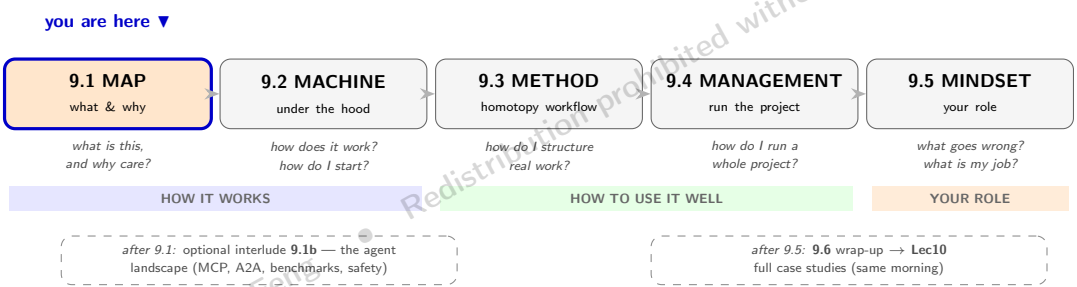
AI for Economic Research: Dynamic Models, Language, and Agents
Lecture 9.1: Framing — What Agentic AI Is and Why It Matters

Zhigang Feng

2026-07-06 07:04:49

Lecture 9 in One Map

One lecture, five decks, three questions: how agentic AI works, how to use it well, and what your role is.



Two research threads run through all five acts: **HA-Ramsey** *(a published research program, full case in Lec10 T8) and the* **ES Fellows dataset** *(a project you could build this week, full case in Lec10 T4). Here we study the workflows; the cases themselves come later this morning.*

Two Research Stories

Why economists should care — and what proper use actually produces

Why Economists Should Care: Execution Is the Bottleneck

The bottleneck in applied research is often not ideas. It is execution.

- Turning a good question into clean code, diagnostics, tables, and figures is slow
- Many research tasks are repetitive: reshape data, rerun specifications, trace sources, refactor scripts, update plots
- Agentic tools compress the time from *idea* to *first defensible result*
- That raises the return to the skills economists uniquely provide: question design, institutional knowledge, validation, and judgment

The promise is not "AI does research for you." It is faster implementation and iteration around a question you still define and defend. The rest of the lecture supplies what that takes: the machine (9.2), the method (9.3), the management (9.4), and your role (9.5).

Story 1 — HA-Ramsey: Weeks, Not Years

Proper use of an agentic workflow turned one published solver into a new research program.

Seed: a working, validated representative-agent (RA) "deep Ramsey" neural solver — the complete code behind Feng–Han–Zhu (2026).
Target: the heterogeneous-agent (HA) Ramsey model of Chien–Feng–Wen (2027) — the entire paper: idiosyncratic risk, borrowing constraints, a 4–5D state.
Outcome: a full HA research codebase *plus* three methodological discoveries that surfaced from the AI iteration itself — α-shapes for non-convex feasible sets, the admissible set as a viability kernel, and a Lyapunov-type penalty for saddle-path selection.

- The AI supplied **throughput** — LaTeX drafting, PyTorch translation, diagnostics, 32+ sweeps; "time from seed to working solver: **weeks, not years**; the binding constraint was economist judgment at the milestones, not implementation labor"

Why show this first: this is what "using it properly" buys. Not autocomplete — research.

Full case with the economics and the figures: Lec10 T8, later this morning. Source: HA-Ramsey case-study package.

Story 2 — 882 Fellows: A Dataset You Could Build This Week

The second thread is deliberately ordinary: build a credible dataset about the economics profession from the public web.

- **Task:** career metadata for all 882 current Econometric Society Fellows — election year, PhD year, research fields, career stage
- **Seed:** the official Society roster, split into 18 batches of 50; every filled field must carry a source URL from an official or institutional page
- **What makes it hard:** roster control, source discipline, state across sessions, auditable provenance — *bookkeeping, not intelligence*
- **What happened:** the agentic workflow scaled the search massively; the honest headline was about what still could *not* be measured (T5 tells that part)

Why two stories: HA-Ramsey shows the ceiling of the method. The fellows dataset is the floor you can stand on immediately — in Exercise B, you build one batch of 50 yourself.

Full case with the coverage results: Lec10 T4, later this morning.

From Chat to Agent

The loop, the four features, and the spectrum

What Modern Web Chat Already Does Well

As of mid-2026, a strong web-chat product is already useful for economists.

- Brainstorming model setups, empirical strategies, review rubrics, and robustness checks
- Summarizing papers, referee reports, and lecture notes
- Inspecting uploaded files and explaining code snippets
- Sometimes browsing the web or running limited sandboxed tools
- Helping you draft a specification before you open your editor

So the weak baseline is not "a browser cannot do anything." The weak baseline is a *single chat session without a durable harness*.

What Breaks Without a Harness: The Fellows Test

Try the fellows task in a plain chat window. It is a job of many sittings — and four things break within the first hour.

1. **Durable context is weak.** The task spans days of separate sessions. The coding rules you settle today — which sources count, how to record a missing PhD year — are not reliably remembered by tomorrow's fresh chat
2. **Reusable workflow is weak.** Your checklist lives in prompts you retype (slightly differently) each time, not in a file the tool reads automatically
3. **Controlled execution is weak.** The chat cannot open the roster file, write the CSV, or rerun a check — *you* become the copy-paste machine between the model and your folder
4. **Audit trail is weak.** Six weeks later: which source URL backed the PhD year of fellow #312? A scrolled-away transcript cannot answer field-level questions

That is the real contrast. Agents are not magic because they are "smarter." They are useful because they operate inside a structured project with files, tools, and repeatable workflows.

Chat vs. Agent: Same Brain, Different Arms

One key technical change made agents possible: modern LLMs (post-2023) support structured tool calls.

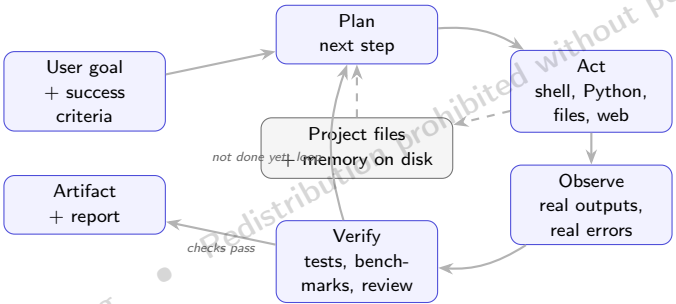
Instead of writing "you should run `python solve.py`," the model emits a machine-readable request — something like `{"tool": "Bash", "command": "python solve.py"}` — and a **harness**, software installed on your machine, executes it and returns the real output. (T2 opens this loop in machine language.)

	claude.ai (browser)	Claude Code (terminal)
LLM model	the same frontier model	
Harness	none (sandboxed)	local CLI on your machine
Can run Python?	no	yes—*your* Python, *your* machine
Sees real errors?	no	yes—loops back automatically

Same brain, different arms. The terminal agent is not a magically smarter model. The harness is what flips the switch from text-only output to text + actions inside a project.

The Agentic Loop: Plan, Act, Observe, Verify, Repeat

An agentic workflow is not one long generation. It is a control loop wrapped around tools, files, and checks.



- **Orchestrator:** the loop deciding what to do next — plan, act, observe, verify
- **Memory:** saved project state on disk, not just chat history (dashed arrows: the loop reads it and writes it)
- **Verifier:** tests, benchmarks, or review checks deciding whether output survives

What Makes Something "Agentic"? Four Features

- **Tool use** — invokes `pandas`, `curl`, `R`, `statsmodels`, the shell, whatever the task requires
- **Autonomy** — plans a multi-step empirical workflow without re-prompting.
  e.g., "download FRED → clean → regress → plot → save table" as one command
- **Iteration** — observes that a regression failed, identifies the bad column, fixes the formula, reruns
- **Persistence** — remembers your project conventions across sessions (variable names, preferred packages, validated benchmarks)

All four are cumulative. Tool use without autonomy = a glorified interactive console (a REPL — *read–eval–print loop*, like the Python ■> prompt). Autonomy without iteration = a script that dies on the first error. You need all four for "agentic."

The Spectrum, and Our Featured Tool

	Web chat	IDE assistant	Terminal agent
Best at	thinking, drafting	editing local code	end-to-end execution
Context	conversation + uploads	open files	full project tree
Typical role	architect's whiteboard	coding copilot	workflow operator
Economic example	design the specification	clean one regression file	run the full pipeline

Concrete example throughout this lecture: Claude Code, a terminal agent. **General lesson:** the same workflow logic appears in Codex CLI, Gemini CLI, Aider, and similar tools.

The only tool-comparison table in Lecture 9 — the workflow matters more than the tool. T3 returns to tool choice per pipeline step.

Four Research Patterns Across the Lec10 Cases

Pattern	Lec10 case(s)	Main risk if you use the wrong workflow
Reusable skill	Case 1: referee-style paper review	Generic comments, hallucinated quotes, missing quality gate
Empirical pipeline	Case 2: MEPS harmonization and figures	Wrong crosswalk, dropped survey weights, silent validation failure
Public-web dataset	Case 3: fellows metadata with field-level provenance	Weak sources, hidden missingness, no audit trail
Full-lifecycle collaborator	Cases 4–6: collaborator, 4-day sprint, HA-Ramsey	Polished output built on weak framing or claims discipline

Four patterns organize Lec10's six cases — and our two threads are instances: the fellows dataset (Case 3) and HA-Ramsey (Case 6, the full-lifecycle pattern at research scale).

Where RAG fits. Agentic RAG (Lec08 T4b) differs from single-shot RAG: the agent decides *dynamically*, via the same tool-call mechanism, whether and when to retrieve. RAG is one tool among many that the agent can compose.

What Remains Human Across the Four Patterns

You still own

- the research question
- the success criteria
- the coding and source rules
- the validation target
- the final interpretation

The agent accelerates

- repetitive execution
- file handling and refactoring
- promptable decomposition
- reruns after fixes
- draft diagnostics and reports

Why this matters for economists: the gain is not that judgment disappears. The gain is that disciplined execution becomes cheaper.

Hold on to the left column. In 9.4 it becomes a job description: the researcher as project manager.

Five Tests for Any Agentic Workflow

Criterion	Question to ask
Correctness	Did it compute, extract, or summarize the right object?
Reproducibility	Can a fresh run recreate the same artifact from the saved files and rules?
Auditability	Can you inspect the commands, checks, and source trail that produced the result?
Privacy boundary	Which files stay local, and which contents may still traverse the network?
Speed / cost	Once you count the tokens you paid for, the time spent checking and fixing the output, and the time spent supervising the run — did the agent actually save you time and money?

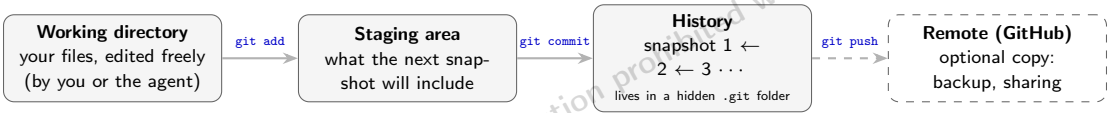
Why this matters for economists: an impressive demo is not enough. A research workflow has to be checkable and defensible. Keep these five tests in hand — every deck that follows is, in the end, machinery for passing them.

Your Safety Net

Git before anything else

What Git Is: A Time Machine for Your Project

Git is a version-control system: it keeps snapshots of a project folder so you can always see what changed and go back.



- A **commit** is a snapshot of the whole project at a moment you chose; the history is a chain of snapshots you can list (`git log`), compare (`git diff`), and return to
- Everything is local, plain files, no server required — the **remote** is optional
- **Why it fits agent work:** the agent edits fast and wide; snapshots give you a clean before/after boundary around every session

Git Is the Safety Net

The risk without Git:

- Claude Code can edit dozens of files autonomously in one session
- OneDrive and Dropbox sync by *copying*—two simultaneous writes create corrupt merge files
- No undo: if the agent makes bad changes, there is no "revert" without Git
- No audit trail: you cannot see what changed or when

What Git gives you:

- `git diff` shows exactly what the agent changed before you accept it
- `git checkout -- .` reverts all uncommitted changes instantly—your emergency brake
- `git log` shows every change with a timestamp and message
- Works offline; cloud sync races with the agent's rapid file writes

Practical rule: keep large data files on OneDrive; keep code and scripts in Git. Mixing them is workable if you are careful — but do not let OneDrive sync a *live repository* (the hidden `.git` folder): the two sync models collide and can corrupt the history. Keep the repo outside the synced folder, or exclude `.git` from sync.

Git Setup: A Four-Step Roadmap (Hand It to Your Agent)

You do not need to memorize commands. You need the roadmap — any LLM can walk you through each step.

Step 0 — check the install:

```
git --version
```

(if missing: git-scm.com or your package manager)

Step 1 — identify yourself (once per machine):

```
git config --global user.name "Your Name"
git config --global user.email "you@edu.cn"
```

Step 2 — start the project (once per project):

```
cd my-project
git init
echo "data/ output/ *.pdf" > .gitignore
git add . && git commit -m "initial setup"
```

Step 3 — the session rhythm (every session):

```
git add -A && git commit -m "pre-session"
# ... agent works, you validate ...
git add -A && git commit -m "validated: ..."
```


Emergency brake: `git checkout -- .` undoes *all* uncommitted changes instantly. And note: this setup is itself a perfect first agent task — paste the roadmap into the agent and say *"walk me through steps 0–3 for this folder."*

Bridge: Widen the Lens, Then Open the Hood

Act 1 framed the terminal-agent workflow: chat alone is not enough; a harness wraps tools, files, and verifiers around the model so the loop can do real work.

- You have seen *what* agents do, *why* the harness exists, and what proper use produced in two real projects
- **Optional interlude (9.1b):** the wider agent landscape — interoperability standards (MCP, A2A), benchmarks, and safety — before we commit to one tool's mechanics
- **Next (9.2):** the question changes from "what is this?" to "*how does it actually work, and how do I start?*" — components, install, initialization, the CLI, and your first session

Arc: **T1 Framing** → *T2 Under the Hood* → *T3 Homotopy* → *T4 Project Org* → *T5 Mistakes/Context* → *T6 Wrap.*