Thinking With Agents

Webinar 1: Agentic Tools for Research

Emily Beam & **Erkmen G. Aslim**

University of Vermont · Department of Economics

June 15, 2026 · Live Online

Welcome! How this works

Questions

Drop questions in the Q & A section — we'll review as we go and raise them during or during Q & A time

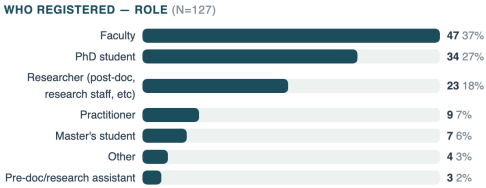
Recording

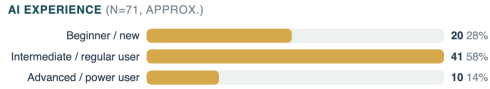
We're **not** posting the full recording publicly — but slides, skills, etc. will be posted on https://thinkingwithagents.github.io/

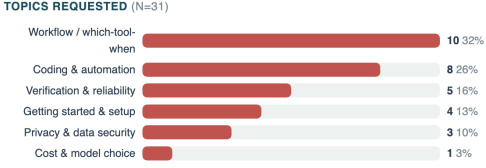
This is **mostly live**. Things may break — that's part of the honest picture.

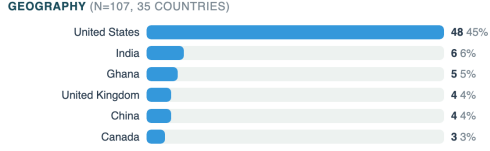
Who's in the room

Or at least, who signed up









Today's plan

Hour 1 — Emily

- **Foundations:** Getting started ⇒ context windows
- **Auditing your research:** AI as an adversarial reader of your code & your paper
- **Q & A**

Hour 2 — Erkmen

- Ideas & the **verification mindset**
- A **website-builder** demo (your CV → a live site)
- A live research **brainstorm**
- **Q & A**

Part 1

Foundations — what these tools are & how to start

Three ways to work with AI

1. Chat window

ChatGPT / Claude web

- ▶ Ask questions, paste text
- ▶ Can write code, make outputs
- ✗ Can't see your files

Like **texting a smart friend**.

2. In your editor

VS Code / Cursor

- ▶ AI inside your coding environment
- ▶ Sees the file you're in
- ▶ Autocomplete & inline edits

Like a **co-pilot** as you type.

3. Agent

Claude Code / Codex

- ✓ Reads & edits your files
- ✓ Runs commands
- ✓ Works across a project

Like an **RA sitting next to you** with unlimited caffeine.

The agent runs two ways: in your **terminal** (type `claude` / `codex`) or a **desktop app** — terminal gets new features first; the app is friendlier to start.

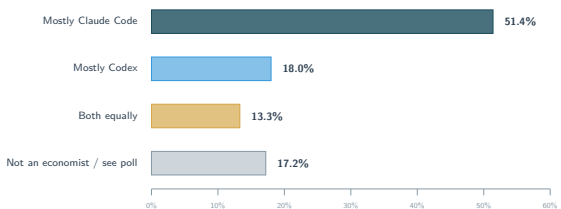
Today we're focusing on **agent-based**, because it's the biggest mindset/workflow shift.

Which agentic tool are economists using?

We'll show Claude — but use both

More tools = more better:

- New tools and features emerging constantly
- Burn through one's tokens? Switch to the other
- Different models are better at different tasks



Source: @aniketapanjwani poll, X.com — 459 votes, March 8, 2026

What we'll cover this hour

1. **Terms** — the vocabulary you'll hear
2. **Getting started** — subscriptions, apps, your first session
3. **Context & memory** — the one big concept, planning, and how it remembers
4. **Skills** — packaging what you do into reusable shortcuts
5. **Demo** — teach it your writing voice

Then the second half: **AI as an adversarial reader** of your code and your paper.

A few terms we'll use

Term	What we mean
Model	The underlying AI system (e.g., Claude Sonnet, GPT-5)
Token	The unit AI models process — roughly $\frac{3}{4}$ of a word
Session	One conversation or working thread with the tool
Context window	The working memory of the current session
Agent	A tool carrying out a bounded task with some autonomy
Artifact	A saved output you can pull back up and reuse — a prompt, file, table, or memo

What do you need to get started?

- **A paid subscription**: $20/mo Claude Pro or ChatGPT Plus will be enough to get started!
  - In general, ChatGPT (currently) subscriptions yield more usage/$
  - Heavier users upgrade to $100/mo or $200/mo subscriptions (perhaps requiring conversations with their spouses)
- **Terminal or AI desktop app (Claude or Codex)** — or both!

Claude desktop app

Codex desktop app

https://thinkingwithagents.github.io/install.html

Your first five minutes

Starting off

1. **Open it** in your project folder (or make a new folder)
2. **Ask** for something small: *"list the files and tell me what this project does"*
3. It **asks permission** to act — you **grant** it (once)
4. Repeat!
5. Have it save/log/create artifacts as needed.

Lab folder approach

- Lab folder (`eb-lab`!)
- Leave your project files, code, etc. where they are
- Generate a hub and project log

Fresh start approach

- Make a project-specific folder and copy stuff in
- Ask Claude to organize, sort, document
- Generate a hub, project log, etc.
- Work entirely within that folder
- *Even better: make it a GitHub repo for version control; save data outside*

Ways to talk to your agent

1. Type

Just type in the terminal or app. The default — nothing to set up.

2. Dictate

Shout at it. Talk instead of type — great for long, messy instructions.

Wispr Flow ($, but very good) dictates anywhere.

3. Notes in your files

Leave comments in your `.md`, `.tex` or do-files (`% AI: fix this`), then it reads them on the next pass.

What it can touch — and how you widen it

1. Default

Your working directory only.

Asks before each action. Where you start, and where most work happens.

2. Widen

Grant more as trust grows.

More folders, standing permissions for actions you've vetted — loosened deliberately.

3. Sandbox / VM

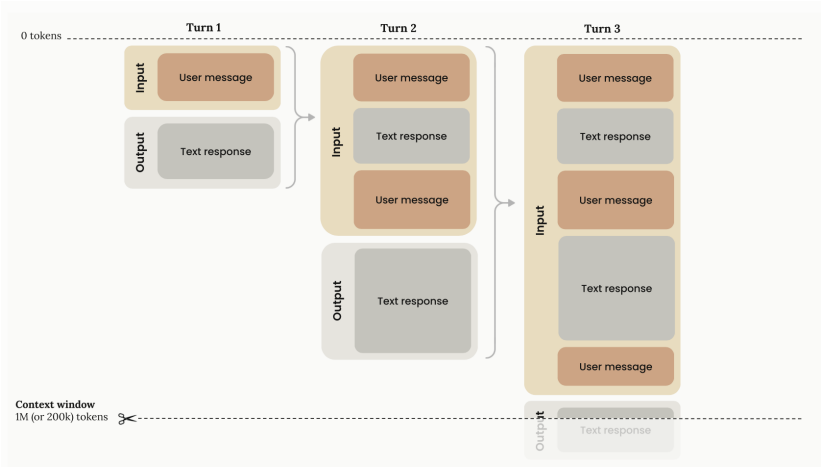
Isolated environment.

For big, autonomous jobs — it runs freely in a space that *can't* touch the rest of your machine.

Advanced

Real isolation: run the agent in a sandbox using Docker, virtual environment, or other tools — free rein inside a container that can't touch your machine. (See Paul Goldsmith-Pinkham's video on "Permissions and OpenClaw" for a great demo!)

How the context window fills up



Each turn re-sends the previous input *and* output — the whole history grows toward the limit, then gets cut.
Source: Anthropic — `platform.claude.com/docs` (Context windows).

Context management

1. **Long sessions degrade** — after 20+ turns, start fresh (`/clear`)

Context management

1. **Long sessions degrade** — after 20+ turns, start fresh (`/clear`)
2. **Write ideas to files** — files persist, context doesn't

Context management

1. **Long sessions degrade** — after 20+ turns, start fresh (`/clear`)
2. **Write ideas to files** — files persist, context doesn't
3. **Break work into chunks** — 5–10 turn focused sessions

Frequent intentional compaction is better than hitting the wall and letting auto-compaction lose details.

The "start fresh" pattern:

1. Write progress to `PLAN.md`
2. Start a new session (`/clear`)
3. "Read `PLAN.md` and continue..."
4. Full context budget restored!

Recent improvements mean this matters less — but still good practice.

Plan first, then build

Ask for a plan before it acts

For anything multi-step, have it **write the plan first** — then you read it, fix it, and say go.

- Catches wrong turns **before** they cost tokens
- You stay in control of the approach
- `Plan Mode`: it reads and proposes, but won't touch files yet

Planning saves time and tokens — invest upstream.

I spend most of my time planning, especially on complex multi-step projects.

Memory as markdown

Files are how it remembers

The agent **forgets between sessions**. Files give it memory:

- **Logs** — what happened
- **Hubs** — where a project stands
- `CLAUDE.md` or `AGENTS.md` — your standing rules

You don't have to build these yourself — **let the agent do it.** Copy one file over, or have it build a whole repo.

Advanced

A **handoff skill** can snapshot memory your way — see Chris Blattman's setup (`claudeblattman.com`).

Save what works — then build your own skills

1. Save & reuse

As you work, **save the prompts and records that worked** — in files you can pull back up.

Next time, **call them up again** instead of starting from scratch.

2. Package a skill

When a workflow **repeats**, bundle it into a **skill** — invoked by name (`/econ-audit`). A skill can:

- run **multiple steps** in order
- call **tools or sub-agents**
- be **shared** and re-run for reproducibility

We have some skills you can use, and they're fine.

A skill is just **a series of actions, packaged** — a shortcut, not a product. Most aren't special. Use them or take them as a template, or and build your own!

Demo: teach it your writing voice

A "voice file"

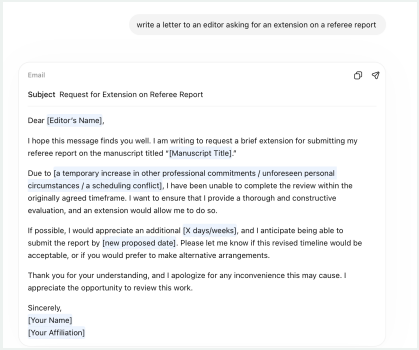
A short file describing *how you write* — tone, sentence length, habits, what you avoid. The AI reads it and matches your style.

Live: build one from a writing sample → then use it on a new task.

You can also instruct Claude to "remember it" or add to your CLAUDE.md file

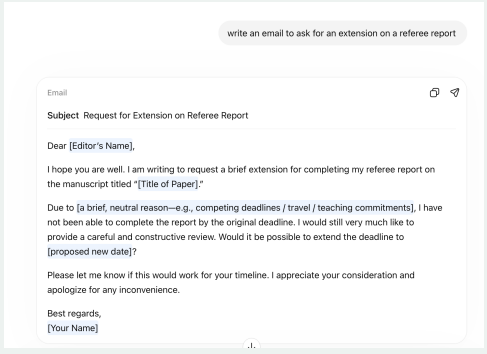
The voice file: making AI sound like you

ChatGPT (not logged in)



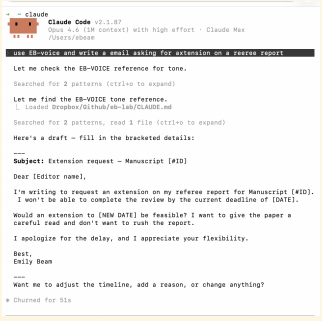
The voice file: making AI sound like you

ChatGPT (logged in)



The voice file: making AI sound like you

Claude + voice file



Same prompt, three different outputs. **Context is the difference.**

Creating a voice file

A "voice file" is a plain Markdown file that describes how you write — tone, structure, habits. The AI reads it and matches your style.

You don't write it from scratch. Give the AI samples of your real writing — emails, letters of recommendation, feedback — and ask it to describe your patterns.

The result is a file like `EB-VOICE.md` that you can edit, refine, and reuse across projects.

A sample prompt:

```
Read these five emails I've sent.
Describe my writing style: tone,
sentence length, how I open and
close messages, what I avoid.
Save it as EB-VOICE.md.
```

What you get back:

- Tone: direct, warm, slightly informal
- Opens with context, not greeting
- Short paragraphs, rarely $>$ 3 sentences
- Avoids: hedging, exclamation points

Using your voice file

Once the file exists

- Tell the AI to **load it automatically** via standing instructions
- **Call it on demand**: "Draft this email using my voice file"
- **Update it** as your preferences change — it's just a text file

Example prompt:

```
Draft a reply to this student
asking for a deadline extension.
Use my voice file.  Oh!  decline it.
```

Part 2

Auditing a replication package

Bangladesh remote education RCT

Before: 3 years of accumulated code

```
02_Do files/
01_master_5th June 2021.do
03_vargen_07sep.do
03_vargen.do
05_endline balance_eb.do
05_endline balance.do
09_attrition_8 July_e.do
09_attrition_8 July.do
...
Archive/ (18 files)
WB Submission Files/ (14 copies)
```

The problems:

- **51 do-files** across 4 subdirectories
- Master file from **June 2021** — 3 years stale
- Date-in-filename: `_07sep`, `_02July1130am`
- Author-in-filename: `_eb`, `_e`
- Hardcoded paths — only runs on my machine
- No documentation, no version control

This is **extremely common** in economics. You probably have a project that looks like this.

After: one clean, reproducible pipeline

Dimension	Before	After
Do-files	51	**8**
Master file	2021	Current
Config system	None	Template
Version control	None	Git
Documentation	None	Full
"Which version?"	Constant	**Zero**

86 old variants preserved in Archive/ — nothing deleted, fully reversible.

```
bd-remote-ed/
code/00_master_repo.do
code/07_Dofiles_academic/
_globals.do, 02_datagen.do,
03_vargen.do (THE one), ...
Archive/ (86 old)
config/config_local_template.do
docs/  justfile  README.md
```

Time: ∼2 Claude Code sessions (a few hours). **By hand:** days, minimum.

The bug it found: a typo hiding in plain sight

The `$flags` variable list

`$flags` = 17 missing-value indicators used as controls in **every regression**. Defined independently in 5 scripts.

3 of 5 scripts (wrong):

```
... _f_c_engaged _f_c_health
    _c_child_work
```

Missing the `_f_` prefix

Correct:

```
... _f_c_engaged _f_c_health
    _f_child_work
```

Result: `_c_child_work` appeared *twice* in regressions; the real missing indicator was left out. Survived 10+ versions across 2+ years. No error, no crash — just quietly wrong. **How AI found it:** cross-referencing every `global flags` definition across all 51 files.

Part 3

Auditing the paper

What is /econ-audit?

An Adversarial Review Skill

A reusable skill that reads a paper and acts as a **skeptical referee**.

- Reads the full paper (LaTeX, tables, appendix)
- Checks identification, power, measurement
- Produces a structured audit report
- Takes ∼60 seconds

What it checks:

1. **Identification:** Is the causal claim valid?
2. **Statistics:** MHT, power, attrition
3. **Measurement:** Are variables measured well?
4. **Methodology:** Design consistency
5. **Presentation:** Framing, length

```
$ npx skills add eabeam/econ-skills \
-skill econ-audit
```

What is /econ-audit?

An Adversarial Review Skill

A reusable skill that reads a paper and acts as a **skeptical referee**.

- Reads the full paper (LaTeX, tables, appendix)
- Checks identification, power, measurement
- Produces a structured audit report
- Takes ∼60 seconds

How I built it — same as you would

1. **Described what I wanted**; said we'd plan first
   - had it review existing skills to build on
   - pointed it to real papers + my referee reports
2. **Reviewed & edited** the plan
3. Had it **write the skill**
4. **Tried it**, gave feedback
5. **Published** it (`npx skills`)

```
$ npx skills add eabeam/econ-skills \
-skill econ-audit
```

Running it on the paper: 5 critical, 8 important, 6 minor

```
$ claude
/econ-audit main.tex
```

Output shown is from a saved run — AI output varies between runs, so I pin the version I'm reasoning about.

Three of the critical findings:

Critical

- **A1 — Identification:** the mediation claim (info → tutoring → learning) is not identified.
- **B1–B3 — Statistics:** no MHT adjustment across domains; headline effect below the MDE; underpowered heterogeneity drives the inequality story.
- **C1 — Measurement:** differential attrition in the learning sample ($p < 0.001$); suggested Lee bounds.

AI audit vs. real referees: 7 of 10 in ∼60 seconds

Referee Criticism	AI?	Notes
High attrition & differential attrition	✓	Flagged as Critical; suggested Lee bounds
Results don't tell a coherent story	◐	Caught ID problem, missed internal contradictions
Theoretical model not useful	✓	Flagged — but recommended *including* it
Short intervention duration (4–8 weeks)	✗	Requires field knowledge
Noisy learning measures (8-item phone test)	✓	Strong match; flagged IRT minimums
Paper too long and hard to follow	✓	Suggested cutting 20–30%
Multiple hypothesis testing issues	✓	Aggressive; added winner's curse framing
Limited external validity / COVID fatigue	◐	Partial; missed publication landscape
Deviations from pre-analysis plan	✓	Caught the spirit, not the specifics
Complex randomization hard to interpret	✓	Strong match

5 full matches, 2 partial, 1 opposite recommendation, 2 misses.
11 referee reports across 8 journals over 4 years — vs. 60 seconds of AI.

What the AI missed — and why it matters

✗ The Misses

- **Short duration:** 4–8 weeks is unusually short for an education RCT. Requires knowing field norms.
- **Internal contradictions:** the data arm raised tutoring *more* than the info arm, yet only info improved learning. Referees caught this; AI didn't.
- **COVID fatigue:** "we've seen too many of these." AI can't read the publication landscape.

The opposite advice:

The AI recommended *including* the conceptual model. Every referee wanted it **cut**.

Sometimes AI gives exactly the wrong recommendation — because it lacks field conventions.

Pattern:

AI reliably catches **statistical / econometric** issues.

AI misses **field knowledge**, **within-paper logic checks**, and **publication strategy**.

Takeaway: a very good pre-submission check

What It Replaces

- The first 80% of what a careful internal seminar discussant would say
- Catches the systematic stuff: identification, power, MHT, attrition
- In one minute, not one month

What it doesn't replace:

- Field-specific intuition
- "Does this make sense given what we know about Bangladesh?"
- Publication strategy
- Colleagues who know your work

Code audit: it found bugs I hadn't found in two years. That alone justified the time.

Paper audit: run it before you send a paper out. It's free, and it catches the embarrassing stuff.

Resources (and over to Erkmen)

Tools & skills

- Our work: `thinkingwithagents.github.io`
- Econ skills: `github.com/eabeam/econ-skills`
- Skills library: `skills.sh`
- Claude Code docs: `docs.anthropic.com/claude-code`

People worth following

- **Paul Goldsmith-Pinkham** — "Claude Code for Economists" video series; `paulgp.substack.com`
- **Aniket Panjwani** — The AI Economist; `aieconomist.io` & YouTube
- **Chris Blattman** — Claude Code for academics; `claudeblattman.com`
- **Scott Cunningham** — Scott's Mixtape, Claude Code series; `causalinf.substack.com`

Slides, links, and the follow-up survey come to your inbox after the session.