M03 — AI-Assisted Coding Lab

Lab — Generative AI Boot Camp for Developers — Exported 2026-09-02

Time budget: 90 minutes
Section: Day 1 PM — The Signature Lab

2026-09-02

Learning Objectives

After completing this lab, you will: 1. Build a complete REST API using at least two different GenAI developer tools 2. Compare tools across key dimensions and identify which fits your workflow 3. Apply prompt engineering (M02) and the AI quality checklist to AI-generated code

---

2026-09-02

Tool Introductions: Your AI Coding Toolkit

This module is about putting tools into practice. Here's a detailed look at what's available on your sandbox VM and when to use each.

OpenCode CLI — The Provider-Agnostic Workhorse

What it is: A terminal-based AI coding assistant that works with any LLM provider. You've used it in M01 and M02. It's our primary lab tool.

Two modes: - **One-shot:** `opencode run "your task"` — generates code, shows diff, done. Best for focused, single-file tasks. - **Interactive:** `opencode` — opens a persistent session. Best for multi-turn conversations, exploring a codebase, iterating on a feature.

Why we use it: Provider flexibility (switch LLMs without switching tools). Scriptable (works in CI/CD). Fast. No GUI dependency.

VS Code Chat Agent — AI Inside Your IDE

What it is: A conversation panel inside VS Code (Ctrl+Shift+I). The AI sees your open files, selections, and workspace structure.

Why use it: Lowest friction — you never leave your editor. Context-aware — the AI knows about your project without you explaining it. Great for "explain this code" and "refactor this function" tasks.

2026-09-02

VS Code Inline Completions — Your Always-On Pair Programmer

What it is: As you type, the AI suggests completions — sometimes entire lines or blocks. Triggered automatically. Accept with Tab.

Why use it: Zero-effort assistance. Best for boilerplate, repetitive patterns, and completing known APIs. It's like having a very fast junior dev watching over your shoulder.

> **Try this during the lab:** In your `models.py` file, start typing a Pydantic model. Watch the inline suggestions appear. Accept what's useful, ignore what's not.

Claude Code (Optional — Anthropic API Key Required)

What it is: Anthropic's CLI agent with deep codebase understanding. Can read your entire project, run shell commands, and iterate autonomously.

Why use it: Best-in-class for multi-file refactoring and complex debugging. Understands architectural context better than most tools.

Codex CLI (Optional — OpenAI API Key Required)

What it is: OpenAI's terminal coding agent with sandbox execution.

Why use it: Fastest iteration speed. Strong at web development and rapid prototyping. Good at explaining and documenting code.

2026-09-02

Concept Review: The AI-Assisted Development Workflow

The Three Modes of AI Development

Mode	How It Works	Best For	Tools
Inline	Suggestions appear as you type	Staying in flow, boilerplate, completing known patterns	VS Code Copilot
Chat	Conversation-based, multi-turn	Complex tasks, explanations, refactoring, generating new code	OpenCode interactive, VS Code Chat
CLI One-Shot	Single command → result	Scriptable tasks, CI/CD, focused generation	`opencode run`, `claude`, `codex exec`

The AI-Assisted Development Workflow

```
1. PLAN     → Think through the task before touching AI
2. PROMPT   → Craft a specific, structured prompt (M02 skills)
3. GENERATE → Let AI produce the first draft
4. REVIEW   → Read every line — treat it like a code review
5. REFINE   → Iterate with AI until quality meets your standard
6. TEST     → Run tests, add edge cases AI missed
7. COMMIT   → You're the author — own the code
```

2026-05-02

The AI Code Quality Checklist

Before committing any AI-generated code, verify every item:

- [ ] **I understand every line** — If you don't, ask the AI to explain it
- [ ] **Tests pass** — Including edge cases I added (AI often misses domain-specific cases)
- [ ] **No hardcoded secrets or API keys** — AI sometimes generates placeholder keys
- [ ] **Error handling is appropriate** — Not just try/except pass
- [ ] **Naming follows team conventions** — AI uses generic names unless you specify
- [ ] **No unnecessary dependencies** — AI loves to import libraries you don't need
- [ ] **I could explain this code to a colleague** — If not, iterate more

Inline vs. Chat vs. CLI: The Decision Framework

Task	Best Mode	Why
Completing a line of code	Inline	Fastest, least disruptive
Generating a new function	Chat	Needs context + instruction
Refactoring across files	Chat (CLI)	Needs multi-file awareness
Understanding complex code	Chat	Needs explanation + Q&A
Writing tests for existing code	Chat	Needs code context + spec
Boilerplate / repetitive code	Inline	Pattern is clear from context
CI/CD integration	CLI One-Shot	Scriptable, no human in loop

2026-09-02

Prerequisites

- LLM API access configured and verified (M01)
- OpenCode CLI and VS Code working
- Optional: Claude Code or Codex API keys if you want to test those

---

2026-09-02

Exercise: Build a REST API Endpoint

You'll build a complete REST API using at least two different AI tools. Choose one scenario or bring your own idea.

Scenarios

A — Task Management API: Endpoints to create and list tasks. Each task: title, description, priority (low/medium/high), status (todo/in_progress/done), due_date.

B — Book Review API: Endpoints to submit and retrieve book reviews. Each review: book_title, author, rating (1–5), review_text, reviewer_name.

C — Your Own Idea: Pick a simple CRUD endpoint relevant to your work.

Getting Started

A FastAPI scaffold is available at `workshop/m03/scaffold.md`. Copy it to get started quickly, or build from scratch — both are valid approaches.

2026-09-02

Step 1: Plan (5 min)

Before touching any AI tool, sketch your plan on paper or in a scratch file:

- What endpoints? (HTTP method + path)
- What does the request body look like?
- What does the response look like?
- What edge cases should you handle?
- What's your directory structure?

> **Why plan first:** If you can't describe what you want, the AI can't generate it. Planning forces clarity.

2026-09-02

Step 2: Generate with Tool #1 — OpenCode (25 min)

Use OpenCode to build the core of your API:

```
cd ~/workshop/m03-api

# Generate models

opencode run "Create Pydantic models for a task management API. Tasks have:
title (str, required), description (str, optional), priority (enum: low/medium/high),
status (enum: todo/in_progress/done), due_date (optional datetime). Use Python 3.12+
type hints. Output as models.py."

# Generate service layer
opencode run "Read models.py. Create service.py with an in-memory task store
(dict) and functions: create_task, list_tasks, get_task. Include input validation
and proper error handling. Use type hints throughout."

# Generate API endpoints
opencode run "Read models.py and service.py. Add FastAPI endpoints to app.py:
POST /tasks (create), GET /tasks (list, with optional status filter),
GET /tasks/{task_id}. Return proper HTTP status codes and error responses."
```

> **Tip:** Start with OpenCode one-shot commands for each file, then switch to interactive mode (`opencode`) for refinements.

Also try — VS Code Inline Completions: Open `models.py` in VS Code. Start typing `class Task(BaseModel):` and watch the inline suggestions appear. Use Tab to accept.

2026-09-02

Step 3: Review & Refine (10 min)

Review the generated code against the quality checklist:

1. Run it: `uvicorn app:app --reload` — does it start?
2. Test it: `curl -X POST http://localhost:8000/tasks -H "Content-Type: application/json" -d '{"title":"Test"}'`
3. Check edge cases from Step 1 — are they handled?
4. Any hardcoded values?

Use **VS Code Chat** for refinements — select the code you want to improve, open Chat (Ctrl+Shift+I), and ask: "Add input validation for the priority field. Only allow 'low', 'medium', or 'high'. Return a 422 with a descriptive error message."

2026-09-02

Step 4: Generate Documentation with Tool #2 (15 min)

Switch tools. If you used OpenCode for Steps 2–3, use **VS Code Chat** or **Claude Code** for documentation. The point is to compare.

VS Code Chat: Select your `app.py`, open Chat, ask: "Generate OpenAPI 3.0 documentation for these endpoints. Include request/response examples."

Claude Code (optional):

```
claude -p "Read app.py and models.py. Generate comprehensive API documentation in
Markdown format. Include: endpoint descriptions, request/response schemas, example
curl commands, and error response formats."
```

Codex CLI (optional):

```
codex exec "Read app.py. Generate API documentation with request/response examples."
```

2026-09-02

Step 5: Add Tests (15 min)

Generate tests using either tool. Try a tool you haven't used yet:

```
opencode run "Read app.py, models.py, and service.py. Write pytest tests for the
task creation endpoint. Cover: happy path, missing required fields, invalid
priority value, empty request body, duplicate task titles. Use httpx for API testing."
```

Run the tests and evaluate: how many passed on the first run? Which failed and why?

2026-09-02

Step 6: Tool Comparison (10 min)

Complete the comparison worksheet at `workshop/m03/tool-comparison-worksheet.md`. Rate each tool you used on all dimensions. Be honest — there's no "right" answer.

2026-09-02

Step 7: Group Debrief (10 min — Facilitator-Led)

Share your findings. Key questions for discussion: - Did anyone prefer CLI over IDE? Why? - Which tool produced the best code on the first attempt? - What did AI do better than you expected? Worse?

2026-09-02

Deliverables

- [ ] Working REST API (`app.py`, `models.py`, `service.py`)
- [ ] `requirements.txt`
- [ ] API documentation (generated by Tool #2)
- [ ] Test file with passing tests
- [ ] Completed Tool Comparison Worksheet

2026-09-02

Troubleshooting

Symptom	Likely Cause	Resolution
OpenCode generates incomplete code	Prompt too vague	Add file structure, endpoint specs, constraints
Generated code doesn't run	Missing imports or deps	Run `pip install` for missing packages; ask AI to include all imports
Wrong framework or patterns	Missing context in prompt	Specify framework, version, and coding conventions
AI adds a `main` block you don't want	No constraint against it	Add "Do not include a **main** block" to your prompt
Claude/Codex not found	Not installed or no API key	Use OpenCode or VS Code — all tools work for this lab
Rate limit exceeded	Too many rapid requests	Wait 30 seconds; reduce request frequency
AI-generated tests fail	Hallucinated assertions or APIs	Review each test; ask AI to fix specific failures
VS Code Inline suggestions not appearing	Copilot not signed in or extension disabled	Check VS Code extensions panel; sign in if needed
Two tools give very different results	Different models, training data	Document the differences — this is a learning opportunity

2026-09-02

Fallback Path

If CLI tools aren't working: 1. Use VS Code Chat for everything — it's fully capable for this lab 2. The comparison still works: compare Chat-only vs. Chat + Inline 3. Focus on code quality and iteration — tool comparison is secondary

2026-09-02

Hints

- **Prompt structure matters:** "Create a FastAPI endpoint for creating tasks" → OK. "You are a senior Python developer. Read models.py. Add a POST /tasks endpoint to app.py that validates input using the Task model and returns 201 with the created task. Use async/await. Include error handling for duplicate titles." → Much better.
- **Iterate in small steps:** Don't ask for the whole API at once. Build models → service → endpoints → tests. Each step produces better output.
- **VS Code Inline is great for models:** Try typing a Pydantic model definition manually. Watch how the AI completes fields based on the first one you write.
- **The comparison IS the point:** Two tools giving different results for the same task is the most valuable learning experience in this lab. Lean into it.

2026-09-02