LLM Coding Workflow Workshop

2-Hour Hands-On Walkthrough

Razvan Marinescu — UC Santa Cruz

February 10, 2026

Goal

By the end of this session, each participant should be able to:

- Use LLMs in **Cursor** to write new code and modify existing code.
- Use an **LLM CLI agent** (Codex or Claude CLI) for file/system operations.
- Build a small local data workflow: load CSV data, compute summaries, and generate plots.
- Produce a short document artifact (Markdown and optional LaTeX table) generated from local data.

Materials in this folder

- `data/students_scores.csv`, `data/model_runs.csv`
- `src/starter_analysis.py`
- `src/text_utils.py`
- `src/generate_report.py`
- `src/plot_metrics.py`
- `tests/test_text_utils.py`
- `requirements.txt`, `Makefile`

Session timeline (120 min)

Time	Activity
0–10 min	Environment checks + quick LLM sanity prompts in Cursor and CLI
10–35 min	Exercise A: write new code with LLM assistance (analysis function)
35–60 min	Exercise B: modify existing code + test-driven bug fixing
60–85 min	Exercise C: use CLI agent for file tasks and report generation
85–100 min	Exercise D: generate plots/charts from local CSV data
100–112 min	Exercise E: generate a stats table script and export LaTeX table

Exercise 0 — Setup check (10 min)

Install uv (pick your OS): *macOS (Homebrew)*

- `brew install uv`

Ubuntu / Linux

- `curl -LsSf https://astral.sh/uv/install.sh` sh|
- `source $HOME/.local/bin/env`

Check install

- `uv --version`

Then run in terminal:

- `cd llm-coding`
- `uv venv .venv`
- `source .venv/bin/activate`
- `uv pip install -r requirements.txt`
- `uv run python src/starter_analysis.py`
- `uv run python src/generate_report.py`
- `uv run python src/plot_metrics.py`

Quick prompts (Cursor and CLI):

Prompt

- "Read this folder and explain what each file is for."
- "Suggest 3 improvements to make this mini-project clearer for a beginner."

Exercise A — Write new code in Cursor (25 min)

Open `src/starter_analysis.py`. There are TODOs in:

- `compute_group_summary(...)`
- `find_top_improvers(...)`

Task:

1. Ask Cursor to implement both TODOs.
2. Ask Cursor to add docstrings and type hints if missing.
3. Run: `uv run python src/starter_analysis.py`

Prompt examples:

Prompt

- "Implement TODO functions in this file only. Keep output deterministic and concise."
- "Before editing, explain your plan in 3 bullets; after editing, show what changed."

Exercise B — Modify code + tests (25 min)

Open `src/text_utils.py`. It contains intentionally weak behavior.

Task:

1. Run tests: `uv run pytest -q` (or `make test`)
2. Use Cursor to fix failing tests in `tests/test_text_utils.py`
3. Ensure all tests pass.

Prompt examples:

Prompt

- "Read failing tests and patch only src/text_utils.py. Do not change tests unless absolutely necessary."
- "Keep functions simple and robust for edge cases (extra spaces, mixed case, punctuation)."

Exercise C — CLI agent workflow (25 min)

Use Codex/Claude CLI to do a small multi-file task:

1. Ask the CLI to inspect project structure.
2. Ask it to run `uv run python src/generate_report.py`.
3. Ask it to improve `reports/workshop_report.md` by adding one extra section: "Potential failure modes in this dataset and mitigation ideas."

Prompt template:

Prompt

1. You are helping with a local Python project.
2. Inspect files.
3. Run report generation.
4. Update the report with a short "failure modes" section.
5. Do not add new dependencies.
6. Show me exactly what you changed.

Exercise D — Plots/charts from local data (20 min)

Run: `uv run python src/plot_metrics.py`

This should generate:

- `reports/score_by_group.png`
- `reports/accuracy_vs_latency.png`

Move back to **Cursor**. In the agent chat, drag and drop `src/plot_metrics.py` into the prompt box before sending your prompt.

Prompt example:

Prompt

- "Using the attached file, add one additional plot: token usage distribution by model. Save it as `reports/tokens_by_model.png`. Keep existing plots unchanged, add axis labels/title, and show exactly what changed."

Once it generated it, ask the LLM to run the script and check the output.

Exercise E — Scripted stats table generation (12 min)

In **Cursor**, ask the LLM to write a new script (for example: `src/build_stats_table.py`) that reads local CSV data and writes a LaTeX table to `reports/stats_table.tex`.

Task:

1. Students must specify the table structure (rows and columns) in the prompt.
2. Table entries must report mean ± std for selected metrics.
3. Add significance stars for p-values less than `0.5`.
4. Save the final table as a standalone LaTeX snippet in `reports/stats_table.tex`.
5. Run the script and inspect the output file.

Prompt example:

Prompt

- "I attached `data/students_scores.csv`. Create `src/build_stats_table.py` that generates `reports/stats_table.tex` as well as an equivalent markdown table. Use rows={baseline, llm_assisted} and columns={assignment, midterm, final}. Each cell should show mean ± std for that group/metric. Also compute a two-group p-value per metric and append stars to the metric header when p-value < 0.5. Use booktabs formatting. After editing, run the script and show exactly what changed."

After the LLM finishes creating the script, ask it to run the script and inspect the output file.

Exercise F — Conference-paper LaTeX workflow (8 min)

Start from an online conference template (for example NeurIPS) and ask the LLM to set up a minimal paper draft locally.

Task:

1. Ask the LLM to create a new folder `paper/`.
2. Ask it to retrieve the NeurIPS style files from `https://neurips.cc/Conferences/2023/PaperInformation/StyleFiles` and save the `.tex` and `.sty` files into `paper/`.
3. Ask the LLM to compile LaTeX in `paper/`.
4. Then ask the LLM to include:
   - plots from `reports/` (including `tokens_by_model.png`),
   - table from `reports/stats_table.tex`.
5. Re-compile and verify the PDF builds.

Prompt examples (run as two separate prompts): Prompt 1 (setup + first compile):

Prompt

- "Create a new folder `paper/`. From `https://neurips.cc/Conferences/2023/PaperInformation/StyleFiles`, download/save the NeurIPS `.tex` template and `.sty` style file into `paper/`. Then compile with `pdflatex` and show me the files created."

After Prompt 1 finishes and `pdflatex` succeeds, run Prompt 2.

Prompt 2 (add figures + table):

Prompt

- "Now edit the paper in `paper/` to include: `../reports/score_by_group.png`, `../reports/accuracy_vs_latency.png`, `../reports/tokens_by_model.png`, and the table from `../reports/stats_table.tex`. Re-compile and show exactly what changed."

Best practices to emphasize

- Ask for a plan first, then ask for edits.
- Scope the edit: file(s), constraints, and expected output.
- Require verification steps (tests, script runs, lint checks).
- Prefer iterative prompts over one giant prompt.
- Keep humans in the loop for correctness, reproducibility, and security.

Common failure modes to discuss

- Silent hallucinations (invented functions, files, APIs).
- Over-broad code edits when prompt scope is vague.
- Incomplete environment assumptions (wrong Python/package versions).
- Data leakage/privacy risks when sharing sensitive content.
- Overconfidence in generated analysis without validation.

Wrap-up checklist

[ ] I can use Cursor to create and edit multi-file code changes.

[ ] I can use an LLM CLI agent to run commands and modify files safely.

[ ] I can generate and validate local data plots and summaries.

[ ] I understand at least 3 failure modes and mitigation strategies.

Optional extension for next session: multi-agent orchestration where one agent writes code, one runs tests, and one performs review.