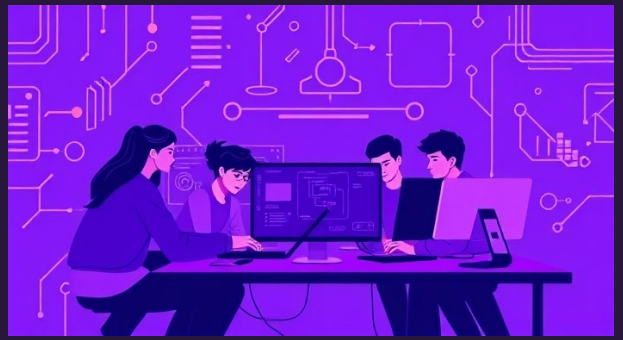
03

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