

# AI Coding Workshop

April 2026

~60 min

# Outline

---

## 01 Claude Code 101

Basic commands, context, MCP, skills, agents, and hooks.

## 02 Best practice & principles

Manage context, delegate goals, verify outcomes, and automate repetition.

## 03 Advanced usage

Project folders, experiment logs, reproducible artifacts, and evidence discipline.

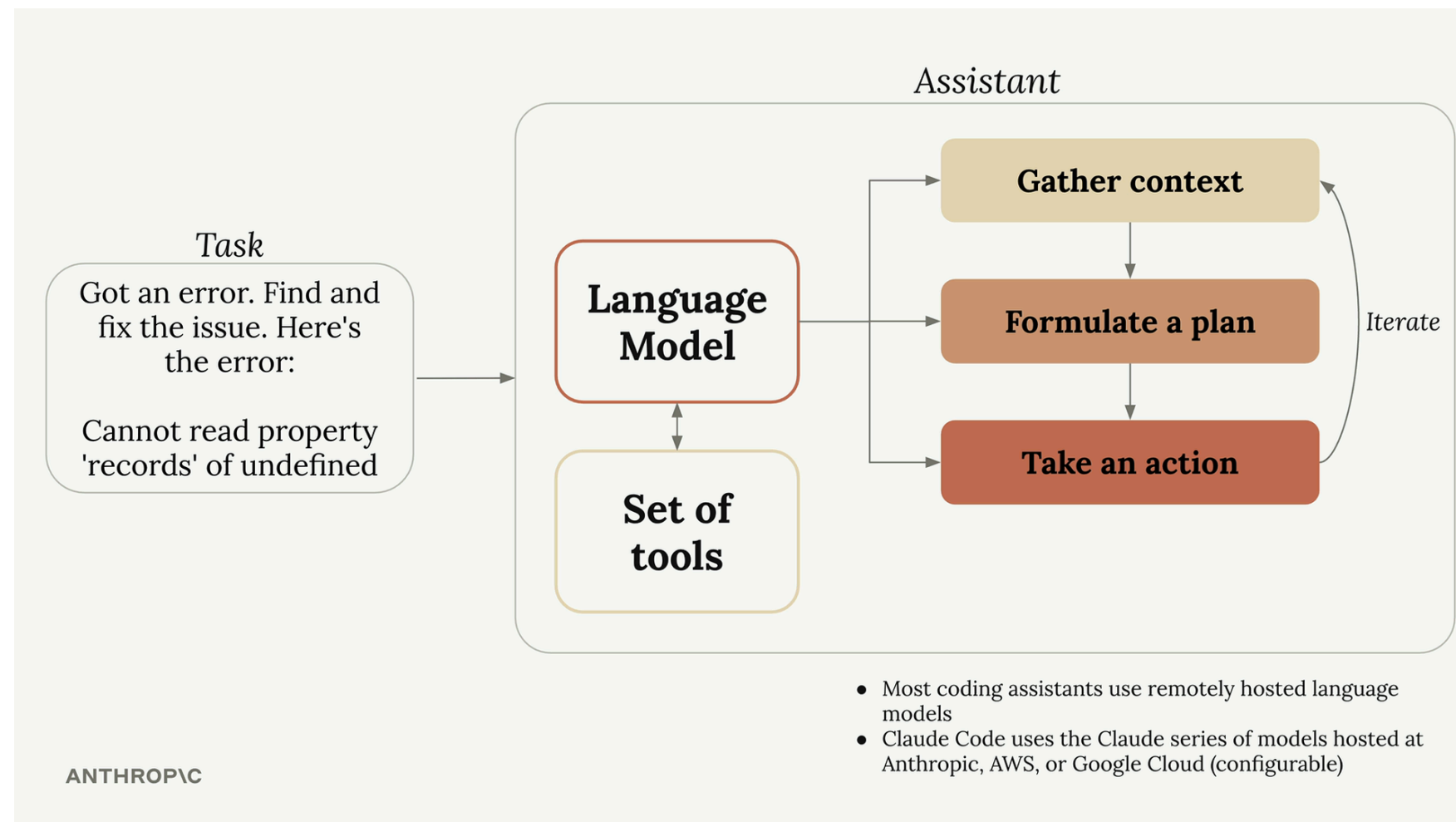
## 04 Lab shared config

A reusable Claude Code setup for Foreseer workflows.

## 05 Agent workspaces

Where to run agents, terminals, previews, and notifications.

# Coding agent architecture (simplified)



# What's new this generation

---

axis	Copilot-era	Claude Code

## What's new this generation

---

axis	Copilot-era	Claude Code
scope	current line / file	whole repo + filesystem

## What's new this generation

axis	Copilot-era	Claude Code
scope	current line / file	whole repo + filesystem
action	single completion	run shells, MCP, subagents, browsers

## What's new this generation

axis	Copilot-era	Claude Code
scope	current line / file	whole repo + filesystem
action	single completion	run shells, MCP, subagents, browsers
loop	tab to accept	plan → act → verify

## What's new this generation

axis	Copilot-era	Claude Code
scope	current line / file	whole repo + filesystem
action	single completion	run shells, MCP, subagents, browsers
loop	tab to accept	plan → act → verify
autonomy	you drive every keystroke	delegate a goal, walk away

§ 01

SECTION

# Claude Code 101

---

## /powerup: Quick start

> /powerup

- › ○ Talk to your codebase @ files, line refs
- Steer with modes shift+tab, plan, auto
- Undo anything /rewind, Esc-Esc
- Teach Claude your rules CLAUDE.md, /memory
- Extend with tools MCP, /mcp
- Multiply yourself subagents, /agents
- Dial the model /model, /effort

↑↓ to select · Enter to open · Esc to close

## **CLAUDE.md**: Teach Claude your rules

---

## **CLAUDE.md**: Teach Claude your rules

---

- A markdown file Claude reads at session start.

## **CLAUDE.md**: Teach Claude your rules

---

- A markdown file Claude reads at session start.
- Write down what you'd otherwise repeat in every prompt.

## `CLAUDE.md`: Teach Claude your rules

---

- A markdown file Claude reads at session start.
- Write down what you'd otherwise repeat in every prompt.
- New repo? Run `/init` to draft a starter `CLAUDE.md`.

## CLAUDE.md: What goes in

markdown

### ## Storage

- /nfs/turbo/si-qmei/\${USER} – datasets, checkpoints, envs.
- /scratch – hot I/O, copy back on job end.

### ## Compute

- Never run heavy jobs on login nodes.
- srun/salloc interactive · sbatch batch.

### ## Experiments

- experiments.md (index) + <date>\_<name>.md (detail).

Full file: [github.com/umich-foreseer/lab-claude-config/shared/CLAUDE.md](https://github.com/umich-foreseer/lab-claude-config/shared/CLAUDE.md)

## `/context`: Inspect the context

```
> /context
```

```
Context Usage  
Opus 4.7 (1M context)  
76.4k/1m tokens (8%)
```

```
Estimated usage by category  
System prompt: 9.1k tokens  
System tools: 29.5k tokens  
MCP tools: 20.9k tokens  
Custom agents: 1.0k tokens  
Memory files: 1.8k tokens  
Skills: 2.0k tokens  
Messages: 14.3k tokens  
Free space: 888.3k tokens
```

### Operate it:

- `/context` — see what's loaded.
- `/compact` — summarize and free space.
- `/rewind` — step back.
- `/clear` — start fresh.

## `/mcp`: **Broader context**

---

Model Context Protocol: wire a service in once, then every agent can use it.

## `/mcp`: Broader context

---

Model Context Protocol: wire a service in once, then every agent can use it.

- `wandb` — query runs, metrics, configs.

## `/mcp`: Broader context

---

Model Context Protocol: wire a service in once, then every agent can use it.

- `wandb` — query runs, metrics, configs.
- `playwright` — drive a real browser.

## `/mcp`: Broader context

---

Model Context Protocol: wire a service in once, then every agent can use it.

- `wandb` — query runs, metrics, configs.
- `playwright` — drive a real browser.
- `slack` — search discussions, summarize threads, draft updates.

## `/mcp`: Broader context

---

Model Context Protocol: wire a service in once, then every agent can use it.

- `wandb` — query runs, metrics, configs.
- `playwright` — drive a real browser.
- `slack` — search discussions, summarize threads, draft updates.
- `github`, `filesystem`, `postgres` — common servers.

## `/mcp`: Broader context

---

Model Context Protocol: wire a service in once, then every agent can use it.

- `wandb` — query runs, metrics, configs.
- `playwright` — drive a real browser.
- `slack` — search discussions, summarize threads, draft updates.
- `github`, `filesystem`, `postgres` — common servers.

Operate: `/mcp` lists connected servers and tools.

## `/skills`: Recipes in natural language

---

A skill is a `SKILL.md`: name, description, and steps in plain English.

## `/skills`: Recipes in natural language

---

A skill is a `SKILL.md`: name, description, and steps in plain English.

- `/submit-experiment` — names the run, writes sbatch, submits, records the job.

## `/skills`: Recipes in natural language

---

A skill is a `SKILL.md`: name, description, and steps in plain English.

- `/submit-experiment` — names the run, writes sbatch, submits, records the job.
- `/harvest` — walks recent jobs, pulls results into `experiments.md`.

## `/skills`: Recipes in natural language

---

A skill is a `SKILL.md`: name, description, and steps in plain English.

- `/submit-experiment` — names the run, writes sbatch, submits, records the job.
- `/harvest` — walks recent jobs, pulls results into `experiments.md`.
- `/slurm-debug` — logs, exit code, node state, diagnosis.

## `/skills`: Recipes in natural language

---

A skill is a `SKILL.md`: name, description, and steps in plain English.

- `/submit-experiment` — names the run, writes sbatch, submits, records the job.
- `/harvest` — walks recent jobs, pulls results into `experiments.md`.
- `/slurm-debug` — logs, exit code, node state, diagnosis.

*“Based on what we just did, create a skill for it.”* — Claude drafts the `SKILL.md`.

## `/agents`: Skills with their own context

---

Like a skill, but it runs in its own session and returns only the answer.

## `/agents`: Skills with their own context

---

Like a skill, but it runs in its own session and returns only the answer.

- Use for long logs, result sweeps, repo search, paper surveys.

## `/agents`: Skills with their own context

---

Like a skill, but it runs in its own session and returns only the answer.

- Use for long logs, result sweeps, repo search, paper surveys.
- Run subagents in parallel, then merge findings.

## `/agents`: Skills with their own context

---

Like a skill, but it runs in its own session and returns only the answer.

- Use for long logs, result sweeps, repo search, paper surveys.
- Run subagents in parallel, then merge findings.
- Built-ins: `Explore`, `general-purpose`, `Plan`.

## `/agents`: Skills with their own context

---

Like a skill, but it runs in its own session and returns only the answer.

- Use for long logs, result sweeps, repo search, paper surveys.
- Run subagents in parallel, then merge findings.
- Built-ins: `Explore`, `general-purpose`, `Plan`.

Operate: `/agents` lists and configures agents.

## `/hooks`: **Code that runs on events**

---

A shell command wired to an event. It fires deterministically.

## `/hooks`: Code that runs on events

---

A shell command wired to an event. It fires deterministically.

- `SessionStart` — load node, env, today's TODOs.

## `/hooks`: Code that runs on events

---

A shell command wired to an event. It fires deterministically.

- `SessionStart` — load node, env, today's TODOs.
- `PreToolUse` — block unsafe commands.

## `/hooks`: Code that runs on events

---

A shell command wired to an event. It fires deterministically.

- `SessionStart` — load node, env, today's TODOs.
- `PreToolUse` — block unsafe commands.
- `PostToolUse` — format, log, notify.

## `/hooks`: Code that runs on events

---

A shell command wired to an event. It fires deterministically.

- `SessionStart` — load node, env, today's TODOs.
- `PreToolUse` — block unsafe commands.
- `PostToolUse` — format, log, notify.
- `UserPromptSubmit` — inject branch or recent failures.

## `/hooks`: Code that runs on events

---

A shell command wired to an event. It fires deterministically.

- `SessionStart` — load node, env, today's TODOs.
- `PreToolUse` — block unsafe commands.
- `PostToolUse` — format, log, notify.
- `UserPromptSubmit` — inject branch or recent failures.

Use hooks for guardrails you need every time.

§ 02

SECTION

# Best practice & principles

---

## Principle #1

---

Context is your  
most valuable resource.

Everything else is a corollary.

## Use subagents for investigation

---

**When** — read a lot to answer a little.

**How:**

## Use subagents for investigation

---

**When** — read a lot to answer a little.

**How:**

- Just ask: *"Find where reward is normalized."*

## Use subagents for investigation

---

**When** — read a lot to answer a little.

**How:**

- Just ask: *"Find where reward is normalized."*
- Or name it: *"Use `Explore` to list configs that set `lr`."*

## Use subagents for investigation

---

**When** — read a lot to answer a little.

**How:**

- Just ask: *"Find where reward is normalized."*
- Or name it: *"Use `Explore` to list configs that set `lr`."*
- Parallelize: one agent per run directory.

## Use subagents for investigation

---

**When** — read a lot to answer a little.

**How:**

- Just ask: *"Find where reward is normalized."*
- Or name it: *"Use `Explore` to list configs that set `lr`."*
- Parallelize: one agent per run directory.

**Why** — it burns its own context; you get the summary.

*Ref: [code.claude.com](https://code.claude.com) — use-subagents-for-investigation*

# Manage context aggressively

---

## Manage context aggressively

---

- One session per task; `/clear` (alias `/new`) between unrelated work.

## Manage context aggressively

---

- One session per task; `/clear` (alias `/new`) between unrelated work.
- `/compact` around 30% remaining — earlier if the conversation is dense.

## Manage context aggressively

---

- One session per task; `/clear` (alias `/new`) between unrelated work.
- `/compact` around 30% remaining — earlier if the conversation is dense.
- Trim oversized `CLAUDE.md` sections; disable MCP servers you aren't using.

## Manage context aggressively

---

- One session per task; `/clear` (alias `/new`) between unrelated work.
- `/compact` around 30% remaining — earlier if the conversation is dense.
- Trim oversized `CLAUDE.md` sections; disable MCP servers you aren't using.
- Prefer `@file:lines` over pasting whole files.

## Manage context aggressively

---

- One session per task; `/clear` (alias `/new`) between unrelated work.
- `/compact` around 30% remaining — earlier if the conversation is dense.
- Trim oversized `CLAUDE.md` sections; disable MCP servers you aren't using.
- Prefer `@file:lines` over pasting whole files.

*Ref: [code.claude.com](https://code.claude.com) — [manage-context-aggressively](#)*

## Principle #2

---

Delegate,  
don't micromanage.

Set the goal. Define how you'll verify. Step back.

## Plan → Auto

---

- Toggle **plan** and **auto** with `shift+tab`.
- Read the plan, redirect once, then let it run.
- For complex changes, review the diff instead of each tool call.

Spend time in the plan and the review, not mid-execution steering.

## Goal + verification loop

---

Hand the agent something it can check itself.

## Goal + verification loop

---

Hand the agent something it can check itself.

- State **done** concretely: *"dev acc  $\geq 0.78$ ."*

## Goal + verification loop

---

Hand the agent something it can check itself.

- State **done** concretely: *"dev acc  $\geq$  0.78."*
- Wire the loop: eval, figure, assertion.

## Goal + verification loop

---

Hand the agent something it can check itself.

- State **done** concretely: *"dev acc  $\geq$  0.78."*
- Wire the loop: eval, figure, assertion.
- Ask for the verification command in the plan.

## Goal + verification loop

---

Hand the agent something it can check itself.

- State **done** concretely: *"dev acc  $\geq$  0.78."*
- Wire the loop: eval, figure, assertion.
- Ask for the verification command in the plan.

When the agent can tell whether it's done, you stop being the loop.

## Let Claude interview you

---

Stuck staring at an empty prompt? Flip the direction.

## Let Claude interview you

---

Stuck staring at an empty prompt? Flip the direction.

- *“Before you start, ask me 5 questions to clarify the goal and constraints.”*

## Let Claude interview you

---

Stuck staring at an empty prompt? Flip the direction.

- *“Before you start, ask me 5 questions to clarify the goal and constraints.”*
- Answer them; the conversation becomes the spec.

## Let Claude interview you

---

Stuck staring at an empty prompt? Flip the direction.

- *“Before you start, ask me 5 questions to clarify the goal and constraints.”*
- Answer them; the conversation becomes the spec.
- Best for fuzzy tasks: refactors, new features, speedups.

## Let Claude interview you

---

Stuck staring at an empty prompt? Flip the direction.

- *"Before you start, ask me 5 questions to clarify the goal and constraints."*
- Answer them; the conversation becomes the spec.
- Best for fuzzy tasks: refactors, new features, speedups.
- Pro tip: share this slide with CC and let it create a skill.

## Let Claude interview you

---

Stuck staring at an empty prompt? Flip the direction.

- *“Before you start, ask me 5 questions to clarify the goal and constraints.”*
- Answer them; the conversation becomes the spec.
- Best for fuzzy tasks: refactors, new features, speedups.
- Pro tip: share this slide with CC and let it create a skill.

*Ref: [code.claude.com](https://code.claude.com) — [let-claude-interview-you](#)*

## `/review`: **An adversarial second pass**

---

Generation and critique are different jobs.

## `/review`: An adversarial second pass

---

Generation and critique are different jobs.

- `/review` — a single PR or diff: bugs, missing tests, style.

## `/review`: An adversarial second pass

---

Generation and critique are different jobs.

- `/review` — a single PR or diff: bugs, missing tests, style.
- `/ultrareview` — branch-level spec, architecture, edge cases.

## `/review`: An adversarial second pass

---

Generation and critique are different jobs.

- `/review` — a single PR or diff: bugs, missing tests, style.
- `/ultrareview` — branch-level spec, architecture, edge cases.
- `/security-review` — auth, file I/O, prod configs.

## `/review`: An adversarial second pass

---

Generation and critique are different jobs.

- `/review` — a single PR or diff: bugs, missing tests, style.
- `/ultrareview` — branch-level spec, architecture, edge cases.
- `/security-review` — auth, file I/O, prod configs.
- Mix reviewers when stakes are high.

## `/review`: An adversarial second pass

---

Generation and critique are different jobs.

- `/review` — a single PR or diff: bugs, missing tests, style.
- `/ultrareview` — branch-level spec, architecture, edge cases.
- `/security-review` — auth, file I/O, prod configs.
- Mix reviewers when stakes are high.

Codex inside Claude Code: [github.com/openai/codex-plugin-cc](https://github.com/openai/codex-plugin-cc)

## Principle #3

---

| Automate what repeats.

If you typed it twice today, it should be a skill.

## The 2x rule

---

Doing something twice in a day is a signal. Stop and ask:

## The 2x rule

---

Doing something twice in a day is a signal. Stop and ask:

- *“Can I create a skill for this?”*

## The 2x rule

---

Doing something twice in a day is a signal. Stop and ask:

- *"Can I create a skill for this?"*
- *"Would a hook catch this automatically?"*

## The 2x rule

---

Doing something twice in a day is a signal. Stop and ask:

- *"Can I create a skill for this?"*
- *"Would a hook catch this automatically?"*
- *"Should this go in `CLAUDE.md` so I never type it again?"*

## Let Claude write the skill

---

You don't author `SKILL.md` from scratch.

## Let Claude write the skill

---

You don't author `SKILL.md` from scratch.

- At the end of the session: *"Based on what we just did, create a skill for it."*

## Let Claude write the skill

---

You don't author `SKILL.md` from scratch.

- At the end of the session: *"Based on what we just did, create a skill for it."*
- For a hook: *"Turn this guardrail into a `PreToolUse` hook."*

## Let Claude write the skill

---

You don't author `SKILL.md` from scratch.

- At the end of the session: *"Based on what we just did, create a skill for it."*
- For a hook: *"Turn this guardrail into a `PreToolUse` hook."*
- Claude drafts the file, names it, and shows the diff.

## Let Claude write the skill

---

You don't author `SKILL.md` from scratch.

- At the end of the session: *"Based on what we just did, create a skill for it."*
- For a hook: *"Turn this guardrail into a `PreToolUse` hook."*
- Claude drafts the file, names it, and shows the diff.

Capture the version you just verified.

§ 03

SECTION

# Advanced usage

---

## Put everything in one folder

tree

```
project/
├── src/           # code
├── experiments/  # runs + logs
├── paper/        # LaTeX, git-backed
├── results/      # tables + figures, auto-generated
└── refs/         # Zotero export, BibTeX
```

One repo, one mental model. Code, results, and prose stay reachable.

## Code controls the paper

---

Every number, table, and figure is generated by script.

## Code controls the paper

---

Every number, table, and figure is generated by script.

- `scripts/artifacts_generation/*` reads `experiments/`, writes `results/`.

## Code controls the paper

---

Every number, table, and figure is generated by script.

- `scripts/artifacts_generation/*` reads `experiments/`, writes `results/`.
- `paper/main.tex` only inputs generated artifacts.

## Code controls the paper

---

Every number, table, and figure is generated by script.

- `scripts/artifacts_generation/*` reads `experiments/`, writes `results/`.
- `paper/main.tex` only inputs generated artifacts.
- Rule: fix the script and rerun; don't patch numbers in `.tex`.

## Code controls the paper

---

Every number, table, and figure is generated by script.

- `scripts/artifacts_generation/*` reads `experiments/`, writes `results/`.
- `paper/main.tex` only inputs generated artifacts.
- Rule: fix the script and rerun; don't patch numbers in `.tex`.

**Test:** delete `results/`, rerun the build, get the same paper.

## Two-level docs: Index + detail

---

**One self-contained index. One detail file per entry.** Both readable on their own.

- **Index** (`experiments.md`) — one paragraph per run, grouped by phase.
- **Detail** (`experiments/<date>_<name>.md`) — reproduce or interpret one run.

Same shape: `papers.md` → `papers/<id>.md`

## The index entry

Each entry: name, trial, headline observation, detail link. <100 words.

markdown

### ## SFT (v1)

- **\*\*sft-qwen3-1.7b\*\*** (1.7B full-FT): Scale-up test. 65.0% acc at epoch 2 – top v1 result in this comparison.  
[detail]([experiments/2026-03-21\\_sft-qwen3-1.7b.md](experiments/2026-03-21_sft-qwen3-1.7b.md))
- ...

## The detail page

---

**Every detail page carries:**

## The detail page

---

**Every detail page carries:**

- git hash of the code that ran.

## The detail page

---

### Every detail page carries:

- git hash of the code that ran.
- exact command or sbatch path.

## The detail page

---

### Every detail page carries:

- git hash of the code that ran.
- exact command or sbatch path.
- dataset version + path.

## The detail page

---

### Every detail page carries:

- git hash of the code that ran.
- exact command or sbatch path.
- dataset version + path.
- log file + wandb run ID.

## The detail page

---

### Every detail page carries:

- git hash of the code that ran.
- exact command or sbatch path.
- dataset version + path.
- log file + wandb run ID.
- config (path + hash).

## The detail page

---

### Every detail page carries:

- git hash of the code that ran.
- exact command or sbatch path.
- dataset version + path.
- log file + wandb run ID.
- config (path + hash).
- observed numbers + plots.

## The detail page

---

### Every detail page carries:

- git hash of the code that ran.
- exact command or sbatch path.
- dataset version + path.
- log file + wandb run ID.
- config (path + hash).
- observed numbers + plots.
- **Hypothesis:** for anything not directly observed.

## The detail page

---

### Every detail page carries:

- git hash of the code that ran.
- exact command or sbatch path.
- dataset version + path.
- log file + wandb run ID.
- config (path + hash).
- observed numbers + plots.
- **Hypothesis:** for anything not directly observed.

**Test:** can a new lab member reproduce it from the file alone?

## No hasty conclusions

---

Store facts by default. Conclusions need evidence.

**Discipline:** separate facts from interpretation.

markdown

- Accuracy rose 0.52 → 0.61 from step 500 → 1400 [observed].
- Hypothesis [needs approval before saving]:  
4B may not amplify the diversity benefit seen at 1.7B.

Unverified hypotheses go into files only with explicit approval.

## § 04

### SECTION

# Lab shared config

---

## `lab-claude-config`: Don't bootstrap from scratch

---

Once Claude Code is installed, point it at the repo and ask it to deploy.

*"Use this repo to set up my lab Claude Code config on Great Lakes."*

[github.com/umich-foreseer/lab-claude-config](https://github.com/umich-foreseer/lab-claude-config)

## `lab-claude-config`: **Skills**

---

A list of useful skills on HPC.

## `lab-claude-config`: Skills

---

A list of useful skills on HPC.

- `/onboard` — detects username, cluster, Slurm accounts; runs setup.

## `lab-claude-config`: Skills

---

A list of useful skills on HPC.

- `/onboard` — detects username, cluster, Slurm accounts; runs setup.
- `/submit-experiment` — drafts sbatch, creates docs, submits, records the job.

## `lab-claude-config`: Skills

---

A list of useful skills on HPC.

- `/onboard` — detects username, cluster, Slurm accounts; runs setup.
- `/submit-experiment` — drafts sbatch, creates docs, submits, records the job.
- `/harvest` — pulls metrics + paths back into the index.

## `lab-claude-config`: Skills

---

A list of useful skills on HPC.

- `/onboard` — detects username, cluster, Slurm accounts; runs setup.
- `/submit-experiment` — drafts sbatch, creates docs, submits, records the job.
- `/harvest` — pulls metrics + paths back into the index.
- `/slurm-status` — current queue, your runs, partition pressure.

## lab-claude-config: Skills

---

A list of useful skills on HPC.

- `/onboard` — detects username, cluster, Slurm accounts; runs setup.
- `/submit-experiment` — drafts sbatch, creates docs, submits, records the job.
- `/harvest` — pulls metrics + paths back into the index.
- `/slurm-status` — current queue, your runs, partition pressure.
- `/connect` — cross-cluster SSH with the right keys + paths.

## lab-`claude-config`: Agents & hooks

---

**Agents** — fresh-context investigators.

- `@slurm-queue` — survey the queue, find a free slot, suggest a partition.
- `@slurm-resource` — pick GPU / memory / walltime.
- `@slurm-storage` — choose Turbo, Scratch, or Data Den.

**Hook** — runs deterministically on every event.

- `node-context.sh` on `PreToolUse(Bash)` — detects login vs compute node and injects an advisory before shell calls.

§ 05

SECTION

# Agent workspaces

---

## Where do you run Claude Code?

---

An **agent workspace** is where you keep agents, code, logs, previews, and notifications visible.

Options:

- **VS Code's terminal** — one pane, close to the editor.
- **A regular terminal** — Ghostty, iTerm, Alacritty; bring your own panes and tabs.
- **Agent-native terminal** — packages panes, tabs, worktrees, and notifications.

My favorite right now: **cmux**.

## `cmux`: The terminal built for multitasking

---

A native macOS terminal built around agent workflows.

## `cmux`: The terminal built for multitasking

---

A native macOS terminal built around agent workflows.

- **Vertical tabs** show branch, directory, ports, notifications.

## `cmux`: The terminal built for multitasking

---

A native macOS terminal built around agent workflows.

- **Vertical tabs** show branch, directory, ports, notifications.
- **Notification rings** light up when an agent needs attention.

## `cmux`: The terminal built for multitasking

---

A native macOS terminal built around agent workflows.

- **Vertical tabs** show branch, directory, ports, notifications.
- **Notification rings** light up when an agent needs attention.
- **Split panes** pair an agent with logs, notebooks, or browser.

## `cmux`: The terminal built for multitasking

---

A native macOS terminal built around agent workflows.

- **Vertical tabs** show branch, directory, ports, notifications.
- **Notification rings** light up when an agent needs attention.
- **Split panes** pair an agent with logs, notebooks, or browser.
- **Socket API** lets scripts spawn tabs and panes.

## `cmux`: The terminal built for multitasking

---

A native macOS terminal built around agent workflows.

- **Vertical tabs** show branch, directory, ports, notifications.
- **Notification rings** light up when an agent needs attention.
- **Split panes** pair an agent with logs, notebooks, or browser.
- **Socket API** lets scripts spawn tabs and panes.
- **Free, open-source.** Works with Claude Code, Codex, Aider, Cline.

## **cmux**: The terminal built for multitasking

---

A native macOS terminal built around agent workflows.

- **Vertical tabs** show branch, directory, ports, notifications.
- **Notification rings** light up when an agent needs attention.
- **Split panes** pair an agent with logs, notebooks, or browser.
- **Socket API** lets scripts spawn tabs and panes.
- **Free, open-source.** Works with Claude Code, Codex, Aider, Cline.

[cmux.com](https://cmux.com)

## Build Your Own

# Thanks!

---

→ [xingjianz.com](https://xingjianz.com) → [jimmyzxj@umich.edu](mailto:jimmyzxj@umich.edu) → [umich-foreseer/lab-claude-config](https://umich-foreseer/lab-claude-config)