Universität der Bundeswehr München

Lokale KI - Wenn Dokumente nicht ins Netz dürfen

Universität der Bundeswehr München
Mathias Schlolaut

1. Juli 2026

Was ist der Fokus?

- Lokale Sprachmodelle (LLMs)
- Arbeiten mit Dokumenten
- Minimalbeispiel zum Nachstellen
- Was braucht es?

Anwendungen

Beispiele[1]

- AnythingLLM
- Kotaemon
- Open WebUI
- RAGFlow

[1] Stand Mai 2026, Projekte können kurzlebig sein

Installation – was braucht es?

Installation der Hintergrundanwendungen:

- Ollama (für die Sprachmodelle)
- Docker Compose (für die einfache Installation von AnythingLLM)

Modelle herunterladen und Anwendung starten

Vereinfachte Schritte. Je nach Betriebssystem können weitere Schritte notwendig sein.

- Herunterladen der Sprachmodelle:
  - ollama pull gemma4:latest
  - ollama pull nomic-embed-text:latest
- Starten der Anwendung:
  - Erstellen der Datei docker-compose.yml (siehe Ende der Folien)
  - docker compose up -d

Relevante Bausteine beim Arbeiten mit Dokumenten

- Embedding (Einlesen der Dokumente)
- Modellgröße (Anzahl der Parameter)
- Gedächtnisgröße (Context Window)
- Tool-Aufrufe

Speicherbedarf

	Downloadgröße	mit 132k Gedächtnis (Kontext)
gemma4:12b	9,6 GB	11 GB
gpt-oss:20b	13 GB	17 GB
gemma4:31b	19 GB	37 GB

- Die Modelle gemma4:12b und gpt-oss:20b haben Probleme mit Tool-Aufrufen.
- Mit einem Kontext von 90000 Tokens passt gemma4:31b auf zwei 16 GB GPUs.

AnythingLLM

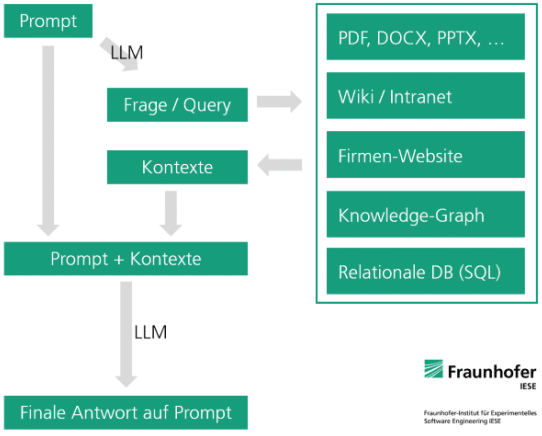
Als Beispiel soll AnythingLLM dienen.

- Einfache Oberfläche ...
- Einfache Einrichtung ...

... im Vergleich zu anderen Anwendungen (subjektiv).

Verwendete Testdatensätze

Datensatz 1	Datensatz 2	Datensatz 3
10 PDFs	50 PDFs	311 PDFs
20 Seiten	100 Seiten	23.500 Seiten



RAG (Retrieval Augmented Generation) – Quelle:
`https://www.iese.fraunhofer.de/blog/retrieval-augmented-generation-rag/`

Live Demo – Wichtige Einstellungen

Model context window

- Wie groß ist das Gedächtnis? Die maximale Größe hängt vom Modell ab.
- Ein zu kleines Gedächtnis vergisst, was es tun soll.
- Ein zu großes Gedächtnis beeinflusst die Rechenzeit negativ.
- Beispielwert: 128000 (Tokens)

Live Demo – Wichtige Einstellungen

Max embedding chunk length

- Zerlegen von Dokumenten/Texten
- Ein zu kleiner Wert führt zur Verteilung von Informationen zwischen den Chunks.
- Ein zu großer Wert führt zu zu vielen Informationen innerhalb eines Chunks.
- Beispielwert: 1000

Es gibt auch embedding-Modelle die für bestimmte Bereiche besser geeignet sind bzw. darauf zugeschnitten sind, wie z.B. Medizin, Programmierung oder bestimmte Sprachen.

Live Demo – Wichtige Einstellungen

Maximale Kontext-Snippets

- Maximale Anzahl an Chunks, die an das LLM geschickt werden.
- Eine zu niedrige Anzahl kann zu Informationsverlust führen.
- Höchstwert in Abhängigkeit von der Größe des LLM-Gedächtnisses (Kontext) wählen.
- Beispielwert: 20

Live Demo – Wichtige Einstellungen

Anzahl an Tool-Aufrufen

- Ein zu kleiner Wert führt zu vorzeitigem Abbruch der Bearbeitung.
- Ein zu hoher Wert kann zu sehr langen Bearbeitungszeiten und Schleifen führen.
- Standardwert von 10 funktioniert (meistens) gut.

Bekannte Probleme und Stolperfallen

- Falsche Kontextgröße bei Ollama, weil Ollama versucht, Kontextgrößen automatisch zu setzen.
- Kommunikation zwischen Anwendung und Ollama funktioniert nicht immer wie gewünscht.
- Falsche Kontextgröße beim Embedding-Modell.
- Eingeschränkte Anzahl an Tool-Aufrufen.
- Das ausgewählte Modell hat Probleme Tools aufzurufen.

Fazit – Erfahrungswerte

- Arbeiten mit wenigen Dokumenten funktioniert sehr gut.
- Bei vielen Dokumenten ist das Prompting entscheidend.
- Wenn Informationen aus vielen Dokumenten verarbeitet werden müssen, ist der Zeitbedarf sehr hoch. Der Aufwand kann bis zu 50-mal so hoch sein wie dann, wenn Informationen nur aus wenigen Dokumenten entnommen werden müssen.
- Wo die genaue Grenze an Dokumenten liegt, kann nicht pauschal festgelegt werden. Hier hilft nur Ausprobieren.
- Mehr Speicher (VRAM) ist von Vorteil, da größere Modelle bessere Ergebnisse erzielen.

Besser, aber komplexer

- Mehrstufiges Retrieval statt einfacher Top-K-Suche
- Hierarchische Chunks mit Parent-Child-Kontext

Theoretisch sollte RagFlow[2] dieses Vorgehen unterstützen. Aufgrund von Komplexität, Problemen und Zeitmangel gehe ich auf diese Annahme heute nicht näher ein.

[2] `https://github.com/infiniflow/ragflow`

Hardware – Empfehlung

Minimum:

- GPU mit 16 GB RAM
  - Ohne dass das System unnötig GPU-Speicher blockiert.
- Alternativ eine CPU mit mindestens 16 Kernen und 32 GB Arbeitsspeicher.

Sonstiges

- Stromkosten (30 bis 50€ bei intensiver Nutzung, Einzelner Nutzer, eine GPU)
- Hitze und Lärm
- Hoher zeitlicher Aufwand bei der Ersteinrichtung

AnythingLLM – docker-compose.yml

```
services:
  anythingllm:
    image: mintplexlabs/anythingllm
    container_name: anythingllm
    ports:
    - "3001:3001"
    cap_add:
      - SYS_ADMIN
    environment:
      - STORAGE_DIR=/app/server/storage
      - JWT_SECRET="aoeiuhdrnspyfgctzaoesridhmwbxcgft"
      - LLM_PROVIDER=ollama
      - OLLAMA_BASE_PATH=http://127.0.0.1:11434
      - OLLAMA_MODEL_PREF=gemma4:latest
      - OLLAMA_MODEL_TOKEN_LIMIT=131072
      - EMBEDDING_ENGINE=ollama
      - EMBEDDING_BASE_PATH=http://127.0.0.1:11434
      - EMBEDDING_MODEL_PREF=nomic-embed-text:latest
      - EMBEDDING_MODEL_MAX_CHUNK_LENGTH=1000
      - VECTOR_DB=lancedb
      - WHISPER_PROVIDER=local
      - TTS_PROVIDER=native
      - PASSWORDMINCHAR=8
    volumes:
      - ./storage:/app/server/storage
    restart: always
```

Mit dem Befehl *ollama ps*, kann, sofern das Modell geladen ist, die eingestellte Kontextgröße überprüft werden. Ja nach Hardwarekonfiguration kann es passieren, dass Ollama einen Wert von 4096 eingestellt hat. Dies ist jedoch in diesem Kontext viel zu wenig.