AI Engineer - 120 Hours

Course Overview

The **AI Engineer Program** is an intensive hands-on training designed for software developers and data professionals who want to build practical AI-powered applications and workflows. The program provides a comprehensive foundation in modern AI engineering, covering Large Language Models (LLMs), Prompt Engineering, Retrieval-Augmented Generation (RAG), AI Agents, evaluation methodologies, observability, security, and deployment practices.

Throughout the course, participants will learn how to design, develop, and deploy production-oriented AI applications using Python and industry-standard tools. The curriculum focuses on real-world engineering challenges, including integrating AI models with external data sources, building agentic workflows, implementing guardrails and monitoring mechanisms, and ensuring reliable and scalable AI solutions.

By combining technical depth with practical implementation, the program prepares participants to take AI applications from concept to production-ready solutions and equips them with the skills required for modern AI engineering roles

Target Audience

- Software Developers and Full-Stack Developers
- Backend Engineers
- Data Engineers
- Data Scientists seeking to expand into AI application development
- Machine Learning Engineers
- Cloud and Platform Engineers
- DevOps and MLOps Professionals

What Participants Will Gain?

- Build production-ready AI applications using Python and modern AI frameworks
- Design and implement Retrieval-Augmented Generation (RAG) solutions
- Develop AI agents and multi-step AI workflows
- Integrate Large Language Models with enterprise data sources and business systems
- Create robust prompts and structured-output workflows
- Implement evaluation, testing, monitoring, and observability mechanisms for AI systems
- Make informed decisions regarding RAG, fine-tuning, LoRA, and model optimization approaches
- Package and deploy AI applications using modern DevOps and LLMOps practices

Course Syllabus

1. Python for AI Engineering

Students build the practical Python foundation needed for AI application development: clean code, API usage, structured data handling, error handling, and service-oriented programming patterns.

- Python refresher for application development: functions, modules, packages, virtual environments, notebooks vs scripts.
- Working with data structures, files, JSON, CSV, APIs, HTTP requests, and external services.
- Object-oriented Python and practical class design for services, clients, tools, and reusable components.
- Error handling, exceptions, logging basics, configuration, environment variables, and secrets handling.
- Type hints, data classes, and structured validation patterns useful for AI applications.

2. AI Engineering Orientation and LLM Foundations

Students learn what AI engineering means, how large language models work at a high level, and how model behavior affects real application design.

- What LLMs are and what kinds of tasks they can perform.
- The basic makeup of an LLM: tokenization, token embeddings, transformers, and model architecture families.
- Why LLMs are fundamentally prediction systems, and why this matters for engineering reliability.
- How LLMs are trained and adapted: pre-training, fine-tuning, RLHF, and LoRA at a conceptual level.
- Inference basics: pipelines, tokenization, chat templates, generation, and simple model interaction.

3. Prompt Engineering, LLM APIs, and Structured Outputs

Students learn how to design prompts and model interactions that are clear, testable, reusable, and suitable for application code rather than casual experimentation.

- Why prompt engineering matters: bad prompts vs better prompts, and the shift from casual chatting to task design.
- Prompt anatomy: role, context, data, task, constraints, output format, and validation.
- Core prompting techniques: instructional prompting, role prompting, structured-output prompting, few-shot prompting, and clarification workflows.
- Prompting across practical workflows: scoping, planning, code generation, code review, and stakeholder-facing summaries.

- Reliability practices: verification loops, prompt-quality rubrics, asking for tests, reviewing generated code, and prompt chaining.

4. From Notebook to AI Application

Students learn how to move from exploratory notebook code to maintainable AI application components with clear interfaces, validation, configuration, and service boundaries.

- What changes when moving from notebook experimentation to application code.
- Common AI application architecture patterns: direct LLM call, RAG backend, tool-using assistant, and workflow service.
- Separating application layers: API endpoint, service logic, model provider, vector store, storage, and client.
- Request and response contracts, structured schemas, validation, and predictable error responses.
- Configuration, environment variables, secrets, and simple project structure for AI applications.

5. Retrieval-Augmented Generation

Students learn how to build RAG systems that connect language models to external knowledge sources and how to reason about retrieval quality, context construction, and system limitations.

- Why RAG is used, common use cases, and the core RAG components: retriever, chunking, embeddings, vector databases, context injection, and query reformulation.
- Document ingestion: inspection, content extraction, cleanup, normalization, and chunking strategies.
- Embeddings and vector-store ingestion using modern embedding models and vector databases.
- Retrieval techniques: lexical vs semantic search, hybrid search, metadata filtering, whole-document search, thresholds, and reranking.
- Context injection and prompt construction strategies: stuffing, map-reduce, refine, multi-query fusion, and self-ask.

6. Reliable LLM Applications: Evaluation, Error Handling, and Observability

Students learn how to make AI applications more reliable by anticipating common failure modes, validating outputs, evaluating behavior, and making system activity visible through logs and traces.

- Why LLM applications fail: malformed outputs, hallucinations, brittle prompts, bad retrieval, tool failures, timeouts, and nondeterminism.
- Reliability practices: validation, retries, recovery, fallback behavior, and workflow-level failure handling.
- Evaluation foundations: golden cases, custom evaluation sets, retrieval quality, answer quality, and regression-style testing.

- Observability concepts: tracing, run records, prompts, retrieved chunks, tool calls, outputs, latency, and failures.
- Practical debugging workflow: inspect inputs, retrieved context, model output, validation errors, and final response.

7. Agents, Tools, and LangGraph Workflows

Students learn how to build bounded agentic workflows that combine model reasoning with tools, state, routing, and controlled multi-step execution.

- Agent concepts: agent loop, memory, tools/APIs, environment, state, planning, policy, and human-in-the-loop.
- Agent reasoning patterns: ReAct, planner-executor separation, self-reflection, router/decision prompts, and prompt patterns for agents.
- Tool integration: structured outputs vs tools, JSON-based tool invocation, tool descriptions, failure handling, retries, and grounding responses.
- LangGraph fundamentals: graph paradigm, conditional execution, state management, reducers, subgraphs, messages, prompt templates, tools, and structured output.
- Agent graphs with tools, agents as tools, tracing, and a cautious overview of memory and multi-agent patterns.

8. AI Security Basics, Guardrails, and Human Escalation

Students learn basic safety and control patterns for AI systems that interact with users, documents, tools, and business workflows.

- Human-in-the-loop patterns and when to use them.
- Guardrails and safety as part of workflow design: permissions, policy checks, and tool boundaries.
- Tool inventory and contracts: read vs write tools, trusted vs untrusted sources, and source-of-truth rules.
- Security and compliance basics for AI workflows, including sensitive data and operational risk.
- Escalation patterns: when the system should answer, ask for clarification, stop, or send the task to a human.

9. Model Adaptation and Optimization

Students gain practical literacy in model adaptation and optimization so they can understand when fine-tuning, LoRA, quantization, or RAG are appropriate.

- Why fine-tune at all, what fine-tuning is good at, and when it is not the right answer.
- The fine-tuning mental model and common fine-tuning data formats.
- Evaluation essentials: model evaluation vs system evaluation, public benchmarks vs custom evaluation, metric families, human evaluation, LLM-as-judge, and golden sets.
- Full fine-tuning vs LoRA: intuition, pros/cons, hyperparameters, and practical diagnostics.
- Decision guide: fine-tuning vs RAG

10. Deployment and LLMOps Basics

Students learn how to package, configure, run, monitor, and reason about a small AI service in a production-aware way.

- What LLMOps adds beyond ordinary backend deployment: prompts, model providers, vector stores, traces, evaluation sets, latency, token cost, and failure modes.
- Packaging an AI application service: application structure, dependencies, environment variables, secrets, and configuration.
- Docker basics for AI applications: Dockerfile, image build, container run, port mapping, and environment injection.
- Health checks, readiness mindset, logging, basic cost and latency tracking, and simple operational dashboards or log tables.
- Lightweight deployment workflow: local run, containerized run, simple hosted deployment or deployment demo, and basic CI/CD discussion.

11. Final Project

- Design a small AI assistant or AI workflow with clear users, inputs, outputs, and success criteria.
- Build a working LLM-powered feature, including RAG over a small document collection.
- Use structured outputs, validation, and at least one tool or workflow step beyond simple question answering.
- Add basic error handling, logging, evaluation, and a guardrail or human-escalation path.