Skills, Subagents and AI Agents

Three words get mixed up constantly: a **skill**, a **subagent** and an **AI agent**. This guide defines each one, shows how to decide which to build, and walks through examples from six industries. It also covers the case most teams miss: **neither**. Every example is self-contained and you do not need the class video.

1. The three definitions in one sentence each

Term	One-sentence definition	Analogy
Skill	A saved set of instructions, examples and sometimes scripts that the model loads when a task matches, so you do not retype them.	A recipe card in your kitchen drawer.
Subagent	A helper model run inside a larger task, with its own separate context, usually for one focused job, and it returns a result to the main conversation.	A specialist you hand one question to, who comes back with an answer.
AI agent	A model in a loop that decides its next step, calls tools (search, read a file, send an email), looks at the result, and repeats until the goal is met or it must stop.	A junior employee who can take actions and must be supervised.

2. The key difference

- **A skill is knowledge you reuse.** It does not act on its own. You, or the model, trigger it, and it shapes the answer. It costs tokens only when loaded.
- **A subagent is a worker you delegate to.** It runs a separate task with its own context window, so the main conversation stays clean. It is useful when a job would fill the main context.
- **An agent is a decision-maker that acts.** It chooses actions, calls tools, and loops. It is the most powerful and the most risky, because each step can change something in the real world.

> **Rule of thumb:** if it only needs the right instructions, it is a **skill**. If it needs to do one bounded job on its own, it is a **subagent**. If it must choose its own steps and use tools toward a goal, it is an **agent**. If none of those fits, it is **neither**, and a plain prompt or a normal script is the better answer.

3. The decision flow

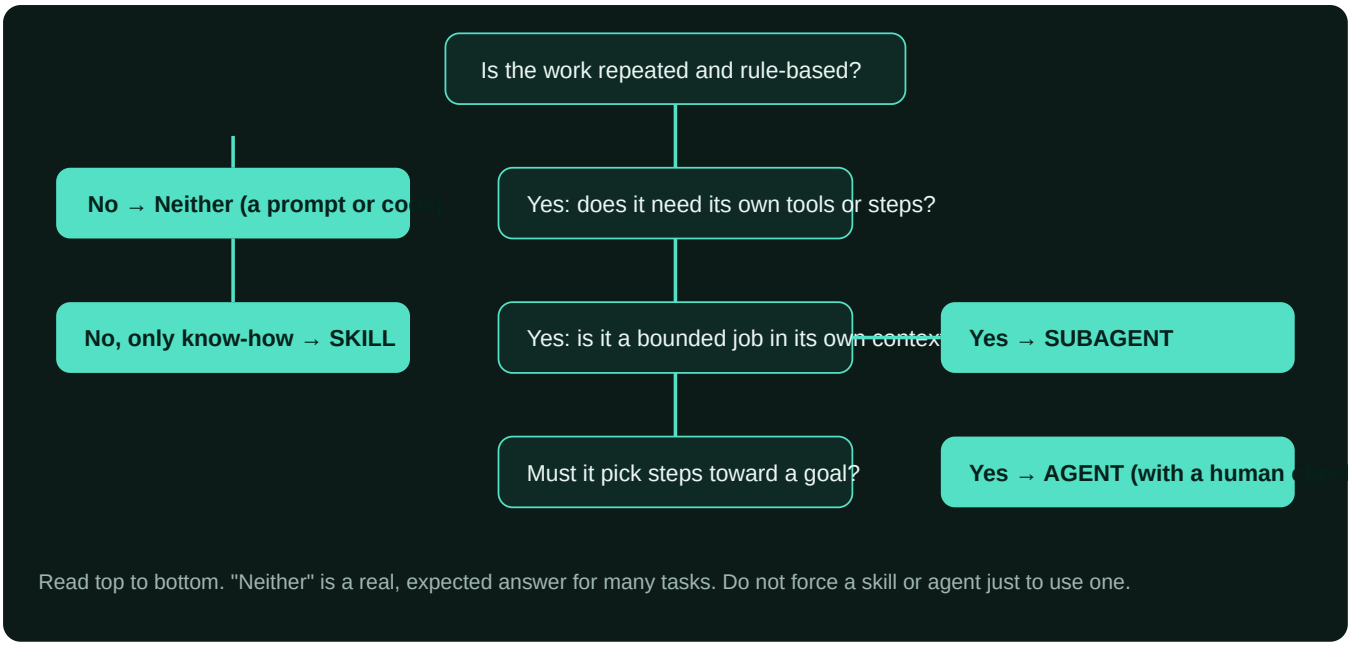


Figure 1. Decision flow. Start with the simplest option and move right or down only when you must.

4. How to create a skill (step by step)

A skill is a folder with one main file, `SKILL.md`. The file starts with a short header that says what the skill is for and when to use it, followed by the instructions.

1. **Pick one repeated task.** Example: "write a product requirements document in our house style". If you do it weekly, it is a skill candidate.
2. **Write the header.** Name, one-line description, and when to use it. The description is how the model decides to load it, so be specific.
3. **Write the instructions.** Role, steps, the required output sections in order, the tone, and what to avoid.
4. **Add two or three examples.** One good input and its ideal output. Examples do more than long rules.
5. **Add a checklist.** The final check before the answer is returned, such as "all required sections present, no invented numbers".
6. **Test on five real inputs**, including one difficult case. Fix the instructions, not the output, so the fix repeats.
7. **Save it** in your skills area (for Claude, under the skills settings) and invoke it by name or by matching request.

```
---
name: prd-copilot
description: Helps a product manager draft a PRD section by section. Use when the
```

```
user asks for a PRD, requirements or product spec.
---
# Role
You are a PRD co-pilot. Draft one section at a time, ask clarifying questions where
depth is missing, and flag gaps before the document is finished.

# Sections, in this order
1. Problem statement (3-5 sentences, no bullet points)
2. Pain points (each tied to a user segment and evidence)
3. Goals and success metrics
4. Scope and non-goals
5. User flow
6. Requirements
7. Risks and open questions

# Checklist before you answer
- Every section present and in order
- No invented numbers; mark assumptions clearly
```

The skill runs only when needed, so it costs tokens on those requests only. It replaces the long prompt you would otherwise paste every time, which is the main token saving.

5. How to create a subagent

A subagent is a named helper with its own description, instructions and allowed tools. Create one when the job is self-contained and would clutter the main conversation, for example "read 40 pages of contracts and list every clause that mentions liability".

1. **Name it for the job**, for example `contract-liability-reviewer`.
2. **Write what it receives and what it returns.** Input: one contract. Output: a table with clause, page and risk level.
3. **Limit its tools.** A reviewer needs read access only. Do not give it write or send permissions.
4. **Ask the main conversation to delegate** that job, and to use only the returned table.

6. How to create an AI agent

An agent needs four things: a goal, a set of tools, a decision rule, and a stopping rule. This is the most important part of the guide.

1. **Define the goal in one sentence**, with a measurable finish, for example "reply to a support ticket with an approved answer, or hand it to a human within 2 minutes".
2. **List the tools, with the least permission each one needs.** Read the knowledge base: yes. Send an email: only after human approval.

3. **Write the decision rule.** "If the ticket mentions a refund over the policy limit, do not answer; route to a human."
4. **Write the stopping rule.** Maximum steps (for example 6), maximum spend, and what counts as done.
5. **Add a human approval gate** before any action that is hard to undo: sending money, sending external messages, changing access, deleting data.
6. **Log every step** (input, tool called, result, decision) so you can explain and debug each run.
7. **Test with fake data first,** then a dedicated test account, then a limited live pilot.

> **Safety for agents:** never give an agent your personal email, your main drive or real customer data while you are learning. Use a test account, minimum permissions and approval before anything external.

7. Worked examples across six industries

Each example shows the right choice and why. The labels "Skill", "Subagent", "Agent" and "Neither" follow the decision flow.

Example A: Retail (e-commerce product descriptions)

Task	Choice	Why
Write a product description in the brand voice	Skill	Same tone and format every time; no tools needed.
Check stock and price, then update the listing	Agent	Chooses steps, reads inventory, writes to the store, needs approval before publishing.
Count the words in a description	Neither	A simple script does this exactly and cheaply.

Example B: Banking and fintech (loan document checks)

Task	Choice	Why
Explain a loan term in plain language	Skill	Fixed style and definitions; no action taken.
Read 80 pages of a loan file and list missing documents	Subagent	A bounded job that would fill the main context.
Chase missing documents, update the file and notify the customer	Agent (human approval before the message)	Multi-step, uses tools, changes records and contacts customers.
Calculate EMI from a rate and tenure	Neither	An exact formula belongs in code, not in a model.

Example C: Healthcare (appointment support)

Task	Choice	Why
Draft a patient-friendly reminder in a fixed, approved style	Skill	Repeated wording, reviewed once by staff.
Summarise a 30-page discharge summary into a checklist	Subagent	Single focused job; reads, returns a table.
Book, move or cancel appointments	Agent (staff approval for cancellations)	Chooses slots and changes a live calendar.
Decide whether a symptom is urgent	Neither (route to a clinician)	Clinical judgement must stay with a qualified person; an AI must not decide alone.

Example D: Education (course and assessment)

Task	Choice	Why
Generate quiz questions in the course format	Skill	Same format each time; the quiz generator skill is one example.
Grade 200 short answers against a rubric	Subagent (one per batch)	Bounded batches keep each context small.
Build a personal study plan from a learner's progress	Agent	Reads progress, picks next lessons, schedules reminders.
Assign a final grade	Neither (teacher decides)	High-stakes judgement belongs to a person.

Example E: Logistics (shipment exceptions)

Task	Choice	Why
Write the standard delay message in the company tone	Skill	Fixed template and tone.
Read carrier tracking logs and find the failed scan	Subagent	Bounded read-and-report job.
Detect a late shipment, reroute it and notify the buyer	Agent (human approves rerouting cost)	Multi-step with money and external messages.
Compute the distance between two warehouses	Neither	A map or distance API is exact.

Example F: Public sector (citizen service requests)

Task	Choice	Why
Explain an application form in simple language	Skill	Approved wording; no action.
Read a request and list which department it belongs to	Subagent	One clear classification job.
Route, log and send status updates across departments	Agent (officer approval for decisions)	Tools, records and communication; a person owns each decision.
Decide eligibility for a benefit	Neither (rule engine plus officer)	Eligibility rules must be auditable and deterministic.

8. Build a skill and an agent: a full walkthrough

The Support Ops case below is the exercise to practise on. Use it to build one skill and one agent, each working on its own.

> **The problem.** A SaaS company handles support through one shared inbox. Five capabilities are needed: (1) read, categorise and route new tickets within minutes, escalating outages and legal or security keywords immediately; (2) make every outbound reply match the company tone and never state a refund or price that is not in the current policy; (3) for an at-risk customer, combine billing, usage logs and past tickets into a one-page briefing before a retention call; (4) grade a weekly sample of closed tickets against a quality rubric and summarise recurring failure patterns; (5) when a customer reports a bug, check the changelog for a known issue and, if none, write it up for engineering with repro steps.

Step 1: map each capability

Capability	Choice	Justification
1. Categorise and route tickets	Agent	Chooses a team from several tools and must escalate immediately; the action (routing) is repeated and changes the workflow. Keep escalation rules in code where possible.
2. Tone-matched, policy-safe replies	Skill (with a check)	Same tone and rules each time; no action taken. The policy check should read the current policy file, not a memorised number.
3. At-risk account briefing	Subagent	A bounded job across three systems that would fill the main context; returns one page.

4. Weekly quality grading	Subagent (batch) plus a skill for the rubric	Batch grading is bounded; the rubric is reusable knowledge.
5. Bug triage against the changelog	Agent, human review before engineering	Searches the changelog, decides whether it is new, and writes a report; a person checks before filing.

Note: no capability needed "neither" here, but step 2's policy number and step 1's escalation rule should be code or data, not model memory. That is the "neither" part of the design.

Step 2: build the skill (reply tone and policy)

```
---
name: support-reply
description: Drafts a customer support reply in the company tone. Use for any
outbound reply to a customer.
---
# Role
You write support replies for a SaaS company. Warm, direct, no jargon.

# Rules
- Never state a refund, discount or price that is not in the policy text provided in
the request.
- If the policy text is missing, write "I'll check this and get back to you" and flag
it.
- Maximum 120 words. One clear next step at the end.

# Examples
Input: "Can I get a refund for last month?" + policy: 14-day refunds
Output: "Thanks for reaching out. Refunds are available within 14 days of purchase,
and yours falls outside that window, so I can't offer one. I'd be glad to help you
with an upgrade or a credit instead."

# Checklist
- Policy figure matches the provided policy text
- No invented numbers
- Under 120 words
```

Test it: give it five real tickets, two with the policy text missing, one asking for a price that is not in the policy. Pass means: no invented figure and a flag where information is missing.

Step 3: build the agent (ticket routing)

1. **Goal:** route each new ticket to one team within 5 minutes, or escalate at once.
2. **Tools:** read the ticket (read only); read the team list (read only); write the route to the ticketing tool (write, but only the route field). No email access.

3. **Decision rule:** if the ticket contains an outage or a legal or security keyword from the escalation list, escalate immediately to the on-call person and stop. Otherwise choose the team that matches the topic.
4. **Stop rule:** a maximum of 4 steps. If no team fits with confidence above the threshold, route to "triage" and flag for a person.
5. **Approval:** escalations notify a person; routes are written automatically only within the rules.
6. **Log:** ticket id, keywords found, team chosen, confidence, and the reason in one sentence.

```
Agent: ticket-router
Goal: route each new ticket to exactly one team, or escalate.
Tools (least permission):
  - read_ticket        (read)
  - read_team_list     (read)
  - set_route_field    (write: route only)
Escalation keywords (checked first, in code): outage, down, data breach, legal, GDPR,
lawsuit
Stop: 4 steps maximum. Unsure → route = triage, flag = needs_human
Log: ticket_id, keywords, team, confidence, reason
```

Test it: send one normal ticket, one with the word "outage", one ambiguous ticket, and one with a broken tool. Pass means: the outage ticket escalates, the ambiguous one goes to triage, and the broken tool stops the run cleanly.

Step 4: check the skill and agent on their own

Each one is tested alone. Do not connect them into a pipeline for this exercise. The skill must produce correct replies on its own; the agent must route correctly on its own, with only the tools it needs.

9. Skills and agents in other tools

Tool	Reusable instructions	Delegated helper	Acting agent
Claude (chat, Desktop and Code)	Skills (SKILL.md) and project instructions	Subagents in Claude Code	Agents with tools, via Claude Code or an agent framework
Lovable	Project knowledge and reusable prompts	Not a separate concept	Connected workflows (for example to n8n) do the acting
ChatGPT	Custom GPTs and Projects with instructions and files	Varies by product	Actions and connected tools, with approval steps

Feature names change often. Check the current documentation for the tool you use before you rely on a name.

10. Common mistakes

- **Making an agent for a fixed template.** If the steps never change, a skill or code is cheaper and safer.
- **Giving an agent write access to everything.** Start read-only, and add one write permission at a time.
- **No stop rule.** An agent with no step limit can loop and spend.
- **Hiding the rules in one long prompt.** Split them: skill for know-how, code for exact rules, agent for choosing steps.
- **Skipping the "neither" answer.** Many tasks need only a prompt or a script.

11. Quick check

1. A team writes the same sales email in the company tone every week. What is it?
2. A model reads 60 invoices and returns a table of mismatches. What is it?
3. A model decides which of three systems to query, then sends a message. What is it?
4. Calculating tax from a fixed table. What is it?

Answers: 1. Skill. 2. Subagent. 3. Agent (with approval before sending). 4. Neither; use code.

12. Glossary and further reading

Term	Plain meaning
SKILL.md	The main file of a skill: header plus instructions.
Subagent	A focused helper with its own context that returns a result.
Agent	A model that chooses steps and uses tools toward a goal.
Tool	An action the agent can call, such as read a file or send a message.
Stop rule	The limit (steps, spend, time) that ends an agent run.
Human-in-the-loop	A person approves an action before it happens.

- Anthropic documentation on agents, tools and skills: docs.anthropic.com.
- Claude Code and skills documentation, with the current feature list; check before you build.

13. Extension: skills, projects and MCP in practice

Exercise A: write and test a skill (30 minutes)

1. Pick one task you repeat every week. Write the header (name, one-line description, when to use it).
2. Write the instructions in this order: role, rules, output sections, checklist.
3. Add two examples: one good input and its ideal output.
4. Test with five inputs, including one hard case and one missing-information case.
5. **Pass when:** every output has the required sections, no invented facts, and the missing-information case is flagged.
6. Version it: write "v1" at the top, and record each change with the reason.

Exercise B: project instructions

A project keeps standing instructions and reference files for one piece of work. Example instruction for a learning project: *"Tone: warm and direct. Audience: beginners. Always define a term before using it. Never invent prices. End each answer with one next step."* Test it with the same question asked two ways; the answers should share the tone and rules.

Exercise C: a safe MCP check

1. List the tool you want an agent to use, and what it can read or write.
2. Connect it to a test account with fake data only.
3. Ask the agent to read one record. Then ask it to write one test record.
4. **Pass when:** the read works, the write needs your approval, and each call appears in the log.

14. Diagram: choosing the right tool for the job

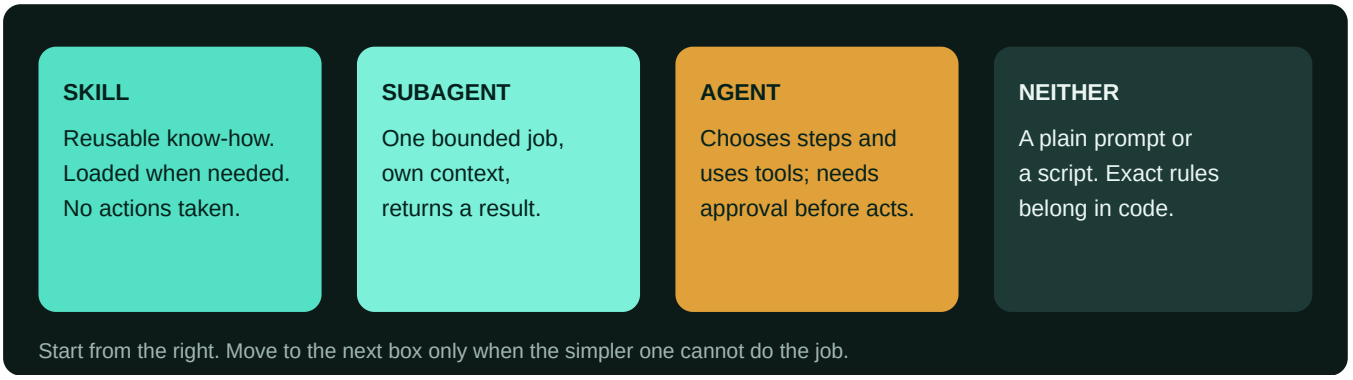


Figure 6. The four choices side by side.

15. Advanced: MCP safety checklist

Check	Pass when
Trusted server	You know who publishes it and read its permissions
Least privilege	Read-only unless a write is required for the task
Data scope	Only the test account and fake data are reachable
Approval boundary	Sends, deletes and access changes need a human
Audit log	Every tool call is recorded without personal data

16. Extended answer key and checklist

Item	Done
Skill written, tested on five inputs, versioned	[ ]
Project instructions tested with two phrasings	[ ]
Agent tools listed with least permission	[ ]
MCP safety checklist completed	[ ]

17. Create a skill from scratch: the full build

This walkthrough builds a skill called `meeting-notes-to-actions`. It turns meeting notes into owned, dated action items. It uses fake notes only.

1. **Define the job in one sentence:** "Turn messy meeting notes into a table of actions with an owner, a due date and a status."
2. **Define inputs and outputs.** Input: notes as plain text. Output: a table with columns action, owner, due date, status (open, done, reassigned, dropped), and a list of open questions.
3. **Write the header.** Name, one-line description, and the words a user would say to trigger it.
4. **Write the rules.** Never invent an owner or date; write "owner needed" instead. Use only the four statuses. Keep each action to one line.
5. **Add two examples.** One complete input and output, and one input with missing owners to show the "owner needed" rule.
6. **Add a checklist** the model runs before answering: every action has a status; no invented names; open questions listed.
7. **Test on five inputs:** a clean set, a messy set, a set with no owners, a set with a date in the past, and a set with a duplicate action.

8. **Version it.** Write "v1" in the header and note every change with the reason and the date.

```
---
name: meeting-notes-to-actions
description: Turns meeting notes into an action table with owner, due date and
status. Use when the user asks for action items, follow-ups or a summary of
commitments from notes.
version: 1
---
# Role
You convert meeting notes into owned, dated action items.

# Rules
- Never invent an owner or due date. Write "owner needed" or "date needed".
- Status is one of: open, done, reassigned, dropped.
- One action per row. Merge duplicates and say so.

# Output
1. Action table: action | owner | due date | status
2. Open questions (bullet list)

# Checklist before answering
- Every row has a status
- No invented names or dates
- Duplicates merged
```

18. Reuse, version and share

Practice	Why it matters
Keep the header accurate	The header decides when a skill is used
Version every change	You can see when an output changed and why
Keep examples current	Examples teach the format better than long rules
Store fake test cases with the skill	Re-testing after a change takes minutes
Remove the skill when unused	Unused skills still add context cost in some tools

19. Project instructions: examples

Project	Instruction (example)	What it prevents
Learning content	"Warm, direct tone. Beginner audience. Define each term before use. Never invent prices."	Jargon and invented figures

Customer replies	"Under 120 words. Never state a refund that is not in the policy text provided."	Unsupported promises
Internal reports	"Summary first. Each figure has a source line. Mark estimates clearly."	Unsourced numbers

Test a project instruction by asking the same question two ways. The answers should share the tone and the rules. If they differ, the instruction is too vague.

20. Claude.ai, Claude Desktop and Claude Code

Surface	Where you use it	Typical use	Skills and projects
Claude.ai	The web chat	Writing, analysis, quick questions	Projects with instructions and files
Claude Desktop	The desktop app	The same chat, with desktop features and connectors	Skills and projects, depending on your plan and settings
Claude Code	A terminal, editor or web session on a code project	Reading and changing code, running tests	Skills, subagents, and CLAUDE.md standing notes

Feature availability depends on plan and version. Check the current help pages before you plan around a feature.

21. Comparable concepts in other tools (and where they differ)

Idea	Claude	Lovable	ChatGPT	Difference to watch
Standing instructions	Project instructions, CLAUDE.md	Project knowledge and reusable prompts	Custom instructions, Projects, custom GPTs	Scope and sharing differ by product
Reusable workflow	Skill	Saved prompt or template in the project	Custom GPT	Skills can include scripts; check each product
Delegated helper	Subagent (Claude Code)	Not a separate concept	Varies by product	Do not assume equivalence
Acting on other apps	Agents with tools, via Claude Code or	Connected services	Actions and connected tools	Approval rules differ

	connectors			

The honest mapping is approximate. Use the product's own documentation before building a process that depends on one feature.

22. Decision drill: skill, subagent, agent or neither

Task	Choice	Reason
Write a weekly update in the company tone	Skill	Repeated know-how
Read 50 pages and list every penalty clause	Subagent	One bounded job that would fill the main context
Find a stuck ticket, reassign it and notify the owner	Agent (approval before notifying)	Chooses steps and acts
Add two prices	Neither	Exact arithmetic belongs in code

23. MCP safety in practice

1. **List each tool** the agent could call, with what it reads and writes.
2. **Connect a test account** that holds only fake data.
3. **Start read-only.** Ask the agent to read one record and confirm the result.
4. **Add a write only when needed,** and make it require approval.
5. **Review the audit log** after each session. Every call should appear.

24. Full quiz (answers follow)

1. A team reuses the same sales email format every week. What is it?
2. A subagent reads 80 invoices. Why not do it in the main chat?
3. Why should an agent have read-only access first?
4. What must every skill header contain?
5. Name the three surfaces discussed for Claude.

Answers: 1. A skill. 2. It would fill the main context and clutter later answers. 3. So a mistake cannot change data, and each added right is a deliberate choice. 4. A name and a one-line description of when to use it. 5. Claude.ai, Claude Desktop and Claude Code.

25. Full checklist

Item	Done
Skill written with header, rules, examples and checklist	[ ]
Skill tested on five inputs, including a hard case	[ ]
Version recorded	[ ]
Project instructions tested with two phrasings	[ ]
Agent tools listed with least permission	[ ]
MCP safety steps followed	[ ]