APPLIED AND GENERATIVE AI

Developing Advanced Generative AI Applications

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

Overview

Getting a large language model to answer a prompt is easy. Getting it to behave like a dependable software component is not: models return prose when you need data, fail in ways ordinary code never does, and sit behind rate-limited APIs with real latency and real cost. The gap between a working demo and a production LLM feature is exactly the set of patterns this course teaches.

This is a hands-on, advanced course. It starts with the judgment layer: a clear mental model of where LLM applications actually fail and which patterns earn their complexity, because the most common advanced mistake is adding machinery a simpler design did not need. From there it layers up one pattern at a time: structured output you can validate, tool and function calling, streaming responses, and the resilience engineering that keeps a feature working when the model or the API misbehaves. The course deliberately covers fewer patterns than a survey would and goes deep on each, grounded in the Claude and OpenAI APIs. Every module includes a lab, and each module builds on the one before it.

Who Should Attend

- Developers who have built a working LLM application and need it to hold up in production
- Engineers responsible for LLM features that other systems and teams depend on
- Technical leads setting patterns and standards for generative AI development

Learners who have not yet built an LLM-backed application should take *Building Generative AI Applications* first.

Prerequisites

- Experience building an application that calls an LLM API (Claude, OpenAI, or similar)
- Working proficiency in Python or TypeScript
- Comfort with REST APIs, JSON, and asynchronous code

What You Will Learn

- Judge which advanced patterns a given LLM feature actually needs, and which it does not
- Design structured output with schemas, validation, and repair strategies
- Build tool and function calling loops the model can drive reliably
- Implement streaming responses, including streaming with structured output and tools
- Engineer for failure: timeouts, retries, rate limits, fallbacks, and graceful degradation
- Harden a complete LLM feature for cost, latency, and testability

Course Outline

Day one: from working call to reliable component

- Thinking in Production Patterns
  - Where LLM applications really fail: a taxonomy of model, API, and integration failures

  - The anatomy of a production LLM feature, and the cost of each added pattern
  - Deciding what your feature needs: a decision framework used throughout the course
  - Lab: audit a naive LLM integration and rank its failure points by likelihood and impact
- Structured Output
  - Why prose is not an interface: getting JSON you can trust
  - Schemas and structured response modes in the Claude and OpenAI APIs
  - Validation, repair loops, and deciding when to retry versus fail
  - Lab: convert a free-text feature to schema-validated output with a repair path
- Tool Use and Function Calling
  - Defining tools the model can call well: names, descriptions, and parameters
  - The tool call loop: request, execute, return results, continue
  - Handling tool errors and results the model did not expect
  - Lab: build a feature where the model chooses among several tools to answer a request

Day two: behavior under real conditions

- Streaming
  - Why streaming matters for perceived latency and user experience
  - Server-sent events and streaming client patterns
  - Streaming combined with structured output and tool calls
  - Lab: convert a blocking endpoint to a streamed response with progressive rendering
- Robust Error Handling and Resilience
  - Timeouts, retries with backoff, and rate limit handling
  - Fallback strategies: alternate models, cached answers, and honest degraded modes
  - Idempotency and making LLM calls safe to repeat
  - Lab: inject failures into a working feature and make it degrade gracefully
- Hardening a Complete Feature
  - Composing the patterns: what a finished production feature looks like
  - Cost and latency budgets, and where each pattern spends them
  - Testing nondeterministic behavior, and the handoff to systematic evaluation
  - Lab: take one feature through a hardening checklist and defend each decision

Extended Version

The three-day version keeps the same gradient and adds room to go deeper:

- Multi-provider design: abstracting across Claude and OpenAI, and gateway patterns
- Caching strategies, including prompt caching, and their cost and correctness tradeoffs
- Deeper evaluation of hardened features, connecting to *Evaluating and Monitoring Generative AI Applications*
- A capstone that carries one LLM feature from naive prototype to a hardened, tested, production-ready implementation