Claude Code: AI-Powered CLI for Research

D-Lab AI Pulse Workshop

AI in the Browser: The Traditional Approach

Most people interact with AI through **web browsers**:

- ChatGPT, Claude.ai, Gemini, etc.
- Convenient for quick questions and conversations

But for coding and research, browsers have limitations:

- Must copy/paste code back and forth
- AI can't see your actual files or project structure
- Memory/Context limits are usually binding
- Can only work with one file at a time
- You have to manually run every command

A Better Way: Command Line Tools

CLI (Command Line Interface) – A text-based way to interact with your computer

CLI tools like **Claude Code** run directly in your terminal, *inside* your project.

This changes everything:

- Direct access to your file system
- Can read, create, and modify files
- Works within your actual project environment
- No copy-pasting code back and forth
- Maintains context across your session

Using Claude Code

Let's see it in practice!

It might feel awkward at first, since nowadays we don't use terminals often, but it has the same functionalities as the browser.

Claude Code can then:

- Read your files to understand the context
- Write or modify code directly
- Run commands and tests
- Iterate based on results

The CLI operates IN your project, not alongside it.

Browser vs CLI: Key Differences

Browser Chat	CLI Tool
Copy/paste code snippets	Directly edits your files
One file at a time	Reads entire codebases
You run the code	Can execute commands for you
Context resets each session	Maintains project memory
General knowledge	Sees YOUR actual code

The CLI operates IN your project, not alongside it.

When to Use Which?

Browser Chat	**CLI Tools**
Quick conceptual questions	Multi-file projects
Learning new concepts	Iterative development
Simple code snippets	Code that needs testing
Brainstorming ideas	Working with existing codebases
	Repetitive tasks across files

Installation: Prerequisites

1. **Install Node.js** (required for all platforms):
   - Download from `https://nodejs.org/` (LTS version recommended)
   - Or use a package manager: `brew install node` (macOS)

2. **Windows users: Install WSL** (Windows Subsystem for Linux):

```
wsl --install
```

   - Restart your computer after installation
   - Run Claude Code from within WSL, not PowerShell/CMD

Use Claude to Debug/Ask for Help!

Installation: Claude Code

Install Claude Code (macOS/Linux/WSL):

```
npm install -g @anthropic-ai/claude-code
```

First run:

```
claude
```

Authentication options:

- **Claude Pro/Team subscription** – Login with your browser (recommended)
- Anthropic API key – For programmatic access

Full docs: `https://docs.anthropic.com/en/docs/claude-code`

Permissions and Security

By default, Claude Code asks permission before:

- Editing or creating files
- Running shell commands
- Accessing external resources

This keeps you in control – you can review each action before it happens.

Customize allowed actions in `.claude/settings.json`:

```
{"permissions": {"allow": ["Bash(npm test)", "Bash(git status)"]}}
```

Skip all prompts (use with caution!):

```
claude --dangerously-skip-permissions
```

Only use this in trusted environments where you understand the risks.

Customization: CLAUDE.md

Create a `CLAUDE.md` file in your project root to give Claude persistent context:

```
# Project: My Research Analysis

## Overview
This project analyzes survey data from...

## Conventions
- Use numpy for numerical operations
- All functions should have docstrings
- Save outputs to the `results/` folder

## Important Files
- `data/raw_survey.csv` - Main dataset
- `src/analysis.py` - Core analysis functions
```

Claude reads this automatically when you start a session.

Model Selection

Claude Code supports three model tiers:

```
/model opus       # Opus 4.5 - most capable, complex reasoning
/model sonnet     # Sonnet 4 (default) - balanced, best for coding
/model haiku      # Haiku 3.5 - fastest, lightweight tasks
```

When to use which:

- **Opus**: Multi-file refactoring, complex debugging, large tasks
- **Sonnet**: Day-to-day coding, documentation, analysis
- **Haiku**: Quick questions, simple edits, cost-sensitive work

Useful Commands

```
/help              # See all available commands
/clear             # Clear conversation history
/compact           # Summarize conversation to clear context
/cost              # Check token usage and costs
/review            # Request a code review
```

Configuration:

```
claude config      # Manage API keys, defaults, preferences
```

Context Management

What is "context"? The information Claude has available to understand your project: files it has read, conversation history, and command outputs.

Claude Code has a 200K token context window. That's roughly:

- ~150,000 words of text (a 500-page book)
- ~10,000 lines of code across multiple files
- The entire DSGE.jl codebase we're using in Demo 2
- Your full PhD thesis codebase (19 files) + conversation history

Managing context effectively:

- Use `/compact` to summarize long conversations (frees up tokens)
- Reference specific files rather than "the whole project"
- CLAUDE.md helps focus on what matters

Subagents

Claude Code can spawn **subagents** – separate instances that work in parallel.

How to use them: Just ask in natural language:

```
"Spawn a subagent to verify that my refactored code
produces the same output as the original"
```

Use cases:

- Explore different parts of a codebase simultaneously
- Verify results independently (one agent writes, another checks)
- Handle complex multi-step tasks in parallel

Subagents provide "second opinions" and catch errors you might miss.

Pro Tip: Write Prompts in a Text File

Why?

1. Prevents accidental Enter sending incomplete prompts
2. Creates a log of your prompts for reproducibility
3. Allows careful editing before sending
4. Easy to reuse prompts across sessions

Workflow:

1. Write prompt in `prompt.txt`
2. Copy and paste into Claude Code interactive prompt
3. Keep the file as documentation

Demo 1: Linear Regression from Scratch

Goal: Create a complete OLS implementation without sklearn

Demo 1: Prompt

```
Create a LinearRegression class in Python that implements
OLS regression from scratch (without sklearn). Include:

1. fit(X, y) method using closed-form OLS solution
2. predict(X) method
3. Properties: coefficients, intercept, r_squared, standard_errors
4. summary() method with formatted table (like statsmodels)
5. plot_fit(X, y) method for visualization

Use only numpy, scipy.stats, and matplotlib. Include docstrings.
```

Demo 2: Documenting Julia Code (FRBNY DSGE)

Goal: Read and document the NY Fed's DSGE model

Demo 2: Prompt

```
Explore the DSGE.jl Julia codebase in
demo2_julia_documentation/DSGE.jl-main/

Give me an overview of:
- Its general purpose (what economic questions does it answer?)
- The main file structure
- Julia version and key dependencies from Project.toml
- Where to find example scripts
```

Demo 3: Refactoring PhD Codebase

Scenario: "2 years of thesis work. Code is a mess. Defense is next month."

19 Python files with hardcoded paths, duplicated functions, old_scripts/ folder...

Demo 3: Prompt

```
Explore this research codebase in the messy_codebase folder.
I'm a PhD student who built this over 2 years of thesis work
and need to clean it up before my defense.

Give me an overview of:
- The overall structure and what each file does
- The data pipeline (download -> process -> merge -> analyze)
- Main issues you see with the organization
```

Demo 3: Follow-up Prompt

```
Create a refactoring plan for this codebase. Goals:
- Make it work on any computer (no hardcoded paths)
- Single source of truth for data loading
- Clear project structure with config file
- Remove dead/duplicate code
- Proper entry point that doesn't use exec()
- Keep all functionality intact
```

Demo 3: Verification Prompt

```
Spawn a subagent to review the refactored code and verify:
- All original functionality is preserved
- Paths are properly relative/configurable
- No duplicate code remains
- The pipeline can run end-to-end
```

Demo 4: Data Consolidation (Bonus)

Scenario: Economic data from multiple sources (FRED, Michigan Survey, SPF)

Mixed formats (CSV, XLS, XLSX), different date conventions, varying frequencies

Demo 4: Analysis Prompt

```
Explore the data files in the fred/, michigan/, and spf/ folders.

For each source, analyze and report:
- What variables are available
- Date format and frequency (monthly, quarterly, etc.)
- Time range covered
- Any data quality issues (missing values, inconsistencies)

Give me a summary report before we proceed.
```

Demo 4: Merging Options Prompt

```
Based on your analysis, propose 2-3 options for merging
this data into a unified dataset. Consider:
- How to handle different frequencies
- Date alignment strategy
- Which variables to include
- How to handle the XLS/XLSX files

Recommend the best approach and explain why.
```

Demo 4: Verification Prompt

```
Spawn subagents to verify the merged dataset:
- One agent: check date alignment is correct
- One agent: verify no data was lost in the merge
- One agent: validate column names and types

Report back any issues found.
```

Best Practices

1. **Be specific** – "Add error handling to load_data()" $>$ "fix the code"
2. **Iterate** – Start simple, refine in follow-up prompts
3. **Verify** – Always review generated code before running
4. **Use context** – Reference specific files and functions
5. **Trust but validate** – Test outputs, check edge cases

Limitations to Keep in Mind

- **Not a replacement for understanding** – You need to verify the code
- **Can hallucinate** – May reference non-existent functions/libraries
- **Security considerations** – Review before running system commands
- **Novel research** – May not know cutting-edge methods
- **Liability** – It is your research - you are responsible for it
- **Interprets the commands you gave only** – Unless instructed correctly, it will "run wild"
- **Overwhelming** – It will produce months worth of work in minutes.

Resources

Documentation:

- `https://docs.anthropic.com/en/docs/claude-code`

Claude Subscription:

- `https://claude.ai/` (Pro/Team plans include Claude Code access)

Gemini (Free for Berkeley accounts!):

- `https://gemini.google.com/`

D-Lab:

- `https://dlab.berkeley.edu/`
- `https://github.com/dlab-berkeley/D-Lab-AI-Pulse--Claude-Code`

Questions? Let's discuss!