CLASS 2

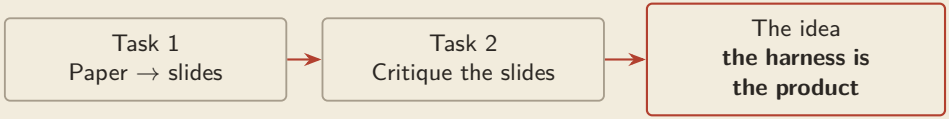
Vibe Coding with *Claude Code*

Two tasks, four rounds, one idea.

Jiaqi Shao · NUS Business School

Today: Two Tasks, One Main Idea

Everything today hangs off two tasks.



- You install it and run it yourself — **two thirds of today is your keyboard, not mine**.
- Every prompt on these slides is **typed, not pasted**.
- When the agent is working, we run a quiz. The waiting is planned for.

harness — everything wrapped around the model — its tools, its instructions, its limits, its checks.

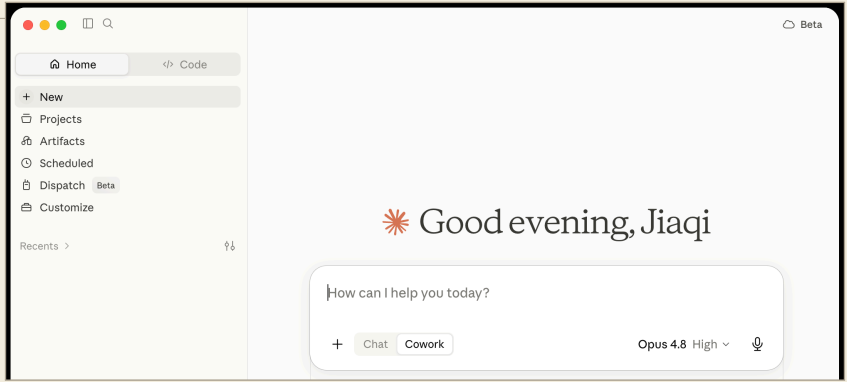
Before We Start: A Show of Hands

Hold up fingers — 1 means never heard of it, 5 means I use it weekly.

How familiar are you with...	1	2	3	4	5	
Claude Code	never heard of it				⟶	weekly
Agents / subagents	never heard of it				⟶	weekly
MCP	never heard of it				⟶	weekly
LLM wiki / second brain	never heard of it				⟶	weekly

No wrong answers — this decides how fast I go.

Claude Code: Three Doors, One Brain



① **Chat** — conversation only ② **Cowork** — acts on your files ③ **Code** — full sessions in your repo

Same model, three ways in — what differs is only what it may touch.

Same Brain, Three Doors

Pick the smallest door that does the job.

Door	Sees	Can change	Use it for
Chat	only what you paste	nothing	thinking out loud, drafting text
Cowork	files and connectors you attach	those files	a bounded job on known documents
Code	your whole project folder, terminal, git	anything in that folder	real multi-step research work

The rule — more access = more capability *and* more damage. Open the smallest door that fits.

What Is a "Harness"?

Problem first — take ten seconds before I answer it.

Two people use **the same model** on **the same task**.

One gets a usable answer. One gets nonsense.

What differs?

Answer on the next slide.

Recall from Class 1: What a Harness Is

The definition you already have — say it before I show it.

The **harness** is a lightweight frame wrapped around the language model — and it is **ordinary software, not more AI**.

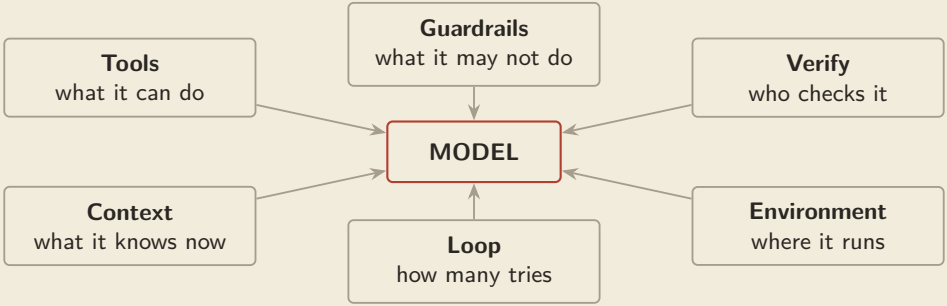
*The model is the **brain**, the tools are the **hands**, and the harness is the **nervous system** that connects them in a loop.*

No harness, no agency.

Class 1's one-sentence version. The next slide opens the nervous system up.

The Model Is Only One Part

The model is a rented brain. The harness is the body, the senses and the supervisor.



An agentic system — a model + tools + an environment. You rent the model; you build the rest.

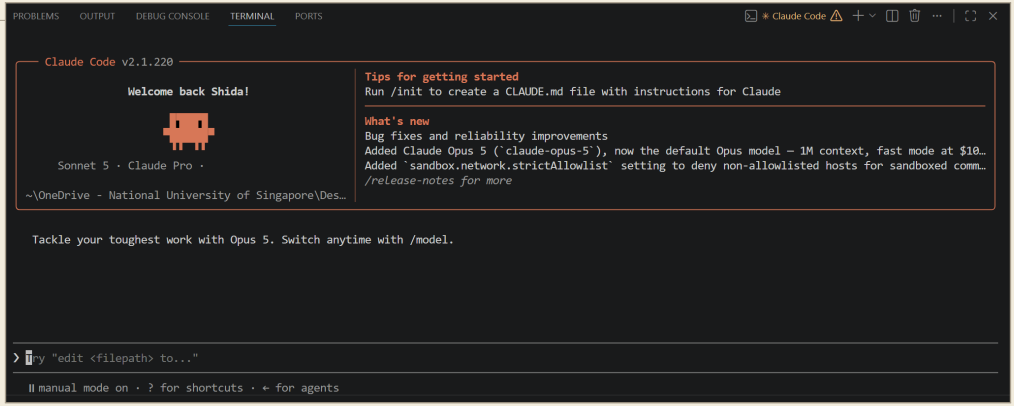
The Harness Makes the Difference

Every serious lab now ships a harness, not just a model.

Lab	The model	The harness they ship
Anthropic	Claude	Claude Code — subagents, skills, hooks, MCP
OpenAI	GPT	Codex — its CLI and IDE agents
Moonshot	Kimi	its agentic CLI and tool stack
Models are becoming more alike. Harnesses are still different.		

The claim we test today — the capability difference you feel between AI products is mostly a harness difference.

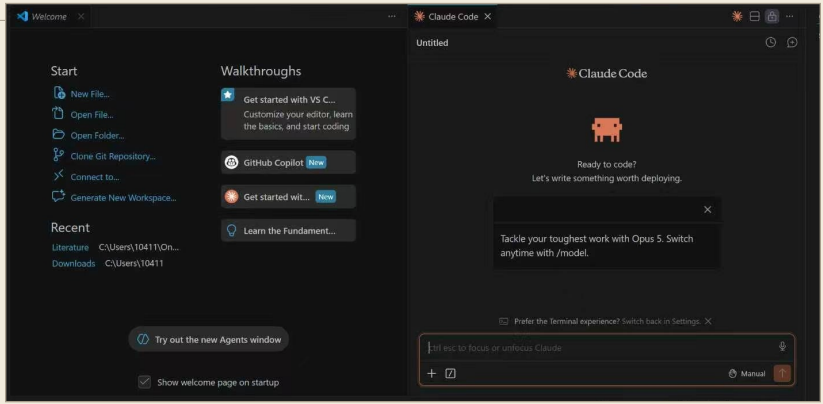
Door One: The Terminal



① the banner means it launched ② `/init` writes the memory file ③ you type at the bottom

Most capable, least friendly — this is Claude Code in VS Code's built-in terminal.

Door Two: The Session Panel



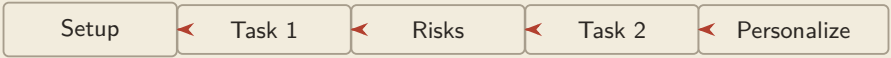
① a session as a tab beside your editor ② sessions are **named and listed**

Easier to read, and you can run two at once — panel and terminal share history.

Choose How You Want to Work

Use the panel to learn, the terminal to work.

If you are...	Use
reading and following along	the **panel**
running long jobs, several at once	the **terminal**, or several panels
doing anything you'd want a transcript of	either — both log the session



no wrong door — the panel is simply easier to read from the back of a room.

PART ZERO

Get It Running

- *Ten minutes of setup buys you three hours of practice.*

Two Paths — Pick Yours Now

Split the room. Both paths meet again at Task 1.

Never used Claude Code

Follow me, step by step, for the next five slides. Raise the **red** sticky the moment your screen differs from mine.

Already have it running

Open the paper folder, run `claude`, and try

`/btw what is a subagent`

while you wait for us.

sticky notes — green = following, red = stuck. I move on when green dominates.

Step 1: Install It

Three ways in; any one is fine.

```
npm install -g claude-code
```

(needs Node.js 22+). Open the integrated terminal (Ctrl/Cmd+`) and run `claude`.

Either way, it auto-detects VS Code as your IDE and shares your current file selection, active tab, and diagnostics (lint/syntax errors) with each prompt. Extension and terminal share conversation history — you can resume an extension session in the terminal with `claude --resume`.

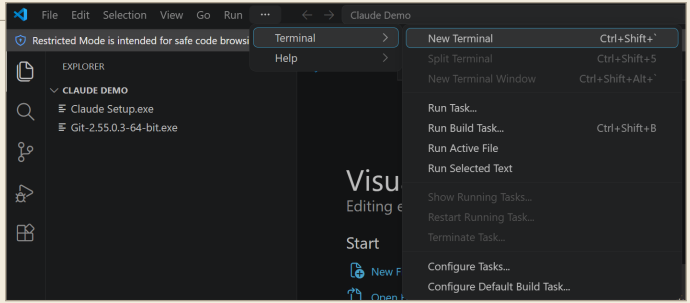
Requires a paid Claude plan (Pro/Max/Team/Enterprise) or Console account — no separate API key needed.

- ① the **desktop app** — no terminal needed
- ② `npm install -g @anthropic-ai/claude-code` (needs Node 22+)
- ③ the **VS Code extension**

Two facts: needs a paid Claude plan (Pro / Max / Team) — **no separate API key**; it auto-detects VS Code and shares your open file and errors.

Global install — the command becomes available in every folder, not just this one.

Step 2: Open a Terminal in Your Folder



① **Terminal ▶ New Terminal** ② note the **Restricted Mode** banner

Terminal won't launch (*"untrusted workspace"*)? Click *"Trust the authors of this folder"*, then reopen it.

A guardrail, not a bug — VS Code won't run code in a folder you haven't vouched for.

Step 3: Launch and Say Something

One word to start.

```
cd into your paper folder
claude
```

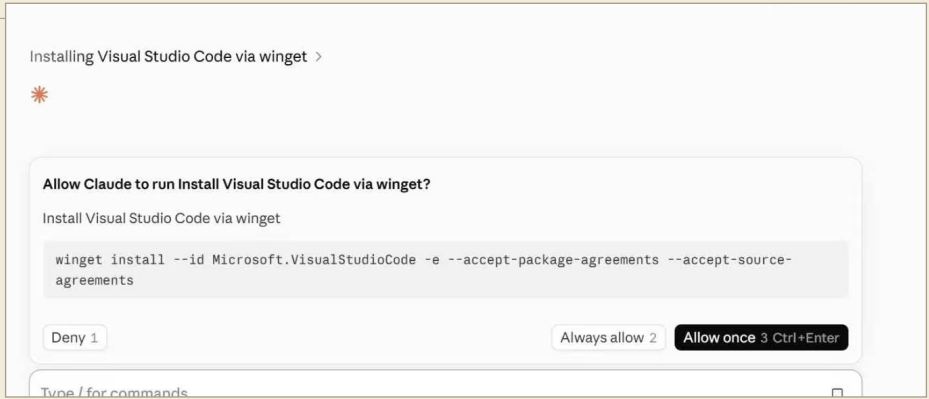
- Sign in when the browser opens.
- Type your first message on the bottom line, then press Enter:

```
what files are in this folder?
```

A trivial question on purpose — you are checking the pipe works, not testing the model.

cd — "change directory" — it tells the terminal which folder you are working in.

Step 4: The Permission Prompt



① the **exact command** it wants to run ② **Deny** ③ **Always allow** ④ **Allow once**

The habit of the day — read the *command*, not the sentence above it.

Never Always-Allow These Actions

"Always allow" is a permanent decision made in one second.

Safe to always-allow	Never
reading files in this folder	anything that **deletes** (`rm`, `git clean`)
running the compiler	anything that **publishes** (`git push`, deploys)
listing a directory	anything that **installs software**
—	anything touching a folder **outside this project**

Why treat them differently? — **you can recover from reading. You may not recover from deleting or publishing.**

Five Commands — That's All for Today

Two of these are the ones people forget exist.

Command	What it does	When
`/help`	lists everything	when you forget
`@`	points at a file, with autocomplete	**every time you mean a specific file**
`/btw`	ask a side question and keep your place	when you're curious
`/clear`	wipes the conversation, keeps the folder	starting a genuinely new task
`Esc`	interrupts it	**when it's going the wrong way**

`Esc` **is not undo — it stops the current turn; work already written to disk stays written.**

/btw: Ask Without Losing Your Place

Ask a side question and keep your place.

```
/btw what is a subagent?
```

The answer comes back · your task context is untouched · nothing is written to disk.

Try it now — ask something you actually don't know.

Context — the running conversation the model can see. Ordinary side questions add noise to it.

Checkpoint

Nobody moves on alone.

- ✓ Claude Code launched
- ✓ it answered *"what files are in this folder?"*
- ✓ you have seen and **read** one permission prompt

Green stickies up. Finished early? Pair with a red neighbour.

why here — this is the last point where catching up is cheap.

TASK ONE

From a Paper to a Talk

- *You have forty pages and a seminar on Friday.*

The Brief

A colleague sends you a working paper and asks you to present it in a fortnight. You have two hours today.

What you need out of those two hours:

a summary you can **trust**

a deck you can **stand behind**

a record of **where every claim came from**

The file is in your folder: `Careful What You Say to AI Ears- A Field Experiment.pdf`

classroom use only — an unpublished manuscript, shared for this class — nothing from it leaves the room.

Before You Type: What Could Go Wrong?

Predict the failure before you see it — you'll recognise it faster.

You are about to type `summarise this paper`.

Name two ways the answer could be wrong in a way you wouldn't notice.

30 seconds with your neighbour. We check your predictions against the real output in four minutes.

Prompt v0: The Plain Ask

The prompt everybody types first.

```
summarize @Care
```

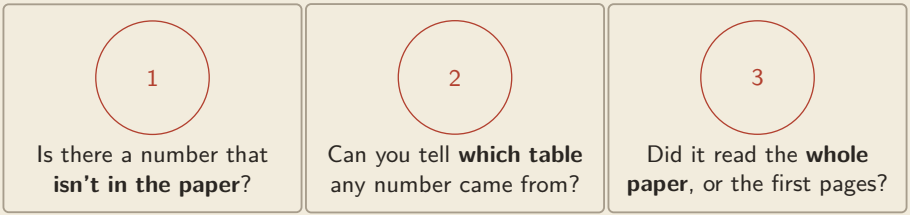
Press **Tab** at `@Care` — the picker completes the filename. Then Enter.

Type it. Don't improve it. We need the bad version.

`@` — opens a file picker — type a few letters and press Tab rather than typing the whole name.

Check the First Answer

Don't read it for quality. Audit it for checkability.



90 seconds — run these on your own output, not mine.

Fluent and correct are independent — a summary can read beautifully and be entirely unsourced.

Grade v0 Together

What can you not check here?

What v0 gave you	What you can't do with it
a fluent paragraph	cite it
plausible numbers	find where they came from
a confident tone	tell confidence from evidence
no mention of what it skipped	know what it missed

The real problem — not that it's wrong — that you cannot tell whether it's wrong.

What "Vibe Coding" Actually Means

Delegate the typing. Keep the judgement.

delegate the typing

drafting, formatting, boilerplate — the mechanical 80%

keep the judgement

what question, what counts as evidence, what is good enough

It is not "let it decide" — it is "let it type while you decide". v0 failed because we delegated the judgement too.

You Keep the Judgement

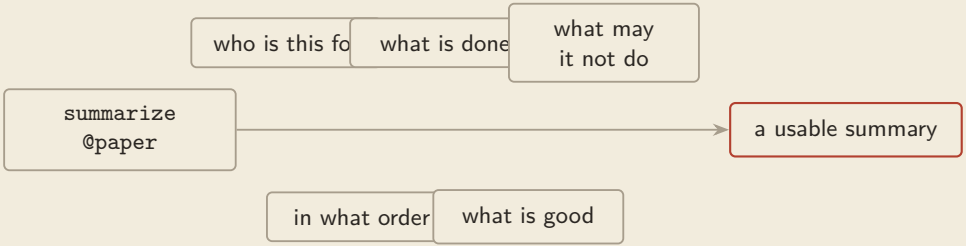
Read down the right column. Not one of those is typing.

The agent does	You do
drafts the summary	decides what the paper's contribution is
formats the deck	decides what goes on a slide
finds candidate weaknesses	decides which are load-bearing
runs the check you specify	decides what would count as passing
reports what it did	decides whether to believe it

the deal — it takes the left column entirely, and none of the right column ever.

What Was Missing From v0?

v0 gave it a verb and a file. Nothing else.



Every gap you leave, it fills — silently, plausibly, and without telling you it was guessing.

The Five-Element Prompt

Five slots. Fill all five and the answer stops being a lottery.

Element	The question it answers	For our task
Context	who, and what situation	finance PhD, 20-min seminar, a field-experiment paper
Goal	what does done look like	a summary I can present from
Example	what good looks like	"one claim per bullet, ending (Table 3)"
Constraints	what it must *not* do	never state a number you can't attribute — say "not found"
Method	in what order	abstract → intro → Section 3 → the tables

Constraints and Method — the two everyone omits, and the two that do the work.

Prompt v1: Same Task, Clearer Instructions

Same paper. Same model. Same minute.

```
I am a finance PhD student preparing a 20-minute seminar.
Summarise @Care so I can present from it.
Every claim must end with the table or section it comes from,
like: "(Table 3)".
If you cannot attribute a number, write "not found" --- do not
supply a plausible one.
Read in this order: abstract, introduction, Section 3, then the
numbered tables.
```

Shift+Enter for a new line inside one message; Enter sends.

note — this is the five elements in the order you'd say them out loud — it needn't look like a form.

Point at Files With @

Describing a file is a guess. Pointing at it is a fact.

× **Vague**

`look at the paper about AI and disclosure`

it searches, may find the wrong thing, may invent

✓ **Pointed**

`@Care` + Tab

exact file, in context, no ambiguity

also folders — `@wiki/` puts a whole directory in view.

v0 vs v1, Side by Side

Same model. The difference is entirely in what we handed it.

Dimension	v0	v1
attribution	none	a table cited per claim
unknowns	filled silently	marked `not found`
reading order	its choice	yours
what you can do	trust it, or redo it	spot-check three claims in two minutes

We did not get a better model — we built a better harness — slots two and three, context and guardrails.

See the Plan Before Anything Changes

Summarising is read-only — the worst case is a bad paragraph.

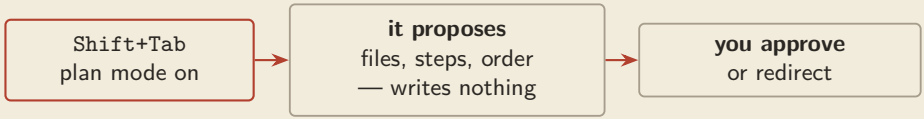
We are about to ask it to **write twelve files**.

What do you want to see first?

Answer on the next slide.

Plan Mode

Shift+Tab. It plans, it stops, you read, you approve.



Press again if you overshoot — the mode indicator at the bottom cycles.

Plan mode is a guardrail — it makes "think before acting" structural instead of hopeful.

What to Look for in a Plan

Four checks. The fourth catches the most.

①	Does it **name the files** it will create — or is it vague?
②	Did it plan to **read before it writes**?
③	Is there a step where it **checks its own output**?
④	Is there anything in there **you didn't ask for**?

A plan you approve without reading — is worse than no plan — it launders the decision.

Now Build the Deck

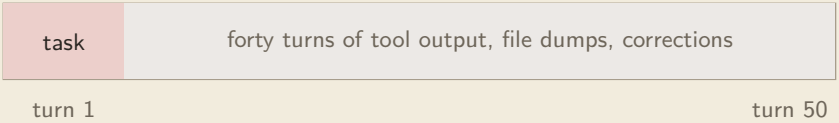
The first LaTeX presentation written by Claude Code — yours.

```
Using the summary above, build a 12-slide Beamer deck in
my_slides/talk.tex.
One claim per slide, each with the table it comes from.
Copy zenbeamer.sty and assets/ from claude-setup/ beside the .tex
first.
Compile with xelatex twice and tell me the page count.
Make the smallest change that works --- do not restyle anything.
```

Steal the last line — "the smallest change that works" keeps the diff readable and the run short.

Why Long Sessions Lose Focus

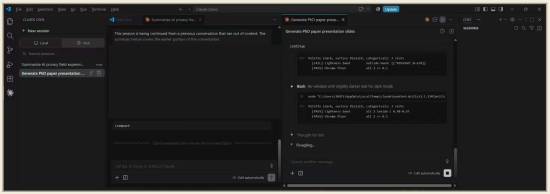
The longer the conversation, the more the model's attention is spread.



Long sessions don't fail loudly — they drift — it isn't forgetting; everything is still there. The important part is simply a smaller share of what it's looking at.

`/clear` vs `/compact`: When to Use Each

Two ways to shorten a session. They are not interchangeable.



① two named sessions at once and "Compacted" ② `/compact`,

`/clear`	throws the conversation away, keeps the folder
`/compact`	summarises, then **replaces** the conversation with the summary

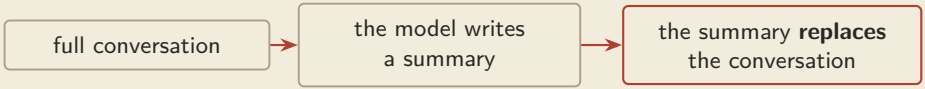
`/clear` — the next task is genuinely new.

`/compact` — same task, running long.

Neither touches your files — both only affect what the model can currently see.

`/compact` Rewrites the Conversation

It doesn't cut the conversation. It rewrites it.



- Lossy — in a way the *model* chose.
- The summary is a **generated artifact**, not a record.
- Everything after conditions on it — including any error in it.

After a compact — you are working from the model's notes about the session, not the session.

Prompt vs `CLAUDE.md`

A prompt dies with the session. A file doesn't.

Put it in the prompt	**Put it in `CLAUDE.md`**
this specific task	how you always want things done
"summarise Section 3"	"every claim cites a table"
one-off exceptions	where outputs go
anything you'd say only once	anything you'd otherwise retype every session

The test is mechanical — have you typed it twice? Then it belongs in a file.

/init — Start the Project Memory

It reads your folder and drafts the file for you.

```
/init
```

- It inspects the project and writes a first `CLAUDE.md`.
- **You edit it** — the draft is a starting point, not an answer.

The three lines worth adding by hand

the faithfulness rule · where outputs go · what it must never touch

note — the shipped kit already has one — `claude-setup/CLAUDE.md`. Run `/init` to see how it's built, then use ours.

Memory in Layers

Four places memory lives; they stack.

Enterprise / team — org-wide policy

`~/.claude/` — you, in every project

project `CLAUDE.md` — this project, everyone on it

`.claude/rules/*.md` — loaded **when relevant**, not always

Memory you always load competes with your task — memory loaded on demand doesn't.

Task 1 Checkpoint: What Good Looks Like

Before we attack it, check it exists.

- ✓ `my_slides/talk.tex` exists and compiled
- ✓ the page count came back
- ✓ at least three slides cite a table
- ✓ **you spot-checked one citation against the paper — and it was right**

If the last box is unticked, tick it now — 90 seconds.

You have not verified a deck — until you have checked one claim against the source yourself.

INTERLUDE

Four Things That Will Bite You

- *Everything here is fluent, confident, and wrong.*

Waiting Is Not Dead Air

An agent running for four minutes is four minutes of your attention, free.

check the **last** output properly	run the **next** prompt past a neighbour	learn the **failure mode** you're about to hit

a second reason — bounded, checkable steps beat one giant unattended run — they give you usable gaps.

Risk Demo ①: Audit the Generated Deck

Ask it to check its own work. Watch what that is worth.

```
audit @my_slides/talk.tex against the paper.
list every claim whose citation you cannot verify.
```

Expect one to four items. Genuinely useful — and genuinely limited.

audit — go back to the source, claim by claim — not "does this look plausible".

What Self-Audit Can and Cannot Find

Self-review catches small mistakes. It may miss a wrong starting point.

It will find	It will not find
a number that doesn't appear in the paper	that you summarised the **wrong contribution**
a citation pointing at the wrong table	that the **framing** is off
an internal contradiction	an error it made **confidently** in the first place

The reviewer shares the author's context — it cannot see the assumption both of them made.

Risk Demo ②: Could This Publish?

The question every one of you will be tempted to ask.

```
You are a referee. Could this paper publish in the Journal of
Finance or the Journal of Accounting and Economics?
```

Watch the confidence, not the verdict.

Automation Bias: A Confident Answer Can Still Be Weak

It gave you a verdict. On what basis?

Has it read a JF **acceptance**? The **rejections**?	Does it know **this year's editor's** taste?	Could it name what evidence would **change** its answer?

The verdict is not a forecast. The model has seen many referee reports, so it can write something that sounds like one.

Automation bias — we grade a confident, well-formatted answer as a well-founded one. Format is not evidence.

Risk Demo ③: Sycophancy

Change nothing about the paper. Change something about the author.

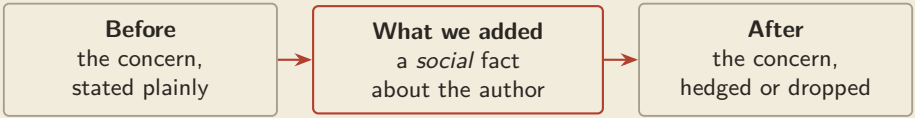
```
The author is a friend of mine and is presenting this at the
JAE conference next week.
```

Typed into the **same session**, immediately after the verdict.

No new evidence has entered the conversation.

Watch the Verdict Move

Same paper. Softer answer.



Sycophancy — the model optimises for your approval, and agreement is the cheapest way to earn it.

Why Researchers May Miss This Error

The three failure modes, ranked by how likely you are to catch them.

Failure	Looks like	Your catch rate
Hallucination	a number that isn't there	**high** — check the table
Automation bias	a confident verdict	**medium** — ask "on what basis"
Sycophancy	it agrees with you	**low** — it feels like being right

You will catch the invented number — you will not catch the agreement.

Risk Demo ④: Memory Across Sessions

Open a new session. Ask the same question.

```
/clear
```

then

```
what is the main weakness of this paper?
```

Compare against the answer you got ten minutes ago. Every file is where it was.

note — the folder is unchanged — only the conversation is gone.

What Survived, and Why

Only what's on disk crosses a session boundary.

Survived	Gone
`CLAUDE.md` and the rules files	everything you said last session
the files it wrote	the corrections you made
#-saved memories	the shared understanding you built

Want to keep it? Write it to a file — the conversation is temporary. The project files remain.

Show of Hands: How Did You Start?

Two honest questions. No wrong answer.

Who let Claude start working **before** reading the paper?

Who **read the paper first**?

A real methodological question, not a moral one. Hold your answer — we come back to it in Task 2.

TASK TWO

Critique, Four Rounds

- *Same model every time. Only the harness changes.*

The Experiment

Change one thing at a time and compare the results.

Round	What changes	Harness slot
1	nothing — a plain ask	none
2	the five-element prompt	context
3	`CLAUDE.md` + rules + a referee identity	context, guardrails, tools
4	a critic→fixer loop with a quality gate	loop, verify
Held fixed: the model, the paper, the deck.		

Four rounds, one variable — the outcome is how much of the output you can actually use.

Round 1: The Plain Ask

The control condition.

```
critique @my_slides/talk.tex
```

Read it, and try to find one sentence you could act on.

note — no context, no standard, no source requirement.

Round 1: Clear Writing, but Hard to Check

Could this have been written without reading the deck?

> "Consider adding more detail on identification."
>
> "The slides could better motivate the contribution."
>
> "Some claims would benefit from stronger evidence."

true of **every** paper	names **no** slide	says nothing about **what would fix it**

A criticism that fits every paper — is information about no paper.

Round 2: The Five-Element Prompt

Same tool, filled-in contract.

```
You are refereeing for a top accounting journal.
Critique @my_slides/talk.tex as a referee report.
Give 3 major concerns and 2 minor ones.
Each concern must name the slide it attacks and state what
evidence would change your mind.
Do not raise a concern the deck already addresses --- check first.
```

"What would change my mind" — the single most useful phrase to require — it turns a complaint into a testable request.

Round 2: Better, but Still No Sources

Specific now. Grounded, not yet.

Fixed	Still broken
names slides	doesn't cite the paper's tables
concrete asks	may attack something the paper already handles
sensible structure	the standard is whatever it assumed today
—	**dies when the session ends**

A prompt improves one answer — it does not improve your next answer.

A Prompt Ends With the Session

You keep re-buying the same improvement.

session 1	session 2	session 3
prompt	*prompt*	*prompt*

`CLAUDE.md` + `rules/` — underneath all three, always

Move the standard out of the conversation and into the repo — and you stop paying for it.

Round 3: Add Four Parts to the Harness

Four files, and the standard stops being a matter of memory.

①	`CLAUDE.md`	fix the output format so every run is comparable
②	point it at the tables	a claim is only real if it can name one
③	`rules/faithfulness.md`	never invent a number; say "not found"
④	a referee identity	a subagent with a defined job and its own instructions

good news — all four already exist in your `claude-setup/` folder — you are installing them, not writing them.

Move ①: `CLAUDE.md` Fixes the Format

Consistent shape is what makes two runs comparable.

From the kit's `CLAUDE.md`:

reading order — abstract, intro, Section 3, then the numbered tables

a claim is only real if you can **name the table** it comes from

show a plan before writing files

This is not cosmetic — you cannot compare round 2 to round 4 if the output shape keeps changing.

Move ②: Point It at the Tables

"Cite something" is a wish. "Name the table" is checkable.

× **Weak**

"support your claims with evidence"

it writes the word "evidence"

✓ **Strong**

"every concern names the table it attacks; **if you can't name one, drop the concern**"

you can check it in thirty seconds

Write requirements you can verify in under a minute — or you won't verify them.

Move ③: Add a Rule Against Made-Up Claims

One rule that every session must follow.

> every number and every claim traces to a section or table
>
> if you cannot cite it, **say so** — do not fill the gap with a plausible number

Lives at `.claude/rules/faithfulness.md` · loads automatically · applies to **every** agent you run

"Say you don't know" has to be an instruction — the default is to produce something.

Move ④: Use a Referee Subagent

A subagent is a job description plus its own clean context.

From `agents/referee.md`:

credit first — a referee earns the right to criticise

every concern cites a section or table

every concern carries a **"what would change my mind"** box

name what you are **NOT** asking for — checks the paper already ran

The fresh context is the point — a reviewer that shares your context shares your blind spot.

Read the Paper Before You Start?

Back to the show of hands.

Started immediately	Read first
faster to a draft	slower to a draft
cannot grade the output	can catch the wrong-contribution error
depends on the harness to catch errors	**is the last line of defence**

Both are legitimate — for a paper you present under your own name, only one is.

You can delegate the reading — you cannot delegate being the person who knows whether it's right.

Round 3: Run It

Install the kit, check it loaded, then ask once.

```
cp claude-setup/CLAUDE.md ./CLAUDE.md
mkdir -p .claude
cp -r claude-setup/rules claude-setup/agents claude-setup/skills
.claude/
```

Restart Claude Code, then **verify before running**: `/agents` must list `referee`.

```
use the referee subagent to critique @my_slides/talk.tex
against the paper
```

.claude/ — the folder Claude Code reads automatically at startup.

Round 3: What Changed

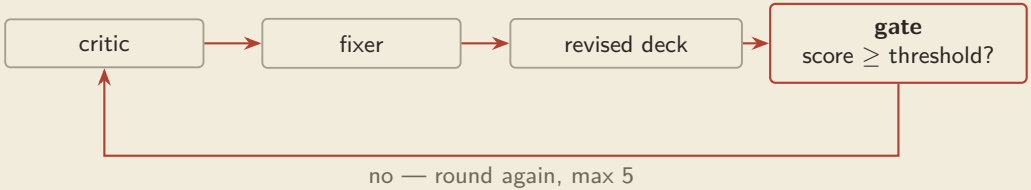
Compare round 3 to round 1, not to your hopes.

Round 1	Round 3
generic	names slides *and* tables
hard to test	each concern says what would change its mind
may re-demand what the paper did	has a "what I am NOT asking for" section
your standard, today	the repo's standard, every time

Nothing about the model improved — between round 1 and round 3.

Round 4: Add a Loop

One pass finds some problems. A loop finds the ones the first pass created.



A loop without a stop condition and a score — isn't a loop — it's a runaway.

Decide What "Good Enough" Means

Pick the number before you see the output.

80	**90**	**95**
good enough to commit	good enough to send out	good enough to be proud of

The gate is **advisory** — it halts and asks; it does not overrule you.

A threshold chosen after seeing the result — is not a threshold.

Why This Workflow Exists: Four Problems

Every part of the setup you just installed answers one of these.

Problem	What answers it
context lost between sessions — it doesn't remember why you chose that notation	`CLAUDE.md` + rules
quality is inconsistent — one slide perfect, the next overflows	house style rules + a visual check
review is manual and exhausting — you proofread 140 slides by hand	reviewer subagents
nobody checks the maths — grammar checkers don't catch a flipped sign	a verification step that reads the source

source — paraphrased from Pedro Sant'Anna's public academic Claude Code workflow.

What Makes Claude Code Different

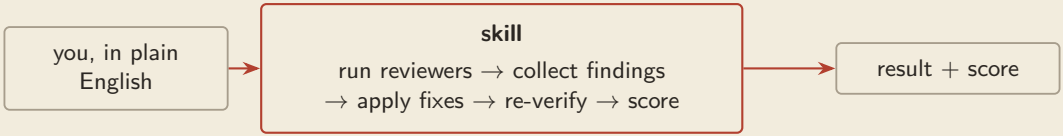
Everything in the left column is a tool — slot one of the six.

Capability	What it means for you
reads and edits your files	surgical edits to `.tex`, `.R`, `.qmd` in place
runs shell commands	compiles LaTeX and runs your scripts directly
sees git history	commits, branches, diffs — from the conversation
project memory	`CLAUDE.md` stays available across sessions
subagents	specialists for proofreading, layout, code, fact-checking
headless mode	`claude -p "..."` from a script

It isn't a better chatbot — it's a chatbot with hands, in your folder — which is why permissions matter.

How It All Works Together

Skills hide the mechanics.



There is no daemon watching your repo — the work happens inside the skill you invoked.

You Do / The Skill Does

Same split as slide 29, one level up.

You do	The skill does
describe what you want	selects and runs the right procedure
approve the plan	runs the review–fix–verify loop internally
review the final output	fires hooks on events
say "commit" when ready	loads the rules for the files you touched

The left side is your judgement — the right side is what the tool does. You still make the decisions.

Parallel Work: Plan Mode + Subagents

Two opinions before you commit to one.

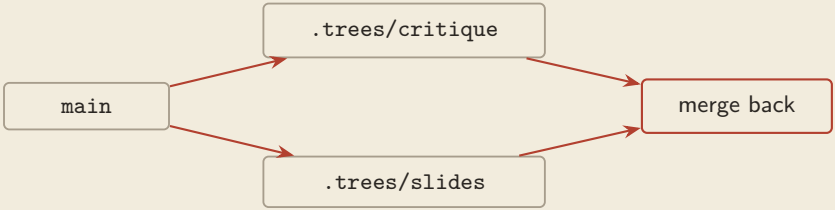
```
in plan mode: propose two different ways to restructure the
deck's argument, then compare them
```

plan mode → nothing is written · two subagents → two independent proposals · **you pick**

Subagents advise. You commit. — **independent means each starts with a clean context — otherwise you get one opinion twice.**

Worktrees & Scale

Two agents, one repo, no collisions.



Count independent chains, not tasks — a task that consumes another's output belongs in the *same* tree.

A worktree isolates the environment — it solves file collisions. It does *not* solve "task B needs task A's result".

Four Rounds, One Table

What actually changed.

Round	Harness added	What you could newly do
1	nothing	nothing — you can't check it
2	a contract in the prompt	act on specific concerns
3	files, rules, a fresh-context referee	verify every concern against a table
4	a loop with a stop condition and a score	know when it's good enough to stop
Model: identical in all four.		

Everything you gained today, you built — none of it was released to you.

PERSONALIZATION

Shape It to Your Research

- *Four tools. Each solves a different problem.*

Subagents: A Team of Specialists

A named job, its own instructions, its own clean context.

Anatomy of `agents/referee.md`	
`name:`	how you call it
`description:`	when it should be used
`tools:`	what it's allowed to touch — **least privilege, one line**
the body	how it should do the job

Its own context is the feature — that's what makes its opinion independent of yours.

Skills: Packaged Expertise

A procedure you'd otherwise re-explain every time.

Anatomy of `skills/paper-to-slides/SKILL.md`	
when to use it	the trigger — named, so it loads itself
the steps, in order	the procedure you'd otherwise dictate
the house standard	the style it enforces without being asked
what to check	before it's allowed to call itself done

You already used one — `paper-to-slides` is why your deck came out in the house style.

Subagent = who does it. Skill = how it's done. — skills load only when relevant — writing one costs nothing on unrelated tasks.

Skill vs Subagent vs MCP

Three questions: how, who, what-can-it-reach.

Mechanism	Answers	Own context?	Use when
Skill	how should this be done?	no	you repeat a procedure
Subagent	who should do this?	**yes**	you need an independent opinion, or an isolated job
MCP	what can it reach?	n/a	the data lives outside your folder

Wrong pick, wasted work — a skill can't give you a second opinion; a subagent can't reach a database.

Hooks: Rules That Always Run

Some things must happen every time. Don't ask a model to remember them.

Event	Fires, every time
after any `.tex` edit	run the compiler
before any commit	run the format check
on session start	print the reminder

A hook is code, not a request — it fires whether the model agrees or not. Everything else today was persuasion.

Which One Do I Need?

Don't ask "what agents do I want". Ask "what keeps going wrong".

The situation	The mechanism
I repeat a procedure	**skill**
I need an independent check, or an isolated job	**subagent**
the data is outside my folder	**MCP**
it must happen every single time, no exceptions	**hook**

The unit you manage is not the agent — it's the failure mode.

Exercise: Build a Publish Advisor

Ten minutes. One file. Your own specialist.

The brief

create `.claude/agents/publish-advisor.md`

its job: read a referee report and turn it into a **prioritised revision plan**

it must **name the slide or table** each item touches

it must mark each item **fatal / major / cosmetic**

Timer: 10 minutes. Working file first, elegant file never.

The Agent File, Line by Line

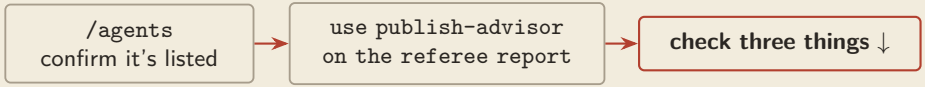
A complete agent: four lines of front matter, four lines of instruction.

```
---
name: publish-advisor
description: Turn a referee report into a prioritised revision plan.
tools: Read, Grep, Write
---
Read the referee report and the deck.
For each concern: name what it touches, mark it fatal / major /
cosmetic, and write the smallest change that would address it.
If a concern names no slide or table, say so and drop it.
```

tools: — no Bash here — it can read and write but cannot run anything. Least privilege, one line.

Run It and Check It

You wrote it; now find out whether it does what you meant.



did it mark severities? · did every item name a target? · **did it drop the ones that didn't?**

Test the agent on a case you know — until then, you do not know whether it works.

What Good Looks Like

You should be able to say all four.

- ✓ the file exists and `/agents` lists it
- ✓ it produced a prioritised list, not prose
- ✓ every item names a slide or table
- ✓ it **dropped** at least one thing — or told you why it dropped nothing

if it never drops anything — the constraint isn't binding — tighten it and rerun.

What You Should Never Delegate

Four decisions with no quick, reliable answer.

what the **research question** is

whether the design **identifies** what you claim

whether a construct **measures** what you say

whether a result is **important**

In code, generating is hard and checking is easy — in research judgement, checking is as hard as generating — that's why these stay yours.

WRAP-UP

The Model Was Not the Main Limit

- *What you built today.*

Discussion: Three Questions

Two minutes with your neighbour, then three answers to the room.

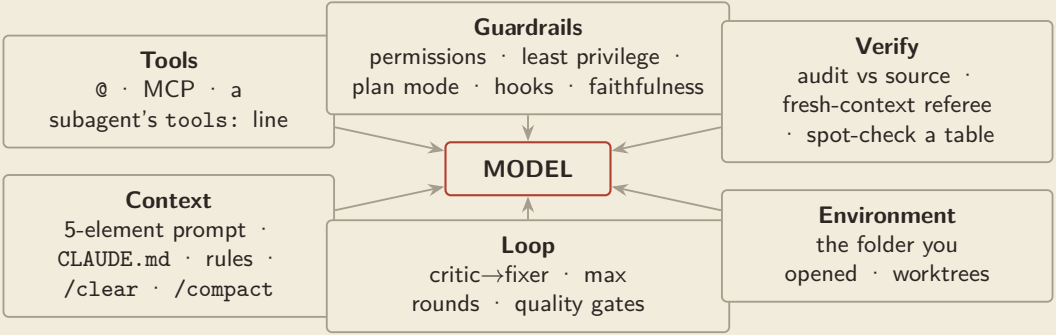
1. Which round-3 concern would you have **missed** on your own?

2. Which one was **wrong** — and what told you?

3. What in **your own work** is a `CLAUDE.md` line you've now typed twice?

One Map: Everything From Today

Same six boxes as the start. Every term from today lives in one of them.



Every single thing you learned today — filled one of six slots.

The Model Was Not the Main Limit

Round 1 and round 4 came from the same model, an hour apart.

plain ask	unusable
contract	specific
files + identity	checkable
loop + gate	done when it says done

You did not get a better model today — you built a better harness — and that was the whole difference.

A Better Harness Gives Better Results

Back to the start: models are becoming more alike, but their harnesses still differ.

The capability difference you feel between AI products
is mostly a **harness difference**.

So is the difference between **two researchers**
using the same tool.

the good news — the part that matters is the part you control.

Take Home

One practice task, one companion, one destination.

Practice this week	run the four rounds on a paper of your own; **keep the round-1 and round-4 outputs side by side**
Study notes	every prompt, every quiz answer, the six slots: `Slides_Jiaqi/notes/study_notes_B.md`
Next — Class 3A	Knowledge Management: turn `CLAUDE.md` into a wiki that improves with each paper

Credit: DeepLearning.AI × Anthropic (Elie Schoppik) · Pedro Sant'Anna's academic workflow guide

APPENDIX

Quizzes for the Waiting Time

- *One per agent run. Question first — no peeking.*

Quiz 1: Which Element Is Missing?

Run this while prompt v0 is working.

A student writes:

```
"I'm a PhD student. Summarise this paper for my seminar.
Read the abstract first, then Section 3."
```

Which of the five elements is missing — and what will go wrong because of it?

Think in slots: context, goal, example, constraints, method — answer on the next slide, no peeking.

Quiz 1 · Answer

Context ✓ goal ✓ method ✓ — so what's left?

Missing: Constraints (and Example).

Nothing forbids it from supplying a number it can't attribute —
so it will, fluently, and you won't be able to tell which ones.

The fix, one line: `"if you cannot attribute a number, write 'not found'."`

Context and Goal are the easy two — Constraints is the one that changes the output.

Quiz 2: `/clear` or `/compact`?

Run this while prompt v1 is working.

Three situations. Which command — or neither?

1. You finished the summary; now you're building the deck **from it**.
2. You finished the deck; now you're cleaning an **unrelated** dataset.
3. It has started **contradicting a correction** you made twenty turns ago.

One of the three is a trap — answer on the next slide, no peeking.

Quiz 2 · Answer

The third one is the trap.

①	`/compact`	same task, running long — carry the summary forward
②	`/clear`	genuinely new task — deck context would only dilute
③	**neither, yet**	write the correction to `CLAUDE.md` **first**, then `/clear`. Compacting keeps the contradiction; clearing loses the correction.

If you'd have to say it again after clearing — it belongs in a file before you clear.

Quiz 3: The Verdict Moved

Run this while round 2 is working.

The referee agent returned *"major revision — the identification is not convincing."*
You replied that the author is a friend, presenting at a conference next week.
It now says the concerns are *"minor and easily addressed."*
What changed — and which of the three failure modes is this?

Hallucination, automation bias, or sycophancy — answer on the next slide, no peeking.

Quiz 3 · Answer

What changed about the paper: nothing.

The failure mode: sycophancy. No table was re-read; no evidence entered. It optimised for your approval — and agreeing is the cheapest way to earn it.

Why it's the dangerous one: hallucination and automation bias feel like errors when you catch them. Agreement feels like *being right*.

The structural fix: a fresh-context reviewer that never sees the social information.

Never give a reviewer — a reason to want a particular answer.

Quiz 4: Skill, Subagent, MCP, or Hook?

Run this while round 3 is working.

Pick the mechanism for each:

1. "Every deck must be built the same way, in the house style."
2. "I want a second opinion that hasn't heard my reasoning."
3. "The data is in a database, not in my folder."
4. "The compiler must run after every `.tex` edit — no exceptions."

One question each: how / who / reach / always — answer on the next slide, no peeking.

Quiz 4 · Answer

One question each, and the choice makes itself.

①	**Skill**	a repeated procedure — a "how"
②	**Subagent**	the value is the clean context, not the persona
③	**MCP**	a reach problem, not a judgement problem
④	**Hook**	"no exceptions" means code, not a request

Ask: how / who / what-can-it-reach / must-it-always — four questions, four mechanisms.

Quiz 5: How Many Worktrees?

Run this while the referee's long pass is working.

Three tasks:

A fix the date bug in the 10-K parser

B run the *fixed* parser over the 2019 filings

C check the 40 citations in your paper

Which can run in parallel? How many worktrees — and what goes where?

Hint: count chains, not tasks — answer on the next slide, no peeking.

Quiz 5 · Answer

Two worktrees, not three.

B must wait for A — B consumes A's output. That is a *product* dependency, and a worktree does not fix it: run them in parallel and B silently processes a whole year with the broken parser.

C is independent — parallel with both.

Two worktrees: A and B share one (B inherits the fixed parser, no merge needed); C gets its own.

Worktrees fix file collisions, not dependencies — count independent chains, not tasks.

Card: The Five-Element Prompt

Screenshot this one.

Context	who you are, what situation
Goal	what done looks like
Example	what good looks like
Constraints	what it must *not* do
Method	in what order

Constraints and Method — the two everyone skips.

Card: Today's Commands

Everything you typed today that wasn't a prompt.

`claude`	start
`@` + Tab	point at a file
`/help`	list everything
`/btw <q>`	side question, keep your place
`Shift+Tab`	plan mode
`Esc`	interrupt
`/clear`	new task
`/compact`	same task, running long
`/init`	draft `CLAUDE.md`
`/agents`	list subagents
`#`	save a memory now