Agentic AI for Political Science Research

Christopher T. Kenny / Data-Driven Social Science

September 10, 2026

Roadmap

- What is agentic AI?
- Using agents in research
- Customizing agents with AGENTS.md and skills
- Verification and common dangers
- LLM Wikis

What is agentic AI?

LLMs: Chat vs. Agents

Chat

- Browser interface to models in the cloud
- You upload files on your own
- Great for explanation, brainstorming, and stand-alone text tasks
- Can’t do (most) things for you

Agents in a project

- App or terminal UI on your laptop
- Reads your files and instructions
- Runs commands and edits files for you
- Can iterate across code, data, and documents

An agent combines a model, context, and tools

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

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

What are tools?

- find and read files
- write or edit files
- run code and commands
- search the web
- inspect outputs, errors, and diff

Model chooses an action -> tool performs it -> result returns as context

What is context?

Context is the information a model has.

- the current conversation
- results from earlier actions
- system instructions, project instructions, and permissions
- relevant files, data, and documentation
- available tools and their descriptions

What is a context window?

- Context is finite: only so much information can be included
- Codex default limit: 272k tokens or ~500 pages

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

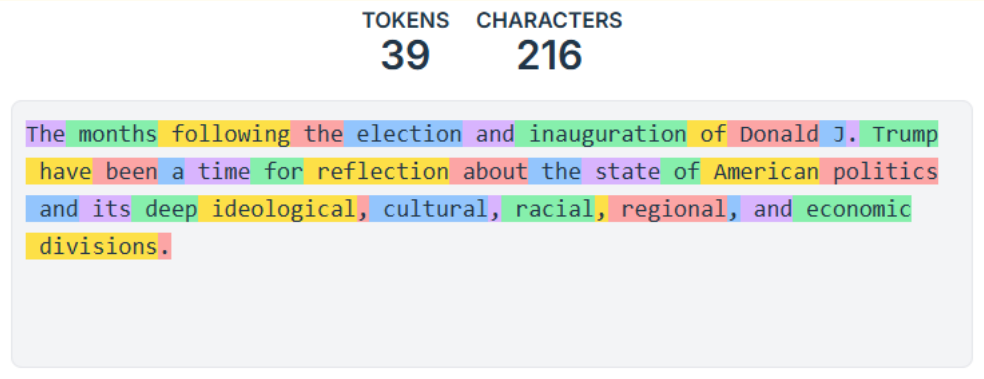


Figure 1: First sentence of McCarty's Polarization

What makes AI agentic?

- It works towards completing a task or a goal
  - Not just a single text response or unverified script
- The model selects the next action until it’s done
- It can inspect and change its own environment with tools
- Results return to the model as new context
- The cycle repeats until the agent stops or a person intervenes

Agents are not always right

Agents may:

- misunderstand the task
- work from incomplete context
- choose the wrong file or tool
- produce a result that looks right but is wrong

Wholesale hallucination is less likely these days

Bigger models + ability to read local documents help immensely

Yes, you need to use version control (git)

- Agents can change several files quickly
- Even if you're watching, it moves faster than you can
- You need to be able to undo and review **all** changes

Git allows you to see exactly what has changed

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

Just use GitHub Desktop

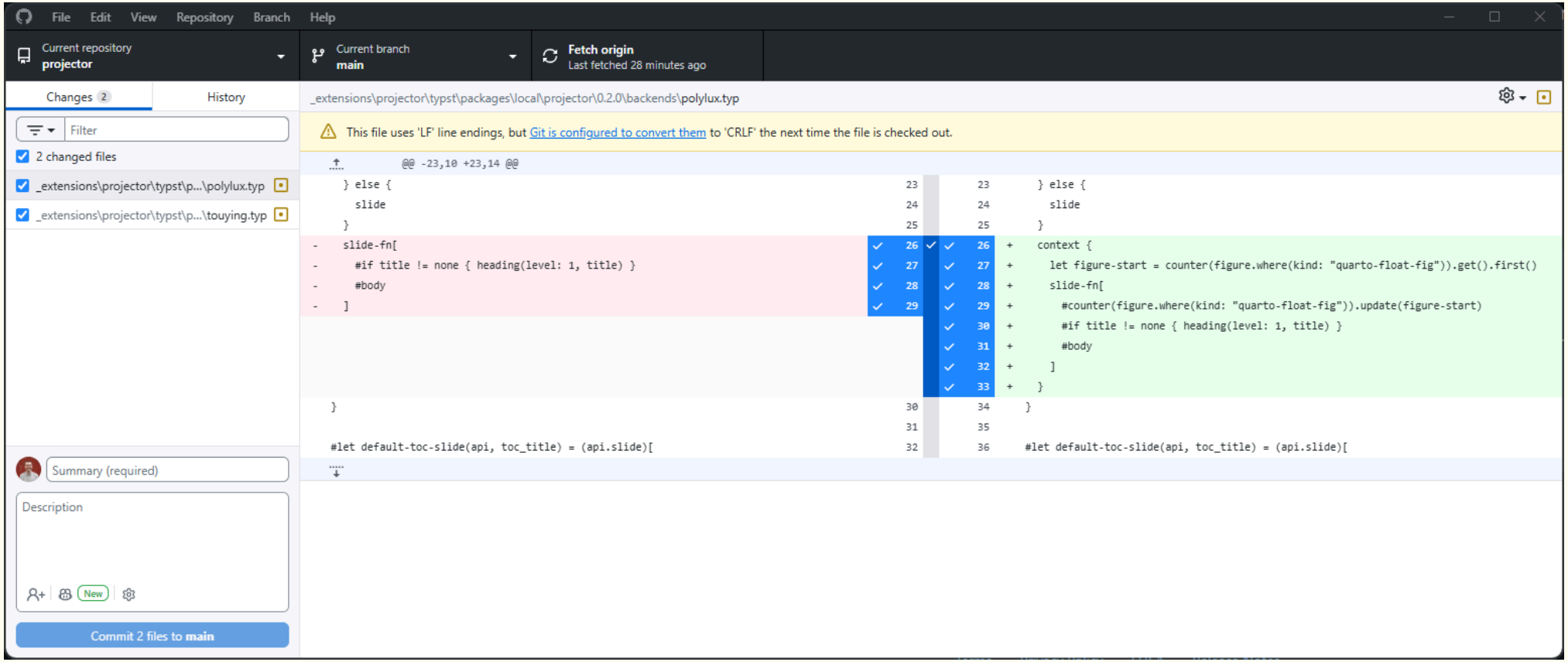


Figure 2: Make the commands into buttons.

Using agents in research

Research vs. Software Development

Software dev

- Goal: a working product or package
- Just needs to work
- Mistakes can be patched over time
- No one person needs to understand every feature (and **impossible** for large projects)

Research dev

- Goal: understand why or how it works
- Need to be quite confident that it works
- Mistakes need to be fixed before writing a paper and/or publishing it
- Authors of the research need to certify its correctness

Correctness matters *much more* for **research**.

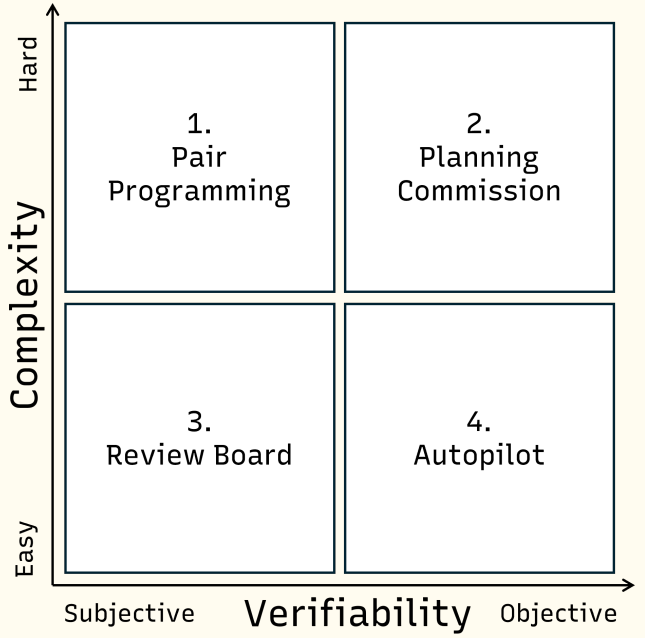
Guiding principles for Claudex in research

1. **Human agency**: Our goal is to conduct research as human researchers and produce knowledge for human understanding. We should lead research projects with the assistance of GenAI rather than letting GenAI take control of our research.
2. **Transparency:** One of the most important principles of scientific research is transparency. All researchers involved should disclose their AI usage to their collaborators so that we can have a proper disclosure to the scientific community at the time of research publication.

From the ALARM Project's GenAI Usage Guidelines, written with Kosuke Imai, Cory McCartan, and Tyler Simko.

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

The Research Workflow

- Verification requires tight supervision
  - Relevant models are “Pair Programming” and “Planning Commission”
- “Pair Programming” is like working on research alone
  - Implement a step, check it, repeat.
- “Planning Commission” is like a research assistant
  - Instruct them once, check results throughout
- Current state of the world: You design the questions, contributions, goals, etc.
  - The robot helps with the implementation

AI-as-an-RA Workshop

October 26, 2026 with Princeton Research Computing

Privacy and permissions

Permissions control what your local agent can do

- Read permission requests before approving them
- Keep secrets (e.g., API keys) out of the project

Privacy concerns: what if there's sensitive data on my machine

- Sandboxes help restrict
- Follow university, department, and data user agreement requirements

Not sure? Password protect the files or use claudex on a different machine

Customizing agents with AGENTS.md and skills

How to make claudex work in your way

- Make information available to the models
- Give it project-level instructions
- Teach it workflows and **your** practices
  - e.g., use the `tidyverse` not base R
- Include details on the goals of the project
- Explain what resources (e.g., data, literature) are available

Start thinking about digital notes

Agents aren't going anywhere. Plain text is best for agents. The LLMs can even turn your handwritten notes into Markdown files.

AGENTS.md and CLAUDE.md

- Instructions to guide LLM behaviors, applied to all sessions
- Claude uses CLAUDE.md, every other model uses AGENTS.md
- Recommended to be < 200 lines, which is a lot of space!
- Anything that you find yourself repeating goes into this

Write a stub of your AGENTS.md

Store a separate, short (60-100 lines) version that you can reuse across projects. Add project-specific details to the top.

The top of my (default) AGENTS.md

R analyst

Analysis posture

- Correctness before cleverness or volume. Code should work on real inputs, not just toy cases or happy-path examples.
- Do not declare an analysis done while the script, report, model, figure, table, or workflow still fails in practice.
- Stay inside the requested analysis. Do not invent extra scope or wander into adjacent tasks.
- When reimplementing an existing method, compare carefully against the reference and identify meaningful behavioral differences.
- Prefer readable, explicit analysis steps over dense abstractions that hide what the analysis is doing.

Reproducible workflows

- Keep portable entrypoints, such as a top-level `run.R`, when the workflow should be runnable directly.
- Use `here::here()` for file paths in analysis scripts. Never use `setwd()`.
- Call `set.seed()` before any randomness.
- If a script queries an API or external source, save the output rather than forcing every run to recompute it.

Style

- Write idiomatic R. Prefer well-tested tidyverse, r-lib, and Posit ecosystem functions over custom wrappers.
- Use the native `|>` pipe. Do not use `%>%` or import `magrittr`.
- Use single quotes for strings.

Full file on github.com/christopherkenny/skills.

Skills

- Bundled prompts for repeatable tasks
- Required structure: YAML header + markdown body
- Rely on **progressive disclosure** to protect its context
  - Model gets a description and invokes the skill to get more information
- Can contain additional references which describe more things
- Programming analogy: skills are like functions. If you are repeating a task, write it as a skill.

Example Skill: LaTeX to Quarto

name	latex-to-quarto
description	Converts LaTeX article-class .tex documents to Quarto .qmd format for multi-format publishing (HTML, PDF, Word). Use when asked to convert, port, migrate, or translate a .tex LaTeX file to Quarto; when porting an academic paper or manuscript from LaTeX to Quarto; when a .tex document needs to render to HTML, Word, or Typst PDF. Covers preamble → YAML front matter, section headings, text formatting, math, citations (natbib/biblatex), cross-references, figures, tables, lists, footnotes, hyperlinks, and special characters.
metadata	**author**: Christopher T. Kenny; **version**: 1.0

LaTeX to Quarto Conversion

Convert a full LaTeX `.tex` article to a Quarto `.qmd` document that renders correctly to HTML, PDF (Typst or LaTeX), and Word. This skill handles the entire conversion pipeline: preamble extraction, document structure, and all inline/block transformations.

This skill writes the output file (overwriting the input or writing to a second path). Before writing, tell the user what file will be changed and remind them that `git diff` or file history can recover the original.

Full file on github.com/christopherkenny/skills.

Example Skill: References

Quick Navigation: Use Which Reference

Task	Reference File
`\documentclass`, `\usepackage`, `\title`, `\author` → YAML	01-preamble-yaml.md
`\begin{document}`, `\maketitle`, `\begin{abstract}`, `\appendix`	02-document-structure.md
`\textbf`, `\emph`, `\texttt`, `\footnote`, `\href`	03-text-formatting.md
`equation`, `align`, `gather`, inline math delimiters	04-math.md
`\cite`, `\citep`, `\citet`, natbib/biblatex commands	05-citations.md
`\label`, `\ref`, `\autoref`, `\eqref`, prefix mapping	06-cross-references.md
`\includegraphics`, `figure` environment, subfigures	07-figures.md
`tabular`, `table`, `booktabs`, `\multicolumn`	08-tables.md
`\section`, `\subsection`, `itemize`, `enumerate`	09-sections-lists.md
Special characters, `\newpage`, `verbatim`, `\input`	10-misc.md

Full file on github.com/christopherkenny/skills.

Other standardized inputs

- rules: transportable specific instructions
  - special case of CLAUDE.md/AGENTS.md
- hooks: run automatic checks on tool use
- subagents: handle independent tasks in parallel
  - You can design custom subagents

Demo: The workspace

Demo 1: Plot out-party feeling thermometer by Dem and Rep ID for the last thirty years.

Verification and common dangers

Making things verifiable

- You need to be able to assert that all of the code is correct
- Have the bot add tests and examples
- Agentic AI enables more ambitious projects: this requires more organization
  - Break up the project into pieces with separate folders (or repos) for different tasks
- Use Git(Hub) diff to see exactly what is changed from step n -> n+1
- Have the robot make a plain text log of tasks with dates

Use a linter and formatter

These provide *automated* checks for your code and can fix simple things. Standardization here makes it easier to read and check!

LLMs want to finish the task

- It will listwise delete, almost every time
  - You need to tell it when that's right or wrong
- It will use `tryCatch()` statements so the code runs
  - Nitpick these to make sure they're not going to silently break things
- It will often use `suppressWarnings()` when you want those warnings
- It will add special cases rather than fixing the problem
  - Check out the `if` statements! `if (case == 'CA')` is a red flag
- It does the simple thing, most of the time
  - Check for things like ignoring weights or counts
- Add in a separate code review from an agent with a fresh context before you review

LLM Wikis

How do I give claudex expert information?

- Claudex are very general tools
- Academics want very specific information
- How do we get the LLM to review our own work, helpfully?
  - Give it the relevant papers
  - Summarize them
  - Assess the contribution of your paper
- Costly to do repeatedly. Papers are long and PDFs cost a lot of tokens.

Big idea: give the models a workspace to write memories and communicate across time.

Many choices! Today: Demo #2 - Karpathy's LLM Wiki concept