Ollama in der Praxis

Lokale LLMs für praktische Anwendungen

Workshop - 13. November 2025
Link to the Repo:

Agenda

1. Was ist Ollama?
2. Warum lokale LLMs?
3. Anwendungsbeispiele
    - Ollama Search, OpenRefine
    - Ollama Spielwiese
    - Local Deep Researcher
4. Fazit

Was ist Ollama?

- **Open Source Software** (MIT-Lizenz)
- Ermöglicht das **lokale Ausführen** von Large Language Models
- Einfache Installation und Bedienung
- API-Zugriff für Integrationen
- Unterstützt viele Modelle (Llama, Mistral, Gemma, etc.)

```
ollama pull llama3
ollama run llama3
```

Warum lokale LLMs (1)?

Vorteile:

- **Datenschutz**: Daten bleiben lokal
- **Keine Internetverbindung** nötig
- **Keine Kosten** pro Anfrage
- **Anpassbar**: Eigene Modelle und Konfigurationen
- **Kontrolle**: Volle Kontrolle über das System

Warum lokale LLMs (2)?

Ideal für:

- Vertrauliche Daten
- Prototyping
- Forschung & Lehre

Ollama bietet API-Zugriff (1)

```
curl http://localhost:11434/api/generate -d '{
  "model": "llama3",
  "prompt": "Was ist Ollama?"
}'
```

Ollama bietet API-Zugriff (2)

Das ermöglicht:

- Integration in Anwendungen
- Workflow-Automatisierung
- Kombination mit Tools

Anwendungsbeispiel 1

Ollama Search

Repository: https://github.com/NbtKmy/ollama_search

Konzept:

- Semantische Suche mit Embeddings
- Vektorbasierte Dokumentensuche
- Vollständig lokal & offline

Demo: Ollama Search

```
# Live-Demo
```

Vorteile:

- Präzisere Suchergebnisse
- Versteht Kontext und Synonyme
- Keine externe API nötig

Anwendungsbeispiel 2

OpenRefine Integration

OpenRefine - Tool für Datenbereinigung

Zwei Ansätze:

1. **LLM-Extension** - GUI-basiert
2. **Python-Code** - Flexibel

OpenRefine + Ollama: LLM-Extension

Installation:

1. LLM-Extension herunterladen
2. In OpenRefine Extension-Ordner kopieren
3. Ollama als Endpoint konfigurieren

OpenRefine + Ollama: Python-Code

```
import json, urllib2

data = {
    "messages": [
        {"content": prompt, "role": "system"},
        {"content": value, "role": "user"}
    ],
    "model": "mistral:latest"
}

req = urllib2.Request(url, json.dumps(data))
response = urllib2.urlopen(req)
```

Demo: OpenRefine + Ollama

```
# Live-Demo
```

Use Cases:

- Biografische Daten extrahieren
- Adressen normalisieren
- Kategorien zuordnen

Anwendungsbeispiel 3

Ollama Spielwiese

Repository: https://github.com/NbtKmy/ollama-spielwiese

Features:

- Interaktive Chat-Oberfläche
- RAG (Retrieval-Augmented Generation)
- Dokumenten-Upload & Analyse

RAG - Retrieval-Augmented Generation

Grundkonzept:

1. Dokumente → Vektoren
2. Relevante Passagen abrufen
3. LLM generiert Antwort mit Kontext

Demo: Ollama Spielwiese

```
# Live-Demo
```

Praktische Anwendungen:

- Interne Wissensdatenbank
- Dokumentenanalyse
- Q&A-System für spezifische Domänen

Anwendungsbeispiel 4

Local Deep Researcher

Repository: https://github.com/langchain-ai/local-deep-researcher

Konzept:

- Autonomer Research-Agent
- Vollständig lokal mit Ollama
- Multi-Step Reasoning

Local Deep Researcher: Funktionsweise

Workflow:

1. **Planung** → Research-Schritte
2. **Suche** → Informationen sammeln
3. **Analyse** → Quellen bewerten
4. **Synthese** → Report erstellen

Demo: Local Deep Researcher

```
# Live-Demo
```

Use Cases:

- Marktforschung
- Literaturrecherche
- Competitive Analysis

Vergleich der Anwendungen

Tool	Zweck	Setup
Search	Sem. Suche	Einfach
OpenRefine	Daten	Mittel
Spielwiese	RAG/Chat	Einfach
Researcher	Research	Komplex

Technische Grundlagen (1)

Gemeinsame Basis:

- Ollama API (localhost:11434)
- Embeddings
- 100% offline

Technische Grundlagen (2)

Hardware:

- CPU: 4+ Cores
- RAM: 8-16 GB min.
- GPU: Optional

Quantisierung: Der Schlüssel

Problem: Große Modelle = viel RAM

Quantisierungs-Level:

- Q4_K_M: Balance (Standard)
- Q5_K_M: Bessere Qualität
- Q8_0: Höchste Qualität

Beispiel Llama 7B:

- FP16: ~14 GB → Q4: ~4 GB → Q2: ~2 GB

Modellauswahl

Chat:

- `llama3:8b` - Allzweck
- `mistral:7b` - Schnell
- `gemma:7b` - Google
- `phi3:3.8b` - Kompakt

Tipp: Klein starten, testen, skalieren

Best Practices

1. **Modellgröße** an Hardware anpassen
2. **Klare Prompts** verwenden
3. **Context Length** minimieren
4. **Temperature**: 0.7-0.8 (kreativ), 0.3 (faktisch)
5. **Ressourcen** monitoren

Herausforderungen & Lösungen

Lösungen:

- Begrenzte Power → Quantisierung
- Langsame Inferenz → GPU nutzen
- Qualität → Richtiges Modell wählen

Nützliche Befehle:

- `ollama list` - Verfügbare Modelle
- `ollama show` - Modelldetails

Sicherheit & Datenschutz

Vorteile:

- Daten bleiben lokal
- Kein Tracking durch Dritte
- DSGVO-konform
- Keine Rate Limits

Wichtig:

- Zugriffskontrollen
- Modellquellen verifizieren

Community & Ressourcen

Links:

- https://ollama.com
- https://github.com/ollama/ollama
- https://ollama.com/library

Modelle:

- https://huggingface.co (GGUF)

Praktische Übungen

Zum Ausprobieren:

1. Ollama + Modell installieren
2. API testen (curl/Python)
3. Demo-Projekt ausprobieren
4. Einfache Integration bauen

Fazit

Ideal für:

- Prototyping ohne Kosten
- Datenschutz-kritische Apps
- Lernen & Experimentieren
- Produktive Tools (offline)

Q&A

Fragen?

GitHub: NbtKmy
Repo: /ollamaintroduction

Vielen Dank!

Backup: Technische Details

Architektur:

- Basiert auf llama.cpp
- GGUF/GGML Format
- REST API (Port 11434)

Backup: Installation

macOS:

```
brew install ollama
```

Linux:

```
curl -fsSL https://ollama.com/install.sh | sh
```

Windows:
Download von ollama.com