



GENERATIVE KI in der Praxis

Retrieval-Augmented Generation



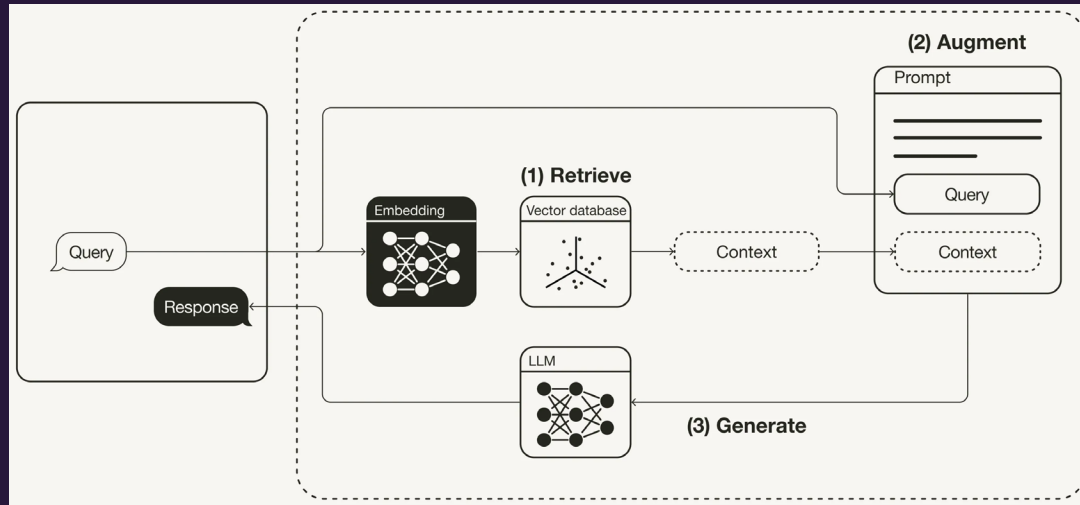
01

Retrieval-Augmented Generation



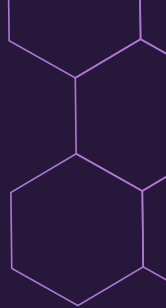
Was ist RAG?

- ◆ Methode zur Verbesserung von LLM-Outputs durch externe Wissensquellen
- ◆ Kombiniert die Stärken von:
 - Abruf relevanter Informationen (Retrieval)
 - Generativer KI (Generation)
- ◆ Ziel: Präzisere und faktenbasierte Antworten



Vorteile durch RAG

- ◆ Probleme klassischer LLMs:
 - Begrenztes / veraltetes Trainings-Wissen
 - Halluzinationen
 - Keine Quellenangaben möglich
- ◆ Vorteile durch RAG:
 - Zugriff auf aktuelle / domänenspezifische Informationen
 - Bessere Faktentreue
 - Nachvollziehbarkeit durch Quellen
 - Kostengünstiger als vollständiges Modell-Training



RAG – Architektur

1. Dokument-Prozessor
2. Embeddings-Komponente
3. Vektor-Datenbank
4. Retrieval-Mechanismus
5. LLM-Komponente

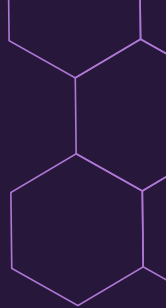


Dokument-Prozessor

- ◆ Preprocessing: Aufbereitung der Dokumente
 - Textbereinigung
 - Entfernung von Metadaten
 - Umwandlung verschiedener Dokumentformate
- ◆ Aufteilung in verwertbare Abschnitte: Chunking-Strategien
 - Semantisches Chunking: basierend auf Satzgrenzen / Absätzen, Kontexterhaltung, rechenintensiv
 - Feste Chunk-Größe (z.B. 200-500 Token): einfach zu implementieren, gleichmäßige Verarbeitung, ggf. Kontextverlust
 - Überlappende Chunks zur Kontexterhaltung

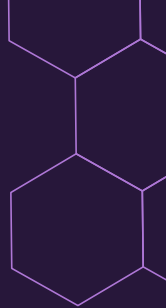
Embeddings-Komponente

- ◆ Zur Umwandlung von Text in Vektorrepräsentationen für semantische Suche
- ◆ Embedding-Modelle
 - Trade-off: Genauigkeit vs. Geschwindigkeit
 - Dimensionalität
 - Domänenanpassung
 - Z.B. Sentence Transformer, OpenAI Embeddings, Contrastive Learning Modelle
- ◆ Prozess
 - Vektorisierung von Textpassagen
 - Dimensionsreduktion
 - Semantische Ähnlichkeitserfassung



Vektor-Datenbank

- ◆ Zur Speicherung der Embeddings für eine effiziente Ähnlichkeitssuche
- ◆ Speichertechnologien:
 - Faiss (Facebook)
 - Pinecone
 - Chroma
 - Milvus
 - Weaviate
- ◆ Funktionen:
 - Skalierbare Speicherung
 - Schnelle Ähnlichkeitssuche
 - Metadaten-Filterung



Retrieval-Mechanismus

◆ Suchmethoden

- Semantic Search:

Suche nach konzeptueller Bedeutung unter Verwendung von Embeddings

- Hybride Suche:

Kombination von Keyword-basierter und semantischer Suche

- Metadaten-basierte Filterung:

Zusätzliche Filterkriterien (Dokumenttyp, Autor, ...) zur Verfeinerung der Suchergebnisse

- Multi-Query Retrieval:

Generierung von mehreren Suchanfragen → Erhöhung der Vollständigkeit der Ergebnisse

◆ Retrieval-Methode

- Similarity Metrics: quantifiziert Ähnlichkeit zwischen Dokumenten / Anfragen

- Anzahl abgerufener Dokumente: Top-k Dokumente

- Re-Ranking Strategien: nachträgliche Neuordnung der Suchergebnisse

LLM-Komponente

- ◆ Generierung der finalen Antwort mit Kontextualisierung der abgerufenen Information
- ◆ Prompt Engineering
 - Kontextuelle Einbettung
 - Quellenangabe
 - Anreicherung mit Retrieval-Kontext



RAG-Workflow – Indexierung

1. Preprocessing:

Bereinigung und Formatierung der Rohdokumente

2. Chunking (Text-Splitting):

Semantische Aufteilung der Dokumente in kleinere Abschnitte (Chunks)

3. Embedding-Generierung:

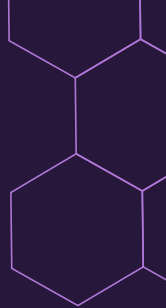
Umwandlung der Chunks in Vektorrepräsentationen

4. Vektor-Indexierung:

Speicherung der Vektoren in einem Index zur schnellen Suche (Vektor-DB)

5. Metadaten-Speicherung:

Zuordnung von zusätzlichen Informationen zu den Vektoren



RAG-Workflow – Abfrage

1. Embedding der Nutzerfrage:

Transformation der Eingabe des Nutzers in eine Vektorrepräsentation

2. Semantische Ähnlichkeitssuche:

Abgleich der Vektoren im Index zur Identifikation relevanter Chunks

3. Chunk-Retrieval:

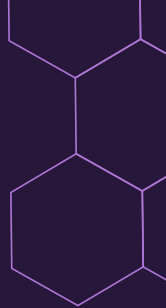
Abruf der relevanten (passenden) Chunks

4. LLM-Prompt zur Kontextanreicherung:

Vorbereitung der Chunks als Input für das Modell & Anreicherung mit Kontext

5. LLM-Generierung:

Erstellung der finalen Antwort



Herausforderungen & Lösungen

- ◆ Herausforderungen
 - Qualität der Quelldokumente
 - Embedding-Genauigkeit
 - Latenz bei großen Dokumentenkorpora
 - Dynamische Wissensaktualisierung
- ◆ Praktische Lösungsansätze
 - Sorgfältige Dokumentenaufbereitung und regelmäßige Dokumentenvalidierung
 - Adaptive Embedding-Strategien
 - Caching-Mechanismen
 - Inkrementelle Indexierung
 - Hybride Retrieval-Ansätze





Evaluierungs- metriken



RAG-Evaluation

Quantitative Metriken

messbare, objektive
Kennzahlen

Qualitative Metriken

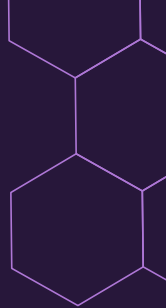
Inhaltliche Bewertung

Automatisierte Metriken

Algorithmus-basierte
Bewertung

Bewertungsdimensionen

- ◆ Quantitative Aspekte
 - Informationsabdeckung
 - Semantische Ähnlichkeit
 - Terminologische Konsistenz
- ◆ Qualitative Aspekte
 - Kontextrelevanz
 - Informationstreue
 - Sprachliche Kohärenz
- ◆ State-of-the-Art Frameworks
 - RAGAS
 - TruLens
 - DeepEval
 - Langsmith



Quantitative Metriken (1)

Retrievalqualität

◆ Recall:

- Anteil relevanter Docs, die gefunden wurden
- Formel: Gefundene relevante Docs / alle relevanten Docs

◆ Precision

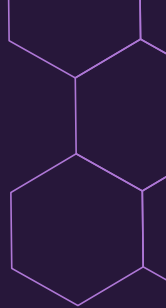
- Anteil relevanter Dokumente unter allen gefundenen
- Formel: Relevante gefundene Docs / alle gefundenen Docs

◆ F1-Score

- Harmonisches Mittel aus Precision und Recall
- Ausgewogene Bewertung der Retrieval-Performance
- Formel: $2 * (Precision * Recall) / (Precision + Recall)$
- Interpretation: 0 (schlechteste Leistung), 1 (perfekte Leistung)

Quantitative Metriken (2)

- ◆ Mean Reciprocal Rank (MRR)
 - Misst Position des ersten relevanten Dokuments
 - Wichtig für Ranking-Qualität



Qualitative Metriken

Generierungsqualität

- ◆ Antwortrelevanz
 - Passt Antwort zur ursprünglichen Anfrage?
 - Menschliche Bewertung / KI-basierte Bewertung
 - ◆ Faktentreue
 - Sind Antworten konsistent mit Quelldokumenten?
 - Halluzinationsgrad messen
 - ◆ Kontextnutzung
 - Wie gut werden retrieve Documents integriert?
 - Explizite Quellenangaben
- LLM-basierte oder menschliche Bewertung möglich

Automatisierte Metriken

- ◆ Embedding-basierte Metriken
 - Semantic Similarity Scores
 - Cosine Similarity (Ähnlichkeit zwischen zwei Vektoren) zwischen:
 - Query und Antwort
 - Dokumenten und Antwort
- ◆ Textbasierte Metriken
 - Contextual Recall
 - Misst Abdeckung von Quelldokument-Informationen
 - Algorithmus vergleicht Terme zwischen Quelle und Antwort
 - Factual Consistency
 - Prüft Übereinstimmung mit Quelldokumenten
 - Verwendet NLP-Techniken zur Inhaltsprüfung (NLP-Metriken: BLEU, ROUGE, Perplexity)



02

RAG-Implementierung

Hilfen für die Entwicklung & Beispiel





KI-Entwicklung

Plattformen, Frameworks und Bibliotheken



HuggingFace

- ◆ Zentrale Bedeutung für Open-Source KI-Modelle
- ◆ "Bibliothek" für KI-Modelle
- ◆ Funktionen:
 - Modell-Repository → Laden von vortrainierten Modellen, Vergleichen von Modellen
 - Einfaches Model Loading
 - Inference APIs
 - Fine-Tuning von Modellen
- ◆ Empfehlung: Grundlegendes Verständnis vermitteln
- ◆ Geeignet für: Modell-Experimente, Custom Training

```
from transformers import pipeline

# Vortrainiertes Modell laden
classifier = pipeline("sentiment-analysis")
result = classifier("Das ist eine tolle Vorlesung!")
```

LangChain

- ◆ Framework zur Entwicklung von LLM-Anwendungen (Abstraktion)
- ◆ Funktionen:
 - Prompt Management
 - Integration verschiedener LLMs
 - Entwicklung komplexerer KI-Workflows
 - Memory-Konzepte
 - Tool/Agent-Frameworks
- ◆ Empfehlung: Für fortgeschrittenere Projekte
- ◆ Geeignet für: Komplexe KI-Workflows, Agents

```
from langchain.llms import OpenAI
from langchain.prompts import PromptTemplate

# Einfache Prompt-Konstruktion
llm = OpenAI()
prompt = PromptTemplate(
    input_variables=["thema"],
    template="Erkläre {thema} wie einem 5-Jährigen"
)
```

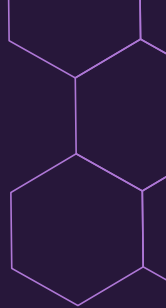

Ollama

- ◆ Lokale Ausführung von LLMs
- ◆ Funktionen:
 - Open-Source Modelle lokal laden
 - Keine Cloud-Abhängigkeit
 - Datenschutz-freundlich
- ◆ Empfehlung: Für Anwendungen unter Beachtung des Datenschutzes, keine Cloud-Dienste erforderlich (Kostenersparnis)
- ◆ Geeignet für: Lokale Entwicklung, Experimente



Lernpfad

- ◆ Grundlagen in Jupyter Notebook
- ◆ HuggingFace zum Modell-Erkunden
- ◆ OpenAI / LangChain für erste Anwendungen
- ◆ Optional: Ollama für lokale Experimente
- ◆ Zusätzliche Empfehlungen für Bibliotheken
 - OpenAI Python Library
 - Anthropic Python Library
 - transformers (HuggingFace)





Beispiel

Assistent für wissenschaftliche Paper



Research-Assistant

- ◆ Beispiel: RAG-basierter Assistent
 - Analyse von wissenschaftlichen Papers
 - Beantwortung von kontextbezogenen Fragen
- ◆ Verwendete Bibliotheken / Frameworks:
 - LangChain: Erstellung von Pipelines für GenAI-Anwendungen (Workflows)
 - LLM & Embeddings:
 - OpenAI-Embeddings + GPT-4
 - Ollama-Embeddings + LLaMA3.2
 - FAISS: Speicherung der Vektor-Daten und effiziente Suche
 - PyPDF: Laden und Extraktion von PDF-Dokumenten
- ◆ Vgl. Jupyter-Notebook in Moodle





03

Laborarbeit

LLM-Anwendung



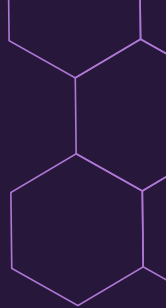
Laborarbeit

- ◆ Aufgabe
 - Entwicklung einer eigenständigen Anwendung
 - Sinnvoller und innovativer Einsatz eines LLMs
 - Konkreter Mehrwert für den Nutzer erkennbar
- ◆ Organisatorisches
 - Gruppenarbeit (ca. 4 Personen)
 - **Präsentation: 10.12.2024 ab 10 Uhr**
 - **Finale Abgabe: 02.02.2025 23:59 Uhr**



Anforderungen

- ◆ Implementierung in Python
- ◆ Nutzung eines LLMs
- ◆ Klare Strukturierung des Codes
- ◆ Readme mit Installationsanleitung und Projektbeschreibung
- ◆ Fehlerbehandlung und Eingabevalidierung
- ◆ Optionale: Einsatz von RAG-Techniken, Custom Prompting
- ◆ Einfache Ausführbarkeit (z.B. mittels Docker Compose)
- ◆ Bedienbarkeit (Benutzeroberfläche)
- ◆ Nachvollziehbare Teamwork (Aufgabenverteilung, Commit-History)



Abgabeumfang

- ◆ Vollständiger Quellcode (GitHub- / GitLab-Repo)
- ◆ Ausführbare, lauffähige und funktionsfähige Anwendung (!)
- ◆ Readme mit
 - Projektbeschreibung
 - Installationsanleitung
 - Beispielnutzung
- ◆ Präsentation
 - Technische Architektur
 - Herausforderungen
 - Lessons Learned
 - Aufgabenverteilung



Bewertungskriterien

- ◆ Technische Umsetzung (50%)
 - Funktionalität
 - Code-Qualität & -Struktur
 - Fehlerbehandlung
- ◆ Konzeptionelle Aspekte (30%)
 - Innovation / Idee
 - Prompt Engineering
 - Dokumentation
- ◆ Präsentation (20%)
 - Projektpräsentation
 - Live-Demo



Ideensammlung


- ◆ Chatbot für eine spezifische Wissensdomäne (z.B. Medizin, Recht, ...)
- ◆ Programmier-Tool (Tutor für eine Programmiersprache, Code-Refactoring-Tool)
- ◆ Testklassifizierungs-Tool (z.B. Spam / Nicht-Spam)
- ◆ Stimmungsanalyse (z.B. Posts aus sozialen Medien)
- ◆ Content-Generator für Marketing (Social-Media-Posts, Blogartikel, ...)
- ◆ Intelligenter Recherche-Assistent (Dokumentensuche mit RAG, Zusammenfassungen)
- ◆ Interaktives Lerntool (Lernkarten-Generator mit Quiz, Lernassistent mit personalisierten Erklärungen)
- ◆ Bewerbungshilfe-Tool
- ◆ Tool zur Datenvisualisierung





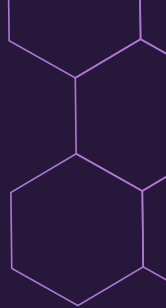
Komponenten

Realisierung einer LLM-Anwendung



Entwicklungsumgebung

- ◆ Programmiersprache: Python
 - Zahlreiche Bibliotheken für Sprachmodelle verfügbar
 - Z.B.: LangChain, Hugging Face, ...
- ◆ IDE: Anaconda mit Jupyter Notebook
 - Einfache Paketverwaltung
 - Interaktive Entwicklung + Dokumentation
 - Integration von Data Science Tools



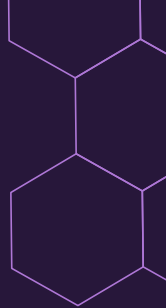
Sprachmodell (LLM)

- ◆ Cloud-basierte Modelle
 - Z.B. GPT-4 (OpenAI), Claude (Anthropic) oder PaLM (Google Bard)
 - Zugang zu Modell-APIs notwendig
- ◆ Lokale Modelle
 - Z.B. LLaMA, Mistral, Falcon, ...
 - Lokale Installation und Verwaltung: Ollama



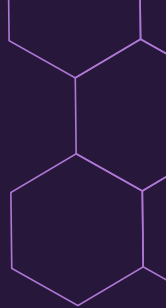
Vektor-DB (RAG)

- ◆ Funktionen:
 - Speichern und Abrufen von Embeddings
 - Schnelles Suchen ähnlicher Daten
- ◆ Beispiele:
 - FAISS
 - Milvus
 - Weaviate
 - Pinecone



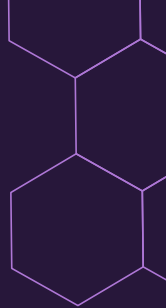
Embeddings-Generator

- ◆ Umwandlung von Text in Vektoren
- ◆ Beispiele
 - Cloud-basiert: OpenAI Embeddings (text-embedding-ada-002)
 - Lokal: SentenceTransformers, Ollama Embeddings



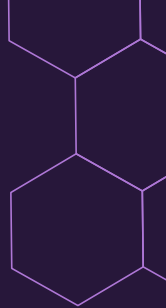
Dokumenten-Verarbeitung

- ◆ Bibliotheken zum Einlesen verschiedenartiger Dokumente
- ◆ Beispiele:
 - PDF: PyPDF2, pdfminer
 - Text: TextLoader
 - Word: DocsLoader
 - HTML: BeautifulSoupHTMLLoader
 - CSV: CSVLoader
 - Text-Splitter-Funktionen zum Chunking von LangChain (z.B. SpacyTextSplitter, TokenTextSplitter, ...)
 - Für Bilder / Scans: OCR (z.B. Tesseract)



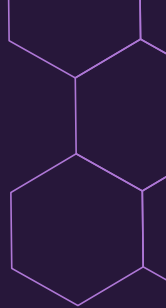
Frontend

- ◆ Benutzeroberfläche zur Bedienung der Anwendung
- ◆ Sehr einfache Möglichkeiten
 - Via Kommandozeile
 - Via Jupyter Notebook (minimalistisches Frontend mittels ipywidgets)
- ◆ More advanced:
 - Web-basiert: Streamlit (einfach), Gradio (schnelle Demos), FastAPI, Flask
 - Desktop: PyQt, Electron.js
 - Mobil: Flutter, React Native



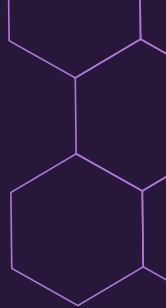
Deployment & Testing

- ◆ Deployment
 - Lokal: Docker, Conda, Hosting auf eigenem Server
 - Cloud: Heroku, AWS, Azure, GCP
- ◆ Testing
 - Unit-Tests
 - Prompt-Evaluierung
 - Feedback (Nutzerfeedback)



Zusatzfeatures

- ◆ Erweiterte NLP-Funktionen (Sentiment-Analyse, Text-Kategorisierung)
- ◆ Daten-Visualisierung (Matplotlib, Plotly)
- ◆ Interaktion: Speech2Text (Whisper), Text2Speech (TTS-Engines)



Empfehlung

- ◆ Einstiegsszenario
 - Streamlit für Frontend
 - OpenAI / HuggingFace für Basis-LLM
 - Optional: LangChain für komplexere Workflows
- ◆ Fortgeschrittenes Szenario
 - FastAPI Backend
 - HuggingFace Modelle
 - LangChain für Advanced Prompting
 - Ollama für lokale Entwicklung



Tipps

- ◆ Praktische Tipps
 - Virtual Environment nutzen
 - Requirements.txt pflegen
 - Docker-Container für Reproduzierbarkeit
 - GitHub für Versionskontrolle
- ◆ Zusätzliche Empfehlungen
 - API Keys sicher speichern (python-dotenv)
 - Logging implementieren
 - Fehlerbehandlung nicht vergessen



To Do

- ◆ Gruppenbildung & Themenfindung → Gruppen & Themen per Mail senden
- ◆ Kickoff: Projektstart

