

Rapid LLM Application Prototyping

A focused Python workflow from prompt to a small application

FA-0678 · 1 day

Agentic & Generative AI · Software Development

Introduction

A one-day intensive for experienced Python developers exploring rapid prototyping with large language models. Build one small scaffolded application using an arranged model endpoint and backend, then review output quality and implementation limits.

The broader ecosystem, local inference, fine-tuning and deployment trade-offs are demonstrations or conceptual comparisons. This course does not promise cost savings, scalable production software or model-training proficiency in one day.

Learning outcomes

- Explain how LLM assistance can support an iterative prototype workflow and where review is necessary.
- Select an available model/tool path against access, licence, context and resource constraints.
- Integrate one model call into a small Python application and test selected outputs.
- Compare prompting and model adaptation at an introductory level.
- Identify cost, latency, data-handling and deployment gaps before further work.

Prerequisites

- Advanced Python skills, environment management and API/backend familiarity.
- Basic Linux/command-line familiarity; Docker knowledge is helpful.

- Access to the arranged API or local model and required licences/permissions; API billing/limits must be checked independently of a chatbot subscription.
- For optional local demonstrations, use the prepared compatible GPU/memory/toolchain setup. Hardware suitability depends on model, precision and workload, not a universal CUDA-core count.

Training outline

1. 1. RAD and the LLM Ecosystem

- Iterative application prototyping, code generation and candidate use cases; test generated code rather than assume reduced errors.
- Transformer/self-attention foundations and representative hosted/open-weight models.
- Compare licences, provider terms, model access and capabilities rather than label every downloadable model open-source.
- Roles of Transformers, orchestration frameworks and provider APIs.

2. 2. Prepared Development Environment

- Python virtual environments and reproducible dependencies; Docker as an optional setup.
- Hosted API versus local inference; memory, precision, context, compatibility and workload considerations.
- GPU/cloud examples as deployment options, not mandatory hardware purchases or a universal A100/H100 requirement.
- Fine-tuning as a conceptual comparison or prepared demonstration: provider/model eligibility must be checked; OpenAI's fine-tuning platform is a legacy/winding-down option, not a guaranteed new-account lab.

3. 3. One Small LLM-powered Application

Design principles for LLM-based applications:

- Prompt engineering and input optimization.
- Managing context windows for better performance.

Rapid prototyping workflows:

- Using LangChain to chain LLM outputs.
- Integrating LLMs with Python backends using Flask and FastAPI.

Use one arranged backend/framework, bounded inputs and non-sensitive teaching data. Keep credentials server-side, validate generated output and separate retrieved/user content from trusted application instructions.

4. Review and Next Steps

- Test a small set of representative and failure cases; assess task correctness rather than fluency alone.
- Inspect cost/latency and error handling; explain model-specific context limits.
- Identify remaining security, ethical, evaluation and operational work before production deployment.