The 100 Sub-Agent Setup for Claude Code

Fan one task across a hundred Claude sub-agents, score every result, and stop re-prompting.

Why I stopped re-prompting

I used to work the way most people do. Ask Claude for something, read a flat first draft, type "make it punchier", read a slightly different flat draft, type "try again but shorter". Forty minutes later I had something I was tired of rather than something I liked.

The fix was not a better prompt. It was running the task many times at once with a different angle in each run, then letting Claude grade them. Claude Code can do that natively, because a sub-agent is a separate agent instance with its own context window, and the main session can launch a lot of them from one instruction.

This guide is the setup I use. Two small files, one launching prompt, and a scoring rule. It works in Claude Code, and it works in Codex with one extra line of wiring. Nothing here needs a paid tool beyond the Claude subscription you already have.

01

How it actually works

1. You give Claude one task and a number. Claude splits the task into that many slices, where each slice is a different angle, audience, constraint or format rather than a repeat of the same request.
2. Claude launches a sub-agent per slice. Each one starts with a clean context window, sees only the instructions it was handed, does the work alone, and returns a short structured result. They cannot see each other, which is exactly why the answers come back genuinely different.
3. Claude collects every result, scores them against the criteria you set when you launched the run, and hands you a ranked shortlist with the reasoning for each winner. You pick from the top, or you tell Claude to run a second fan-out on the winner.

02

The one thing almost everyone gets wrong

People ask for a hundred agents and give all hundred the same instruction. You get a hundred versions of one sentence and a bill a hundred times the size. The count is not what buys you quality. The difference between the slices is.

> **WATCH OUT**
>
> Write the scoring criteria into the same prompt that launches the fan-out. If you add them afterwards, Claude grades on whatever it guessed you wanted, and you are back in the re-prompting loop you were trying to leave.

The second thing people miss: the agents do not all fire at the same instant. Claude Code issues them in batches and works through the queue, so a hundred-agent run finishes in waves. Plan for minutes, not seconds, and start at ten while you are still tuning the slices.

03

The files

A sub-agent is a markdown file with YAML front matter. Put reusable ones in **~/.claude/agents/** so every project can reach them, or in **.claude/agents/** inside a project when the agent needs to know that codebase. The worker is the agent that does one slice. Keep its output format rigid, because the scorer has to compare a hundred of them.

~/.CLAUDE/AGENTS/ANGLE-WORKER.MD

```
---
name: angle-worker
description: Produces one version of a creative asset from a single assigned angle. Use
when fanning one task across many parallel attempts.
tools: Read, Grep
model: sonnet
---

You produce exactly one version of the asset you are given,
from exactly one angle, and nothing else.

You will receive:
- TASK: what to write or build
- ANGLE: the single lens you must work through
- CONSTRAINTS: hard limits (length, format, banned words)

Rules:
1. Commit fully to your ANGLE. Do not hedge toward the obvious version.
2. Obey every CONSTRAINT literally.
3. Do not explain yourself, apologise, or offer alternatives.

Return this and nothing else:

ANGLE: <your angle in four words>
OUTPUT: <the asset>
RISK: <the one reason this version could fail, in one line>
```

The judge is the second file. It never writes the asset, it only ranks. Keeping scoring in its own agent stops the main session from quietly rewriting the winners into its own house style, which it will do if you let it.

~/.CLAUDE/AGENTS/JUDGE.MD

```
---
name: judge
description: Scores a batch of parallel sub-agent outputs against fixed criteria and
returns a ranked shortlist. Never rewrites the entries.
tools: Read
model: sonnet
---

You rank. You never rewrite.

You will receive a batch of entries, each with an ANGLE,
an OUTPUT and a RISK, plus a list of CRITERIA.

Score every entry out of 10 on each criterion.
Break ties on specificity: the entry naming a concrete
thing beats the entry naming a category.

Disqualify any entry that:
- breaks a stated constraint
- repeats another entry's angle in different words
- is generic enough to belong to any brand

Return only:

1. <score> | <angle> | <output verbatim> | <why it won>
2. ...
3. ...

Then one line: DISQUALIFIED: <count> and the most common reason.
```

That is the whole library. Two files. Everything else is the prompt you type.

THE FAN-OUT PROMPT (THE ONE THAT DOES THE WORK)

```
Use the angle-worker sub-agent to run [NUMBER] parallel attempts at this task.

TASK: [WHAT YOU WANT MADE]
CONSTRAINTS: [LENGTH, FORMAT, BANNED WORDS, AUDIENCE]

Before you launch anything, write me the list of [NUMBER] angles
you are going to assign, one line each. No two may share a lens.
If you cannot find [NUMBER] genuinely different angles, say so
and give me the number you can actually fill.

Then launch one angle-worker per angle, in parallel.

When they all return, pass the full batch to the judge sub-agent
with these CRITERIA: [CRITERION 1], [CRITERION 2], [CRITERION 3].

Give me the judge's top five verbatim. Do not edit them.
```

Set it up in Claude Code or Codex

1. Make the folder. In your terminal, run **mkdir -p ~/.claude/agents**. That path holds agents you want available in every project. For agents that only make sense inside one repo, use **.claude/agents/** in that repo instead.
2. Create the two files. Save the angle-worker block above as **~/.claude/agents/angle-worker.md** and the judge block as **~/.claude/agents/judge.md**. Paste them exactly, front matter included, starting with the three dashes on line one.
3. Match the names. The **name:** value inside each file is what you reference in a prompt. Keep the filename and the name identical (angle-worker.md holds name: angle-worker). A mismatch is the single most common reason a run comes back saying the agent does not exist.
4. Write real descriptions. The **description:** field is what the main session reads when deciding which agent to hand a job to. Vague descriptions get ignored. Say what the agent does and when to use it, in one sentence.
5. Restart your session. Open a new Claude Code session so it picks up the new agent files. Then ask it plainly: **list the sub-agents you can see**. Both names should come back. If they do not, you are in the wrong folder or the front matter is malformed.
6. Run a small first test. Use the fan-out prompt with [NUMBER] set to 5 and a throwaway task, something like five Instagram hooks for a post about morning routines. Read the five angles Claude proposes before it launches anything. If the angles already look samey, fix the task description, not the agent count.
7. Wire it into Codex if you use it. Keep the same two markdown files in your project under **.codex/skills/** or a plain **agents/** folder, then add a line to your **AGENTS.md** pointing at them, for example: **Agent personas live in ./agents/. Use angle-worker for parallel attempts and judge for ranking.** The fan-out prompt itself does not change.
8. Scale the count. Once five slices come back genuinely different, go to twenty, then fifty. Treat a hundred as the ceiling for tasks with a lot of real variation, like hooks, offers or names, and stay low for anything with one right answer.

> 💡 **QUICK TIPS**
>
> - Ask Claude to print the angle list before it launches. Thirty seconds of reading saves a run you were going to throw away.
> - Set **model: sonnet** on the worker and keep the heavier model for the judge. You are paying for the worker a hundred times and the judge once.
> - Never point many agents at the same output file. Either let them return text only, as these two do, or give each one its own path.

04

The commands I actually run

HOOKS FOR A SHORT-FORM VIDEO

```
Use the angle-worker sub-agent to write [NUMBER] hooks for a video about [TOPIC].

CONSTRAINTS: under 12 words, spoken out loud, no questions,
no "in this video", must work with the sound off.

List the angles first. One must be a loss, one must be a number,
one must be a confession, one must be a contradiction.
The rest are yours.

Then fan out, then judge on: stops the scroll, specific to
[AUDIENCE], survives being read as a caption.
Top five verbatim.
```

POSITIONING LINES FOR AN OFFER

```
Fan [NUMBER] angle-worker agents across one task:
write the one-line positioning statement for [OFFER].

Each agent takes a different buyer objection as its ANGLE.
List the objections before launching so I can see the spread.

CONSTRAINTS: one sentence, names the outcome not the feature,
no superlatives, readable by someone who has never heard of me.

Judge on: believable, specific, impossible to say about a competitor.
```

REVIEW ONE FILE FROM SEVERAL ANGLES AT ONCE

```
Launch [NUMBER] angle-worker agents on [FILE PATH].

Assign each a different review lens: correctness, failure modes,
naming, dead code, performance, readability for someone new,
and what a hostile reviewer would say.

Each returns its single most important finding with the line number.

Then judge on: would this actually break something, and how cheap
is the fix. Give me the top three only.
```

RESEARCH SWEEP WITH NO OVERLAP

```
Use [NUMBER] angle-worker agents to research [TOPIC].

Assign each one a different source type: official docs, a named
competitor, a community thread, a pricing page, a changelog,
a critic. Print the assignment list first.

Each returns three facts with where it found them, and flags
anything it could not verify.

Judge on: verifiable, recent, not already obvious to me.
Then give me the contradictions between agents as a separate list.
```

SECOND PASS ON A WINNER

```
Take entry number [N] from the last judge output.

Fan [NUMBER] angle-worker agents across it, but this time every
angle is a variation on that one entry: tighter, longer, blunter,
warmer, more specific, with a number in it, with the subject
changed, read aloud.

Same CONSTRAINTS as before. Judge on the same CRITERIA.
Show me the original alongside the top three so I can compare.
```

When it breaks

ERROR	FIX
Claude says the agent type does not exist	The name: field inside the file does not match what you typed. Open the file, read the name: value, and use that exact string. Filename and name should be identical.
All the outputs look the same	The slices were not different. Make Claude print the angle list before it launches and reject the list if two angles rhyme. Fix the task description, not the agent count.
The run stalls and nothing comes back for minutes	Normal. Agents go out in batches and the queue works through them in waves. Drop the count to ten, confirm the shape is right, then scale back up.
Usage limit hit halfway through a run	Every agent is a full context window being filled. Lower the count, set model: sonnet on the worker, and keep the expensive model for the judge only.
Agents overwrite each other's work	Two or more agents were given the same output file. These two agents return text only, which avoids it. If yours must write files, give each one its own path.
Claude returns a summary with no ranking	The scoring criteria were missing from the launching prompt. They have to be in the same message that starts the fan-out, not added afterwards.
A sub-agent ignores its instructions	The description: field is too vague, so the main session handed it a job it was not built for. Rewrite the description to say what it does and when to use it.
Codex cannot see the agents	Codex does not read ~/.claude/agents/. Keep copies in the project and add a line to AGENTS.md pointing at the folder and naming each agent.

Ready to go deeper?

This playbook is the short version. Inside the hub you get the full build walkthroughs, the prompt library, and the systems I use to ship this in a weekend.

Join the Build →

https://hub.digicuratoragency.com/join