Miyabi

Claude Code & Codex
Multi-Agent Coordination Architecture

CCG (Claude Code Guide) + CG (Codex Guide) Subagent System

Miyabi Development Team

November 2025

Abstract

This document provides a comprehensive technical specification for coordinating Claude Code (CCG) and OpenAI Codex (CG) agents within the Miyabi autonomous development platform. We detail the subagent architectures of both systems, define coordination patterns, and establish best practices for multi-agent orchestration. The goal is to enable hybrid AI development workflows that leverage the strengths of both Claude and OpenAI models.

Contents

1 Introduction

1.1 Miyabi Platform Overview

Miyabi is a complete autonomous AI development operations platform implementing:

- **GitHub-as-OS**: Issues = Tasks, PRs = Deliverables, Actions = CI/CD
- **21 AI Agents**: 7 coding agents + 14 business agents
- **59 Rust Crates**: Modular architecture with 50%+ faster compilation
- **28+ MCP Servers**: Extensible tool ecosystem
- **A2A Protocol**: Agent-to-Agent communication standard

1.2 CCG and CG Definition

Abbreviation	Full Name	Description
CCG	Claude Code Guide	Claude Code subagent system
CG	Codex Guide	OpenAI Codex subagent system

Table 1: Agent System Abbreviations

1.3 Document Objectives

1. Define CCG and CG subagent capabilities and limitations
2. Establish coordination patterns for hybrid workflows
3. Provide implementation guidelines for Miyabi integration
4. Document best practices for multi-agent orchestration

2 Claude Code (CCG) Architecture

2.1 Core Mechanism

Claude Code's Task tool operates as an ephemeral worker spawning system:

- Each Task receives an **isolated 200k context window**
- Subagents maintain **independent state** from parent
- Maximum **10 concurrent tasks** in parallel
- **No nested subagents**: Subagents cannot spawn other subagents

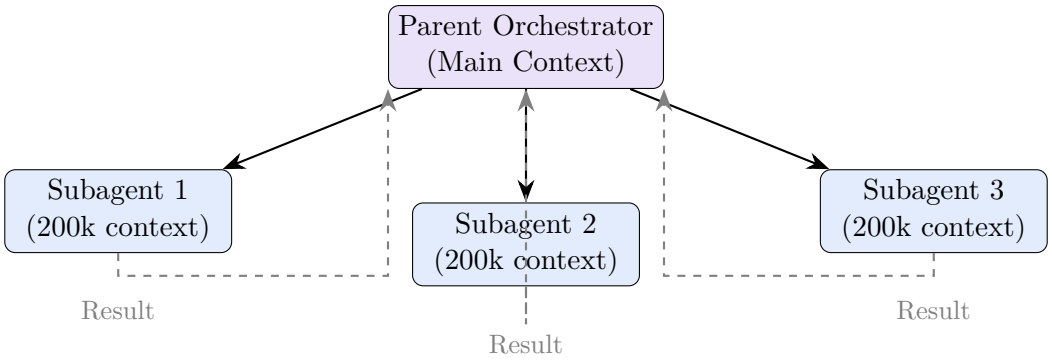

Figure 1: CCG Subagent Execution Model

Type	Model	Tools	Use Case
General-Purpose	Sonnet	All	Full-stack implementation
Plan	Sonnet	Read-only	Architecture analysis
Explore	Haiku	Read-only	Fast codebase discovery
claude-code-guide	—	Docs	Documentation lookup

Table 2: CCG Built-in Subagent Types

2.2 Built-in Subagent Types

2.3 Custom Subagent Configuration

Custom subagents are defined via YAML frontmatter in Markdown files:

Listing 1: Custom CCG Subagent Definition

```
---
name: security-reviewer
description: "Security audit specialist"
model: sonnet
tools:
  - Read
  - Glob
  - Grep
  - Bash
permissionMode: default
---

You are a security expert analyzing code for vulnerabilities...
```

Storage Locations:

- Project-level: `.claude/agents/` (version-controlled)
- User-level: `~/.claude/agents/` (global)

2.4 Claude Agent SDK Architecture

SDK Components

- `ClaudeSDKClient`: Stateful multi-turn sessions
- `query()`: Stateless one-off queries
- `ClaudeAgentOptions`: Configuration object
- Hooks: PreToolUse, PostToolUse, SessionStart, etc.

2.4.1 Python SDK Usage

Listing 2: Claude Agent SDK Example

```
from claude_agent_sdk import ClaudeSDKClient, ClaudeAgentOptions

options = ClaudeAgentOptions(
    allowed_tools=["Read", "Write", "Bash"],
    model="sonnet",
    permission_mode="acceptEdits",
    max_turns=20
)

async with ClaudeSDKClient() as client:
    await client.connect(initial_prompt)
    async for message in client.query(task):
        process_message(message)
```

2.5 Communication Patterns

Key Principles:

1. Subagents receive **only task-relevant context**
2. Results return to parent **without context pollution**
3. **No peer-to-peer communication** between subagents
4. Orchestrator maintains **separation of concerns**

3 OpenAI Codex (CG) Architecture

3.1 Core Mechanism

Codex CLI integrates via Model Context Protocol (MCP) server architecture:

- Exposes `codex()` and `codex-reply()` tools
- Persistent session across agent turns
- Configurable sandbox and approval policies
- Native Agents SDK integration

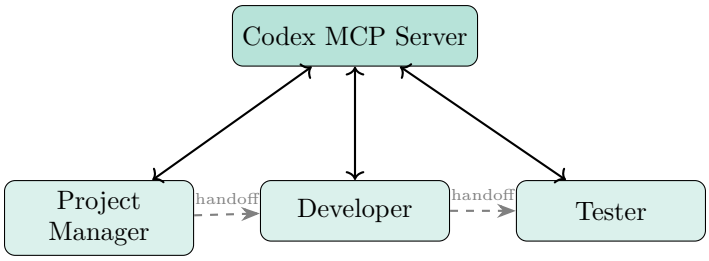

Figure 2: CG MCP-Based Architecture

3.2 Agent Configuration

Codex agents are configured via TOML files at `~/.codex/agents.toml`:

Listing 3: Codex Agent Configuration (agents.toml)

```
[agents.code-reviewer]
name = "Code Reviewer"
system_prompt = """
You are an expert code reviewer focusing on:
- Security vulnerabilities
- Performance issues
- Code quality and maintainability
"""
model = "gpt-5"

[agents.security-auditor]
name = "Security Auditor"
system_prompt = "You are a security specialist..."
```

3.3 Multi-Agent Orchestration

Codex Orchestration Features

- **Hierarchical Coordination**: Project Manager gates handoffs
- **Parallel Execution**: Independent agents run concurrently
- **File-Based Checkpoints**: Deliverables gate progression
- **Bidirectional Handoffs**: Return to PM for validation

3.3.1 Agents SDK Integration

Listing 4: Codex + Agents SDK

```
from agents import Agent, Runner
from agents.mcp import MCPServerStdio

async with MCPServerStdio(
    name="Codex CLI",
    params={"command": "npx", "args": ["-y", "codex", "mcp"]},
) as codex_mcp:

    developer = Agent(
```

```
        name="Developer",
        instructions="Build features based on specs...",
        mcp_servers=[codex_mcp],
        model="gpt-5"
    )

    reviewer = Agent(
        name="Reviewer",
        instructions="Review code quality...",
        mcp_servers=[codex_mcp],
        handoffs=[developer]
    )

    result = await Runner.run(reviewer, task)
```

3.4 Sandbox and Permissions

Setting	Description
`approval-policy: never`	Autonomous execution
`approval-policy: always`	Require user approval
`sandbox: workspace-write`	Write within workspace only
`sandbox: none`	Full system access

Table 3: Codex Sandbox Configuration

4 CCG + CG Coordination Architecture

4.1 Hybrid Orchestration Model

The Miyabi platform coordinates CCG and CG through a unified orchestration layer:

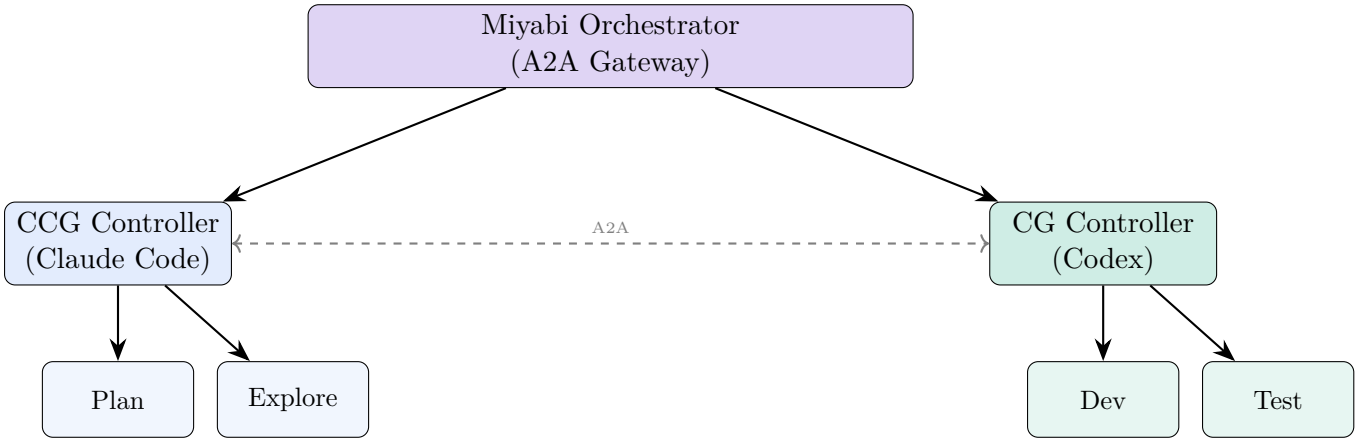

Figure 3: Miyabi Hybrid CCG/CG Architecture

Role	System	Responsibility	Strengths
Coordinator	CCG	Task decomposition, DAG	Long context, reasoning
Planner	CCG	Architecture design	Planning mode
Explorer	CCG	Codebase analysis	Fast Haiku search
Developer	CG	Code generation	GPT-5 creativity
Tester	CG	Test creation	Structured output
Reviewer	CCG	Quality assurance	Opus deep analysis

Table 4: Hybrid Agent Role Assignment

4.2 Role Definitions

4.3 Coordination Patterns

4.3.1 Pattern 1: Sequential Handoff

Listing 5: Sequential CCG to CG Handoff

```
# Phase 1: CCG Planning
plan = await ccg_planner.query("Analyze and design feature X")

# Phase 2: CG Implementation
impl = await cg_developer.query(f"Implement: {plan}")

# Phase 3: CCG Review
review = await ccg_reviewer.query(f"Review: {impl}")
```

4.3.2 Pattern 2: Parallel Execution

Listing 6: Parallel CCG and CG Execution

```
# Run CCG and CG agents in parallel
results = await asyncio.gather(
    ccg_explore.query("Search for auth patterns"),
    cg_developer.query("Generate auth boilerplate"),
    ccg_security.query("Review security requirements")
)

# Synthesize results
synthesis = await ccg_coordinator.query(
    f"Synthesize findings: {results}"
)
```

4.3.3 Pattern 3: Competitive Validation

Listing 7: Cross-System Validation

```
# Generate with one system
ccg_impl = await ccg_developer.query("Implement feature")

# Validate with the other
cg_review = await cg_reviewer.query(f"Review: {ccg_impl}")
```

```
# Or vice versa
cg_impl = await cg_developer.query("Implement feature")
ccg_review = await ccg_reviewer.query(f"Review: {cg_impl}")
```

4.4 A2A Protocol Integration

Miyabi's A2A protocol enables standardized communication:

Listing 8: A2A Tool Naming Convention

```
# CCG agents
a2a.claude_code_planning_agent.analyze_architecture
a2a.claude_code_review_agent.review_code

# CG agents
a2a.codex_development_agent.generate_code
a2a.codex_testing_agent.create_tests

# Cross-system routing via A2A Gateway
a2a_gateway.route(
    source="ccg.planner",
    target="cg.developer",
    payload=plan_document
)
```

5 Implementation Guidelines

5.1 Miyabi Integration Steps

1. **Configure CCG Agents**
   - Create `.claude/agents/` directory
   - Define agent specs with YAML frontmatter
   - Set up Claude Agent SDK hooks
2. **Configure CG Agents**
   - Create `~/.codex/agents.toml`
   - Define agent system prompts
   - Configure MCP server integration
3. **Implement A2A Gateway**
   - Register both CCG and CG agents
   - Define routing rules
   - Implement message transformation
4. **Create Orchestration Workflows**
   - Define task decomposition rules
   - Implement parallel execution handlers
   - Set up result aggregation

5.2 Configuration Files

5.2.1 CCG Configuration (.claude/agents/miyabi-coordinator.md)

```
---
name: miyabi-coordinator
description: "Miyabi task coordinator with CCG/CG routing"
model: opus
tools:
  - Read
  - Glob
  - Grep
  - Task
  - Bash
permissionMode: default
---

You are the Miyabi Coordinator responsible for:
1. Decomposing tasks into subtasks
2. Routing tasks to CCG or CG agents
3. Aggregating results
4. Managing cross-system handoffs
```

5.2.2 CG Configuration (~/.codex/agents.toml)

```
[agents.miyabi-developer]
name = "Miyabi Developer"
system_prompt = """
You are a Miyabi development agent specializing in:
- Feature implementation
- Code generation
- Test creation

Always output to designated folders and create checkpoints.
"""
model = "gpt-5"
```

5.3 MCP Server Setup

Listing 9: .mcp.json Configuration

```
{
  "servers": {
    "miyabi-ccg": {
      "command": "claude-code",
      "args": ["mcp"],
      "protocol": "stdio"
    },
    "miyabi-cg": {
      "command": "npx",
      "args": ["-y", "codex", "mcp"],
      "protocol": "stdio"
    }
  }
```

```
}
```

6 Best Practices

6.1 Orchestration Guidelines

Critical Rules

1. **Pure Orchestrator**: The coordinator must NOT do actual work
2. **No Nested Subagents**: CCG subagents cannot spawn subagents
3. **Respect Parallelism Limits**: Max 10 CCG tasks, variable for CG
4. **Context Isolation**: Do not share full history between systems

6.2 Task Routing Heuristics

Task Type	Recommended System	Reason
Long-form analysis	CCG (Opus)	200k context window
Fast exploration	CCG (Haiku)	Speed-optimized
Code generation	CG (GPT-5)	Creative output
Structured output	CG	JSON mode support
Security review	CCG (Opus)	Deep reasoning
Test generation	CG	Template-based
Documentation	CCG	Long-form writing

Table 5: Task Routing Recommendations

6.3 Error Handling

Listing 10: Robust Error Handling

```
async def execute_hybrid_task(task):
    try:
        # Try CCG first
        result = await ccg_agent.query(task, timeout=300)
    except CCGTimeoutError:
        # Fallback to CG
        result = await cg_agent.query(task)
    except Exception as e:
        # Log and escalate
        await a2a_gateway.escalate(task, error=e)
        raise

    return result
```

6.4 Monitoring and Observability

- Enable logging on both CCG and CG agents

- Use hooks to track tool usage
- Implement cost tracking per agent system
- Monitor context window utilization
- Track cross-system latency

7 Limitations and Constraints

7.1 CCG Limitations

- No subagent nesting (prevents infinite complexity)
- Maximum 10 concurrent tasks
- No direct peer communication between subagents
- Batch queue waits for completion before pulling next

7.2 CG Limitations

- Requires MCP server for agent integration
- Session persistence depends on MCP connection
- Sandbox restrictions may limit file access
- Model availability varies by plan

7.3 Cross-System Constraints

- No native inter-system communication
- Context must be explicitly passed between systems
- Different authentication mechanisms
- Varying response formats require normalization

8 Conclusion

The CCG + CG coordination architecture enables Miyabi to leverage the strengths of both Claude Code and OpenAI Codex:

- **CCG** excels at planning, analysis, and deep reasoning with large context windows
- **CG** excels at code generation and structured workflows via MCP
- **Miyabi's A2A Gateway** provides the bridge for seamless coordination

By following the patterns and guidelines in this document, development teams can build sophisticated multi-agent workflows that combine the best of both AI systems.

References

1. Claude Code Documentation: https://docs.anthropic.com/claude-code
2. Claude Agent SDK: https://github.com/anthropics/claude-agent-sdk-python
3. OpenAI Codex: https://github.com/openai/codex
4. Codex Agents SDK Guide: https://developers.openai.com/codex/guides/agents-sdk
5. Miyabi A2A Protocol: Internal Documentation