AI-as-an-RA

Christopher T. Kenny / Data-Driven Social Science

October 26, 2026

Roadmap

- Agentic AI: A refresher
- Why use AIRAs?
- Version Control 101
- AIRA workflows
- Application to a large redistricting project
- Common pitfalls
- Choosing a model

Agentic AI: A refresher

What is an agent?

Model

Proposes the next step or response.

Context

Project files, instructions, and previous results.

Tools

Search, read, write, run, and inspect.

Task -> Model -> Tool -> Result -> Model -> Answer or output

The agentic loop

Common agent harnesses: Codex, Claude Code, GitHub Copilot, Cursor, Gemini CLI, Posit AI; herein “claudex”

What is context?

- Context: all of the information the model has seen, as tokens
- Context is finite: only so much information can be included (**the context window**)
- Codex default limit: 272k tokens or ~500 pages

A token is approximately a word or a piece of punctuation.

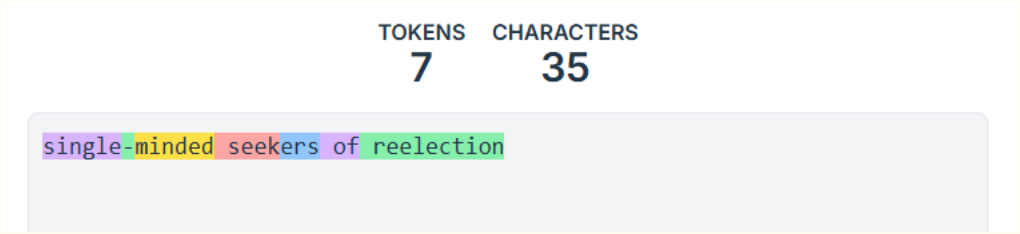


Figure 1: Mayhew on the goals of electeds

Why use AIRAs?

What do you use RAs for?

- Data collection and basic analysis
- Classifying data into specific groups
- Summarizing or reviewing additional data (e.g., open survey responses)

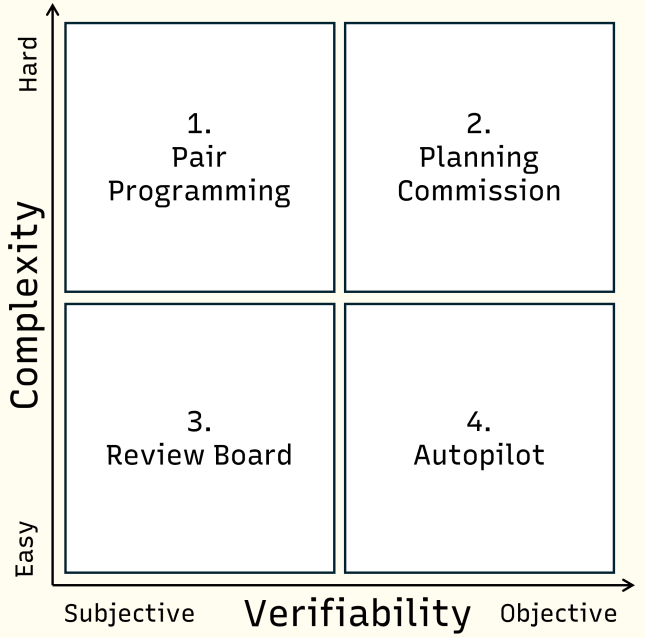
If you could automate it, you wouldn't use an RA (probably).

RAs help with *well-defined* tasks that require *customization* or *judgement*.

LLMs work well on *well-defined* tasks, are great at *customization*, and can simulate *judgement*.

Models of Claudex usage

1. Pair Programming: Human and LLM work together on every step
2. Planning Commission: Human builds a **plan**, LLM implements
3. Review Board: LLM plans and implements, human reviews
4. Autopilot: Human instructs, LLM implements and validates



Adopted from Colin Swaney's Spring 2026 "Claude Overload" DDSS workshop

Planning commissions are like RAs

- You build the plan, the LLM implements it
- You check in when specific goals are reached
- The LLM asks you questions when instructions are unclear

This is very much like managing **undergraduate** RAs, but with advantages:

1. An LLM can implement it all in tens of minutes or a few hours, rather than on a weekly cadence.
2. LLMs are available to help you finish tasks when working on a deadline, if you set up the tooling first.
3. LLMs are cheaper!
4. LLMs already know general info about your research topic.

Version Control 101

Collaborating with agents on your own computer

- Agents can edit files faster than you can stop them
- Unlike an RA, they are editing **your** files
- You need to be able to undo and review **all** changes

Git allows you to:

- See exactly what changed
- Make multiple copies of your project folder (**worktrees**)
- Make **pull requests** with proposed edits
- Revert bad changes to the last time you were happy with code

Git and GitHub

It's just a folder (a *repository*), with metadata in a `.git` subfolder.

Git: the software that handles version control.

GitHub: A popular and largely trusted (Microsoft-owned) cloud host for Git.

- Generous educational features (for faculty, students, and staff)
- Has a custom `gh` CLI that your agents know well

You can turn any existing work into a GitHub repo!

Drawback: file sizes are limited on Git. You can only track modest-sized files.

Recommendation: Store large data on Dropbox, SharePoint, etc. and *symlink* into the repo.

Git Vernacular

- **commit**: a collection of edits to one or more files
- **push**: send your laptop commits to the cloud
- **pull**: grab the cloud version to your laptop
- **branch**: a separate copy of the work that can be independently edited
- **pull request**: a proposal to copy commits from one branch to another
- **worktree**: a branch in a separate folder on your laptop

For working with an AI RA:

Pattern 1: Commit -> Prompt(s) -> Review diff -> Commit (or revert)

Pattern 2: Create a branch -> Prompt(s) -> Robot commits -> Review a Pull Request

Just use GitHub Desktop

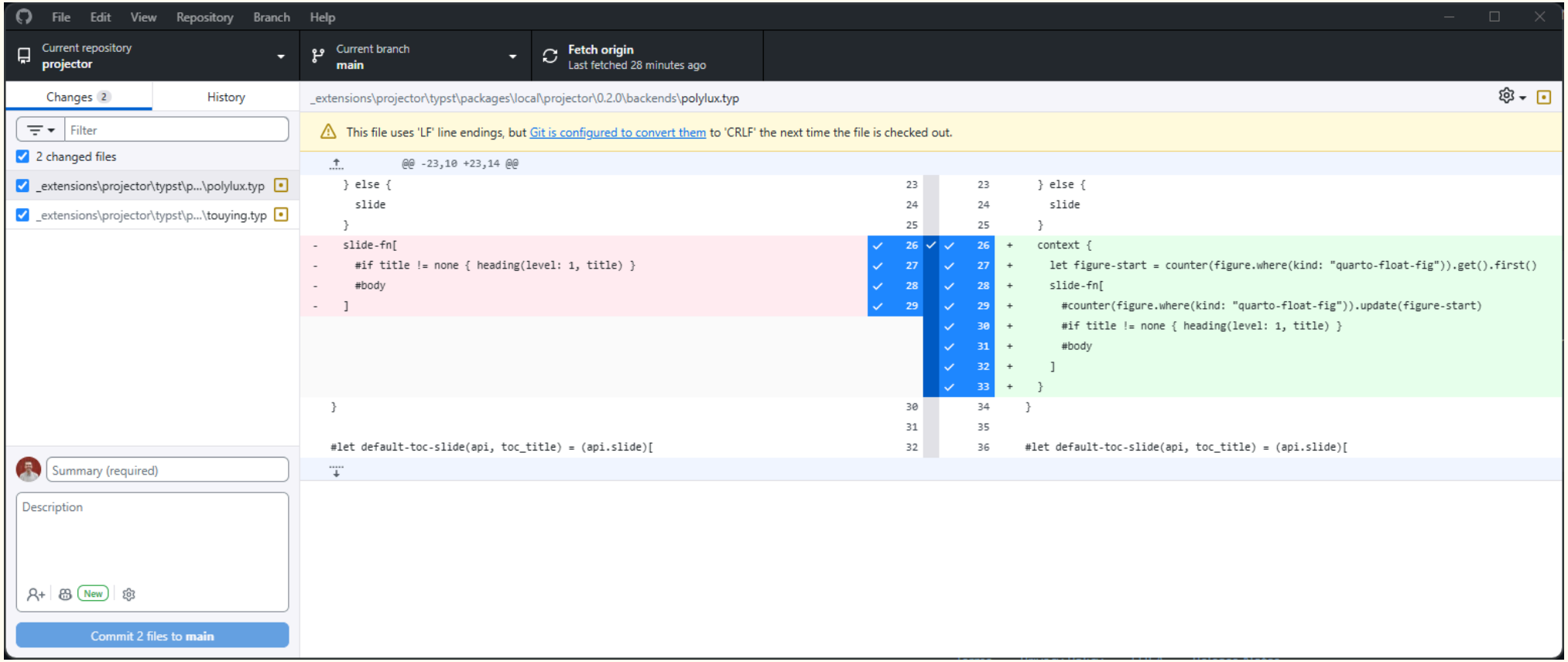


Figure 2: Make the commands into buttons.

Demo 1: GitHub and your Agents

- Using GitHub Desktop
- Reviewing a *diff*
- Reviewing a *pull request*

AIRA workflows

Transforming human instructions into agent instructions

Human RAs

- RA instructions focus on motivation, learning outcomes, and specific examples
- RA instructions assume learning over time, starting with easier tasks and moving towards harder ones
- “Good” RAs get pushed towards the hardest tasks

AIRAs

- Each run of an AIRA is a fresh context
- The base training means they have a lot more info
- They don’t learn (mostly) and need specific, succinct guidelines
- Instructions need to rely on **progressive disclosure**: relevant info available with links to additional info

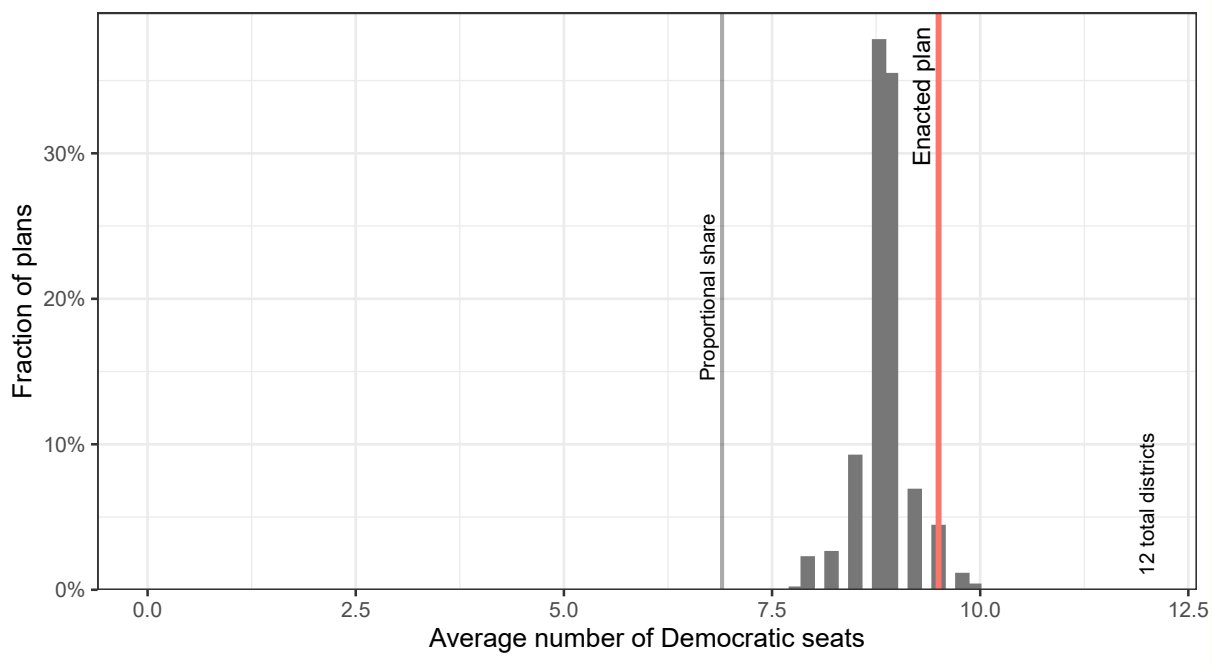
Skills, Rules, Hooks

- Skills: bundled prompts for repeatable tasks
  - Required structure: YAML header + markdown body
  - Rely on **progressive disclosure** to protect its context
  - Programming analogy: skills are like functions. If you are repeating a task, write it as a skill.
- Rules: transportable specific instructions
  - special case of CLAUDE.md/AGENTS.md
- Hooks: run automatic checks on tool use

Application to a large redistricting project

The fifty-states project

How do we assess if a redistricting plan is a gerrymander? Draw large sets of alternative plans and compare to the enacted plan.



From https://alarm-redist.org/fifty-states/.

The fifty-states project

- For every state, we simulate 5,000 redistricting plans that follow each state's legal criteria
- 25+ graduate and undergraduate students involved over the last 5 years
- Common code template that gets edited after interpreting the state's criteria
  - Not easily automatable:
    - every state has different rules
    - judgements are subjective
- Can we augment human work with AIRAs?
  - Demo 2: Converting human instructions into skill(s) and AGENTS.md instructions
  - Demo 3: AIRA in action

Demo 2: Skills vs Human Instructions

- The tutorial we use with new students redist 101
- The fifty-states contributing guidelines
- Instructions for the fifty-states agent
- R package instructions as a redistverse skill

Demo 3: An agent runs redistricting simulations

You are working on a redistricting project as the fiftystates-analyst agent in R as a research assistant. Your goal is to complete an analysis for congressional districts in [state] for the 2020 cycle.

Details for running analyses with these packages are available `/redistricting-analysis`. Then, review the full instructions for an analysis in CONTRIBUTING.md.

Follow all instructions in the CONTRIBUTING.md file. Once you are ready, you may begin for [state].

If you are unsure about anything, ask questions.

AIRAs vs. human RAs

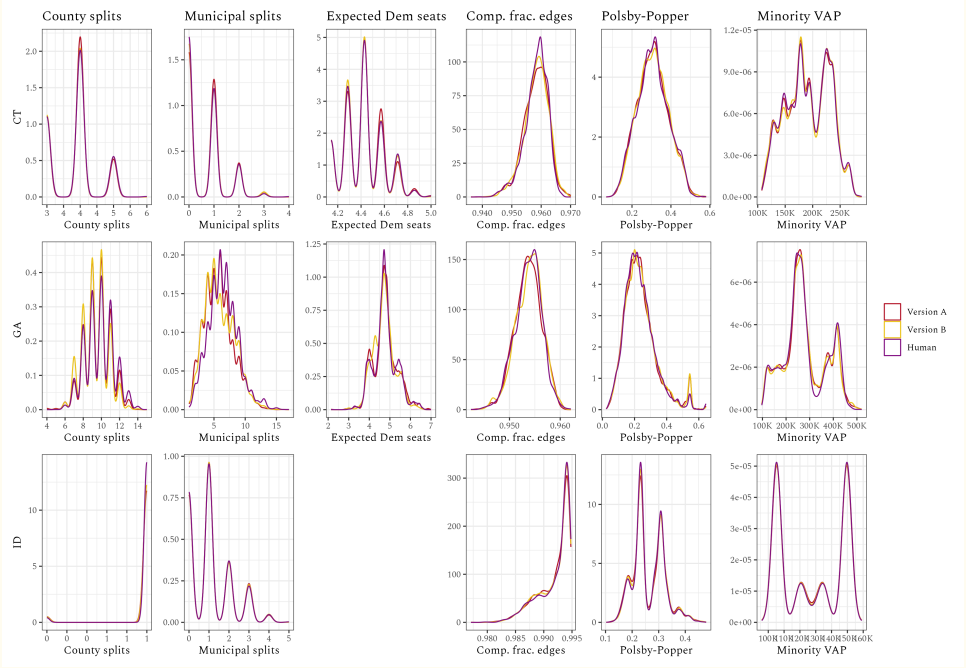


Figure 3: Human vs two agent baselines

Common pitfalls

LLMs want to finish the task

Things to watch for:

- Dropping data to make the code run (especially NAs)
- Suppressing errors and warnings (e.g., with `tryCatch()`)
- Implementing special cases (e.g., `if (state == 'CA')`)
- Swapping similar variables by name (e.g., `pseudo_county` vs. `county`)
- Ignoring rules `git add --force` on an ignored file

When you know what the problem is, have your agent write a **hook** for you to ban it.

Being insufficiently clear on guidelines

Claudex are set up to finish at clear cutpoints

These are not always the same points that you want

Stopping early wastes time

Going too far often wastes more time

Recommendation: Formalize checkpoints and check-ins. For example, only when a pull request is ready

Messy code

- LLMs are trained on lots of code
- Some of it is very bad
- You will get fatigued from reviewing bad code

Recommendation: Use a formatter and a linter

- A formatter edits the code to conform with a style: `air` or `styler` for R, `ruff` for python
- A linter provides feedback on preferred coding practices and flags bad ones: `jarl` or `lintr` for R, `ruff` for python

Choosing a model

Models

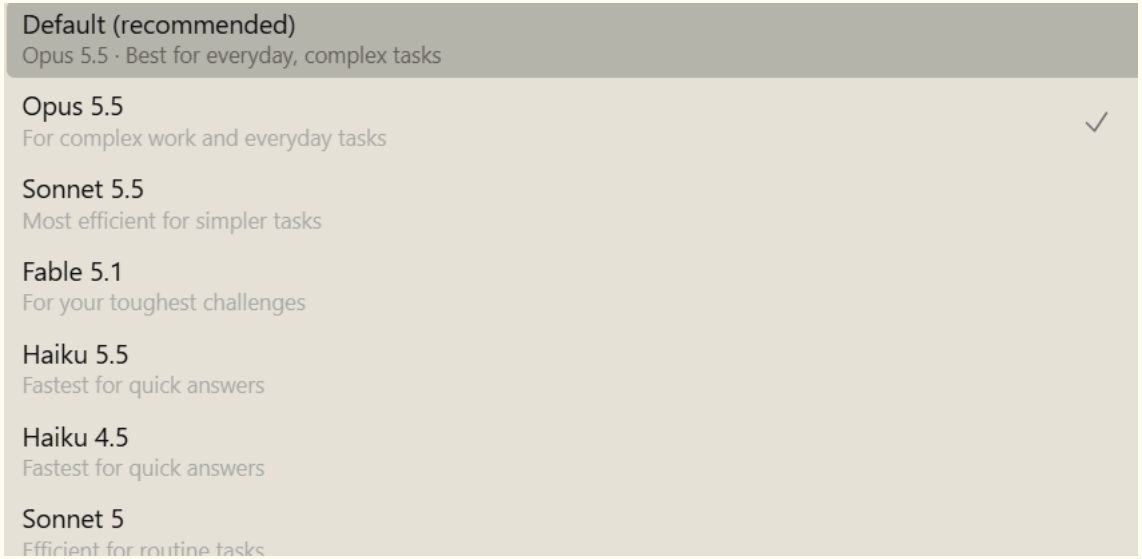


Figure 4: Claude model choices as of this morning

Tradeoff: big models "think" more deeply but may listen less

Goal: use the smallest model that completes the task

Effort levels

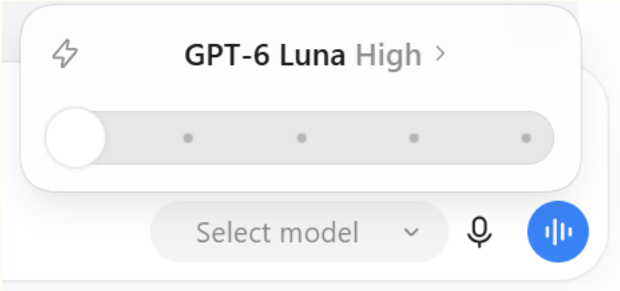


Figure 1: Low toggle bar

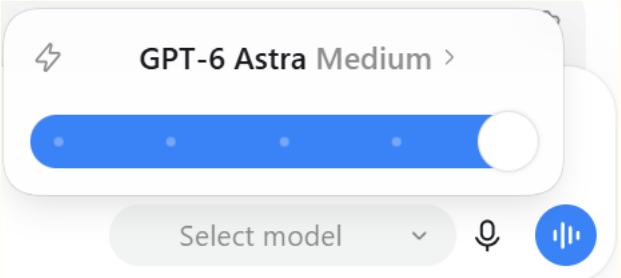


Figure 2: High toggle bar

- Each model has a recommended “effort” level
- Effort gives you **some** control over how long agents work on tasks
- Big model + big effort burn through your token limits fast
- Smaller models are cheaper to run
- You may want a smaller model with a bigger effort
  - e.g., Luna 6.0 Max <-> Sol 6.1 Medium

Picking your RA model

General workflow:

- Start small (Haiku 5.5, Luna 6.0) with High effort
- If it can't do it, step up once (Sonnet 5.5, Sol 6.0)
- If it can't do it, step up once (Opus 5.5, Sol 6.1)
- If it can't do it, restructure the task and/or instructions

Opinion: You probably don't need Fable or Astra (or Gemini Argon).

Opinion: Well-defined RA tasks that can be reviewed >> letting the model produce a lot of stuff at once. Treat it like working with an undergrad: regular review and clear success criteria.

Thank you!

Questions? *ctkenny@princeton.edu*