

Getting Started with Generative AI in Research

Quick-start tips from four mid-career economists who began at zero

A resource of the CSWEP Peer-to-Peer (P2P) Mentoring Program

Shooshan Danagouliau, Wayne State University

Amanda Gaulke, Kansas State University

Leah Lakdawala, Wake Forest University

Xi Yang, University of North Texas

September 2026

CSWEP does not endorse any particular method or product. Neither do we. Everything here is our own experience over the summer of 2026, and the tools change monthly.

Contents

1. Why we wrote this (and why now is a good time)	3
2. Getting set up: the ten-minute version	3
3. If you only try four things.....	4
4. Level one: the chat window	5
4.1 Writing and editing	5
4.2 Literature review and understanding papers	6
4.3 Admin, service, and the rest of the job.....	7
5. Level two: letting the AI into your folders	8
6. Coding and empirical work	9
7. Checking its work without reading every line.....	10
8. Guardrails: data agreements, journals, and disclosure	11
9. What it costs and what we would pay for	12
10. Quick reference.....	14
10.1 Cheat sheet	14
10.2 Phrases worth stealing.....	14
11. Further reading	15

Read this first

This guide is a snapshot from September 2026. The tools we mention change monthly, the prices change, and the names of the models change even faster. We have tried to focus on the habits and workflows that will survive the next update, but even those might not. Treat this as notes from friends who went first, not as a manual.

It is also deliberately incomplete. We are four applied microeconomists who work mostly in Stata. We tried what seemed useful to us, wrote down what happened, and skipped the rest.

1. Why we wrote this (and why now is a good time)

We are four mid-career economists who spent the summer of 2026 figuring out how to work generative AI into our day-to-day jobs, with a small grant from CSWEP's Peer-to-Peer mentoring program to cover the subscriptions. When we started, most of us had done little more than type a question into a chat window. By the end, all of us were using AI on real research projects, and some of us now use it for most of what we do at a keyboard.

This document is what we wish someone had handed us in June. It is not a comprehensive guide to what AI is and isn't. The set of tools is too big for four people to cover, it changes too fast for any review to stay accurate, and we don't endorse any particular product. What we can offer is honest notes on what worked for us, what didn't, and how we would tell a colleague to start.

Why mid-career is harder. We have workflows that already work, and it is painful to disrupt them. Our jobs have expanded, so there is less slack for experimenting. And the stakes are higher: data agreements, journal policies, and coauthor expectations all assume we are the ones who know the rules.

Why it is also the right time. Mid-career security is exactly what makes it possible to try something new without it feeling like an existential risk. And the academic landscape is changing fast enough that staying current feels less optional every year. If a bunch of tenured people with too many committee assignments can do this over a summer, so can you.

How to read this. Sections 2 and 3 get you started in an afternoon. Sections 4 and 5 describe two levels of use: the chat window, and then giving the AI access to your files. Sections 6 and 7 are for when you are ready to let it touch your code. Section 8 is the fine print on data agreements and disclosure, Section 9 is what all of this costs, and Section 10 is a one-page cheat sheet. Skip around.

2. Getting set up: the ten-minute version

Step 1. Open an account. Pick one of the major general-purpose platforms: Claude (Anthropic), ChatGPT (OpenAI), or Gemini (Google). All of them have free tiers. Don't agonize over which one; you can switch later, and several of us ended up using more than one.

Step 2. Ask it something. Type a question into the chat window the same way you would type into a search bar. If you can't think of anything, ask it to walk you through Stata's reshape command. (Yes, really. Everyone forgets reshape.) Then ask a follow-up. Then paste in a paragraph from a paper you are reading and ask it to explain the identification strategy. That's it. You are using AI.

Step 3. Decide what to pay, later. Free tiers are enough to kick the tires. Paid plans run from roughly \$20 a month for standard access to \$100 to \$200 a month for the top models and much higher usage limits. Our advice is to start free or at \$20 and upgrade only once you hit a wall (though we have heard that response quality, not just usage limits, improves with higher paid tiers). Section 9 explains how we thought about the money.

Step 4. Keep an eye on usage. Every platform has some kind of usage limit, and hitting it in the middle of a task is annoying. Click on your account or settings and look for "usage," or just ask the AI where to find it. The subscription level you need depends on how much you actually integrate it into your work, and you won't know that for a few weeks.

Step 5. Know the two ways to use it.

- **The chat window** (browser or app). You type, it answers. You upload files you want it to read and download files it creates. This is where everyone starts, and it is genuinely useful on its own.
- **The desktop apps with file access.** Claude Code, Claude Cowork, and ChatGPT's Codex are desktop apps that can read and write files in folders you point them at, on your computer or in Dropbox, with your permission. This is where things get powerful. It is also where several of us hesitated for months because the setup and the permissions felt intimidating. It turned out to be easier than we expected and was the single biggest step-change in what AI could do for us. See Section 5.

Tip: pointing the AI at a file

In the desktop apps, the easiest way to say which file or folder you mean is to right-click it in File Explorer and choose "Copy as path," then paste that into your prompt. On a Mac, hold Option while right-clicking and choose "Copy as Pathname." You can also just describe it ("the do-file in the Analysis folder that makes Table 2") and the AI will usually find it.

3. If you only try four things

If you read nothing else, try these. They are the uses each of us kept coming back to.

1. **Get feedback on a draft.** Upload a paper, or point the desktop app at your draft folder, and ask for big-picture comments: is the argument clear, do the results support the claims, what would a skeptical referee say. Then argue with it. This is the use that most surprised us. (Sections 4.1 and 5.)
2. **Have it explain a paper to you.** Save the PDF and have a conversation about it. What is the identifying assumption? How is this different from what I am doing? Could I use this estimator? It is like having a patient colleague who read the paper more carefully than you did. (Section 4.2.)

3. **Turn messy notes into a polished document.** Referee reports, recommendation letters, emails to coauthors, a memo for the dean. Type your thoughts as messily as you like, then ask for a clean version in your own style. (Section 4.3.)
4. **Clean up code or a project folder.** Ask it to add comments to an old do-file, turn a chaotic project folder into a replication package, or write the robustness check you have been putting off. (Section 6.)

4. Level one: the chat window

Most of us used the browser chat for a bit before trying anything fancier, and for some of us, a lot of what we do still happens there.

4.1 Writing and editing

The voice problem. Left to its own devices, AI writes like a very polite committee. It hedges, softens, reaches for vivid phrases the evidence doesn't support, and generally doesn't sound like you. Everything it writes needs real revision before it is yours.

The agreeableness problem. The other default worth knowing about is that AI is built to agree with you. Ask whether your argument holds up and it will find reasons it does. Ask for supporting evidence and it will produce some, whether or not it exists. Faced with a gap, it would rather manufacture a citation, a number, or a quote than admit it doesn't know. Think of it as a confirmation bias that ships turned on. Part of getting started is turning it off. At the start of a project, and in any chat where it matters, tell it never to invent evidence, literature, or material, to flag anything it is unsure of, and to just say "I don't know." It won't follow this perfectly, which is why Section 4.2 is about verifying, but the instruction meaningfully changes what you get back.

What helped:

- **Feed it your own writing.** Published papers, proposals, emails both formal and casual. It meaningfully improves how well the output matches your voice. You will still adjust the tone for each task, especially email, where only you know your relationship with the recipient.
- **Edit in small pieces.** Start with a few lines or a paragraph. Once you like that, ask it to review a section. Don't hand it the whole paper; that is how it develops a strange voice that isn't yours.
- **Ask for several versions** of the same passage and pick the one that makes the point the way you would make it.
- **Treat the output as a suggestion,** not as final text. One of us puts it this way: "I more often use AI as a starting point than as text that goes into the draft."
- **When it is too timid, say so.** "Use stronger language here" works. So does rewriting the sentence yourself and asking AI to review and learn from it.
- **Ask for the case against.** "What is the strongest objection to this paragraph?" gets a more honest read than "Is this good?"

We also tried asking for target-journal suggestions, with mixed results. Asking about fit to a specific journal generated useful results, especially when we asked the AI to find the five most recent papers on

our topic published in that journal. But when we asked the broader question of which journal would be a good fit, the AI sometimes suggested one that proved to be a poor fit.

One of us: a full collaboration, start to finish

One of us completed an entire project in collaboration with the AI. Starting from data and preliminary results, the AI pointed out a weakness in the identification strategy and suggested an alternative. It wrote the Stata code to implement it, which our colleague ran personally after checking every line, because the data were proprietary with limits on AI use (see Section 8). With results in hand, our colleague picked a target journal and downloaded a few recent articles from it so the AI could learn the journal's voice and structure, then had it draft an outline. From there they worked section by section: the AI asked questions, our colleague answered, the AI proposed a paragraph, our colleague revised heavily, and the AI cleaned up the language and asked for confirmation.

The verdict: it took about 70 percent of the time that drafting alone would have, and the revisions were substantial. The AI sometimes claimed more than the results warranted, occasionally misread a table or the significance of an estimate, and tended to offer the most obvious interpretation while missing nuance or context. It would drop insignificant results without asking why significance was lost. Its suggested robustness checks were the low-hanging-fruit kind; the harder ones our colleague had to think of alone. After the first full draft, our colleague re-read the paper three or four times, correcting and reframing. Collaborative, yes. Autopilot, no.

Where the line is. Decide before you start how much AI involvement you want on a given project. You can use it to check your work (code and prose), to discuss your work (talk through a question, suggest directions), to implement (write or run code), or to collaborate on the whole thing. AI is better at some stages than others, and it is worth noticing where it saves time and where it is faster to do the task yourself. Crucially, because AI can still make errors or hallucinate details, checking and verifying its output are essential. The disclosure rules in Section 8 may also make the decision for you.

4.2 Literature review and understanding papers

What it is good for:

- **A first sweep of the related literature.** Ask it to find and summarize papers related to your project. Then, and this is essential, vet each one yourself.
- **A deep dive on one paper.** Upload the PDF (or use a tool like NotebookLM) and have a conversation. Break down a new econometrics paper to see whether the tools apply to your work, figure out how your paper is distinct from it, or just understand one confusing result.
- **Writing and citing in one pass.** A general platform can search and draft together, so you are not shuttling between a search tool and a writing tool. Because it remembers your project, its searches tend to be more relevant than a specialized tool that starts cold every time. The flip side is that your literature then lives in a chat history rather than a reference manager, which is a problem on the day the AI is down.

Trust, but verify. We did not find that AI invents papers, which was the classic failure a couple of years ago. What we did find:

- It doesn't always distinguish a paper's central conclusion from a supporting result.
- It misses nuance in interpretation and occasionally mislabels a design, such as calling an experiment "quasi-experimental."
- It confidently calls things "the canonical paper" that aren't even published.
- It rates peer-reviewed papers above working papers in a way that suggests it doesn't understand publication lags in economics.

So use it to build a reading list and get an overview quickly, and then read the papers. Finding papers is much faster with AI. Verifying them still takes real time. Before anything goes into a submitted paper, check every citation yourself. Even a single hallucinated or mischaracterized reference will look bad.

Use more than one platform. Several of us run the same literature search on two or three platforms because they find different papers. The overlap is high but not complete, and the differences are worth having.

4.3 Admin, service, and the rest of the job

This is where the time savings are most visible, and where the lowest-stakes practice lives.

- **Email.** Draft, shorten, polish, and calibrate tone, with different versions for different audiences.
- **Referee reports.** Several of us type our comments messily, then have the AI turn them into a polished report in our own format and style, and draft the letter to the editor from the report plus our recommendation. None of us asks it to review the paper for us; the substantive judgment stays with us. What gets outsourced is the formatting and the phrasing.
- **Letters and nominations.** As with referee reports, recommendation or nomination letters require substantive judgment. So, the first draft should originate from you. Revisions and other versions, however, can be updated by AI based on a style template you provide.
- **Proposals and budget narratives,** drafted and then trimmed to a page limit.
- **Meeting prep and follow-up.** Turn discussion notes into a clean summary for coauthors. Prepare talking points for a meeting with a visitor or an administrator.
- **Teaching.** Updating a syllabus or an assignment works well. Writing new problem sets from scratch worked less well for us; the AI wanted to explain general equilibrium rather than write a problem about it. What did work: write the problem yourself, have the AI solve it to catch errors, and then have it produce the LaTeX.
- **Slides.** When you have an early draft or a completed paper, this method works well, but only if you supply the constraints, such as how many slides, what structure (motivation, design, results, robustness), and what style.
- **Meeting notes.** Zoom's built-in AI notes worked well for us. The "My Notes" option transcribes and summarizes, and the summary is good enough to paste into an email to coauthors. We also tried a third-party notetaker (Calendly). The notes were good, but it joins the meeting as a separate participant that you have to explain to everyone, and the notes live in yet another account.
- **Reading the firehose.** One of us has the AI scan NBER and journal table-of-contents emails and flag relevant papers against a template. The results were mixed: the fastest platform missed a lot, and the more thorough one was slow. But it can run overnight while you sleep.

5. Level two: letting the AI into your folders

The shift that mattered most for all of us was moving from the chat window to the desktop apps that can read and write files: Claude Code and Claude Cowork on the Anthropic side, Codex on the OpenAI side. Instead of uploading a draft and downloading a revision, you point the app at a project folder and it can read the draft, the code, the data, the log files, and the output tables all at once. That is what makes it possible to say “check whether the numbers in the text match Table 3” or “the standard errors in column 2 look wrong, figure out why.”

Why we hesitated, and why we shouldn’t have. The setup, the permissions, the idea of an AI writing into our folders: it all felt intimidating. In practice, the learning curve was steep in the good sense. We learned the workflow in days and started using it heavily. If you take one thing from this document, it is to try the desktop app sooner than you think you are ready.

Setup tips that would have saved us time:

- **Sync everything through Dropbox.** Overleaf drafts can sync to Dropbox (the integration writes to Dropbox\Apps\Overleaf). Note that the Dropbox-Overleaf integration requires a paid Overleaf subscription. Make sure that folder is set to sync locally on your computer; one of us spent a while wondering why it wasn’t there. With drafts, code, and data all in Dropbox, the AI can see the whole project, and you can work from more than one machine.
- **Refresh, don’t just recompile.** Overleaf can lag behind Dropbox. After editing in the browser, do an actual page refresh so your latest edits land in the local Dropbox copy before you ask the AI to work on the draft. Check the Dropbox icon in your system tray to confirm the .tex file has updated.
- **Back up before it edits.** Ask the AI to make a backup copy before it overwrites a .tex file or a do-file. Overleaf and Dropbox both keep version history too, so a bad overwrite is recoverable, but a backup is faster.
- **Settle on a folder structure early, and tell the AI what it is.** If you don’t, it will invent one, and your files will end up scattered across platforms and computers. You can ask the AI to help design your folder structure right from the start.
- **Have it flag, not fix.** When the AI finds a discrepancy between your tables and your code, one of us strongly prefers being told about it rather than having the AI re-run things and “fix” them. Auto-fixing can take a long time, and along the way it often creates and deletes temporary files, leaving you no way to verify what it did.
- **Turn on the permissive mode, eventually.** By default the desktop apps ask permission for every file they read and every tool they install. Most of us switched to the “auto” mode after a week because the constant prompts were too much to manage. That is a trust decision; Section 7 is about how we check the work.

Which Claude app? (as of September 2026)

Claude's desktop app has two modes that can read and write your files, Cowork and Claude Code, and the difference confused several of us. Our rule of thumb: Claude Code is the more advanced one and can do everything Cowork does, though it may be overkill for non-technical tasks. It organizes conversations by project folder and, in our experience, carries more background knowledge from one chat to the next, which makes it a good home for ongoing research where you return to the same code, data, and drafts week after week. Cowork is simpler and may be all you need for email, scheduling, or drafting documents.

Two cautions. First, the two modes don't share folders or chat history. To move a project from one to the other, ask Claude to write a briefing file summarizing where things stand, then open a new chat in the other mode and have it read the file. Second, both are different from running Claude in a terminal, which we are told is the most powerful and least beginner-friendly option. None of us went there. (Running Claude in a terminal gives you direct control over your local file system, making it easy to automate complex multi-file workflows. However, this added speed and power come with a tradeoff: it requires basic command-line comfort and careful API usage management to avoid unexpected API costs or rate limits.)

Codex, ChatGPT's equivalent, works similarly. One of us likes that it lets you organize chats into self-chosen folders (research, admin, and so on). Claude Code doesn't, which is annoying.

Memory. Both major platforms remember context within and across projects, which is what makes the literature search and the writing help project-specific. The downside is that when you are working on unrelated tasks, the AI sometimes tries to connect them in ways that make no sense.

6. Coding and empirical work

This is where the biggest productivity gains were, and also where we are most cautious. A few of us now have the AI write essentially all of our new code, while we spend our time reviewing what it did and thinking about the big-picture questions: is this the right approach, is that the right sample, does this specification make sense? But in every case, we started with code we had already written ourselves. The AI built on our templates. As we start new projects from scratch, this could change, and we are not sure yet how.

Our experience is limited to applied, reduced-form empirical work, mostly in Stata.

Make it write the plan before the code. A bad plan is cheap to reject. Bad code is expensive to debug, and worse, it can head in a direction you didn't want or make decisions you would have made differently. So give it lots of context, then ask for a plan. Ask it to list the assumptions it is making and the decisions that need your input before it starts. Tell it to stop and ask you about any substantive coding decision, and give examples of what counts as substantive: sample restrictions, how a variable gets coded, how missing values are handled. This is a good place to use the most capable model, because it preempts a lot of wasted work.

Make it check itself. Instead of prompt, code, results, structure the work as plan, implement, verify, revise. After each step, ask the AI to inspect its own code and output against the plan and check for the usual suspects: bad merges, unexpected changes in the number of observations, sample changes, variable construction errors. Then move to the next step and repeat.

Verify at each step, not at the end. Catching a bad merge after the tables are built is expensive. Catching it at the merge is not.

Match the model to the task. Use the strongest model for identification reasoning, subtle debugging, and anything where a wrong assumption is costly. Once you have a template you trust, say the code that processes one year of data or produces the main results table, a faster and cheaper model can replicate it for additional years or for robustness checks that are variations on the main specification.

Translation and replication. This is where it is strongest. One of us had it replicate a Stata difference-in-differences analysis in R and Python, which doubles as a robustness check on the original. Turning a messy project folder into a clean replication package is another task it handles well, with the caveat that it sometimes confuses raw and processed data.

Log the decisions. When it makes a coding choice, prompt it to state the choice, the alternatives, and the reasoning. Have it keep a running file.

Name your versions. Exploration generates a lot of versions. Label each attempt (Version 1, Version 2), or better, use file names that say how the versions differ (`_withcontrols`, `_pre2002only`). You can ask the AI to maintain a master file that summarizes the versions, like a table of contents. This makes trying a new specification cheap: have it create a new version of all results without overwriting the originals, plus a short briefing you can share with coauthors to discuss which version is right.

Clean up as you go. The flip side of cheap exploration is file proliferation. Several of us ran into disk-space limits on laptops and Dropbox. Ask the AI to identify outdated files, suggest which versions to archive, and clean up the abandoned rabbit holes. It can be good at this if you make it. A colleague suggested git for version control; none of us has fully adopted it yet.

Break the project into small, organized tasks and sessions. Use a main session to establish the overarching structure and map out subtasks, then spin off smaller, separate sessions to execute each individual task. This modular approach is far more efficient, both for your own focus and for managing the AI's context window and usage limits.

The unresolved question. Whether to trust the AI to run an empirical project start to finish, autonomously, is something we genuinely haven't settled. We are comfortable with it as a very fast, very capable RA whose work we check. We are not yet comfortable with it as a coauthor whose work we don't. That may change. We would rather say so plainly than pretend otherwise.

7. Checking its work without reading every line

The big worry with AI-written code isn't syntax errors. Those it catches itself. The worry is silent decisions: a sample restriction, a way of handling missing values, a merge type, that it makes on your behalf and doesn't mention. The plan-first approach in Section 6 catches a lot of these, but not all of them.

The good news is that you already know how to do this. You have been checking work you didn't do yourself for years: RAs, coauthors, your own past self. Most of those habits transfer directly.

Our go-to checks:

- **Tell it what should be true, and have it confirm.** After a merge: are identifiers unique? What was the match rate? What happened to the sample size? For key variables: value ranges, summary statistics, and missingness by year or subgroup. Do the correlations between variables look sensible?
- **Benchmark against something you trust.** Replicate your earlier results before you expanded the sample or changed the specification.
- **Have the platforms check each other.** Have Codex review Claude Code's work and vice versa. This is easy and it catches real things.
- **Make it report its defaults.** Ask it to list every decision it made and every default it accepted.
- **Spot-check the critical places yourself.** Main results, main merges, key variable construction. Not every line, just the ones that matter.
- **Lightweight diagnostics over line-by-line review.** Summarize key variables by year and look at whether the pattern makes sense.

One thing we noticed: the AI found real errors in our own code, but usually not when we asked it to do a sweep. It found them while doing something else, like running a new robustness check. That is an argument for letting it into the project rather than only asking it to audit.

Where we are headed. Several of us are trying to move from checking each piece of code by hand to setting up standard diagnostic checks that run automatically after the AI writes anything. We haven't gotten there yet.

8. Guardrails: data agreements, journals, and disclosure

Data use agreements. Some public datasets restrict AI use. Many restricted-access datasets do. Read your DUAs carefully for AI language, and re-read every new DUA you sign, because AI rules are changing quickly. If your data are restricted, you cannot hand the AI the data file, so no "here's the folder, run the analysis." Two workarounds kept us inside our agreements:

- **Code only.** Have the AI write code that you run yourself. Generate a codebook first (a short do-file can do this), give the AI the codebook, and it can write correct code without ever seeing a row of data. Make sure you only export information in agreement with your DUA.
- **Synthetic data.** If the data live on a remote server where the AI can't go, create a synthetic dataset with the same variable names and moments (means, standard deviations). Export it, develop the code with the AI on your own machine, then run the real thing on the server. Some servers will not allow you to upload files, so check whether this is possible under your specific DUA.

Either way, the restriction changes the cost-benefit calculation. If you can't use the coding features to their full extent, the top-tier subscription may not be worth it for you.

Journal policies. They differ, and they are moving. Most journals allow, and often encourage, using AI to edit language without disclosure. Some require an explicit description of AI use beyond language editing: drafting, coding and analysis, study design, interpretation of results. Some restrict AI-generated

images, which may or may not extend to figures. Read the policy for each target journal, and read it again at submission, because it may have changed since you started the paper.

Coauthors and funders. Set the ground rules with coauthors up front. Some co-authors may feel uncomfortable with various AI tools. Some government agencies and foundations have explicit rules around use of AI to prepare proposals.

Over-reliance on the first draft. Once an AI-written paragraph exists, it is hard to get away from it. This is an argument for deciding the scope of AI involvement up front, not mid-task. It is also an honest worry some of us have about our own habits: if the AI makes first-pass polishing too easy, do we lose the discipline of doing it ourselves?

Gender and the perception of AI use. An emerging concern, raised in our group and elsewhere, is that disclosure of AI use may be handled differentially by gender and be read differently depending on who is disclosing. We don't have a solution. But it complicates a decision that otherwise looks purely procedural, and it is worth naming.

9. What it costs and what we would pay for

We spent the grant deliberately trying different platforms and tiers so we could compare. Here is the honest accounting, as of September 2026. Prices will have changed by the time you read this.

Tool	Cost	What it does	Our verdict
Claude (Anthropic)	\$20/mo standard; \$100 to \$200/mo top tier	General platform. Chat plus desktop apps (Claude Code, Cowork) with file access. Best for us: paper feedback, coding, writing edits.	Several of us kept the \$100 tier. At \$20, the desktop app hit usage limits repeatedly on long paper and coding sessions.
ChatGPT (OpenAI)	\$20/mo standard; \$100 to \$200/mo top tier	General platform. Chat plus the Codex desktop app. Best for us: coding, admin drafting, slides. Chats can be organized into folders.	Kept, at least at \$20. One of us upgraded to \$100 once usage got heavy.
Gemini (Google)	Free tier	General platform. We used it as a cross-check for literature searches.	Occasional use.
Elicit	About \$50/mo	Literature review. Builds detailed tables of related papers (paper, method, findings) and suggests related topics.	Nice tables, but a general platform found a very similar set of papers. Not kept.
Consensus	Subscription; also available as a Claude add-on	Reports the consensus in a literature and rates claims by the supporting evidence.	Less useful when you already know your field; may help when pivoting to a new one. Mixed. Claude + Consensus: Improves Claude's ability to search and synthesize the academic literature, but its usefulness is somewhat limited by Consensus's monthly usage caps.

Tool	Cost	What it does	Our verdict
ResearchRabbit	Free	Builds citation-network diagrams from seed papers, showing which papers are central.	Pretty and useful for seeing the structure of a literature. Not much beyond that.
Refine	About \$120 for three reviews	Technical consistency checker for drafts: mismatched table numbers, model as written versus described. Not a substantive reviewer.	Does what it says. We didn't need it often.
NotebookLM (Google)	Free	Upload documents and ask questions about them.	Useful for understanding one paper; less needed now that general platforms handle long documents.
Zoom AI notes	Included with Zoom	Meeting transcription and summary.	Kept. "My Notes" is the good option.
Calendly AI notetaker	Subscription	Joins the meeting as a separate participant and produces notes.	Notes were good; the logistics were awkward. Not kept.

The cost arithmetic.

- **Stacking specialist subscriptions adds up fast**, and for most of what we tried, a single upgraded general platform did the same jobs. One of us concluded it was more worthwhile to invest in general platforms than to spread money across specialist tools that do one thing a bit better and everything else worse. Someone with more funding might decide differently.
- **Two \$20 plans, one Claude and one ChatGPT, cost less than one \$100 plan** and have real advantages. You can have them check each other's work, and when one hits its usage limit you switch to the other.
- **But the top-tier models are where the magic is.** The models we found most useful for deep paper feedback and hard debugging are only available on the \$100+ tiers or via purchased credits. On the \$20 tier, the desktop app hit hard usage limits precisely on our highest-value uses: long paper reviews, multi-file Stata pipelines, refining referee work. If your use is heavy, the \$100 tier stops being a luxury.
- **Check your institution first.** One of our departments started offering an institutional Claude account partway through the summer. Ask before you pay.
- **Model quality moves around.** New models tend to be great when released and sometimes feel worse a few months later while the next one is in the works. Having two platforms hedges against that too.

The productivity paradox

A warning from one of our meetings. Upgrading to the high-capacity tier didn't just make us faster; it made us more tired. When you can run three projects in parallel, you do, and the cognitive load of managing all of them at once is real. Multitasking with AI is feasible and efficient. It is not restful. Budget your attention, not just your subscription.

10. Quick reference

10.1 Cheat sheet

Task	Where	What to say
Explain a Stata command	Chat	“Walk me through reshape from wide to long with a small example.”
Feedback on a draft	Desktop app pointed at the draft folder	“Read the draft and give me big-picture comments: is the argument clear, do the results support the claims, what would a skeptical referee say? Don’t edit anything yet.”
Edit a paragraph in my voice	Chat or desktop	“Here is a paragraph. Give me three tighter versions that keep my voice and don’t hedge.”
Understand a paper	Chat, PDF uploaded	“Explain the identification strategy in this paper as if to a colleague. What are the key assumptions, and how does this differ from a standard two-way fixed effects design?”
Literature sweep	Chat, on two platforms	“Find the ten most relevant papers on X since 2015. For each, give the question, design, data, and main finding, and tell me which are published and where.”
Plan before code	Desktop app	“Before writing any code: write a plan, list every assumption and decision, and ask me about anything substantive (sample restrictions, variable definitions, missing data). Wait for my answers.”
Check a merge	Desktop app	“After the merge, confirm identifiers are unique, report the match rate and sample sizes before and after, and show summary statistics for [variables] by year.”
Referee report	Chat	“Here are my rough comments on a paper. Turn them into a polished referee report in the format of the attached example, keep my judgments, and draft a short letter to the editor recommending [X].”
Replication folder	Desktop app	“Turn this project folder into a clean replication package: separate raw data, processed data, code, and output; write a README; flag anything you’re unsure is raw versus processed. Don’t delete anything.”
Move between apps	Either	“Write a briefing file summarizing this project, where things stand, and the open questions, so a new session can pick up where we left off.”
Plan my month	Chat	“Here are my deadlines and goals for October. Build a timeline by project and a daily plan for this week.”

10.2 Phrases worth stealing

- “Stop and ask me before making any substantive decision. For example: ...”
- “Make a backup of the file before you edit it.”
- “Flag discrepancies. Don’t fix them.”
- “Report every default you accepted and every decision you made on my behalf.”

- “Give me the plan first. Don’t write code until I say go.”
- “Check your own work against the plan before you tell me you’re done.”
- “Which of these files are outdated and safe to archive?”
- “Never invent evidence, literature, or material. If you don’t know, say ‘I don’t know.’”

11. Further reading

- **Paul Goldsmith-Pinkham, “Getting started with Claude Code”** (Substack). An accessible orientation for researchers new to Claude Code. <https://paulgp.substack.com/p/getting-started-with-claude-code>
- **Pedro Sant’Anna, “Claude Code: My Workflow.”** A more advanced academic workflow. Much of it is beyond where we are, but it is useful for seeing where this can go. <https://psantanna.com/claude-code-my-workflow/>
- **Scott Cunningham, the Claude Code series on Scott’s Mixtape Substack.** A long-running series on agentic coding for empirical research. <https://causalinf.substack.com/s/claude-code>
- **Aniket Panjwani, The AI Economist newsletter.** Weekly notes on Codex, Claude Code, and model updates, only when they matter for economics research. <https://aieconomist.io/>
- **Matthew Gentzkow and Jesse Shapiro, “Code and Data for the Social Sciences,”** and **Sean Higgins’ RA guide and Stata guide.** Not about AI, but the habits they teach (folder structure, version naming, documentation) are exactly what makes AI-assisted work go smoothly. <https://web.stanford.edu/~gentzkow/research/CodeAndData.pdf> and https://github.com/skhiggins/ra_guide
- **Anton Korinek, “Generative AI for Economic Research: Use Cases and Implications for Economists,”** Journal of Economic Literature (2023), with updates every six months. The August 2025 update, “AI Agents for Economic Research,” is a hands-on introduction to agents. <https://genaiforecon.substack.com/p/ai-agents-for-economic-research>
- **Will Rinehart (American Enterprise Institute), “AI Tools for Economists and Policy Analysts”** (August 2025). Tool overviews, prompting strategies, and use cases, pitched at intermediate users. <https://www.aei.org/technology-and-innovation/ai-tools-for-economists-and-policy-analysts/>
- **Jesse Lastunen, “AI for Economists.”** Prompt examples organized around Korinek’s use cases. <https://policy.fi/ai-econ/>
- **“Awesome AI for Economists”** (GitHub, maintained by the OpenEcon team). A curated list of AI tools and resources for economics research and teaching. <https://github.com/hanlulong/awesome-ai-for-economists>