OpenAgents Control

AI Agent Orchestration That Learns Your Patterns

v0.7.1 · MIT License · Built on OpenCode

The Problem

Most AI coding tools produce **generic code** that doesn't match your project's patterns

You spend more time **refactoring AI output** than you saved

Every team member gets **different results** for the same request

There's **no approval gate** — the AI just does things

The OAC Solution

Context-Before-Code — Load your patterns *before* generating anything

Human-in-the-Loop — AI proposes, you approve

Markdown-as-Code — Everything is editable text files

Token Efficient — ~80% reduction via Minimal Viable Information

Architecture

Five Subsystems

Everything lives under `.opencode/` — plain Markdown + JSON, no compilation.

`agent/` — Agent definitions

- `core/` — OpenAgent, OpenCoder
- `subagents/` — 11 specialists

`context/` — Knowledge base

- `core/standards/` — Universal rules
- `project-intelligence/` — YOUR patterns

`command/` — Slash commands

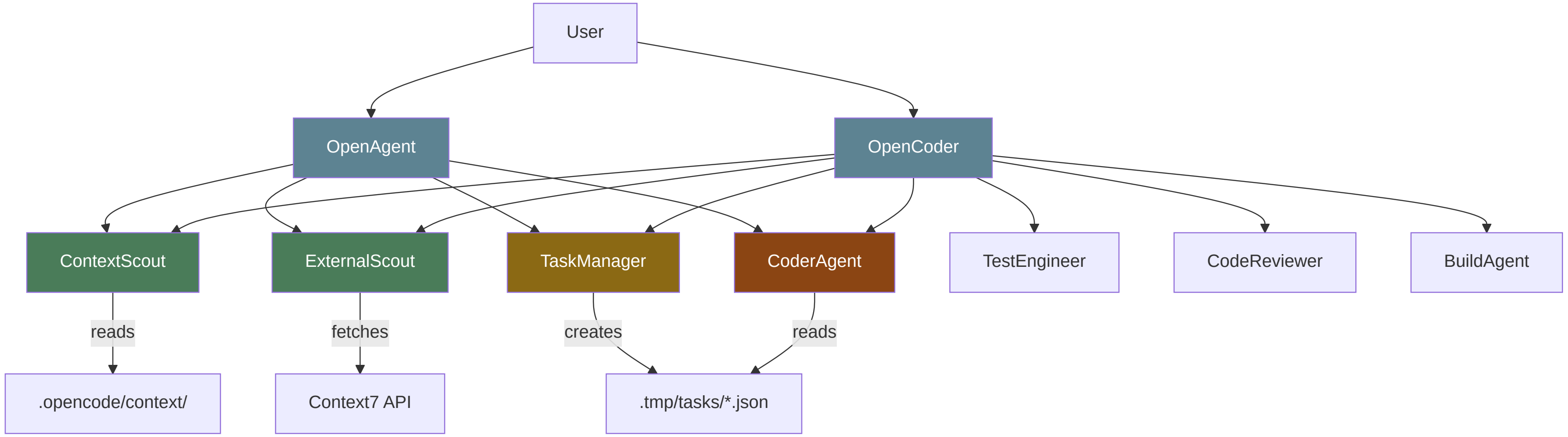
`skills/` — Reusable capabilities

- Task management CLI
- Context7 API integration

`tool/` — Custom tools

```
.opencode/
├── agent/
├── context/
├── command/
├── skills/
└── tool/
```

Agent Hierarchy



Primary Agents

Agent	Purpose	Best For
OpenAgent	Universal — questions, tasks, coordination	Simple features, analysis, learning
OpenCoder	Production — sessions, parallel execution	Complex features, multi-file refactoring

Key Difference

```
OpenAgent:  Analyze → Discover → Approve → Execute → Validate → Summarize
OpenCoder:  Discover → Propose → Init Session → Plan → Execute (Parallel) → Validate
```

OpenCoder adds **session management** (`.tmp/sessions/`) and **parallel batch execution** via TaskManager.

Agent Definition Format

Agents are **Markdown files with YAML frontmatter** — edit text to change behavior:

```
---
name: CoderAgent
description: "Executes coding subtasks with self-review"
temperature: 0
permission:
  bash:
    "*": "deny"
    "bash .opencode/skills/task-management/router.sh complete*": "allow"
    "bash .opencode/skills/task-management/router.sh status*": "allow"
  edit:
    "**/*.env*": "deny"
    "node_modules/**": "deny"
  task:
    contextscout: "allow"
    externalscout: "allow"
---

# Agent instructions in Markdown + XML ...
```

Three permission levels: `"allow"` (silent) · `"ask"` (prompt user) · `"deny"` (blocked)

The Context System

OAC's Core Innovation

Context Categories

- `core/standards/` — Code quality, tests, docs, security
- `core/workflows/` — Code review, delegation, sessions
- `project-intelligence/` — YOUR tech stack & patterns
- `development/` — Language/framework patterns
- `ui/` — Design system, components

Mandatory Loading

Task Type	Must Load
Write code	`code-quality.md`
Write tests	`test-coverage.md`
Write docs	`documentation.md`
Code review	`code-review.md`
Delegate	`task-delegation.md`

> This is **enforced**, not optional. Agents cannot skip context loading.

MVI — Minimal Viable Information

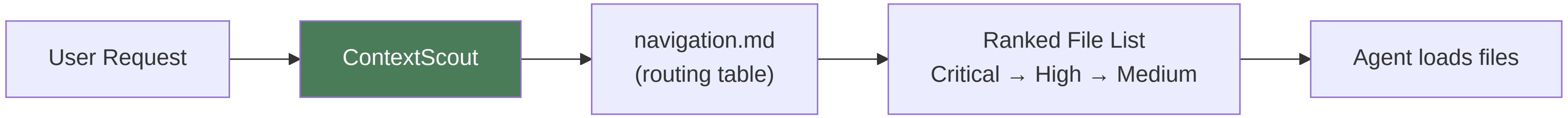
Every context file follows MVI for token efficiency:

- **Core concept:** 1-3 sentences
- **Key points:** 3-5 bullets
- **Minimal example:** 5-10 lines of code
- **Reference link:** to full docs
- **File size:** <200 lines (scannable in <30s)

```
Traditional:  Load entire codebase context → 8,000+ tokens per request
OAC:          Load only relevant patterns  →   750  tokens per request
                                             ─────────────────────────
                                             ~80% token reduction
```

ContextScout — Discovery Agent

Read-only subagent that discovers relevant context files:



Constraints: Can only `read`, `grep`, `glob` — cannot write, edit, bash, or delegate.

ExternalScout complements it by fetching **live docs** for external libraries via Context7 API — preventing outdated training data from producing broken code.

Project Intelligence

Teach agents YOUR patterns with `/add-context` (~5 min wizard):

1. **Tech stack** — Framework, language, database, styling
2. **API pattern** — Paste your actual endpoint code
3. **Component pattern** — Paste your actual component code
4. **Naming conventions** — Files, components, functions, database
5. **Code standards** — TypeScript strict, Zod validation, etc.
6. **Security requirements** — Input validation, parameterized queries

Creates `project-intelligence/technical-domain.md` — marked **Priority: critical**, loaded before every code generation task.

Workflow Engine

Critical Rules

Enforced across **all** agents — higher tiers always win:

Rule	Enforcement
Approval Gate	Request approval before ANY write/edit/bash
Stop on Failure	STOP on errors — NEVER auto-fix
Report First	REPORT → PROPOSE → APPROVE → FIX
Context Loading	NEVER execute without loading context

```
Tier 1: Safety & Approval Gates      ← Always wins
Tier 2: Core Workflow (stages)
Tier 3: Optimization (speed)
```

OpenCoder Workflow

CoderAgent Self-Review

Before signaling completion, every CoderAgent runs a **mandatory 4-check loop**:

1. **Type & Import Validation** — Mismatched signatures, missing imports, circular deps
2. **Anti-Pattern Scan** — `console.log`, `TODO`, hardcoded secrets, `any` types
3. **Acceptance Criteria** — Every criterion from the subtask JSON is met
4. **ExternalScout Verification** — Usage matches documented API

```
Self-Review: ✅ Types clean | ✅ Imports verified | ✅ No debug artifacts
             ✅ All acceptance criteria met | ✅ External libs verified
```

If ANY check fails → fix before signaling completion.

Task Management

Task Decomposition

For complex features (4+ files), TaskManager creates atomic JSON subtasks:

```
.tmp/tasks/user-auth/
├── task.json              ← Feature metadata
├── subtask_01.json        ← parallel: true, no deps
├── subtask_02.json        ← parallel: true, no deps
├── subtask_03.json        ← depends_on: [01, 02]
└── subtask_04.json        ← depends_on: [03]
```

Key distinction:

- `context_files` = standards to follow (conventions, patterns)
- `reference_files` = source code to study (existing project files)
- **Never mix them.**

Subtask Schema

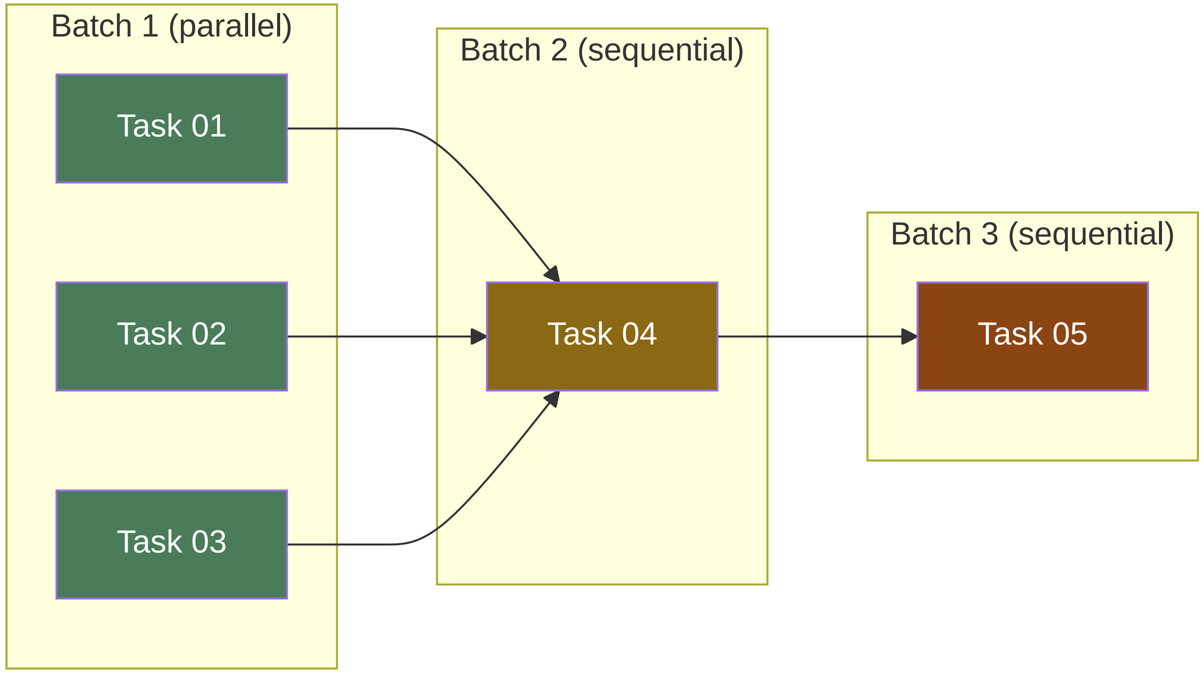
```json
{
  "id": "user-auth-02",
  "seq": "02",
  "title": "Implement JWT service",
  "status": "pending",
  "depends_on": ["01"],
  "parallel": false,
  "suggested_agent": "CoderAgent",
  "context_files": [".opencode/context/core/standards/code-quality.md"],
  "acceptance_criteria": [
    "JWT tokens signed with RS256",
    "Access tokens expire in 15 minutes"
  ],
  "deliverables": ["src/auth/jwt.service.ts"]
}
```

Enhanced schema v2.0 adds **line-number precision** for large context files:

```json
"context_files": [
  { "path": "code-quality.md", "lines": "53-95", "reason": "Pure function patterns" }
]
```

Parallel Execution

Tasks grouped into dependency-ordered batches:



- **Within a batch:** All tasks start simultaneously
- **Between batches:** Wait for 100% completion

Task CLI

```bash
# Check overall progress
bash .opencode/skills/task-management/router.sh status

# Find next eligible tasks (dependencies satisfied)
bash .opencode/skills/task-management/router.sh next

# Find parallelizable tasks
bash .opencode/skills/task-management/router.sh parallel

# Mark subtask complete
bash .opencode/skills/task-management/router.sh complete user-auth 01 \
  "Implemented JWT auth with refresh tokens"

# Validate JSON files and dependency graph
bash .opencode/skills/task-management/router.sh validate
```

Commands & Skills

Slash Commands

Command	Purpose
`/add-context`	Teach patterns
`/commit`	Smart git commit
`/test`	Test pipeline
`/optimize`	Code analysis
`/context harvest`	Clean summaries
`/context validate`	Check integrity
`/context migrate`	Global → local

Skills

Task Management

```
router.sh status|next|parallel
router.sh complete <f> <s> "msg"
router.sh validate
```

Context7 API

```
# Search for library
curl -s "https://context7.com/api/v2/\
libs/search?libraryName=react"

# Fetch docs
curl -s "https://context7.com/api/v2/\
context?libraryId=ID&query=hooks"
```

No API key required.

End-to-End Data Flow

Complete Feature Request Flow

Customization

What You Can Edit

Agent behavior

```
nano .opencode/agent/core/opencoder.md
```

Agent model

```
model: anthropic/claude-sonnet-4-5
```

Permissions

```
permission:
  bash:
    "docker *": "allow"
```

Project patterns

```
/add-context          # wizard
/add-context --update # update
```

Custom commands

```
# Create .opencode/command/my-cmd.md
/my-cmd
```

Custom skills

```
# Create .opencode/skills/my-skill/
# with SKILL.md
```

Team sharing

```
git add .opencode/context/
git commit && git push
# Everyone uses same patterns
```

For Teams

Share patterns via git — commit `.opencode/context/project-intelligence/`

New developers inherit standards on day 1 — no onboarding docs to read

Local overrides global — project patterns always win

Different projects, different patterns:

```
project-a/.opencode/context/project-intelligence/
  technical-domain.md  → React 19 + TypeScript + PostgreSQL

project-b/.opencode/context/project-intelligence/
  technical-domain.md  → FastAPI + Python + MongoDB
```

Knowledge Persistence

How Project Knowledge Is Saved

Four Knowledge Layers

Layer	Where	Trigger	Auto?
Project Intelligence	`.opencode/context/project-intelligence/`	User runs `/add-context`	No
External Docs Cache	`.tmp/external-context/`	Agent fetches in approved workflow	Semi
Session Context	`.tmp/sessions/`	Agent creates after approval	Semi
Harvested Knowledge	`.opencode/context/`	User runs `/context harvest`	No

Key insight: OAC has **no background automation**. No file watchers, no git hooks, no cron jobs. Permanent knowledge always requires explicit user action.

Permanent Knowledge

Project Intelligence (user-triggered)

Event	Action
First setup	`/add-context`
Stack changes	`/add-context --update`
New decision	Edit `decisions-log.md`
New debt	Edit `living-notes.md`

Git-committed. Versioned. Reviewed per PR.

Harvested Knowledge (user-triggered)

```
/context harvest
# Scans for *OVERVIEW.md, SESSION-*.md
```

Ephemeral Knowledge

External Docs Cache (semi-automatic)

```
Agent detects library
  → ExternalScout checks cache
  → Hit (<7 days)? Return paths
  → Miss? Fetch via Context7 API
         Write to .tmp/
         Return paths
```

Never git-committed. 7-day TTL.

Session Context (semi-automatic)

```
User approves plan
  → Agent creates context.md
  → All subagents read from it
  → Task completes
  → Agent asks: "Clean up?"
  → User confirms deletion
```

Agent-created, user-destroyed.

Knowledge Lifecycle

Comparison

OAC vs Alternatives

Dimension	OAC	Cursor/Copilot	Aider
Pattern learning	Built-in context	None	None
Approval gates	Always required	Optional	None
Agent editability	Markdown files	Proprietary	Limited
Token efficiency	MVI (~80% less)	Full context	Full context
Team standards	Shared via git	Per-user	No support
Parallel execution	Batch-based	N/A	N/A

When to Use OAC

- You have **established patterns** and want AI to follow them
- You need **approval gates** for quality control

Design Principles

1. **Markdown-as-Code** — No compilation, no vendor lock-in
2. **Context-Before-Code** — Load standards before generating
3. **Human-in-the-Loop** — AI proposes, human approves
4. **Token Efficiency (MVI)** — <200 lines per context file
5. **Standards ≠ Source** — `context_files` vs `reference_files`
6. **Parallel-Aware Tasks** — Dependency-ordered batch execution
7. **Self-Review Before Handoff** — 4-check mandatory loop
8. **Stateless Subagents** — Everything passed via context file
9. **Local-First Resolution** — Project patterns always win
10. **Least Privilege** — Each agent gets only what it needs

Quick Start

```
# Install (developer profile)
curl -fsSL https://raw.githubusercontent.com/darrenhinde/\
  OpenAgentsControl/main/install.sh | bash -s developer

# Teach it your patterns (~5 min)
/add-context

# Start coding
opencode --agent OpenCoder
> "Create a user authentication system"
```

GitHub · Docs · Context7 · Community

Thank You

Full guide: `docs/oac_guide/00-overview.md` through `09-knowledge-persistence.md`

Repository: github.com/darrenhinde/OpenAgentsControl