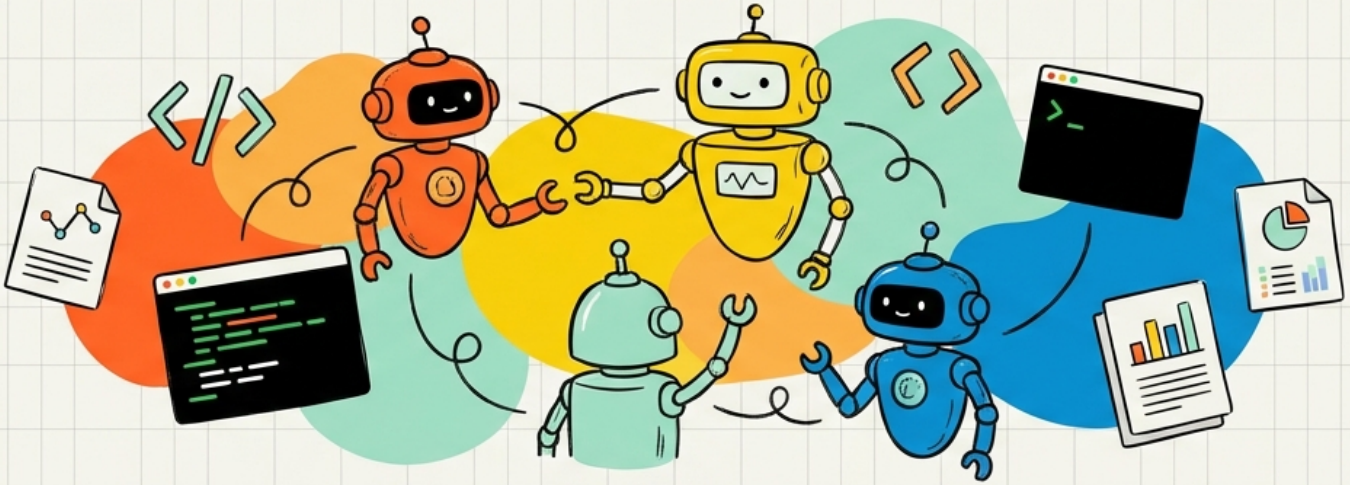
AI Native Development Workshop

Building with Multi-Agent Systems & Context Engineering



Selcuk Atli
selcuk.atli@gmail.com

Introduction

Fulbright Scholar, serial entrepreneur, BS and MS in Computer Science

SocialWire – An advertising platform acquired by Rakuten

Boostable – A Y Combinator–backed advertising tool for marketplace sellers, acquired by Metric Collective

Bunch – A group video chat app for multiplayer games, 10 million+ downloads, raised over $35 million from General Catalyst, EA, Take-Two, Ubisoft, Tencent, Riot Games, Supercell.

Agenda

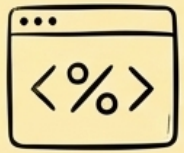
1. My AI Native Journey & Background
2. The AI Development Spectrum
3. Tools & Environment Setup
4. Claude Code Deep Dive
5. Context Engineering
6. MCP Servers & Browser Automation
7. Multi-Agent & Parallel Workflows
8. Staying in Flow
9. Stack & Best Practices
10. Live Demo: Building a Todo App

My AI Native Journey

18+ months

AI Native Coding for 18+ months

Early adopter of Cursor, Claude Code since March 2024



90%+

90%+ of our code is AI generated

Leveraging AI throughout the entire development lifecycle

1-2 products

Ship 1-2 products per month

Rapidly delivering new products with AI-assisted development

The AI Development Spectrum

Three approaches to AI-assisted development

Vibe Coding

Code Assist

AI Native Development

VIBE CODING

For non-technical roles (PMs, designers, marketers)

Tools like Lovable & Replit turn ideas into prototypes

Accelerates communication & iteration

Risk: Technical debt, lack of understanding

AI Native Development

- For technical teams
- Tools (Cursor, Claude Code) transform workflows
- Output & iteration speed increase 5-10x
- Engineering with AI at its core, not just prompting

This is what we're learning today

Code Assist

For developers who primarily hand-code

AI tools (GitHub Copilot) assist with completion, refactoring, debugging

Developers write 50%+ of code, retaining control

Limitation: Single-turn, no context persistence

Categories of Coding Tools

IDEs

Cursor, Antigravity, VS Code

Cursor has best agent mode, Antigravity good with Gemini

Terminals

Warp, Ghostty

Warp supports agentic features.

Agents

Claude Code, Codex, Gemini CLI

Terminal-based AI agents.

Why Warp for AI Agents

Built-in AI Agent Mode
Natural language detection, seamless delegation

Modern UX
Block-based navigation, IDE-like editing, smart auto-complete

Cross-Platform
PowerShell, Git Bash, WSL support

Warp Drive
Share runbooks, workflows, commands across teams

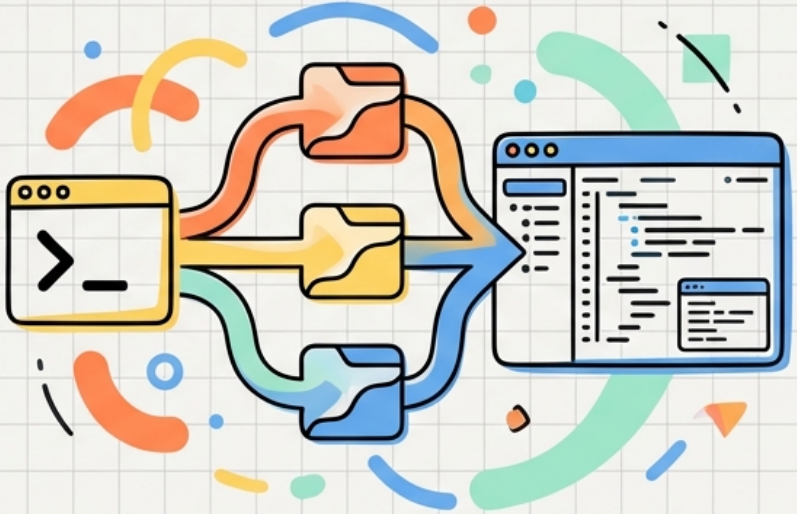
Performance
Built in Rust, faster than traditional terminals

Recommended AI-Native Coding Setup

1. Terminal (Warp)
Primary interface, tab management

2. Multiple Agent Tabs
Dedicate each tab to an AI agent

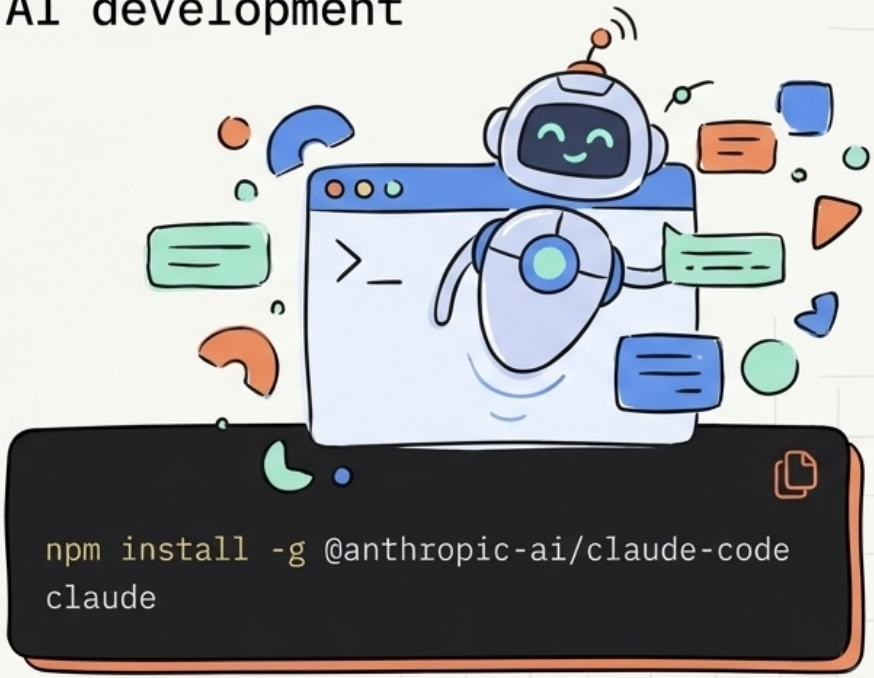
3. IDE for Review
Cursor/VS Code for final review, debugging, edits



This integrates AI agents directly into your terminal workflow.

Claude Code Overview

- CLI tool for terminal-based AI development
- Full codebase context
- Agentic capabilities
- MCP server support
- 200k token context window



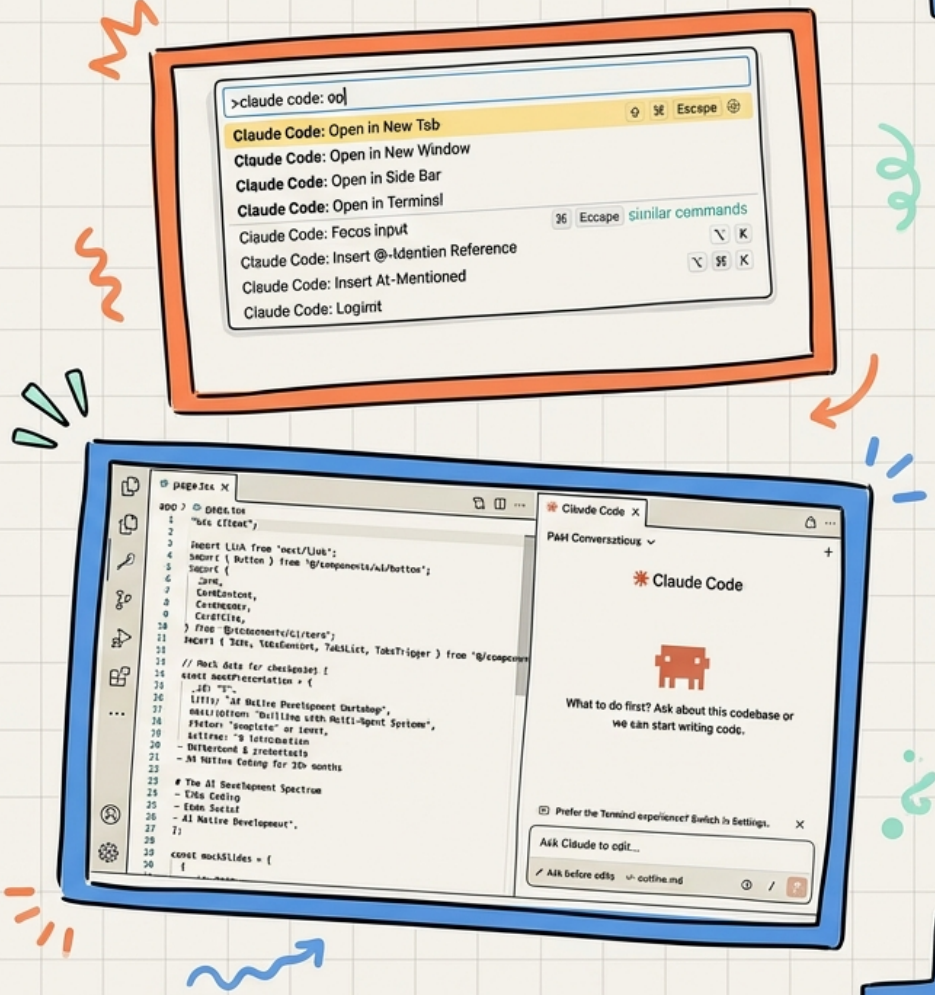
Accessing Claude Code

Three ways to use Claude Code:

1. **Terminal of your choice**
   Run 'claude' in any terminal (Warp, iTerm, etc.)

2. **VS Code / Cursor Extension**
   Integrated IDE experience

3. **Claude Code App**
   Desktop app for local or remote access via GitHub



How Agents Work

Agents work through your code like a developer:

1. 

Search & grep
files and folders

2. 

Read documentation
and architectural rules

3. 

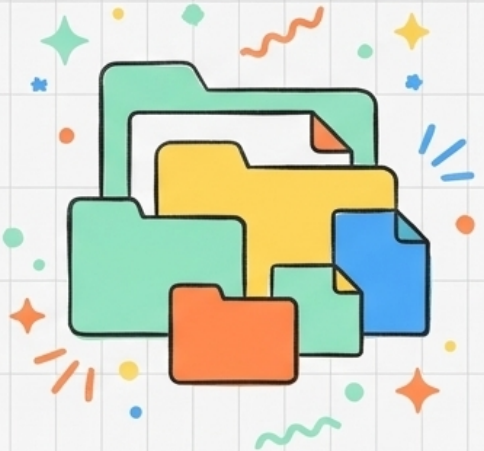
Access tools
to execute tasks

Organize Code for Agents

Clear File Naming - Descriptive, consistent naming conventions

Logical Folder Structure - Group related functionality together

Context Tags - Tag important files/folders for quick identification



Good organization helps agents understand, modify, and extend your code.

The DOCS Folder Pattern

Create a docs/ folder to capture information agents can load:

```
docs/
├── rules/      # Reusable patterns, integrations (e.g., Clerk + Convex)
├── features/   # spec.md and progress.md per feature
└── learnings/  # Lessons discovered while solving problems
```

This gives agents reliable cotext, just like docunentation for engineers.

Plan Mode vs Agent Mode

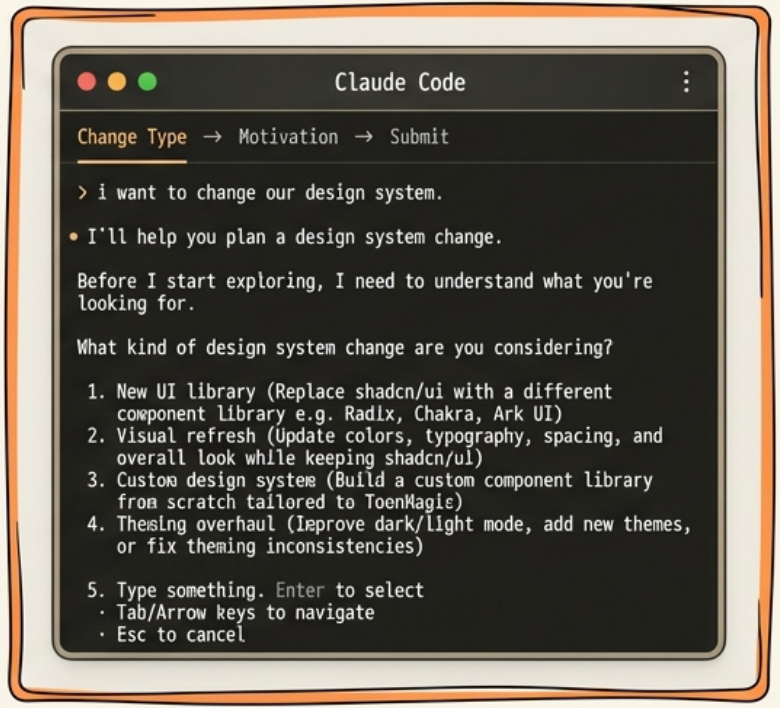
Plan Mode

- Think first, execute later
- Review before changes
- Complex/risky changes

Agent Mode

- Autonomous execution
- Changes happen in real-time
- Well-defined tasks

Shift + Tab to switch modes.
When unsure → Start with Plan Mode.



Environment Setup with Agents

Let agents handle the **tedious setup** work

Project Bootstrapping

Agents scaffold new projects automatically
React, Next.js, Expo, and more

Dependency Management

Install deps, resolve version conflicts
No more dependency hell

Automated Troubleshooting

Reads error logs, fixes PATH issues
Faster than manual debugging

Testing Setup

Jest, Playwright, Vitest from scratch
Run tests and interpret failures

Managing Context with CLAUDE.md

claude.md serves as your project guide - conventions, scripts, entry points, do's and don'ts.

- **"/init command"** - Generate claude.md automatically.
- **"Subfolder claude.md"** - Root for global, subfolders for tailored instructions.
- **"Dynamic Rule Loading"** - Reference docs/rules/* and docs/features/*.
- **"# [remember this]"** - Tell Claude to remember information.

Don't write claude.md yourself - let Claude Code generate and maintain it

Context Window Management

The agent operates within 200k tokens. Manage it:

/clear - Reset context when looping or history is polluted

/compact - Summarize and prune history, keep continuity

Subagents - Isolated contexts reduce token pressure

Continue Sessions - Resume via -c flag or Recent menu

Recovering from Mistakes

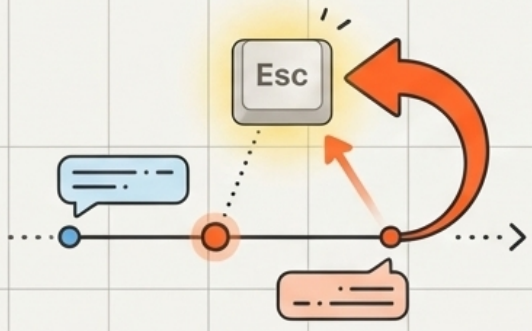
When the agent goes off track, you can rewind

Esc Esc

Double-tap Escape

- Rewind conversation to a previous point

Clears messages, but doesn't undo file changes (use git)



Other useful shortcuts:

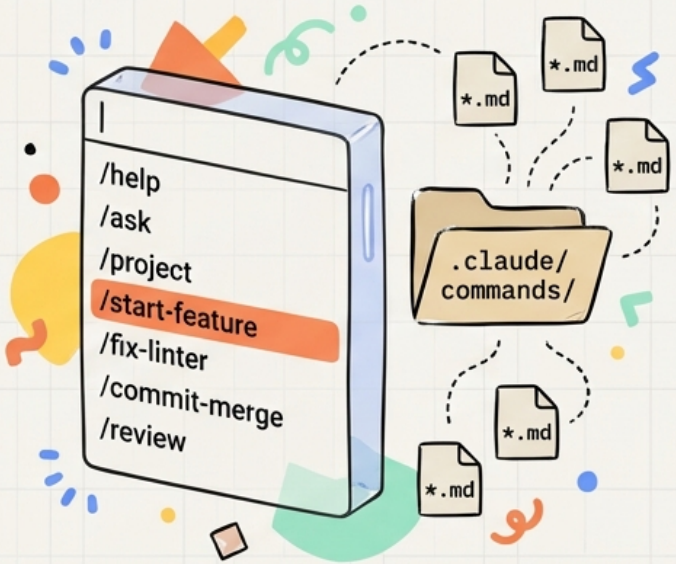
Ctrl+C	Cancel generation
Ctrl+L	Clear screen
Up/Down	Previous inputs

"Don't let a bad response derail your session - just rewind and try again"

Custom Commands (Slash Commands)

Store commands in `.claude/commands/*.md`:

/start-feature	Initialize feature branch, boilerplate
/fix-linter	Auto-fix linting errors
/commit-merge	Stage, commit, merge with conflict resolution
/review	Code review workflow



Ask Claude Code to create these commands for you.

Asking the Model to Think

When you need deeper reasoning, add cues:

think	think more	ultrathink
Normal depth reasoning	Slower, more detailed steps	Maximum reasoning, high token cost

Use for design decisions, migrations, or root-cause analysis.

Subagents & Task Delegation

Specialized subagents for different parts of the system:

Frontend Agent

UI/UX, styling, design systems

Backend Agent

APIs, databases, server logic

QA Agent

Testing, quality assurance, bug detection

```
Use the frontend-agent to review my UI changes
```

Configs stored in `.claude/agents/`

Subagent Example: Extract Pattern

I built an extract-pattern subagent to search GitHub repos for implementation patterns.

The problem:

Searching repos dumps raw results into main context, blowing up tokens

Use cases:

- "How did we implement auth in project X?"
- "Find our Stripe checkout integration pattern"
- "Onboarding new team members to existing patterns"

Why a subagent?

Isolated context

Heavy search operations happen in subagent

Token efficiency

~8x cleaner context vs raw search results

Specialized prompts

Optimized for code pattern extraction

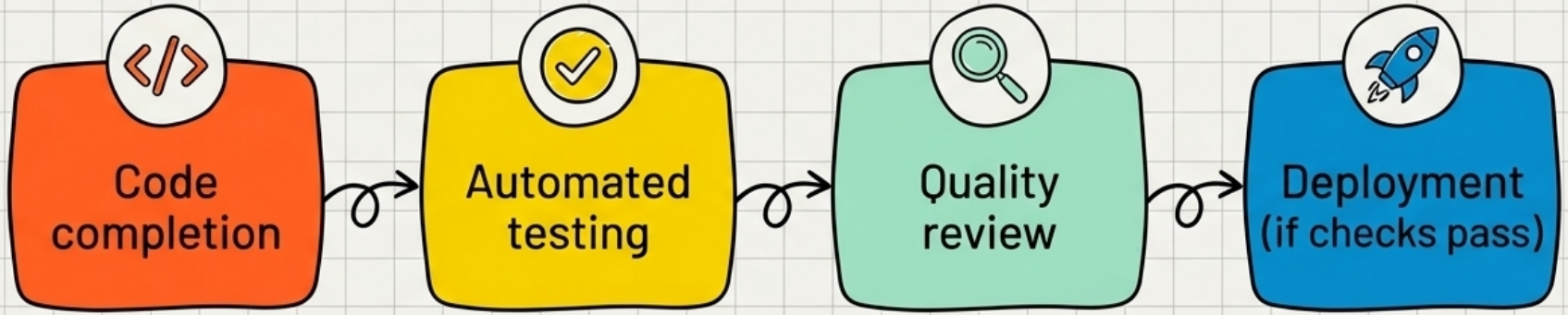
Cost savings

Can use lighter models (Sonnet) for research

github.com/hypersocialinc/claude-code-agents

Hooks for Automation

Hooks trigger follow-up actions when tasks complete:



Examples: run tests, send notifications, auto-review code.

Configure via **/hooks** or in **.claude/settings.json**

Skills: Progressive Context Loading

Skills are markdown files that teach Claude domain-specific knowledge:

Where to add:

- Store in **.claude/skills/** folder
- Structure: markdown with description, instructions, and optional resources

When to use Skills:

- Complex multi-step workflows
- Domain knowledge too extensive for CLAUDE.md
- Team conventions and best practices
- Tool-specific patterns

Skills can include:

- Step-by-step instructions
- Code examples and templates
- Links to external resources/docs
- Tool configurations

Skills vs Commands: Commands trigger actions; Skills provide knowledge

Skills vs Subagents: Subagents are isolated workers; Skills are shared context

Skills load progressively - Claude reads them when relevant, not all upfront

Skills Example: Music Production App

Building a music app with Strudel required extensive domain knowledge:

The problem: Too much context would blow up the context window

- Strudel syntax and API
- Music theory fundamentals
- Genre-specific production styles (house, techno, ambient, etc.)

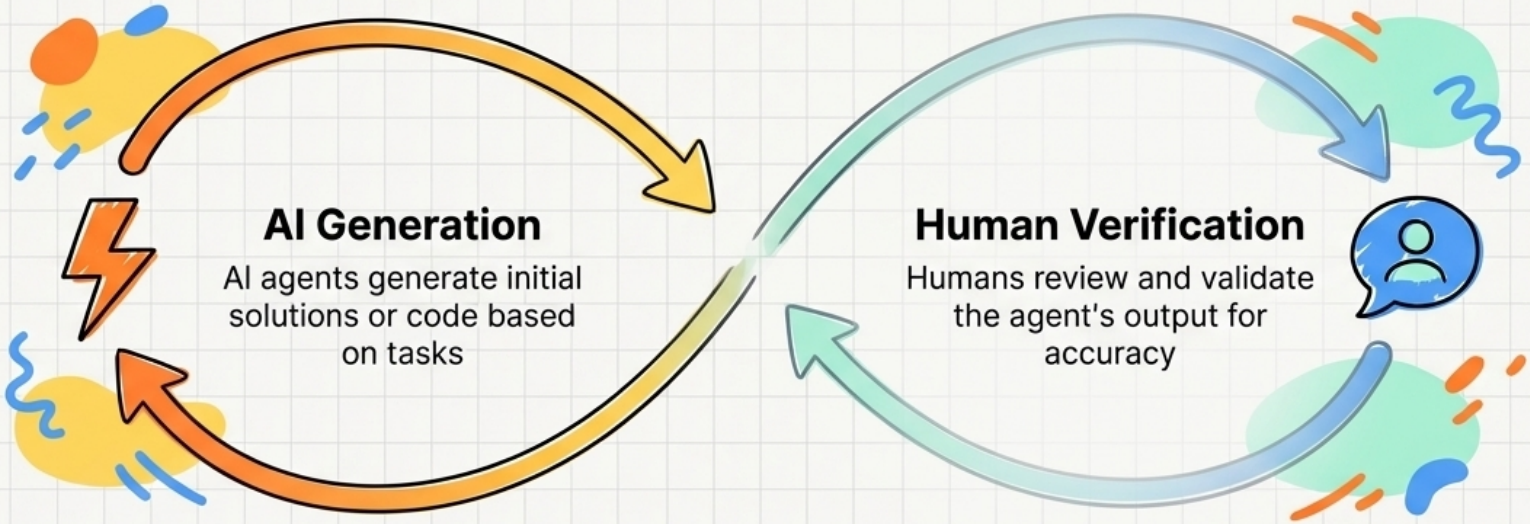
The solution: Organized skills by topic

```
.claude/skills/
├── strudel-syntax.md # Core Strudel patterns & API
├── music-theory.md # Scales, chords, progressions
└── genres/
    ├── house.md # House production techniques
    ├── techno.md # Techno patterns & sounds
    └── ambient.md # Ambient textures & pads
```

Result: Agent picks up only relevant context based on current task

The AI-Native Development Loop

In AI-native development, agents iterate on tasks, guided by human review and feedback. This cycle refines agent performance over time.



Your goal: take yourself out of the loop as much as possible

Working with External Documentation

How to give agents the docs they need

Copy & Paste Markdown

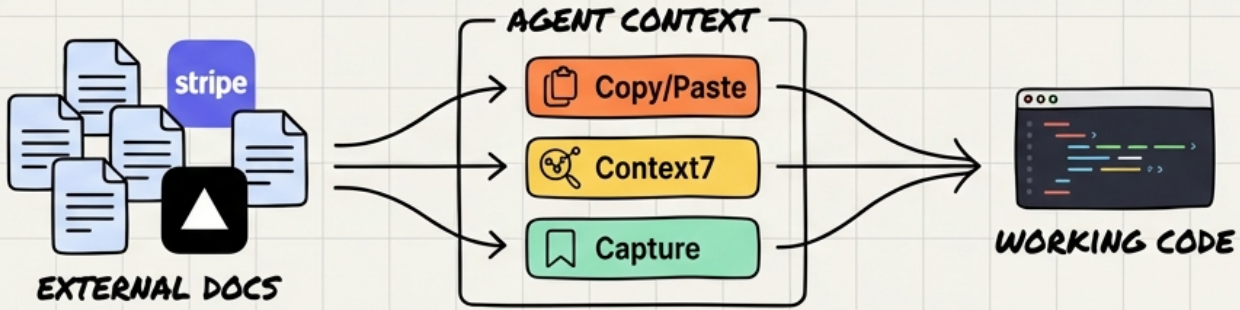
- Many tools have "Copy as MD" buttons
- Paste entire doc pages into conversation
- Quick context without leaving flow

Use Context7 MCP

- Semantic search across library docs
- Agent looks up docs on demand
- Works with popular frameworks

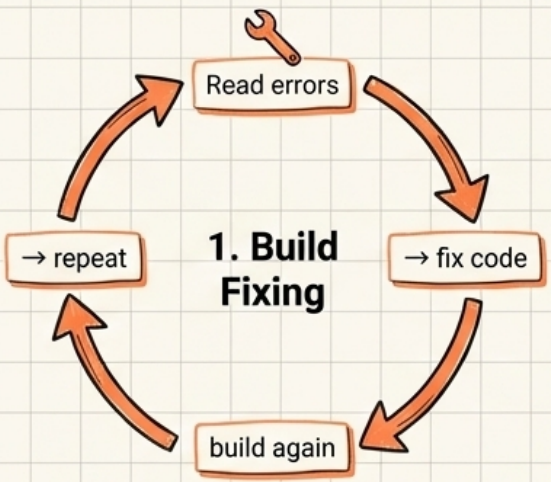
Capture What Works

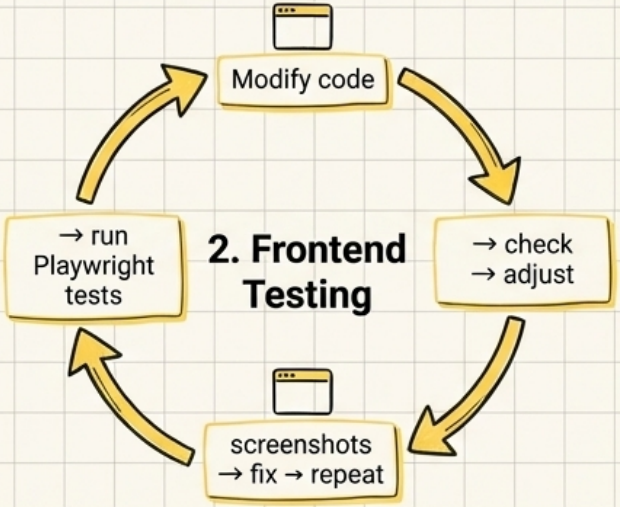
- Use `#[remember this]` for patterns
- Save to `docs/rules/` for reuse
- Build your own "how we do X" library

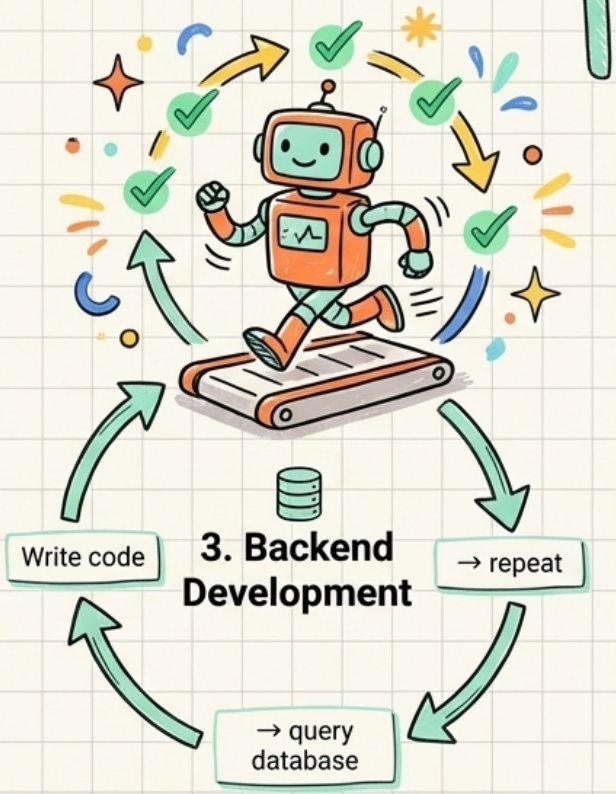


Looping Until Done

Get agents to loop until task completion:







Combine with MCPs for QA or databases for continuous testing.

What To Do When Stuck

Reset Context

`/clear` and try again fresh

Switch Agents

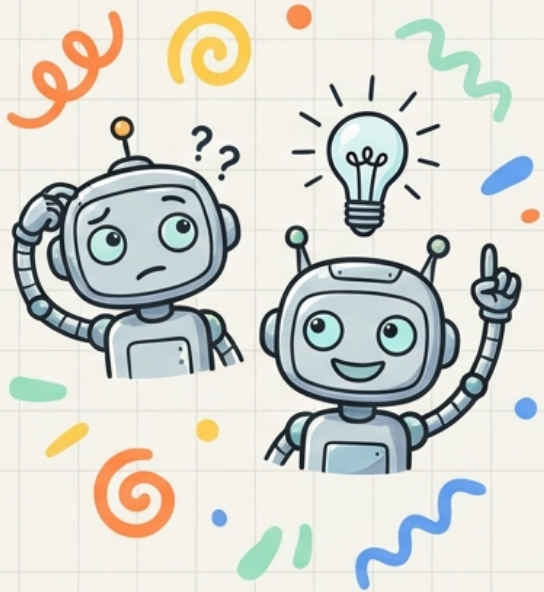
Move from Claude to Codex or vice versa

Consult Other Models

Use Zen MCP for second opinions

Ask to THINK MORE

Explicitly request deeper reasoning



REFACTORING WITH AI AGENTS

Big refactors become practical with AI agents as copilots

Systematic Code Changes

- Large-scale transformations
- API updates, class → function components

Legacy Cleanup

- Update deps, patterns, error handling
- Modernize older codebases

Database Migrations

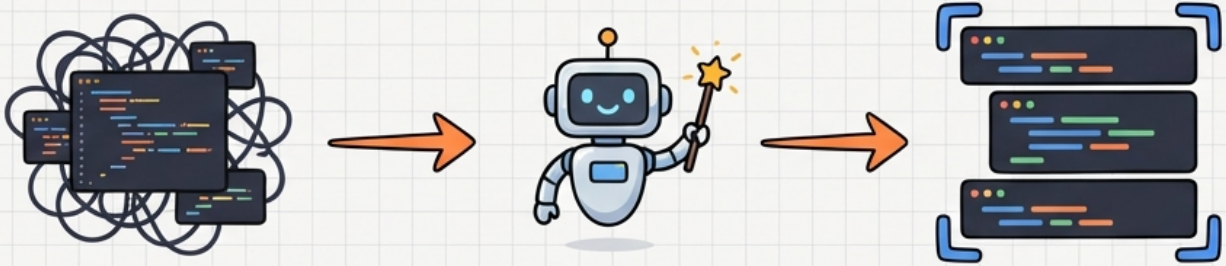
- Complex migration scripts
- Schema changes with data integrity

Why agents excel here:

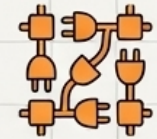
Don't get bored or make typos

Hold full context of changes

Review the diff, not every line



Popular MCPs



Playwright
Browser automation, QA, screenshots

Context7
Semantic search across code & docs

Figma
Design file access, component specs

Supabase/Convex
Database queries, schema inspection

GitHub
Repo management, PR reviews, issues

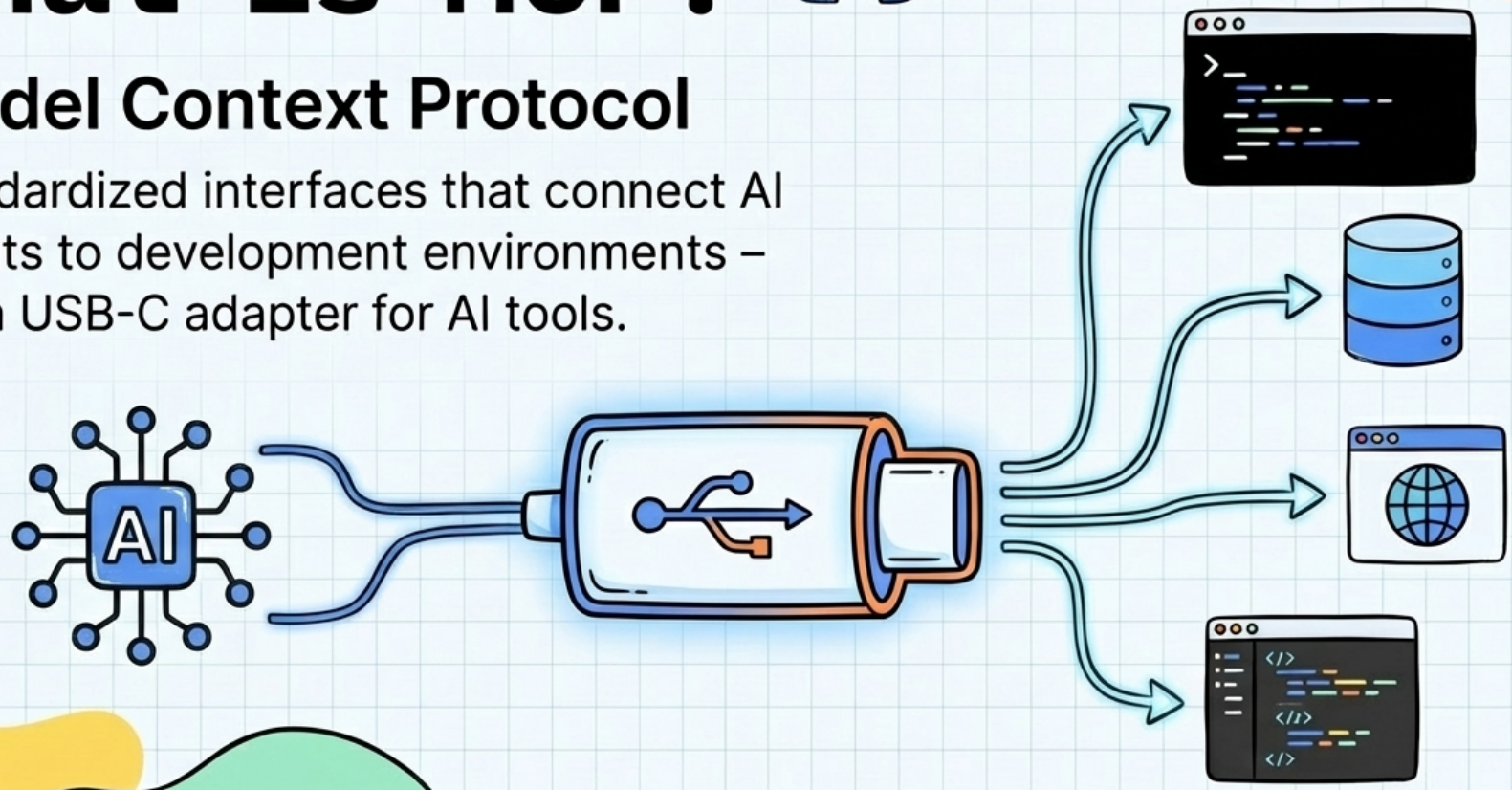
Zen
Consult other AI models

🔗 Find more at MCP Registry or @modelcontextprotocol/server-*

What is MCP?

Model Context Protocol

Standardized interfaces that connect AI agents to development environments – like a USB-C adapter for AI tools.



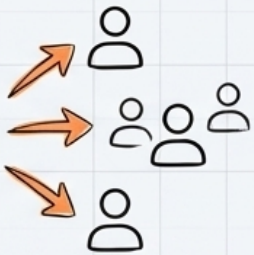
MCP Configuration

```
claude mcp add server-name --scope project
claude mcp list
```

Scope Levels:

Project

.mcp.json at repo root, shared via git



Team Members

Global

~/.config/claude/mcp.json, personal utilities

Single User

Tip: Project for team tools, Global for personal tools

Browser MCP in Claude Code

Give Claude Code the ability to browse and debug web apps

Chrome DevTools MCP

Official from Google

- Direct Chrome DevTools access
- Debugging & performance
- Network inspection
- Real-time insights

Browser MCP

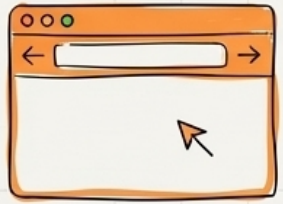
browsermcp.io

- Chrome extension
- Uses your real browser profile
- Navigate, click, type, fill forms
- Take screenshots

Use cases:

- Test your app
- Debug UI issues
- Automate web tasks
- Verify deployments

Native Browser in Cursor & Antigravity



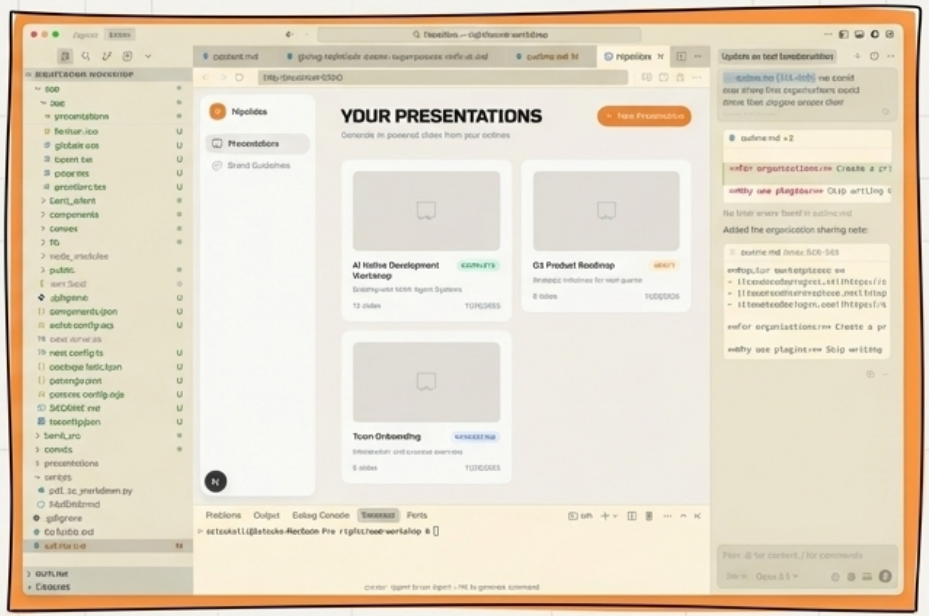
Both IDEs have built-in browser capabilities - no MCP installation needed

Cursor

- Native browser tool for agents
- Take screenshots, interact with your running app
- Visual regression testing built-in
- Works across web, desktop, and mobile

Antigravity

- Built-in browser preview and extension
- Agents interact directly with Chrome
- Automated UI testing and visual debugging
- Seamless integration with Gemini



No extra setup – browser automation works out of the box

Claude Code Plugins & Marketplaces

Extend Claude Code with community plugins - commands, agents, MCP servers, and hooks

```
Adding a marketplace:
/plugin marketplace add jeremylongshore/claude-code-plugins
```

```
Installing a plugin:
/plugin install [plugin_name]@[marketplace_name]
```

Popular marketplaces:

claudecodeplugins.io
243+ production-ready plugins

claudecodemarketplace.net
Community submissions

claudecodeplugin.com
Curated directory

For organizations: Create a private GitHub repo with plugins, share across teams

Why plugins: Skip boilerplate, leverage community best practices

Remote & Async Coding

Work on code without your dev machine

Claude Code

Web & iOS App

- Access via claude.ai or iOS app
- Connect GitHub repos
- Runs in Anthropic's cloud
- Creates PRs to review later

OpenAI Codex

OpenAI Codex

- Async workflow via GitHub
- Queue tasks, review later
- Background execution

Use cases:

- Fix bugs while commuting
- Kick off features overnight
- Iterate from anywhere

Working on Multiple Features in Parallel

While agents loop (2-15 min), maximize your time:

1. Multiple Agent Tabs

Separate tabs per feature (risk: file conflicts)

2. Separate Projects

Different repos entirely

3. Git Worktrees

Replicate project to branch, merge when ready

Conductor.build - Automates worktrees, parallel agents, AI-assisted merging

Multi-Agent Code Reviews

Agents review each other before the team sees the PR

Tools:

/review command

CodeRabbit - AI PR reviews

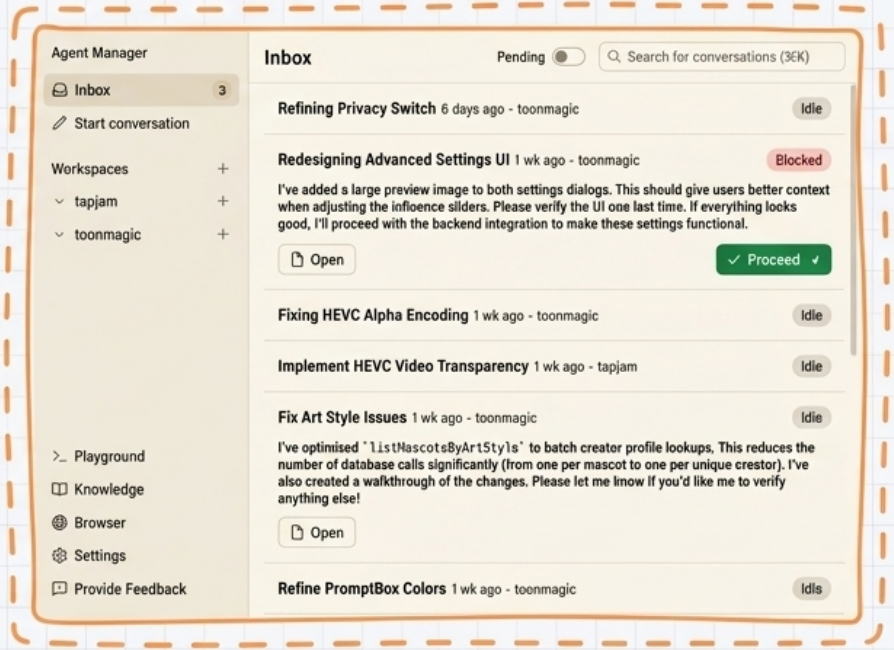
Benefits:

- Catch issues early
- Different perspectives
- Faster iterations
- Cleaner PRs

ANTIGRAVITY

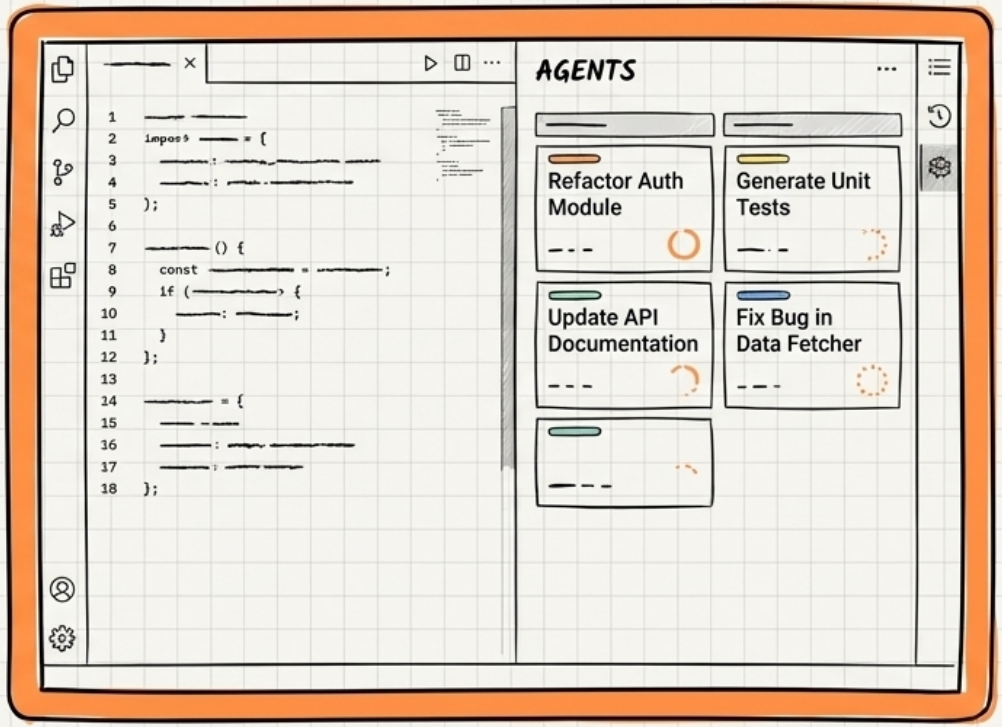
Google's VS Code fork with agent-first design

- Unified inbox for all agent tasks
- Agents work across workspaces
- Get notified when agents finish or need input
- Deep Gemini 3 Pro integration
- Browser subagent for visual testing



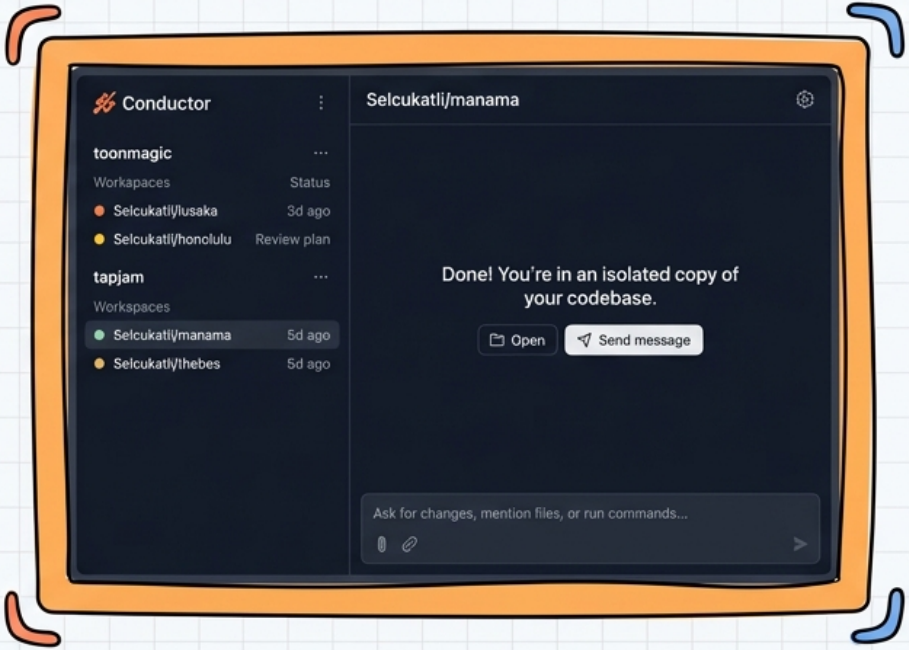
Cursor Agent Mode

- Switch between Agents and Editor mode
- Multiple parallel threads
- Background agents work while you code
- Monitor progress in real-time



Conductor.build

- Spins up isolated git workspaces in the cloud
- Each agent gets its own branch
- Full codebase access, no conflicts
- Review diffs, merge when ready
- Think CI/CD for AI agents



Conductor Configuration

Define your workspace setup in `conductor.json` at repo root

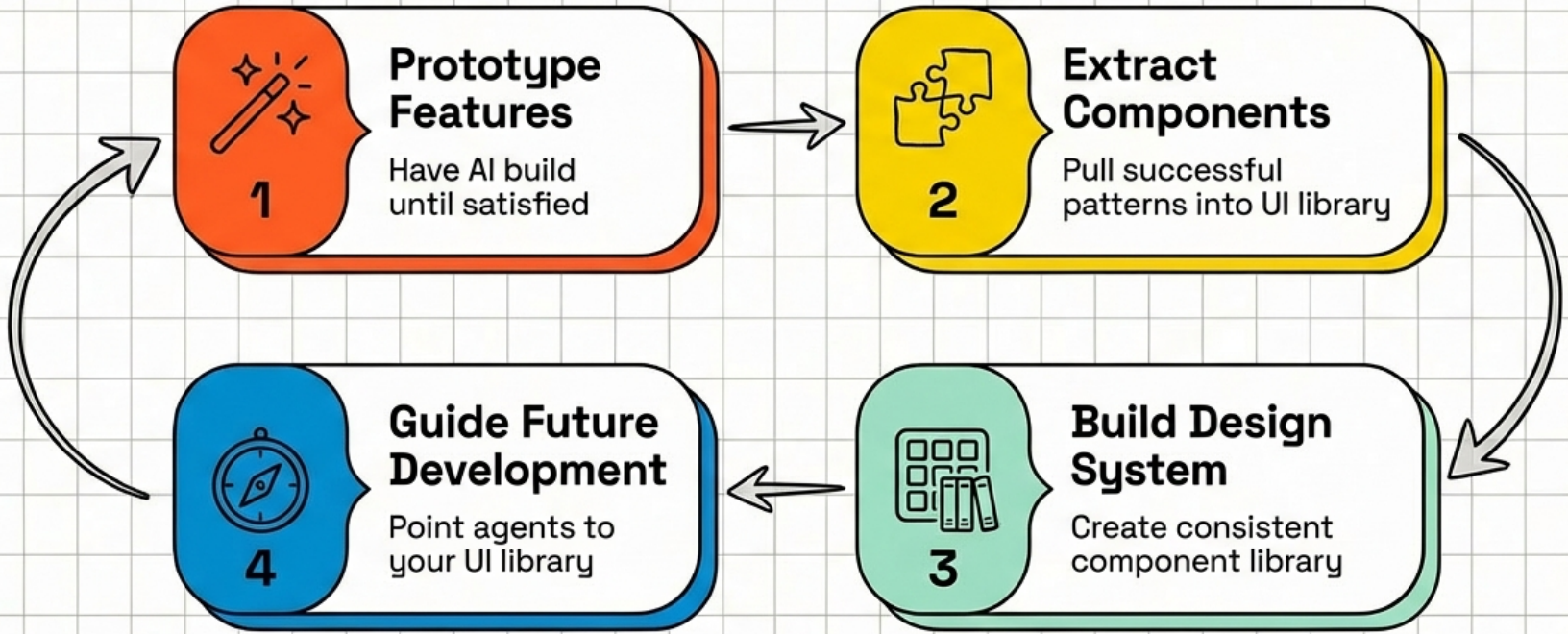
```
{
  "scripts": {
    "setup": "cp ../../.env.local .env.local && ...",
    "run": "npm run dev"
  }
}
```

"setup"
Runs when workspace is created
Copy env vars, install deps, init services

"run"
Starts your dev server
In the isolated workspace

Conductor reads this automatically - agents get fully configured environments

Designing with AI Agents



UI Library Foundations

Radix UI

Low-level, unstyled, accessible components

BUTTON

shadcn/ui

Customizable, copy-paste components

Headless UI

Unstyled, works with Tailwind

Tamagui

Universal kit for React Native + web

STAYING IN THE FLOW

Minimize context switching to maximize productivity

Use your voice and visuals to communicate with agents:

Speech-to-text

Talk to your agent instead of typing

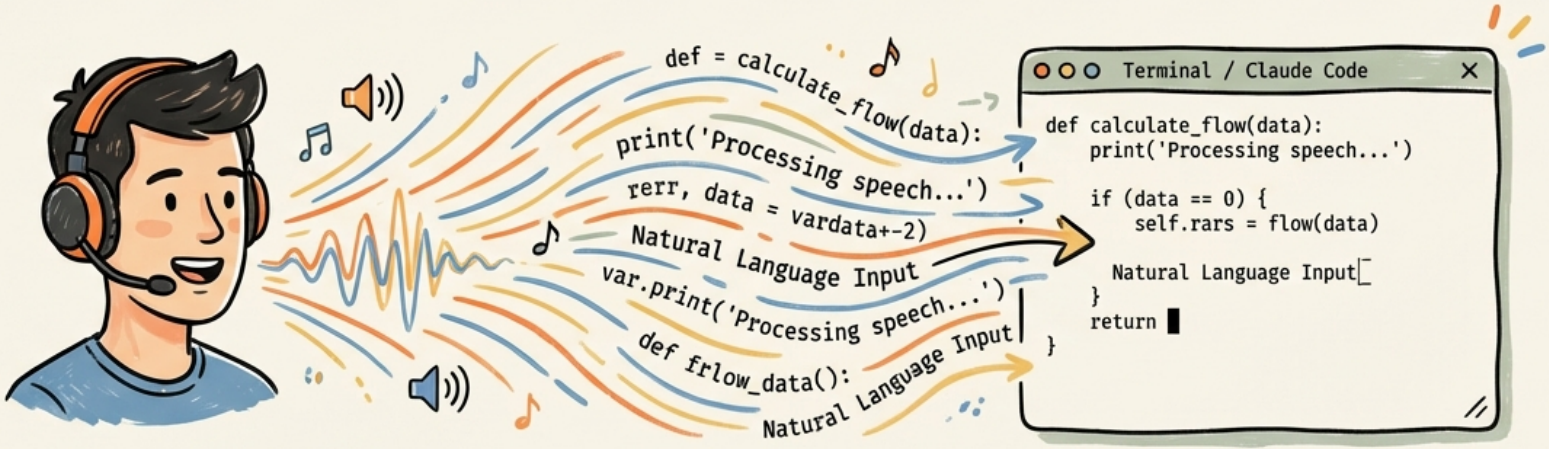
Visual markup

Show the agent what you mean with screenshots

Keep your hands on the keyboard, your eyes on the code, and let the agent understand you faster

Speech-to-Text for Flow

Wispr Flow – Talk to your agent instead of typing



Understands code

Works everywhere

Runs locally

wisprflow.ai

Visual Feedback for Agents

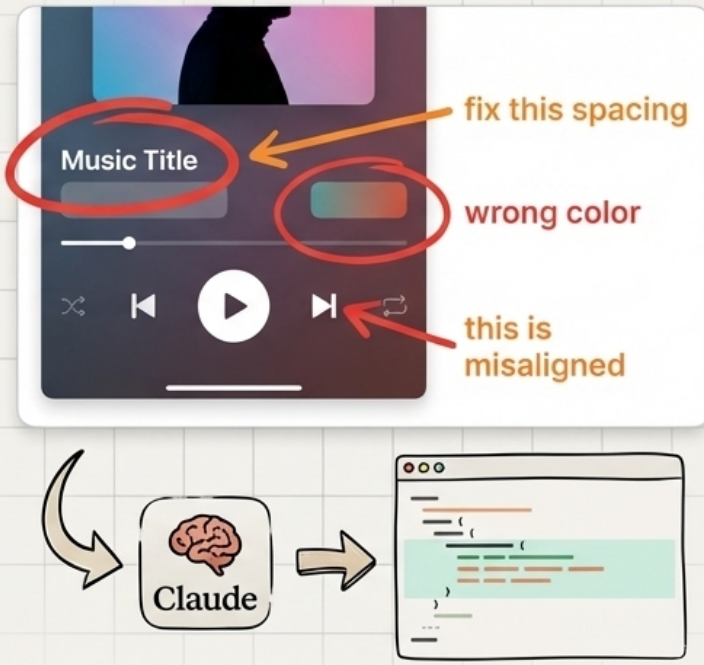
Give agents visual context by sharing screenshots and markups

Take a screenshot of the issue
UI bug, design reference, error

Mark it up with annotations
Arrows, circles, text

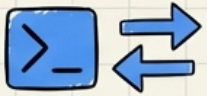
Copy to clipboard and paste into Claude Code

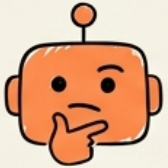
VISUAL EXAMPLE



Tools: Shottr, CleanShot X, or any screenshot tool with markup

Comparing Terminal Agents





Claude Code

Most feature-rich

- Subagents, commands, plugins, MCP

200k tokens

Codex

GitHub integration

- Background tasks, async workflows

Varies by model

Gemini CLI

Massive context

- Multimodal, Google ecosystem

1-2M tokens

Tip: Don’t rely on one agent. Switch when stuck.

Suggested Models based on Use Case

Claude Opus 4.5
Daily driver

Gemini 3.0 Pro
Great for UI/UX

Cursor Composer 1
Quick easy tasks

Codex 5.1
Code Reviews

Why Agents Love Monorepos

Co-locating code makes agents more effective

Full visibility

Agent sees frontend, backend, and shared code together

Type safety across boundaries

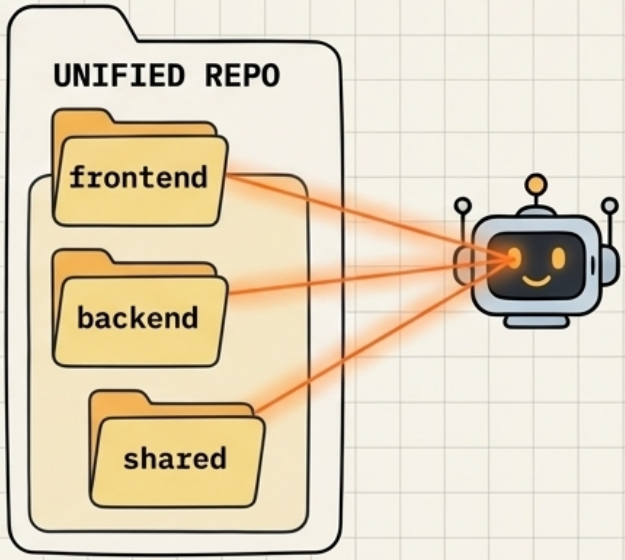
Shared types prevent agent mistakes

Single context

No jumping between repos, stays in context window

Easier refactoring

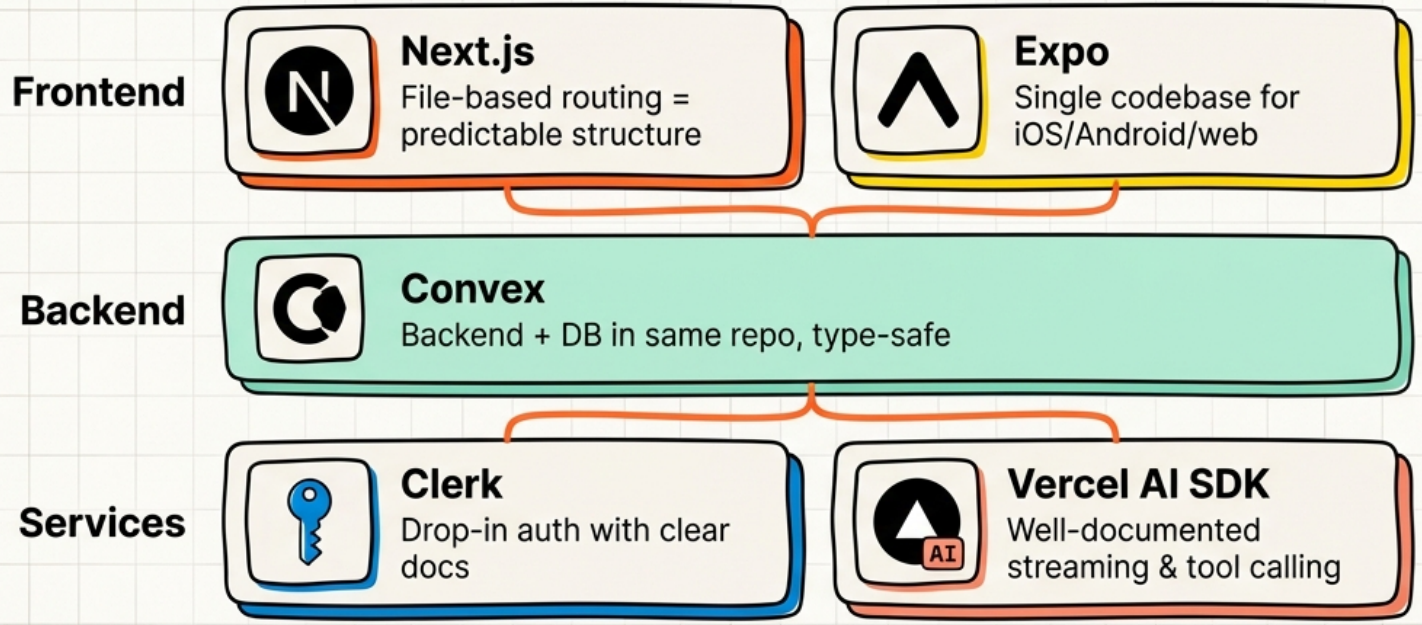
Agent can update all references in one pass



Not saying microservices are bad - just that agents work better with unified codebases

Our Stack (And Why It's Agent-Friendly)

Tools we use that work great with AI agents



Pick tools with clear conventions and good docs

Best Practices Summary

RECAP

- Build small, testable features
- Start in plan mode for complex tasks
- Document specs & progress in .md files
- Commit frequently with /commit
- Keep feature scope small
- Review at checkpoints, not just at the end

summary

KEY TAKEAWAYS

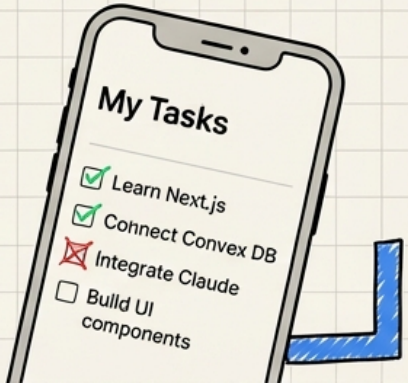
1. **Context is everything**
   CLAUDE.md, skills, and docs folders make or break agent effectiveness
2. **Get out of the loop**
   Set up agents to iterate autonomously, review at checkpoints
3. **Use multiple agents**
   Switch when stuck, use different agents for different tasks
4. **Stay in flow**
   Voice input, visual markup, minimize context switching
5. **Organize for agents**
   Monorepos, clear structure, type-safe boundaries

Live Demo: Building a Todo App

Next.js | Convex | Claude Code

What we'll cover:

1. Set up `CLAUDE.md` and `docs/` structure
2. Plan mode → Agent mode workflow
3. Context7 MCP for docs lookup
4. Looping until build passes
5. Multi-agent code review
6. Cursor agent view - parallel threads
7. Conductor.build - isolated git workspaces



RESOURCES

Agents & Tools

- Claude Code Docs
- Cursor
- Warp
- Codex

Productivity

- Wispr Flow
- Code
- Conductor.build
- Shottr

MCP & Integrations

- MCP Registry
- Browser MCP
- Context7
- Chrome DevTools MCP

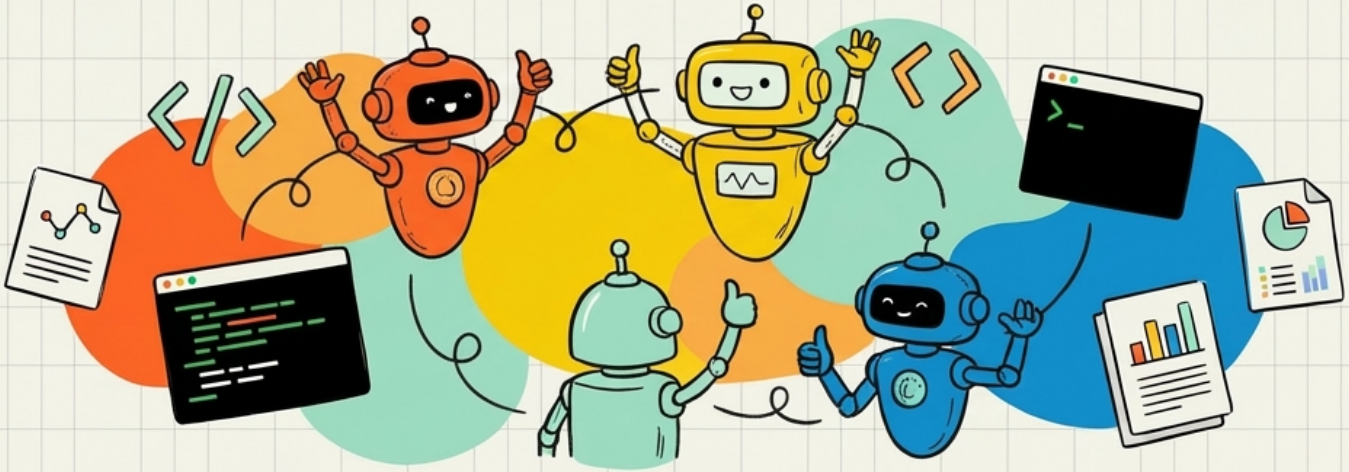
My Resources

- github.com/hypersocialinc/claude-code-agents

Plugins & Extensions

- claudecodeplugins.io
- CodeRabbit

Thank You!



Selcuk Atli
selcuk.atli@gmail.com