AI Coding Workflow

Workflow for using AI coding assistants (Cursor, Codex, Claude Code) as research assistants for economic research.

Overview

AI coding assistants can serve as RAs when properly supervised:

- Accelerate iterations
- Parallelize tasks
- Integrate research and implementation
- Improve implementation quality

Require the same supervision level as human RAs.

Setting Up: Cursor

1. Download from cursor.sh
2. Sign in, configure Settings (Ctrl+,)
3. Install recommended extensions (see README.md)

Key shortcuts:

- Ctrl+K / Cmd+K: AI chat
- Ctrl+L / Cmd+L: Inline editing

Setting Up: OpenAI Codex

```
npm install -g @openai/codex-cli
codex auth
```

Install VS Code/Cursor extension, configure API key. Conversation history:

`~/.codex/sessions/YYYY/MM/DD/*.jsonl`

Setting Up: Claude Code

1. Install from claude.com/product/claude-code
2. Authenticate with Anthropic API key
3. Install VS Code/Cursor extension

Conversation history: `~/.claude/history.jsonl`

Core Principles

Supervise AI like human RAs: clear instructions, review outputs, validate.

Theory-driven approach:

1. Model Setting
2. Solution Method
3. Pseudocode
4. Implementation

Oral exam: Explain goal → Ask if AI knows → Search until legitimate answer → Summarize.

Issue-Driven Development: Planning

1. Create docs in `docs/issue/` (summary, approach, outcomes)
2. Create GitHub issue, link docs
3. Create branch: `git switch -c feature/issue-description`

For economic models: Write model → Oral exam equilibrium → Oral exam pseudocode → Write issue/plan.

Plan → Show → WAIT for "proceed". "continue" ≠ "execute".

Implementation Phase

- Write functions in `src/` following pseudocode
- Named arguments: `f(value=x)` not `f(x)`
- Verb-based names: `ComputeMeanReward`, `FitNeuralNetwork`
- No timestamps: `file.py` not `file_20250101.py`
- No versions: `file.py` not `file_v2.py`
- Delete wrong code (use git for history)

Verification Phase

- Unit tests in `test/` or `scripts/test/`
- Test properties (monotonicity, bounds), not exact values
- `./run.sh test` or `pytest test/`
- List inconsistencies between pseudocode and implementation
- Grep to verify function names and APIs

Validation Phase

- Execution scripts in `scripts/` or `scripts/pipeline/`
- Reporting in `scripts/report/` or `scripts/analyze_data/`
- Use Quarto (.qmd) or R Markdown (.Rmd)
- Generate comparative statics, spot strange behavior
- Never ask AI to solve the issue; you determine the solution

Commit and Integration

- Explicitly ask: "commit these changes"
- One session = one commit = minimal and complete changes
- Push: `git push -u origin branch-name`
- Create PR linked to issue, request review, merge after approval

Debugging Workflow

1. Reproduce: Create validation script
2. Document: Summarize in `docs/issue/`
3. Root cause: List pseudocode vs implementation discrepancies
4. Fix: You determine solution; ask AI to implement your fix

Never ask AI to solve; you decide.

Model-Free Analysis

Dynamic reporting:

1. Make report loading data
2. Set up live server for review
3. Summarize dataset variables

Oral exam process → Implementation → Validation

Best Practices

- Package structure: `src/`, `test/`, `scripts/`
- Specify what can/cannot be changed
- Config files for parameters
- DRY, Config = Data, stable paths
- Test properties, not exact values

What AI is Good At

- On-demand data analysis (mathematically defined)
- Implementation from specification
- Repetitive tasks, documentation
- Code refactoring

What AI is NOT Good At

Economics concepts: equilibrium, endogenous/exogenous, observable/unobservable. Translate to CS/statistical concepts.

Solving bugs: AI may invent fake solutions. Ask for root cause; solution must be yours.

Critical Checklist

```
□ REPORT BEFORE FIXING/EDITING/EXECUTING
□ NO GIT without explicit "commit" request
□ NO TIMESTAMPS, NO VERSIONS in filenames
□ NO BACKWARD COMPATIBILITY
□ TEMPORARY → scripts/temporary/
□ NAMED ARGUMENTS
□ PLAN → WAIT → "proceed"
□ TEST → show results → "done"
□ USE run.sh wrapper
□ DELETE wrong code
```

Integration

- Version Control (version_control.md)
- Data Pipeline (data_pipeline.md)
- Reporting Pipeline (reporting_pipeline.md)
- Weekly Iterations (weekly_iterations.md)