

# M05 — Building AI-Powered Apps Lab

---

Lab — Generative AI Boot Camp for Developers — Exported 2026-08-03

**Time:** ~20 min | **Day 2 AM** (shared with M04)

---

2026-08-03

## Learning Objectives

Build a simple AI-powered chatbot API. Implement a basic LLM endpoint, then extend it with prompt chaining — a foundational pattern for AI applications.

---

2026-08-03

# Concept Review: LLM APIs & Prompt Chaining

## Every LLM API Follows the Same Pattern

Regardless of provider (OpenAI, Anthropic, Google), the core interface is identical:

```
Your App → [prompt + config] → LLM API → [response] → Your App
```

**Common parameters across all providers:** - **model** — which model to use (gpt-4o, claude-sonnet-4, gemini-pro) - **temperature** — randomness (0 = deterministic, 1 = creative) - **max\_tokens** — response length limit - **system prompt** — persistent instructions for the model

## Provider-Agnostic Architecture

```
# Pattern: abstract the provider behind a simple interface
def call_llm(prompt: str, system: str = "") -> str:
    """Call the configured LLM provider. Provider is set via env var."""
    provider = os.getenv("LLM_PROVIDER", "openai")
    if provider == "openai":
        return call_openai(prompt, system)
    elif provider == "anthropic":
        return call_anthropic(prompt, system)
    # ...
```

## Prompt Chaining

Prompt chaining is the simplest AI workflow pattern: the output of one LLM call becomes the input to the next.

```
User text → LLM Call 1: "Extract topics" → topics
           → LLM Call 2: "Summarize focusing on: {topics}" → summary
```

**Why chain instead of one prompt?** - Each step has a focused task → higher quality output - You can validate intermediate results - You can add non-AI processing between calls (filtering, formatting, database lookups)

## Production Considerations

Before deploying an AI-powered endpoint, consider: - **Rate limiting:** Prevent abuse and control costs - **Caching:** Don't re-call the LLM for identical prompts - **Error handling:** LLM APIs fail — timeouts, rate limits, content filters - **Authentication:** Don't expose your AI endpoint to the public internet - **Cost monitoring:** Track token usage per request

## Exercise: Build a Chatbot API

A scaffold is provided at `workshop/m05/scaffold.md`. Copy it to `~/workshop/m05-chatbot/` to get started quickly.

### Step 1 — Basic Chat Endpoint (8 min)

Implement `POST /chat`. It accepts `{"prompt": "..."}` , calls your LLM, and returns `{"response": "..."}` .

Use **OpenCode** to implement the endpoint:

```
cd ~/workshop/m05-chatbot
opencode run "Read app.py. Implement the /chat endpoint. Use the OpenAI Python SDK (or anthropic, google-generativeai – whichever you have keys for). Read the API key from an environment variable. Handle: missing API key (return 500 with message), API timeout (return 504), invalid response (return 502)."
```

Test it:

```
curl -X POST http://localhost:8001/chat \
  -H "Content-Type: application/json" \
  -d '{"prompt": "What is 2+2?"}'
```

### Step 2 — Prompt Chaining (8 min)

Add `POST /analyze`. This endpoint demonstrates prompt chaining: 1. Call LLM: "Extract the top 3 key topics from this text: {text}" 2. Call LLM: "Summarize this text, focusing on these topics: {topics}" 3. Return both `topics` (list) and `summary` (string)

Use a **different tool** than Step 1 — if you used OpenCode, try **VS Code Chat**. Select the `/analyze` stub, open Chat, and describe the two-step chain.

### Step 3 — Reflection (4 min)

What 3 things would you need to add before deploying this to production?

---

## Deliverable

Working chatbot API with two endpoints: `/chat` (basic) and `/analyze` (prompt chaining).

---

2026-08-03

## Troubleshooting

| Symptom                            | Fix                                                                                       |
|------------------------------------|-------------------------------------------------------------------------------------------|
| API key not found                  | Verify env var: <code>echo \$OPENAI_API_KEY</code> (or ANTHROPIC_API_KEY, GOOGLE_API_KEY) |
| Import error for LLM SDK           | <code>pip install openai</code> (or anthropic, google-generativeai)                       |
| Timeout on LLM call                | Add timeout parameter to API call; default is often 60s                                   |
| Chained prompts produce bad output | Validate Step 1 output before feeding to Step 2; add error handling                       |
| Port 8001 already in use           | Change port or kill existing process: <code>lsof -ti:8001   xargs kill</code>             |

## Using Your Own Provider

Replace `openai` with `anthropic`, `google.generativeai`, or any provider SDK. The pattern is identical — only the import and API call syntax changes. Use AI to help you translate between providers:

```
opencode run "Convert this OpenAI code to use the Anthropic Python SDK instead.  
[ paste your /chat implementation ]"
```

2026-08-03