THE ULTIMATE GUIDE

THE CLAUDE STACK

How I Run My 30-Person AI Company on Claude.
And how you can run your life on it too.

MILES DEUTSCHER
Founder & CEO, AI Edge

5 Hacks • 20+ prompts • 5 agent templates • Replicable today

Welcome to the Stack.

If you watched the video, you saw five hacks. This is everything that didn't fit. Every prompt. Every template. Every config. Every pattern.

I run AI Edge on these five layers. Skills. Voice. Code + Memory. Claudeception. Sub-agents. Stack them and Claude stops being a chatbot and becomes the operating system underneath your work.

This guide is built so you can use it three ways:

- **Skim it** in 10 minutes — read the section openers and the cheat sheet at the end.
- **Build with it** over a weekend — start with Skills, layer the rest in order.
- **Reference it forever** — every prompt is copy-pasteable, every template is yours to fork.

Who this is for

- People who already use Claude and want 10x more out of it
- Operators, founders, and anyone who wants Claude to actually run things — not just chat
- Builders who want a starting point for their own agent stack

My background, in one paragraph

I'm 25, Australian, founder and CEO of AI Edge — a 30-person AI media and software company. I've used Claude every day since launch (3+ years). The systems in this guide are not theory. They're what's running in production right now, including five always-on agents on the Mac Minis behind my desk. If a tip is in here, it's because I rely on it.

> **If you only do one thing after reading this: build the Meeting Prep skill in Hack 1. It pays for the time you spent reading the entire guide on the first use.**

The Stack.

Five layers. Each one extends Claude's relationship to your work. Stack them in order and Claude becomes irreplaceable.

01	**TEACH IT** Skills	Hand Claude a file. It follows that workflow perfectly, every time, forever.
02	**TALK TO IT** Voice	Send Claude a voice memo. Your computer does the work while you walk to coffee.
03	**RUN IT** Claude Code + Memory	Claude lives on your machine. It learns your preferences. It becomes your agent.
04	**BUILD WITH IT** Claudeception	Build apps that have Claude inside them. Real tools. Persistent storage. Forever.
05	**MULTIPLY IT** Sub-Agents	Spawn an entire team. Each agent has its own role, skills, and memory. Always on.

The arc: teach it → talk to it → run it → build with it → multiply it. Every layer compounds the last. The full stack is what 'AI native' actually means.

01

TEACH IT

Skills · Hand Claude a workflow it follows perfectly, every time.

What is a Skill, exactly?

A Skill is a markdown file. You give it a name, a trigger description, and a set of instructions. When you mention something that matches the trigger, Claude finds the file, follows the instructions, and produces the output.

That's it. No code. No setup. A text file.

The reason this is the most important hack: **Skills make Claude repeatable**. The same input produces the same quality of output, every time. You stop re-prompting. You stop drift.

The anatomy of a Skill

Every good Skill has four parts:

- **Name** — short, hyphenated, like a filename. `meeting-prep`
- **Trigger description** — the words that activate it. Include synonyms.
- **Behavior** — exactly what Claude does, in dot points.
- **Output spec** — exactly what gets returned, in dot points, with constraints.

If any of those are vague, the skill is vague. If all four are specific, the skill is bulletproof.

The Meeting Prep skill (full)

This is the skill from the video. Paste this prompt into Claude.ai and you'll have it in 30 seconds.

Build prompt — copy/paste

```
Build me a skill called "meeting-prep".

Trigger: when I say I'm prepping for a meeting, 1:1, sync, call, or interview.

Behavior:
- Ask up to 4 questions max: who is it with, what's the topic,
what outcome I want, how much time I have.
- If I paste a calendar event, email thread, or LinkedIn profile,
skip the questions and use that as the input.
- If I name a person or company you don't recognize, search the web first.

Output a one-page brief with:
- 3 lines of context on the person and company
- 3 talking points framed as questions, not statements
- My desired outcome restated in one sentence
- 1 risk or objection to watch for

Constraints:
- Under 200 words total
- Bullet points only, no prose
- No fluff, no hedging, no "depending on the situation".
```

How to test it

Paste this right after, in the same conversation:

Test prompt

```
I'm prepping for a meeting with [person's name and role] in [time].
Topic is [what it's about]. I want to walk out with [your outcome].
```

Why this skill works for anyone

- Sales call → input is the prospect's LinkedIn, output is the call plan
- Performance review → input is your role and last quarter's wins, output is the framing
- Job interview → input is the job description, output is the prep sheet
- Parent-teacher conference → input is the email thread, output is the questions to ask

- Investor meeting → input is the firm + partner name, output is your one-pager

4 more Skill templates.

Each one solves a recurring task most people brute-force every time. Build all four and you reclaim ~5 hours a week.

Daily Review

Trigger: when you say "review my day" or "daily wrap"

daily-review

```
Build me a skill called "daily-review".

Trigger: when I say "review my day", "daily wrap", "end of day",
or paste my calendar.

Behavior:
- Ask me 3 questions max: what got done today, what got blocked,
what's the one thing I'm carrying into tomorrow.
- If I paste my calendar or task list, skip the questions and infer.

Output:
- "Done" — top 3 wins, no hedging
- "Blocked" — what stalled, what unblocks it
- "Carry" — the ONE thing for tomorrow
- "Reflection" — one sentence on how I'm tracking against my week

Constraints:
- Under 150 words. No prose. No motivational tone. Direct.
```

Cold Outreach

Trigger: when you paste a LinkedIn profile or company URL

cold-outreach

```
Build me a skill called "cold-outreach".

Trigger: when I paste a LinkedIn profile, a company URL, or say
"draft outreach to [person]".

Behavior:
- Research the person and company (web search if needed)
- Find ONE specific reason to reach out — recent post, hire,
product launch, or shared connection
- Draft 3 message variants

Output:
- Variant A: Direct (state the ask in line 2)
- Variant B: Curious (open with a question about their work)
- Variant C: Warm (reference shared context, no ask)

Each variant: 50-80 words, no "hope you're well", no "I came across
your profile", first person, conversational.
```

Decision Stress-Test

Trigger: when you say you're trying to decide between things

decision-stress-test

```
Build me a skill called "decision-stress-test".

Trigger: when I describe a decision I'm weighing, or say "should I
do X or Y" or "I'm thinking about [decision]".

Behavior:
- Refuse to validate. Find the assumption I'm making.
- Steelman the option I'm leaning AWAY from
- Identify what I'd need to be true for my preferred option to work
- Name the version of me 12 months from now that regrets each path

Output:
- "Assumption you're making" — one sentence
- "Steelman the other side" — 3 strongest reasons
- "What needs to be true" — required conditions
- "12-month regret check" — 2 sentences each option

Push back. Don't agree.
```

Skill best practices.

Get the trigger description right

This is where most skills fail. If your trigger is too narrow, the skill never fires. If it's too broad, it fires when you don't want it to. The fix:

- List 3-5 phrases YOU would actually say to invoke it. Use those.
- Include synonyms for the noun ("meeting / 1:1 / sync / call")
- Include synonyms for the verb ("prep / get ready / brief me on")
- Test by saying it 3 different ways. If it doesn't fire on at least 2 of 3, rewrite.

Be specific about the output, not the input

Vague behavior is fine — Claude is smart. Vague output is what makes skills feel inconsistent. Always specify:

- Word count or character count cap
- Format (bullets vs prose vs table)
- Tone constraints ("no hedging", "no apologies")
- Things you DON'T want ("never start with 'depending on...'")

Common failure modes

- **The skill is too long.** Aim for 200-400 words total. If it's longer, it's two skills.
- **The skill apologizes.** Skills that say "I'll do my best" or "depending on the situation" produce hedging output. Strip them.
- **The skill duplicates Claude's defaults.** Don't tell it to "be helpful" or "think carefully". Tell it the SPECIFIC thing only this skill should do.
- **The skill is invoked but ignored.** Usually because trigger phrases don't match how you actually talk. Audit and rewrite the trigger.

Iterate on real use, not in your head

Use a skill 3 times in real situations. After each use, edit the skill in 30 seconds — tighten the constraint that drifted, add the edge case you hit. Within a week the skill is dialed in. Don't try to write a perfect skill on day one.

> **Action today: build Meeting Prep + Daily Review. That's 2 skills, 4 minutes, immediate compounding ROI.**

02

TALK TO IT

Voice · Drive Claude with voice memos from anywhere.

Two ways to do voice.

There's the easy way (built into Claude — works today, 2-minute setup). And there's the operator way (custom-built — runs 24/7, hits multiple agents). Both are useful. They serve different jobs.

Way 1 — Dispatch (the easy way)

Dispatch is built into Claude. It pairs your phone to your desktop. You send a voice memo or text from your phone. Claude on your desktop runs the task. The result comes back to your phone.

Setup (90 seconds)

- Open Claude Desktop on Mac or Windows
- Click the Cowork tab
- Click Dispatch — a QR code appears
- Open Claude on your phone, scan the QR
- Paired.

What to know

- Your desktop must stay awake while Dispatch is active. Disable energy saver.
- Dispatch can use any connector you have set up in Claude Desktop (Gmail, Calendar, Drive, etc.)
- Best for: tasks that need 30 seconds to 5 minutes of work.

- Reliability is around 50% for complex multi-step tasks today. Improving fast.

Way 2 — Custom voice pipeline (the operator way)

This is what I actually use. It's a custom Telegram bot that pipes voice memos through Whisper transcription, routes the text to specific Claude agents on my Mac Mini server, and sends responses back to Telegram. Always on. Routes to multiple specialist agents.

> **I'm building a full walkthrough video for this stack. If you want it next, leave a comment with 'voice' on the video.**

Voice pipeline architecture (high level).

If you want to build your own version of the operator path, here are the components and what each one does. Pick your stack and research from here.

The five components

- **Capture surface.** Where you send the voice memo from. Telegram, WhatsApp, Discord — anywhere with a bot API.
- **Transcription.** Voice → text. OpenAI Whisper API is the easiest. Local Whisper models (whisper.cpp) work offline.
- **Router.** Decides which agent handles the message. Pattern match on names ("Aria, ...") or use a small classifier.
- **Agent backend.** Claude API calls (or Claude Code instances) configured per role. Each one has its own system prompt + skills + memory.
- **Return surface.** Sends results back. Same channel as capture. Reply in-thread for context.

Stack options to research

- **Easy mode:** n8n or Make.com — visual workflow builder. Telegram trigger → Whisper → Claude API → Telegram reply. No code. Set up in an afternoon.
- **Custom mode:** Python script using `python-telegram-bot` + `openai` + `anthropic`. Run it on a Mac Mini, Raspberry Pi, or VPS.
- **Pro mode:** Multi-agent backend where each "agent" is a separate Claude conversation thread with its own CLAUDE.md and skill set. (See Hack 5.)

Cost reality check

- Whisper API: ~$0.006 per minute of audio. 10 voice memos a day = under $1/month.
- Claude API for the responses: depends on model and tokens. Budget $5-20/month for personal use.
- Hosting: Mac Mini you already own. VPS is $5-10/month.
- Total: under $30/month for the operator setup.

> **The full setup walkthrough — every line of code, every config — is the next video. Comment 'voice' to vote it up the queue.**

10 voice memos to try this week.

Voice works because the constraint forces specificity. You can't write a wall of text into a voice memo. The pattern that works: **action verb + specific input + specific output + constraint**.

Daily operations

1 — Morning prep

```
Look at my calendar for tomorrow. For every meeting before noon,
run my meeting-prep skill on it. Put the briefs in a single doc
on my desktop called Tomorrow Briefs.
```

2 — Inbox triage

```
Pull every email from the last 24 hours. Filter to ones that
need a reply from me. Sort by sender priority. Draft replies for
the top three.
```

3 — Slack triage

```
Look at my unread Slack messages from today. Summarize what's
urgent, what's an FYI, and what I can ignore. Three lists, one
sentence each.
```

Strategic / creative

4 — Strategic gut check

```
I'm thinking about [topic / decision]. Run my decision-stress-test
skill on it. Don't agree with me.
```

5 — Pattern check

```
Open the doc called [project name]. Read the last 10 entries.
Tell me the one pattern I keep avoiding.
```

Content / research

6 — Daily intel brief

```
Search the web for the top 5 stories on [topic] from the last
24 hours. For each, give me the headline, the source, and a one-line
take. Save it as today's brief.
```

7 — Content pattern mining

```
Pull my last week of tweets. Find the top 3 by engagement.
Tell me what they had in common.
```

Personal / admin

8 — Expense pull

```
Go through my receipts folder. Pull anything from the last 30
days. Categorize as business or personal. Output the totals.
```

9 — Personal logistics

```
Read my grocery list note. Check the recipes I bookmarked this
week. Tell me what I'm missing for any of them.
```

10 — Walking notes

```
I'm walking. Just talking. Take whatever I'm rambling about and
turn it into a structured note in my Ideas folder.
```

03

RUN IT

Claude Code + Memory · Claude lives on your machine. It learns you.

Claude Code is not just for developers.

Claude Code is a general-purpose agent that lives on your machine, has terminal access, can read and write files, can run commands, and can use any MCP connectors you set up.

Most people think it's for coding. The biggest unlock is everything else.

10 ways to use Claude Code without writing code

- **File organization** — "Sort my Downloads folder by content type, rename meaningless filenames"
- **Document analysis** — "Read these 30 PDFs in /research and tell me the 3 themes"
- **Receipt processing** — "Extract date, vendor, amount from every receipt PDF in this folder, output a CSV"
- **Photo organization** — "Sort the iCloud Photos folder into subfolders by year and month"
- **Web scraping (one-off)** — "Scrape the top 10 search results for [keyword] into a spreadsheet"
- **Meeting prep at scale** — "For every meeting on my calendar this week, generate a prep brief"
- **Personal CRM upkeep** — "Update my contacts list with everyone I emailed in the last 90 days"
- **Email cleanup** — "Find every newsletter I haven't opened in 60 days, draft unsubscribe replies"
- **Custom tool building** — "Build me a small script that converts my voice memos into a journal"
- **System monitoring** — "Every morning, summarize anything weird from my server logs in plain English"

If you used Claude.ai chat, switching to Claude Code is the biggest single jump in capability you can make. Same intelligence — but with hands.

The 3-layer Memory system.

Claude Code reads three different memory files, in this order. The deeper layers override the shallower ones. Most people only know about one layer.

Layer 1 — Global memory

Path: ~/.claude/CLAUDE.md

This file is read by every Claude Code session, on every project, everywhere on your machine. Put facts about YOU here — name, role, communication preferences, tools you use, people you work with.

If you only build one Memory file, build this one. Build it once. Update it monthly.

Layer 2 — Project memory

Path: ./CLAUDE.md (in your project root)

This file is read whenever you start Claude Code in this folder. Put facts about THIS project here — its purpose, its conventions, its key files, its tech stack.

Useful for: any folder you return to repeatedly. Code repos, writing projects, content folders, research vaults.

Layer 3 — Local memory

Path: ./.claude/CLAUDE.local.md

Same as project memory, but private to YOU. Don't commit this to git. Use it for personal preferences specific to this project — your debugging shortcuts, your in-progress notes, your scratchpad.

How to think about the layers

- **Global** = who you are
- **Project** = what you're working on
- **Local** = how YOU specifically work on this project

When the layers conflict, the deepest layer wins. So your project Memory can say "use prose paragraphs in this codebase" and it'll override your global Memory's "dot points only" preference, but only for this project.

Annotated CLAUDE.md template (global).

Copy this. Adapt the values. Save it to `~/.claude/CLAUDE.md`. Live with it for a week, then iterate.

Global CLAUDE.md template

```
# CLAUDE.md

## About me
- [Your name]. [Age, location].
- [Role / company / what you do in one line].
- [One line on your domain expertise].

## My team / network
- [Name]: [role / how they help you]
- [Name]: [role / how they help you]
- [Add 3-8 frequent collaborators]

## How I want you to work with me
- Dot points. Never long prose unless I ask.
- No em-dashes. Use a period or a hyphen.
- Don't agree with me reflexively. Push back when I'm wrong.
- Skip the preamble. Get to the answer.
- One specific recommendation, not three options.

## Tools and stack I use
- [Editor: VS Code / Cursor]
- [Terminal: zsh / bash + main shortcuts]
- [Languages / frameworks]
- [Anything else recurring]

## Things I'm working on right now
- [Current project 1]
- [Current project 2]
- [Update this section monthly]

## Pet peeves (don't do these)
- Don't say "great question"
- Don't add disclaimers about your own limitations unless asked
- Don't restructure my files without asking first
- [Add yours as you find them]
```

Why each section matters

- **About me / team** — context Claude can't infer. Saves hours of re-explaining.

- **How to work with me** — the single biggest output quality lever. This is where the personality of your Claude is set.
- **Tools and stack** — Claude stops suggesting things you don't use.
- **Currently working on** — Claude can connect new requests to existing context. Update monthly minimum.
- **Pet peeves** — the negative examples. Often more useful than positive instructions.

5 prompts to test your Memory.

After you save your CLAUDE.md, run these 5 prompts in fresh Claude Code sessions. The output should clearly reflect the preferences you wrote. If not, your Memory file is too vague.

Test 1 — Decision making (does it push back?)

```
Help me think through a strategic decision I'm weighing.
[Describe a decision]
```

Test 2 — Drafting (does it match your voice?)

```
Draft a quick email to [name] saying we need to push the launch.
```

Test 3 — Teaching (does it match your level?)

```
Explain how transformers work to me.
```

Test 4 — Advice (does it factor in your context?)

```
I'm thinking about hiring a [role]. What should I look for?
```

Test 5 — Reflection (does it know what you're working on?)

```
Review the last week's worth of work. What did I miss?
```

Memory tuning tips

- **Add the negative.** "Don't do X" is more powerful than "do Y" because it removes a default behavior.
- **Use examples.** If you want a specific tone, paste 2-3 lines of your own writing as an example.
- **Update the "working on" section weekly.** This is the most time-sensitive part of your memory.
- **Audit monthly.** Re-read your CLAUDE.md. Anything outdated? Anything you keep manually overriding? Edit it.
- **Don't make it too long.** 30-60 lines is the sweet spot. Past that, key instructions get diluted.

> **Skills teach Claude your processes. Memory teaches Claude about you. Use both. Use them today.**

04

BUILD WITH IT

Claudeception · Build apps that have Claude inside them.

Claudeception, defined.

An artifact is a self-contained app you build inside Claude. Most people use them as throwaway demos. The hack: **artifacts can call the Claude API as their backend**.

That means you can build a real, working app — with Claude as its brain — in 60 seconds. The app saves data across sessions. It runs on demand. You use it forever.

The architecture

- **Frontend:** The artifact UI (React or HTML). You design it with one prompt.
- **Backend:** A call from the artifact to `api.anthropic.com`. Auto-handled inside Claude — no API key needed.
- **Storage:** `window.storage` — persistent key-value store. Survives session refresh, browser close, days later. It just works.
- **Tools (optional):** The API call can use web search, MCP servers, etc.

You don't need to know how any of this works. You just describe what you want. Claude builds the whole stack.

When Claudeception beats other options

- When you do the same thinking task more than 3 times a week — bake it into an app.
- When the input is messy and the output structure is consistent.
- When you want to save the inputs/outputs forever (instead of losing them in chat history).

- When you'd build a Streamlit app or write a script — but faster, no setup.

> **This is the layer that makes most people realize Claude isn't a chatbot. It's a platform.**

The Build Prompt template.

This is the structure that produces a high-quality Claudeception artifact in one shot. Follow it. Fill in the blanks. Paste.

Build prompt template

```
Build me an artifact called "[NAME]".

WHAT IT DOES:
- [Input description — what the user provides]
- [Action — what happens when they trigger it]
- [Output — what they get back]

BEHAVIOR:
- On [trigger], call the Claude API with [the input]
- Use claude-sonnet-4-20250514, max_tokens [budget]
- Ask the API to return strict JSON only, no preamble
- Parse the JSON and render [the output]

EACH OUTPUT CARD/ITEM SHOWS:
- [Field 1]
- [Field 2]
- [Field 3]
- [Optional: a Save button]

PERSISTENT STORAGE (if applicable):
- Save button pushes the item to window.storage
under "[storage-key]"
- A "[Vault / History / Saved]" tab at top shows all saved items

SYSTEM PROMPT FOR THE API CALL:
- [Tone / voice instructions]
- [Output structure constraints]
- [Things to avoid]

STYLE:
- [Light or dark mode]
- [Layout: cards, list, single panel]
- [Animation: fade-in, instant, etc.]
```

Filling it in well

- **Be specific about output structure.** JSON format with field names. "Return as {angle: string, text: string, chars: number}".

- **System prompt is everything.** The API call is where your voice and constraints live. Treat it like a Skill.
- **Pick the cheapest model that works.** Sonnet for most tasks. Haiku for high-volume. Opus only when you need depth.
- **Set max_tokens low.** 500-1000 is usually plenty. Smaller = faster + cheaper.

5 Claudeception artifacts worth building.

1 — Tweet Riff Machine

Paste any tweet. Get 5 alternatives in your voice from different angles.

Build prompt

```
Build me an artifact called "Tweet Riff Machine".

Input: any tweet.
Output: 5 alternative tweets in my voice — operator, contrarian,
data point, question, story angles.

Backend: claude-sonnet-4-20250514, max_tokens 1000, JSON output.
Each riff: angle label + tweet + char count (green if <280) + Save.
Storage: window.storage key "saved-riffs". Vault tab.
Voice: punchy, declarative, no hedging, no em-dashes, under 280 chars.
Style: dark mode, cards stack vertically, smooth fade-in.
```

2 — Cold Outreach Generator

Paste a LinkedIn URL. Get 3 first-message variants.

Build prompt

```
Build me an artifact called "Cold Outreach Generator".

Input: a LinkedIn URL or a name + company.
Output: 3 first-message variants — direct, curious, warm.

Backend: claude-sonnet-4-20250514. Use web search for the person.
Each card: variant label + message + word count + Copy + Save.
Storage: window.storage key "outreach-saved". Sent tab.
Voice: 50-80 words, no "hope you're well", first-person.
Style: light mode, single-column cards.
```

3 — Daily Standup Writer

Paste yesterday's notes / commits. Get a daily standup message.

Build prompt

```
Build me an artifact called "Standup Writer".

Input: free text — yesterday's notes, commits, or a brain dump.
Output: a paste-ready standup message.

Backend: claude-haiku for speed.
Output structure: Done / Doing / Blocked, 2 bullets each, no fluff.
Storage: keep last 7 days under "standup-history" key.
Style: minimal, monospace, copy button prominent.
```

4 — Decision Pre-Mortem

Describe a decision. Get the 3 ways it goes wrong and what to watch.

Build prompt

```
Build me an artifact called "Pre-Mortem".

Input: a decision I'm about to make.
Output: 3 failure modes with leading indicators.

Backend: claude-opus for depth, max_tokens 1500.
Each mode: scenario name + how it unfolds + 2 leading indicators
to watch + a check-in date.
Storage: every pre-mortem saved under "premortems" key.
Style: serious, dark mode, no playfulness.
```

5 — Voice → Email

Paste a voice transcript. Get a clean email reply.

Build prompt

```
Build me an artifact called "Voice to Email".

Input: a transcript of me rambling about how I want to reply.
Output: a clean, sent-ready email.

Backend: claude-sonnet-4-20250514.
Tone: professional, direct, my voice (samples in the prompt).
Output: subject + body. Word count target shown.
Storage: drafts saved under "email-drafts" with timestamps.
```

Persistent storage patterns.

The Vault pattern is the difference between a demo and a real tool. Three patterns to know:

Pattern 1 — Single key, growing list

Save everything to one storage key as an array. Best for: history, saved items, drafts.

Code shape

```
// Save
const existing = await window.storage.get("saved-riffs") || [];
await window.storage.set("saved-riffs", [...existing, newRiff]);

// Read
const all = await window.storage.get("saved-riffs") || [];
```

Pattern 2 — Date-keyed entries

Each save is its own key, prefixed with date. Best for: journals, daily logs, scheduled outputs.

Code shape

```
const today = new Date().toISOString().slice(0,10);
await window.storage.set(`journal-${today}`, entry);

// List all entries
const keys = await window.storage.list("journal-");
```

Pattern 3 — User-tagged

User can tag their saves. Best for: organizing by category. Build a filter UI on top.

API best practices in artifacts

- **Always wrap API calls in try/catch.** Show a loading state. Show a fail state. Never let the artifact silently break.
- **Stream the response if you can.** The artifact feels 5x more alive when output appears progressively.
- **Cache common responses.** If users hit the same input twice, return the cached version.
- **Validate the JSON.** Sometimes the API returns prose instead of JSON. Have a fallback parser or retry once.

• **Don't ask for the kitchen sink in one call.** Two cheap focused API calls beat one expensive everything-call.

Action today: build the Tweet Riff Machine OR pick the artifact closest to a task you do daily. Use the build prompt template. 60 seconds. Then use it for a week and iterate.

05

MULTIPLY IT

Sub-Agents · Run a team of specialized Claudes.

How an agent is actually made.

Most people hear "AI agent" and picture something complicated. It's not. An agent is just three things stacked together:

- **Memory** — the agent's IDENTITY (CLAUDE.md). Who it is, what it knows about you.
- **Skills** — the agent's CAPABILITIES (skill files). The processes it runs.
- **Tools** — the agent's AGENCY (MCP connectors). What it can touch in the world.

That's it. Memory + Skills + Tools = an agent. You already learned all three layers in the previous hacks. Hack 5 is just stacking them deliberately, per role.

The first key insight

Most people use one Claude for everything. One CLAUDE.md, one set of skills, generalist instructions. Then they wonder why outputs feel inconsistent — content drafts get the operator framing, financial questions get content-friendly hedging, code reviews get prose explanations.

The fix isn't a smarter Claude. It's **multiple Claudes, each specialized**. One for content. One for finance. One for ops. Each with its own memory and skills. You stop fighting drift and start getting on-target output every time.

The second key insight

Specialization compounds. A content agent that knows your voice writes better captions than general Claude. A research agent that always cites sources writes better briefs. A coach agent that pushes back gives better feedback. Each agent gets sharper the more you use it. The team gets sharper the more agents you have.

4 architectural patterns.

There are four ways to structure a multi-agent setup. Pick the one that matches the problem. Most people graduate through them in this order.

Pattern A — Single Specialist

One Claude. One role. Always-on.

This is the entry point. You pick ONE recurring task type and build a dedicated Claude for it. Personal Editor. Strategic Coach. Research Analyst. Use a Project on Claude.ai to give it dedicated CLAUDE.md + skills, and only use that project for that task.

Use case: you're tired of re-explaining context. Building a single specialist gives you 80% of the multi-agent value with 20% of the complexity.

Pattern B — Parallel Execution

Multiple Claudes spawned in one task.

In Claude Code, you spawn sub-agents that work simultaneously on different parts of a problem. They run in parallel, return their outputs, and the main Claude consolidates.

Use case: a complex task that decomposes cleanly. Research X + draft Y + build Z + review all three. 4x faster than serial.

Pattern C — Specialist Team

Multiple persistent agents, each with their own role.

Each agent runs as its own Claude conversation / Project / instance. Each has its own CLAUDE.md and skills. You invoke the right agent for the job. They don't talk to each other directly — you orchestrate.

Use case: you have multiple recurring task types. Content + finance + ops. This is OpenClaw-tier.

Pattern D — Orchestrator + Workers

One agent coordinates. Others execute.

Most advanced. One agent (the orchestrator) decomposes incoming tasks, decides which specialist should handle each piece, dispatches the work, and consolidates results. You only talk to the orchestrator. It talks to the workers.

Use case: when you have enough specialists that picking the right one is itself a task. Or when the same task needs multiple agents to weigh in.

Most readers should start with Pattern A. Build ONE specialist. Live with it for a week. Add another. Skip ahead only if you've already done this.

Two paths: Solo or Operator.

Where you start depends on what you do. Both paths are real. Pick one — don't try to do both at once.

Solo path — 3 personal agents anyone can build

If you're not running a team or business, build these three. They cover ~80% of personal/professional use cases.

- **The Strategist** — pushes back on your decisions. Devil's advocate. Always finds your hidden assumptions.
- **The Editor** — reviews and improves your writing in your voice. Catches your tics. Tightens your drafts.
- **The Coach** — accountability and reflection. Weekly reviews, blocker identification, growth tracking.

You can build all three on Claude.ai using Projects. No code required. Each gets its own Project with its own custom instructions (= CLAUDE.md) and uploaded skills.

Operator path — 5 business agents (OpenClaw structure)

If you're running a team, a business, or multi-stream creator operations, here's the structure that runs my company. Five always-on specialists.

- **The CFO / Orchestrator** — financial overview, daily P&L;, dispatches tasks to the right specialist. (In OpenClaw: Oscar)
- **The Researcher** — deep dives, source synthesis, competitive intel, market scans. (Samuel)
- **The Content Lead** — channels, copy, hooks, performance review. (Aria)
- **The Engineer** — automations, scripts, internal tooling, system maintenance. (Cyrus)
- **The Quant / Specialist** — domain-specific deep work. In my case trading. In yours: legal, design, ops, whatever your edge is. (Roman)

Operator-path agents typically run as their own Claude Code sessions with their own CLAUDE.md, skills, and connectors. They run on a Mac Mini server, a VPS, or even on your laptop with energy saver disabled.

Agent Template 1 — The Strategist.

SOLO Decision pushback. Hidden-assumption finder. Never agrees on the first pass.

Identity (CLAUDE.md)

CLAUDE.md

```
# Strategist Agent

## Role
You are my strategic advisor. Your job is to find what I'm missing.
You disagree first, agree second, never validate without analysis.

## How you work
- Refuse to validate decisions on first pass.
- Always identify the assumption I'm making.
- Steelman the option I'm leaning AWAY from.
- Name the version of me 12 months from now that regrets each path.
- One specific recommendation per decision, not options.

## What you know about me
- [Your name, role, current major projects]
- [Your decision-making patterns — what you over-index on,
what you under-weight]
- [Recent decisions and their outcomes — for learning]

## What you don't do
- Don't agree with me reflexively
- Don't hedge ("it depends...")
- Don't restate my decision back to me as analysis
- Don't moralize. Just analyze.
```

Skills

Skill 1 — decision-stress-test

```
Build me a skill called "decision-stress-test".

Trigger: when I describe a decision I'm weighing.

Behavior:
- Find the assumption I'm making
- Steelman the option I'm leaning away from
- Identify what would have to be true for my preferred option
- Name the 12-month-regret version for each path

Output: 4 sections, dot points, no preamble. Push back hard.
```

Skill 2 — pre-mortem

```
Build me a skill called "pre-mortem".

Trigger: when I say I'm about to launch / commit to / decide on X.

Behavior:
- Imagine it's 6 months from now and the decision failed
- Identify the 3 most likely failure modes
- For each, name the leading indicator I should watch for now

Output: failure mode + indicator + check-in date.
```

How to invoke

Use a dedicated Project on Claude.ai called "Strategist". Paste the CLAUDE.md as the custom instructions. Add the two skills as project files. Whenever you're weighing something — before you commit to it — open this project and dump the decision in.

Agent Template 2 — The Editor.

SOLO Reviews everything you write in YOUR voice. Catches drift. Tightens drafts.

Identity (CLAUDE.md)

CLAUDE.md

```
# Editor Agent

## Role
You are my personal editor. Your job is to make my writing
sound more like me, not less.

## My voice characteristics
- [List 5-7 traits — punchy, declarative, etc.]
- [List your verbal tics — words you use, words you avoid]
- [Tone register — operator energy / academic / casual]

## Examples of my voice (paste 3-5 of your best paragraphs here)
- [Example 1 — copy a paragraph from your best post / email / essay]
- [Example 2]
- [Example 3]

## How you edit
- Cut adverbs ruthlessly. Cut "very" always.
- Replace passive with active.
- Find the strongest sentence and put it first.
- Kill any opener that says "in today's [topic]" or "let's dive in".
- Em-dashes: replace with periods or hyphens.

## What you don't do
- Don't make my writing more polite — it makes it weaker
- Don't add disclaimers I didn't write
- Don't over-edit short pieces (under 100 words: trust them)
- Don't summarize. EDIT.
```

Skills

Skill 1 — voice-match-check

```
Build me a skill called "voice-match-check".

Trigger: when I paste a draft and ask if it sounds like me.

Behavior:
- Compare against my voice examples in memory
- Identify 3 lines that DON'T sound like me — tell me why
- Suggest a rewrite for each

Output: original line + reason it's off + rewrite. No prose
introduction. Just the diff.
```

Skill 2 — tighten-draft

```
Build me a skill called "tighten-draft".

Trigger: when I paste anything I've written and say tighten /
trim / cut.

Behavior:
- Cut every word that doesn't add meaning
- Combine sentences where possible
- Find the strongest line and lead with it
- Aim for 30% shorter

Output: tightened version + word count before/after + the
3 highest-impact cuts you made.
```

How to invoke

Same as Strategist — dedicated Project. The voice examples in your CLAUDE.md are the most important part. The more high-quality samples you paste, the better the editor gets at imitating you. Update monthly.

Agents 3, 4, 5 — Researcher / Operator / Coach.

Compact templates. Same structure as the first two — Identity + Skills + How to invoke.

Agent 3 — The Researcher

SOLO + OPERATOR

Researcher CLAUDE.md (excerpt)

```
# Researcher Agent

## Role
You are my research analyst. Source-grounded. Skeptical. Synthesizing.

## How you work
- Always cite. Link is not optional.
- Distinguish primary sources from secondary
- Flag conflicting evidence — don't smooth it over
- Give me the conclusion FIRST, then the support
- One executive summary line + 3 supporting findings

## Skills attached
- deep-research-brief: 5+ sources, 1-page synthesis, opposing views
- claim-check: stress-test a specific claim against evidence
- competitor-scan: structured analysis of [N] competitors
```

Agent 4 — The Operator

OPERATOR

Operator CLAUDE.md (excerpt)

```
# Operator Agent

## Role
You handle my admin. Calendar. Inbox triage. Daily prep.
Reactive, not strategic.

## How you work
- Speed over depth. Most tasks should resolve in under 30 seconds.
- Group similar items — never give me 8 separate decisions if
3 categories would do
- Default action is the obvious one. Ask only when you can't infer.

## Tools you need
- Gmail / calendar / Slack connectors
- Read access to my CLAUDE.md (so you know my team and priorities)

## Skills attached
- morning-prep: calendar + inbox + priorities, 1-page output
- inbox-triage: action / FYI / archive, draft replies for action
- end-of-day-wrap: what got done, what carries, what needs human
```

Agent 5 — The Coach

SOLO

Coach CLAUDE.md (excerpt)

```
# Coach Agent

## Role
You are my growth coach. Not therapy. Not motivation. Pattern
recognition + accountability.

## How you work
- Look across my recent journal / notes for repeating themes
- Surface the pattern I'm avoiding seeing
- Ask one question at a time, not three
- Never platitudes. Never "trust the process". Specific.

## What you know about me
- [Your stated goals, current and past]
- [Your recurring blockers — read from journal entries]
- [Your values — what you say matters most]

## Skills attached
- weekly-review: structured Sunday session
- blocker-pattern: identify the blocker showing up repeatedly
- value-alignment-check: is what I'm working on aligned with
what I said matters?
```

Making agents work together.

The five agent templates above are powerful alone. They become a SYSTEM when they share context. Three patterns that scale:

Pattern 1 — Shared upstream memory

All agents read from the same global CLAUDE.md (you, your team, your stack). Each agent then has its OWN role-specific instructions on top. That way, every agent knows who you are without you maintaining 5 separate identity files.

Setup: keep ~/.claude/CLAUDE.md as the shared identity. Each agent's specific role goes in their Project's custom instructions on top.

Pattern 2 — Handoff via files

Agents communicate by writing to and reading from files. Researcher writes a brief.md. Strategist reads brief.md. Editor polishes brief.md. You never copy-paste between conversations.

Setup: use Claude Code with sub-agents, OR use a shared Drive folder with each agent reading/writing to it.

Pattern 3 — Orchestrator routes

One agent (often the CFO/Orchestrator role) takes your input, decides which specialist should handle it, and routes. You stop having to know which Claude to use for which task.

Setup: build a CLAUDE.md for the orchestrator that lists the other agents and their specialties. Trigger phrases match incoming tasks to specialists.

Practical setup options, easiest to hardest

- **Easiest:** 5 separate Projects on Claude.ai. Each is one agent. You manually pick which to use.
- **Middle:** Claude Code with role-specific CLAUDE.md files in different folders. Spawn sub-agents from a master orchestrator session.
- **Advanced:** Always-on agents on a Mac Mini or VPS. Each runs as its own service. Telegram or webhook frontend routes to them.

> **If you remember one thing from this hack: don't try to build the operator stack first. Build ONE solo agent. Live with it for a week. Add the next. Compound from there.**

The Stack, tied together.

You've now seen all five layers. Here's what they look like working as one system:

A normal Tuesday morning, 30 seconds end to end

```
I send a voice memo (HACK 2 — VOICE)
↓
Whisper transcribes, dispatches to the right agent (HACK 5 — SUB-AGENTS)
↓
The agent runs Claude Code on my machine (HACK 3 — CLAUDE CODE)
with my preferences loaded from memory (HACK 3 — MEMORY)
↓
It triggers the meeting-prep skill (HACK 1 — SKILLS)
↓
It writes the output to an artifact I built (HACK 4 — CLAUDECEPTION)
that saves it permanently for later review (PERSISTENT STORAGE)
↓
The result lands on my phone before I sit down for coffee.
```

This is what "AI native" actually means. Not using Claude. **Running on Claude.**

Every layer in this stack uses skills you can build today. Every prompt in this guide is real. Every template can be adapted to your work in 10 minutes.

The only thing between where you are and where I am is sequence. Build them in order. Layer by layer.

Cheat Sheet — every prompt, one place.

Skim this page. Bookmark this page. Build from this page.

HACK 1 — Skills

Skill build template

```
Build me a skill called "[name]".
Trigger: [phrases that activate it]
Behavior: [step-by-step instructions]
Output: [exact format spec]
Constraints: [length, tone, what to avoid]
```

HACK 2 — Voice

Voice memo template

```
[Action verb] + [specific input] + [specific output] + [constraint].

Example: "Look at my calendar tomorrow morning. For each meeting
before noon, run my meeting-prep skill on it. Put the briefs in
a doc on my desktop called Tomorrow Briefs."
```

HACK 3 — Memory

Memory template (sections)

```
# CLAUDE.md
## About me
## My team / network
## How I want you to work with me
## Tools and stack
## Currently working on
## Pet peeves
```

HACK 4 — Claudeception

Build prompt template

```
Build me an artifact called "[name]".
WHAT IT DOES: [input → action → output]
BEHAVIOR: [API call spec]
PERSISTENT STORAGE: [storage key + Vault tab]
SYSTEM PROMPT: [voice + constraints]
STYLE: [light/dark, layout, animation]
```

HACK 5 — Sub-Agents

Agent CLAUDE.md template

```
# [Agent name]
## Role
## How you work
## What you know about me
## What you don't do
## Skills attached
```

Your first 30 days.

If you're going to build the stack, here's the order that works. I've watched dozens of people try this. The ones who succeed do it in this sequence.

Week 1 — Skills (2-3 hours total)

- Day 1: Build Meeting Prep skill. Use it 3 times.
- Day 3: Build Daily Review skill.
- Day 5: Build one custom skill from your own recurring task.
- Day 7: Audit. Edit any skill that didn't fire correctly.

Week 2 — Voice (1 hour + ongoing usage)

- Day 8: Set up Dispatch. Send your first voice memo task.
- Day 9-14: Use voice memos for at least one task per day. Learn what works.

Week 3 — Claude Code + Memory (3-4 hours)

- Day 15: Install Claude Code if you haven't. Run one non-coding task.
- Day 16: Write your global CLAUDE.md. Use the template.
- Day 17-21: Test it daily. Tune the file as you find drift.

Week 4 — Claudeception + first agent (3-5 hours)

- Day 22: Build one Claudeception artifact for your most repeated task.
- Day 24: Set up your first dedicated agent (Strategist or Editor recommended).
- Day 28: Audit the full stack. What compounds? What's redundant?
- Day 30: Document your stack. Share with one person. Teach to learn.

After day 30

If you've followed this, you have 3-5 skills, voice working, Memory tuned, one Claudeception app, and one dedicated agent. You're now 90% of the way to the operator-tier stack. The next agents are layered on the same foundation.

THAT'S THE STACK.

Now build it.

If you build something with this guide, send it. I read everything that comes in.

MILES DEUTSCHER
Founder & CEO, AI Edge

Want the next video — the full TG voice pipeline build?

Drop "voice" in the comments on the YouTube video.

Want more operator-tier content?

Subscribe to AI Edge on YouTube. New stack, new system, every week.