AI Field Guide

CHAPTER FIVE

The Anthropic Playbook

How the people who build Claude actually use Claude — the techniques that still work, the ones that have quietly become no-ops, and the ones that are now actively harmful.

Compiled 14 July 2026 · from Anthropic's engineering blog, docs, and staff interviews

CHAPTER FIVE

The one thing that matters most

> *Give Claude a check it can run: tests, a build, a screenshot to compare. It's the difference between a session you watch and one you walk away from.*
>
> ANTHROPIC — CLAUDE CODE BEST PRACTICES

Claude stops when work *looks* done. Without a check it can run itself, **you are the verification loop** — and you become the bottleneck in your own build.

Boris Cherny, who runs Claude Code, calls verification loops his single biggest recommendation: "*give Claude a way to verify its work… 2–3× the quality.*"

> YOU ARE ALREADY BETTER AT THIS THAN MOST PEOPLE
>
> You made **2,001 preview-eval and screenshot calls in two weeks.** You wrote "*prefer-deployed-url*" into your own memory. You ask Claude to prove things. This is genuinely your standout habit and I want to be clear that the rest of this chapter is not telling you to start verifying — **it's telling you to stop doing it by hand.**

LEVEL	MECHANISM	USE WHEN
Prompt	"…run the tests after implementing"	Any task, today
Session	`/goal all tests in test/auth pass and lint is clean`	A multi-turn grind
Deterministic	A `Stop` **hook** runs your script and blocks the turn from ending	An invariant that must always hold
Second opinion	`/code-review`, or a verification subagent in a fresh context	Before shipping

And always demand **evidence, not assertion**: "show me the test output, the command you ran, the screenshot" — never accept "I've verified it works."

CHAPTER FIVE

Prompt techniques that are now dead

Delete these. Several of them are things you may still be typing out of habit, and two of them *actively make output worse* on current models.

> **"THINK STEP BY STEP" NO LONGER DOES ANYTHING**
>
> In Claude Code, **only the literal keyword** `ultrathink` is recognised. "Think", "think hard", "think carefully", "reason step by step" are now passed through as **ordinary prose**. They do nothing at all.
>
> The replacement is the `/effort` dial. If reasoning looks shallow, **raise the effort — don't prompt around it.**

- `budget_tokens` **extended thinking → returns a 400 error** on Opus 4.7+, Fable 5, Mythos 5. Replaced by adaptive thinking plus an effort setting.
- **Prefilled assistant turns → 400 error** on 4.6+.
- **"CRITICAL: You MUST..." → actively harmful.** Anti-laziness scaffolding written for older models now causes *over-triggering*. Anthropic's docs say it plainly: where you'd have written *"CRITICAL: You MUST use this tool when..."*, just write *"Use this tool when..."*.
- **"After every 3 tool calls, summarise progress"** — remove it. Opus 4.8 gives good progress updates natively.

What still works

- **Explicit beats implicit.** "Can you suggest changes" → Claude suggests. "Change this function to improve performance" → Claude edits. Be imperative when you want action.
- **Give the *why*.** "Never use ellipses" is weak. "Your response is read by a text-to-speech engine, which can't pronounce ellipses" is strong — Claude generalises correctly from the rationale.
- **Long context: put the data at the TOP and the question at the BOTTOM.** Anthropic's docs claim up to a **30% quality improvement** from this alone on 20k+ token inputs. *You paste large files and then ask a short question — you're already doing this right.*
- **Modifiers genuinely work.** "Create an analytics dashboard" gives you something basic. Adding *"Include as many relevant features and interactions as possible. Go beyond the basics."* produces dramatically more. Above-and-beyond has to be *requested*.
- **The golden rule, unchanged:** show the prompt to a colleague with no context. If they'd be confused, so is Claude.

Effort levels — your main dial now

LEVEL	USE FOR
`low`	Short, scoped, latency-sensitive work
`medium`	Cost-sensitive work
`high`	The minimum for anything intelligence-sensitive

LEVEL	USE FOR
`xhigh`	**"The best setting for most coding and agentic use cases." Start here.**
`max`	Toughest debugging. Can overthink; diminishing returns
`ultracode`	`xhigh` + auto-orchestrated dynamic workflows

> *Effort is likely to be more important for this model than for any prior Opus, so experiment with it actively.*
>
> ANTHROPIC — PROMPTING CLAUDE OPUS 4.8

CHAPTER FIVE

Three Opus 4.8 behaviours you must know

It follows instructions literally and does not generalise them. It "does not silently generalise an instruction from one item to another." If you want a rule applied everywhere, you must say "*apply this to every section, not just the first.*"

It favours reasoning over tool calls. If it isn't reading or searching enough, *raise the effort* — that's the lever that increases tool use, not more nagging.

> IT HAS A DESIGN HOUSE STYLE, AND IT IS FIGHTING YOU
>
> Opus 4.8 has a persistent default aesthetic: **cream and off-white backgrounds (~ `#F4F1EA`), serif display faces (Georgia, Fraunces, Playfair), italic accents, terracotta and amber.** Lovely for editorial and hospitality. **Wrong for dashboards, dev tools, and fintech** — which is most of what you build.
>
> And here's the trap: **generic pushback doesn't fix it.** "Make it cleaner," "don't use cream," "it's ugly" just swap one fixed palette for another fixed palette. This is *exactly* the loop you got stuck in on Noor and on Core 2.0 — you described the problem in adjectives and got a different set of defaults back.
>
> **Two things reliably work:**
>
> 1. **Name a concrete alternative** — exact hex codes, typeface, radius, spacing. (This is what your Core 2.0 skill is *for*.)
> 2. **Force divergence before it builds:**
>   *"Before building, propose 4 distinct visual directions for this brief (each as: background hex / accent hex / typeface — one line of rationale). Ask me to pick one, then implement only that direction."*
>
> That second prompt replaces what `temperature` used to do for design variety — and `temperature` now returns a 400 error, so this is the only lever you have left.

CHAPTER FIVE

CLAUDE.md — the rules Anthropic actually gives

Hard rule: keep it under 200 lines. "Bloated CLAUDE.md files cause Claude to ignore your actual instructions."

The test for every single line: "*Would removing this cause Claude to make a mistake?*" If no, cut it.

✅ INCLUDE	❌ EXCLUDE
Bash commands Claude can't guess	Anything Claude can learn by reading the code
Style rules that **differ from defaults**	Standard language conventions
Test runner and how to run it	API docs (link instead)
Repo etiquette — branch and PR conventions	Info that changes frequently
Architectural decisions specific to this project	File-by-file codebase descriptions
Environment quirks, required env vars	"Write clean code"

✅ INCLUDE	❌ EXCLUDE
Non-obvious gotchas	Long tutorials

Diagnostic signals

- Claude **ignores a rule you wrote** → the file is too long; the rule is drowning in noise.
- Claude **asks a question CLAUDE.md answers** → the phrasing is ambiguous.
- You make **the same correction twice** → it belongs in CLAUDE.md, or better, in a hook.

> NEW: `/DOCTOR` NOW TRIMS YOUR CLAUDE.MD FOR YOU
>
> It strips the things Claude can derive (directory layouts, dependency lists, architecture overviews) and keeps the pitfalls, rationale, and non-default conventions. **Run it on every active project.**

Scoped rules — the feature that solves your problem

Put conventions in `.claude/rules/` with a `paths:` frontmatter, and they load **only when Claude touches matching files:**

```
---
paths:
  - "src/**/*.{ts,tsx}"
---
# React conventions
- Function components only; no class components
- Co-locate tests as *.test.tsx
```

That's real context savings versus `@imports`, which load in full at launch whether they're relevant or not.

> THE DISTINCTION THAT WILL SAVE YOU THE MOST
>
> **CLAUDE.md is injected as a user message. It is context, not enforcement.** If a rule *must* hold every time — no exceptions — **make it a hook**, not a memory line. A hook is code. A CLAUDE.md line is a suggestion that Claude is *very likely* to follow.
>
> Every rule you've had to repeat is a rule that wanted to be a hook.

CHAPTER FIVE

Skills — and the feature nobody uses

When to build one: you've pasted the same playbook into chat **three times**, or a CLAUDE.md section has turned into a *procedure* rather than a *fact*.

Why skills are nearly free: progressive disclosure. Only the name and description sit in the system prompt at startup. The full `SKILL.md` loads when invoked. Reference files and scripts load only if referenced. Which means "*the amount of context that can be bundled into a skill is effectively unbounded.*"

The frontmatter you should actually be using

```
---
name: ship
description: Build, typecheck, deploy to Cloudflare Pages, verify the live URL.
             Use when the user says deploy / ship / push live.
disable-model-invocation: true          # only I can trigger it — it has side effects
allowed-tools: Bash(npm run build) Bash(npx wrangler *)   # no permission prompts
effort: xhigh                            # per-skill effort override
context: fork                            # run it in an isolated subagent
---
```

THE KILLER FEATURE ALMOST NOBODY KNOWS ABOUT

Backtick-bang syntax — `!`command`` — **runs a command *before* Claude sees the skill** and inlines the output into the prompt:

```
## PR context
- Diff: !`gh pr diff`
- Comments: !`gh pr view --comments`

Summarise this PR...
```

Claude never runs those commands — it simply *receives* the fully-rendered prompt. Enormous savings in both context and latency versus letting Claude shell out itself.

Authoring rules that matter

- Keep `SKILL.md` **under 500 lines.** Push detail into referenced files.
- **Put the most important instructions at the TOP.** After compaction, skill bodies are re-injected but **truncated to the first 5,000 tokens.**
- `disable-model-invocation: true` costs **zero context** until you type `/name`. Use it for anything with side effects.
- Too many skills and the descriptions get **silently truncated** and stop matching. The whole listing is budgeted at ~1% of the context window.

- **Custom commands and skills have merged.** `.claude/commands/ship.md` and `.claude/skills/ship/SKILL.md` both give you `/ship`.

CHAPTER FIVE

Subagents — including when they backfire

> THE COST REALITY, FROM ANTHROPIC'S OWN RESEARCH
>
> Agents use **~4× the tokens** of chat. Multi-agent systems use **~15×**. And "*token usage by itself explains 80% of the variance*" in performance — parallelisation mostly works by **brute-force token scaling**, not architectural elegance. Know what you're buying.

Fan out when: exploration is breadth-first, the information exceeds one context window, workstreams are genuinely independent, or you're testing competing hypotheses.

Don't fan out when: work is sequential, edits touch the same files, or tasks share heavy dependencies. Anthropic is blunt about this: "*Most coding tasks involve fewer truly parallelisable tasks than research.*"

The highest-ROI subagent use: adversarial review

A reviewer in a **fresh context** sees only the diff — not the reasoning that produced it. "*A fresh context improves code review since Claude won't be biased toward code it just wrote.*"

But constrain it, or it will invent work: "*A reviewer prompted to find gaps will usually report some, even when the work is sound. Chasing every finding leads to over-engineering.*" Tell it: "**Report gaps, not style preferences.**"

Agent Teams (experimental) run 3–5 peers. The killer pattern, straight from the docs: "*Spawn 5 teammates to investigate different hypotheses. Have them talk to each other to try to disprove each other's theories, like a scientific debate.*" That defeats anchoring — which is precisely what makes sequential debugging fail.

CHAPTER FIVE

Context engineering — why "add more context" is wrong

The physics: *context rot*. Models have a finite **attention budget**, and the transformer's pairwise relationships stretch thin as tokens accumulate. **More context makes things worse past a point**, not better.

Aim for "*the minimal set of information that fully outlines your expected behaviour.*"

Daily hygiene

- `/clear` **between unrelated tasks.** Non-negotiable.
- `/compact focus on the API changes` — directed compaction beats the automatic guess.
- `/btw` — ask a side question; the answer lands in a dismissible overlay and **never enters your history**. Perfect for "wait, what does this flag do?"
- `/context` — a live breakdown of what's eating your window.

> THE RULE THAT WOULD HAVE SAVED YOU THE MOST TOKENS THIS MONTH
>
> **Two failed corrections** = `/clear`.
>
> "*A clean session with a better prompt almost always outperforms a long session with accumulated corrections.*"
>
> Your Core 2.0 and Noor design loops ran for *dozens* of turns of "still not faithful yet." At correction number two, the right move was to stop, clear, and write a better prompt with a named token and a screenshot. You even sensed this yourself — you asked, mid-loop, whether you should "*create a new session, and dedicate it ENTIRELY to importing the Figma project.*" **The answer was yes.**

CHAPTER FIVE

How Anthropic's own teams work

- **Product Design** feed **Figma files** straight to Claude and let it build the feature autonomously with a continuous testing loop.
- **Data Infrastructure**, during outages, **paste dashboard screenshots** and get step-by-step guidance plus exact commands.
- **Security Engineering** go design doc → pseudocode → periodic check-ins, and report "*fewer abandoned projects.*"
- **Legal — non-engineers** — built working prototypes with no developer involved.

Boris Cherny, who runs Claude Code

(From public interviews — directional, not official policy.)

- He runs **five Claude instances in parallel** across five git checkouts, tabs numbered 1–5, with system notifications when any needs input. He calls it "*the single biggest productivity unlock.*"

- *"Anytime we see Claude do something incorrectly we add it to the CLAUDE.md."* His team updates it multiple times a week.
- And the line worth sitting with:

> *I don't prompt Claude anymore. I have loops running that prompt Claude... My job is to write loops.*
>
> BORIS CHERNY, HEAD OF CLAUDE CODE

CHAPTER FIVE

The anti-patterns Anthropic names explicitly

1. **The kitchen-sink session.** Task A → unrelated question → back to task A. → `/clear` between unrelated tasks.
2. **Correcting over and over.** Context fills with failed approaches. → After two failed corrections, `/clear` and rewrite the prompt.
3. **The over-specified CLAUDE.md.** → Prune ruthlessly. If Claude does it right *without* the rule, delete the rule.
4. **The trust-then-verify gap.** Plausible code, unhandled edge cases. → If you can't verify it, don't ship it.
5. **The infinite exploration.** "Investigate X" with no scope reads hundreds of files. → Scope it, or send it to a subagent.
6. **Over-prompting.** Anti-laziness scaffolding causes over-triggering on modern models.
7. **Reviewer-induced over-engineering.** A reviewer told to find gaps will find gaps.

CHAPTER FIVE

Ten things to do tomorrow

1. Run `/doctor` in each active project. Get every CLAUDE.md under 200 lines.
2. Set `/effort xhigh` as your default for build sessions.
3. Add a `.worktreeinclude` with `.dev.vars` and `.env` to every Cloudflare project — this is the thing that makes worktrees actually usable for your stack.
4. **Strip "think step by step" and any "CRITICAL: YOU MUST"** from your files. They're no-ops or harmful now.
5. Add the **anti-over-engineering** snippet to your global CLAUDE.md.
6. Convert your two most-repeated playbooks into **skills with** `disable-model-invocation: true` — deploy, and release.

7. Write **one** `PostToolUse` **hook** that runs `tsc --noEmit` after edits. Just ask Claude to write it for you.
8. For your next real feature: **interview → SPEC.md →** `/clear` **→ implement →** `/code-review`**.**
9. For the next design task, **force four visual directions before it builds** — or Opus 4.8 will hand you cream, Playfair, and terracotta again.
0. **Two failed corrections →** `/clear`**.** Make it a reflex.

Sources. Claude Code best practices · Prompting best practices & Prompting Claude Opus 4.8 · Effective context engineering for AI agents · Equipping agents for the real world with Agent Skills · Building a multi-agent research system · How Anthropic teams use Claude Code · Claude Code docs (memory, skills, sub-agents, agent teams, workflows, hooks, worktrees, output styles, model config) · public interviews with Boris Cherny (unofficial, treat as directional).