Claude Code V4 Guide

85% Context Reduction, Custom Agents & Session Teleportation

JANUARY 25, 2026

THEDECIPHERIST.COM

TABLE OF CONTENTS

V4: THE JANUARY 2026 REVOLUTION

Previous guides: V1 | V2 | V3

Because of the overwhelming support on V1-V3, I'm back with V4. Huge thanks to everyone who contributed to the previous guides: u/BlueVajra, u/stratofax, u/antoniocs, u/GeckoLogic, u/headset38, u/tulensrma, u/jcheroske, and the rest of the community. Your feedback made each version better.

Claude Code 2.1.x shipped 1,096+ commits in January alone. This isn't an incremental update - it's a fundamental shift in how Claude Code manages context, delegates work, and scales.

What's new in V4:

- **Part 9: MCP Tool Search** - 85% context reduction with lazy loading
- **Part 10: Custom Agents** - Automatic delegation to specialists
- **Part 11: Session Teleportation** - Move sessions between devices
- **Part 12: Background Tasks** - Parallel agent execution
- **Part 13: New Commands & Shortcuts** - `/config` search, `/stats` filtering, custom keybindings
- Updated GitHub repo with V4 templates coming soon

TL;DR: MCP Tool Search reduces context overhead by 85% (77K -> 8.7K tokens) by lazy-loading tools on-demand. Custom Agents let you create specialists that Claude invokes automatically - each with isolated context windows. Session Teleportation lets you move work between terminal and claude.ai/code seamlessly. Background Tasks enable parallel agent execution with Ctrl+B. And the new Setup hook automates repository initialization.

TABLE OF CONTENTS

Foundation (From V1-V3)

New in V4

Reference

PART 1: THE GLOBAL CLAUDE.MD AS SECURITY GATEKEEPER

The Memory Hierarchy

Claude Code loads CLAUDE.md files in a specific order:

LEVEL	LOCATION	PURPOSE
Enterprise	`/etc/claude-code/CLAUDE.md`	Org-wide policies
Global User	`~/.claude/CLAUDE.md`	Your standards for ALL projects
Project	`./CLAUDE.md`	Team-shared project instructions
Project Local	`./CLAUDE.local.md`	Personal project overrides

Your global file applies to **every single project** you work on.

What Belongs in Global

1. Identity & Authentication

```
## GitHub Account
**ALWAYS** use **YourUsername** for all projects:
- SSH: `git@github.com:YourUsername/<repo>.git`

## Docker Hub
Already authenticated. Username in `~/.env` as `DOCKER_HUB_USER`
```

Why global? You use the same accounts everywhere. Define once, inherit everywhere.

2. The Gatekeeper Rules

```
## NEVER EVER DO

These rules are ABSOLUTE:

### NEVER Publish Sensitive Data
- NEVER publish passwords, API keys, tokens to git/npm/docker
- Before ANY commit: verify no secrets included

### NEVER Commit .env Files
- NEVER commit `.env` to git
- ALWAYS verify `.env` is in `.gitignore`
```

Why This Matters: Claude Reads Your .env

Security researchers discovered that Claude Code **automatically reads `.env` files** without explicit permission. Backslash Security warns:

> *"If not restricted, Claude can read `.env`, AWS credentials, or `secrets.json` and leak them through 'helpful suggestions.'"*

Your global CLAUDE.md creates a **behavioral gatekeeper** - even if Claude has access, it won't output secrets.

Syncing Global CLAUDE.md Across Machines

If you work on multiple computers, sync your `~/.claude/` directory using a dotfiles manager:

```
# Using GNU Stow
cd ~/dotfiles
stow claude  # Symlinks ~/.claude to dotfiles/claude/.claude
```

This gives you:

- Version control on your settings
- Consistent configuration everywhere
- Easy recovery if something breaks

Defense in Depth

LAYER	WHAT	HOW
1	Behavioral rules	Global CLAUDE.md "NEVER" rules
2	Access control	Deny list in settings.json
3	Git safety	.gitignore

Team Workflows: Evolving CLAUDE.md

Boris Cherny shares how Anthropic's Claude Code team does it:

> *"Our team shares a single CLAUDE.md for the Claude Code repo. We check it into git, and the whole team contributes multiple times a week."*

The pattern: Mistakes become documentation.

```
Claude makes mistake -> You fix it -> You add rule to CLAUDE.md -> Never happer
```

PART 2: GLOBAL RULES FOR NEW PROJECT SCAFFOLDING

Your global CLAUDE.md becomes a **project factory**. Every new project automatically inherits your standards.

The Problem Without Scaffolding Rules

Research from project scaffolding experts:

> *"LLM-assisted development fails by silently expanding scope, degrading quality, and losing architectural intent."*

The Solution

```
## New Project Setup

When creating ANY new project:

### Required Files
- `.env` - Environment variables (NEVER commit)
- `.env.example` - Template with placeholders
- `.gitignore` - Must include: .env, node_modules/, dist/
- `CLAUDE.md` - Project overview

### Required Structure
project/
├── src/
├── tests/
├── docs/
├── .claude/
│   ├── skills/
│   ├── agents/
│   └── commands/
└── scripts/

### Node.js Requirements
Add to entry point:
process.on('unhandledRejection', (reason, promise) => {
  console.error('Unhandled Rejection:', reason);
  process.exit(1);
});
```

When you say "create a new Node.js project," Claude reads this and **automatically** creates the correct structure.

PART 3: MCP SERVERS - CLAUDE'S INTEGRATIONS

MCP (Model Context Protocol) lets Claude interact with external tools.

Adding MCP Servers

```
claude mcp add <server-name> -- <command>
claude mcp list
claude mcp remove <server-name>
```

When NOT to Use MCP

MCP servers consume tokens and context. For simple integrations, consider alternatives:

USE CASE	MCP OVERHEAD	ALTERNATIVE
Trello tasks	High	CLI tool (`trello-cli`)
Simple HTTP calls	Overkill	`curl` via Bash
One-off queries	Wasteful	Direct command

Rule of thumb: If you're calling an MCP tool once per session, a CLI is more efficient. MCP shines for *repeated* tool use within conversations.

UPDATE V4: With MCP Tool Search (Part 9), this tradeoff changes significantly. You can now have many more MCP servers without paying the upfront context cost.

Recommended MCP Servers for Developers

Core Development

SERVER	PURPOSE	INSTALL
Context7	Live docs for any library	`claude mcp add context7 -- npx -y @upstash/context7-mcp@latest`
GitHub	PRs, issues, CI/CD	`claude mcp add github -- npx -y @modelcontextprotocol/server-github`
Filesystem	Advanced file operations	`claude mcp add filesystem -- npx -y @modelcontextprotocol/server-filesystem`
Sequential Thinking	Structured problem-solving	`claude mcp add sequential-thinking -- npx -y @modelcontextprotocol/server-sequential-thinking`

Databases

SERVER	PURPOSE	INSTALL
MongoDB	Atlas/Community, Performance Advisor	`claude mcp add mongodb -- npx -y mongodb-mcp-server`
PostgreSQL	Query Postgres naturally	`claude mcp add postgres -- npx -y @modelcontextprotocol/server-postgres`
DBHub	Universal (MySQL, SQLite, etc.)	`claude mcp add db -- npx -y @bytebase/dbhub`

Documents & RAG

SERVER	PURPOSE	INSTALL
Docling	PDF/DOCX parsing, 97.9% table accuracy	`claude mcp add docling -- uvx docling-mcp-server`
Qdrant	Vector search, semantic memory	`claude mcp add qdrant -- npx -y @qdrant/mcp-server`
Chroma	Embeddings, vector DB	`claude mcp add chroma -- npx -y @chroma/mcp-server`

Browser & Testing

SERVER	PURPOSE	INSTALL
Playwright	E2E testing, scraping	`claude mcp add playwright -- npx -y @anthropic-ai/playwright-mcp`
Browser MCP	Use your logged-in Chrome	browsermcp.io

Cloud & DevOps

SERVER	PURPOSE	INSTALL
AWS	S3, Lambda, CloudWatch	`claude mcp add aws -- npx -y @anthropic-ai/aws-mcp`
Docker	Container management	`claude mcp add docker -- npx -y @anthropic-ai/docker-mcp`
Kubernetes	Cluster operations	`claude mcp add k8s -- npx -y @anthropic-ai/kubernetes-mcp`

PART 4: COMMANDS - PERSONAL SHORTCUTS

Commands are personal macros that expand into prompts. Store them in:

- `~/.claude/commands/` - Available everywhere
- `.claude/commands/` - Project-specific

Basic Command

Create `~/.claude/commands/review.md`:

```
---
description: Review code for issues
---

Review this code for:
1. Security vulnerabilities
2. Performance issues
3. Error handling gaps
4. Code style violations
```

Usage: Type `/review` in any session.

Command with Arguments

Create `~/.claude/commands/ticket.md`:

```
---
description: Create a ticket from description
argument-hint: <ticket-description>
---

Create a detailed ticket for: $ARGUMENTS

Include:
- User story
- Acceptance criteria
- Technical notes
```

Usage: `/ticket Add dark mode support`

Advanced: Commands with Bash Execution

```
---
description: Smart commit with context
allowed-tools: Bash(git add:*), Bash(git status:*), Bash(git commit:*)
argument-hint: [message]
---

## Context
- Current git status: !`git status`
- Current git diff: !`git diff HEAD`
- Current branch: !`git branch --show-current`
- Recent commits: !`git log --oneline -5`

## Task
Create a commit with message: $ARGUMENTS
```

The `!` backtick syntax runs bash commands before the prompt is processed.

PART 5: SKILLS - REUSABLE EXPERTISE

Skills are **triggered expertise** that load only when needed. Unlike CLAUDE.md (always loaded), skills use progressive disclosure to save context.

Creating a Skill

Create `.claude/skills/code-review/SKILL.md`:

```
---
name: Code Review
description: Comprehensive code review with security focus
triggers:
  - review
  - audit
  - check code
---

# Code Review Skill

When reviewing code:
1. Check for security vulnerabilities (OWASP Top 10)
2. Look for performance issues (N+1 queries, memory leaks)
3. Verify error handling (edge cases, null checks)
4. Assess test coverage
5. Review naming and documentation
```

Progressive Disclosure

Skills use **progressive disclosure** for token efficiency:

1. **Startup**: Only name/description loaded (~50 tokens)
2. **Triggered**: Full SKILL.md content loaded
3. **As needed**: Additional resources loaded

Rule of thumb: If instructions apply to <20% of conversations, make it a skill instead of putting it in CLAUDE.md.

V4 Update: Automatic Skill Discovery

Claude Code now automatically discovers skills from nested `.claude/skills` directories when working with files in subdirectories. No need to reference the root - skills are found recursively.

PART 6: WHY SINGLE-PURPOSE CHATS ARE CRITICAL

Research consistently shows mixing topics destroys accuracy.

Studies on multi-turn conversations:

> *"An average **39% performance drop** when instructions are delivered across multiple turns."*

Chroma Research on context rot:

> *"As tokens in the context window increase, the model's ability to accurately recall information decreases."*

The Golden Rule

> ***"One Task, One Chat"***

SCENARIO	ACTION
New feature	New chat
Bug fix (unrelated)	`/clear` then new task
Research vs implementation	Separate chats
20+ turns elapsed	Start fresh

Use `/clear` Liberally

```
/clear
```

Anthropic recommends:

> *"Use `/clear` frequently between tasks to reset the context window."*

V4 Update: Context Window Visibility

You can now see exactly where your context is going:

```
/context
```

New status line fields:

- `context_window.used_percentage`
- `context_window.remaining_percentage`

PART 7: HOOKS - DETERMINISTIC ENFORCEMENT

CLAUDE.md rules are **suggestions** Claude can ignore under context pressure. Hooks are **deterministic** - they always run.

The Critical Difference

MECHANISM	TYPE	RELIABILITY
CLAUDE.md rules	Suggestion	Can be overridden
Hooks	**Enforcement**	Always executes

Hook Events

EVENT	WHEN	USE CASE
`PreToolUse`	Before tool executes	Block dangerous ops
`PostToolUse`	After tool completes	Run linters
`Stop`	Claude finishes turn	Quality gates
`Setup`	**On init/maintenance**	**Repo initialization (V4)**

Example: Block Secrets Access

Add to `~/.claude/settings.json`:

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Read|Edit|Write",
        "hooks": [{
          "type": "command",
          "command": "python3 ~/.claude/hooks/block-secrets.py"
        }]
      }
    ]
  }
}
```

The hook script:

```
#!/usr/bin/env python3
import json, sys
from pathlib import Path

SENSITIVE = {'.env', '.env.local', 'secrets.json', 'id_rsa'}

data = json.load(sys.stdin)
file_path = data.get('tool_input', {}).get('file_path', '')

if Path(file_path).name in SENSITIVE:
    print(f"BLOCKED: Access to {file_path} denied.", file=sys.stderr)
    sys.exit(2)  # Exit 2 = block and feed stderr to Claude

sys.exit(0)
```

Hook Exit Codes

CODE	MEANING
0	Allow operation
1	Error (shown to user)
2	**Block operation, tell Claude why**

V4: Setup Hook Event

New in January 2026 - trigger hooks during repository setup:

```
claude --init          # Triggers Setup hook
claude --init-only     # Triggers Setup hook, then exits
claude --maintenance   # Triggers Setup hook for maintenance
```

Example Setup hook for auto-installing dependencies:

```
{
  "hooks": {
    "Setup": [{
      "type": "command",
      "command": "npm install && npm run prepare"
    }]
  }
}
```

PART 8: LSP - IDE-LEVEL CODE INTELLIGENCE

In December 2025 (v2.0.74), Claude Code gained native Language Server Protocol support.

What LSP Enables

LSP gives Claude the same code understanding your IDE has:

CAPABILITY	WHAT IT DOES
Go to Definition	Jump to where any symbol is defined
Find References	See everywhere a function is used
Hover	Get type signatures and docs
Diagnostics	Real-time error detection
Document Symbols	List all symbols in a file

Why This Matters

Before LSP, Claude used **text-based search** (grep, ripgrep) to understand code. Slow and imprecise.

With LSP, Claude has **semantic understanding** - it knows that `getUserById` in file A calls the function defined in file B, not just that the text matches.

Performance: 900x faster (50ms vs 45 seconds for cross-codebase navigation)

Supported Languages

Python, TypeScript, Go, Rust, Java, C/C++, C#, PHP, Kotlin, Ruby, HTML/CSS

Setup

LSP is built-in as of v2.0.74. For older versions:

```
export ENABLE_LSP_TOOL=1
```

PART 9: MCP TOOL SEARCH - THE 85% CONTEXT REVOLUTION

This is the biggest change in V4.

The Problem

Every MCP server you connect brings tool definitions - descriptions, parameters, schemas. Before Tool Search, Claude loaded **all of them** at startup:

```
Before:
Loading 73 MCP tools... [39.8k tokens]
Loading 56 agents... [9.7k tokens]
Loading system tools... [22.6k tokens]
Ready with 92k tokens remaining.  ← 54% context GONE before you type anything
```

Users reported 50-70% of their 200K context consumed before writing a single prompt.

The Solution: Lazy Loading

Claude Code 2.1.7 introduced MCP Tool Search:

```
After:
Loading tool registry... [5k tokens]
Ready with 195k tokens available.  ← 95% context preserved

User: "I need to query the database"
> Auto-loading: postgres-mcp [+1.2k tokens]
> 193.8k tokens remaining
```

How It Works

1. **Detection**: Claude Code checks if MCP tool descriptions would use >10% of context
2. **Registry Creation**: Builds lightweight index of tool names and descriptions
3. **On-Demand Loading**: Tools load only when Claude determines they're needed
4. **Intelligent Caching**: Loaded tools stay available for session duration

The Numbers

METRIC	BEFORE	AFTER	IMPROVEMENT
Initial context usage	~77K tokens	~8.7K tokens	**85% reduction**
Opus 4 accuracy	49%	74%	+25 percentage points
Opus 4.5 accuracy	79.5%	88.1%	+8.6 percentage points

Configuration

MCP Tool Search is **enabled by default** when tools would consume >10% of context.

To check your context usage:

```
/context
```

To disable for specific servers (if you always need certain tools immediately):

```
{
  "mcpServers": {
    "always-needed": {
      "command": "...",
      "enable_tool_search": false
    }
  }
}
```

To configure the auto-enable threshold:

```
{
  "mcp": {
    "tool_search": "auto:15"  // Enable at 15% context usage
  }
}
```

What This Means for You

- **More MCP servers**: Connect dozens without penalty
- **Better accuracy**: Less noise = better tool selection
- **Larger tasks**: More context for actual work
- **No workflow changes**: Tools work exactly as before

Simon Willison commented:

> *"This fixes one of the most painful scaling issues with MCP setups. Was running 5 servers and watching context evaporate before any actual work began."*

PART 10: CUSTOM AGENTS - AUTOMATIC DELEGATION

Custom Agents are specialized assistants that Claude invokes **automatically** - like how it automatically selects tools.

Why Custom Agents?

PROBLEM	SOLUTION
Context pollution from diverse tasks	Each agent has isolated context window
Generic advice for specialized work	Agents have focused system prompts
Manual orchestration overhead	Automatic delegation based on task

Creating a Custom Agent

Method 1: Interactive (Recommended)

```
/agents
```

Select "Create new agent" -> Choose location (User or Project) -> Generate with Claude or Manual.

Method 2: Manual

Create `~/.claude/agents/code-reviewer.md`:

```
---
name: code-reviewer
description: Reviews code for security, performance, and best practices
tools: Read, Grep, Glob
model: sonnet
---

You are a senior code reviewer specializing in:
- Security vulnerabilities (OWASP Top 10)
- Performance antipatterns
- Error handling gaps
- Code maintainability

When reviewing:
1. Start with security concerns
2. Then performance issues
3. Then style/maintainability
4. Provide specific line references
5. Suggest concrete fixes

Be critical but constructive. Explain WHY something is a problem.
```

Agent Configuration Options

```
---
name: agent-name              # Required
description: When to use      # Required - Claude uses this to decide delegatic
tools: Read, Write, Bash      # Optional - inherits all if omitted
model: sonnet                 # Optional - sonnet, opus, or haiku
---
```

How Automatic Delegation Works

Claude delegates based on:

1. Task description in your request
2. `description` field in agent configurations
3. Current context
4. Available tools

Example:

```
You: "Review the authentication module for security issues"

Claude thinks: "This is a code review task focusing on security"
-> Delegates to code-reviewer agent
-> Agent runs with isolated context
-> Returns findings to main conversation
```

Built-in Agents

Claude Code includes these by default:

AGENT	PURPOSE	WHEN USED
Explore	Read-only codebase analysis	Searching, understanding code
Plan	Research for planning	Plan mode context gathering
General-purpose	Complex multi-step tasks	Exploration + modification needed

Best Practices

1. **Keep agents focused**: One specialty per agent
2. **Write clear descriptions**: Claude uses these to decide delegation
3. **Limit tools**: Read-only agents shouldn't have Write access
4. **Test delegation**: Verify Claude routes tasks correctly
5. **Start with 3-4 agents max**: Too many options can confuse routing

Hot Reload

New or updated agents in `~/.claude/agents/` or `.claude/agents/` are available **immediately** - no restart needed.

PART 11: SESSION TELEPORTATION

Move your work between terminal and claude.ai/code seamlessly.

Teleport to Web

```
/teleport
```

Opens your current session at `claude.ai/code`. Perfect for:

- Switching from terminal to visual interface
- Sharing session with collaborators
- Continuing on a different device

Configure Remote Environment

```
/remote-env
```

Set up environment variables and configuration for remote sessions.

Resume Sessions

```
# Continue most recent session
claude --continue
# or
claude -c

# Resume specific session by ID
claude --resume abc123
# or
claude -r abc123

# Resume with a new prompt
claude --resume abc123 "Continue with the tests"
```

VSCode: Remote Session Browsing

OAuth users can now browse and resume remote Claude sessions directly from the Sessions dialog in the VSCode extension.

PART 12: BACKGROUND TASKS & PARALLEL EXECUTION

Backgrounding Tasks

Press Ctrl+B to background:

- Currently running agents
- Shell commands
- Both simultaneously (unified behavior in V4)

Managing Background Tasks

```
/tasks
```

Shows all background tasks with:

- Status indicators
- Inline display of agent's final response
- Clickable links to full transcripts

Task Notifications

When background tasks complete:

- Notifications capped at 3 lines
- Overflow summary for multiple simultaneous completions
- Final response visible without reading full transcript

Disabling Background Tasks

If you prefer the old behavior:

```
export CLAUDE_CODE_DISABLE_BACKGROUND_TASKS=true
```

Or in settings.json:

```
{
  "enableBackgroundTasks": false
}
```

PART 13: NEW COMMANDS, SHORTCUTS & QUALITY OF LIFE

New Commands

COMMAND	WHAT IT DOES
`/config`	Now has **search functionality** - type to filter settings
`/stats`	Press **r** to cycle: Last 7 days, Last 30 days, All time
`/doctor`	Now shows **auto-update channel** and available npm versions
`/keybindings`	Configure custom keyboard shortcuts
`/context`	See exactly where your tokens are going

Custom Keyboard Shortcuts

Create `~/.claude/keybindings.json`:

```
{
  "ctrl+shift+r": "/review",
  "ctrl+shift+d": "/deploy",
  "ctrl+shift+t": "/test",
  "ctrl+shift+c": "/commit"
}
```

Run `/keybindings` to get started.

Essential Shortcuts Reference

SHORTCUT	ACTION
Ctrl+C	Cancel current operation
Ctrl+D	Exit Claude Code
Ctrl+B	Background current task
Shift+Tab	In plan mode: auto-accept edits
Esc Esc	Rewind to previous state (double-tap)
Tab	Autocomplete commands, files, agents
Shift+Enter	Insert newline without submitting
Up/Down	Navigate command history
Ctrl+R	Reverse search history

Plan Mode Improvements

When Claude presents a plan:

- **Shift+Tab**: Quickly select "auto-accept edits"
- **Reject with feedback**: Tell Claude what to change before rerunning

PR Review Indicator

The prompt footer now shows your branch's PR state:

- Colored dot (approved, changes requested, pending, draft)
- Clickable link to the PR

Language Setting

Configure output language for global teams:

```
{
  "language": "ja"  // Japanese output
}
```

Or in CLAUDE.md:

```
## Language
Always respond in Spanish.
```

External CLAUDE.md Imports

Load CLAUDE.md from additional directories:

```
export CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD=1
claude --add-dir ../shared-configs ../team-standards
```

VSCode Improvements

- **Clickable destination selector** for permission requests
- Choose where settings are saved: this project, all projects, shared with team, or session only
- **Secondary sidebar support** (VS Code 1.97+) - Claude Code in right sidebar, file explorer on left
- **Streaming message support** - see responses in real-time as Claude works

Environment Variables Reference

VARIABLE	PURPOSE
`CLAUDE_CODE_DISABLE_BACKGROUND_TASKS`	Disable background task functionality
`CLAUDE_CODE_TMPDIR`	Override temp directory location
`CLAUDE_CODE_ADDITIONAL_DIRECTORIES_CLAUDE_MD`	Enable `--add-dir` CLAUDE.md loading
`FORCE_AUTOUPDATE_PLUGINS`	Allow plugin autoupdate when main auto-updater disabled
`IS_DEMO`	Hide email and organization from UI (for streaming)

QUICK REFERENCE

TOOL	PURPOSE	LOCATION
Global CLAUDE.md	Security + Scaffolding	`~/.claude/CLAUDE.md`
Project CLAUDE.md	Architecture + Team rules	`./CLAUDE.md`
MCP Servers	External integrations	`claude mcp add`
MCP Tool Search	**Lazy loading (85% savings)**	**Automatic when >10% context**
Skills	Reusable expertise	`.claude/skills/*/SKILL.md`
Custom Agents	**Automatic delegation**	`~/.claude/agents/*.md`
Commands	Personal shortcuts	`~/.claude/commands/*.md`
Hooks	Deterministic enforcement	`~/.claude/settings.json`
LSP	Semantic code intelligence	Built-in (v2.0.74+)
Keybindings	**Custom shortcuts**	`~/.claude/keybindings.json`
`/clear`	Reset context	Type in chat
`/context`	View token usage	Type in chat
`/teleport`	Move to claude.ai/code	Type in chat
`/tasks`	Manage background tasks	Type in chat

GITHUB REPO

All templates, hooks, skills, and agents:

github.com/TheDecipherist/claude-code-mastery

- CLAUDE.md templates (global + project)
- Ready-to-use hooks (block-secrets.py, setup hooks)
- Example skills
- **Custom agent templates (V4)**
- **Keybindings examples (V4)**
- settings.json pre-configured

SOURCES

Anthropic Official

- Claude Code Best Practices - Anthropic
- Effective Context Engineering - Anthropic
- Model Context Protocol - Anthropic
- Agent Skills - Anthropic
- Building Agents with Claude Agent SDK - Anthropic
- Custom Subagents Documentation - Claude Code Docs

V4 Feature Coverage

- Claude Code 2.1.0 Update - VentureBeat
- MCP Tool Search Announcement - VentureBeat
- MCP Tool Search Explained - JP Caparas
- Claude Code 2.1 Features - Datasculptor
- Claude Code Changelog - ClaudeLog

Research & Analysis

- Context Rot Research - Chroma
- LLMs Get Lost In Multi-Turn - arXiv
- Claude loads secrets without permission - Knostic
- Compound Engineering - Every

Community Resources

- Awesome Claude Code Subagents - GitHub
- Claude Code Cheatsheet - AwesomeClaude
- How I Use Every Claude Code Feature - Shrivu Shankar

What's in your setup? Drop your agents, hooks, and keybindings below.