

M01 — Generative AI Foundations Lab

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

Time budget: ~20 minutes

Section: Day 1 AM (shared with M02 lab)

2026-09-02

Learning Objectives

After completing this lab, you will be able to: 1. Identify real engineering use cases where GenAI can add value 2. Classify each use case as automation, augmentation, or risk 3. Make your first LLM API call using both CLI and IDE-based tools

2026-09-02

Tool Introductions

Before starting the exercises, familiarize yourself with the GenAI tools available on your sandbox VM. You'll use these throughout the workshop.

OpenCode CLI — Your Provider-Agnostic Assistant

What it is: A terminal-based AI coding assistant that works with any LLM provider (Anthropic, OpenAI, Google, open-source). You run it from the command line.

Why use it: Provider flexibility — switch between LLMs without changing tools. Scriptable — can be integrated into CI/CD pipelines. Works over SSH — no desktop needed. Fast and lightweight.

```
# Verify it's available (you may need to start a new terminal first)
opencode --version    # Should show v1.18+
```

VS Code with AI — AI in Your IDE

What it is: VS Code integrates AI assistance directly into your editor through extensions. Two main modes:

- **Inline Completions (Copilot):** Suggests code as you type — like autocomplete on steroids. Triggered automatically as you write.
- **Chat Agent:** A conversation panel inside VS Code. Select code, ask questions, request changes — all within your editor.

Why use it: Lowest friction — you stay in your editor. Sees your open files and workspace for context-aware suggestions. Great for staying in flow.

Claude Code (Optional — Anthropic Only)

What it is: Anthropic's CLI coding agent. Deeply integrated with Claude models.

Why use it: Excellent at understanding large codebases and complex refactoring tasks. **Requires:** An Anthropic API key. Skip if you don't have one — OpenCode handles the same workflows.

```
claude --version
```

Codex CLI (Optional — OpenAI Only)

What it is: OpenAI's terminal-based coding agent.

Why use it: Fast iteration, strong at web development and rapid prototyping. **Requires:** An OpenAI API key. Skip if you don't have one.

```
codex --version
```

2026-09-02

Concept Review: What We Covered in M01

Take 3 minutes to read through this review. It reinforces the key concepts from the presentation before you apply them in the exercises.

How LLMs Work — The 30-Second Version

When you send a prompt to an LLM, three things happen in sequence:

1. **Tokenization** — Your text is split into tokens (sub-word chunks). "Understanding" might become ["under", "standing"]. Every LLM has a token limit (the context window).
2. **Embedding** — Each token is converted to a vector — a list of numbers capturing its meaning. Similar words get similar vectors.
3. **Transformer Processing** — The attention mechanism weighs which parts of your input matter most, building understanding layer by layer. The model predicts the most probable next token — repeatedly — until it produces a complete response.

Key insight: LLMs don't "know" facts. They predict statistically likely sequences of tokens based on patterns learned from training data. This is why they sometimes hallucinate — the most probable token isn't always the most truthful one.

Critical Constraints

Constraint	What It Means	How to Handle It
Context Window	The model can only "see" a fixed number of tokens at once	Put critical info at the beginning and end of your prompt
Hallucinations	The model confidently generates plausible but incorrect information	Always verify critical outputs; use grounding (RAG)
Probabilistic Output	Same prompt can produce different outputs	Design prompts to be robust; test with multiple runs
No Real Knowledge	The model knows only what was in its training data	Provide context in your prompt; don't ask for post-training facts

The AI Decision Framework

This is the framework you'll use throughout the workshop — and in your real work — to decide when to reach for AI:

Use AI When	Use Deterministic Logic When
The problem is fuzzy or creative	The solution has a known algorithm
You need rapid exploration of options	Correctness is non-negotiable
Human-like language or reasoning helps	Performance is critical
The cost of errors is low	Every output must be verified
You can review and validate the output	The output goes directly to production

Classification: Automation vs. Augmentation vs. Risk

- **Automation:** AI performs the task with minimal human intervention. The output is immediately usable or requires only light review. *Example: generating boilerplate CRUD endpoints from a database schema.*
- **Augmentation:** AI assists but the human remains in control. The AI output is a starting point or suggestion. *Example: AI suggests refactoring options; the developer evaluates and chooses.*
- **Risk:** Involving AI introduces new vulnerabilities that need active mitigation. These aren't "don't use AI" — they're "use AI with guardrails." *Example: AI-generated SQL that could have injection vulnerabilities without review.*

Prerequisites

- Your LLM API key for at least one provider (Anthropic, OpenAI, or Google)
 - Workshop sandbox VM access (SSH or Guacamole desktop)
-

2026-09-02

Environment Verification

Step 1: Configure OpenCode with Your Provider

OpenCode works with any LLM provider. Configure it once using the interactive setup:

```
# Start OpenCode in interactive mode
opencode
```

Once inside OpenCode, use these commands:

```
/connect    # Choose your provider (Anthropic, OpenAI, Google, etc.)
            # Follow the prompts to enter your API key
/model      # Select which model to use
            # Pick any available model from your provider
```

This stores your credentials securely. You only need to do this once. After setup, `opencode` run "your prompt" will use your configured provider.

Step 2: Verify OpenCode

```
# Make sure OpenCode is in your PATH (needed in new terminals)
source ~/.bashrc
opencode --version    # Should show 1.18+
```

Step 3: Verify Python Packages (for Labs that Need pip)

Some labs use Python packages. Set up the shared virtual environment:

```
# One-time setup
sudo apt install -y python3.14-venv
python3 -m venv ~/workshop/venv

# Activate before Python labs (do this in each new terminal)
source ~/workshop/venv/bin/activate
```

Step 4: Run the Setup Check

```
cd ~/workshop
bash m01/check-setup.sh
```

If anything shows , resolve it before starting.

2026-09-02

Exercise 1: Your First LLM Interaction

Try this with both OpenCode (CLI) and VS Code Chat (IDE) to compare the experience.

Option A — OpenCode CLI

```
opencode run "Explain what a token is in the context of large language models. Keep it under 100 words and use a concrete example."
```

Option B — VS Code Chat Agent

1. Open VS Code from the desktop
2. Open the Chat panel: **Ctrl+Shift+I** (or Cmd+Shift+I on Mac layout)
3. Type the same prompt in the chat input

What to Check

- Did the response explain that tokens are sub-word units?
- Did it give a concrete example (like "Hello, world!" being split)?
- Was the response too technical or too vague?
- If it wasn't great, try refining the prompt — this is your first taste of prompt engineering (Module 2).

Optional — Claude Code: `claude -p "Explain what a token is in the context of large language models. Keep it under 100 words and use a concrete example." # Requires ANTHROPIC_API_KEY set`

Optional — Codex: `codex exec "Explain what a token is in the context of large language models. Keep it under 100 words and use a concrete example."`

Exercise 2: Identify 5 Engineering Use Cases

Think about your own development work — your current project, your team's workflow, your organization's challenges. Identify **5 specific tasks or problems** where GenAI could potentially help.

Be concrete. "Write unit tests" is too vague. Good: "Generate unit tests for the `PaymentProcessor.validate()` method covering edge cases for negative amounts, zero amounts, and currency conversion."

Use Case Worksheet

Copy the worksheet from `workshop/m01/use-case-worksheet.md` or fill it out below:

#	Use Case	Automation	Augmentation	Risk	Why This Category?	Verification / Guardrails
1		☐	☐	☐		
2		☐	☐	☐		
3		☐	☐	☐		
4		☐	☐	☐		
5		☐	☐	☐		

See `workshop/m01/use-case-worksheet.md` for a completed example row.

Exercise 3: Classify Each Use Case

For each use case, decide: is it **Automation**, **Augmentation**, or **Risk**?

Read the classification guide in the Concept Review above. Then for each classification, ask yourself:

- If **Automation**: What verification do I need before trusting this output?
 - If **Augmentation**: Where does the human add the most value in this workflow?
 - If **Risk**: What specific guardrails would make this safe to use?
-

2026-09-02

Exercise 4: Test One Classification

Pick your most promising Automation or Augmentation use case. Actually try it with your LLM tool — send a prompt designed to accomplish that task.

Try with at least two approaches: 1. **OpenCode CLI** — one-shot: `bash opencode run "your prompt here"` 2. **VS Code Chat** — select relevant code (if any), open chat, describe the task

Reflection questions: - Did the AI produce useful output on the first try? - What would you change about your prompt to get better results? - Does your classification (Automation vs. Augmentation vs. Risk) still feel right? - Which tool experience did you prefer — CLI or IDE? Why?

2026-09-02

Deliverable

Complete the Use Case Worksheet (5 items with classifications, reasoning, and verification/guardrails) and save it in your workshop directory. You'll reference these throughout the workshop.

2026-09-02

Troubleshooting

Symptom	Likely Cause	Resolution
<code>opencode: command not found</code>	New terminal — PATH not loaded	Run <code>source ~/.bashrc</code> or start a new terminal
API key rejected	Key not configured or expired	Check with <code>opencode config get api_key</code> ; re-export your key
Rate limit error	Too many requests quickly	Wait 30 seconds and retry
Response is gibberish or off-topic	Prompt too vague	Add context, specify output format, give examples
"Connection refused" or timeout	Network or provider issue	Check internet: <code>ping api.openai.com</code> ; switch to offline exercises if persistent
VS Code Chat not responding	Extension not activated	Open Chat panel (Ctrl+Shift+I); ensure you're signed in
Claude Code fails	No Anthropic key	Use OpenCode instead — same patterns work
Codex fails	No OpenAI key	Use OpenCode instead — same patterns work

Fallback Path

If you cannot connect to any LLM provider: 1. Work through the classification exercise manually (it's a thinking exercise first) 2. Use a free web tier (chat.openai.com, claude.ai, gemini.google.com) for Exercises 1 and 4 3. The facilitator can provide a shared API key for the session if needed

2026-09-02

Hints

Stuck on Exercise 2? Here are prompts to get you thinking: - "What's the most tedious, repetitive task in my current project?" - "What do I dread writing? (Tests? Docs? Boilerplate?)" - "Where do I spend time debugging that AI might catch faster?" - "What task do I put off because it's boring but necessary?"

2026-09-02