Lokale LLMs und eigene AI-Infrastruktur

Sprachmodelle lokal und in eigener Infrastruktur betreiben

01 INTRO

Cloudbasierte AI-Dienste ermöglichen einen schnellen Einstieg, sind aber nicht für jeden Anwendungsfall die passende Betriebsform. Datenschutz, vertrauliche Informationen, Kosten, Latenz, Verfügbarkeit oder die vollständige Kontrolle über Modelle und Daten können dafür sprechen, Sprachmodelle auf eigener Hardware oder in einer kontrollierten Infrastruktur zu betreiben.

Der Kurs vermittelt den praktischen Aufbau einer eigenen LLM-Infrastruktur. Die Teilnehmer wählen geeignete Modelle und Hardware, betreiben Modelle mit unterschiedlichen Inference-Lösungen, stellen standardisierte APIs bereit und untersuchen Performance, Quantisierung, Speicherbedarf und Parallelität. Darauf aufbauend werden Betrieb, Monitoring, Zugriffsschutz und die Integration lokaler Modelle in bestehende Anwendungen behandelt.

DAUER

3 Tage

FORMAT

Präsenz · Live Online · Inhouse

VORKENNTNISSE

Praktische Erfahrung mit Linux und Softwareentwicklung oder Systemadministration

02 TRAININGSPROGRAMM

Inhalte

Warum lokale LLMs?

- Cloud API und eigener Betrieb
- vertrauliche Daten
- Datenschutz
- Kontrolle über Modelle und Infrastruktur
- Verfügbarkeit
- Latenz
- Kosten
- Offline- und abgeschottete Umgebungen

Was bedeutet lokaler Betrieb?

- Arbeitsplatzrechner
- Entwickler-Workstation
- eigener GPU-Server
- Rechenzentrum
- Private Cloud
- gemietete dedizierte GPU
- vollständig abgeschottete Systeme
- unterschiedliche Betriebsmodelle

Modelle verstehen und auswählen

- Modellfamilien
- Modellgrößen
- Parameter
- Kontextfenster
- Sprachunterstützung
- Coding-Modelle
- multimodale Modelle
- Modellwahl nach Aufgabe

Open-Weight-Modelle

- Modellgewichte
- frei verfügbare Modelle
- unterschiedliche Lizenzen
- Nutzungsbedingungen
- kommerzielle Verwendung
- Modellkarten
- Herkunft
- Lizenzprüfung als Teil der Auswahl

Modellgröße und Ressourcenbedarf

- Parameterzahl
- Speicherbedarf
- Gewichte
- KV Cache
- Kontextgröße
- Runtime Overhead
- mehrere Benutzer
- tatsächlichen Bedarf statt nur Modellgröße betrachten

Quantisierung

- reduzierte numerische Präzision
- Speicherbedarf
- unterschiedliche Quantisierungsstufen
- Performance
- Qualitätsverlust
- Hardwareunterstützung
- geeignete Quantisierung auswählen
- Qualität praktisch vergleichen

Modellformate

- unterschiedliche Modellartefakte
- native Framework-Formate
- quantisierte Formate
- Metadaten
- Tokenizer
- Kompatibilität mit Inference Engines
- Konvertierung
- Format nach Runtime auswählen

CPU oder GPU?

- CPU Inference
- GPU Inference
- Arbeitsspeicher
- VRAM
- Speicherbandbreite
- Latenz
- Durchsatz
- geeignete Hardware nach Workload

GPU-Grundlagen für LLM-Betrieb

- VRAM
- Compute
- Speicherbandbreite
- GPU-Generationen
- Treiber
- Runtime
- mehrere GPUs
- Hardware und Software müssen zusammenpassen

Hardware dimensionieren

- Modellgröße
- Quantisierung
- Kontext
- erwartete Benutzer
- parallele Requests
- Antwortlänge
- Latenzanforderung
- Reserve für reale Last

Workstation oder Server?

- Einzelbenutzer
- Entwicklerteam
- zentraler Dienst
- Dauerbetrieb
- Kühlung
- Stromverbrauch
- Erweiterbarkeit
- Kosten

Inference Engines

- Aufgabe einer Inference Engine
- Modell laden
- Token generieren
- Speicher verwalten
- Batching
- parallele Requests
- APIs
- unterschiedliche Engines nach Einsatzgebiet

Lokaler Einstieg mit Ollama

- Installation
- Modelle beziehen
- Modelle verwalten
- Modell starten
- lokale API
- Konfiguration
- Modellvarianten
- typische Entwicklungsumgebung

llama.cpp und effiziente lokale Inference

- CPU- und GPU-Unterstützung
- quantisierte Modelle
- lokale Ausführung
- Hardware Offloading
- Kontext
- Serverbetrieb
- Performance-Parameter
- Einsatz auf unterschiedlicher Hardware

Serverorientierte Inference

- mehrere Benutzer
- hoher Durchsatz
- kontinuierliches Batching
- Speicherverwaltung
- parallele Requests
- API-Kompatibilität
- serverorientierte Engines
- Auswahl nach Lastprofil

Modelle als API bereitstellen

- lokale HTTP API
- standardisierte Schnittstelle
- Chat Completion
- strukturierte Requests
- Streaming
- Modellparameter
- Fehler
- Anwendungen vom konkreten Backend entkoppeln

API-Kompatibilität

- gemeinsame Client-Schnittstellen
- lokale und externe Provider
- Providerwechsel
- Modellnamen
- unterschiedliche Fähigkeiten
- nicht jede kompatible API verhält sich identisch
- Capability Detection
- Integrationsschicht

Modellkonfiguration

- Temperature
- Sampling
- maximale Ausgabe
- Kontextgröße
- Stop Conditions
- System Instructions
- deterministische Anforderungen
- Konfiguration nach Anwendungsfall

Kontext und Speicherbedarf

- Prompt Tokens
- generierte Tokens
- KV Cache
- große Kontextfenster
- Speicherverbrauch
- mehrere Sessions
- Kontextlänge gegen Performance
- maximal möglich ist nicht automatisch sinnvoll

Performance messen

- Ladezeit
- Time-to-first-token
- Tokens pro Sekunde
- Gesamtlatenz
- Durchsatz
- parallele Requests
- Speicherverbrauch
- Messbedingungen dokumentieren

Benchmarking

- reproduzierbare Eingaben
- Warm-up
- Modell
- Quantisierung
- Kontextgröße
- Hardware
- Parallelität
- Ergebnisse vergleichbar machen

Qualität und Performance vergleichen

- unterschiedliche Modelle
- unterschiedliche Quantisierungen
- kleinere und größere Modelle
- Geschwindigkeit
- Speicherbedarf
- fachliche Qualität
- Evaluationsdatensatz
- Entscheidung auf Messwerte stützen

Parallelität

- mehrere Benutzer
- gleichzeitige Requests
- Batching
- Queueing
- Speicherbedarf
- Latenz
- Durchsatz
- Workload kennen

Multi-GPU

- warum mehrere GPUs?
- Modell passt nicht auf eine GPU
- Modellverteilung
- Parallelisierung
- Kommunikationsaufwand
- Speicher
- Skalierung
- Multi-GPU nicht automatisch schneller

Containerisierung

- Inference Server im Container
- Images
- GPU-Zugriff
- Modelle
- Volumes
- Konfiguration
- Netzwerk
- reproduzierbare Umgebung

Modellverwaltung

- Modelle beziehen
- Versionen
- lokale Ablage
- Speicherbedarf
- Modellartefakte
- Updates
- Rollback
- dokumentierte Modellstände

Modellquellen und Supply Chain

- Herkunft des Modells
- Modellrepository
- Dateien
- ausführbarer Modellcode
- Abhängigkeiten
- Vertrauenswürdigkeit
- Versionen fixieren
- fremde Artefakte kontrolliert übernehmen

Zugriffsschutz

- lokale API ist nicht automatisch ungefährlich
- Netzwerkzugriff
- Authentifizierung
- Autorisierung
- Benutzer
- Services
- Firewall
- Zugriff auf notwendige Systeme begrenzen

Netzwerkarchitektur

- localhost
- internes Netz
- zentraler AI-Service
- Reverse Proxy
- TLS
- externe Erreichbarkeit
- Segmentierung
- Trust Boundaries

Secrets und Credentials

- API-Zugänge
- interne Services
- Tokens
- Konfiguration
- Secret Stores
- Container Secrets
- Logs
- Modell benötigt nicht automatisch Zugriff auf Secrets

Datenschutz

- Prompts
- Dokumente
- Antworten
- Conversation History
- Logs
- Persistenz
- Backups
- lokaler Betrieb löst Datenschutzfragen nicht automatisch

Logging

- Requests
- Modell
- Laufzeiten
- Tokenmengen
- Fehler
- Benutzer
- sensible Inhalte
- Logging-Policy

Monitoring

- CPU
- GPU
- VRAM
- RAM
- Auslastung
- Queue
- Latenzen
- Fehlerraten
- Dienstverfügbarkeit

GPU-Monitoring

- Speicherauslastung
- Rechenlast
- Temperatur
- Leistungsaufnahme
- Throttling
- mehrere Prozesse
- Ressourcenengpässe
- technische Ursachen von Performanceproblemen

Kapazitätsplanung

- Benutzerzahl
- Requests pro Zeit
- Kontextgrößen
- Modellgrößen
- gewünschte Antwortzeiten
- Parallelität
- Reserven
- Wachstum

Kosten des Eigenbetriebs

- Hardware
- GPU-Miete
- Strom
- Kühlung
- Betrieb
- Administration
- Auslastung
- Total Cost of Ownership

Cloud API oder eigenes Modell?

- variable API-Kosten
- Investitionskosten
- Betriebskosten
- Auslastung
- Datenschutz
- Qualität
- Skalierbarkeit
- hybride Strategien

Hybridbetrieb

- lokale Modelle
- externe Modelle
- Routing nach Aufgabe
- vertrauliche Daten lokal
- leistungsfähigere Modelle extern
- Fallback
- Kosten
- einheitliche Anwendungsschnittstelle

Model Routing

- unterschiedliche Aufgaben
- unterschiedliche Modelle
- kleines Modell für einfache Aufgaben
- großes Modell für schwierige Aufgaben
- Coding-Modell
- multimodales Modell
- Regeln
- Routing evaluieren

Lokale Embedding-Modelle

- Embeddings lokal erzeugen
- Modelle
- Hardwarebedarf
- Batch-Verarbeitung
- mehrsprachige Daten
- Performance
- Datenschutz
- Einsatz in RAG-Systemen

Lokale RAG-Infrastruktur

- Inference Server
- Embedding Service
- Vector Store
- Dokumentdaten
- Netzwerk
- Zugriffsschutz
- Komponenten verteilen
- RAG selbst bleibt Gegenstand des RAG-Kurses

Lokale LLMs für AI Agents

- Agent Runtime
- lokaler Model Endpoint
- Tool Calls
- strukturierte Ausgaben
- Kontext
- Modellfähigkeit
- Performance
- Agent Engineering bleibt Gegenstand des Agentenkurses

Lokale Modelle und MCP

- MCP Client und Server
- Modell lokal betreiben
- Tools getrennt vom Modell
- interne Services
- Netzwerkgrenzen
- Berechtigungen
- lokale Infrastruktur als Plattform
- MCP selbst bleibt Gegenstand der MCP-Kurse

Coding mit lokalen Modellen

- Coding-Modelle
- Repository-Kontext
- Code Completion
- Chat
- Coding Agents
- Hardwarebedarf
- Latenz
- Qualität gegen Cloudmodelle evaluieren

Multimodale Modelle lokal betreiben

- Text und Bild
- zusätzliche Modellkomponenten
- VRAM
- Inference
- Eingabegrößen
- Performance
- geeignete Use Cases
- Hardwarebedarf

Fine-Tuning einordnen

- Inference gegenüber Training
- Prompting
- RAG
- Fine-Tuning
- Adapter
- zusätzlicher Hardwarebedarf
- wann Fine-Tuning sinnvoll sein kann
- Training ist nicht Schwerpunkt dieses Kurses

Updates

- neue Modellversion
- neue Runtime
- Treiber
- Container
- Abhängigkeiten
- Qualität neu evaluieren
- Performance neu messen
- kontrollierter Rollout

Rollback

- Modellversionen
- Konfiguration
- Container Images
- reproduzierbare Deployments
- alte Version verfügbar halten
- Qualitätsregression
- Betriebsprobleme
- schnell zurückwechseln können

Hochverfügbarkeit einordnen

- Single Point of Failure
- mehrere Inference Server
- Load Balancing
- Health Checks
- Queue
- Failover
- Kosten
- Verfügbarkeit nach tatsächlicher Anforderung

Backup und Recovery

- Konfiguration
- Modelle
- eigene Adapter
- Datenbanken
- RAG-Indizes
- Logs
- Infrastrukturdefinition
- wiederherstellbare Umgebung

Automatisierter Betrieb

- Infrastructure as Code
- Konfiguration
- Container
- Deployment
- Health Checks
- Monitoring
- Updates
- reproduzierbare Systeme

Eigene AI-Plattform

- Inference Layer
- Model Registry
- API Layer
- Embedding Service
- Knowledge Services
- Agent Services
- Tool Services
- gemeinsame Infrastruktur statt einzelner Insellösungen

Verantwortlichkeiten trennen

- Modellbetrieb
- Anwendung
- Daten
- RAG
- Agent
- Tools
- Security
- Monitoring
- klare Schnittstellen erleichtern Betrieb und Austausch

Vom Experiment zum internen AI-Service

- lokales Modell starten
- API bereitstellen
- Anwendung anbinden
- Performance messen
- Zugriffsschutz
- Monitoring
- Modellverwaltung
- Betriebsprozess definieren

Praktische Übungen

- lokales Modell installieren und ausführen
- unterschiedliche Modelle vergleichen
- Quantisierungsvarianten untersuchen
- VRAM- und RAM-Verbrauch messen
- Inference-Performance benchmarken
- lokalen Model Server bereitstellen
- Python-Anwendung über API anbinden
- Streaming testen
- mehrere parallele Requests untersuchen
- Inference Server containerisieren
- Zugriff auf den Dienst begrenzen
- Monitoring aufbauen
- lokales Embedding-Modell einsetzen
- Modellwechsel über eine gemeinsame Anwendungsschnittstelle durchführen
- geeignete Infrastruktur für verschiedene Lastszenarien dimensionieren
- vollständige Architektur für einen internen AI-Service entwerfen

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 Linux-Umgebung, Docker und ausreichend Arbeits- oder Grafikspeicher für lokale Modelle

Inhouse-Schulungen
Vorhandene Workstations, GPU-Server oder geplante Infrastruktur können berücksichtigt und Modelle sowie Betriebsvarianten anhand der tatsächlichen technischen Rahmenbedingungen untersucht werden.

04 ZIELGRUPPE

Für wen ist der Kurs gedacht?

Der Kurs richtet sich an Softwareentwickler, AI Engineers, DevOps Engineers, Systemadministratoren, Softwarearchitekten und technische Consultants, die Sprachmodelle unabhängig von öffentlichen AI-Diensten betreiben oder eine eigene AI-Infrastruktur aufbauen möchten.

Er eignet sich sowohl für einzelne leistungsfähige Entwickler-Workstations als auch für Teams, die zentrale interne LLM-Dienste für Anwendungen, RAG-Systeme oder AI Agents bereitstellen wollen.

Voraussetzungen
Praktische Erfahrung mit Linux und grundlegendes Verständnis von Software- oder Systemarchitekturen werden vorausgesetzt. Erfahrung mit Docker und Python ist für die praktischen Übungen hilfreich, tiefgehende Kenntnisse von Machine Learning oder GPU-Programmierung sind nicht erforderlich. Grundkenntnisse zu generativer KI und Sprachmodellen sollten vorhanden sein.

05 LERNZIELE

Was Sie nach dem Kurs können

Nach dem Kurs können die Teilnehmer:

- geeignete Einsatzszenarien für lokal betriebene LLMs bestimmen
- Open-Weight-Modelle nach technischen und fachlichen Anforderungen auswählen
- Modellgröße, Quantisierung und Hardwarebedarf einordnen
- CPU-, GPU- und VRAM-Anforderungen abschätzen
- unterschiedliche Inference-Lösungen einsetzen und vergleichen
- Modelle über standardisierte APIs bereitstellen
- lokale AI-Dienste in eigene Anwendungen integrieren
- Performance reproduzierbar messen
- Qualität und Geschwindigkeit unterschiedlicher Modelle vergleichen
- Parallelität und Kapazitätsbedarf beurteilen
- Inference-Server containerisieren
- Modelle und Versionen kontrolliert verwalten
- Zugriffsschutz und Netzwerkgrenzen gestalten
- Datenschutz und Logging beim Eigenbetrieb berücksichtigen
- GPU- und Service-Monitoring aufbauen
- Kosten von Cloud- und Eigenbetrieb vergleichen
- hybride Modellstrategien entwickeln

- lokale Modelle für RAG-, Agenten- und Entwicklungsanwendungen bereitstellen
- eine eigene AI-Infrastruktur vom Experiment zum internen Dienst weiterentwickeln

06 FAQ

Häufige Fragen

Was unterscheidet diesen Kurs von der LLM-Anwendungsentwicklung mit Python?

Der Entwicklungskurs baut die Anwendung, die ein Sprachmodell verwendet. Ob dieses Modell über einen Cloudanbieter oder lokal bereitgestellt wird, ist für die grundlegende Anwendungsarchitektur zweitrangig.

Dieser Kurs behandelt die andere Seite dieser Schnittstelle: Auswahl, Bereitstellung und Betrieb der Modelle und der dafür notwendigen Infrastruktur.

Ist das nur ein Ollama-Kurs?

Nein. Ollama eignet sich gut für bestimmte lokale Einsatzszenarien und kann im Kurs verwendet werden. Der Kurs behandelt aber grundsätzlich unterschiedliche Inference-Ansätze vom Entwicklerrechner bis zum zentralen GPU-Service.

Brauche ich eine NVIDIA-GPU?

Nicht zwingend. Modelle können je nach Größe und Runtime auch auf CPUs oder anderer unterstützter Hardware betrieben werden. Für leistungsfähige Inference größerer Modelle ist eine geeignete GPU jedoch häufig sinnvoll.

Wie viel VRAM brauche ich?

Das hängt insbesondere von Modellgröße, Quantisierung, Kontextlänge, Runtime und Parallelität ab. Gerade diese Zusammenhänge werden im Kurs praktisch untersucht, statt eine pauschale VRAM-Zahl für alle Anwendungen zu nennen.

Werden Modelle trainiert?

Nein. Der Schwerpunkt liegt auf Inference und Infrastruktur. Fine-Tuning wird gegenüber Prompting und RAG eingeordnet, ein eigener Training- oder Fine-Tuning-Workflow ist aber nicht Gegenstand des Kurses.

Wird RAG behandelt?

Nur aus Infrastrukturperspektive. Lokale Embeddings, Vector Stores und die Bereitstellung der benötigten Services werden eingeordnet. Die Entwicklung und Evaluation einer vollständigen RAG-Pipeline ist Gegenstand des RAG-Kurses.

Werden AI Agents behandelt?

Nur als möglicher Verbraucher der Infrastruktur. Der Kurs zeigt, wie lokale Modelle für agentische Anwendungen bereitgestellt werden können. Die Entwicklung der Agenten selbst gehört in den Agentenkurs.

Ist ein lokales Modell automatisch datenschutzkonform?

Nein. Eigener Betrieb kann die Kontrolle über Daten erheblich verbessern, beseitigt aber Anforderungen an Zugriffsschutz, Logging, Persistenz, Berechtigungen und Datenverarbeitung nicht automatisch.

Können lokale Modelle Cloudmodelle vollständig ersetzen?

Das hängt vom Anwendungsfall ab. Qualität, Modellfähigkeiten, Hardwarebedarf, Kosten und Datenschutz müssen gemeinsam betrachtet werden. Für manche Systeme ist vollständiger Eigenbetrieb sinnvoll, für andere ein Cloud- oder Hybridansatz.

Werden mehrere Modelle verglichen?

Ja. Modellgröße, Quantisierung, Ressourcenverbrauch, Geschwindigkeit und Qualität werden anhand reproduzierbarer Aufgaben verglichen.

Wird auch der Betrieb für mehrere Benutzer behandelt?

Ja. Parallelität, Batching, Netzwerkzugriff, Zugriffsschutz, Monitoring und Kapazitätsplanung sind Bestandteile des Kurses.

Warum dauert der Kurs drei Tage?

Der lokale Start eines Modells dauert nur wenige Minuten. Eine belastbare AI-Infrastruktur erfordert dagegen Modell- und Hardwareauswahl, Inference, Performanceanalyse, APIs, Container, Security, Monitoring und Betriebsprozesse. Drei Tage ermöglichen, diese Ebenen praktisch miteinander zu verbinden, ohne daraus eine allgemeine DevOps- oder ML-Operations-Schulung zu machen.

Kann der Kurs an unsere vorhandene Hardware angepasst werden?

Ja. Bei Inhouse-Schulungen können vorhandene Workstations, GPU-Server oder geplante Infrastruktur berücksichtigt und Modelle sowie Betriebsvarianten anhand der tatsächlichen technischen Rahmenbedingungen untersucht werden.

UC IT Service · Ulrich Cuber · kontakt@uc-it.de
https://www.uc-it.de/coaching/lokale-llms/