TEAM 400

TRAINING SESSION

Claude Code

A hands-on tour — from installing and running it to project memory, skills, sub-agents, and connecting your tools.

From setup to advanced · Hands-on training

AGENDA

What we'll cover

01

Setup

Plans, install, and your first run.

02

Everyday use

Drive it well: the loop, control, and prompting.

03

CLAUDE.md

Give Claude lasting memory of your project.

04

Skills

Reusable know-how it loads automatically.

05

Sub-agents

Hand focused work to a specialist.

06

MCP

Connect the tools your team already uses.

SECTION 01

Setup

Everything to get Claude Code ready — your plan, the install, your account, and a first run.

SETUP · PLANS

Free, Pro, or Max

Individual plans · claude.com/pricing

Free

For trying Claude out

$0

- ✓ Chat on web & desktop
- ✓ Generate code & search the web
- ✗ No Claude Code

INCLUDES CLAUDE CODE

Pro

For everyday productivity

$17 /mo annual · $20 monthly

- ✓ Everything in Free
- ✓ **Includes Claude Code**
- ✓ More usage & more models

Max

For power users

$100 /mo and up

- ✓ Everything in Pro
- ✓ 5× or 20× more usage
- ✓ Priority at busy times

Bottom line Claude Code is included from **Pro** and up. The Free plan does not include it.

SETUP · INSTALL

Two ways in: terminal or desktop

Most people run it in the **terminal** — copy the one line for your system below. Prefer not to? Grab the **desktop app** instead.
Same Claude Code either way.

Terminal — install

macOS · Linux · WSL	`curl -fsSL https://claude.ai/install.sh | bash`
Windows PowerShell	`irm https://claude.ai/install.ps1 | iex`
Windows CMD	`curl -fsSL https://claude.ai/install.cmd -o install.cmd && install.cmd && del install.cmd`

Prefer a package manager? `brew install --cask claude-code` (Mac) `winget install Anthropic.ClaudeCode` (Windows)

Prefer the desktop app? Download it at `claude.com/download` · also on web, VS Code & JetBrains

SETUP · CONNECT

Three ways to connect

The first time you run `claude`, it asks how you want to sign in:

01

USE THIS ONE

Claude subscription

Sign in with your Claude account — **Pro, Max, Team, or Enterprise** .

02

Anthropic Console

Pay as you go — billed on API usage instead of a subscription.

03

3rd-party platform

Through Amazon Bedrock, Microsoft Foundry, or Vertex AI.

SETUP · FIRST SESSION

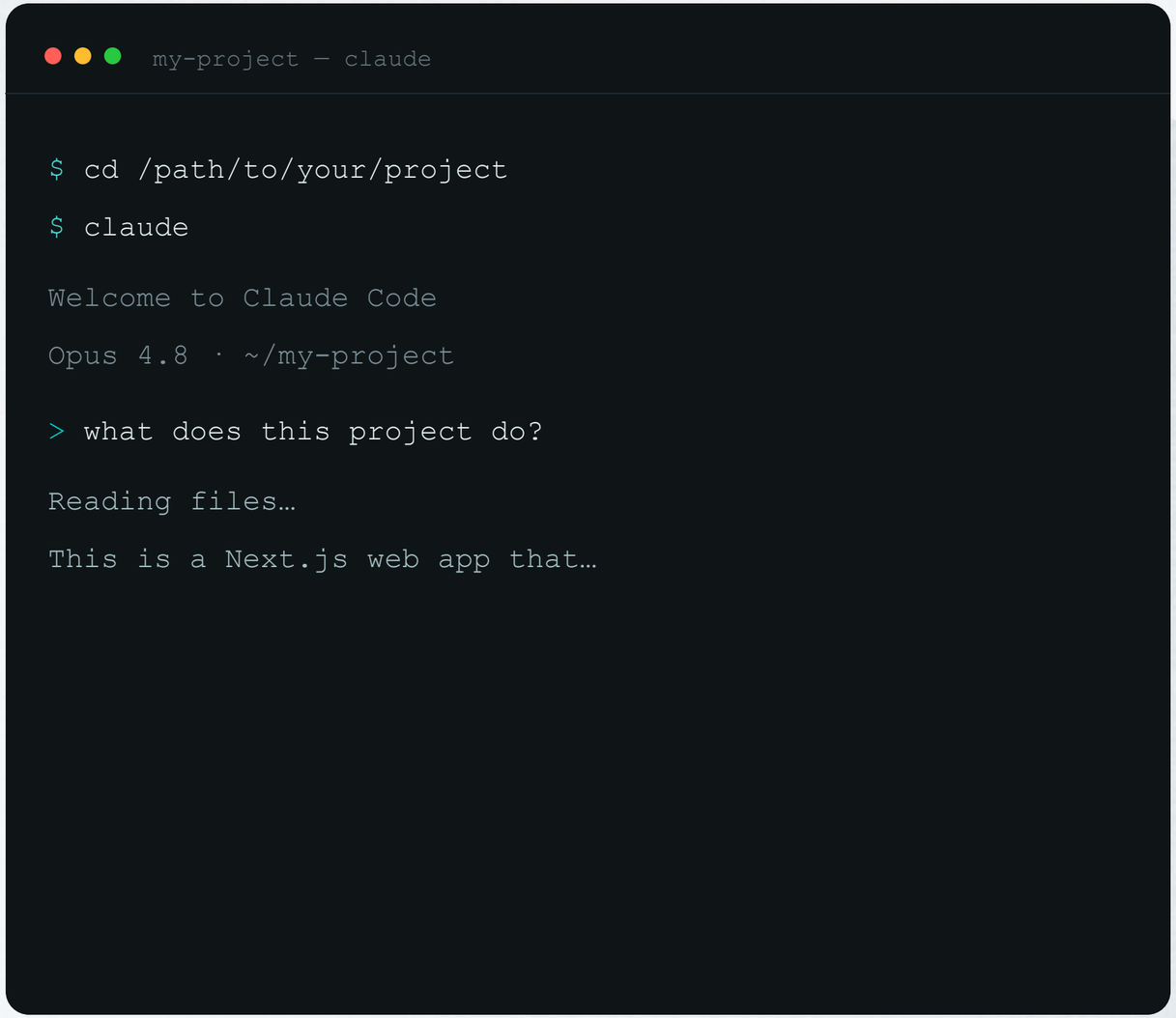
Your first session

1 **Open your project**
Move into the folder with `cd`

2 **Type `claude`**
Starts an interactive session

3 **Log in**
A browser opens the first time only

4 **Just ask**
Talk to it in plain English



SETUP · REFERENCE

Commands to know

In your terminal — to start

`claude`	Start an interactive session
`claude "task"`	Run a one-time task
`claude -p "query"`	Quick answer, then exit
`claude -c`	Continue your last conversation

Inside a session — type "/"

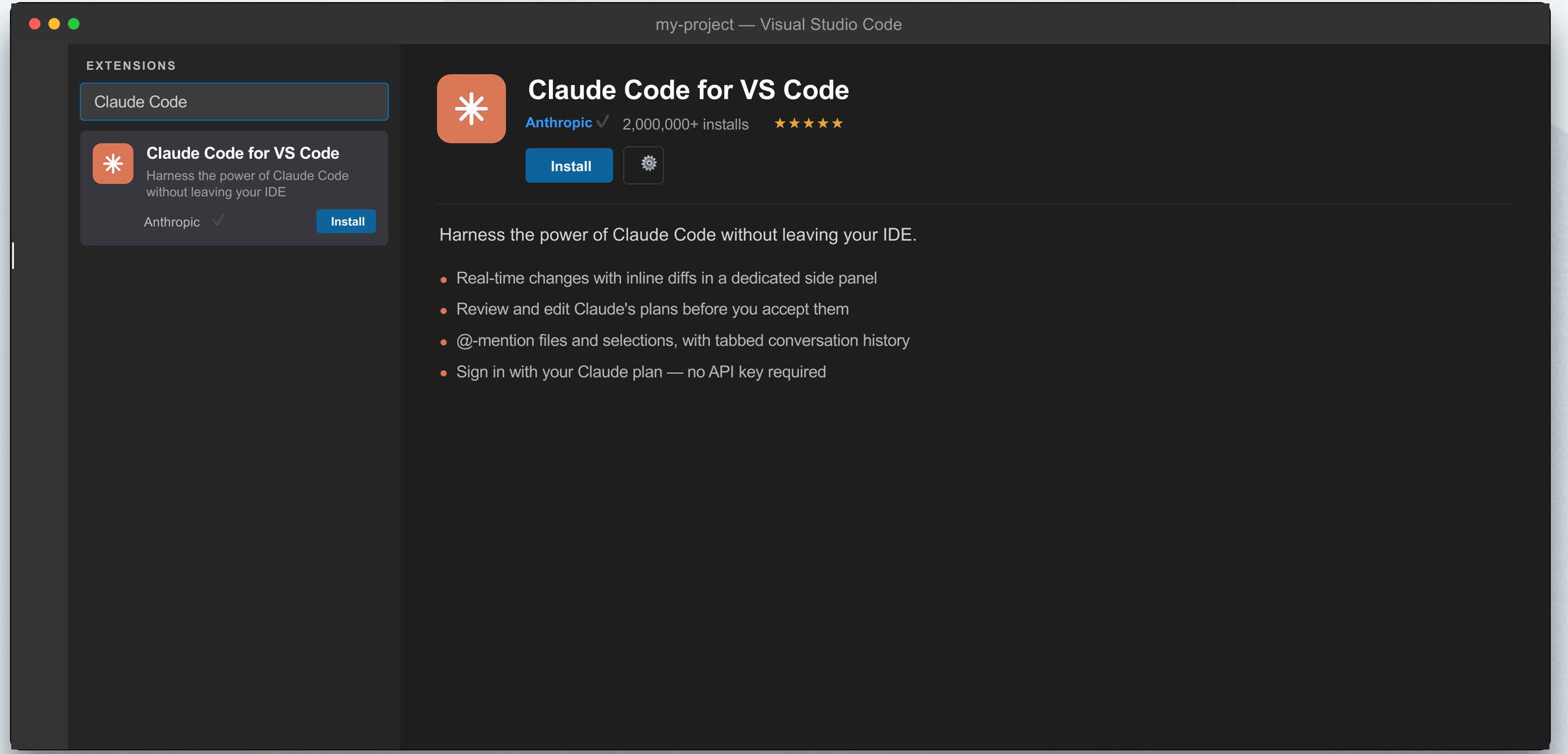
`/help`	Show all available commands
`/clear`	Clear the conversation history
`/login`	Switch or re-authenticate accounts
`/exit`	Leave Claude Code

Tip — inside a session, press `Tab` to autocomplete and `↑` for previous commands.

SETUP · VS CODE

Or use it inside VS Code

Open **Extensions**, search **"Claude Code"** , and click Install — the verified Anthropic publisher.



SECTION 02

Everyday use

Driving Claude Code well, day to day.

EVERYDAY · HOW IT WORKS

It works in a loop

01

Explore

Reads the files it needs — no manual setup.

→

02

Plan

Proposes what it'll change so you can steer first.

→

03

Edit

Makes the changes — asking permission first.

→

04

Verify

Runs tests or the build to check itself, then iterates.

AND YOU STAY IN CONTROL

Asks first

Shows every change and waits for your approval.

Accept-all mode

Trust a task? Let it run without stopping each edit.

Rewind anytime

Checkpoints undo with `/rewind` or double-Esc.

EVERYDAY · PROMPTING

Ask like you mean it

VAGUE

```
fix the bug
```

SPECIFIC

```
fix the login bug where users see a blank screen after
entering wrong credentials
```

+ Break big asks into numbered steps
+ Let it explore the code before it changes anything

STARTER PROMPTS TO STEAL

Understand code	`what does this project do?`
Fix a bug	`forms can submit empty — fix it`
Write tests	`write unit tests for the cart`
Refactor	`use async/await here, not callbacks`
Update docs	`update the README install steps`
Use git	`commit with a descriptive message`

SECTION 02 · LIVE DEMO

Fix a bug together

Watch the loop happen live — Claude finds the code, proposes a fix, and asks before changing anything.

```
> there's a bug where users can submit empty forms — find and fix it
```

SECTION 03

CLAUDE.md

Give Claude lasting memory of your project.

CLAUDE.MD · WHAT IT IS

Memory for your project

A plain Markdown file that sits in your project. Claude reads it **at the start of every session** — automatically.

All the context you'd otherwise repeat each time, you write once. Then Claude just knows it.

`CLAUDE.md` → loaded into every session

WHAT GOES IN IT

- What the project is & its stack
- How to build, run & test it
- Code conventions & style
- Key commands & folders
- Hard rules — the do's and don'ts

CLAUDE.MD · SET IT UP

Where it lives — and how to make one

`./CLAUDE.md`

Project root

Shared with your whole team through the repo.

`~/.claude/CLAUDE.md`

Global

Your personal rules, applied to every project.

`/feature/CLAUDE.md`

Subfolder

Extra rules that only apply to one area.

Make one in seconds `/init` scans your project and drafts it — or just ask Claude to write one.

CLAUDE.MD · EXAMPLE

What a good one looks like

Short and clear beats long and exhaustive.

CLAUDE.md

```
# Project
Next.js storefront — TypeScript,
Tailwind, Postgres.

## Commands
- dev: npm run dev
- test: npm test
- build: npm run build

## Conventions
- Components in /src, PascalCase
- Use tokens.css — never hardcode
colors
- A test for every new util

## Don't
- Don't touch /legacy
- Don't commit straight to main
```

SECTION 03 · LIVE DEMO

Make a CLAUDE.md

Run it in a real project and watch Claude read the code and draft the file — then tweak one rule together.

```
> /init
```

SECTION 04

Skills

Reusable know-how Claude loads exactly when it's needed.

SKILLS · WHAT IT IS

Packaged know-how

A skill is a small folder of instructions that teaches Claude how to do **one task your way** — and produce a consistent result every time.

Claude loads the right skill **automatically** when your request matches. Some are built in; you can write your own.

`.claude/skills` — where your skills live

HOW IT KICKS IN

1. You make a request
2. Claude spots a matching skill
3. It loads those instructions
4. And does it your way

SKILLS · THREE KINDS

Three kinds of skills

Create output

Consistent, high-quality documents, decks, reports, and designs — every time.

- Frontend-design skill
- docx / pptx / xlsx generation
- Embedded style guides

Automate workflows

Multi-step processes that follow a set method, with validation at each stage.

- skill-creator (builds skills)
- Sprint planning & tasks
- Review & improvement loops

Enhance your tools

Teach Claude how to use your connected integrations effectively.

- Sentry code-review skill
- Jira / Linear management
- Figma → Linear → Slack

SECTION 04 · LIVE DEMO

Build a skill

Use the skill-creator to package something your team repeats — then watch it kick in automatically.

```
> use skill-creator to make a skill for our PR review checklist
```

SECTION 05

Sub-agents

Hand focused work to a specialist — with its own context and tools.

SUB-AGENTS · WHAT THEY ARE

A specialist for the job

A focused assistant Claude can hand a task to — with its **own context, own tools, and own instructions** .

It goes deep on its one job and reports back, while your main conversation stays clean.

WHY USE THEM

Focus
A tight brief and only the tools it needs.

Parallel
Run several at once on different parts.

Clean context
Keeps your main thread uncluttered.

SUB-AGENTS · SET ONE UP

Define a role, give it tools

Lives in `.claude/agents` — Claude delegates automatically, or ask by name.

.claude/agents/code-reviewer.md

```
name: code-reviewer
description: Reviews diffs for bugs, style & security. Use after changes.
tools: Read, Grep, Bash
---

You are a senior code reviewer. When invoked:
- Run git diff to see what changed
- Flag bugs, security issues, and style problems
- Be concise — suggest concrete fixes
```

SECTION 05 · LIVE DEMO

Delegate to an agent

Make a change, then hand the review off — and watch the main thread stay focused.

```
> have the code-reviewer check my latest changes
```

SECTION 06

MCP

Connect Claude Code to the tools your team already uses.

MCP · WHAT IT IS

A standard plug for your tools

MCP lets you plug external tools and data straight into Claude Code. Once connected, Claude can **read from them and act in them** as part of its work — no copy-paste.

GitHub
Issues & PRs

Sentry
Error triage

Jira / Linear
Tickets

Figma
Designs

Postgres
Your data

Slack
Messages

MCP · ADD A CONNECTOR

Plug one in

`claude mcp add` add a server

`/mcp` manage in a session

Local servers

Run on your machine — good for your own files, databases, and scripts.

Remote servers

Hosted and shared — the easiest way to give a whole team the same tools.

Trust matters Only connect servers you trust — they can read your data and take actions.

MCP · IN PRACTICE

One request, many tools

FIGMA

Read the design

Pull the spec for the new screen.

→

LINEAR

Create the tickets

Break the work into tasks.

→

SLACK

Post the summary

Tell the team it's queued up.

```
> turn the new Figma screen into Linear tickets and post a summary to #eng
```

SECTION 06 · LIVE DEMO

Connect a tool

Hook up one service live, then ask Claude to use it — for example, summarize the open pull requests.

```
> claude mcp add github → summarize our open PRs
```

SECTION 07

Prompt engineering

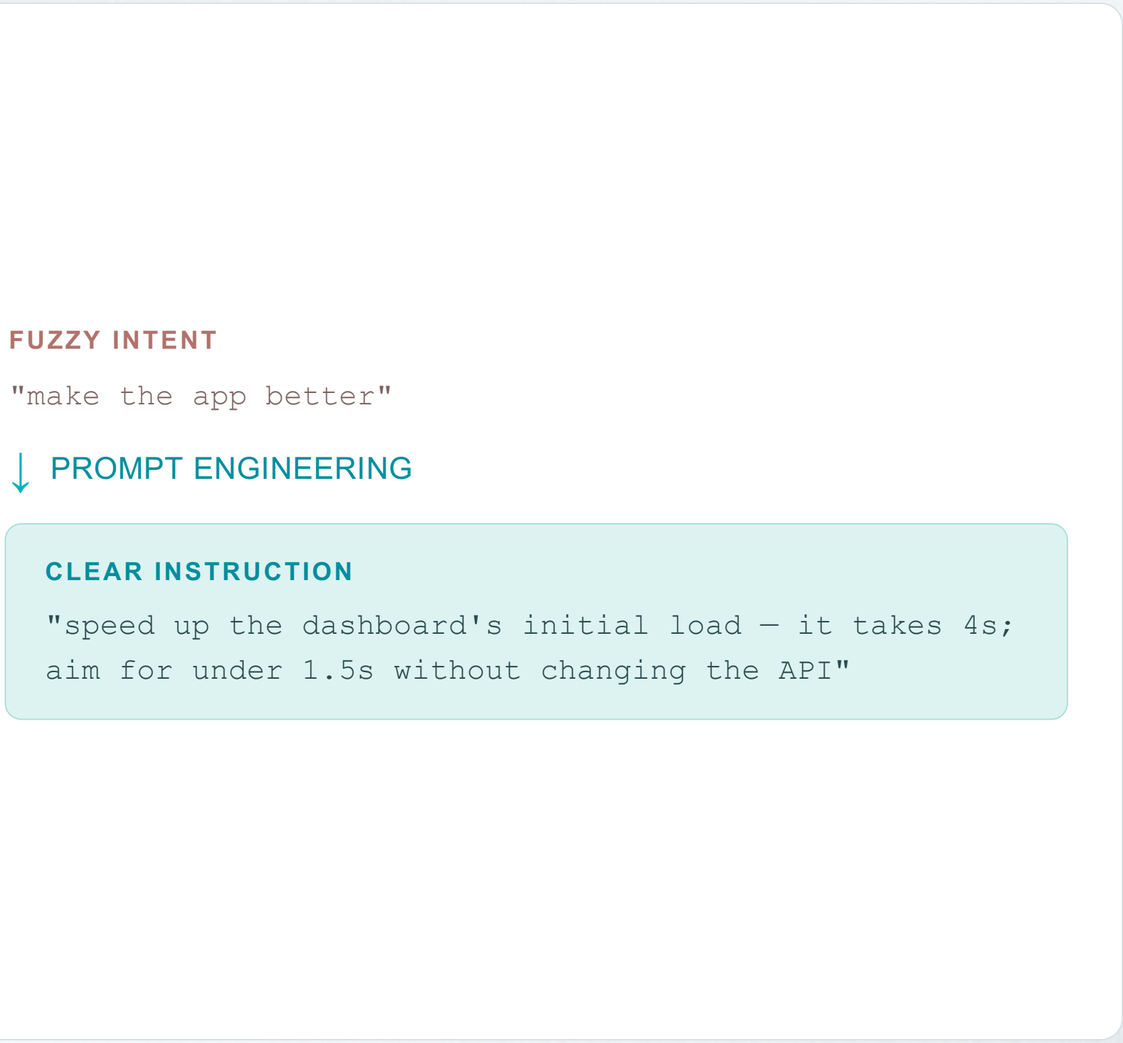
The biggest lever on quality — turning fuzzy intent into clear instructions.

PROMPT ENGINEERING · WHAT IT IS

Turning intent into instructions

Prompt engineering is the craft of saying what you want **clearly enough that Claude can act on it** — and get it right the first time.

The model is capable. Most weak results come from **vague instructions**, not weak models.



PROMPT ENGINEERING · THE FORMULA

A formula for good prompts

You won't always need all five — but naming them turns a wish into a brief.

01 Role — Who Claude should act as.

+

02 Task — The exact job to do.

+

03 Context — Files, data, background.

+

04 Constraints — Rules & limits to respect.

+

05 Output format — The shape of the result.

WORKED EXAMPLE

```
You are a senior React engineer — refactor the checkout form in src/Checkout.tsx
without changing its public API, then return a unified diff.
```

• Role • Task • Context • Constraints • Output format

PROMPT ENGINEERING · REFINE IT

Weak prompt → strong prompt

Build it up one layer at a time — you don't need it perfect on the first try.

START · VAGUE	`fix the styles`
+ TASK & CONTEXT	`fix the broken mobile layout on the pricing page`
+ CONSTRAINTS	`…without touching desktop, using our existing Tailwind tokens`
+ OUTPUT · STRONG	`fix the broken mobile layout on the pricing page without touching desktop, using our existing Tailwind tokens — then summarize what you changed`

PROMPT ENGINEERING · TECHNIQUES

Techniques worth knowing

01

Few-shot examples

Show one or two examples of what good looks like.

02

Structured outputs

Ask for JSON, a table, or a diff — name the shape.

03

Personas

Set a role to shape tone, depth, and rigor.

04

Constraints

State limits up front — what to avoid or keep.

05

Iterative refinement

Treat it as a conversation — react, don't restart.

Combine them A persona + an example + a named output format is often all it takes.

TEAM 400

THAT'S A WRAP

Go build

You can set up, run, and extend Claude Code — from your first session to memory, skills, sub-agents, and connected tools. The best next step is to use it on something real today.

Setup | Everyday use | CLAUDE.md | Skills | Sub-agents | MCP | Prompt engineering

Where to go next

- `code.claude.com/docs`
- Type `/help` in any session
- Questions? Let's talk.