Natural Language Processing - StudentChatbot

Projektreport

Lars Kurschilgen[1], *Nicholas* Link[1], *Alexander* Paul[1], *Adrian* Setz[1], *Lucas* Wätzig[1], and *Jan* Wolter[1]

[1] *Duale Hochschule Baden-Württemberg Mannheim*

Abstract. Die Arbeit befasst sich mit der Entwicklung eines Chatbots für PDF-Dateien basierend auf Techniken des Natural Language Processing (NLP). Sie behandelt Themen wie das Verarbeiten von Dateien, dem Extrahieren von Texten und Bildern, sowie das Aufteilen des Textes in sogenannte 'chunks'. Außerdem werden verschiedene Praktiken des NLP, beispielsweise das 'Embedden' (vektorisieren) von Wörtern bzw. Sätzen, das Speichern dieser in sogenannten Vektordatenbanken, sowie das Nutzen von Sprachmodellen, beleuchtet. In einem Evaluationsverfahren wurden drei verschiedene Sprachmodelle auf ihre Eignung anhand drei ausgewählter Kriterien bewertet und mit dem etablierten GPT-4 verglichen. Ziel der Arbeit ist neben dem erfolgreichen Verarbeiten bereitgestellter Texte und dem Generieren von Antworten, auch eine nutzerfreundliche Oberfläche mit Hilfe von *streamlit* zu erstellen.

1 Einleitung

Ein Chatbot bietet die Möglichkeit zur Interaktion und Unterstützung von Personen. Das Projekt hat das Ziel, Studierende bei der Nachbereitung von Vorlesungen und der Vorbereitung auf Klausuren zu unterstützen. Es soll eine interaktive Plattform entstehen, welche es ermöglicht, Fragen zu ihren Lernmaterialien zu stellen und personalisierte Unterstützung zu erhalten. Das Projekt basiert auf der Nutzung von Natural Language Processing (NLP). Die Architektur umfasst ein Backend, zuständig für die Verarbeitung von Dateien sowie die Implementierung der Antwort-Logik. Zugänglich gemacht werden soll das Projekt mittels einer benutzerfreundlichen und interaktiven Webapplikation.

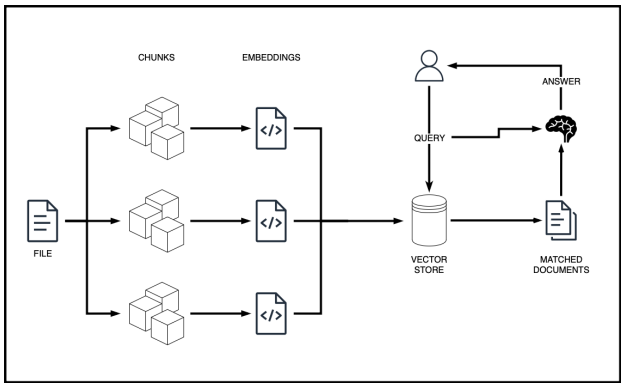

Figure 1. Schematischer Aufbau des Chatbots

2 Verwandte Arbeiten

Die folgenden zwei Projekte beschreiben Ansätze zur Erstellung eines Chatbots. Arbeit [1] befasst sich mit dem Entwurf eines FAQ-Chatbots der basierend auf einem Datensatz auf alle Fragen eine effiziente und genaue Antwort geben kann. [2] beschäftigt sich mit einer Chatbot-Oberfläche für eine Hochschulwebsite unter der Verwendung von KI und NLP.

3 Umsetzung

Für die Implementierung des Student-Chatbots, wurde eine Reihe von Prozessen durchlaufen, angefangen von der Verarbeitung hochgeladener PDFs bis hin zur Integration geeigneter Large Language Models (LLM). Abbildung 1 stellt den Ablauf schematisch dar.

Der Prozess startet mit der automatisierten Extraktion des Dateiinhalts unmittelbar nach dem Upload. Anschließend wird der aus der PDF extrahierte Textblock in kleinere Teile, sogenannte Chunks, unterteilt. Nach diesem Schritt werden die einzelnen Chunks in Vektoren transformiert ('embedded') und in einer Vektordatenbank gespeichert. Die Datenbank dient später dazu die Relevanz von Chunks für die Beantwortung einer Frage zu ermitteln. Bei einer Eingabe des Nutzers wird dessen Anfrage auf die gleiche Weise wie die PDFs 'embedded', um anschließend die Chunks aus der Datenbank inhaltlich mit der Nutzeranfrage vergleichen zu können. Die drei inhaltlich relevantesten Chunks werden anschließend gemeinsam mit der Eingabe des Nutzers an ein LLM übergeben. Dieses generiert daraufhin eine Antwort basierend auf den gegebenen Informationen. Dadurch wird gewährleistet, dass der Nutzer eine sauber formulierte Antwort, die fachlich den Informationen des bereitgestellten Dokuments entspricht, erhält. Im nachfolgenden werden die individuellen Komponenten der einzelnen Prozessschritte genauer betrachtet.

3.1 Text Extraktion

Bei RAG (Retrieval Augmented Generation)–Anwendungen wie der unseren, kommt das '*garbage in – garbage out*' Prinzip besonders zum Tragen. Das bedeutet, dass bessere Antworten unseres Modells davon abhängig sind, wie gut die Informationen aus den gegebenen PDFs extrahiert wurden. Die gängigste Art der PDF-Extraktion in Python sind Bibliotheken wie PyPDF2 oder PyMuPDF, die anhand der Struktur des Quelltextes der Datei den Text extrahieren. Jedoch enthält das Format beispielsweise keine Hinweise auf Strukturen von Texten oder Formatierungen, sodass diese Informationen und zusätzlicher Kontext wie beispielsweise durch Tabellen dabei verloren geht (vgl. [3]). Ein Lösungsansatz für dieses Problem stammt aus dem Bereich der Optical Character Recognition (OCR). Statt der Extraktion aus dem Code der Datei selbst wird hier der Inhalt optisch erkannt (vgl. [4]). Für unsere Anwendung bietet dieser Ansatz ebenfalls Vorteile, da Vorlesungsfolien beispielsweise Kopfzeilen, Fußnoten oder andere wiederkehrende Inhalte aufzeigen, die den Datensatz mit unnötigen Informationen verunreinigen würden. Bilder und Tabellen können zu dem weitere Informationen beinhalten. Außerdem gibt es auch Bilder oder Tabellen, die durchaus mehr Informationen geben können als reiner extrahierter Text. Um eine eigene Lösung dafür zu entwickeln, wurde sich an der Python Bibliothek LayoutParser orientiert und ein eigenes OCR-Modell entwickelt. Genauere Informationen und Beschreibungen finden sich in A.1.

3.2 Text Verarbeitung

Zur effizienten Verarbeitung des Textes und aufgrund der Token-Beschränkung einiger LLMs, muss dieser in kleine Teile unterteilt werden. Der Ansatz des Text-Chunkings erlaubt es, Texte oder Sätze auf verschiedenen Bedeutungsebenen zu betrachten, abhängig von der Betrachtung eines einzelnen Satzes oder eines gesamten Absatzes. Dies beeinflusst die Aufmerksamkeit und wie auf Fragen geantwortet wird. (vgl. [5]). Gängige Methoden sind das Teilen an zum Beispiel Satzzeichen, nach Umbrüchen oder nach einer bestimmten Anzahl von Zeichen. Während des Entwicklungsprozesses hat sich herausgestellt, dass sich diese bewährten Ansätze für das Verarbeiten von Vorlesungsfolien nicht eignen, da diese meist Stichpunkte oder Wortgruppen anstelle von Sätzen beinhalten. Daher wurde ein alternativer Ansatz implementiert, der die Semantik des Textes mit einbezieht. Eine genaue Beschreibung dafür befindet sich im Anhang A.2.
Ein weiter Punkt der Verarbeitung ist das Erstellen von sogenannten 'Embeddings'. Dabei werden Wörter, Sätze oder ganze Textabschnitte in Vektoren umgewandelt, um den Kontext des Textes für Maschinen darzustellen (vgl. [6]; [7]). Diese Vektoren werden anschließend in einer sogenannten Vektordatenbank abgespeichert und bilden das 'Gedächtnis' des Chatbots, auf welches es bei Fragen zugreift.

3.3 Texte identifizieren und auswählen

Nach dem Bereitstellen einer PDF-Datei durch den Nutzer, finden die in den vorangegangenen Kapiteln beschriebenen Prozesse der Vorverarbeitung statt. Im Anschluss daran ist es möglich, Fragen zu dem Dokument zu stellen. Der Prozess im Hintergrund generiert in Kombination mit einem Large Language Model basierend darauf eine Antwort.
Dafür werden passenden Passagen bzw. Chunks aus dem bereitgestellten Text benötigt. Um diese in der Vektordatenbank zu finden, ist eine Alternative zur herkömmlichen Keyword-basierten Suche notwendig. Viele dieser Datenbanken haben die Möglichkeit einer sogenannten Similarity Search bereits implementiert. Im Gegensatz zur Keyword-basierten Suche, benötigt dieser Ansatz keine exakten im Text enthaltenen Schlüsselwörter, sondern verwendet Vektoren. Zur Bestimmung der Ergebnisse kommen verschiedene Metriken und Methoden, wie die Kosinus-Ähnlichkeit, Manhattan oder Euklidische Distanz oder auch 'k-Nearest-Neighbour' (kNN) Bestimmung, zum Einsatz (vgl. [8]). Dies ermöglicht die Abfrage einer Vektordatenbank mittels natürlicher Sprache. Um nach der Eingabe des Nutzers also die passenden Chunks aus der Datenbank zu erhalten, muss diese ebenfalls in einen Vektor 'embedded' werden. Anschließend werden unterstützt durch die Similarity Search die drei thematisch ähnlichsten Textchunks ausgegeben. Mithilfe dieses Wissens und der Eingabe des Nutzers generiert das Modell eine Antwort, welche anschließend an den Nutzer zurückgegeben wird.

3.4 Funktion der Nutzeroberfläche

Die Entwicklung der Benutzeroberfläche (UI) für den PDF-Chatbot erfolgt mithilfe von *streamlit*, einem einfachen und intuitiven Framework für die schnelle und verlässliche Entwicklung von Prototypen oder vollständigen Webanwendungen.
Die Benutzeroberfläche besteht aus mehreren Schlüsselelementen, die den reibungslosen Ablauf der Interaktion mit dem Chatbot gewährleisten. Als zentrales Element bietet die Webanwendung die Möglichkeit, Nachrichten zu schreiben und in Echtzeit Antworten zu erhalten. Die Nachrichten werden direkt nach dem Senden angezeigt, sodass der Nutzer einen Überblick über vergangene Nachrichten hat. Neben dieser Komponente bietet ein Sidebarelement die Möglichkeit, PDF-Dateien hochzuladen und den Prozess zur Verarbeitung und Analyse anzustoßen. Zusätzlich dazu sind Funktionen, wie das Löschen der Chathistorie, das Darstellen des Verarbeitungs- und Analyseprozesses und die Simulation des Schreibprozesses des Chatbots implementiert, um dem Nutzer eine visuell ansprechende und natürliche Interaktion zu vermitteln.

4 Modelle

Die Entwicklung eines Chatbots erfordert eine sorgfältige Auswahl und Evaluierung von Modellen, die den spezifischen Anforderungen des Projektes entsprechen. Im Rahmen der Entwicklung wurden drei bedeutende Ansätze betrachtet: Text-Generierung, Text-zu-Text-Generierung und Frage-Antwort-Systeme.

4.1 Modellarten und Repräsentanten

Text-Generierung oder auch *Casual Language Modelling* ist eine Methode, die auf der Modellierung allgemeiner Sprachstrukturen basiert, wobei der Fokus auf der Erzeugung von informellen und menschenähnlichen Schriftstücken liegt. Modelle dieser Art besitzen das charakteristische Merkmal der sogenannten ‚Decoder Only'-Architektur, wodurch sie in der Lage sind, auf vorhandene Informationen zu reagieren und diese weiterzuverarbeiten oder zu vervollständigen. Die Methode findet insbesondere Anwendung in Szenarien wie der Satzvervollständigung oder der Generierung von beispielsweise der nächsten Gedichtzeile (vgl. [9]). Als beispielhaftes Modell dieser Art wurde das 'Falcon-7B-instruct' eingebunden. Das Modell gehört der 'Falcon'-Klasse an, besitzt demnach eine typische 'Decoder-only' Architektur und ist optimiert auf einem Chat- und Instruct-Datensatz. Zusätzlich dazu wurde die sogenannte *inference time*, also die Zeit, die das Modell benötigt, um Vorhersagen oder generierten Text auf neuen Daten zu erstellen, optimiert. Die dadurch gesteigerte Performance und Effizienz des Modells übersteigt die von ChatGPT-3 und benötigt nur ein Fünftel der *inference time*. Die einzige Begrenzung des Modells betrifft die Sprache. Das 'Falcon-7B-instruct' ist ausschließlich für englischsprachige Texte geeignet (vgl. [10]; [11]).

Text-zu-Text Generation oder auch *Sequence-to-Sequence Modelling* ist ein strukturierter Ansatz der Textgenerierung. Die Methode beruht darauf Textfragmente von einer Sequenz in eine andere zu überführen. Die Basis ist eine ‚Encoder-Decoder' Architektur, bei der der Encoder relevante Informationen der Eingabesequenz extrahiert, während der Decoder diese Informationen nutzt, um die gewünschte Textsequenz zu generieren. Text-zu-Text-Generierung wird vorwiegend für die Übersetzung oder Textzusammenfassung genutzt (vgl. [12]). Als Repräsentant dieser Kategorie wurde sich für ein von Google entwickeltes FLAN (Fine-tuned Language Net) - T5 (Text-to-Text Transfer Transformer) entschieden. Das FLAN-T5 Modell ermöglicht es verschiedene Arten von Sprachverarbeitungsaufgaben in ein einheitliches Text-zu-Text Schema zu überführen. Das Modell besitzt eine typische 'Encoder-Decoder' Architektur und wurde während des Trainingsprozess mit einem umfangreichen Textdatensatz versorgt und darauf trainiert, fehlende Wörter in einem Lückentext vorherzusagen. Dieser Prozess wurde iterativ durchgeführt, bis das Modell die Fähigkeit Texte zu generieren erlangte. Dadurch ist es in der Lage verschiedene Aufgaben, wie das Zusammenfassen oder Klassifizieren von Texten, die Sentiment Analysis, die Übersetzung oder Generierung von Texten zu lösen. In der Modellklasse der T5 hat sich das FLAN eben gerade dadurch etabliert und sich für den Einsatz in dem PDF-Chatbot qualifiziert (vgl. [13]; [14]; [15]).

Frage-Antwort-Systeme (*Question Answer Systems*) sind eine der leistungsfähigsten Anwendungen, die mit Hilfe von LLMs ermöglicht werden. Systeme dieser Art sind auf umfangreichen Textdaten trainiert und besitzen tiefgründiges Verständnis der Sprachstruktur. Der Trainingsprozess eines Modells erfolgt in zwei Schritten: Initial wird versucht das nächste Wort in einem gegebenen Kontext vorherzusagen. Im Anschluss erfolgt die Spezialisierung auf konkrete Frage-Antwort-Aufgaben. Dadurch entwickelt das Modell die Fähigkeit, semantische Strukturen in Fragen zu erfassen und kontextabhängige Antworten zu generieren (vgl. [16]; [17]). Unterschieden werden diese Systeme in drei Kategorien: Extraktive-, Offen-Generative und Geschlossen-Generative-Systeme. Extraktive Modelle benötigen neben der Frage dazu passenden Kontext, aus dem die Antwort entnommen werden kann. Offen-Generative und Geschlossen-Generative Systeme hingegen generieren Text basierend auf bzw. ohne Kontext (vgl. [18]). Als Modell dieser Klasse wurde das 'distilBERT-base-uncased-distilled-squad' gewählt. Im direkten Vergleich zu dem Basismodell 'BERT' (Bidirectional Encoder Representations from Transformers) ist es um circa 40% kleiner bei annähernd gleichbleibendem Sprachverständnis. Außerdem ist die *inference time* 60% geringer. Diese Merkmale sind entscheidend für Chatbots. Die Effizienz des 'distilBERT' resultiert aus dem Prozess der Wissensdestillation während der Trainigsphase. Dabei wurde ein kleines Modell, auch als Schüler bezeichnet, darauf trainiert das Verhalten des Größeren, dem Lehrer, nachzuahmen. Dies ermöglicht es dem Modell, gute Leistungen auf einer Vielzahl von Aufgaben zu erreichen, während die Flexibilität größerer Modelle erhalten bleibt (vgl. [19]; [20, S. 2]). Um zusätzlich das Leseverständnis und Hintergrundwissen des Modells zu verbessern wurde das 'distilBERT' auf dem *SQuAD* (Stanford Question Answering Dataset) feinabgestimmt. Dieser enthält Passagen aus Wikipedia-Artikeln sowie annotierte Listen mit Fragen und Antworten. Das Modell soll mit Hilfe dieser Liste und unter Vorenthaltung der Antworten die relevanten Informationen in den Textpassagen finden und korrekt auf die gestellte Frage antworten (vgl. [21]).

4.2 Evaluierung der Modelle

Nach erfolgreicher Implementierung und Testung aller drei beschriebenen Modelle gilt es die Leistungen zu evaluieren. Zu diesem Zweck wurde ein Foliensatz zum Thema Mikroökonomie von MITopencourseware (vgl. [22]) heruntergeladen. Diese PDF wurde anschließend an die drei ausgewählten Modelle und OpenAIs GPT-4 als Datengrundlage übergeben. Darauf hin wurden zehn verschiedene Fragen zum Inhalt der PDF gestellt. Bei sechs Fragen handelte es sich um einfache Fragen, welche Definitionen als Antwort erforderten, während vier Fragen detailliertere Antworten erwarteten und ein „Verständnis" der Frage voraussetzten. Um sicherzustellen, dass GPT4 lediglich den bereitgestellten Foliensatz als Datengrundlage nutzt, wurde der Zusatz '*Refer exclusively to information from the file*' an die Fragen angehängt. Zur Überwachung des Evaluierungsprozesses wurden alle Antworten der vier Modelle in eine Excel-Datei gespeichert und anhand von drei Bewertungskriterien auf einer Skala von 0 bis 10 bewertet. Diese gliederten sich in inhaltliche Richtigkeit, Spezifität sowie Klarheit und Verständlichkeit. Richtigkeit definiert sich als die richtige Wiedergabe der Informationen aus der bereitgestellten Datei. Die Spezifität beschreibt hingegen den Detailgrad der Antwort bei gleichzeitiger Minimierung von Irrelevantem. Klarheit und Verständlichkeit beschreibt, wie gut die Antwort nachvollzogen werden kann. Die vergebenen Punktzahlen der jeweiligen Kategorien wurden kumuliert

und der Durchschnitt wurde gebildet.
Den höchsten durchschnittlichen Punktestand über alle drei Bewertungskriterien hat GPT-4 mit 8.98, gefolgt von dem FLAN-T5 mit 4.93 und und dem Falcon-7B mit 4.1 Punkten. Am schlechtesten schnitt 'distilBERT' mit einer Punktzahl von 1.67 ab. Dabei ist es hier jedoch wichtig zwischen einfachen Definitionsfragen (Fragekategorie 1) und Fragen, die ein tiefgreifendes Verständnis erfordern (Fragekategorie 2) zu differenzieren. Während für erstere die Performance über alle Modelle hinweg, verglichen mit GPT-4 deutlich ähnlicher ist, kristallisieren sich für Fragekategorie 2 deutliche Unterschiede heraus. Dabei ist ein Abfall von 0,72 Punkten für GPT-4 von Kategorie 1 zu Kategorie 2 zu erkennen, wobei über die restlichen 3 Modelle sogar ein gemittelter Abfall von 2.38 Punkten zu beobachten ist. Den stärksten Abfall hat jedoch das Falcon-7B. Dieses verzeichnet für Fragekategorie 1 einen durchschnittlichen Punktestand von 6.67, für Fragekategorie 2 allerdings nur 0.25 Punkte. Damit schneidet es, in Bezug auf Fragekategorie 2, jedoch immer noch besser ab als 'distilBERT', welches für einen Großteil der Fragen aus Fragekategorie 2 keine Antworten geben konnte und daher für tiefgreifende Fragen durchschnittlich nur 0 Punkte erzielte. Die zustande gekommenen Ergebnisse liegen in einer Vielzahl an Faktoren begründet. Die Fähigkeit von GPT-4, komplexe Fragen besser zu beantworten, liegt darin, dass es trotz der Anweisung sich nur auf die Dateie zu beziehen, sein umfangreiches Wissen nutzt, um Informationslücken zu füllen. Dies können die ausgewählten Modelle nicht, weshalb die Performance deutlich hinter der von GPT-4 liegt. So sind das Falcon-7B und das FLAN-T5 reine Textgenerierungsmodelle, die stark auf den gegebenen Kontext der PDF angewiesen sind. Da 'distilBERT' hingegen einem extraktiven Ansatz folgt, ist es bei der Bewältigung komplexer Fragen aus Fragekategorie 2 noch weniger effektiv. Für einfache Fragen ist in dieser Hinsicht zwar eine Verbesserung zu erkennen, jedoch zeigt sich, dass auch hier der extraktive Ansatz schlechtere Ergebnisse zeigt als der der Textgenerierung. Ein weiterer interessanter Aspekt ist der Umgang mit der Datei bei GPT-4. Die Unklarheit darüber, wie GPT-4 mit der Datei umgeht, weist auf mögliche Unterschiede im Chunking-Verfahren und der Verwendung von Embeddings hin. Diese Faktoren könnten zu einer verbesserten Leistung und einem tieferen Verständnis der gestellten Fragen beitragen, was grundsätzlich jedoch schwer zu beurteilen ist.

5 Fazit

Der Chatbot bietet dem Nutzer die Möglichkeit PDF-Dateien hochzuladen und Fragen an diesen zu stellen. Die PDF-Dateien können über ein benutzerfreundliches User Interface in Form einer Webapplikation hochgeladen werden. Die Dateien werden dann verarbeitet, um Informationen zu extrahieren. Die Integration von Large Language Models (LLM) stellt sicher, dass der Chatbot nicht nur statische Informationen liefert, sondern auch interaktive Dialoge führen kann. Dies ermöglicht eine personalisierte Unterstützung, indem der Chatbot auf individuelle Fragen der Studierenden reagiert und präzise Antworten bereitstellt. Bei der Umsetzung des Projektes traten verschiedene Herausforderungen auf, die eine ganzheitliche Entwicklung beeinflussten. Ein bedeutendes Problem stellte der Datenschutz von Vorlesungsfolien dar. Die Unterscheidung zwischen öffentlichen und geschützten Inhalten erforderte eine sorgfältige Handhabung, um das Urheberrecht zu wahren und gleichzeitig die Funktionalität des Chatbots zu gewährleisten. Um diese Herausforderung zu bewältigen, kann man öffentliche Vorlesungsfolien verwenden oder das Modell lokal ausführen anstelle der Einbindung über API. Die Handhabung von PDF-Dateien, erwies sich allgemein als anspruchsvoll, da unterschiedliche Layouts und mangelnde einheitliche Formatierung die Extraktion von Informationen beeinträchtigten. Damit einhergehend stellte auch die Wahl der optimalen Chunk-Größe eine Herausforderung dar. Ein weiterer Aspekt war die Entscheidung zwischen verschiedenen Umsetzungsmöglichkeiten, wie der Integration von APIs oder der lokalen Einbindung der Modelle. Die vielseitigen Optionen erforderten eine Abwägung, was umsetzbar ist und was nicht. Die Entscheidung fiel am Ende auf die Nutzung der Huggingface-API aufgrund von Performance und Speicherplatz. Wichtige Erkenntnisse aus dem Projekt umfassten die Bedeutung struktureller Planung von Update-Terminen, um eine klare Kommunikation im Team zu ermöglichen. Die Aufgabenverteilung erwies sich als entscheidend, basierend auf Erfahrungen aus früheren Semestern, um im zeitlichen Rahmen zu bleiben und doppelte oder überflüssige Arbeit zu vermeiden. Die Tatsache, dass das Team selbst die Zielgruppe des Projektes repräsentierte, erwies sich als Vorteil, da eine bessere Anpassung an die Bedürfnisse der Nutzer ermöglicht wurde. Die Arbeit mit Huggingface, also mit Open-Source-Methoden erwies sich als gute Alternative. Die Arbeit mit den verschiedenen Modellen führte zu einem guten Learning bei allen Teammitgliedern. Es war möglich verschiedene Modelle auszuprobieren und kennenzulernen. In Bezug auf Verbesserungen, Erweiterungen und Zukunftsideen wurden mehrere Möglichkeiten identifiziert. Dazu gehört die Verarbeitung von Bildern in den PDF-Dateien und die Anzeige eines Seitenindex. Des Weiteren könnte man den Chatbot so erweitern, dass dieser in mehreren Sprachen unterstützt wird. Eine weitere Möglichkeit ist es, auf einem Modell ein Feintuning direkt auf vorlesungsspezifischen oder anderen wissenschaftlichen Daten durchzuführen. Da aktuell nur eine PDF-Datei verarbeitet werden kann, könnte man das zudem Projekt so erweitern, dass es möglich ist, mehrere PDF-Dateien zu verarbeiten. Diese und weitere Aspekte bieten Potenzial für die Weiterentwicklung des Chatbots und die Steigerung seiner Funktionalität.

References

[1] B.R. Ranoliya, N. Raghuwanshi, S. Singh, *Chatbot for university related FAQs*, in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (2017), pp. 1525–1530

[2] T. Lalwani, S. Bhalotia, A. Pal, V. Rathod, S. Bisen, *Implementation of a Chatbot System using AI and NLP* (2018), https://papers.ssrn.com/abstract=3531782

[3] P.C. Mathieu Fenniak, *Extract Text from a PDF — pypdf 4.0.0 documentation*, https://pypdf.readthedocs.io/en/stable/user/extract-text.html

[4] M. Azure, *Azure ki dokument intelligenz*, https://azure.microsoft.com/de-de/products/ai-services/ai-document-intelligence

[5] N.Z. Berlin, *Chunking - NLP-glossar*, https://nlp-zentrum-berlin.de/infothek/nlp-glossar/chunking

[6] Turing, *Word embeddings in NLP: A complete guide*, https://www.turing.com/kb/guide-on-word-embeddings-in-nlp

[7] M. Walter, *So werden aus LLMs und vektordatenbanken hocheffiziente NLP-suchmaschinen* (2024), https://t.ly/2U2vL

[8] R. Tripathi, *What is similarity search? | pinecone*, https://www.pinecone.io/learn/what-is-similarity-search/

[9] H. Face, *What is text generation?*, https://huggingface.co/tasks/text-generation

[10] Clarifai, *tiiuae-falcon-7b-instruct*, https://clarifai.com/clarifai/LLM-OpenSource-Models-Training-Inference-Test/models/tiiuae-falcon-7b-instruct

[11] G.C. Console, *Falcon-instruct (PEFT)* (2023), https://console.cloud.google.com/marketplace/product/tii/falcon-instruct?hl=de&pli=1&project=big-data-392309

[12] S.S. Hebbar, *Text generation v/s text2text generation*, https://t.ly/hyegE

[13] H.W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma et al., *Scaling instruction-finetuned language models* (2022), https://doi.org/10.48550/arXiv.2210.11416

[14] J. Jacob, *Accurate transcription and AI-assisted editing & analysis*, https://exemplary.ai/blog/flan-t5

[15] H. Face, *T5*, https://huggingface.co/docs/transformers/model_doc/t5

[16] F. Höhn, *Natural language question answering-systeme*, https://t.ly/v2tdf

[17] F. Chiusano, *Two minutes NLP — quick intro to question answering*, https://t.ly/A4DSn

[18] H. Face, *What is question answering?*, https://huggingface.co/tasks/question-answering

[19] V. Sanh (2019)

[20] V. Sanh, L. Debut, J. Chaumond, T. Wolf, CoRR **abs/1910.01108** (2019)

[21] S. University, *The stanford question answering dataset*, https://rajpurkar.github.io/SQuAD-explorer/

[22] P.J. Gruber, *Lecture Notes and Handouts | Principles of Microeconomics | Economics*, https://ocw.mit.edu/courses/14-01-principles-of-microeconomics-fall-2018/pages/lecture-notes/

A Anhang

A.1 Entwicklung eines eigenen OCR-Modells

LayoutParser bietet sowohl eine Auswahl an vortrainierten Modellen als auch die Möglichkeit des Nachtrainierens basierend auf eigenen Daten an. Da der in dem Projekt vorliegende Datensatz sehr spezifisch ist, haben die Standardmodelle keine guten Ergebnisse geliefert, weswegen ein eigenes Modell basierend auf den Folien als *Proof of Concept* trainiert wurde. Dazu wurde ein eigener annotierter Datensatz mittels Label Studio erstellt, bei dem auf einem Satz Folien Kopfzeilen, Titel, Bilder, Text und Fußzeile gelabelt wurden. Auf diesem Datensatz wurde dann ein Detectron2 Modell circa zwei Stunden lang mit der Unterstützung einer RTX2070S Grafikkarte nachtrainiert und war danach in der Lage, die bestimmten formalen Eigenheiten der Folien zu erkennen. Auch wenn dieser Ansatz aufgrund von Limitationen in den Bereichen Zeit und Umfang nicht Teil der finalen Anwendung wurde, zeigen kommerzielle Lösungen wie die von Microsoft, wie gut ausgereifte OCR-basierte Anwendungen zur Textextraktion funktionieren können. Aus den genannten Gründen wurde sich trotzdem für die Verwendung einer Python Bibliothek für strukturbasierte Textextraktion mit PyPDF2 bzw. PyMuPDF entschieden.

A.2 Semantisches Unterteilen des Textes

Der gesamte Text wird zunächst in grobe Chunks mit einer Größe von 100 Zeichen unterteilt, wobei ein sogenannter 'Overlap' von 20 Zeichen berücksichtigt wird, um sicherzustellen, dass keine Informationen verloren gehen, wenn beispielsweise chunks inmitten eines Wortes enden. Damit diese groben Teile auf Ähnlichkeiten untersucht werden können, werden diese mit Hilfe des Sentence Transformer Modells „all-MiniLM-L6-v2“ 'embedded' (vektorisiert). Die Wahl dieses Modells entstand durch die Eigenarten, da dieses für Aufgaben wie Clustering und semantische Suche trainiert und optimiert wurde. Der Sentence Transformer bildet die Chunks auf einen 384-dimensionalen Vektor ab.
Um thematische Schnittpunkte identifizieren zu können, werden im Anschluss sogenannte aktivierte Ähnlichkeiten berechnet. Um die Ähnlichkeit vektorisierter Sätze bzw. Chunks zu berechnen ist die Kosinus-Ähnlichkeit ein *state-of-the-art* Ansatz. Durch diese Berechnung ergibt sich eine Matrix. Aus dieser Matrix werden die Diagonalen nach der Initialen als Vektor extrahiert. Diese stellen die Ähnlichkeit zwei aufeinander folgender Chunks dar. Da jeder Chunk eine gewisse Anzahl Vorgänger und Nachfolger hat, werden die Vektoren durch das Hinzufügen von Nullen auf eine Länge gebracht. Alle Vektoren werden erneut in einer Matrix angeordnet. Anschließend werden Aktivierungsgewichte auf jede Zeile angewendet, so dass die nächstgelegenen Chunks das größte Gewicht zur Bestimmung der Ähnlichkeit erhalten. Zur Bestimmung der Gewichte wird eine umgekehrte Sigmoid Funktion benutzt. Abschließend wird die gewichtete Summe jeder Zeile errechnet, um die Ähnlichkeiten als Vektoren darzustellen. Die daraus resultierende Funktion wird auf Minima untersucht, welches die Punkte repräsentieren, wonach der gesamte Text final unterteilt wird.

A.3 Gruppendynamik

Mitglieder	Themenbereiche
Adrian Setz (6483663)	Entwicklung Language Model, App
Alexander Paul (5722106)	Präsentation, PDF Preprocessing
Nicholas Link (3967704)	Support Text Verarbeitung, Text Search, Evaluierung
Lars Kurschilgen (3694647)	Preprocessing, Entwicklung Language Model
Jan Wolter (5222965)	Text Extraktion, Chunking
Lucas Wätzig (8167493)	GUI, Text Verarbeitung, Text Search, Language Model

*An Bericht und Präsentation haben alle gearbeitet

Figure 2. Verteilung der Aufgaben innerhalb der Gruppe