LLM API Basics and OpenAI Python Client

LLM Engineering Lab Day 2

- Navigated to Week 1, Day 2 directory in Cursor
- Reminder to select correct Python environment (VENV, Python 3.12)
- Resource page available and updated regularly
- Repeating yesterday’s workflow with more structure

Chat Completions API Overview

- Chat Completions API: standard way to interact with LLMs
- Originated by OpenAI, now widely adopted by other providers
- API predicts next likely message in a conversation (like predictive text)
- Underlying mechanism: predicts next tokens, not just words

Making an API Call

- Use load_env to load secrets from .env file, check for OpenAI API key
- If API key not found, check .env and troubleshooting resources
- Endpoint explained: HTTP URL for API requests (see Technical Foundations guide if unfamiliar)
- OpenAI endpoint: https://api.openai.com/v1/chat/completions
- HTTP headers: specify content type (JSON), authorization with Bearer token (API key)
- Payload: JSON dictionary with model (e.g., GPT-5 Nano) and messages (list of dicts with role and content)
- Example message: user role, content "Tell me a fun fact"

API Response Details

- Response is JSON with fields like id, object, choices
- Choices: list, first item contains message dictionary with content
- Example fun fact returned: more possible unique games of chess than atoms in the observable universe (~10^120)
- Demonstrated how to extract content from JSON response in Python

Improving the Workflow

- Direct HTTP requests and manual JSON parsing are messy and inefficient
- OpenAI Python client library simplifies the process
- Client library wraps HTTP requests, returns Python objects
- Not special code—just a lightweight, open-source wrapper for API calls
- Allows for more elegant and readable Python code

General Notes

- Emphasis on understanding both the low-level (HTTP/JSON) and high-level (client library) approaches

- Encouragement to use resources and ask questions if stuck
- Transitioning soon to using non-OpenAI/free models