LLM-Anwendungen mit Python entwickeln

Sprachmodelle zuverlässig in Python-Anwendungen integrieren

01 INTRO

Ein Sprachmodell aufzurufen ist einfach. Eine belastbare Anwendung daraus zu entwickeln ist deutlich anspruchsvoller. Eingaben müssen strukturiert, Modellantworten verarbeitet und validiert, Fehler behandelt, Kosten kontrolliert und Ergebnisse getestet werden. Hinzu kommen Anforderungen an Security, Datenschutz, Observability und den produktiven Betrieb.

Der Kurs vermittelt die Entwicklung professioneller LLM-Anwendungen mit Python. Die Teilnehmer bauen schrittweise eine vollständige Anwendung auf und integrieren Sprachmodelle über APIs in eine saubere Softwarearchitektur. Im Mittelpunkt steht nicht ein bestimmtes Framework, sondern die Frage, wie sich probabilistische Modelle zuverlässig in klassische, testbare und wartbare Software integrieren lassen.

DAUER

3 Tage

FORMAT

Präsenz · Live Online · Inhouse

VORKENNTNISSE

Sichere Python-Grundkenntnisse und praktische Erfahrung in der Softwareentwicklung

02 TRAININGSPROGRAMM

Inhalte

LLM-Anwendungen einordnen

- Sprachmodell und Anwendung unterscheiden
- Modell als externe Komponente
- probabilistische und deterministische Software
- typische LLM-Anwendungen
- Textverarbeitung
- Informationsgewinnung
- Klassifikation
- Assistenzfunktionen
- Grenzen geeigneter Anwendungsfälle

Architektur einer LLM-Anwendung

- Benutzer oder aufrufendes System
- Application Layer
- Model Client
- Prompt und Kontext
- Modell
- strukturierte Antwort
- Validierung
- Persistenz und externe Systeme
- klare Verantwortlichkeiten

Python-Projekt aufsetzen

- Projektstruktur
- virtuelle Umgebung
- Abhängigkeiten
- Konfiguration
- API Credentials
- Entwicklungs- und Produktionsumgebung
- Tests
- reproduzierbares Setup

Modelle über APIs verwenden

- Model Provider
- Client Libraries
- Authentifizierung
- Request
- Response
- Modellparameter
- Fehler
- Timeouts
- Provider-spezifischen Code kapseln

Modelle auswählen

- unterschiedliche Modellklassen
- Qualität
- Geschwindigkeit
- Kontextgröße
- Kosten
- multimodale Fähigkeiten
- strukturierte Ausgaben
- Modellwahl nach Aufgabe
- größtes Modell ist nicht automatisch die beste Lösung

Prompts als Bestandteil der Anwendung

- System Instructions
- Benutzerinput
- Anwendungskontext
- klare Aufgabenbeschreibung
- gewünschtes Ausgabeformat
- Beispiele
- Prompts versionieren
- Prompt Engineering nicht mit Anwendungsarchitektur verwechseln

Prompt Templates

- statische und variable Bestandteile
- Parameter
- wiederverwendbare Templates
- unterschiedliche Aufgaben
- Kontext einsetzen
- Templates testen
- Versionierung
- Prompt-Logik nicht über die Anwendung verteilen

Strukturierte Ausgaben

- Freitext gegenüber strukturierten Ergebnissen
- JSON
- Schemas
- typisierte Python-Objekte
- Enums
- optionale Felder
- verschachtelte Strukturen
- maschinenlesbare Antworten

Validierung

- syntaktisch korrekt ist nicht fachlich korrekt
- Schema-Validierung
- Datentypen
- Wertebereiche
- Pflichtfelder
- fachliche Regeln
- Cross-Field Validation
- ungültige Antworten kontrolliert behandeln

LLM-Ausgaben in Python-Modelle überführen

- Type Hints
- Dataclasses
- Validierungsmodelle
- Parsing
- Schema und Anwendung
- saubere Schnittstellen
- Modelloutput nicht ungeprüft weiterreichen
- Domänenmodell vom Providerformat trennen

Fehlerbehandlung

- API nicht erreichbar
- Timeout
- Rate Limit
- ungültige Modellantwort
- abgeschnittene Antwort
- verweigerte Antwort
- Validierungsfehler
- kontrollierte Fehlerzustände

Retry-Strategien

- technische und fachliche Fehler unterscheiden
- wiederholbare Fehler
- Backoff
- Rate Limits
- erneute Modellaufrufe
- maximale Versuche
- Kosten berücksichtigen
- nicht jeden Fehler durch Retry verstecken

Streaming

- vollständige Antwort gegenüber Stream
- Tokens schrittweise verarbeiten
- Benutzerfeedback
- Time-to-first-token
- Abbruch
- Fehler während des Streams
- strukturierte Verarbeitung
- geeignete Anwendungsfälle

Asynchrone Modellaufrufe

- async und await
- I/O-lastige LLM-Aufrufe
- mehrere unabhängige Requests
- Concurrency
- Timeouts
- Rate Limits
- Fehlerbehandlung
- kontrollierte Parallelität

Conversation State

- einzelne Anfrage
- mehrstufiger Dialog
- Conversation History
- relevante Nachrichten
- Systemkontext
- Zustandsverwaltung außerhalb des Modells
- Session
- Anwendung kontrolliert den Zustand

Kontextmanagement

- begrenztes Kontextfenster
- relevante Informationen auswählen
- Gesprächsverlauf
- Anwendungsdaten
- unnötigen Kontext entfernen
- Zusammenfassungen
- Tokenbudget
- Kontext nicht unbegrenzt wachsen lassen

Persistenz

- Conversations speichern
- Anwendungszustand
- Modellantworten
- Benutzerinformationen
- Datenbank
- Session Store
- Lebenszyklus von Daten
- Datenschutz berücksichtigen

Mehrstufige LLM-Workflows

- mehrere Modellaufrufe
- Ausgabe eines Schritts als Eingabe des nächsten
- Analyse und Generierung trennen
- deterministische Zwischenschritte
- Validierung zwischen Schritten
- Fehler lokalisieren
- Workflow explizit modellieren
- nicht automatisch einen Agenten bauen

Deterministische und probabilistische Komponenten

- was Python besser kann
- was ein LLM besser kann
- Berechnungen
- Regeln
- Datenbankabfragen
- Transformationen
- Sprachverständnis
- klare Systemgrenzen

Function Calling und Tool Use

- Modell erzeugt strukturierten Tool-Aufruf
- Tool-Auswahl
- Parameter
- Validierung
- Python-Funktion ausführen
- Ergebnis zurückgeben
- Tool-Aufruf und Ausführung trennen
- Anwendung behält Kontrolle

Tools sicher integrieren

- erlaubte Funktionen
- Parameter validieren
- Seiteneffekte
- Berechtigungen
- externe Systeme
- kritische Aktionen
- Modell darf nicht automatisch alles ausführen
- Least Privilege

Klassische Workflows statt Agenten

- feste Prozessschritte
- bekannte Reihenfolge
- deterministische Steuerung
- LLM nur an geeigneten Stellen
- Fehler einfacher lokalisieren
- bessere Testbarkeit
- geringere Kosten
- Agent nur bei tatsächlichem Bedarf

RAG anbinden

- externes Wissen als Kontext
- Retrieval als vorgelagerte Komponente
- Anfrage an Wissenssystem
- Quellen
- Kontext an das Modell
- Antwort verarbeiten
- bestehende RAG-Komponente integrieren
- RAG-Engineering bleibt Gegenstand des RAG-Kurses

MCP anbinden

- externe Fähigkeiten über MCP
- MCP Client als Anwendungskomponente
- Tools entdecken
- Tool-Aufrufe
- Ergebnisse verarbeiten
- Berechtigungen
- vorhandene MCP-Server verwenden
- MCP-Server-Entwicklung bleibt Gegenstand der MCP-Kurse

Agenten integrieren

- Agent als externe oder interne Komponente
- Auftrag übergeben
- Status
- Ergebnisse
- Fehler
- Abbruch
- kontrollierte Systemgrenze
- Agent Engineering bleibt Gegenstand des Agentenkurses

Dokumentverarbeitung

- Textdateien
- strukturierte Dokumente
- längere Inhalte
- Inhalte segmentieren
- Informationen extrahieren
- strukturierte Ergebnisse
- Quellen erhalten
- Verarbeitungspipeline

Information Extraction

- Entitäten
- Attribute
- Klassifikation
- Beziehungen
- strukturierte Daten aus Text
- Schema
- fehlende Informationen
- Confidence nicht erfinden

Klassifikation mit LLMs

- Kategorien
- Label
- Few-Shot-Beispiele
- strukturierte Antwort
- unbekannte Kategorie
- Mehrdeutigkeit
- Testdatensatz
- klassische Verfahren als Alternative

Transformation und Generierung

- Texte zusammenfassen
- Inhalte umformulieren
- Formate konvertieren
- Informationen zusammenführen
- Zielgruppe
- Stil
- strukturierte Ergebnisse
- fachliche Fakten erhalten

Multimodale Anwendungen

- Text und Bilder
- Dokumentseiten
- Screenshots
- Bildanalyse
- strukturierte Ergebnisse
- Modellfähigkeiten
- größere Inputs
- Grenzen und Kosten

Halluzinationen behandeln

- Modellantwort klingt plausibel
- erfundene Fakten
- erfundene Quellen
- fehlende Informationen
- Unsicherheit
- Grounding
- externe Verifikation
- Anwendung muss mit Fehlern rechnen

Input Validation

- Benutzerinput
- Länge
- Dateitypen
- strukturierte Daten
- unerwartete Eingaben
- externe Inhalte
- Normalisierung
- sichere Verarbeitung

Prompt Injection

- Benutzer versucht Instruktionen zu verändern
- indirekte Prompt Injection
- Dokumente
- Webseiten
- externe Daten
- Daten und Instruktionen trennen
- Modellinstruktionen sind keine Sicherheitsgrenze
- Auswirkungen auf Tools begrenzen

Output Handling

- Modelloutput als untrusted Input
- HTML
- Markdown
- URLs
- Code
- strukturierte Daten
- Escaping
- Validierung
- Output nicht ungeprüft ausführen

Datenschutz und vertrauliche Daten

- personenbezogene Informationen
- Unternehmensdaten
- Quellcode
- Dokumente
- API Provider
- Logging
- Speicherung
- Datenminimierung

Secrets und Credentials

- API Keys
- Tokens
- Umgebungsvariablen
- Secret Stores
- Rotation
- keine Secrets in Prompts
- keine Secrets in Logs
- unterschiedliche Umgebungen

LLM-Anwendungen testen

- klassische Unit Tests
- deterministische Komponenten
- Model Client mocken
- strukturierte Antworten
- Fehlerfälle
- Integrationstests
- End-to-End-Tests
- probabilistische Komponenten getrennt betrachten

Tests ohne echten Modellaufruf

- Model Gateway abstrahieren
- Test Doubles
- gespeicherte Antworten
- deterministische Tests
- Validierungslogik
- Fehlerbehandlung
- schnell und kostengünstig testen
- Modelltests separat durchführen

LLM-Evaluation

- Testdatensatz
- repräsentative Eingaben
- erwartete Eigenschaften
- fachliche Kriterien
- strukturierte Metriken
- menschliche Bewertung
- automatische Evaluation
- Ergebnisse über Modellversionen vergleichen

Regressionstests

- Promptänderung
- Modellwechsel
- Parameteränderung
- neuer Provider
- neue Anwendungslogik
- Evaluationsdatensatz
- vorher und nachher vergleichen
- Qualitätsverlust erkennen

Grenzen von LLM-as-a-Judge

- probabilistischer Evaluator
- Bias
- Bewertungs-Prompt
- Modellabhängigkeit
- Referenzdaten
- deterministische Kriterien
- menschliche Stichproben
- mehrere Signale kombinieren

Model Gateway

- Providerzugriff kapseln
- einheitliche Anwendungsschnittstelle
- Modellwahl
- Konfiguration
- Fehlerbehandlung
- Logging
- Providerwechsel
- Testbarkeit

Fallback-Strategien

- Modell nicht verfügbar
- Rate Limit
- alternatives Modell
- reduzierte Funktion
- Warteschlange
- kontrollierte Fehlermeldung
- fachliche Auswirkungen
- Fallback nicht blind automatisieren

Kostenkontrolle

- Tokenverbrauch
- Input und Output
- Modellwahl
- Kontextgröße
- wiederholte Aufrufe
- Parallelität
- Budgets
- Kosten pro Anwendungsfall messen

Performance

- Latenz
- Time-to-first-token
- Modellgeschwindigkeit
- Netzwerk
- Parallelisierung
- Caching
- kleinere Modelle
- Benutzererwartungen

Caching

- identische und ähnliche Anfragen
- deterministische Zwischenergebnisse
- Modellantworten
- Kontext
- Cache Keys
- Invalidierung
- Datenschutz
- Kosten gegen Aktualität

Logging und Observability

- Request-ID
- Modell
- Laufzeit
- Tokenverbrauch
- Fehler
- Validierungsprobleme
- Workflow-Schritte
- sensible Inhalte nicht unkontrolliert protokollieren

Tracing

- mehrstufige Workflows
- einzelne Modellaufrufe
- externe Systeme
- Tool-Aufrufe
- Latenzen
- Fehlerursachen
- Kosten
- End-to-End-Nachvollziehbarkeit

API für die eigene LLM-Anwendung

- Anwendung als Service
- REST API
- Request-Modelle
- Response-Modelle
- Fehlercodes
- asynchrone Verarbeitung
- Authentifizierung
- klare Trennung von AI- und Web-Layer

Hintergrundverarbeitung

- lange Modellaufrufe
- Jobs
- Queues
- Status
- Retry
- Abbruch
- Ergebnisse speichern
- Benutzer über Fortschritt informieren

Deployment

- Konfiguration
- Umgebungen
- Container
- Secrets
- Skalierung
- Netzwerk
- Health Checks
- Modellprovider als externe Abhängigkeit

CI/CD

- Unit Tests
- Integrationstests
- Evaluation
- Quality Gates
- Packaging
- Container Build
- Deployment
- Modell- und Promptänderungen kontrollieren

Frameworks sinnvoll einsetzen

- direkte Provider-SDKs
- LLM Frameworks
- Abstraktionen
- schnelle Prototypen
- versteckte Komplexität
- Debugging
- Lock-in
- Framework nach Bedarf statt als Architektur

Eine produktionsnahe Architektur entwickeln

- Application Layer
- Model Gateway
- Prompt Layer
- Validation Layer
- Persistence
- externe Systeme
- Observability
- Security
- klare Schnittstellen zwischen probabilistischen und deterministischen Komponenten

Vom Prototyp zur Anwendung

- ersten Modellaufruf entwickeln
- strukturierte Ausgabe
- Validierung
- Fehlerbehandlung
- Tests
- Evaluation
- Observability
- schrittweise Produktionsreife herstellen

Praktische Übungen

- Python-Anwendung mit LLM-API aufsetzen
- Model Client kapseln
- Prompt Templates entwickeln
- strukturierte Ausgaben definieren und validieren
- Fehler- und Retry-Handling implementieren
- Streaming einsetzen
- Conversation State verwalten
- mehrstufigen Workflow entwickeln
- Function Calling integrieren
- externe Komponente kontrolliert anbinden
- Unit Tests ohne Modellaufruf entwickeln
- Evaluationsdatensatz erstellen
- Modell- oder Promptvarianten vergleichen
- Kosten und Laufzeiten messen
- vollständige LLM-Anwendung als API bereitstellen

03 DURCHFÜHRUNG

Rahmen und Organisation

Dauer	3 Tage
Durchführung	Präsenz oder Live Online, vorzugsweise als Inhouse-Schulung
Schwerpunkt	Inhouse-Schulungen für Unternehmen und Teams

Sprache	Deutsch, englische Fachbegriffe und Dokumentation
Unterlagen	Kursunterlagen, Beispiele und Übungsaufgaben
Technik	Eigener Rechner mit Python, Entwicklungsumgebung und Zugang zu einer LLM-API

> **Inhouse-Schulungen**
> Vorhandene Python-Services, APIs, Datenmodelle und typische fachliche Anwendungsfälle können als Grundlage für die Übungen verwendet werden.

04 ZIELGRUPPE

Für wen ist der Kurs gedacht?

Der Kurs richtet sich an Python-Entwickler, Softwareentwickler, AI Engineers und technische Consultants, die Sprachmodelle in eigene Anwendungen, Services und Geschäftsprozesse integrieren möchten.

Er eignet sich insbesondere für Entwickler, die über einzelne API-Aufrufe und experimentelle Skripte hinaus wartbare, testbare und produktionsfähige LLM-Anwendungen entwickeln wollen.

> **Voraussetzungen**
> Sichere Python-Grundkenntnisse und praktische Erfahrung in der Softwareentwicklung werden vorausgesetzt. Die Teilnehmer sollten Funktionen, Klassen, Type Hints, Exceptions, Module und grundlegende automatisierte Tests kennen; Kenntnisse von HTTP und REST APIs sind hilfreich. Grundkenntnisse über generative KI und LLMs sind sinnvoll, RAG-, MCP- oder Agentenkenntnisse werden nicht vorausgesetzt.

05 LERNZIELE

Was Sie nach dem Kurs können

Nach dem Kurs können die Teilnehmer:

- LLMs sauber in Python-Anwendungen integrieren
- Modellzugriffe von der Anwendungslogik entkoppeln
- Prompts und Templates strukturiert verwalten
- strukturierte Modellantworten definieren und validieren
- robuste Fehler- und Retry-Strategien entwickeln
- Streaming und asynchrone Modellaufrufe einsetzen
- Conversation State und Kontext kontrollieren
- mehrstufige LLM-Workflows entwickeln
- Function Calling sicher integrieren
- RAG-, MCP- und Agentenkomponenten über klare Schnittstellen anbinden
- LLM-Ausgaben als nicht vertrauenswürdige Eingaben behandeln
- Prompt-Injection- und Datenschutzrisiken berücksichtigen
- deterministische Komponenten unabhängig vom Modell testen

- LLM-Komponenten systematisch evaluieren
- Regressionen bei Prompt- und Modelländerungen erkennen
- Kosten, Performance und Modellwahl berücksichtigen
- Logging und Tracing integrieren
- LLM-Anwendungen als Services bereitstellen
- eine Python-LLM-Anwendung vom Prototyp zur produktionsnahen Architektur weiterentwickeln

06 FAQ

Häufige Fragen

Was unterscheidet diesen Kurs vom RAG-Kurs?

Der RAG-Kurs konzentriert sich auf Wissenssysteme: Ingestion, Chunking, Embeddings, Retrieval, Ranking, Quellen und Evaluation des Wissenszugriffs.

Dieser Kurs behandelt die umgebende Python-Anwendung. Ein RAG-System kann angebunden werden, seine Entwicklung ist aber nicht Gegenstand dieses Kurses.

Was unterscheidet ihn vom Agentenkurs?

Hier stehen überwiegend explizit gesteuerte Anwendungen und Workflows im Mittelpunkt. Die Python-Anwendung entscheidet, wann ein Modell aufgerufen wird und welche Verarbeitungsschritte folgen.

Der Agentenkurs behandelt Systeme, in denen ein Agent selbst über weitere Arbeitsschritte und Tool-Aufrufe entscheidet. Agentische Autonomie ist deshalb bewusst kein Schwerpunkt dieses Kurses.

Was unterscheidet ihn vom MCP-Entwicklungskurs?

Der MCP-Kurs entwickelt Python-Komponenten, die Fähigkeiten über MCP für KI-Anwendungen bereitstellen. Hier wird die KI-Anwendung selbst entwickelt, die vorhandene MCP-Server als externe Integrationen nutzen kann.

Ist das ein Prompt-Engineering-Kurs?

Nein. Prompts sind Bestandteil jeder LLM-Anwendung und werden entsprechend behandelt. Die systematische Vertiefung von Prompt-Techniken und Prompt-Evaluation ist Gegenstand des Prompt-Engineering-Kurses.

Wird RAG programmiert?

Ein vorhandener Retrieval- oder RAG-Dienst kann in die Anwendung integriert werden. Der Aufbau einer vollständigen RAG-Pipeline gehört in den RAG-Kurs und wird hier bewusst nicht wiederholt.

Werden Agenten programmiert?

Nein. Der Kurs zeigt die Grenze zwischen klassischen LLM-Workflows und Agenten und kann einen vorhandenen Agenten anbinden. Die Entwicklung agentischer Systeme gehört in den Agentenkurs.

Wird MCP behandelt?

Aus Anwendungssicht ja: Ein vorhandener MCP-Server kann als Integrationskomponente verwendet werden. Protokollarchitektur und Serverentwicklung sind Gegenstand der MCP-Kurse.

Wird ein bestimmtes LLM-Framework vorausgesetzt?

Nein. Der Schwerpunkt liegt auf Python- und Software-Engineering-Prinzipien. Provider-SDKs und geeignete Bibliotheken werden praktisch verwendet, die Architektur soll jedoch nicht von einem bestimmten Framework abhängen.

Werden mehrere Modellanbieter behandelt?

Die Anwendung wird so strukturiert, dass Provider-spezifische Details gekapselt werden können. Je nach Kursumgebung können unterschiedliche Modelle verglichen werden. Der Kurs ist keine Produktschulung für einzelne Anbieter.

Werden LLM-Anwendungen wirklich automatisiert getestet?

Ja. Ein wesentlicher Bestandteil des Kurses ist die Trennung zwischen deterministisch testbarer Anwendungslogik und probabilistischem Modellverhalten. Für letzteres werden Evaluation und Regressionstests eingesetzt.

Warum dauert der Kurs drei Tage?

Der eigentliche Modellaufruf ist nur ein kleiner Teil einer LLM-Anwendung. Strukturierte Ausgaben, Validierung, State, Workflows, Fehlerbehandlung, Tests, Evaluation, Security, Kosten und Betrieb müssen gemeinsam praktisch behandelt werden. Zwei Tage würden im Wesentlichen für einen Prototypen-Workshop reichen, nicht für den beschriebenen Engineering-Anspruch.

Kann der Kurs an unsere Python-Anwendung angepasst werden?

Ja. Bei Inhouse-Schulungen können vorhandene Python-Services, APIs, Datenmodelle und typische fachliche Anwendungsfälle als Grundlage für die Übungen verwendet werden.

UC IT Service · Ulrich Cuber · kontakt@uc-it.de
https://www.uc-it.de/coaching/llm-anwendungen-python/