Beauty by Tari Ibaba

This insane new coding model is 13 times faster than Claude Sonnet 4.5 😮

Just wow.

6d ago 207 3

See more recommendations