M20 - KI-Challenge

Anwendung Generativer KI

KI-Challenge

1 | Überblick KI-Challenge

Die KI-Challenge dient als praktische Anwendung und Integration der in den Kursmodulen erlernten Konzepte und Techniken. Ziel ist es, eine funktionsfähige KI-Anwendung zu entwickeln, die mehrere Aspekte der generativen KI kombiniert und einen praktischen Nutzen bietet.

1.1 Lernziele

- Integration mehrerer Technologien aus den Basismodulen
- Praktische Anwendung von LLM-basierten Lösungen
- Entwicklung einer vollständigen End-to-End-Anwendung
- Präsentation und Dokumentation der eigenen Lösung

1.2 Voraussetzungen

- Abschluss der Basismodule (Module 1-12)
- Module aus dem Bereich Erweiterung
- Kenntnisse in Python und LangChain
- Zugriff auf API-Keys (OpenAI, Hugging Face)
- Grundlegende Vertrautheit mit Gradio für UI-Entwicklung

2 | Projektoptionen

Zur Auswahl stehen vier verschiedene Projekttypen, die jeweils unterschiedliche Aspekte der generativen KI betonen. Wählen Sie eine Option aus oder kombinieren Sie Elemente verschiedener Optionen.

2.1 Dokumentenanalyse-Assistent

Beschreibung: Ein System, das PDF-Dokumente, Word-Dateien oder Textdateien verarbeitet und intelligente Zusammenfassungen, Antworten auf Fragen oder strukturierte Analysen liefert.

Kernelemente:

- RAG-Pipeline mit Vektordatenbank (ChromaDB)
- Dokumentenverarbeitung und Chunking
- Intelligentes Prompting für die Analyse
- Benutzeroberfläche mit Gradio

Erwartete Module:

- Modul 4 (LangChain)
- Modul 7 (Output Parser)
- Modul 8 (RAG)
- Modul 11 (Gradio)

2.2 Multimodaler Assistent

Beschreibung: Ein Assistent, der Bild, Text und optional Audio verarbeiten kann, um komplexe Aufgaben zu erfüllen oder Informationen zu analysieren.

Kernelemente:

- Integration von Bild- und Texterkennung
- Multimodale Prompt-Strategien
- Kontextbewusste Antworten
- Interaktive Benutzeroberfläche

Erwartete Module:

- Modul 5 (LLMs und Transformer)
- Modul 6 (Chat und Memory)
- Modul 9 (Multimodal Bild)
- Modul 14 (optional: Multimodal Audio)

2.3 Agentenbasiertes System

Beschreibung: Ein System mit mehreren spezialisierten Agenten, die zusammenarbeiten, um komplexe Aufgaben zu lösen oder Workflow-Prozesse zu automatisieren.

Kernelemente:

- Multi-Agenten-Architektur
- Werkzeugintegration (APIs, Datenbanken)
- Planung und Zielverfolgung
- Benutzerinteraktion und Transparenz

Erwartete Module:

- Modul 3 (Codieren mit GenAI)
- Modul 10 (Agents)
- Modul 12 (Lokale Modelle)
- Modul 18 (optional: Advanced Prompt Engineering)

2.4 Domänen Fachexperte

Beschreibung: Ein spezialisierter Assistent für ein bestimmtes Fachgebiet (z.B. Recht, Medizin, Finanzen, Marketing), der tiefgreifendes Fachwissen bereitstellt und domänenspezifische Aufgaben löst.

Kernelemente:

- Fachspezifische Wissensdatenbank
- Spezialisierte Prompts und Output-Strukturen
- Benutzeroberfläche für Fachexperten
- Optional: Feinabstimmung eines bestehenden Modells

Erwartete Module:

- Modul 2 (Grundlagen Modellansteuerung)
- Modul 8 (RAG)
- Modul 16 (optional: Fine-Tuning)
- Modul 19 (optional: EU AI Act/Ethik)

3 | Projekt-Setup

Hier finden Sie den Code für das grundlegende Setup Ihres Projekts, ähnlich wie in den Kursmodulen.

```
#@title
#@markdown   <p><font size="4" color='green'>  Colab-Umfeld</font> </br></p>
# Installierte Python Version
import sys
print(f"Python Version: ",sys.version)
# Installierte LangChain Bibliotheken
```

```
print()
print("Installierte LangChain Bibliotheken:")

!pip list | grep '^langchain'
# Unterdrückt die "DeprecationWarning" von LangChain für die Memory-
Funktionen
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=UserWarning,
module="langsmith.client")
```

```
#@title
#@markdown   <p><font size="4" color='green'>  SetUp API-Keys
(setup_api_keys)</font> </br></p>
def setup_api_keys():
    """Konfiguriert alle benötigten API-Keys aus Google Colab userdata"""
    from google.colab import userdata
    import os
    from os import environ

    # Dictionary der benötigten API-Keys
    keys = {
        'OPENAI_API_KEY': 'OPENAI_API_KEY',
        'HF_TOKEN': 'HF_TOKEN',
        # Weitere Keys bei Bedarf
    }

    # Keys in Umgebungsvariablen setzen
    for env_var, key_name in keys.items():
        environ[env_var] = userdata.get(key_name)

    return {k: environ[k] for k in keys.keys()}

# Verwendung
all_keys = setup_api_keys()
# Bei Bedarf einzelne Keys direkt zugreifen
# WEATHER_API_KEY = all_keys['WEATHER_API_KEY']
```

4 | Projektstruktur

Ein erfolgreiches Abschlussprojekt sollte folgende Komponenten enthalten:

4.1 Problemdefinition und Anforderungen

- Klare Beschreibung des Problems oder der Aufgabe

- Definition der Anforderungen und Erfolgskriterien
- Abgrenzung des Projektumfangs

4.2 Datenstrukturen und Modellauswahl

- Auswahl und Begründung der verwendeten Modelle
- Datenstrukturen und Datenvorbereitung
- Embedding-Strategien (bei RAG-Anwendungen)

4.3 Kernfunktionalität

- LangChain-Pipelines oder -Ketten
- Prompt-Engineering und Templates
- Integration mit externen APIs oder Datenquellen

4.4 Benutzeroberfläche und Interaktion

- Gradio-Interface für die Interaktion
- Benutzerführung und Feedback
- Fehlerbehandlung und Robustheit

4.5 Evaluation und Tests

- Testfälle für verschiedene Szenarien
- Bewertung der Modellleistung
- Benutzerfeedback und Verbesserungen

4.6 Dokumentation und Präsentation

- Projektdokumentation (Markdown oder PDF)
- Code-Kommentare und Erklärungen
- Präsentation der Ergebnisse

5 | Bewertungskriterien

Die KI-Challenge wird anhand folgender Kriterien bewertet:

Kriterium	Beschreibung	Gewichtung
Funktionalität	Die Anwendung erfüllt die definierten Anforderungen und funktioniert zuverlässig	30%

Kriterium	Beschreibung	Gewichtung
Integration	Erfolgreiche Kombination mehrerer Technologien und Module aus dem Kurs	25%
Code-Qualität	Sauberer, lesbarer und gut strukturierter Code mit angemessenen Kommentaren	15%
Innovation	Kreative Lösungsansätze und eigenständige Weiterentwicklung der Konzepte	15%
Dokumentation	Vollständige und verständliche Dokumentation des Projekts	15%

6 | Beispielprojekt: Doku-Assi

Als Orientierung dient hier ein vereinfachtes Beispiel für einen Dokumentenanalyse-Assistenten:

```
# Import der benötigten Bibliotheken
import os
from langchain.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings import OpenAIEmbeddings
from langchain.vectorstores import Chroma
from langchain.chat_models import ChatOpenAI
from langchain.retrievers.multi_query import MultiQueryRetriever
from langchain.chains import ConversationalRetrievalChain
import gradio as gr

# API-Keys einrichten
os.environ["OPENAI_API_KEY"] = "Ihr-OpenAI-Key"

# Funktion zum Laden und Verarbeiten von Dokumenten
def load_and_process_document(file_path):
    """
    Lädt ein PDF-Dokument und bereitet es für die Verarbeitung vor

    Args:
        file_path: Pfad zur PDF-Datei

    Returns:
        Chroma-Vektordatenbank mit den Dokumentenchunks
    """
    # PDF laden
    loader = PyPDFLoader(file_path)
    pages = loader.load()
```

```
    # Text in Chunks aufteilen
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=1000,
        chunk_overlap=100
    )
    chunks = text_splitter.split_documents(pages)

    # Embeddings erstellen und Vektorstore initialisieren
    embeddings = OpenAIEmbeddings()
    vectorstore = Chroma.from_documents(
        documents=chunks,
        embedding=embeddings
    )

    return vectorstore

# Chat-Modell und Retrieval-Kette initialisieren
def setup_qa_chain(vectorstore):
    """
    Erstellt eine Konversations-Retrieval-Kette für Frage-Antwort-
Interaktionen

    Args:
        vectorstore: Chroma-Vektordatenbank

    Returns:
        ConversationalRetrievalChain für QA
    """
    # LLM initialisieren
    llm = ChatOpenAI(temperature=0, model="gpt-3.5-turbo")

    # Retriever mit Multi-Query-Strategie
    retriever = MultiQueryRetriever.from_llm(
        vectorstore.as_retriever(search_kwargs={"k": 3}),
        llm
    )

    # QA-Kette erstellen
    qa_chain = ConversationalRetrievalChain.from_llm(
        llm=llm,
        retriever=retriever,
        return_source_documents=True
    )

    return qa_chain

# Gradio-Interface für die Benutzerinteraktion
def create_interface():
    """
```

```
    Erstellt ein Gradio-Interface für die Benutzerinteraktion

    Returns:
        Gradio-Interface
    """
    # Zustandsvariablen
    state = {
        "qa_chain": None,
        "chat_history": []
    }

    # PDF-Upload-Funktion
    def upload_pdf(file):
        try:
            vectorstore = load_and_process_document(file.name)
            state["qa_chain"] = setup_qa_chain(vectorstore)
            state["chat_history"] = []
            return "Dokument erfolgreich geladen und verarbeitet!"
        except Exception as e:
            return f"Fehler beim Laden des Dokuments: {str(e)}"

    # Frage-Antwort-Funktion
    def ask_question(question):
        if state["qa_chain"] is None:
            return "Bitte laden Sie zuerst ein Dokument hoch."

        try:
            result = state["qa_chain"](
                {"question": question, "chat_history":
state["chat_history"]}
            )

            # Chat-Historie aktualisieren
            state["chat_history"].append((question, result["answer"]))

            # Quellen hinzufügen
            sources = set()
            for doc in result["source_documents"]:
                page_content = doc.page_content[:150] + "..." if
len(doc.page_content) > 150 else doc.page_content
                sources.add(f"Quelle (Seite {doc.metadata.get('page',
'N/A')}): {page_content}")

            sources_text = "\n\n".join(sources)
            full_response = f"{result['answer']}\n\n---\n\nVerwendete
Quellen:\n{sources_text}"

            return full_response
        except Exception as e:
            return f"Fehler bei der Verarbeitung der Frage: {str(e)}"
```

```
    # Gradio-Interface erstellen
    with gr.Blocks(title="Dokumentenanalyse-Assistent") as interface:
        gr.Markdown("# 📚 Dokumentenanalyse-Assistent")
        gr.Markdown("Laden Sie ein PDF-Dokument hoch und stellen Sie Fragen
dazu.")

        with gr.Row():
            with gr.Column():
                file_input = gr.File(label="PDF-Dokument hochladen")
                upload_button = gr.Button("Dokument verarbeiten")
                status_text = gr.Textbox(label="Status", interactive=False)

            with gr.Column():
                question_input = gr.Textbox(label="Ihre Frage zum Dokument",
placeholder="Stellen Sie eine Frage zum Inhalt des Dokuments...")
                answer_output = gr.Textbox(label="Antwort",
interactive=False, lines=15)
                ask_button = gr.Button("Frage stellen")

        # Ereignisbehandlung
        upload_button.click(upload_pdf, inputs=[file_input], outputs=
[status_text])
        ask_button.click(ask_question, inputs=[question_input], outputs=
[answer_output])

    return interface

# Hauptfunktion
def main():
    interface = create_interface()
    interface.launch(share=True)

# Ausführung
if __name__ == "__main__":
    main()
```

7 | Ressourcen und Hilfestellung

Folgende Ressourcen können bei der Entwicklung des Abschlussprojekts hilfreich sein:

- **Dokumentation:**
  - LangChain Dokumentation
  - OpenAI API Dokumentation
  - Hugging Face Dokumentation

    - Gradio Dokumentation
- **Beispielprojekte und Tutorials:**
    - LangChain Cookbook im GitHub-Repository
    - Beispiel-Implementierungen aus den Kursmodulen
    - Hugging Face Spaces für Beispielanwendungen
- **Online-Tools:**
    - GenAI Tutor
    - ChatBots, wie ChatGPT, Gemini, ...
    - ...

Bei Fragen oder Problemen während der Projektentwicklung können Sie das `Kurs-Forum` nutzen.

 Viel Erfolg!