Claude Code as a Co-Scientist

A human-led, agent-assisted workflow for reproducible computational research

Jonathan Shock

Department of Mathematics & Applied Mathematics, University of Cape Town

June 18, 2026

A practical guide for research students. It describes a set of *skills* for the Claude Code agent, the research *workflow* they compose into, and the *reproducibility conventions* that make agent-assisted research trustworthy. Above all it is explicit about what stays *human* — the ideation, the reading, the writing, and the judgement that *are* the research. The accompanying `skills/` folder holds the skills, ready to install.

Adapting these for yourself

These skills are largely generic and work as shipped. A couple mention the author's personal setup as *optional* examples — for instance, filing a note in an Obsidian vault in `pre-register`. Treat any reference to a vault, email address, or project name as an example: swap in your own, or ignore it. The one skill with a real setup step is `academic-pdf-to-mkd` (a one-off local install — see Section 9).

Contents

1 Introduction

Large language model "agents" — systems that can read your files, run commands, and edit code in a loop — are now good enough to be genuine research collaborators. Used carelessly they are also an excellent way to manufacture plausible nonsense — or, worse, to quietly do your thinking for you. This guide is about the difference: how to use one such agent, **Claude Code**, as a *co-scientist* — not a substitute scientist — whose work is reproducible, source-grounded, and honest.

The single most important idea here: **the agent does the work that is mechanical, repetitive, or adversarial; you do the work that is generative, interpretive, and authorial.** The ideation, the reading, the writing, and the judgement stay yours — they *are* the research (Section 2).

It is organised around four things:

- **The human core** — what must stay yours, and why automating it away is a mistake (Section 2).
- **A workflow** — the research lifecycle, from triaging an idea to delivering a paper, with the human and agent roles separated at each step (Section 3).
- **A toolkit** — a set of *skills* that package the agent's part of each step so it is repeatable (Section 4).
- **Conventions** — the file-based habits that make any of it reproducible (Section 6), plus the working practices that get the most out of the agent (Section 7).

1.1 The one mental model that matters

Before anything else, internalise the distinction between four things that all contribute to what you see on screen:

1. **The model** interprets and generates text. It has a prior; it can hallucinate.
2. **The agent** is the model *in a loop* with tools, able to act and observe results.
3. **The harness** (here, Claude Code) gives the model its tools, permissions, files, and feedback.
4. **The computer / CLI** actually runs the commands and holds the files.

The single most useful question you can ask of any agent output is: *what did the model infer, what did it actually inspect, and what did the computer actually do?* Hallucination lives in the model layer; evidence lives in the file and command layer. Keeping them separate is how you know what is grounded.

The governing principle: the chat is not the archive

A conversation is ephemeral. If a source, a script, an output, a decision, or an assumption matters, it must end up **on disk**, linked to what produced it. Reproducibility is not a property of code; it is a practice made of documentation, versioning, logging, review, and shareable outputs. Everything in this guide is a way of honouring that one sentence.

1.2 How to read this guide

Read Section 2 first — it is the point of the whole guide. Section 3 is the spine; Section 4 is a reference to skim and return to. Sections 6 and 7 are the habits that separate a reproducible project from a pile of agent transcripts. Section 9 tells you how to install everything.

2 The human core — what not to automate

A workflow this automated has a failure mode that matters more than any bug: it lets you produce a paper without doing the research. The skills here are worth using precisely because they clear away the drudgery — but the drudgery was never the research. The research is the thinking, and the thinking has to be yours.

A co-scientist, not a substitute scientist

The agent is for the work that is **mechanical** (fetching, extracting, formatting), **repetitive** (checking fifty claims or two hundred citations), and **adversarial** (an honest critic that will not flatter you). It is *not* for the work that is **generative, interpretive, or authorial**. The moment you let it do your thinking, reading, or writing, you have stopped doing research.

2.1 The idea is yours

Taste — noticing what is strange, what is beautiful, what is worth a year of your life — is the whole game, and it comes from immersion, conversation, teaching, and play, not from a prompt. Do not ask the agent for your research questions. Bring an idea you already care about and use the agent to *stress-test* it: surface the load-bearing assumption, find the paper that already did it, ask the awkward question. `pre-register` is a gate for *your* idea, not a generator of ideas.

2.2 Reading is yours

Turning a PDF into Markdown is not reading it. The argument you have *with* a paper, the marginal note, the slow accretion of a mental map of a field, the connection that fires only in your own head — that is where research understanding is actually built. The ingestion skills (`academic-pdf-to-mkd`, `arxiv-fetcher`) put a paper in front of you in a workable form; they do not do the reading. Read deeply, and read more than the agent summarises.

2.3 Writing is thinking

Take this one most seriously. Writing is not the transcription of finished thoughts — it is how the thoughts get finished. Putting an argument into sentences is what exposes the gap in the logic, forces the definition you were fudging, and tells you what you actually believe. If the agent writes your draft, you skip exactly the thinking the writing was meant to do, and you end up defending prose you never reasoned your way to. So: **you write the draft.** The agent's job around your writing is to *critique* it (`paper-review`), check its claims (`claim-fact-checker`), and catch inconsistencies — not to produce it, and not to supply your voice.

2.4 Judgement is yours

What a surprising result *means*, whether a finding is interesting or merely true, which thread to pull next — these are judgements, and judgement does not come off a shelf. The agent can lay out options and evidence; choosing is yours.

2.5 The struggle is not a bug

Some of the friction in research — the stuck week, the fourth rewrite, the confusion that sits just before understanding — is not waste to be optimised away. It is often where the understanding is forged. Automate the drudgery, by all means; be wary of automating away the productive struggle along with it.

The acid test

At the end, can you defend every idea, every claim, and every sentence as your own thinking? If yes, the agent helped you do research. If the agent did the thinking, you have a paper but you have not done research — and your examiners, your reviewers, and your future self will eventually find the hollow centre.

3 The co-scientist workflow

The research lifecycle runs left to right. At each stage there is a *human* part (the row above the line — the research) and an *agent* part (the row below — the support). The agent never leads: it removes drudgery and adds rigour while you do the thinking. Figure 1 is the map; the rest of this section takes the stages one at a time, each split into "your part" and "the agent's part".

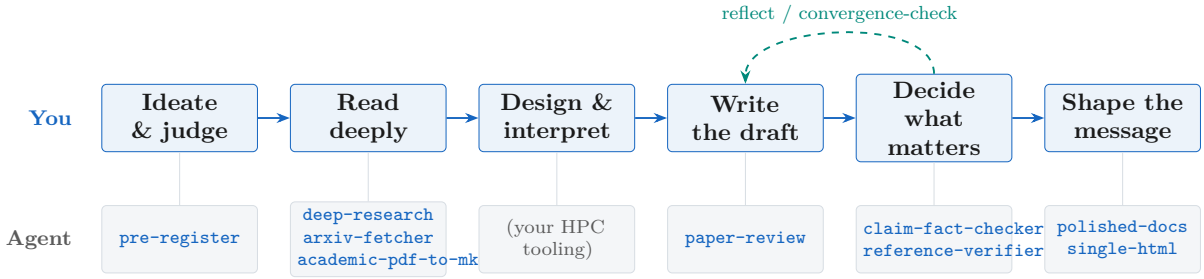

Figure 1: The workflow is human-led. The top row is *your* work — the research; the bottom row is the agent skill that supports each step. The agent removes drudgery and adds rigour; it never does the ideating, reading, writing, or judging. The dashed arc is the honesty loop: after each round, reflect on whether you are converging or just churning.

3.1 Triage the idea — `pre-register`

Before spending time or compute, decide honestly whether an idea is worth doing, and commit predictions *before* you pilot. The discipline is to ask "is this *interesting*?" not merely "is this *publishable*?", and to write down what you expect to see so you cannot rationalise afterwards. For a cheap computational pilot this is the highest-leverage five minutes in the project.

Your part: the idea, and the honest call on whether it excites you. *The agent's part:* stress-test the idea, surface the load-bearing assumption, and hold you to your written predictions.

3.2 Survey the literature — `deep-research`, `arxiv-fetcher`

`deep-research` runs a fan-out of web searches, fetches sources, adversarially verifies claims, and synthesises a cited report — use it for an open question where you must gather sources. `arxiv-fetcher` pulls the LaTeX source and PDF of a specific arXiv paper (often as a prerequisite for review). Prefer arXiv LaTeX source where it exists; it is cleaner than extracting from a rendered PDF.

Your part: read the key papers deeply and build your own map of the field. *The agent's part:* find and retrieve candidates, and draft a first cited survey for you to check — not trust.

3.3 Ingest any paper — `academic-pdf-to-mkd`

Most of the literature is not arXiv-with-source: journal PDFs, scanned chapters, supplements. `academic-pdf-to-mkd` converts *any* academic PDF into clean, structured Markdown — body text, figures, tables, and page thumbnails as separate artefacts — choosing an extraction engine

per file. For equation-dense papers it has a dedicated math engine (MinerU) so formulas survive. This is the ingestion layer: it turns the one format your other tools cannot read (PDF) into the one they all need (Markdown).

Your part: engage with the extracted text — annotate it, argue with it, connect it to your own work. *The agent's part:* get the paper into a clean, workable form.

3.4 Run experiments

For computational work you run jobs on your own cluster or machine. This toolkit does *not* ship an experiment-runner skill — schedulers, accounts, and paths vary too much from site to site to be portable — but the reproducibility convention still applies: stamp a *manifest* into every results directory (Section 6.3) so a run's provenance — code commit, arguments, environment — is never lost to a folder name.

Your part: design the experiment, decide what is worth running, and interpret the surprises. *The agent's part:* submit, monitor, and retrieve runs, and stamp each with a manifest.

3.5 Review the draft — `paper-review`

`paper-review` is a multi-agent referee: it spawns specialist reviewers (theory, experiments, writing, positioning, consistency, a devil's advocate), reconciles them, and produces a venue-style mock review with a ranked change list — gated on your approval before any change is made. It also ships single-pass "lenses" (a quick-read triage, a methods deep-dive, a Gelman-style statistical critique, and a claims-audit) for lighter passes.

Your part: write the draft, and decide which critiques to act on. *The agent's part:* play the adversarial referee — never the author.

3.6 Verify — `claim-fact-checker`, `reference-verifier`

These are complementary and easily confused, so keep the boundary sharp:

- `reference-verifier` checks that citations *exist* — DOIs resolve, metadata matches, nothing is hallucinated. It does *not* check whether a source supports the claim.
- `claim-fact-checker` checks that each load-bearing *claim* actually agrees with its archived *source* — one isolated verifier per claim, so quality does not decay on a long document. It does *not* gather new sources from the web.

Your part: judge which claims are load-bearing and what the evidence really shows. *The agent's part:* mechanically check each claim against its source, and each citation for existence.

3.7 Check convergence — `convergence-check`

After two or more iterations on a paper, an abstract, a proof, or an experiment, ask honestly whether you are converging, escalating, or reframing on weak data. If the abstract keeps getting rewritten on the same data, the data is the bottleneck — no amount of further iteration will fix it. This skill is the antidote to revise-and-resubmit-with-yourself.

Your part: the honest reflection on whether you are still learning. *The agent's part:* hold up an undistorted mirror.

3.8 Deliver — `polished-docs-and-decks`, `single-html-document`

`polished-docs-and-decks` builds themed slide decks (`.pptx`) and typeset PDFs from Markdown, and verifies them by rendering to images. `single-html-document` builds self-contained,

interactive single-file HTML deliverables (briefings, dashboards, review workbooks, browser slide decks). Use the former when the deliverable is a deck or typeset PDF; the latter when it is a shareable, interactive web page.

Your part: the message and the argument. *The agent's part:* typeset, package, and render-check the artefact.

4 The skill toolkit

4.1 What a skill is

A *skill* is a folder containing a `SKILL.md` file: YAML front matter with a `name` and a `description` (the trigger), followed by concise instructions for the agent, plus optional `scripts/`, `references/`, and `templates/`. The `description` is what the agent matches against your request to decide whether to use the skill, so it is written as a rich set of triggers, not a summary. A skill packages a *repeatable procedure* so the same vetted method is applied identically across projects — the reusable cousin of project-specific instructions.

Rule of thumb: skill vs. project instructions

If a practice should repeat *across* projects, make it a skill. If it belongs to *one* project, put it in that project's `CLAUDE.md` (Section 6.8).

4.2 The research skills at a glance

Table 1 summarises the bundled set. Each is in the `skills/` folder.

Skill	Stage	What it does / when to reach for it
`pre-register`	triage	Decide if an idea is worth the compute; commit predictions before piloting.
`deep-research`	literature	Fan-out web research → verify → cited report. (Built into Claude Code.)
`arxiv-fetcher`	literature	Fetch LaTeX source + PDF for an arXiv paper/ID/URL.
`academic-pdf-to-mkd`	ingest	Any PDF → structured Markdown (+ figures, tables); math + OCR engines.
`paper-review`	review	Multi-agent mock peer review with venue rubrics + single-pass lenses.
`claim-fact-checker`	verify	Per-claim check against archived sources (content agreement).
`reference-verifier`	verify	Citation existence / DOI / metadata audit (not claim support).
`convergence-check`	meta	Are you converging or churning? Is the data the bottleneck?
`polished-docs-and-decks`	deliver	Themed `.pptx` + typeset PDF from Markdown, render-verified.
`single-html-document`	deliver	Self-contained interactive single-file HTML deliverables.

Table 1: The research / co-scientist skill set.

4.3 Keeping the boundaries sharp

The most common failure with a rich toolkit is the *wrong* skill firing. Three boundaries are worth memorising, because the verbs overlap:

- `reference-verifier` (do the citations *exist*?) vs. `claim-fact-checker` (do the claims

match their sources?) vs. `deep-research` (go *find* sources on the web). Existence, agreement, discovery — three different jobs.

- `academic-pdf-to-mkd` (a *local* PDF → Markdown) vs. `arxiv-fetcher` (an *arXiv* paper → LaTeX source). Prefer source over extraction when you can get it.
- `polished-docs-and-decks` (deck / typeset PDF) vs. `single-html-document` (interactive HTML). Pick by the *deliverable format*.

5 Skill reference

This section is the detailed reference: one entry per skill, grouped by stage, giving its purpose, the kinds of request that trigger it, what it consumes and produces, and practical notes. Skim once; return when you need a particular tool.

5.1 Anatomy of a skill

Every skill is a folder containing a `SKILL.md` of this form:

```
---
name: paper-review
description: Multi-agent academic paper review. Use when the user asks to
  review, critique, or get feedback on a paper / manuscript / preprint;
  triggers include "review my paper", "what would reviewers say",
  "find weaknesses in my draft", "prepare for submission".
---
# Paper Review
...concise instructions telling the agent what to DO...
```

The `description` is what the agent matches against your request to decide whether to use the skill, so it is written as concrete triggers, not a summary. The body is procedure, not essay. Optional `scripts/`, `references/`, and `templates/` are loaded only when needed.

5.2 Triage

pre-register. **Purpose:** an idea-triage and pre-registration gate — decide honestly whether an idea is worth the time or compute, and commit predictions before piloting. **Triggers:** "is this worth doing?", "should I run this experiment?", "pre-register this", "what would make this interesting?". **In/out:** an idea or hypothesis in; a go / no-go judgement plus written predictions out. **Notes:** highest-leverage before a cheap computational pilot; forces the question "is this *interesting*?", not only "is this publishable?". Pairs with `convergence-check` at the other end of the loop.

5.3 Literature and ingest

deep-research. **Purpose:** a deep research harness that fans out web searches, fetches sources, adversarially verifies claims, and synthesises a cited report. **Triggers:** "research X", "what is known about…", "do a literature review of…". **In/out:** an open question in (it will ask a couple of clarifying questions if the scope is too broad); a cited, fact-checked report out. **Notes:** built into Claude Code. Use it to *gather* sources; once you already have them, switch to `claim-fact-checker` for verification.

arxiv-fetcher. **Purpose:** fetch the LaTeX source and PDF of an arXiv paper. **Triggers:** an arXiv ID or URL, "download this arXiv paper", "pull this preprint". **In/out:** an arXiv identifier

in; `.tex` source and PDF out. **Notes:** prefer the LaTeX source to PDF extraction whenever it exists; commonly a prerequisite step for `paper-review`.

academic-pdf-to-mkd. **Purpose:** convert any academic PDF into structured Markdown — body text, figures, tables, and page thumbnails as separate artefacts — auto-selecting an extraction engine (a layout-aware default, a fast engine for short born-digital papers, a math engine for equation-dense papers, and a no-dependency fallback) behind an OCR preflight for scans. **Triggers:** "extract this PDF to markdown", "OCR this scanned paper", "get the text and figures out of this PDF". **In/out:** a local PDF path in; a fulltext Markdown file plus `figures/`, `tables/`, `pages/` out. **Notes:** the ingestion layer for non-arXiv literature; the math engine downloads ∼4.5 GB of models on first use. It also has a branch that produces a searchable, selectable PDF for a reference manager.

5.4 Experiments

No experiment-runner skill is bundled — cluster setups (scheduler, accounts, paths) are too site-specific to ship portably. Run jobs with your own tooling, and apply the reproducibility conventions of Section 6 (immutable inputs, a manifest per results directory, figure-to-source binding) so the runs stay reproducible.

5.5 Review and verification

paper-review. **Purpose:** a multi-agent mock peer review — parallel specialists (theory, experiments, writing, positioning, consistency, and a devil's advocate), a synthesis agent that ranks changes, and human-gated implementation and verification — plus single-pass lenses for lighter work (a quick-read triage, a methods deep-dive, a Gelman-style statistical critique, and a claims audit). **Triggers:** "review my paper", "critique this manuscript", "what would reviewers say", "prepare for submission". **In/out:** a `.tex` or PDF draft (and optionally a target venue) in; a venue-style mock review and a ranked, de-duplicated change list out. **Notes:** it never auto-implements changes — the human gate is the point; it supports venue rubrics (NeurIPS, ICML, Nature, PNAS, and more).

claim-fact-checker. **Purpose:** fan-out fact-checking — one isolated verifier per claim, plus a source-quality skeptic — to check that every load-bearing claim in a deliverable agrees with its archived sources, producing a findings file and a fix-plan without editing the deliverable. **Triggers:** "fact-check this report", "verify the claims", "check this review against the paper". **In/out:** a finished deliverable plus its archived sources in; findings and a fix-plan out. **Notes:** checks claim–source *content agreement*, not citation existence; it needs sources already on disk; token cost scales with the number of claims, so scope it to what changed.

reference-verifier. **Purpose:** audit a bibliography — existence checks, DOI resolution, metadata cross-validation, and duplicate / orphan detection — to catch hallucinated or malformed citations. **Triggers:** "verify the references", "are these citations real?", "check the bibliography", "find fake references". **In/out:** a `.tex`/`.bib`, PDF, or `.docx` in; a per-reference verdict out. **Notes:** it confirms a cited work *exists* — not that it supports the claim it is cited for (that is `claim-fact-checker`).

5.6 Meta

convergence-check. **Purpose:** diagnose whether an iterative loop — on a paper, an abstract, a proof, or an experiment — is genuinely converging, escalating, or reframing on weak data. **Triggers:** "am I going in circles?", "are we converging?", "should I stop iterating?", "this is the

third time I have reframed this". **In/out:** the history of iterations in; an honest read — keep going, stop, or change the question — out. **Notes:** if the abstract keeps being rewritten on the same data, the data is the bottleneck, and more iteration will not fix it.

5.7 Delivery

polished-docs-and-decks. **Purpose:** build themed slide decks (`.pptx`) and typeset PDFs from Markdown, verified by rendering to images with a headless office suite. **Triggers:** "make a slide deck / talk", "turn these notes into a handout / typeset PDF". **In/out:** Markdown or notes in; a `.pptx` or a typeset PDF out, render-checked. **Notes:** bundles a figure-overlap pre-check, a Unicode-math preamble, and matplotlib patterns for equation and node/arrow figures.

single-html-document. **Purpose:** build self-contained, single-file HTML deliverables — briefings, dashboards, research syntheses, review workbooks, browser slide decks — from structured content blocks (timeline, flowchart, module-map, inline SVG, and more). **Triggers:** "make a standalone / single-file HTML", "an interactive briefing or dashboard", "a review workbook". **In/out:** content or a small site in; one browser-openable `.html` out, with everything inlined. **Notes:** use it when the deliverable is interactive HTML; use `polished-docs-and-decks` when it is a deck or a typeset PDF.

6 Reproducibility conventions

Skills are tools; these conventions are what make the output trustworthy. They are all instances of "the chat is not the archive".

6.1 File-based evidence

Judge any agent session by five questions: are the inputs preserved? is there a script? are outputs saved separately from inputs? are assumptions and uncertainties recorded? could another person inspect what happened? If the answer to all five is yes, the work is reproducible even if it was done in conversation with an agent.

6.2 Immutable sources and raw data

Keep inputs you depend on in a `source-material/` folder and raw data in `data/raw/`, and **never edit them**. Derived data is produced into a separate folder *by a script*. The agent is instructed never to touch raw inputs and never to hand-edit generated outputs — it changes the script and re-runs.

6.3 Results manifests

The version of a result must not live only in a folder name. Have your job script write a manifest into every results directory at run time:

```
run_utc:    2026-06-17T14:00:00Z
slurm_job:  1234567   array: 1234560
git_commit: a4c2bc9 (clean)
account:    l40sfree   partition: l40s   node: gpu-l40s-003
env:        my-env (module python/miniconda3-py3.12)
command:    python -u run_single_seed.py --method convex_psrl --seed 3
note:       /scratch is NOT backed up; GPU runs need the l40s partition
```

Now any directory's provenance — code commit, arguments, environment, and what *cannot* be reproduced without the cluster — is captured on disk.

6.4 Decision logs and ADRs

Keep a `decision-log.md` with one row per non-trivial choice:

```
| Date | Decision | Reason | Evidence / files used |
```

For the big, hard-to-reverse choices (a modelling assumption, a dropped baseline), write a short *Architecture/Analysis Decision Record* (ADR): one file stating the decision, the alternatives, the trade-off, and why. The decision log is the running ledger; ADRs are the deep entries.

6.5 Binding figures to their source

A figure with no record of where it came from is a reproducibility hole. The convention: every figure is produced by a script that takes its data directory as an explicit input and writes a *sidecar* next to the figure naming the results directory, the code commit, the generating script, and the date. Chained with the results manifest you get an unbroken trail (Figure 2).

Figure 2: The provenance chain: from a figure in the paper back to the exact code commit that produced its data. Each link is a small file written automatically.

6.6 Shared language: `CONTEXT.md`

Research domains are dense with precise terms. A `CONTEXT.md` "ubiquitous language" glossary records them once, so the agent (and your collaborators) use them consistently instead of re-deriving fuzzy definitions each session. The discipline (from the *grill-with-docs* technique) is that the agent *challenges* colliding or vague terms and asks clarifying questions *before* writing:

```
# Ubiquitous language (CONTEXT.md)
MYULA        Moreau-Yosida Unadjusted Langevin Algorithm; proximal Langevin
             sampler for the non-smooth L1 prior.
K=1 regime   One Langevin step after the MAP estimate; empirically matches
             K=50 (the "K=1 phenomenon").
sign pattern One ReLU activation pattern; P of them index the lifted problem.
```

6.7 Version control and Overleaf

The manuscript is a research artefact like any other, so version it. Keep the paper — `.tex`, `.bib`, and figures — in a **git** repository on **GitHub**. That gives you a complete history of every change, an off-machine backup, and the commit hashes the manifest and figure-binding conventions point back to: the sidecar beside a figure names the exact commit its data and code came from, so a compiled PDF is always traceable to a source state.

For writing and collaboration, link that GitHub repository to **Overleaf**. Overleaf gives you a browser editor, one-click compilation, and real-time co-authoring; its GitHub sync (*Menu* → *GitHub* inside an Overleaf project) keeps the two in step — GitHub stays the canonical history, Overleaf is the editing and compiling surface. A typical loop: draft and commit (locally, or via the agent), push to GitHub, pull into Overleaf to compile and share with co-authors, then pull their edits back to GitHub. (Overleaf's GitHub integration is a Premium feature; many universities provide it through an institutional licence.)

This keeps the writing — which is yours (Section 2) — under version control without giving up

a shared online editor, and it means the source of any PDF you circulate is never more than a commit away.

6.8 The project template

A new paper or analysis repository starts from a template `CLAUDE.md` that encodes all of the above: the immutable `source-material/`, the data rules, the `decision-log.md` format, the figure-binding rule, and routing to the skills (ingest with `academic-pdf-to-mkd`, review with `paper-review`, verify with `claim-fact-checker` and `reference-verifier`, deliver with the document skills). `CLAUDE.md` is Claude Code's per-project instruction file: it loads automatically when you open the folder, so the conventions apply on every task without re-explaining them.

7 Working practices

7.1 Self-healing: correct the rule, not the symptom

When you correct the agent, do not just fix the immediate output — have it *mutate the rule* so the correction sticks. The protocol: restate the rule it thinks you are asserting and confirm; then edit at the narrowest scope that holds it (a global rule in `CLAUDE.md`; a skill's behaviour in its `SKILL.md`; a project rule in the project's `CLAUDE.md`; a durable fact in memory); prefer editing an existing rule over adding a new one; and report what changed. Corrections compound into a system that gets better, rather than the same mistake recurring.

7.2 Multi-agent workflows for breadth and confidence

For tasks that are too big for one context — reviewing a large diff, auditing a whole repository, surveying many papers — Claude Code can *fan out* parallel sub-agents and then *adversarially verify* their findings: each candidate finding is checked by an independent skeptic before it survives. Use this when you want either coverage (decompose and parallelise) or confidence (independent verification before committing). It is the right tool for a systematic review or a reproducibility audit; it is overkill for a quick question.

Why adversarial verification matters

A single agent that both finds and confirms its own findings will defend its earlier judgements. Separating the finder from an independent verifier catches plausible-but-wrong claims — the same logic as `claim-fact-checker`'s one-verifier-per-claim design.

7.3 Diagrams and visualisation

You rarely need a separate tool to draw a diagram: Claude Code can render SVG diagrams, flowcharts, and charts directly in the conversation. For diagrams embedded in a shareable document, `single-html-document` provides structured blocks (timeline, flowchart, module-map, inline SVG). For figures destined for a paper or deck, `polished-docs-and-decks` drives matplotlib (equations via mathtext, node/arrow diagrams). Match the tool to the destination: chat, web document, or paper.

7.4 Honesty as a working style

The thread running through `pre-register`, `convergence-check`, the verification skills, and adversarial workflows is the same: value honest pushback over reflexive iteration. Ask the agent to tell you when an idea is weak, when a loop is not converging, and when a claim is not supported. An agent that only agrees is a liability.

7.5 Common pitfalls

The recurring ways agent-assisted research goes wrong — and the habit that fixes each:

- **Trusting the model layer.** Treat any specific number, claim, or citation as a hypothesis until a tool has checked it. "It sounds right" is the model's prior, not evidence.
- **The wrong skill firing.** Overlapping verbs (verify, audit, check) make mis-triggering easy — keep the three boundaries in Section 4 in mind, and name the skill explicitly when in doubt.
- **The chat as archive.** If a decision or result exists only in the conversation, it is lost. Land it on disk, linked to what produced it.
- **Hand-edited figures.** A plot tweaked by hand breaks the provenance chain. Change the script and re-run.
- **Iterating past convergence.** More review rounds cannot fix a data problem — run `convergence-check` before another lap.
- **Ephemeral data.** Source data in `/tmp` or on an un-backed-up scratch disk is a reproducibility hole. Record what cannot be reproduced, and keep canonical data somewhere durable.
- **Silent landmines.** A broken or superseded results directory left beside live data will eventually be globbed into a figure. Quarantine it out of any aggregation path with a `DO_NOT_USE` marker.

7.6 Writing your own skills

When you notice yourself re-explaining the same procedure across projects, promote it to a skill. Keep it narrow; focus the `SKILL.md` on what the agent should *do*, not on background; write the `description` as concrete triggers; and add `scripts/` or `references/` only when they make the procedure easier to repeat. A good skill is a vetted method made portable.

8 A worked example

Putting it together, a typical project flows like this — *you* lead each step and the agent supports it:

1. **Triage.** *You* bring an idea you care about; `pre-register` stress-tests it and records your predictions.
2. **Read.** `deep-research`, `arxiv-fetcher`, and `academic-pdf-to-mkd` gather and ingest the literature into `source-material/`; *you* read the key papers deeply.
3. **Experiment.** *You* design the runs; your cluster executes them; each results directory gets a manifest (Section 6.3); *you* interpret what comes back.
4. **Draft & review.** *You* write the paper — that is where the thinking happens; `paper-review` then plays referee, and you decide what to act on.
5. **Verify.** `claim-fact-checker` audits every load-bearing claim against `source-material/` and `reference-verifier` confirms the bibliography; *you* judge what matters.
6. **Check yourself.** `convergence-check` after each round — are you improving or churning?
7. **Deliver.** `polished-docs-and-decks` or `single-html-document` package the result; the message is yours.

At every step the artefacts — sources, scripts, manifests, decisions, reviews — are on disk, so the project is reproducible by someone who was never in the room. And at every step the thinking is yours: at the end, you can defend every idea, claim, and sentence as your own.

9 Getting started

9.1 Prerequisites

- **Claude Code** (the CLI agent). `deep-research` is built in.
- **Homebrew** tools for the PDF extractor: `poppler`, `ocrmypdf`, `ghostscript`; and `uv` for its Python environment.
- **A LaTeX install** (MacTeX / TeX Live) for typeset PDFs.
- **git** and a **GitHub** account for versioning the paper; optionally **Overleaf** for browser-based, collaborative writing linked to that repository (Section 6.7).
- Optional: **Zotero** (a reference manager) if you want a library to ingest from.

9.2 Installing the bundled skills

The `skills/` folder beside this document holds the skills as ready-to-use folders. Copy them into Claude Code's skills directory:

```
# Install the skills
cp -R co-scientist-toolkit/skills/* ~/.claude/skills/

# One-off setup for the PDF extractor (the ~4.5 GB math models are optional
# and only downloaded when you first use the math engine)
brew install uv poppler ocrmypdf ghostscript
uv sync --project ~/.claude/skills/academic-pdf-to-mkd
```

9.3 Invoking a skill

Skills fire in two ways: by their trigger (just describe what you want — "review this paper" selects `paper-review`) or explicitly by name (`/paper-review`). You do not need to remember the catalogue; the `description` in each `SKILL.md` is what the agent matches against your request.

9.4 Per-project setup

For a new paper or analysis repository, copy the template `CLAUDE.md` into the repo root, create a `CONTEXT.md` glossary, and let the conventions in Section 6 do the rest. `CLAUDE.md` loads automatically, so every session inherits the immutable sources, the decision log, the manifest and figure-binding rules, and the skill routing. Put the repository on GitHub for history and backup, and link it to Overleaf for browser-based writing and collaboration (Section 6.7).

A Bundled skills and attribution

The `skills/` folder contains the following, ready to install. Several are adapted, with thanks, from Dominik Lukeš's public *agent-skills* collection (MIT licence); the *grill-with-docs* / `CONTEXT.md` practice is from Matt Pocock (AI Hero).

B Glossary

A short shared vocabulary for this guide — the kind of thing a project's own `CONTEXT.md` would hold.

Agent
A model in a loop with tools, able to act and observe results, as opposed to a chatbot that only responds.

Skill	One line	Origin
pre-register	Idea triage / pre-registration gate	local
arxiv-fetcher	Fetch arXiv LaTeX source + PDF	local
academic-pdf-to-mkd	Any PDF → structured Markdown (+ math/OCR)	Lukeš (MIT)
paper-review	Multi-agent mock peer review + lenses	local
claim-fact-checker	Per-claim verification vs. sources	Lukeš (MIT)
reference-verifier	Citation existence / DOI audit	local
convergence-check	Convergence / churn diagnostic	local
polished-docs-and-decks	`.pptx` + typeset PDF, render-verified	local
single-html-document	Self-contained interactive HTML	Lukeš (MIT)
deep-research	Fan-out web research (built into Claude Code)	built-in

Harness
The system (here, Claude Code) that gives the model its tools, permissions, files, and feedback loop.

Skill
A reusable, packaged procedure the agent can invoke; a folder with a `SKILL.md`.

`SKILL.md`
A skill's manifest: YAML front matter (name + trigger description) plus concise instructions.

`CLAUDE.md`
A project's instruction file, loaded automatically when the agent opens the folder; holds project-specific rules.

`CONTEXT.md`
A project's "ubiquitous language" glossary of precise domain terms, kept consistent across code, prose, and agent sessions.

ADR
An Architecture / Analysis Decision Record: one short file per hard-to-reverse choice, stating the decision, alternatives, and trade-off.

Manifest
A small file written into a results directory recording how it was produced (commit, arguments, environment, date).

Sidecar
A small companion file beside an output (e.g. a figure) recording its provenance.

Provenance chain
The unbroken trail from a paper figure back to the data and code commit that produced it.

Fan-out / multi-agent workflow
Running many sub-agents in parallel over a task, then combining their results.

Adversarial verification
Having an independent skeptic check a finding before it is accepted, rather than letting the finder confirm itself.

Self-healing
Turning a correction into a durable rule edit at the right scope, so the same mistake does not recur.

C Further reading

- Dominik Lukeš, *agent-skills*: https://github.com/techczech/dominiks-agent-skills

- Dominik Lukeš, *Using AI Agents for Reproducible Research* (workshop): https://github.com/techczech/agents-for-reproducibility
- Matt Pocock, *grill-with-docs*: https://www.aihero.dev/grill-with-docs
- Claude Code documentation: https://docs.claude.com/en/docs/claude-code