OH-MY-CLAUDECODE: MULTI-AGENT ORCHESTRATION FOR CLAUDE CODE

ZERO LEARNING CURVE. MAXIMUM POWER.

[Speaker Name]

Version 3.6.3

AGENDA

Time	Topic
0:00	What is OMC?
0:10	The 5 Key Execution Modes
0:30	The Agent System
0:40	Live Demo Scenarios
0:48	Developer Experience
0:54	Getting Started
0:58	Q&A

THE PROBLEM

Developers today face:

THE PROBLEM

Developers today face:

- Manual coordination of complex multi-step tasks

THE PROBLEM

Developers today face:

- Manual coordination of complex multi-step tasks
- Constant context-switching between different concerns

THE PROBLEM

Developers today face:

- Manual coordination of complex multi-step tasks
- Constant context-switching between different concerns
- Single-threaded AI interactions that don't scale

THE PROBLEM

Developers today face:

- Manual coordination of complex multi-step tasks
- Constant context-switching between different concerns
- Single-threaded AI interactions that don't scale
- No persistence - AI gives up when tasks get hard

THE PROBLEM

Developers today face:

- Manual coordination of complex multi-step tasks
- Constant context-switching between different concerns
- Single-threaded AI interactions that don't scale
- No persistence - AI gives up when tasks get hard
- Token waste - using expensive models for simple tasks

SECTION 1

WHAT IS OMC?

WHAT IS OH-MY-CLAUDECODE?

A multi-agent orchestration system for Claude Code

```
      +------------------+
      |     You (User)   |
      +--------+---------+
               |
               v
      +------------------+
      |  Claude (Conductor)  |
      +--------+---------+
               |
+--------------+--------------+
|              |              |
v              v              v
      +---------+    +---------+    +---------+
      | Skill 1 |    | Skill 2 |    | Skill N |
      +---------+    +---------+    +---------+
|              |              |
```

WHAT IS OH-MY-CLAUDECODE?

A multi-agent orchestration system for Claude Code

```
      +------------------+
      |     You (User)   |
      +--------+---------+
               |
               v
      +------------------+
      |  Claude (Conductor)  |
      +--------+---------+
               |
+--------------+--------------+
|              |              |
v              v              v
      +---------+    +---------+    +---------+
      | Skill 1 |    | Skill 2 |    | Skill N |
      +---------+    +---------+    +---------+
```

- 32 specialized agents

WHAT IS OH-MY-CLAUDECODE?

A multi-agent orchestration system for Claude Code

```
      +------------------+
      |     You (User)   |
      +--------+---------+
               |
               v
      +------------------+
      |  Claude (Conductor)  |
      +--------+---------+
               |
+--------------+--------------+
|              |              |
v              v              v
      +----------+   +----------+   +----------+
      | Skill 1  |   | Skill 2  |   | Skill N  |
      +----------+   +----------+   +----------+
|              |              |
```

- 32 specialized agents
- 35+ skills

WHAT IS OH-MY-CLAUDECODE?

A multi-agent orchestration system for Claude Code

```
      +-------------------+
      |     You (User)   |
      +--------+----------+
               |
               v
      +-------------------+
      |  Claude (Conductor)  |
      +--------+----------+
               |
+--------------+--------------+
|              |              |
v              v              v
      +---------+    +---------+    +---------+
      | Skill 1 |    | Skill 2 |    | Skill N |
      +---------+    +---------+    +---------+
|              |              |
```

- 32 specialized agents
- 35+ skills
- Zero configuration required

THE PHILOSOPHY

"You are a CONDUCTOR, not a performer."

Traditional AI Workflow:

```
User -> Claude -> [Does everything itself]
```

OMC Workflow:

```
User -> Claude (Conductor) -> [Delegates to specialists]
                     |
       +-------------+-------------+
       |             |             |
     architect          executor          designer
    (analysis)    (implementation)    (UI/UX)
```

Claude becomes an intelligent orchestrator that delegates to the right specialist for each task.

BEFORE VS AFTER OMC

Aspect	Before OMC	After OMC
Task execution	Single-threaded	Parallel agents
Complex tasks	Manual breakdown	Automatic decomposition
Model selection	Always same model	Smart routing (Haiku/Sonnet/Opus)
Persistence	Gives up easily	Continues until verified

Aspect	Before OMC	After OMC
Cost	Expensive	30-50% savings
Learning curve	Command memorization	Natural language

Example - "Fix all TypeScript errors":

Before: You manually find and fix each error sequentially

After: 5 parallel agents claim and fix errors simultaneously

KEY STATISTICS

Metric	Value
Specialized Agents	32
Skills	35+
Execution Modes	8
Lifecycle Hooks	19
Model Tiers	3 (Haiku, Sonnet, Opus)
License	MIT

Token Cost Comparison:

Model	Input	Output
Haiku	$0.25/1M	$1.25/1M
Sonnet	$3/1M	$15/1M
Opus	$15/1M	$75/1M

ARCHITECTURE OVERVIEW

```
+------------------------------------------------------------
|
               USER INPUT                                 |
|
        "autopilot: build a REST API"                     |
+------------------------------------+-----------------------
                         |
                         v
+------------------------------------------------------------
|
          CLAUDE CODE (CONDUCTOR)                         |
|  +----------------+  +----------------+  +-----------------
|  | Keyword        |  | Skill          |  | Agent
   |    |
|  | Detection      |->| Resolution     |->| Delegation
```

SECTION 2

THE 5 KEY EXECUTION MODES

MODE 1: AUTOPILOT - WHAT IS IT?

Full autonomous execution from idea to working code

```
"autopilot: build a REST API for a bookstore"
```

5 Phases:

1. **Expansion** - Turn vague idea into detailed spec
2. **Planning** - Create implementation plan with validation
3. **Execution** - Build with parallel agents (Ralph + Ultrawork)
4. **QA** - Test until everything passes (up to 5 cycles)
5. **Validation** - Multi-reviewer approval (Architect + Security + Code Review)

MODE 1: AUTOPILOT - HOW IT WORKS

```
Phase 0: EXPANSION
    |
    +-> Analyst (Opus) extracts requirements
    +-> Architect (Opus) creates technical spec
    |
    v
Phase 1: PLANNING
    |
    +-> Architect creates plan (direct mode)
    +-> Critic validates plan
    |
    v
Phase 2: EXECUTION
    |
    +-> Ralph + Ultrawork activated
```

MODE 1: AUTOPILOT - WHEN TO USE IT

Best For:

- New projects from scratch
- Complete feature implementations
- End-to-end workflows

Trigger Keywords:

```
autopilot, auto pilot, autonomous
build me, create me, make me
full auto, handle it all
I want a/an...
```

Example Commands:

```
autopilot: build a REST API with CRUD for inventory

/oh-my-claudecode:autopilot Add OAuth2 authentication

autopilot: create a CLI tool that tracks daily habits
```

MODE 2: ULTRAPILOT - WHAT IS IT?

Parallel autopilot with up to 5 concurrent workers

3-5x faster than standard autopilot for suitable tasks.

```
"ultrapilot: build a full-stack todo app"
```

Key Innovation: File ownership partitioning

MODE 2: ULTRAPILOT - HOW IT WORKS

```
User Input: "Build a full-stack todo app"
        |
        v
          [ULTRAPILOT COORDINATOR]
        |
        Task Decomposition + File Partitioning
        |
        +-----------+-----------+-----------+-----------+
        |           |           |           |           |
        v           v           v           v           v
    [Worker-1]  [Worker-2]  [Worker-3]  [Worker-4]  [Worker-5
     backend     frontend    database    api-docs      tests
    (src/api/)  (src/ui/)   (src/db/)   (docs/)    (tests/)
        |           |           |           |           |
        +-----------+-----------+-----------+-----------+
```

MODE 2: ULTRAPILOT - WHEN TO USE IT

Best For:

- Multi-component systems (frontend + backend + database)
- Large refactorings with clear module boundaries
- Multi-service architectures
- Parallel test generation

Speed Comparison:

Task	Autopilot	Ultrapilot
Full-stack app	~75 min	~15 min
Multi-service refactor	~32 min	~8 min
Test coverage	~50 min	~10 min

Trigger:

```
ultrapilot, parallel build, swarm build
```

MODE 3: SWARM - WHAT IS IT?

N coordinated agents with atomic task claiming

```
/swarm 5:executor "fix all TypeScript errors"
```

Architecture:

- SQLite-based task pool
- Atomic claiming via transactions
- 5-minute lease timeout with auto-release
- Heartbeat monitoring for fault tolerance

MODE 3: SWARM - HOW IT WORKS

```
/swarm 5:executor "fix all TypeScript errors"
  |
  v
      [SWARM ORCHESTRATOR]
  |
   +--+--+--+--+--+
   |  |  |  |  |
   v  v  v  v  v
  E1 E2 E3 E4 E5    <-- 5 Executor agents
   |  |  |  |  |
   +--+--+--+--+
          |
          v
    [SQLITE DATABASE]
    +----------------------+
```

Claim Protocol:

1. Agent calls `claimTask()`
2. SQLite transaction atomically updates status
3. Agent works on task
4. Agent calls `completeTask()` or `failTask()`

MODE 3: SWARM - WHEN TO USE IT

Best For:

- Many independent parallel tasks
- File-by-file operations
- Batch processing

Use Cases:

```
# Fix all TypeScript errors
/swarm 5:executor "fix all TypeScript errors"

# Style all UI components
/swarm 3:designer "implement Material-UI styling for all comp
```

```
# Security audit all endpoints
/swarm 4:security-reviewer "review all API endpoints"

# Add documentation
/swarm 2:writer "add JSDoc comments to all exported functions
```

MODE 4: PIPELINE - WHAT IS IT?

Sequential agent chaining with data passing

Like Unix pipes, but for AI agents.

```
/pipeline explore -> architect -> executor "add authenticatio
```

Output of one agent becomes input to the next:

```
[explore findings] -> [architect analysis] -> [executor imple
```

MODE 4: PIPELINE - BUILT-IN PRESETS

Preset	Stages	Use For
`review`	explore -> architect -> critic -> executor	Major features, refactorings
`implement`	planner -> executor -> tdd-guide	New features with tests

Preset	Stages	Use For
`debug`	explore -> architect -> build-fixer	Bugs, build errors
`research`	parallel(researcher, explore) -> architect -> writer	Technology decisions
`refactor`	explore -> architect-medium -> executor-high -> qa-tester	Safe refactoring
`security`	explore -> security-reviewer -> executor -	Security fixes

Preset	Stages	Use For
	> security-reviewer-low	

Usage:

```
/pipeline review "add rate limiting to API"
/pipeline debug "login fails with OAuth"
/pipeline security "audit user authentication"
```

MODE 4: PIPELINE - WHEN TO USE IT

Best For:

- Multi-stage processing workflows
- Code review processes
- Research-to-implementation flows

Custom Pipeline Syntax:

```
# Basic sequential
/pipeline agent1 -> agent2 -> agent3 "task"

# With model specification
/pipeline explore:haiku -> architect:opus -> executor:sonnet
```

```
# With parallel stages
/pipeline [explore, researcher] -> architect -> executor "tas
```

Data Flow:

```
{
  "pipeline_context": {
    "original_task": "user's request",
    "previous_stages": [
      {"agent": "explore", "findings": "..."}
    ],
    "current_stage": "architect"
  }
}
```

MODE 5: ECOMODE - WHAT IS IT?

Token-efficient parallel execution

30-50% cheaper than standard execution.

```
eco: implement new feature
```

Strategy:

- Prefer Haiku (cheapest) for all tasks
- Only upgrade to Sonnet when needed
- Avoid Opus unless absolutely essential

MODE 5: ECOMODE - HOW IT WORKS

Routing Rules:

Task Type	Standard Mode	Ecomode
Simple lookup	architect-low	architect-low
Standard impl	executor	executor-low (first attempt)

Task Type	Standard Mode	Ecomode
Complex analysis	architect	architect-medium
Planning	planner (Opus)	Avoid if possible

Agent Routing Table:

Domain	Preferred (Haiku)	Fallback (Sonnet)	Avoid (Opus)
Analysis	architect-	architect-	~~architect~~

Domain	Preferred (Haiku)	Fallback (Sonnet)	Avoid (Opus)
	low	medium	
Execution	executor-low	executor	~~executor-high~~
Search	explore	explore-medium	~~explore-high~~
Frontend	designer-low	designer	~~designer-high~~

MODE 5: ECOMODE - WHEN TO USE IT

Best For:

- Budget-conscious projects
- Iterative development (many small changes)
- Exploratory work
- Personal projects

Cost Savings Example:

Task	Standard Cost	Ecomode Cost	Savings
100 simple fixes	~$3.00	~$0.50	83%
Feature impl	~$1.50	~$0.75	50%
Full build	~$10.00	~$5.00	50%

Trigger:

```
eco, ecomode, efficient, save-tokens, budget
```

SECTION 3

THE AGENT SYSTEM

32 SPECIALIZED AGENTS

Domain	Agents
Analysis	architect, architect-medium, architect-low
Execution	executor, executor-high, executor-low
Search	explore, explore-medium, explore-high
Research	researcher, researcher-low

Domain	Agents
Frontend	designer, designer-high, designer-low
Documentation	writer
Visual	vision
Planning	planner, analyst
Critique	critic
Testing	qa-tester, qa-tester-high
Security	security-reviewer, security-reviewer-low

Domain	Agents
Build	build-fixer, build-fixer-low
TDD	tdd-guide, tdd-guide-low
Code Review	code-reviewer, code-reviewer-low
Data Science	scientist, scientist-high, scientist-low

3-TIER MODEL ROUTING

```
+-------------------+-------------------+-------------------+
|   LOW (Haiku)     |  MEDIUM (Sonnet)  |   HIGH (Opus)     |
|-------------------|-------------------|-------------------|
| $0.25/$1.25/1M    | $3/$15/1M         | $15/$75/1M        |
|-------------------|-------------------|-------------------|
| Simple lookups    | Standard work     | Complex reasoning|
| Quick searches    | Feature impl      | Architecture      |
| Basic fixes       | Moderate debug    | Deep debugging    |
| Documentation     | UI components     | Security audits   |
+-------------------+-------------------+-------------------+
         ^
      ^                  ^
         |
      |                  |
   Use by default    Upgrade when      Only when truly
```

Cost Example:

- 1000 simple questions: Haiku = $0.25 vs Opus =$15 (60x cheaper!)

SMART DELEGATION

OMC automatically picks the right agent:

Task	Agent Selected	Model
"What does this function return?"	architect-low	Haiku
"Find where UserService is defined"	explore	Haiku
"Add validation to login form"	executor-low	Haiku

Task	Agent Selected	Model
"Implement OAuth2 flow"	executor	Sonnet
"Debug race condition in auth"	architect	Opus
"Refactor entire auth module"	executor-high	Opus

Delegation Code:

```
Task(
  subagent_type="oh-my-claudecode:executor-low",
  model="haiku",
```

AGENT COMPOSITION

Skills + Agents combine for powerful workflows:

```
"ralph ultrawork: migrate database"
   |        |
   |        +-> Parallel execution (ultrawork)
   +-----------> Persistence (ralph)
```

Real Example:

```
ralph ultrawork git-master: refactor authentication
  |       |         |
  |       |         +-> Git expertise (atomic commits)
  |       +------------> Maximum parallelism
  +--------------------> Won't stop until verified complete
```

Result: Persistent, parallel, git-aware refactoring

DELEGATION CATEGORIES

Semantic task categorization with auto-detection:

Category	Tier	Temp	Thinking	Auto-Detec From
`visual-engineering`	HIGH	0.7	high	"UI", "comp "style
`ultrabrain`	HIGH	0.3	max	"debu "archi

Category	Tier	Temp	Thinking	Auto-Detect From
`artistry`	MEDIUM	0.9	medium	"creat "brain
`quick`	LOW	0.1	low	"find", is", "w
`writing`	MEDIUM	0.5	medium	"docu "expla

How It Works:

```
User: "debug the race condition in auth"
|
v
     Detected: "debug" keyword
|
v
     Category: ultrabrain
|
v
     Settings: HIGH tier, temp=0.3, max thinking
```

SECTION 4

LIVE DEMO SCENARIOS

DEMO 1: AUTOPILOT

Command:

```
autopilot: build a REST API for a bookstore with CRUD operati
```

What Happens:

1. **Expansion Phase** (~2 min)
    - Analyst extracts: entities (Book, Author), operations (CRUD), constraints
    - Architect creates: technical spec, database schema, API design
2. **Planning Phase** (~1 min)
    - Architect creates implementation plan
    - Critic validates completeness
3. **Execution Phase** (~10-15 min)
    - Executors implement routes, models, tests in parallel

DEMO 2: ULTRAWORK

Command:

```
ulw fix all TypeScript errors
```

What Happens:

```
[ULTRAWORK ACTIVATED]

Scanning for TypeScript errors...
Found 23 errors across 8 files.

Spawning parallel agents:
  [executor-low:1] -> src/api/routes.ts (5 errors)
  [executor-low:2] -> src/api/handlers.ts (3 errors)
  [executor-low:3] -> src/ui/App.tsx (4 errors)
  [executor-low:4] -> src/db/models.ts (6 errors)
  [executor-low:5] -> src/utils/helpers.ts (5 errors)
```

```
Progress: [====================] 100%

All 23 errors fixed in 2m 34s
```

DEMO 3: PIPELINE

Command:

```
/pipeline review "add rate limiting to the API"
```

What Happens:

```
PIPELINE: review
Stages: explore -> architect -> critic -> executor

[Stage 1/4: explore] RUNNING
  Finding rate limiting patterns in codebase...
  Found: middleware pattern in src/middleware/
  Found: express-rate-limit in dependencies
  OUTPUT: 3 relevant files identified

[Stage 2/4: architect] RUNNING
  Analyzing: existing middleware pattern
  Recommendation: Use express-rate-limit with Redis store
```

```
Design: Per-route configuration
OUTPUT: Implementation plan created
```

DEMO 4: PLANNING

Command:

```
plan the authentication system
```

What Happens:

```
[PLAN SKILL ACTIVATED]

I'll help you plan the authentication system. Let me ask a fe

Q1: What authentication methods do you need?
    [ ] Username/Password
    [ ] OAuth2 (Google, GitHub)
    [ ] Magic Link (email)
    [ ] API Keys

Q2: What session management approach?
    [ ] JWT tokens
    [ ] Server-side sessions
    [ ] Hybrid
```

DEMO 5: RALPH

Command:

```
ralph: refactor the auth module to use dependency injection
```

What Happens:

```
[RALPH ACTIVATED - Will not stop until verified complete]

Iteration 1/10:
  Analyzing auth module structure...
  Creating refactoring plan...
  Executing changes...
  ERROR: Test failure in auth.test.ts

Iteration 2/10:
  Analyzing failure: Mock not updated for new DI pattern
  Fixing test mocks...
  Re-running tests...
  ERROR: Type error in UserService

Iteration 3/10:
```

SECTION 5

DEVELOPER EXPERIENCE

MAGIC KEYWORDS

Optional shortcuts for power users:

Keyword	Effect	Example
`autopilot`	Full autonomous execution	`autopilot: build todo app`
`ralph`	Persistence until complete	`ralph: fix auth bugs`
`ulw`	Maximum parallelism	`ulw fix all errors`

Keyword	Effect	Example
`eco`	Token-efficient execution	`eco: add validation`
`plan`	Interactive planning	`plan the API`
`ralplan`	Iterative planning consensus	`ralplan new feature`

Combinations work:

```
ralph ulw: migrate database
  ^     ^
```

HUD STATUSLINE

Real-time visibility into OMC state:

```
+-----------------------------------------------------------
| OMC | autopilot:exec | 3 agents | 5/12 tasks | ctx:45% | $2
+-----------------------------------------------------------
     ^
  ^          ^             ^         ^
     |
  |          |             |         |
  Active mode    # running   Progress     Context     Cost
     agents                  window
```

Setup:

```
/oh-my-claudecode:hud setup
```

Presets:

- `minimal` - Just active mode
- `focused` - Mode + progress (default)
- `full` - Everything including cost

NOTEPAD WISDOM SYSTEM

Plan-scoped knowledge capture:

Location: `.omc/notepads/{plan-name}/`

File	Purpose	Example
`learnings.md`	Technical discoveries	"Redis requires explicit TTL for rate limit keys"
`decisions.md`	Design decisions	"Chose JWT over sessions for

File	Purpose	Example
		stateless scaling"
`issues.md`	Known issues	"OAuth callback URL must be HTTPS in prod"
`problems.md`	Blockers	"Need Redis instance for rate limiting"

API:

ANALYTICS & COST TRACKING

Track token usage and costs:

```
$ omc-analytics summary

Session Summary (last 7 days)
-----------------------------
Total sessions: 23
Total tokens: 1,234,567
Total cost: $18.45

By Model:
  Haiku:  890,000 tokens  ($0.89)
  Sonnet: 300,000 tokens  ($4.50)
  Opus:    44,567 tokens  ($13.06)

By Mode:
  autopilot:  45% of cost
```

SECTION 6

GETTING STARTED

INSTALLATION

Method 1: Plugin Marketplace (Recommended)

```
/plugin marketplace add https://github.com/Yeachan-Heo/oh-my-
/plugin install oh-my-claudecode
```

Method 2: NPM Global

```
npm install -g oh-my-claude-sisyphus
```

Method 3: Manual Git Clone

```
git clone https://github.com/Yeachan-Heo/oh-my-claudecode.gi
cd oh-my-claudecode
npm install && npm run build
```

Requirements:

- Claude Code CLI
- Claude Max/Pro subscription OR Anthropic API key
- Node.js 20+

FIRST STEPS

Step 1: Install

```
/plugin marketplace add https://github.com/Yeachan-Heo/oh-my-
/plugin install oh-my-claudecode
```

Step 2: Setup

```
/oh-my-claudecode:omc-setup
```

(Configures defaults, HUD, preferences)

Step 3: Build something

```
autopilot: build a REST API for managing tasks
```

That's it. Everything else is automatic.

CONFIGURATION

Project-level: `CLAUDE.md` in project root **Global:** `~/.claude/CLAUDE.md`

Key Settings:

```
// ~/.claude/settings.json
{
  "omc": {
    "defaultExecutionMode": "ultrawork",  // or "ecomode"
    "autopilot": {
      "maxIterations": 10,
      "maxQaCycles": 5,
      "skipValidation": false
    },
    "hud": {
      "preset": "focused"
    }
    }
}
```

Agent Customization:

- Modify agent prompts in `agents/*.md`
- Override tools per agent
- Create custom agents

SECTION 7

CLOSING

REAL-WORLD USE CASES

Use Case	Best Mode	Why
Backend API development	autopilot	Full end-to-end workflow
Frontend component library	ultrapilot	Many independent components
Database migrations	ralph	Needs persistence

Use Case	Best Mode	Why
		through errors
CI/CD pipeline setup	pipeline:implement	Sequential stages
Documentation generation	swarm:writer	Parallel doc writing
Bug triage & fixing	swarm:executor	Many independent fixes

Use Case	Best Mode	Why
Security audit	pipeline:security	Structured review process
Exploratory prototyping	ecomode	Budget-conscious iteration

RESOURCES

GitHub Repository

```
github.com/Yeachan-Heo/oh-my-claudecode
```

Website & Documentation

```
yeachan-heo.github.io/oh-my-claudecode-website
```

NPM Package

```
npm install -g oh-my-claude-sisyphus
```

Documentation Directory

```
/docs/REFERENCE.md       - Complete feature reference
/docs/MIGRATION.md       - Upgrade guide
/docs/ARCHITECTURE.md    - How it works
```

Getting Help

```
/oh-my-claudecode:help     - Usage guide
/oh-my-claudecode:doctor   - Diagnose issues
```

Q&A

Common Questions:

Question	Answer
Does OMC work with Claude API keys?	Yes, both Max/Pro subscription and API keys work
Can I use OMC with other AI models?	No, OMC is specifically for Claude Code

Question	Answer
How do I stop a runaway autopilot?	Say "stop", "cancel", or `/oh-my-claudecode:cancel`
Why is my HUD not showing?	Run `/oh-my-claudecode:hud setup`
Can I create custom agents?	Yes, add `.md` files to `agents/` directory
Is there a cost limit?	No built-in limit, but ecomode helps control costs

Questions?

THANK YOU

oh-my-claudecode

Zero learning curve. Maximum power.

```
github.com/Yeachan-Heo/oh-my-claudecode
```

Get Started Now:

```
/plugin marketplace add https://github.com/Yeachan-Heo/oh-my-
/plugin install oh-my-claudecode
autopilot: build something amazing
```

APPENDIX A: COMPLETE AGENT REFERENCE

Agent	Model	Best For
architect	opus	Complex architecture, deep debugging
architect-medium	sonnet	Moderate analysis
architect-low	haiku	Quick code questions
executor	sonnet	Standard implementation

Agent	Model	Best For
executor-high	opus	Complex refactoring
executor-low	haiku	Simple fixes
explore	haiku	Fast file search
explore-medium	sonnet	Pattern matching
explore-high	opus	Architectural search
designer	sonnet	UI components

Agent	Model	Best For
designer-high	opus	Design systems
designer-low	haiku	Simple styling

--

APPENDIX A: COMPLETE AGENT REFERENCE (CONTINUED)

Agent	Model	Best For
researcher	sonnet	External docs, APIs

Agent	Model	Best For
researcher-low	haiku	Quick lookups
writer	haiku	Documentation
vision	sonnet	Image analysis
planner	opus	Strategic planning
analyst	opus	Requirements extraction
critic	opus	Plan review
qa-tester	sonnet	CLI testing
qa-tester-high	opus	Comprehensive QA

Agent	Model	Best For
security-reviewer	opus	Security audits
security-reviewer-low	haiku	Quick security scan

--

APPENDIX A: COMPLETE AGENT REFERENCE (CONTINUED)

Agent	Model	Best For
build-fixer	sonnet	Build error resolution

APPENDIX B: COMPLETE SKILL REFERENCE

Skill	Purpose	Trigger
autopilot	Full autonomous execution	"autopilot", "build me"
ultrapilot	Parallel autopilot	"ultrapilot", "parallel build"
ralph	Persistence mode	"ralph", "don't stop"

Skill	Purpose	Trigger
ultrawork	Maximum parallelism	"ulw", "ultrawork"
ecomode	Token-efficient mode	"eco", "budget"
swarm	Coordinated agents	`/swarm N:agent`
pipeline	Sequential chaining	`/pipeline preset`
plan	Planning interview	"plan the"

Skill	Purpose	Trigger
ralplan	Iterative planning	"ralplan"
cancel	Stop any mode	"stop", "cancel"

--

APPENDIX B: COMPLETE SKILL REFERENCE (CONTINUED)

Skill	Purpose	Trigger
analyze	Deep investigation	"analyze", "debug"

Skill	Purpose	Trigger
deepsearch	Thorough search	"search", "find"
deepinit	Generate AGENTS.md	"index codebase"
frontend-ui-ux	Design sensibility	UI context (auto)
git-master	Git expertise	Git context (auto)
ultraqa	QA cycling	"test", "QA"
learner	Extract skills	"extract skill"

Skill	Purpose	Trigger
note	Save to notepad	"remember", "note"
hud	Configure HUD	`/hud`
doctor	Diagnose issues	`/doctor`

--

APPENDIX B: COMPLETE SKILL REFERENCE (CONTINUED)

Skill	Purpose	Trigger
help	Show usage guide	`/help`
omc-setup	Setup wizard	`/omc-setup`
ralph-init	Initialize PRD	`/ralph-init`
release	Release workflow	`/release`
review	Review plan	"review plan"
research	Scientist orchestration	"research", "statistics"
tdd	TDD enforcement	"tdd", "test first"

APPENDIX C: KEYBOARD SHORTCUTS SUMMARY

Shortcut	Full Command	Effect
`autopilot:`	`/oh-my-claudecode:autopilot`	Full autono mode
`ralph:`	`/oh-my-claudecode:ralph`	Persist mode
`ulw`	`/oh-my-claudecode:ultrawork`	Paralle execut

Shortcut	Full Command	Effect
eco:	/oh-my-claudecode:ecomode	Token-efficier mode
plan	/oh-my-claudecode:plan	Planni intervi

Combinations:

```
ralph ulw: task          # Persistent + Parallel
ralph eco: task          # Persistent + Efficient
autopilot eco: task      # Auto + Efficient (eco wins)
```