APPLIED AND GENERATIVE AI

Prompt Engineering for Developers

Level: Practitioner • 2 days (expandable to 3) • Virtual, In-person

Overview

For a developer, a prompt is not a conversation, it is an interface. When a prompt lives inside an application, it runs thousands of times, unattended, against inputs nobody previewed, and the rest of the system depends on what comes back. That changes everything about how prompts must be written: they need structure, they need output a program can parse, and they need the same engineering discipline as any other code that ships.

This is a hands-on, practitioner course. It builds from the API-level mental model developers actually need: messages, roles, system prompts, and parameters, then treats prompts as engineered artifacts, structured inputs producing validated, structured outputs, with techniques like few-shot examples and prompt chaining layered on top. The second day moves the prompts into an application: tool and function calling, integrating and versioning prompts in a codebase, defending against prompt injection, and testing prompts so changes do not silently break behavior. The course goes deep on this one path rather than surveying every prompting trick. Every module includes a lab, and each module builds on the one before it.

Who Should Attend

- Developers integrating LLM calls into applications and services
- Engineers who prompt well in a chat window but need prompts that run unattended in code
- Technical leads defining prompting standards for a development team

Non-developers who want strong prompting skills should take *Foundations of Prompt Engineering* instead.

Prerequisites

- Working proficiency in Python or another mainstream language
- Comfort with REST APIs and JSON
- Some hands-on experience with an LLM chat tool; API experience is helpful but not required

What You Will Learn

- Explain the API-level anatomy of an LLM call: messages, roles, system prompts, and parameters
- Build structured prompts from templates and get structured, validated output back
- Apply few-shot examples, step-by-step reasoning, and prompt chaining in code
- Implement tool and function calling so the model can invoke your code
- Integrate prompts into an application with versioning, safe data injection, and injection defenses
- Test prompts systematically and catch regressions before users do

Course Outline

Day one: prompts as engineered artifacts

- The API-Level Mental Model
  - Messages and roles: system, user, and assistant, and what belongs in each
  - Parameters that matter: temperature, max tokens, and stop conditions
  - The context window as a budget, and what happens when it runs out
  - Lab: call the Claude or OpenAI API directly and observe how each parameter changes behavior
- Structured Prompts, Structured Output
  - Prompt templates: separating instructions from data with clear delimiters
  - Getting JSON out: schemas, structured output modes, and parsing defensively
  - Validating output and deciding when to retry
  - Lab: build a prompt template that extracts structured data and validates it against a schema
- Techniques That Raise Reliability
  - Few-shot examples chosen and formatted programmatically
  - Eliciting step-by-step reasoning, and when it earns its token cost
  - Prompt chaining: decomposing one unreliable ask into several dependable ones
  - Lab: take an unreliable single-prompt task and rebuild it as a reliable chain

Day two: prompts inside applications

- Tool and Function Calling
  - Defining functions the model can call: names, descriptions, and schemas
  - The calling loop: handling the model's request, executing, and returning results
  - Multiple tools, ambiguous choices, and handling errors in tool results
  - Lab: expose two functions to the model and build the full calling loop around them
- Integrating Prompts into a Codebase
  - Where prompts live: templates, configuration, and versioning alongside code
  - Injecting user data safely, and prompt injection: what it is and practical defenses
  - Handling model and prompt changes without breaking callers
  - Lab: wire a versioned prompt into a small application with sanitized user input
- Testing and Hardening Prompts
  - Why prompt changes need regression tests, and what a prompt test looks like
  - Building a small test set and scoring outputs automatically
  - Cost and latency as engineering constraints
  - Lab: build a regression test suite for your prompts and use it to evaluate a prompt change

Extended Version

The three-day version keeps the same gradient and adds depth on the production path:

- Prompting with retrieved context, as a bridge to *Retrieval-Augmented Generation (RAG) with Vector Databases*
- Deeper evaluation practice, connecting to *Evaluating and Monitoring Generative AI Applications*
- Advanced structured output and multi-model strategies

- A capstone that takes one prompt-driven feature from first draft to tested, versioned, injection-hardened integration