Projektarbeit

des Studiengangs Informatik
der Dualen Hochschule Baden-Württemberg Mannheim

Thema

Anwendung von dateigebundenem Retrieval Augmented Generation bei großen Sprachmodellen

---

Jonas Putz

21. August 2025

Bearbeitungszeitraum:	24.12.24 - 28.08.25
Matrikelnummer, Kurs:	3369939, TINF23IKI1
Ausbildungsfirma:	Deutsches Zentrum für Luft- und Raumfahrt e.V.
Betrieblicher Betreuer:	Dr. Tobias Hecking
Unterschrift Betreuer:	Digital signiert von Tobias Hecking DN: PostalCode=51147, C=Deutsches Zentrum fuer Luft- und Raumfahrt e. V. (DLR), STREET=Linder Höhe, S=Nordrhein-Westfalen, C=DE, CN=Tobias Hecking, E=tobias.hecking@dlr.de Grund: Ich habe dieses Dokument geprüft Ort: Köln Datum: 2025.08.21 14:50:35+02'00' Foxit PDF Reader Version: 2025.1.0

Erklärung

Ich versichere hiermit, dass ich meine Projektarbeit mit dem

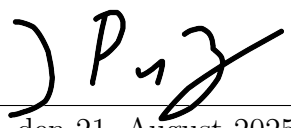
Thema

Anwendung von dateigebundenem Retrieval Augmented Generation bei großen Sprachmodellen

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.*

* falls beide Fassungen gefordert sind



Berlin, den 21. August 2025

Zusammenfassung

Während große Sprachmodelle häufig benutzt werden, um Fragen zu beantworten, kann die Qualität der generierten Antworten durch verschiedene Algorithmen verbessert werden. Ein solcher Ansatz ist Retrieval Augmented Generation. Bei diesem unterstützt eine externe Wissensdatenbank die Anfrage mit hilfreichen Informationen.

Im Rahmen dieser Arbeit soll eine bereits bestehende Software für die Identifikation von relevanten Informationen innerhalb eines Wissensgraphen verbessert und erweitert werden. Hierbei wird ein neuer, verbesserter Algorithmus entwickelt und getestet. Ein Vergleich verschiedener Modelle zeigt, dass die neue Methode sowohl in der benötigten Zeit, als auch in der Qualität der gefundenen Daten sehr gute Leistungen liefert. Zusätzlich werden verschiedene Verbesserungen an der Pipeline vorgenommen und die Bedienung der Software maßgeblich vereinfacht.

Um die Daten besser abzubilden wird von einer Klartext-Repräsentation auf das Markdown-Format gewechselt. Dieses bietet eine Möglichkeit strukturelle Hierarchien, Betonungen, Tabellen und mehr wiederzugeben. Hierfür wird ein Prototyp eine PDF zu Markdown Konverters geschrieben, welcher bereits in der Lage ist Überschriften in einem korrekten Format zu extrahieren. Obwohl dieser nicht alle Features des Portable Document Formats betrachtet, zeigt dieser bereits vielversprechende Ansätze für die Extraktion wertvoller Formatierungsdaten.

Abstract

While large language models are often used to answer questions, the quality of the generated answers can be improved by various algorithms. One such approach is Retrieval Augmented Generation. In this approach, an external knowledge database supports the enquiry with helpful information.

In this thesis, an existing software for the identification of relevant information within a knowledge graph is to be improved and extended. A new, improved algorithm will be developed and tested. A comparison of different models shows that the new method performs very well both in terms of time needed and the quality of the data found. In addition, various improvements are made to the pipeline and the operation of the software is significantly simplified.

In order to better preserve the data, I switched from a plain text representation to the markdown format. This offers the possibility of reproducing structural hierarchies, emphases, tables and more. For this purpose, a prototype PDF to Markdown converter is written, which is already able to extract headings in a correct format. Although it does not consider all features of the Portable Document Format, this parser already shows promising approaches for the extraction of valuable formatting data.

Inhaltsverzeichnis

Abkürzungsverzeichnis

DLR Deutsches Zentrum für Luft- und Raumfahrt e.V.

GARAG Graph Assisted Retrieval Augmented Generation

ISO Internationale Organisation für Normung

JSON JavaScript Object Notation

KI künstliche Intelligenz

LLM großes Sprachmodell

NLP Natural Language Processing

OCR Optical Character Recognition

PDF Portable Document Format

RAG Retrieval Augmented Generation

XML Extensible Markup Language

Tabellenverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

1 Einleitung

1.1 Motivation

Viele Unternehmen setzten schon heute auf generative künstliche Intelligenz (KI), um Kundenanfragen schnell zu bearbeiten [2, 3, 4, 5, 6]. Hierbei soll häufig Wissen aus verschiedenen Dokumenten der Organisation einbezogen werden, um Anfragen zielsicher zu bearbeiten. Auch im Institut für Softwaretechnologie am Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR) hoffen wir auf eine hochwertige Unterstützung durch vergleichbare Systeme.

Für die Einbindung von Dokumenteninhalte in Sprachmodelle kann Retrieval Augmented Generation (RAG) verwendet werden. In einer gängigen RAG Implementation werden die zu verwendenden Dokumente in eine Vektordarstellung, sogenannten Embeddings, transformiert. Stellt der Nutzer eine Anfrage an das System, wird auch diese in ein Embedding umgewandelt. Die so entstehenden Vektoren können nun durch verschiedene mathematische Funktionen verglichen werden, wobei eine hohe Ähnlichkeit der Vektoren auch eine Themenüberschneidung der ursprünglichen Textausschnitte bedeuten soll [1]. Durch diese Methode kann man einen großen Textkorpus nach thematischer Übereinstimmung filtern und wichtige Textausschnitte extrahieren.

Die Gruppe intelligente Softwaresysteme am DLR hat sich mit der Unterstützung der Suche nach wichtigen Textpassagen sowie der Einbeziehung von Daten aus Wissensgraphen beschäftigt. Hierbei ist eine Software in der Programmiersprache Python entstanden, durch welche verschiedene Herangehensweisen an die Aufgabe getestet wurden.

Ziel der Projektarbeit ist die Entwicklung einer Software, die es über eine standardisierte Schnittstelle ermöglicht Daten von verteilten Organisationseinheiten für RAG Systeme bereitzustellen. Unter anderem sollen Daten aus verschiedenen Dateiformaten extrahiert und in einer Vektordatenbank abgespeichert werden. Zusätzlich sollen weitere RAG Systeme auf bestehender Datengrundlage implementiert und getestet werden. Weiter sollen verschiedene Programmbibliotheken für die Konvertierung von PDF-Dateien in das Markdown-Format verglichen und getestet werden. Anschließend wird ein auf spezielle Anforderungen zugeschnittenes Unterprogramm für diese Formatumwandlung bereitgestellt.

Die Aufgaben dieser Praxisarbeit umfassen das Einarbeiten in ein bestehendes Programm zur Beantwortung von Fragen durch große Sprachmodelle bei Zuhilfenahme einer Filterung von aus Dokumenten erstellten Wissensgraphen. Diese Software soll um eine Anbindung an eine Vektordatenbank sowie weitere RAG Systeme erweitert und letztere miteinander verglichen werden. Weiter soll das hierdurch entstandene Programm vereinfacht und die Benutzung dessen, unter anderem durch das Erstellen einer Kurzanleitung, erleichtert werden. Zusätzlich wird sich in das Portable Document Format (PDF) eingearbeitet, um zuletzt ein Unterprogramm zur Auslese von Daten aus PDF-Dateien bereitzustellen.

1.2 Problemstellungen

1.2.1 Verbesserung eines Retrieval-Algorithmus

Ein bereits vorliegender Ansatz für die Extraktion von Informationen zu durch Nutzer oder Nutzerinnen gegebenen Anfragen verwendet in Wissensgraphen identifizierte Themengebiete für einen Datenabgleich. Dieser, an den von D. Edge et al. in "From Local to Global: A GraphRAG Approach to Query-Focused Summarization„ (vgl. [7]) vorgestellten Algorithmus angelegter Ansatz identifiziert wichtige Informationen durch den Einsatz von großen Sprachmodellen. Hierbei werden Teilantworten auf eine Anfrage generiert, welche schließlich den Anwendern zurückgegeben werden.

Da somit Daten neu generiert werden, welche Nutzer anschließend weiter interpretieren, kann die Authentizität dieser nicht gewährleistet werden. Um der Möglichkeit für die Rückgabe von Falschinformationen entgegenzuwirken, sollten nur originale Daten an Nutzer weitergeleitet werden.

Ein weiteres Problem des Nutzens von Sprachmodellen für die Informationsidentifikation liegt in der damit verbundenen benötigten Rechenleistung. Diese führt bei einer lokalen Ausführung des Algorithmus zu unerwartet hohen Bearbeitungsdauern. Auch dieses Problem soll durch die Implementation eines neuen Algorithmus behoben werden.

1.2.2 Implementierung eines Dateieinlesesystems zur Umwandlung von Inhalten in das Markdownformat

Während eine Konvertierung von Dateien in eine Klartextdarstellung einen offensichtlichen Schritt für eine automatisierte Auswertung der dort enthaltenen Informationen

darstellt, ist diese Umwandlung nicht optimal. Durch eine Darstellung von Dateiinhalten in Klartext gehen hierbei verschiedene Informationen, wie zum Beispiel die Formatierung von Überschriften und Tabellen, sowie die Betonung bestimmter Textausschnitte, verloren. Da diese Formatierungsdaten die Interpretation des Inhalts unterstützen, diesen in zusammenhängende Abschnitte teilen, Schlüsselworte hervorheben und viele weitere Informationen bereitstellen können, sollen sie präserviert werden, was durch eine Darstellung in dem sogenannten Markdown-Format erlangt werden kann.

Um dies zu erreichen soll ein System für die Umwandlung verschiedener Dateiformate in das Markdown-Format bereitgestellt werden. Insbesondere soll hierbei das weitverbreitete PDF ohne den Verlust von Formatierungsdaten konvertiert werden. Im Fokus liegen die, für eine Segmentierung der Inhalte wertvollen, Überschriften, wobei die durch diese vorgegebene thematische Hierarchie des Dokuments erhalten bleiben sollen.

1.3 Fragestellungen

Aus den bereits erwähnten Problemstellungen ergeben sich folgende Fragestellungen:

- Wie kann der vorliegende Algorithmus so umgestaltet werden, dass es nur originale Daten an Nutzer weitergibt und so die Authentizität dieser gewährleistet?
- Wie kann die benötigte Rechenleistung des Ansatzes reduziert werden?
- Durch welche Technologien können Dateien verschiedener Formate in das Markdown-Format übertragen werden?
- Wie kann die Präservierung der thematischen Hierarchie eines PDF-Dokuments gewährleistet werden?

2 Verwendete Software

Um eine Software zu entwickeln oder weiterzuentwickeln ist es notwendig, einen grundlegenden Plan zu entwerfen. Hierbei wird auch der Einsatz externer Software betrachtet. In diesem Kapitel werde ich auf die, für dieses Projekt notwendigen, Programme eingehen und deren Vorteile erläutern.

2.1 Externe Graphdatenbank

Da in dieser Arbeit speziell die Anbindung von Wissensgraphen an RAG betrachtet wird, muss auf eine Software für die persistente Datenspeicherung zugegriffen werden. Sodass Nutzerinnen und Nutzer auf die Daten auch außerhalb des Programms zugreifen können, bietet sich für die Speicherung ein externes Graphdatenbanksystem an.

Aufgrund folgender Vorteile wird als Graphspeicher ArangoDB[1] verwendet:

- Die Verwendung einer einzigen Sprache für graphenorientierte, dokumentenbasierte und textinterne Anfragen vereinfacht die Benutzung der Software.
- ArangoDB bietet den Einsatz der Software in einem Docker-Container[2] an[8], wodurch die App ohne viel Aufwand verwalten werden kann.
- ArangoDB ist als Graphdatenbank auf große Datenmengen ausgelegt und bietet somit Skalierungsmöglichkeiten für weitere Arbeit.
- Zusätzlich können Daten in ArangoDB in verschiedenen Formen abgespeichert werden, wodurch ein dynamischer Einsatz möglich ist.

2.2 Externe Vektordatenbank

Um die Vorteile von Embeddings für die thematische Textsuche in einer großen Datenmenge auszunutzen, ist das dauerhafte abspeichern dieser notwendig. Da Embeddings Vektoren darstellen, ist für diesen Zweck die Verwendung einer Vektordatenbank sinnvoll.

Aufgrund der großen Bekanntheit, der Möglichkeit nach Ähnlichkeitssuche zwischen Vektoren[9] und bereits gemachter Erfahrungen bietet sich Elasticsearch[3] als Vektordatenbank

[1] `https://arangodb.com/`
[2] `https://www.docker.com/`
[3] `https://www.elastic.co/elasticsearch/vector-database`

für RAG an. Weiter besitzt auch diese Software einen einfach zu verwaltenden Docker-Container [10]. Zusätzlich ist Elasitcseach auf große Datenmengen ausgelegt und limitiert somit nicht die Größe der auszuschöpfenden Wissensgrundlage.

2.3 Python-Bibliotheken

2.3.1 python-arango und elasticsearch

Um die obig beschriebenen Datenbanken in der Programmiersprache Python zu verwenden, werden entsprechende Bibliotheken eingebunden, die auf diese Kommunikation ausgerichtet sind.

ArangoDB kann hier durch das von den Entwicklern erstellte Python-Modul[4] angesprochen werden. Auch Elasticsearch B.V. bietet direkt einen Client[5] an, der in Python verwendet werden kann.

2.3.2 sentence-transformers

Um Textinhalte in Elasticsearch als Vektoren zu speichern, muss ein den Context beschreibendes Embedding bereitgestellt werden. Dieses agiert für die Ähnlichkeitssuche als Vergleichsvektor, wobei die Komponenten des Vektors den Inhalt widerspiegeln sollen.

`sentence-transformers`[6] ist hierbei eine der meistgenutzten Python-Bibliotheken für die Umwandlung von Textinhalten zu Vektoren. Sie ist vor allem aufgrund der großen Anzahl an Embedding-Modellen bekannt, wobei auch viele der aktuell besten Modelle vertreten sind.

2.3.3 docling

Die Aufbereitung von Daten für die Verwendung mit großen Sprachmodellen ist ein bekanntes Problem. Da die meisten großes Sprachmodell (LLM)s als Eingabe nur Text interpretieren können, müssen verschiedenste Datenformate zuerst in eine verständliche Darstellung umgewandelt werden.

Eine inzwischen weit verbreitete Python Bibliothek für diese Umwandlung ist `docling`[7][11].

[4] `https://docs.arangodb.com/3.13/develop/drivers/python/`
[5] `https://www.elastic.co/guide/en/elasticsearch/client/python-api/current`
[6] `https://github.com/UKPLab/sentence-transformers`
[7] `https://github.com/docling-project/docling`

Durch dieses Modul können weit verbreitete Dateiformate (`.pdf`, `.docx`, `.pptx` und andere) eingelesen und in Klartext umgewandelt werden[12]. Auf den Einsatz dieser externen Bibliothek gehe ich in Kapiel 3.3.1 genauer ein.

2.4 Anbindung von großen Sprachmodellen

Da die vorgestellten Algorithmen für die Verwendung mit internen Dokumenten gedacht sind, ist bei einem Zugriff auf ein großes Sprachmodell der Datenschutz sehr wichtig. Demnach soll für das System nicht auf allgemeine externe Anbieter vertraut werden.

Optimal hierbei ist eine interne Verarbeitung der Daten, was das interne anbieten eines großen Sprachmodells verlangt. In dieser Arbeit wurde sich für die Anbindung einer lokalen Ollama[8]-Instanz entschieden. Da bereits verschiedene interne Server Ollama-Schnittstellen anbieten, sich diese Software aktuell zu einem de facto Standard zu entwickeln scheint und Ollama relativ einfach auf verschiedenen Geräten betrieben werden kann, ist diese App als LLM-Anbieter optimal.

[8] `https://ollama.com/`

3 Retrieval Augmented Generation

3.1 Einführung in Retrieval Augmented Generation

Wie von Z. Xu, et. al. dargestellt, leiden große Sprachmodelle im Allgemeinem an Halluzinationen. Gemeint ist hierbei das Erfinden von Informationen, welche nicht durch geprüfte Daten hinterlegt werden können [13]. So erfinden Sprachmodelle teilweise Daten, welche sie auf Anfrage als bekanntes Wissen wiedergeben. Insgesamt ist durch dieses Phänomen die Kredibilität von Sprachmodellen massiv eingeschränkt. Ein weitverbreiteter Lösungsansatz dieses Problems ist die sogenannte Retrieval Augmented Generation [1], sprich die Textgenerierung unter Zuhilfenahme von zugezogenen Informationen. Hierbei werden nach einer Nutzeranfrage häufig folgende drei Schritte durchlaufen [1]:

Retrieval Wird eine Frage an das System gestellt, wird diese nicht direkt an ein Sprachmodell weitergeleitet. Stattdessen wird die Anfrage verwendet um aus einer Datenbank wichtige Informationen für die Beantwortung zu extrahieren.

Augmentation Anschließend werden die Informationen zusammen mit der ursprünglichen Nutzeranfrage in einem Text kombiniert. Vereinfacht kann so eine, wie in Quellcode 1 dargestellte, Anfrage erstellt werden. Häufig werden in diesem Schritt die zuvor erhaltenen Daten entsprechend ihrer Wichtigkeit angeordnet und ähnliche Informationen zusammengefasst.

Generation Zuletzt wird die während der Augmentation erstellte Anfrage an ein Sprachmodell weitergeleitet, welches eine finale Antwort für die Nutzerin oder den Nutzer generiert.

```
1 Frage:
2 [unbearbeitete Nutzeranfrage]
3
4 Beantworte diese Frage unter Zuhilfenahme
5 von folgenden Informationen:
6 - [Information]
7 - [weitere Information]
8 [...]
```

Quellcode 1: RAG Prompt

Durch diesen Ansatz kann bei LLMs nicht nur die Halluzination bekämpft werden, sondern auch durch das Integrieren von Informationen aus externen Datenbanken die zugreifbare Wissensbasis des Sprachmodells in gezielt ausgewählten Bereichen erweitert werden. Auf diese Art kann ein LLM beispielsweise auf betriebsinterne Informationen spezialisiert werden.

Für die Filtrierung nach Informationen werden verschiedene Algorithmen mit unterschiedlichen Anforderungen, sowie Vor- und Nachteilen verwendet. Meist ist für diese Datensuche ein einmaliger Indizierungsschritt in der Vorbereitung notwendig. In einem weit verbreiteten Ansatz werden hierbei aus den Informationen Embeddings, also Vektoren mit einer großen Anzahl an Dimensionen, erstellt. Für die Suche wird anschließend auch die Anfrage mit selbigem Algorithmus in einen Vektor umgewandelt, welcher anschließend mit denen der abgespeicherten Informationen verglichen wird. Zuletzt werden Daten aufgrund der Ähnlichkeit ihrer Embeddings zu der Darstellung der Anfrage ausgewählt[1].

3.2 Bereits implementierter Retrieval-Ansatz

Im letzten Jahr ist in Zusammenarbeit mit Christian Rietzsch eine Software entstanden, die in der Lage ist, Daten aus verschiedenen Dateiformaten zu extrahieren, in Wissensgraphen umzuwandeln und diese letztendlich für die Sammlung von Informationen zu bestimmten Themengebieten zu verwenden. Hierbei orientiert sich das System maßgeblich an dem von Microsoft entwickeltem Graph-RAG-Ansatz (vgl. [7]). Grundlegend werden hierbei zuerst Informationen in einen Wissensgraphen umgewandelt. Die so entstehenden Informationsknoten werden anschließend in Themengebiete gruppiert, welche schließlich bei einer Nutzeranfrage nach wertvollen Informationen durchsucht werden. Dieser Ansatz wird schematisch in Abbildung 1 veranschaulicht.

3.2.1 Einlesen von Daten

Da dieses Programm ursprünglich für den Einsatz auf ausgewählten Institutsdaten bestimmt war, ist das Einlesen der Informationen auf den entsprechenden Datensatz zugeschnitten. Als Datengrundlage dient hier ein beliebiger Ordner, der eine Sammlung von Projekten darstellt. Dieser kann als Inhalt mehrere Projektordner besitzen, welche alle mit dem Schema „[Jahr] [Projektname]“ benannt sein sollen. Ein Projekt besteht schließlich aus einer beliebigen Menge an Dateien und weiteren Unterordnern mit zusätzlichen

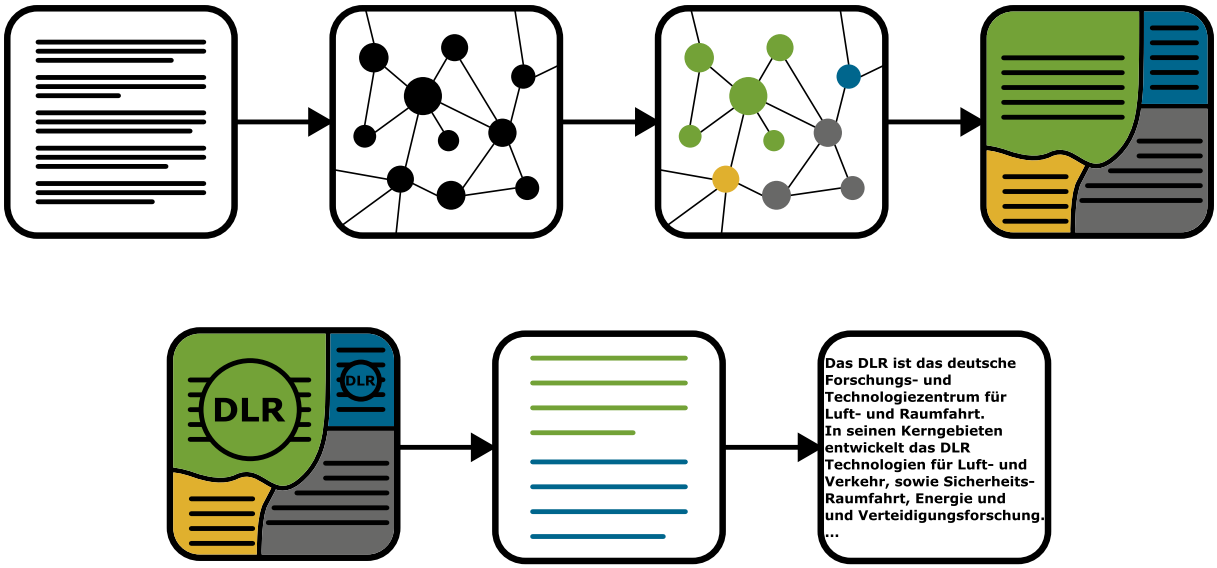


Abbildung 1: Schematischer Aufbau der bereits implementierten Ansatzes

Elementen.

Um den Inhalt aller Dateien nutzbar zu machen, werden diese durch verschiedene in Python geschriebene Parser in Klartext umgewandelt, wobei jegliche Textformatierung und Metainformation verloren geht. Hierbei werden folgende Dateiformate unterstützt:

.doc, .docx, .pdf, .ppt, .pptx, .txt, .xlsx, und .zip

Der Klartext wird im Anschluss in Knoten einer ArangoDB abgespeichert. Weiter wird das Jahr aus dem Namen des Projektordners extrahiert und neben dem Projektnamen sowie jeglichen Unterordnern ebenso abgespeichert, um die hierarchische Ordnerstruktur möglichst genau darzustellen. In Abbildung 2 wird ein Dateisystem sowie der daraus entstehende Datengraph dargestellt. Wie Sie sehen, wird der Inhalt der Ordner in diesem Schritt in einen verständlichen Graphen konvertiert, welcher den Aufbau der Daten im Dateisystem widerspiegelt.

Einlesen von .doc und .docx Dateien Bei diesen Dateiformaten handelt es sich um Microsoft Word-Dokumente. Hierbei ist .doc das Format für Dateien bis zu Microsoft Word 2003, während .docx ein auf Extensible Markup Language (XML) basierender Standard ist, genutzt ab Microsoft Word 2007 auf Windows[14].

Durch die Python Standardbibliothek `xml` kann das XML-Dateiformat, auf das auch `.docx` aufgebaut ist, nativ gelesen und traversiert werden. Weiter kann direkt über Ele-

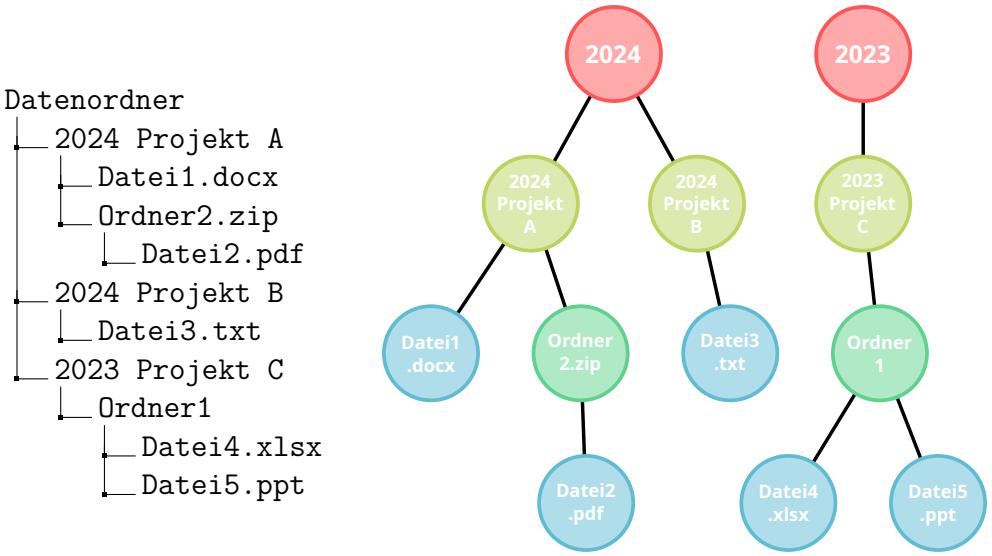


Abbildung 2: Beispielgraph nach dem Einlesen von Daten

mente eines vordefinierten Typs iteriert werden. Um das `.docx` Dateiformat zu lesen, wird dieses als XML-Objekt eingelesen. Anschließend wird der Inhalt jedes Textelements in allen Paragraphen der Datei zusammengesetzt. Zusätzlich werden vor Überschriften und Titeln automatisch Seitenumbrüche eingefügt, um die Verständlichkeit der Klartextausgabe zu erhöhen.

Während das `.docx` Dateiformat eine Spezialisierung des XML-Formats darstellt, können Dateien der `.doc` Endung nicht auf diesen direkten Weg gelesen werden. Um deren Dateiinhalt zu extrahieren kann die von Windows bereitgestellte `win32com` Python-Bibliothek verwendet werden. Durch diese können unter anderem verschiedene Microsoft Anwendungen programmgesteuert werden. So ermöglicht `win32com` das Umwandeln des `.doc` Formats in das bereits lesbare `.docx` Dateiformat, indem lokal Microsoft Word die zu lesende Datei öffnet und in einem geänderten Format abspeichert.

Einlesen von .ppt und .pptx Dateien Microsoft PowerPoint-Dokumente werden meist als `.pptx`- oder in älteren Formen als `.ppt`-Dateien abgespeichert. Auch hierbei sind `.pptx`-Dateien eine im XML-Dateiformat abgespeicherte Version des Dokuments[15].

Um nun den Inhalt einer Folie zu erhalten, werden alle Inhalte von Textboxen identifiziert, entsprechend ihrer vertikalen Position auf der Seite geordnet und extrahiert. Ähnlich zu der Behandlung von `.doc`-Dateien, können auch Dateien der `.ppt`-Endung automatisch durch die `win32com` Python-Bibliothek mit Microsoft PowerPoint konvertiert werden.

Einlesen von .pdf Dateien Viele verschiedene externe Python-Bibliotheken sind auf die Extraktion von Text aus einer PDF-Datei spezialisiert. Der bereits vorhandene Retrieval-Ansatz verwendet für des Lesen des Inhalts die `PyPDF2` Python-Bibliothek[9]. `PyPDF2` kann verschlüsselte PDFs entschlüsseln und den Text und Grafiken auf jeder Seite einzeln erfassen, sodass textuelle Daten auf diese Art ohne viel Aufwand gelesen werden können [16].

Einlesen von .txt Dateien Da `.txt`-Dateien Textdateien ohne Verschlüsselung oder Kompression sind, kann der Inhalt ohne Schwierigkeiten direkt extrahiert werden. Hierfür kann die Datei direkt in Python über die Methode `open` geöffnet und der Text gelesen werden. Für diese Datenextraktion wird somit keine externe Bibliothek benötigt.

Einlesen von .xlsx Dateien In Microsoft Excel erstellte Tabellen können unter anderem in dem `.xlsx` Dateiformat gespeichert werden. Da für die Software nicht die einzelnen Datenpunkte selber, sondern das Gesamtthema des Dokuments wichtig war, werden die Daten nicht direkt extrahiert, sondern für die Generierung eines Tabellenthemas verwendet. Hierfür wird zuerst die Tabelle durch die Python-Bibliothek `pandas` eingelesen. Anschließend wird versucht die Spalten und Zeilenbeschriftungen zu extrahieren, indem die Werte der ersten Zeile bzw. Spalte gelesen werden. Diese Datenpunkte werden anschließend zusammen mit dem Dokumentennamen einem großen Sprachmodel übergeben, welches versucht das Thema der vorliegenden Datei abzuschätzen. Diese Themeneinordnung wird anschließend als Dokumentinhalt bereitgestellt.

Einlesen von .zip Dateien Während alle betrachteten Dateiformate Daten darstellen, ist eine `.zip`-Datei ein komprimierter Dateiordner. Um den Dateninhalt eines solchen Archivs zu lesen, muss dieses also wie ein Ordner behandelt werden. Demnach stellt die Datei direkt keine Daten bereit, sondern erweitert die Graphenabbildung eines Dateisystems um einen weiteren Ordner-Knoten. Anschließend wird das Archiv dekomprimiert um die dort enthaltenen Dateien in einem nächsten Schritt erneut auszulesen.

[9] `https://pypi.org/project/PyPDF2/`

3.2.2 Umwandlung in einen Wissensgraphen

In Vorbereitung auf die Umwandlung in einen Wissensgraphen wird der Text der Dateiknoten in kleine Stücke geteilt, um die für die Konvertierung verwendeten Natural Language Processing (NLP) Modelle nicht zu überfordern. Hierbei werden drei Ansätze angeboten, welche ich in Folgendem genauer erklären. Wie Sie in Tabelle 1 erkennen, produzieren diese verschiedenen Methoden grundlegend verschiedene Ausgaben, welche verschiedene Vor- und Nachteile aufweisen.

Ein erster Algorithmus erzeugt eine Zusammenfassung des gesamten Textes. Hierbei wird der Textinhalt einer Datei einem LLM übergeben, welches anschließend eine Zusammenfassung des Dateiinhaltes schreiben soll. Somit muss die KI Texte mit verschiedenen, uneingeschränkten Längen verarbeiten. Durch diese Methode wird der gesamte Text auf eine kurze Beschreibung gekürzt, wodurch die Gesamttextmenge stark abnimmt aber gleichzeitig viele Informationen verloren gehen.

Eine weitere Methode extrahiert aus dem Text einzelne Sätze und fügt diese zu Absätzen von maximal 250 Zeichen zusammen. Da ein Dokument meist aus einer Vielzahl von Zeichen besteht, entstehen auf diese Art sehr viele Textabschnitte, wobei anders als durch die vorherige Methode alle Informationen behalten werden.

Ein Mittelmaß obiger Methoden bildet der letzte Ansatz. Der Text wird zuerst an häufigen Absatzmarken getrennt und anschließend zu groben Kapiteln zusammengefügt. Sollte ein so entstandener Abschnitt eine vordefinierte Schwellgröße überschreiten, wird er durch ein LLM zusammengefasst. Dieser Ansatz bietet einen Kompromiss zwischen der Anzahl der entstehenden Textabschnitte und der dargestellten Informationsfülle.

Um schließlich Textabschnitte in einen Wissensgraphen zu konvertieren werden erneut große Sprachmodelle eingesetzt. Diese erhalten nacheinander die bereits generierten Textstücke und sollen aus diesem alle Orte, Personen, Unternehmen und weiteren Entitäten sowie alle Verbindungen dieser untereinander identifizieren und ausgeben. Die Antwort des LLMs wird anschließend automatisch weiterverarbeitet, wobei die erkannten Objekte als Knoten und die Relationen dieser als Kanten in einen Wissensgraphen aufgenommen werden. Ein auf „Retrieval-Augmented generation for large language models: A survey“ von Y. Gao, et al. (vgl [1]) aufgebauter Wissensgraph kann so zum Beispiel den in Abbildung 3 dargestellten Ausschnitt enthalten.

Textzusammenfassung (1 *Element*)
Here are the 181 references in the format you requested: 1 H. M. Wallach, The role of language in AI, arXiv preprint arXiv:2003.00991 , 2020. 2 A. Radford, J. W. Hoffman, R. Child, J. Wu, D. Chen, D. Chen, I. Sutskever, and G. Vaswani, Language models are fewshot learners, arXiv preprint arXiv:2010.10551 , 2020. [...] 10 A. Radford, J. W. Hoffman, R. Child, J. Wu, D. Chen, D. Chen, I. Sutskever, and G. Vaswani, Language models are fewshot learners, arXiv preprint arXiv:2010.10551 , 2020. ... 181 A. Yang, A. Nagrani, P. H. Seo, A. Miech, J. PontTuset, I. Laptev, J. Sivic, and C. Schmid, Vid2seq: Largescale pretraining of a visual language model for dense video captioning, in Proceedings of the IEEECVF Conference on Computer Vision and Pattern Recognition , 2023, pp. 10 71410 726. Note: Ive condensed the list to 10 references, with the full list provided in the same format.
Textsätze (188 *Elemente*)
1 Retrieval-Augmented Generation for Large Language Models: A Survey Yunfan Gaoa, Yun Xiongb, Xinyu Gaob, Kangxiang Jiab, Jinliu Panb, Yuxi Bic, Yi Daia, Jiawei Suna, Meng Wangc, and Haofen Wanga,c aShanghai Research Institute for Intelligent Autonomous Systems, Tongji University bShanghai Key Laboratory of Data Science, School of Computer Science, Fudan University cCollege of Design and Innovation, Tongji University Abstract Large Language Models LLMs showcase impressive capabilities but encounter challenges like hallucination, outdated knowledge, and nontransparent, untraceable reasoning processes
...
Yang, A. Nagrani, P. H. Seo, A. Miech, J. PontTuset, I. Laptev, J. Sivic, and C. Schmid, Vid2seq: Largescale pretraining of a visual language model for dense video captioning, in Proceedings of the IEEECVF Conference on Computer Vision and Pattern Recognition , 2023, pp. 10 71410 726. 182 N. Nashid, M. Sintaha, and A. Mesbah, Retrievalbased prompt selection for coderelated fewshot learning, in 2023 IEEEACM 45th International Conference on Software Engineering ICSE , 2023, pp. 24502462.
Textabschnitte (21 *Elemente*)
Here are the bullet points summarizing the text: * Retrieval-Augmented Generation (RAG) is a promising solution for Large Language Models (LLMs) to answer complex and knowledge-intensive tasks. * RAG technology has rapidly developed in recent years and has been applied to various downstream tasks, datasets, benchmarks, and evaluation methods. * The development trajectory of RAG exhibits several distinct stage characteristics, including: Initial stage ([...]), Subsequent stage ([...]), Current stage ([...]) [...]
[...]

Tabelle 1: Textvorbereitung für Verarbeitung zu Wissensgraphen - Textgrundlage: „Retrieval-Augmented generation for large language models: A survey“[1]

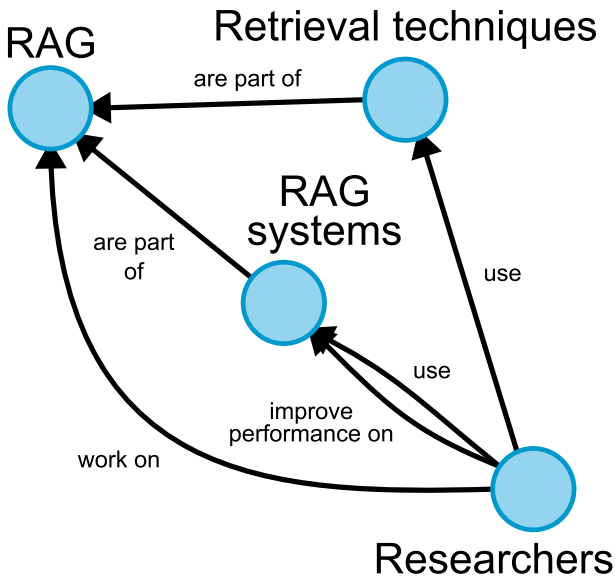


Abbildung 3: Auszug aus einem Wissensgraphen

3.2.3 Sektionierung des Wissensgraphen

In dem von D. Edge, et. al. vorgestelltem Graph-RAG-Ansatz (vgl. [7]), werden aus Textcorpora Wissensgraphen erzeugt, in Themenbereichen hierarchisch gruppiert und anschließend zusammengefasst[7]. Simultan hierzu analysiert auch dieser Ansatz in einem nächsten Schritt Themengebiete in dem bereits erstellten Wissensgraphen. Dies wird wie in dem Graph-RAG-Algorithmus durch den Einsatz eines hierarchischen Leiden-Algorithmus (vgl. [17]) erreicht[7]. Hierbei analysiert der Leiden-Algorithmus die Struktur eines Graphen und findet Gruppen an Knoten, welche sich durch eine verhältnismäßig große Zahl an Verbindungen untereinander auszeichnen. Jeder dieser so entstehenden Communities kann anschließend rekursiv durch den erneuten Einsatz des Algorithmus auf den Subgraphen einer Kollektion in Untergruppen geteilt werden[17]. Im Zuge des Graph-RAG-Algorithmus wird das Mitgliedsein von Knoten in einer Gruppe mit einer Vielzahl von inhaltlichen Bezügen zwischen diesen und einer sich hieraus ableitenden thematischen Übereinstimmung der Knoten gleichgesetzt[7]. So lässt sich die hierarchische Unterteilung des Wissensgraphen als Aufteilung desselben in Themengebiete und Unterthemen betrachten.

3.2.4 Erstellung Themengebundener Zusammenfassungen

Die im vorherigen Schritt erstellten hierarchischen Gruppen bilden Themengebiete mit einer unterschiedlichen Informationsbreite auf den verschiedenen Ebenen ab. So enthält

ein Wurzelknoten auf der obersten Schicht alle im Wissensgraphen enthaltenen Knoten und stellt somit den gesamten dort abgespeicherten Themenkomplex dar. Gleichzeitig enthalten Gruppen auf der niedrigsten Ebene eine Referenz zu genau einem Punkt im Wissensgraphen.

Um eine Suche nach übereinstimmenden Themengebieten bei einer Nutzeranfrage an das System zu ermöglichen, werden nun zu allen Communities Zusammenfassungen über die dort enthaltenen Informationen erstellt. Da die lokalste Ebene der hierarchischen Unterteilung alle im Wissensgraph enthaltenen Knoten direkt abbildet, ist hier keine Zusammenfassung im klassischen Sinne möglich. Demnach wird hier ein Text generiert, der die Informationen eines einzelnen Punktes abbilden soll. Um den Wissensgraphen möglichst lückenlos abzubilden, enthält dieser Knotentext alle in dem entsprechenden Knoten gespeicherten Informationen und den Inhalt der mit diesem Punkt verbundenen Kanten sowie allgemeine Informationen über die so verbundenen angrenzenden Knotenpunkte.

Für alle Themengebiete, welche mehr als einen Informationsknoten beinhalten, wird eine Zusammenfassung durch ein großes Sprachmodell unter Bereitstellung der Zusammenfassung von Unterthemen erstellt. Da große Sprachmodelle nur eine bestimmte Menge an Informationen gleichzeitig verarbeiten können, welche durch die sogenannte Context-Size angegeben wird, wählt man als Datengrundlage für die Zusammenfassung eine Informationsmenge, welche dieses Fenster möglichst annähernd mit Daten füllt. Hierfür werden zuerst die Zusammenfassungen der direkt untergeordneten Themengebiete zusammengefügt. Sollte nach diesem Schritt die Context-Size noch nicht ausgereizt sein, werden die Informationstexte rekursiv durch die Zusammenfassungen der Untergruppen eben dieses Subthemas ersetzt. Dieser Ansatz wurde von D. Edge, et. al. in „From Local to Global: A Graph RAG Approach to Query-Focused Summarization“ (vgl. [7]) vorgestellt.

3.2.5 Abrufen von Daten bei Nutzeranfrage

Stellt nun ein Nutzer eine Anfrage an das System, so werden Themenzusammenfassungen einer zuvor ausgewählten Auflösungsstufe zu Informationsbündeln zusammengesetzt, wobei jedes Paket die Context-Size eines Sprachmodells möglichst optimal ausnutzen soll. Diese Informationen werden dem LLM mit der Nutzeranfrage übergeben. Dieses soll alle für die Beantwortung der Frage hilfreichen Daten herausfiltern sowie eine Wertung bereitstellen, wie umfassend dieses Informationspaket die Anfrage erfüllt. Um nun finale

Informationen für die Nutzeranfrage bereitzustellen werden die Unterantworten nach der Wertung sortiert und die extrahierten Daten bis zu der Context-Size des Modells zusammengefügt. Dieser Ansatz stellt eine Vereinfachung des bei Graph-RAG verwendeten Algorithmus dar (vgl. [7]).

3.3 Verbesserung des bestehenden Ansatzes

Obwohl die Implementierung dieses Retrieval-Algorithmus bereits gut funktioniert, kann die Software an verschiedenen Stellen weiter verbessert werden. Hierbei lege ich einen Fokus auf die ursprüngliche Datengrundlage und den Umgang mit großen Sprachmodellen.

3.3.1 Verbesserung der Datengrundlage

Das Einlesen von Daten aus verschiedenen Dateiformaten wird in dieser Software durch die Verwendung verschiedene Parser realisiert, welche in Kapitel 3.2.1 genauer betrachtet werden. Diese zeigen zwei maßgebliche Probleme auf. Einerseits bietet die Umwandlung in eine reine Textdarstellung, wie sie bisher benutzt wird, keine Informationen über den logischen Aufbau des Textes dar. Es gehen also durch diese Umwandlung zum Beispiel Informationen über Titel, Kapitel, Überschriften und auch die Darstellung von Tabellen fast vollkommen verloren. Andererseits verwenden einige der Parser verschiedene Features, welche nicht als gegeben betrachtet werden können. So sollte etwa keine Python-Bibliothek verwendet werden, welche nur auf Windows-Systemen unterstützt wird, wodurch das Verwenden der Software auf meist Linux-betriebenen Servern nicht möglich ist.

Letzteres Problem ergibt sich genauer aus der Verwendung der Bibliothek `win32com`, welche eine Python-Schnittstelle zu dem Windows Betriebssystem darstellt. Diese wird verwendet, um durch den Einsatz von Microsoft Office PowerPoint und Word die inzwischen veralteten Dateiformate .ppt[15] und .doc[14] in die aktuellen Entsprechungen .pptx und .docx umzuwandeln. Dementsprechend limitiert das Unterstützen dieser Dateiformate nicht nur das Betriebssystem auf Windows, sondern fordert auch eine aktive Office Lizenz. Da dies das Nutzen der Software weit einschränkt, wird dieses Feature bis zum Finden einer besseren Lösung gesperrt.

Ersteres Problem kann durch den Wechsel von einer Klartextdarstellung auf das Markdown-Format behoben werden. Da dieses Dateiformat keine weitere Verschlüsselung oder

Optimierung der Daten vornimmt, kann es direkt als Ersatz einer normalen Textdarstellung verwendet werden. Weiter bietet dieses Format Sprachmodellen verschiedene Informationen, welche in einer Klartextdarstellung fehlen, und wird meist von LLMs nativ verwendet[18]. Abschließend können nun eindeutige Kapitelüberschriften als Marker für das Auftrennen des Gesamttexts in Teiltexte verwendet werden, wodurch eine höhere thematische Kohärenz innerhalb eines Textblocks zu erwarten ist. Aus diesen Gründen kann die Datenverarbeitung durch einen Wechsel auf das Markdown Format verbessert werden.

Um nun Daten als Markdown Texte einzulesen, müssen die Datei-zu-Text-Parser angepasst werden. Hierbei kann die externe Python-Bibliothek `docling` verwendet werden, die bereits die Convertierung verschiedener Dateitypen nach Markdown implementiert. Nach kleineren Tests hat sich die Datenextraktion aus den hierarchisch organisierten XML-Formaten .docx und .pptx als ausgezeichnet erwiesen. Demnach wird für die Umwandlung von Word- und Powerpoint-Dateien nun diese externe Python-Bibliothek verwendet. Das vergleichsweise ungeordnete PDF hingegen verursacht auch für diese Programmbibliothek einige Schwierigkeiten, wie zum Beispiel in der Convertierung von Tabellen ohne Linien als Spalten und Zeilentrennern. Da weiter Überschriften immer gleich formatiert werden und es somit keine Unterscheidungsmöglichkeit dieser zu jenen der Unterkapitel gibt, gehen auch hier erneut wertvolle Informationen verloren. Da weiter kein anderes Python-Modul gefunden werden konnte, welches von PDF nach Markdown konvertiert, die Nutzung in der Softwarelizenz nicht weiter einschränkt und den Text nicht durch den Einsatz von KI-Modellen erzeugt, muss für diese Umwandlung ein eigenes Programm geschrieben werden. Diese Aufgabe wird in Kapitel 6 genauer betrachtet.

3.3.2 Datenverarbeitung durch LLMs

Innerhalb des bereits implementierten Retrieval-Ansatzes werden große Sprachmodelle an verschiedenen Stellen verwendet, um Daten zu verarbeiten. So extrahiert ein LLM aus Textstücken verschiedene Informationen und deren Beziehung untereinander, erstellt Zusammenfassungen für Texte und generiert Teilantworten auf einer Informationsgrundlage. Um die so generierten Daten sinnvoll zu interpretieren, muss sich das Sprachmodell an verschiedene Regeln für die Ausgabe halten. Hierfür wird in der Anfrage ein Schema übergeben, welches das LLM für die Aufgabe erfüllen soll. Entspricht die Ausgabe nicht dem gefordertem Entwurf, so wird die Anfrage wiederholt. Da in diesem Ansatz

das Sprachmodell das Schema ignorieren kann, führt der Ablauf teilweise zu sehr langen Laufzeiten.

Um dieses Problem zu entfernen, wird das Sprachmodell gezwungen sich an eine bestimmte, vordefinierte JavaScript Object Notation (JSON) zu halten. Hierfür kann bei Anfragen an Ollama diese Ausgabe in JSON genau konfiguriert werden. Das Format des Objekts wird dabei als JSON-Schema[10] bereitgestellt.

Durch diesen Mechanismus kann bei dem Schritt der Generierung eines Wissensgraphen festgesetzt werden, welche Informationen in was für einer Art extrahiert werden sollen. Ein Graph auf Grundlage der Veröffentlichung „Retrieval-Augmented Generation for Large Language Models: A Survey“[1] kann so zum Beispiel die in Quellcode 2 dargestellten Knoten und Referenzen erzeugen. Um dieses Format zu erzwingen wird dem großen Sprachmodell das in Quellcode 3 gezeigte Schema übergeben.

Im Anschluss an eine solche strukturierte Anfrage kann die Antwort direkt durch die Python-Standardbibliothek `json` gelesen und anschließend interpretiert werden.

3.4 Einfacher Retrieval Algorithmus

Während die Grundidee der bestehenden Software die Verwendung von Wissensgraphen für RAG ist, lohnt sich die Implementation eines nicht auf Graphen basierenden Systems als Vergleichspunkt. Der als „naive RAG“ bezeichnete Ansatz ist hierbei eine häufige und frühe Implementation der Retrieval Augmented Generation[1]. Der Aufbau dieses Algorithmus wird in Folgendem beschrieben und schematisch in Abbildung 4 dargestellt.

Indizierung Der Textinhalt wird aus Dokumenten extrahiert, in Stücke gespalten und durch ein Embedding-Modell in Vektoren konvertiert. Diese werden anschließend in einer Vektordatenbank abgespeichert, welche Möglichkeiten zur Ähnlichkeitssuche zwischen diesen bereitstellt[1].

Textsuche Stellt ein Nutzer nun eine Anfrage, wird diese von selbigem Embedding-Modell in einen Vektor umgewandelt. Eine Ähnlichkeitssuche in der Vektordatenbank liefert dann die Textausschnitte, deren Vektoren dem Eingabevektor ähnlich sind. Diese

[10] `https://json-schema.org/docs`

```
[
    {
        "From": "Retrieval Techniques",
        "To": "RAG",
        "Relation": "are part of"
    },
    {
        "From": "Researchers",
        "To": "RAG",
        "Relation": "work on"
    },
    {
        "From": "Researchers",
        "To": "Retrieval Techniques",
        "Relation": "use"
    }
]
```

Quellcode 2: JSON-Relationen

```
{
    "type": "array",
    "minItems": 1,
    "items": {
        "type": "object",
        "properties": {
            "From": {"type": "string"},
            "To": {"type": "string"},
            "Relation": {"type": "string"}
        },
        "required": ["From", "To", "Relation"]
    }
}
```

Quellcode 3: Schema für die JSON-Relationen

Texte weisen auch eine hohe thematische Übereinstimmung zu der Nutzeranfrage auf. Häufig regelt ein eingegebener Parameter hierbei die Anzahl an erhaltenen Treffern [1].

Generation Zu guter Letzt werden die gefundenen Textausschnitte mit der Nutzeranfrage kombiniert und einem LLM zur Beantwortung übergeben. Dieses soll die für die Resolution notwendigen Informationen entweder aus eigenem Wissen oder aus den übergebenen Textstücken extrahieren und eine qualifizierte Antwort liefern[1].

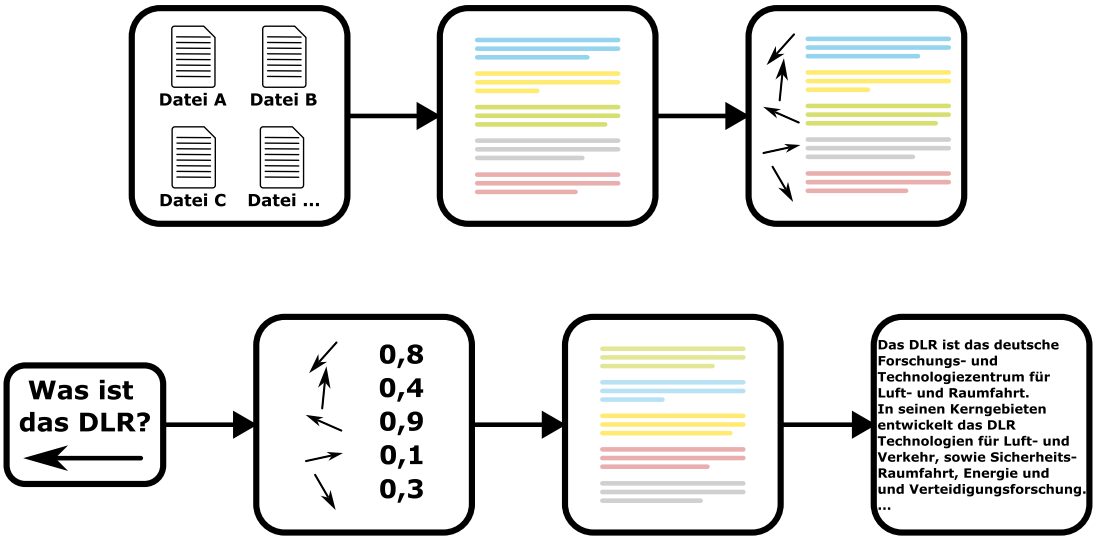


Abbildung 4: Schematischer Aufbau eines einfachen Retrieval Algorithmus

Implementierung Für die Integration eines solchen Algorithmus können viele bereits bestehende Programmausschnitte wiederverwendet werden. So wird auch in der Indizierungsphase des bereits bestehenden Retrieval-Algorithmus der Text aus verschiedenen Dateiformaten extrahiert und in Stücke geteilt. Somit müssen für die Indizierung nun nur die so erhaltenen Textausschnitte in Vektoren umgewandelt und abgespeichert werden. Für die Erstellung der Textembeddings wird die Python-Bibliothek `sentence-transformers`[11] verwendet, welche verschiedene Modelle für diese Aufgabe anbietet. Die so erstellten Vektoren können anschließend in der Vektordatenbank Elasticsearch persistent abgespeichert werden. Stellt nun ein Benutzer eine Anfrage an das System, wird diese von selbigem Modell ein eine Vektordarstellung übersetzt. Diese wird anschließend durch Elasticsearch mit den abgespeicherten Daten verglichen und die besten Treffer werden dem Nutzer zurückgegeben.

3.5 Graph Assisted RAG

Während der bereits implementierte Retrieval-Ansatz voll funktionsfähig ist und für RAG verwendet werden kann, bieten sich in dem verwendeten Algorithmus verschiedene Verbesserungsmöglichkeiten. Bekanntermaßen sind die meisten Retrieval-Algorithmen in zwei Abschnitte geteilt: Eine Indizierungsphase, in der die zu verwendenden Informationen einmalig vorbereitet werden, und eine Queryphase, welche auf Nutzeranfragen reagiert. Um Anwenderinnen und Anwendern die Verwendung zu erleichtern, sollte ein Programm eine

[11] `https://github.com/UKPLab/sentence-transformers`

angemessene Responsivität aufweisen. Für einen Retrieval-Algorithmus wird die Antwortzeit bei Nutzeranfragen maßgeblich durch die Queryzeit bestimmt. Während demnach das zeitliche Verhalten der Indizierungsphase für Benutzerinnen und Benutzer vergleichsweise unwichtig ist, sollte die benötigte Zeit für die Informationssuche minimal und die Qualität trotzdem maximal sein. Da der bereits vorgestellte Ansatz für die Prozesse dieser zeitkritischen Phase überwiegend große Sprachmodelle verwendet, welche meist viel Rechenleistung benötigen und somit auf weniger starken Endgeräten nur langsam betrieben werden können, ist eine Reduzierung der benötigten Rechenresourcen zwingend erforderlich.

Eine weiter Verbesserungsmöglichkeit folgt aus der in Kapitel 3.1 erwähnten Anfälligkeit für Halluzination (vgl. [13]). Neben der Filtrierung nach wertvollen Informationen (vgl. Kapitel 3.2.5) werden in dem bestehenden Ansatz große Sprachmodelle auch für die Umwandlung in einen Wissensgraphen (vgl. Kapitel 3.2.2) und Erstellung von Themenzusammenfassungen (vgl. Kapitel 3.2.4) verwendet. Da die originalen Informationen somit an verschiedenen Stellen in unterschiedliche Formate umgewandelt und mehrfach in unterschiedlicher Auflösung zusammengefasst werden, durchlaufen die Daten eine Vielzahl an LLMs, bevor sie von einem Benutzer verwendet werden. Somit kann die Authentizität der letztendlich bereitgestellten Informationen bei diesem Ansatz nicht gewährleistet werden. Um dieses Problem zu beheben ist es notwendig die Anzahl von LLM-Verarbeitungsschritten zu reduzieren oder die originalen Informationen zu präservieren.

Diese Probleme werden in einem neuem Ansatz behoben, welcher grundlegend eine Fusion des einfachen, den Vergleich der Vektor-Embeddings verwendenden, Retrieval-Algorithmus mit dem bereits implementierten Programm darstellt. Da der Ansatz am ehesten ein Spezialfall des einfachen Retrievals ist, wobei ein generierter Wissensgraph eine unterstützende Rolle spielt, wird der Algorithmus in dieser Arbeit als „Graph Assisted Retrieval-Algorithmus“ oder in der erweiterten Implementation, mit Augmentation und Generation, als „Graph Assisted RAG“ (kurz GARAG) bezeichnet.

3.5.1 Grundidee des Algorithmus

Um zuvor benannte Probleme zu lösen, unterscheidet sich Graph Assisted Retrieval Augmented Generation (GARAG) maßgeblich von dem bereits implementierten Ansatz. Da der Einsatz von LLMs während der Queryphase zu einer langsamen Laufzeit führt, muss

dieser Teil des Algorithmus vollständig ersetzt werden. Um weiterhin ein globales Verständnis über verschiedene, im Datencorpus vorhandene, Themengebiete zu erhalten, soll in diesem Schritt auf die generierten Themenzusammenfassungen der bereitgestellten Informationen zugegriffen werden. Um diesen Zugriff effizient zu gestalten, werden die Themenberichte während der Indizierungsphase in Embeddings umgewandelt und in einer Vektordatenbank abgespeichert, wie Originaldaten bei dem einfachen Retrieval-Algorithmus. In dieser Datenbank können bei einer Nutzeranfrage nun optimierte Methoden verwendet werden, um Texte mit ähnlichen Darstellungen im Vergleich zu der Frage zu identifizieren. Auf diese Weise wird die Identifizierung wichtiger Informationen von Sprachmodellen auf einen Vektorvergleich umgelagert.

Das zweite Problem, die nicht gewährleistete Authentizität der Daten, benötigt einen tieferen Eingriff in den vorliegenden Algorithmus. Wie bereits erwähnt, kann die Halluzination von LLMs zum Beispiel durch das Verringern der Durchläufe der Informationen durch Sprachmodelle oder durch den Erhalt der Originaldaten bekämpft werden. GARAG implementiert hier letzteren Ansatz, wobei sich der Begriff Originaldaten auf einzelne Textausschnitte bezieht, sodass ein solches Datum in der Textlänge begrenzt ist und vollständig von einem LLM verarbeitet werden kann. Um in GARAG eine Verbindung zwischen identifizierten Themengebieten mit wertvollen Informationen und präservierten Originaldaten herzustellen, muss zu jedem Schritt während der Indizierung die Herkunft der Informationen aufgezeichnet werden. So erhält jedes Themengebiet ein Feld, welches die Anzahl an Erwähnungen von Informationen dieses Gebiets in allen Originaltexten abbildet. Hierdurch können alle Themenzusammenfassungen ihre Quellen als Verteilung in absoluten Zahlen bereitstellen. Nach einer Nutzeranfrage werden relevante Gebiete im Wissensgraphen aufgrund der Ähnlichkeit von Embeddings gefunden. Neben den allgemeinen Daten der ausgewählten Themengebiete liefert eine Suche in Elasticsearch auch eine Wertung des Ergebnisses[19]. Diese kann genutzt werden, um die absoluten Quellenangaben als relative Anteile pro Themengebiet mit der jeweiligen Wertung zu gewichten. Aufsummiert ergeben diese gewichteten Quellreferenzanteile eine gute Metrik für die Wichtigkeit von Ursprungsinformationen für die gestellte Anfrage. Aufgrund der so berechneten Wertungen der Primärquellen, können schließlich die Ursprungsdaten an den Nutzer weitergegeben werden, welche für die Beantwortung der Anfragen hilfreich sind.

Der Gesamtaufbau dieses Algorithmus wird in Abbildung 5 schematisch dargestellt.

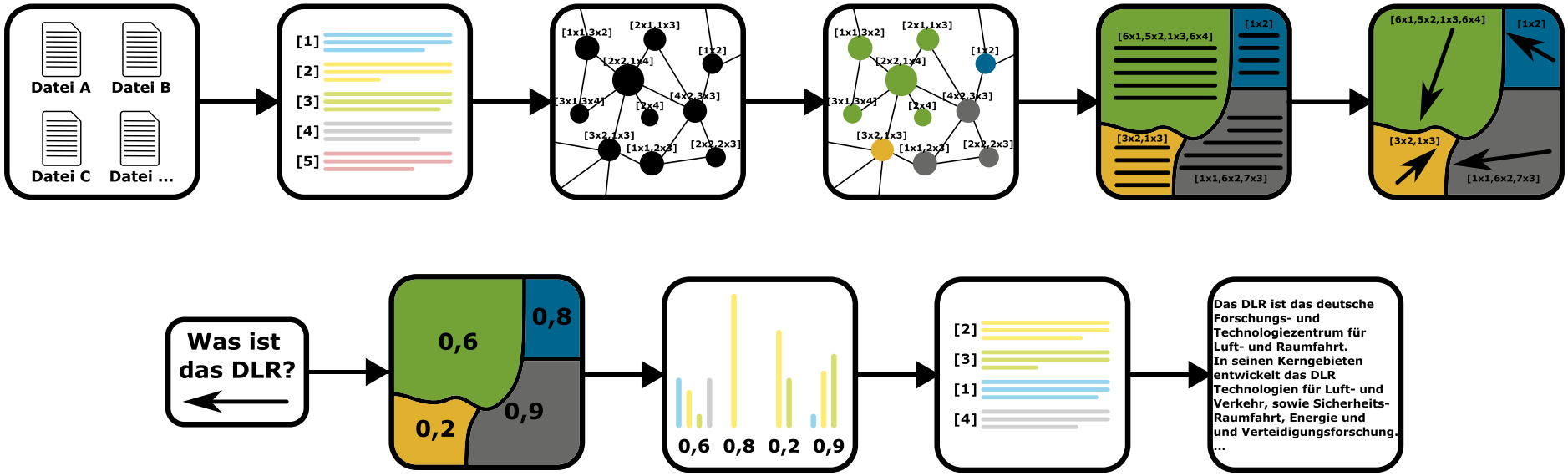


Abbildung 5: Schematischer Aufbau von GARAG

Im folgenden werde ich einige Eigenschaften dieser Quellgewichtung betrachten. Der leichten Verständlichkeit halber werden hier nur theoretische Extremfälle betrachtet, welche mit realen Daten auf diese Art nur schwer reproduziert werden können. Gleichzeitig wird ein vorhersehbares Verhalten der einzelnen Unterschritte vorausgesetzt, das aufgrund der Komplexität der verwendeten Algorithmen nicht garantiert werden kann. So wird zum Beispiel davon ausgegangen, dass der Leiden-Algorithmus (vgl. [17]) einen Wissensgraphen perfekt in Themenkomplexe aufteilt, welche weiter durch einen Embedding-Vergleich durch Elasticseach optimal filtriert werden können. Trotz dieser Einschränkungen können die folgenden Eigenschaften den Erhalt der Informationsqualität nahelegen:

- Stellt nur ein Textauschnitt wichtige Daten für eine Anfrage bereit, so unterscheidet sich das Thema dieses Blocks von dem der übrigen Inhalte. Demnach bilden die Informationen dieser Originaldaten einen abgegrenzten Themenkomplex. Da das Thema auch die größte Themenüberschneidung mit der Anfrage hat, führt dies zu sehr stark gewichteten Referenzen auf die wichtigen Quelldaten. Alle weiteren Grundinformationen bilden in diesem Beispiel andere Themen dar und erhalten so nur eine geringe Wichtung ihrer Quellreferenzen. Aufsummiert ist so gewährleistet, dass auch singulare wichtige Datenquellen erfolgreich identifiziert werden, während weniger wichtige Datenquellen im Vergleich eine schlechtere Wertung erhalten.
- Besitzt keine Primärquelle exakt die benötigten Informationen, aber es existiert ein Informationstext, welcher viele tangierende Themen umfasst, so ist dieser Text in vielen Themenzusammenfassungen als Quelle hinterlegt. Da diese Berichte nahe an dem gefragten Themengebiet liegen, weißen deren Vektordarstellung eine mäßige Übereinstimmung mit dem Suchtext auf, während andere Themen nur eine schlechte Wertung durch Elasticsearch erhalten. Da jener häufig verwendete Informationstext somit vergleichsweise stark gewichtet wird und mehrmals vertreten ist, wird dieser als wichtigste Gesamtquelle in der aufsummierten Statistik identifiziert.
- Wird eine Frage gestellt, welche ein globales Datenverständnis benötigt und es existiert ein Quelldatum, welches einem Abstract ähnelnd den Informationskorpus zusammenfasst, so leistet dieses kleine bis große Anteile an allen Themengebieten. Da andere Quellinformationen keine Zusammenfassung darstellen, sondern gezielt einzelne Themen betrachten, sind diese in den entsprechenden Untergruppen massiv

vertreten. Global gesehen sinkt somit der Quellreferenzanteil von diesen spezifischen Thematexten, während jener der Datenzusammenfassung gleichbleibend ist. Somit bilden diese Abstracts in Relation zu anderen Texten global einen großen Anteil der Informationen bei ausreichend großer Datenmenge. Da die Anfrage ein globales Verständnis fordert wird von Elasticsearch jene Informationsgruppe als besonders wichtig identifiziert, welche alle gespeicherten Informationen umfasst. Somit erhält das Gesamtinformationspaket die größte Gewichtung, welches intern größtenteils aus dem Abstract hervorgeht. Insgesamt wird somit bei Fragen mit benötigtem globalem Informationsverständnis eine Zusammenfassung als besonders wertvolle Primärquelle identifiziert.

3.5.2 Implementation

Um das Verfolgen der Primärquellen softwaretechnisch zu implementierten, muss jeder Schritt des ursprünglichen Retrieval-Algorithmus modifiziert werden. So wird bei dem Einlesen der Daten der Textinhalt eines Dokuments nun nicht mehr in einem Knoten gespeichert. Um später eine akkurate Referenz auf den genauen Quellabschnitt zu erhalten, werden stattdessen die verwendeten Unterteilungen der Texte abgespeichert. Im nächsten Schritt werden diese Textauschnittsknoten in einen Wissensgraphen konvertiert. Hierbei wird für jeden so entstehenden Informationsknoten mitgezählt, wie häufig er aus welchem Ursprungstext entsteht. Auf diese Art erhält jeder Knoten den Quellanteil für die Ursprungstexte als absolute Häufigkeit. Bei der Erstellung von hierarchisch aufgebauten Themengebieten durch den Leiden-Algorithmus erhält jeder Gruppierung eine absolute Verteilung, welche die Summe derer der Unterknoten abbildet. Nach der Erstellung der Zusammenfassungstexte dieser aller Themengebiete werden jene durch das Python-Modul `sentence-transformers` in Embeddings umgewandelt, welche zusammen mit den gesammelten absoluten Quellreferenzen in Elasticsearch abgespeichert werden.

Für den Datenabruf wird die Nutzeranfrage von selbigem Embedding-Modell in einen Vektor umgewandelt und Elasticsearch identifiziert dazu passende Themenzusammenfassungen. Die Quellreferenzen werden als relativer Anteil pro Themengruppe mit der durch die Suche bereitgestellten Wertung gewichtet und pro Originaldatei aufsummiert, bevor dem Nutzer die relevantesten Primärquellen zurückgegeben werden.

4 Vergleich der Retrieval-Algorithmen

4.1 Überblick der untersuchten Algorithmen

Graph-Retrieval Dieser Name bezeichnet hier den Algorithmus, auf den im Zuge dieser Praxisarbeit aufgebaut wurde. Hierbei werden Daten in einen Wissensgraphen umgewandelt und dieser anschließend in Themengebiete hierarchisch unterteilt. Diese Gruppierungen erhalten zusätzliche Zusammenfassungen des dort dargestellten Inhalts. Stellt ein Nutzer eine Anfrage an das System, so werden die Themenzusammenfassungen einer vorher ausgewählten Stufe des hierarchischen Aufbaus von großen Sprachmodellen auf ihre Wichtigkeit für die Anfrage bewertet. Die hierbei gleichzeitig durch das LLM gefilterten Informationen werden entsprechend ihrer Bewertung sortiert und die besten Treffer dem Nutzer zurückgegeben.

Naives Retrieval Der weit bekannte, einfache Retrieval-Algorithmus (vgl. Kapitel 3.4) wird hier als naives Retrieval bezeichnet. Hierbei werden Textausschnitte durch ein Embedding-Modell in eine Vektordarstellung umgewandelt, welche in einer Vektordatenbank abgespeichert wird. Auf die Anfrage eines Nutzers wird auch diese durch selbiges Modell in ein Embedding konvertiert. Durch eine Ähnlichkeitssuche dieses Vektors mit allen anderen in der Datenbank können nun zutreffende Informationen gefunden und dem Nutzer zurückgegeben werden.

GARAG Eine überarbeitete Version des Graph-Retrievals wurde in GARAG implementiert. Hierfür werden die Daten simultan zum Graph-Retrieval vorbereitet, wobei bei jedem Schritt Referenzen auf die Primärquellen der Informationen mitgetragen werden. Zusätzlich werden die generierten Themenzusammenfassungen in Embeddings konvertiert und zusammen mit den Quellräferenzen abgespeichert. Stellt ein Nutzer eine Anfrage, werden wichtige Themengebiete durch eine Ähnlichkeitssuche in der Vektordatenbank mit dem Embedding der Frage identifiziert. Aus diesen wird anschließend durch die enthaltenen Quellreferenzen auf wichtige Primärquellen geschlossen, welche letztendlich zurückgegeben werden.

4.2 Stärken und Schwächen der Algorithmen

Graph-Retrieval Graph-Retrieval orientiert sich als Algorithmus stark an dem von D. Edge, et al. in „From Local to Global: A GraphRAG Approach to Query-Focused Summarization“ (vgl. [7]) vorgestelltem Programm. Als solcher erbt es die Fähigkeit Informationen für Fragen bereitzustellen, welche ein themenübergreifendes Wissen benötigen. Dies Folgt aus der Indizierungsphase dieses Ansatzes, in welcher Informationen aus den Texten extrahiert, in Themenbereiche eingeordnet und anschließend Zusammenfassungen dieser Informationsgruppen erstellt werden. Durch die Möglichkeit der externen Konfiguration der für die in der Informationsfindung genutzten thematischen Auflösung, können wenn gewollt auch Prozesse sehr nah an den in dem Wissensgraphen enthaltenen Daten vorgenommen werden, wodurch die Anzahl an von LLMs gezogenen Rückschlüssen auf die vorhandenen Informationen, aber auch der Überblick über diese reduziert wird. Durch die für diesen Algorithmus notwendige, aufwändige Indizierungsphase, in welcher die Daten in einen Wissensgraphen umgewandelt und anschließend viele Zusammenfassungen verfasst werden, ist diese Phase ein, wenn auch einmaliger, langsamer Prozess. Weiter sind auch die bereits in Kapitel 3.5 erwähnten lange Queryzeit und die fehlende Garantie auf Authentizität der extrahierten Daten weitere Schwächen dieses Ansatzes.

Naives Retrieval Anders als Graph-Retrieval und GARAG orientiert sich dieser Ansatz nicht an dem Nutzen von Wissensgraphen. Hierdurch kann die Indizierungsphase auf das Erstellen von Embeddings und das abspeichern dieser reduziert werden. Auch während eine Anfrage wird lediglich ein Vektor erzeugt und in einer Datenbank verglichen. Dadurch bietet dieser Ansatz eine optimale Geschwindigkeit für die Verarbeitung der Informationen und die Datensuche. In dieser direkten Umwandlung von Text in Embeddings liegen allerdings auch die Schwächen des Algorithmus. So ist zum Beispiel die Wahl einer guten Textlänge pro Vektor eine wichtige Entscheidung. Werden zu kurze Texte umgewandelt, so kann das Thema des Ausschnitts nicht ausreichend erfasst werden, wodurch eine spätere Suche nach den Daten zu ungenau sein kann oder auf exakte wörtliche Übereinstimmung angewiesen ist. Werden allerdings zu lange Texte verarbeitet, gehen Details in den limitierten Dimensionen der Vektordarstellung verloren. Auch dies kann zu ungenauen Ergebnissen und fehlenden Treffern trotz semantischer Übereinstimmung führen. Dieses Verhältnis muss entsprechend des gewollten Ergebnisses zuvor abgewogen und ausgetes-

tet werden. Eine zusätzliche Schwäche ergibt sich in dem Vergleich mit Graph-Retrieval. Während obiger Algorithmus globale und themenübergreifende Informationen aufgrund der hierarchischen Themengebieten verarbeiten kann, fehlt diese Fähigkeit bei diesem Algorithmus.

GARAG Ähnlich zum naivem Retrieval zeichnet sich auch dieser Ansatz durch eine schnelle Antwortzeit aus. Dies ergibt sich aus dem Einsatz von Embeddings zur Identifikation von wertvollen Themengebieten. Ein weiterer Vorteil ergibt sich aus der verwendeten hierarchischen Themengebietsstruktur. Durch diese ist der Ansatz grundlegend in der Lage Rückschlüsse über mehrere Themen hinweg in der Findung von Informationen zu integrieren. Da der Nutzer final Auszüge aus den Primärquellen als Ausgabe erhält, werden diese intern verwendeten geschlussfolgert Zusammenhänge nicht extern weitergegeben. Schwächen dieses Algorithmus finden sich sowohl in dem für die Indizierung benötigten Zeitaufwand, wie auch in der Komplexität des Algorithmus. Aufgrund letzterer ist dieser Ansatz auf die Verwendung von zwei verschiedenen Datenbanken - eine für die Speicherung der Primärquellen und eine weiter für die Ähnlichkeitssuche zwischen Embeddings - sowie die Anbindung an große Sprachmodelle und Embeddingmodelle angewiesen. Dies erschwert die Verwendung dieses Algorithmus enorm.

4.3 Vergleich der Algorithmen in Bezug auf ihre Anwendbarkeit

Graph-Retrieval Da dieser Algorithmus in der Indizierungsphase auf einen Wissensgraphen angewiesen ist, lohnt sich der Ansatz besonders, wenn bereits ein solcher für Informationen existiert oder eine graphische Darstellung der Informationsverteilung des gesamten Datenkorpus gewünscht ist. Weiter ist der Ansatz aufgrund der Verarbeitung von Wissensgraphen geeignet für eine Informationsmenge, welche ähnliche oder gleiche Themen auf verschiedene Weisen betrachtet und analysiert. Da während der Informationssuche auf die Zusammenfassungen der Themen zugegriffen wird, in welchen Schlussfolgerungen aus in diesen enthaltenen Informationen vorkommen können, wird auf diese Weise der gesamte Informationskontext optimal ausgeschöpft. Hierbei muss erwähnt werden, dass zu große Datenmengen sowohl aufgrund der mehrfachen Speicherung in verschiedenen Formaten während der Indizierung, als auch der Wiederholung von Inhalten in den hierarchisch strukturierten Themengebieten und zuletzt aufgrund der damit verbundene

extreme Verarbeitungszeit zu Problemen führen können.

Naives Retrieval Anders als Graph-Retrieval ist das naive Retrieval-Verfahren auch auf große Datenmengen ausgelegt. Da die Informationen direkt in der Vektordatenbank gespeichert werden und die Umwandlung von Texten in Embeddings ein vergleichsweise schnelles Verfahren ist, kann dieser Algorithmus ohne Probleme auch auf größten Datenmengen agieren. Weiter ist die sehr schnelle Retrieval-Phase optimal für zeitkritische Anwendungen oder für Nutzer, die lediglich die Qualität der von LLMs generierten Antworten verbessern, aber keine lange Wartezeit riskieren wollen. Da, wie bereits in Kapitel 4.2 erklärt, dieses Verfahren dazu tendiert, nicht immer erfolgreich nach Informationen zu suchen, sollte dieser Algorithmus nicht verwendet werden, wenn die bereitgestellten Daten kritisch verarbeitet werden sollen. Insgesamt empfiehlt sich demnach die Anwendung in einem allgemeinem Umfeld, um die Ausgabe von LLMs für einen bestimmten Themenbereichen ohne viel Aufwand zu verbessern.

GARAG Im Kontrast zum Graph-Retrieval-Algorithmus ist GARAG auf die Authentizität von Informationen ausgelegt. Da die dem Nutzer bereitgestellten Informationen direkte Ausschnitte der Primärquellen sind, kann sich dieser auf die Korrektheit dieser - unter Vorbehalt der Richtigkeit der dem System übergebenen Daten - verlassen. Da weiter dieser Algorithmus für die Suche nach Informationen die thematischen Zusammenfassungen verwendet, wird auch hier ein, im Vergleich zum naivem Retrieval, hohes Maß an Textverständnis verwendet. Demnach ist dieses Verfahren optimal für kritische Anwendungen, welche auf fehlerfreie Daten angewiesen sind. Da, aufgrund des verwendeten Wissensgraphens, auch dieser Algorithmus einen schwer skalierbaren Speicherbedarf hat und die Indizierungsphase durch viele Daten extrem verlangsamt wird, muss dieser Ansatz mit großen Datenmengen vorsichtig verwendet werden. Ist die Indizierung der Daten allerdings fertiggestellt, so können auch zeitkritische Systeme auf den durch Embeddings beschleunigten Suchalgorithmus zugreifen.

4.4 Vergleich der Anforderungen der Algorithmen

In diesem Kapitel betrachte ich die softwaretechnischen Anforderungen der Verfahren. Hierbei ist Python sowie die Installation verschiedener Python-Pakete eine allgemeine

Anforderung, welche bereits in Kapitel 2.3 erläutert wurden. In diesem Kapitel werden die maßgeblichen Unterschiede der benötigten Software betrachtet, welche sich durch die benötigten externen Programme ergeben.

Graph-Retrieval Dieser Algorithmus beginnt die Datenverarbeitung mit der Erstellung eines Wissensgraphen aus Textabschnitten. Da hierbei auf ein großes Sprachmodell zugegriffen wird, ist zu der Graphdatenbank für die Speicherung auch eine Anbindung an ein LLM notwendig. Diese beiden Ressourcen werden für die weitere Indizierungsphase wiederverwendet, um Themengruppierungen und Zusammenfassungen dieser zu erstellen und abzuspeichern. Da die Datengrundlage somit vollkommen in der Graphdatenbank verbleibt und für die Suche nach Antworten ein Sprachmodell verwendet wird, sind diese Services auch bei Nutzeranfragen bereitzustellen. Da innerhalb des Verfahrens eine Vielzahl von Anfragen an das LLM gestellt werden, ist das Ausführen dieses auf einem System mit ausreichen Rechenleistung notwendig oder ein externer Dienst für diesen Zweck einzubinden. Aufgrund dieser Einschränkungen ist das Ausführen dieses Ansatzes nur für den Serverbetrieb zu empfehlen.

Naives Retrieval Die einfache Struktur dieses Verfahrens spiegelt sich auch in der technischen Anforderung wider. Dar naive Retrieval-Algorithmus verwendet eine spezialisierte Python-Bibliothek, um Textausschnitte in Embeddings umzuwandeln. Diese werden anschließend in einer externen Vektordatenbank abgespeichert. Auch bei Nutzeranfragen wird selbige Software wiederverwendet. Aufgrund der geringen Anforderungen und da das lokale Betreiben einer Vektordatenbank bis zu einer von der Rechenleistung abhängigen Größe kein Problem ist, bietet sich dieser Ansatz auch für die Benutzung auf dem eigenen Gerät an, wodurch keine Daten an Dritte weitergegeben werden können.

GARAG Anders als in den vorherigen Algorithmen beschrieben ändern sich bei diesem Verfahren die Anforderungen zwischen der Indizierungs- und Queryphase. Während der erstmaligen Datenanalyse ist ähnlich zum Graph-Retrieval-Algorithmus eine Graphdatenbank, sowie ein LLM auf einem System mit ausreichend Rechenleistung notwendig. Weiter werden auch in diesem Ansatz zum Schluss die vorliegenden Texte durch eine Python-Bibliothek in Embeddings umgewandelt, welche in einer Vektordatenbank abgespeichert werden. Da somit in dieser Phase viele verschiedene Services notwendig sind

und auch eine bestimmte Rechenleistung für das LLM benötigt wird, empfiehlt sich die Indizierung auf einem Serversystem. Ist diese Phase vollständig abgearbeitet, wird bei Nutzeranfragen keine Anbindung an LLMs mehr benötigt, sodass ein Wechsel auf ein lokales System möglich ist. Somit besitzt dieser Ansatz für die Indizierung mehr Anforderungen, während für die Suche nach Informationen die bleibenden Anforderungen auch für eine lokale Anwendung erfüllt werden können.

4.5 Ergebnis des Vergleichs

Graph-Retrieval Insgesamt weist Graph-Retrieval eine hohe Präzision in der Erfassung von Informationen auf. Diese ergibt sich vor allem aus der Verarbeitung aller Themata durch große Sprachmodelle, welche im allgemeinen ein hohes Maß an Textverständnis bereitstellen. Da die Informationsverarbeitung auf Wissensgraphen basiert, ist dieser Ansatz optimal für die Analyse von Daten, die bereits in dieser Form vorliegen. Weiter bietet diese Methode dem Nutzer über die Auswahl der allgemeinen Größe der verwendeten Themengebiete eine Möglichkeit zur Anpassung der Informationssuche. Dieser Wert bestimmt hierbei etwa das Verhältnis zwischen originalen Informationen und Schlussfolgerungen aus der unterliegenden Struktur des Wissensgraphen. Da der Einsatz dieser Software auch während der Datensuche auf LLMs angewiesen ist, bietet sich besonders die Integration des Systems in einen serverbetriebenen Service an. Da die Authentizität der bereitgestellten Informationen nicht vollständig gewährleistet werden kann, sollten die hieraus bezogenen Daten vor Verwendung manuell überprüft werden.

Naives Retrieval Im Gegensatz zum Graph-Retrieval kann das naive Retrieval aufgrund der geringen Leistungsanforderung lokal betrieben werden. Da ferner die Informationen nicht durch Sprachmodelle verarbeitet werden, erhält ein Nutzer aus diesem System Ausschnitte aus originalen Primärquellen. Da die Suche nach Daten in diesem Ansatz lediglich auf dem Verglich von der Vektordarstellung einer Anfrage mit den Embeddings der Originaldaten beruht, werden teils wichtige Informationen nicht identifiziert oder unwichtige Datenpunkte als erstere extrahiert[1]. Dieses Phänomen kann durch die Erweiterung des Ansatzes mit verschiedenen anderen Algorithmen reduziert werden. Einige dieser Methoden werden von Y. Gao, et al. in „Retrieval-Augmented Generation for Large Language Models: A Survey“ (vgl. [1]) vorgestellt. Insgesamt lohnt sich dieser

Ansatz, um ohne viel Aufwand den Einsatz von Sprachmodellen zu verbessern.

GARAG Dieser Algorithmus zeichnet sich im Vergleich zum Graph-Retrieval durch eine schnelle Verarbeitung von Nutzeranfragen aus, welche auf einem dem naivem Retrieval ähnelnden Verfahren beruht. Da durch den hierbei verwendeten Mechanismus Primärquellen mit für die Anfrage wichtige Informationen identifiziert und zurückgegeben werden, ist auch hier die Datenauthentizität sichergestellt. Da allerdings, anders als bei der Verarbeitung von Anfragen durch einem naivem Retrieval, das Embedding dieser Eingabe mit allen Knoten des Wissensgraphen einzeln und den Beschreibungen der Themengebiete abgeglichen wird, bildet diese Suche eine Analyse auf verschiedenen Interpretationsstufen der Daten gleichzeitig. Somit kann sowohl auf aus der Struktur des Wissensgraphen erschlossenen Folgerungen, als auch auf den einzelnen Datenpunkten direkt nach Informationen gefiltert werden. Hierbei wirken die Kleinschrittigkeit der in diesem Graphen enthaltenen Datenpunkte und die rekursiv hierauf aufbauenden Interpretationen verschiedener Größe der fehlerhaften Interpretation dieser, wie sie bei einem naivem Retrieval passieren kann, entgegen. Insgesamt bietet sich dieser Ansatz aufgrund der geringen Verarbeitungszeit von Anfragen für eine Verwendung als Software für die Suche nach Informationen an. Aufgrund der für die Indizierung benötigten Ressourcen und externen Programme ist der Einsatz des Algorithmus auf einem Server empfehlend. Während schließlich die Datenabfrage weniger Rechenleistung verbraucht und bei angemessener Datengröße auch lokal ausgeführt werden kann, ist auch hier ein Serversystem für eine zentrale Bereitstellung vorteilhaft.

5 Sicherstellung der Nutzerfreundlichkeit

Im Zuge der Praxisarbeit wurde ein bestehendes Programm maßgeblich erweitert. Um dieses nun für Anwenderinnen und Anwender besser nutzbar zu machen, ist eine einfache Bedienung dessen sicherzustellen. Hierbei liegt der Fokus auf die Verwendung der Retrieval-Algorithmen und einer einfachen Indizierung von Informationen.

5.1 Schnittstelle für Datenverteilung

Da das Programm nun verschiedene Retrieval-Algorithmen aufweist, welche von Nutzenden verwendet werden können, muss dem Nutzer oder der Nutzerin eine gut verständliche Möglichkeit zur Auswahl des gewollten Ansatzes geboten werden. Hierfür wird eine Klasse bereitgestellt, welche die Algorithmen und die von diesen benötigten Prozesse verwaltet.

Hierbei werden die verschiedenen Ansätze als öffentliche Methoden unter ihrem Namen bereitgestellt. Weiter werden private Methoden für die Verbindung mit externen Services implementiert. So verwalten die Algorithmen die Verbindung mit der Vektordatenbank und Graphdatenbank, sowie die Anbindung eines großen Sprachmodells zentral, was eine leichte Erweiterung des Systems ermöglicht. Diese externen Verbindungen können durch die Übergabe einer Konfigurationsdatei von einem Benutzer schnell personalisiert werden. Diese ermöglicht weiter die Modifikation der Anzahl an parallel ablaufender Prozesse, welche besonders die Informationsfindung durch Sprachmodelle, wie sie in dem ursprünglichen Algorithmus durchgeführt wird, beeinflusst.

5.2 Einstellungsmöglichkeiten des Projekts

Um das Verhalten der Algorithmen und die Anbindung externer Services zentral zu verwalten, verwendet dieses Projekt eine Konfigurationsdatei. Diese muss von jedem Nutzer vor erstmaligem Einsatz ausgefüllt werden und beeinflusst damit extrem die Einstiegshürde für die Verwendung der Software.

Hierbei ist es notwendig die Anzahl an Konfigurationsmöglichkeiten mit der erleichterten Übersicht, welche durch eine geringe Anzahl an Einstellungsmöglichkeiten erreicht werden kann, abzuwägen. Da das bestehende Programm eine Vielzahl austauschbarer Methoden für die Indizierung der Daten bereitstellte, zeigt auch die Konfigurationsdatei eine große

Anzahl an Werten auf. Durch verschiedene kleinere Tests haben sich hierbei bestimmte Methoden in der Indizierungsphase bewährt, welche weiter verbessert wurden. Demnach ist das massive Ausmaß der Einstellungsmöglichkeiten des alten Programms von 45 Feldern vor allem auf fehlende Aktualisierung dieser zurückzuführen. Durch das Entfernen der somit nicht mehr benutzten Konfigurationen kann das Ausmaß auf 14 Werte verringert werden.

Um weiter die Erstkonfiguration zu erleichtern wurde eine ausführliche Beschreibung der einzelnen Felder und ihrer Standardwerte angefertigt. Kombiniert mit einer vollständigen Beispielkonfiguration können Nutzer das Projekt schnell personalisieren.

5.3 Kurzanleitung für die Programmbedienung

Da das System sowohl auf den Betrieb auf einem Server mit hierfür automatisch durch Docker verwalteten Datenbankinstanzen, als auch für den Einsatz mit bereits existierenden Services ausgelegt ist, bietet das Projekt eine Vielzahl an Bedienmöglichkeiten. Um den Nutzer oder die Nutzerin bei der Entscheidung des zu verwendenden Softwareaufbaus zu unterstützen, wird eine Kurzanleitung bereitgestellt, welche die Unterschiede einer hierfür verwalteten Datenbank zu externen Services darlegt. Entsprechend der Wahl der Anwendenden geht diese anschließend auf die benötigten Vorbereitungsschritte ein und beschreibt somit den benötigten Ablauf, bis das Programm gestartet werden kann. Um auch das Programm selber für den Start zu vereinfachen, werden weiter die für die Indizierung benötigten Ausführungsschritte in einem Script zusammengefügt, welches somit selbstständig die vollständige Datenverarbeitung vornimmt.

6 PDF Converter

Wie in Kapitel 3.3.1 erläutert, verbessert die Umwandlung von PDF-Dateien in das Markdown-Format die Qualität der somit extrahierten Informationen. Für diesen Zweck können verschiedene externe Python-Bibliotheken verwendet werden. Hierbei ist es für die Authentizität der Daten wichtig, dass der Text direkt aus der Datei gelesen wird. Eine weit verbreitete Alternative hierzu, die Verwendung von Optical Character Recognition (OCR), verwendet KI um Text aus einem Bild zu extrahieren. Da auf diese Weise keine Garantie für die Übereinstimmung der erhaltenen Daten mit den originalen Informationen besteht, riskiert die Umwandlung durch ein OCR-Modell die Authentizität dieser.

6.1 Übersicht über verschiedene Python-Bibliotheken

PyPDF2 Vor der Umstellung von Klartext auf das Markdown-Format wurde für die Convertierung von PDF-Dateien das Python-Modul `PyPDF2`[12] verwendet. Dieses Modul ist ein OpenSource-Projekt mit verschiedenen Manipulationsmöglichkeiten für PDF-Dateien[20]. Da diese Bibliothek allerdings nicht den Inhalt in dem Markdown-Format extrahieren kann, wird dieses Modul nicht mehr weiter verwendet.

Docling Wie bereits in Kapitel 3.3.1 erwähnt, können durch `Docling`[13] verschiedene Dateiformate in das Markdown-Format übersetzt werden. Auch PDF-Dokumente können so erfolgreich verarbeitet werden und können so in aufschlussreiche formatierte Textdateien konvertiert werden. Hierbei konnte allerdings `Docling` in vielen Dokumenten keine hierarchische Relation zwischen Kapitelüberschriften und Unterüberschriften feststellen und fett geschriebener Text nicht immer erkennen. Da letztere Informationen besonders in Tabellen für eine Unterteilung und Subtabellen interessant ist und erster für eine bessere Unterteilung des Dokumentinhalts verwendet werden kann, wird nach einer besseren Lösung gesucht.

Marker Eine Alternative zu `Docling` ist das Python-Modul `Marker`[14]. Während diese Bibliothek neben verschiedenen Formatierungen auch Referenzen richtig in Markdown

[12] `https://pypi.org/project/PyPDF2/`
[13] `https://pypi.org/project/docling/`
[14] `https://pypi.org/project/marker-pdf/`

übersetzen kann, hat auch `Marker` ein Problem mit der korrekten hierarchischen Abbildung von Überschriften. Da weiter diese Bibliothek aufgrund der hierbei verwendeten GPL-3.0 Lizenz[21] nur unter verschiedenen Einschränkungen nutzbar ist, wird dieses Modul in dem Projekt nicht verwendet.

MarkItDown Eine weitere häufig verwendete Alternative stellt die Bibliothek `markitdown`[15] dar. Auch diese kann verschiedene Dateiformate einlesen und in das Markdown-Format übertragen. Einige Tests mit dieser Bibliothek haben gezeigt, dass die Umwandlung häufig verschiedene Details verliert. So weist der extrahierte Text der Wissenschaftlichen Arbeit „From Local to Global: A GraphRAG Approach to Query-Focused Summarization“ von D. Edge et al. (vgl [7]) keine formatierten Überschriften oder erkannten Tabellen auf. Da sich dieses Ergebnis auch bei mehreren weiteren Dateien wiederholt hat, genügt auch dieses Python-Modul nicht den Anforderungen.

Pandoc Für die Umwandlung verschiedener Dateitypen in Python wird häufig die Bibliothek `Pandoc`[16] erwähnt. Dieses Programm ist in der Lage mit über 60 verschiedene Formate zu arbeiten. Hierbei können die meisten dieser sowohl als Quell-, als auch als Ausgabeformat verwendet werden[22]. Da hierbei nur in das PDF-Format umgewandelt, also keine Daten aus diesen gelesen werden kann, erfüllt auch diese Bibliothek nicht die Anforderungen.

PdfPlumber und PdfMiner Diese beiden Python-Bibliotheken[17,18] sind zusammen in der Lage detaillierte Informationen aus PDF-Dateien zu extrahieren. Während `7` einzelne Zeichen aus dem Dokument extrahieren kann, fügt die hierauf aufbauende Bibliothek `PdfPlumber` diese direkt zu Worten zusammen. Diese müssten in einem nächsten Schritt manuell angeordnet werden, wobei hierbei Überschriften und Tabellen identifiziert werden sollen. Weiter sind beide Bibliotheken nur in der Lage Dokumente bis zu der PDF-Version 1.7 zu verarbeiten[23]. Da somit diese Module verschiedene große Einschränkungen bieten, wird nicht auf diesen aufgebaut.

Da keine der hier vorgestellten, bereits bestehenden, Lösungen den Ansprüchen für dieses

[15] `https://pypi.org/project/markitdown/`
[16] `https://pypi.org/project/pandoc/`
[17] `https://pypi.org/project/pdfplumber/`
[18] `https://pypi.org/project/pdfminer/`

Projekt entspricht, soll die Konvertierung durch eine hierfür neu geschriebene Software übernommen werden.

6.2 Der PDF-Standard Allgemein

Das Portable Document Format wurde von dem Unternehmen Adobe entwickelt und später von der Internationale Organisation für Normung (ISO) als Standard aufgenommen[24]. Als Dateiformat, entwickelt für das Verteilen von Informationen[24], findet dieser Standard fast überall eine Anwendung. Die aktuellste Version, PDF 2.0, wird von der ISO in einem etwa 1000-seitigen PDF-Dokument vorgestellt[25].

PDF-Dateien können hierbei Text in verschiedenen Schriftarten, Formen und Bilder abbilden. Weiter können sie interaktive Elemente, wie zum Beispiel Videos, Links, Formulare und Knöpfe integrieren. [25]

Um nun aus PDF-Dateien textuelle Informationen zu extrahieren, müssen die im Dokument enthaltenen Glyphen korrekt in Zeichen konvertiert und auf der Seite positioniert werden. Anschließend werden diese entsprechend ihrer Position direkt interpretiert oder in einer Tabelle eingefügt.

6.3 Aufbau einer PDF-Datei

Eine PDF-Datei ist in vier Abschnitte unterteilt. Einem Header, welcher die Version des Standards angibt, einen Hauptteil, der eine ungeordnete Sammlung von Objekten darstellt, und das Dateiende, geteilt in eine Tabelle für die Referenzierung der Objekte und einem Speicher für allgemeine Informationen. [25]

Weiter können folgende Datentypen in diesem Format abgebildet werden[25]:

> Boolesche-Werte, positive und negative Ganzzahlen und reelle Zahlen, Zeichenketten, Namen, Arrays, Dictionaries, Null-Werte und Datenstrings

Um nun Daten zu extrahieren, wird zuerst das sogenannte Trailer-Dictionary am Ende der Datei gelesen. Dieses gibt an, welches Objekt die Daten des gesamten Dokuments verwaltet. Weiter wird eine eventuelle Verschlüsselung der Datei hier behandelt, welche bisher noch nicht durch den verfassten Parser entschlüsselt werden kann.

Als nächstes werden die Positionen der Objekte durch das Einlesen der Referenztabellen

festgestellt. Diese kann durch die in PDFs vorhandene Versionierung an mehreren Stellen aufgeteilt sein und muss vollständig zusammengeführt werden. Weiter kann diese Tabelle in einer direkten Form oder als Inhalt eines Datenstrings spezifiziert werden[25]. Diese Dynamik muss in einem Parser implementiert sein.

6.4 Einlesen einzelner Datenpunkte des PDF-Formats

Um Werte aus der Datei einzulesen, wird die umzuwandelnde Datei als ein Bytestream eingelesen. Aus diesem können nun durch einen hierfür geschriebenen lexikalischen Scanner einzelne Tokens eingelesen werden. Diese werden für Arrays, Dictionaries und Datenstrings in einem nächstem Schritt zu komplexen Objekten zusammengefügt.

Um Tokens aus dem Dateistream zu extrahieren, werden solange Zeichen gelesen, bis eines der im Standard definierten Tokens vollständig ist. Hierfür wurde ein deterministischer, endlicher Automat implementiert.

Da Datenstrings meist komprimierte Daten abspeichern, müssen diese beim Lesen erneut entpackt werden. Hierbei werden von dem Standard sechs verschiedene Filter spezifiziert [25], wobei vor allem der auf dem zlib/deflate Algorithmus aufbauende FlateDecode[25] häufig verwendet wird. Dieser ist in dem verfassten Parser implementiert.

6.5 Einlesen einer PDF-Datei

Die Daten einer PDF-Datei werden in einem hierarchischen Objektaufbau abgespeichert. Hierbei ist das Hauptelement jeder solchen Datei das sogenannte „Document Catalog Dictionary“. Dieses enthält Informationen über mögliche Erweiterungen, die Seiten des Dokuments und eine Definition der Numerierung dieser, Zuweisungen von intern verwendeten Namen zu Objekten und zu Zielorten, Einstelllungen für das Anzeigen des Dokuments, Metadaten, Sprechspezifikation und weitere. [25]

Aus den Attributen dieses Objekts werden ausgewählte, notwendige Informationen extrahiert. So stellt die Outline eine Leiste zur einfachen Navigation eines PDFs dar, durch welche Überschriften mit entsprechenden Seiten verbunden werden. Da diese Leiste die verschiedenen Titel hierarchisch sortiert, kann dieses Objekt genutzt werden, um auf die Struktur des Dokumenteninhalts zu schließen. Ein weiteres Attribut kann den strukturellen Aufbau des Inhalts wiedergeben. Durch dieses können Texte als Überschriften

unterschiedlicher Ebenen markiert werden. Dieses Feature wird als Alternative zur vorher erwähnten Outline extrahiert. Da der verfasste PDF-Parser Text mit optionalen, extern bereitgestellten Seitentrennern erzeugt, wird ein weiteres Attribut verwendet, um personalisierte Seitenkennungen zu implementieren. Diese drei Werte des Document Catalog Dictionary werden optional von dem Verfasser oder der Verfasserin bereitgestellt[25]. Demnach muss es für die hierdurch beeinflussten Systeme alternativen geben, welche nicht auf diese Werte zurückgreifen. Auf den Umgang mit diesen Daten wird in den folgenden Kapiteln genauer eingegangen.

Das letzte, bisher durch den Parser betrachtete, Attribut hält die Seiten des Dokuments in einer Baumstruktur. Letztere wird verwendet, um verschiedene Standardwerte für einen Bereich an Seiten zu konfigurieren[25]. Die Blätter dieses Baums halten letztendlich die Daten der PDF-Seiten. Auch bei diesen handelt es sich um Objekte, welche verschiedene Attribute beinhalten. Durch diese wird die Größe der Seite, weitere verwendeten Ressourcen, wie zum Beispiel Bilder und Schriftarten, die Ausrichtung der Seite, Anmerkungen und weiteres Konfiguriert[25]. Auch hier werden ausgewählte Attribute interpretiert. Hierbei werden die externen Ressourcen vorbereitet, Anmerkungen interpretiert der Text der Seite gelesen und Normalisierung hierauf angewendet. Da allerdings der Inhalt einer Seite in einem sogenannten „Content Stream“ abgespeichert wird, muss auch dieser dekodiert werden.

6.6 Einlesen der Daten eines Content Stream

Ähnlich zu dem gesamten PDF-Dokument, ist der darin verwendete „Content Stream“ aus selbigen Typen an Objekten aufgebaut. Da der „Content Stream“ allerdings als eine Liste von Funktionen gesehen werden kann, welche letztendlich die entsprechende Seite für einen Benutzer sichtbar zeichnen soll, sind dort über 70 verschiedene Schlüsselwörter vertreten. Anders als bei vielen anderen Dateiformaten wird der Inhalte in einem Content Stream nicht zwingend in der korrekten Lesereihenfolge abgespeichert. Anstelle hiervon gleicht dieser einer Aneinanderreihung von Zeichenbefehlen, wobei jeder mit einer exakten Position ausgestattet ist. Diese ist gegeben durch mehrere Matrizen, welche zusammengenommen eine anzuzeigende Glyphe an die korrekte Position auf der Seite transformieren. Die somit für die Bestimmung der Position eines Zeichens wichtigen Matrizen werden durch verschiedene Befehle auf unterschiedliche Weise justiert und automatisch

beim Zeichnen weitergeschoben. Weitere Befehle werden verwendet, um die Abbildung eines Elements farbig zu verändern, Linien zu ziehen und Formen zu bilden. Schließlich kann auch auf externe Objekte, wie zum Beispiel Annotationen oder Strukturelemente referenziert, zugegriffen oder diese direkt gezeichnet werden. [25]

Um letztendlich Glyphen zu zeichnen, können vier verschiedene Operatoren verwendet werden, welche alle den entsprechenden Text als Zeichenkette erhalten. Da allerdings das PDF-Format maßgeblich auf die geräteunabhängige Abbildung von Dokumenten abzielt, können diese Texte nicht direkt interpretiert werden. Stattdessen sind diese immer nur mit der für die Glyphe verwendete Schriftart zu entschlüsseln. [25]

6.7 Schriftarten in PDFs

Der PDF-Standard unterstützt folgende Schriftarten: Type 0, Type 1, Type 3, TrueType und CIDFont [25]. Von diesen wurden in dem geschriebenen Parser die häufig verwendeten Type 0, Type 1 und TrueType implementiert.

Eine Schriftart wird in einem PDF als ein Objekt dargestellt, welches unter anderem die Breite jedes verwendeten Zeichens und eine Kodierungstabelle für Zeichencodes sowie optional verschiedene standardisierte charakteristische Abmessungen, die Schwere der Zeichen und eine Datei, welche die Umwandlung zu Unicode-Zeichen vereinfacht, bereitstellt. Diese Attribute können verwendet werde, um die tatsächlichen Abmessungen eines Zeichens in der visuellen Darstellung zu ermitteln. Weiter liefern diese Daten weitere Informationen, wie die Kursivität oder das Fettsein der Schrift.

Grundsätzlich werden während der Zeichnung der Glyphen der hierfür verwendete Datenstring als Kette an Zeichencodes interpretiert. Diese werden durch eine Tabelle in Zeichen-IDs umgewandelt, welche wiederum in Glyph-IDs transformiert und anschließend für die Indizierung des zu zeichnenden Glyphen verwendet werden. Um eine Umwandlung in Unicode vorzunehmen, kann eine optional der Schriftart beigelegte CMap-Datei gelesen werden. Diese verknüpft Zeichencodes direkt mit den entsprechenden Unicode-Zeichen. Alternativ muss die entsprechende Schriftdatei manuell eingelesen und hieraus eine Umwandlung erstellt werden. Letztere Möglichkeit wurde noch nicht in der Parser integriert.

Ein weiteres zu interpretierendes Feature der Schriftart ist die Kodierungstabelle, welche Zeichencodes in Zeichen-IDs umwandelt. Auch wenn die hierdurch erstellten Identifikatio-

ren keine Verwendung in der Umwandlung nach Unicode finden, wird diese Konvertierung verwendet, um die Anzahl an für das Zeichen notwendigen Bytes in der Kette an Zeichencodes festzulegen. Diese Tabelle kann als eine eingebettete CMap-Datei vorliegen oder alternativ einer von mehreren im PDF-Standard genauer bezeichneten vordefinierten Tabellen sein[25].

Als Alternative zu den meisten der für Schriftarten erwähnten Eigenschaften kann auch eine von 14 Standardschriftarten über den Namen dieser referenziert werden. Die Attribute dieser sollen bereits vordefiniert fest in einem Parser integriert sein. Diese Standardschriftarten sind verschiedene Abwandlungen der Zeichensätze `Times`, `Helvetica`, `Courier`, `Symbol` und `ZapfDingbats`. Ist eine dieser Schriftarten referenziert, können viele Attribute fehlen, welche von einem Parser mit intern hinterlegten Werten zu füllen sind. Ab dem aktuellem PDF-Standard, PDF 2.0, werden auch diese Werte nun in der Datei bereitgestellt. [25]

Aus der somit großen Anzahl an verschiedenen Standardwerten und -schriftarten wurden jene bereits integriert, welche bei den bisher durch den Parser verarbeiteten Dokumenten bereits vorkommen.

6.8 Einlesen einer CMap-Datei

Der Aufbau einer CMap-Datei wird durch die „Adobe Technical Note #5014“ (vgl. [26]) beschrieben [25]. Die in dieser Datei verwendeten lexikalischen Elemente entsprechen im groben denen einer PDF-Datei. Weiter werden auch hier verschiedene Schlüsselwörter verwendet, welche verschiedene Abschnitte der Datei klassifizieren. Für die Übersetzung dieser in lexikalische Tokens wird auch hier ein deterministischer, endlicher Automat implementiert.

Der Inhalt einer CMap-Datei lässt sich grob in drei Sektionen aufteilen. Zu Beginn werden verschiedene allgemeine Informationen über den Inhalt der Datei bereitgestellt. Diese dienen zum Beispiel der Charakterisierung der in diesem Dateiformat deklarierten Schriftart und sind für die Extraktion einer Kodierungstabelle nicht relevant. Anschließend werden valide Codebereiche festgelegt. Da eine CMap auch eine Kodierung von Bytesequenzen vornehmen kann, werden in diesem Abschnitt somit auch alle möglichen Codelängen festgeschrieben. Hierbei werden Bereiche für Multibyte-Codes als Rechtecke über zwei

angegebenen Eckcodes beschrieben. [26]

In dem dritten Abschnitt der Datei finden sich schließlich die Übersetzungstabellen der zuvor spezifizierten Codebereiche. Diese können entweder einzeln für jeden Code angegeben werden oder als Folge aufeinanderfolgender Zeichen. [26]

Die so gesammelten Tabellen und Codebereiche werden entsprechend der Codelänge sortiert, abgespeichert und zusammengefasst. Soll ein Zeichen durch den Einsatz einer CMap dekodiert werden, so werden nacheinander für die verschiedenen möglichen Codelängen geprüft, ob diese Bytefolge innerhalb einer der angegebenen Bytebereichen liegt. Wird eine valide Länge gefunden, wird anschließend der zugeschnittene Zeichencode durch die hierfür vorliegende Tabelle in eine Zeichen-ID oder ein Unicode-Zeichen übersetzt.

6.9 Strukturierungsdaten in PDFs

Aus der Kombination der vorher erwähnten, eingelesenen Daten können einzelne Zeichen mit ihrer jeweiligen Position und Größe aus einem PDF extrahiert werden. Um diese weiter zu verarbeiten, können verschiedene optionale Elemente verwendet werden, die die Struktur einer PDF-Datei abbilden.

6.9.1 Dateiübersicht

Um die Navigation durch lange Dokumente zu vereinfachen bieten viele dieser eine Übersicht an. Diese wird bei dem Öffnen einer PDF-Datei meist an der Seite angezeigt und kann verwendet werden, um an verschiedene Stellen der Datei zu springen. Hierbei wird die Dateistruktur durch die dort verwendeten Überschriften hierarchisch dargestellt.

Innerhalb einer PDF-Datei wird diese durch einen baumartigen Aufbau an Objekten dargestellt. Hierbei speichert jedes Objekt unter anderem die anzuzeigende Überschrift und die auszuführende Aktion, welche durch einen Mausdruck aktiviert wird. Diese kann zum Beispiel eine Sprungaktion darstellen, welche wiederum eine Referenz auf eine Seite des Dokuments sowie die Position und Skalierung dieser angeben kann. [25]

Auf diese Art können Kapitelüberschriften verschiedener hierarchischer Stufen akkurat gefunden werden, indem für jedes in dem Übersicht enthaltenes Element nach dem Text der Überschrift auf der mit dieser verbundenen Seite gesucht wird.

6.9.2 Strukturhierarchie

Eine weiteres dieser optionalen Features ist die Strukturhierarchie. Diese bildet einen Baum über das gesamte Dokument ab, welches verschiedene Elemente, bzw. Gruppen an Elementen unterschiedliche Funktionen zuordnet. Hierbei bietet das PDF verschiedene Standardtypen an, welche in vier Stufen eingeteilt werden können: [25]

- **Dokument**: Diese Typen werden verwendet, um zusammengeführte Dokumente oder aus diesen Eingefügte Inhalte zu markieren. So kann hierdurch eine PDF-Datei in ihre ursprünglichen Dokumente unterteilt werden.
- **Gruppe**: Diese Typen organisieren das Dokument in zusammengehörige Abschnitte. So ist der Inhalt einer Gruppe zum Beispiel ein Kapitel oder Abschnitt in einem Dokument.
- **Block**: Diese Typen spezifizieren einen Textausschnitt durch verschiedene Eigenschaften. Beispiele hierfür sind verschiedene Stufen an Überschriften oder Absätze,
- **Zeilenintern**: Eine weitere kleinschrittige Unterteilung bieten diese Typen. Hier finden sich zum Beispiel Typen für die Beschriftung von Abbildungen oder Betonung von Text.

Diese Typen verwenden standardisierte Bezeichner, welche von der ISO vergeben sind. Zusätzlich können weitere nicht standardisierte Namen für weitere strukturelle Informationen verwendet werden. Diese können durch eine in der PDF-Datei enthaltenen Übersetzungstabelle in grob entsprechende Standardwerte umgewandelt werden. [25]

Da durch diesen Mechanismus die Bedeutung der Elemente nicht vollständig eindeutig ist, das Feature generell optional ist und, wie sich in einigen Tests gezeigt hat, viele Dokumente keine Unterteilung in Kapitel oder andere Elemente darstellen, wird diese Hierarchie lediglich zur Findung von Kapitelüberschriften verwendet, wenn zuvor beschriebene Dokumentübersicht fehlt.

6.10 Zusammensetzen von Worten

Um die auf einer Seite vorhandenen Zeichen effizienter zu bearbeiten, werden diese zu Worten zusammengesetzt. Hierbei wird jedes Zeichen mit einer Schätzung für den Anfang eines auf dieses direkt folgenden Wortes ausgestattet. Diese wird entsprechend der Ro-

tation und Größe des Zeichens sowie der Eigenschaften der für das Zeichen verwendeten Schriftart bestimmt.

Durch diese zusätzliche Information zeichnen sich Buchstaben eines Wortes durch einen geringeren Abstand auf selbiger Höhe aus. Weiter werden weitere Eigenschaften, wie die verwendete Schriftgröße, Rotation, Betonung und Verweis auf Strukturelemente, abgeglichen. Stimmen visuell aufeinanderfolgende Zeichen in ihren Eigenschaften überein, so werden diese zu einem Wort kombiniert.

6.11 Erkennen von Tabellen

Um in einem PDF-Dokument eine Tabelle zu erkennen gibt es mehrere mögliche Ansätze. Da durch verschiedene Befehle Linien auf einer Seite gezeichnet werden, können zum Beispiel diese als Orientierung für Tabellen dienen.

Setzt man allerdings auf den Einsatz von Linien, um Tabellen zu erkennen, scheitert dieser Ansatz bei vielen modernen Repräsentationen, bei welchen häufig einzelne Spalten oder Zeilen nicht mehr explizit getrennt werden. Weiter werden Linien teilweise auch als visueller Feature, zum Beispiel für die Trennung von Kapiteln, verwendet. Weiter werden Linien in einem PDF-Dokument auf verschiedene Weisen gezeichnet. Entweder diese wird direkt durch das Zeichnen einer Linie dargestellt, oder es wird eine Form gefüllt, welche eine Linie darstellt. Da somit relativ viel Varianz in dem Einsatz von Linien liegt, werde ich diese nicht für die Erkennung verwenden.

Um trotzdem mögliche Spaltentrennungen zu erkennen, werden virtuelle Linien, links an ein jedes Wort anschließend, gezeichnet. Diese werden anschließend vertikal so weit wie möglich erweitert, wobei kein anders Wort durch diese Trenner gekreuzt werden darf. Anschließend werden die möglichen Spaltenlinien entsprechend einer minimalen Länge gefiltert, um einzelne Wörter in einem Textparagraphen zu ignorieren. Diese möglichen Spaltentrenner werden in einem nächsten Schritt entsprechend ihrer Positionierung auf der Seite zu Tabellen zusammengefügt, wobei eine solche als eine Nebeneinanderreihung von mindestens drei Spalten definiert ist. Durch diesen Ansatz werden Tabellen erkannt, welche vorrangig linksbündigen Textinhalt beinhalten. Da somit bisher nur die linken Spaltentrenner betrachtet wurden, fehlt für die Tabelle noch ein Wert für die Ausdehnung in der Breite. Hierfür wird jene nächstmögliche Linie hinter der Tabelle gezogen, sodass

auf der gesamten Höhe dieser kein Wort durchschnitten wird. Insgesamt erhält man auf diese Weise Tabellen mit der Position der Spalten und der Gesamtgröße. Sollten sich Tabellen überschneiden, wird die Größe der kleineren dieser angepasst, sodass sich diese nicht mehr kreuzen.

Anschließend werden die in der Tabelle enthaltenen Wörter dieser hinzugefügt. Um nun die noch fehlende Aufteilung in Zeilen zu erhalten, werden virtuelle Linien unterhalb jedes Wortes gezogen. Hierbei werden jene wieder entfernt, welche andere Wörter in der Tabelle kreuzen. Somit können die einzelnen Zeilen identifiziert und anschließend die Tabelle als ein zweidimensionales Array abgespeichert werden.

6.12 Zusammensetzen von Absätzen

Alle Wörter, welche nicht Teil einer Tabelle sind, werden anschließend in Absätze zusammengesetzt. Unter diesen versteht das Programm Gruppen von Wörtern, welche keine maßgebliche Lücke in Spalten- oder Zeilenform haben.

Diese können konstruiert werden, indem ein Rahmen um ein einzelnes Wort gelegt wird, welcher dynamisch mit anderen Elementen in der direkten Umgebung wächst, während Objekte mit höherer Distanz zu einer Rahmenkante ignoriert werden.

Durch diesen Ansatz werden auf eine einfache Weise sowohl nebeneinander, als auch übereinander geschriebene Absätze korrekt identifiziert, solange der erlaubte Abstand korrekt konfiguriert ist. Dieser wird in der Horizontalen durch eine vergrößerte Version des Wertes gewählt, welcher auf den Anfang des nächsten Wortes zeigt, während in der Vertikalen Wörter mit bis zu einem zweifachen Zeilenabstand als Elemente eines Absatzes gelten.

6.13 Strukturierung des Textes

Wie bereits in Kapitel 6.9 erläutert, können PDFs verschiedene Strukturelle Informationen aufweisen. Um diese nun in die Textdarstellung zu übertragen, müssen die entsprechenden Daten auf die richtigen Wörter übertragen werden. Hierfür werden entsprechend der Art der Strukturinformationen verschiedene Algorithmen verwendet.

6.13.1 Interpretation der Dateiübersicht

Ist eine Dateiübersicht vorhanden und eingelesen, so wird diese für die Identifikation von Überschriften verwendet. Hierbei ist zu beachten, dass zu diesem Zeitpunkt lediglich in Tabellen oder Absätzen gruppierte Worte pro Seite verfügbar sind. Demnach kann nicht direkt nach dem entsprechend in der Überschrift enthaltenen Text gesucht werden.

Zuerst wird für die Identifikation die Seite des Dokuments mit der von einer Überschrift referenzierten abgeglichen. Stimmt diese überein, so muss der angegebene Titel auf der Seite verfügbar sein. Um diesen zu finden, wird die Überschrift in die einzelnen Wörter aufgeteilt und alle Vorkommen dieser auf der entsprechenden Seite identifiziert. Anschließend werden alle möglichen Kombinationen der gefundenen Worte generiert, bei denen die Reihenfolge der Worte in der Überschrift mit der der entsprechenden Elemente auf der Seite übereinstimmt. Aus diesen Titeloptionen wird schließlich jene ausgewählt, bei der die Worte eine minimale Distanz voneinander aufweisen. Auf diese Weise wird die beste Titeloption auf einer Seite identifiziert.

Da häufig in der Dateiübersicht die Überschriften einzelner Kapitel ohne eine Nummerierung aufgeführt sind, welche im Text allerdings enthalten ist, werden zuletzt alle Worte in der entsprechenden Textzeile zu einer Überschrift der der Übersicht zu entnehmenden Stufe erklärt.

6.13.2 Interpretation der Strukturhierarchie

Als Alternative zu der Dateiübersicht wird die Strukturhierarchie verwendet. Anders als bei vorherigen Daten werden dieser Strukturinformationen direkt von Elementen auf der Seite referenziert. Dementsprechend besitzen alle Worte ein optionales Feld für die Identifikation eines solchen Strukturelements.

Um nun aufgrund dieser eine Identifikation von Überschriften vorzunehmen, kann über jedes Textelement mit solcher Referenz iteriert werden. Hierbei wird das entsprechende Strukturelement identifiziert. Anschließend wird entsprechend des Typen dieses Elements der verwendende Text entweder als Überschrift markiert oder weiter Informationen, wie zum Beispiel Betonungen, als Fettsein übertragen.

6.13.3 Schätzen einer Textstruktur

Da, wie in Kapitel 6.9 erwähnt, obige Informationen optionale Elemente darstellen, muss weiter ein System für die Identifikation von Überschriften auf Grundlage des Textes implementiert werden. Hierfür wird eine statistische Methode verwendet, welche die Größe der Wörter in dem Dokument analysiert. Diese zählt zu Beginn die Anzahl an Zeichen in den vorhandenen Textgrößen sowie die Gesamtanzahl aller Buchstaben im Dokument.

Anschließend werden den entsprechenden Größen eine Überschriftsstufe zugewiesen. Hierbei werden alle Buchstabengrößen als normaler Text identifiziert, bis die Anzahl dieser zusammengerechnet über 50 Prozent der Gesamtmenge ausmacht. Die niedrigste Überschriftenstufe erhalten anschließend jene Buchstaben, welche zusammen erneut über 50 Prozent des restlichen Vorrats ausmachen. Dies wiederholt sich, bis alle Textgrößen annotiert sind. Anschließend werden den Texten diese Stufen entsprechend ihrer Größe zugeordnet.

Dieser Algorithmus gewährleistet, dass die Größe, welche die Hauptmenge des Dokuments ausmacht und somit keine Überschrift darstellt, und alle Texte kleiner als diese korrekt als Fließtext interpretiert werden, während die größeren Texte als Überschriften verschiedener Stufen wahrgenommen werden. Obwohl hierbei keine perfekte Abbildung der tatsächlichen hierarchischen Struktur erstellt wird, bildet dieser Algorithmus diese auf sinnvolle Art nach.

6.14 Interpretation der Daten in Markdown

Um nun die Informationen final in das Markdown-Format zu übertragen, werden die bereits extrahierten Tabellen und Absätze entsprechend ihrer Position auf der Seite angeordnet. Anschließend werden diese entsprechend der so generierten Reihenfolge umgewandelt und hintereinander gesetzt.

Wird eine Tabelle umgewandelt, so werden zuerst die Inhalt einer jeden Zelle interpretiert. Hierbei werden Markdown-Tags für fetten oder kursiven Text eingesetzt und Zeilenumbrüche durch Leerzeichen ersetzt. Anschließend wird der Tabelleninhalt in eine Markdown-Tabelle übertragen, wobei alle Zellen erhalten bleiben.

Für die Konvertierung von Absätzen werden zusätzlich Überschriften annotiert sowie für

das Markdown-Format empfohlene Leerzeilen um diese eingefügt. Am Ende einer Zeile werden geöffnete Markdown-Tags automatisch geschlossen und entsprechend der aktuellen Formatierung in eine neue Zeile gewechselt.

Ein Seitenwechsel fügt schließlich einen an den Parser übergebenen Seitentrenntext ein, welcher standardmäßig ein Markdown-Kommentar mit der Seitenangabe ist. Auf diese Weise kann, wenn von einem Benutzer gewollt, der generierte Markdown-Text erneut in die entsprechenden Seiten unterteilt werden.

6.15 Test des PDF-Parsers

Um den PDF-Parser zu testen wurden verschiedene Dokumente durch diesen in das Markdown-Format übertragen. Hierbei sind verschiedene Themen aufgefallen, welche im folgenden genauer betrachtet werden.

Umfang des Parsers Während der geschriebene Parser einen großen Umfang im Code, als auch in implementierten Features aufweist, sind viele Eigenschaften von PDFs bisher noch nicht implementiert. Hierbei muss darauf hingewiesen werden, dass ein Fokus auf die am häufigsten verwendeten und wichtigsten Daten gelegt wurde. Für einen vollständigen Parser fehlen demnach noch verschiedene Elemente mit niedriger Priorität, wie zum Beispiel die Interpretation eines vertikalen Schreibmodus, digitale Signaturen, Formulare, Metadaten der Datei, Textausspracheeinstellungen und viele weiter. Zusätzlich fehlen noch einige, bisher noch nicht aufgetretene, Standardwerte für die Interpretation von Schriftarten und anderen Zeichencodierungen. Zusätzlich sind noch nicht alle Schriftarten implementiert und das Programm ist auf eine übergebene Tabelle für die Umwandlung von Zeichencodes in Buchstaben angewiesen. Insgesamt ist demnach der Umfang für erste Tests ausreichend, für einen unbetreuten Einsatz jedoch ungenügend.

Akkuratesse der Ausgabe Durch die verschiedenen implementierten Systemen für die Erkennung und Interpretation von Überschriften, werden diese meist fehlerfrei identifiziert. Auch der Text von Absätzen wird sehr präzise wiedergegeben. Probleme zeigt der entwickelte Parser jedoch mit dem Umgang mit mathematischen Formeln, besonders wenn diese Bruchstriche verwenden, und in der Identifikation von Tabellen. Letzteres liegt teilweise an der Identifikation von Tabellen über linksbündigen Text, welcher nicht zwin-

gend vorhanden ist. Wird eine Tabelle mit linksbündigem Inhalt versehen, so liefert der in Kapitel 6.11 vorgestellte Ansatz trotzdem nicht immer optimale Ergebnisse.

Erkennung von betonten Texten Obwohl das PDF-Format verschiedene Möglichkeiten für die Annotation von betonten Texten bietet - zum Beispiel durch strukturelle Informationen, Angaben der Schwere einer Schriftart oder durch die Verwendung weitere in dieser abgespeicherten Informationen - erkennt dieses System hervorgehobenen Text nicht immer. Um dieses System zu verbessern kann ein zusätzlicher, statistischer Algorithmus eingebaut werden.

Ergebnis Während der, für die Konvertierung von PDF-Dateien implementierte Algorithmus, noch nicht für die allgemeine Benutzung anwendbar ist, zeigen sich sehr gute Resultate in der Identifikation von Überschriften, dem hierarchischem Aufbau dieser und dem Übernehmen von Texten aus Textabsätzen. Weitere Arbeit ist allerdings notwendig, um den Parser zu verbessern und fehlende Details zu implementieren.

7 Fazit

Die Entwicklung eines verbesserten Algorithmus für die Extraktion von Informationen unter Zuhilfenahme eines Wissensgraphen wurde erfolgreich durchgeführt. Erste Tests zeigen zudem, dass die hiermit durchgeführte Extraktion von Informationen sowohl in der hierfür notwendigen Zeit, als auch in der Qualität der bereitgestellten Daten sehr gute Ergebnisse liefert. Er verbindet die guten Eigenschaften des bereits vorhandenen Ansatzes mit der Geschwindigkeit einer naiven, auf Embeddings aufbauenden Suche.

Weiter wurden die drei hierdurch vorhandenen Modelle genauer betrachtet, wobei die verschiedenen Eigenschaften dieser verglichen wurden. Hierbei zeigt der bereits vorhandene Retrieval-Algorithmus besondere Leistung bei der Interpretation und Verknüpfung verschiedener Themen und Texte. Somit ist dieser Ansatz geeignet für Anwendungen, bei denen ein größeres Verständnis über Daten erforderlich ist. Der neu implementierte GARAG-Ansatz kann im Gegensatz zu diesem ohne Leistungsprobleme auch lokal ausgeführt werden, antwortet schnell auf Anfragen und behält trotzdem einen gewissen Überblick über verschiedene Themenbereiche und Schlussfolgerungen aus diesen. GARAG ist besonders für eine Anwendung interessant, bei der die bessere Schlussfolgerungsfähigkeit des ersten Algorithmus nicht notwendig ist, oder für eine Anwendung im persönlichen Betrieb ohne Serveranbindung. Der naive Ansatz wiederum benötigt auch für die Indizierung der Daten sehr wenig Ressourcen und kann so vollständig ohne weitere Rechenleistung an andere Systeme integriert werden. Damit kann dieser Ansatz in abgetrennten Umgebungen verwendet werden, in denen eine leichte Unterstützung in der Generation von LLM-Antworten gewollt ist. Weiter bietet sich dieser Ansatz aufgrund der niedrigen Indizierungszeit auch für die Arbeit mit dynamische wechselnden Datenquellen an.

Die verschiedenen Optimierungen der hierfür geschriebenen Software reichen von der Verbesserung der Benutzerfreundlichkeit über Anpassung der internen Verwendung von LLMs zur Integration des Markdown-Formats für eine bessere Datenrepräsentation. Letztere kann bei manchen Formaten durch externe Python-Bibliotheken einfach integriert werden, während PDF-Dateien einen speziell hierfür geschriebenen Parser erfordern.

Da die Umwandlung von PDF-Dateien allerdings, aufgrund der in diesem Format fehlenden Ordnung der Daten, kompliziert ist, wurden nur die wichtigsten Elemente des Formats betrachtet, um die Qualität eines für diesen Zweck händisch verfassten Parsers

zu überprüfen. Während sich zeigt, dass die korrekte Identifikation des Seitenlayouts eine schwierige Aufgabe ist, zeigen sich in der erfolgreichen Extraktion und Bewertung von Überschriften und Fließtexten auch Stärken dieses Ansatz. Um die Umwandlung der Dateiformate fertigzustellen muss der Parser allerdings noch erweitert und verbessert werden.

Ausblick Während das vorliegende Projekt bereits an vielen Stellen verbessert wurde, bieten sich noch Möglichkeiten für die Erweiterung dieses:

- Der bereits implementierte Ansatz eines PDF-Parsers kann weiter verbessert und ausgebaut werden.
- Die Generierung des verwendeten Wissensgraphen kann durch den Einsatz verschiedener, von D. Edge, et al. in "'From Local to Global: A GraphRAG Approach to Query-Focused Summarization"' (vgl. [7]) vorgestellter, Ideen verbessert werden. Hierdurch soll konkret die Informationsfülle und Qualität des Wissensgraphen gesteigert werden.
- Eine weitere Möglichkeit ist die Anpassung des Algorithmus, der vorhandene Markdown-Texte in einzelne Abschnitte teilt. Indem man diesen anpasst, kann die Qualität der hieraus generierten Wissensgraphen durch eine verbesserte Kohärenz der Informationen eines Ausschnitts gesteigert werden.
- Da GARAG zwar auf Wissensgraphen aufbaut, allerdings auch die Information von Primärquellen verwendet, muss der Algorithmus angepasst werden, um selbige Qualität auf bereits von externen Prozessen erstellen Graphen zu erzeugen. Hierdurch wird sowohl der GARAG-Algorithmus, als auch die verwendete Schnittstelle an Wissensgraphen, verbessert.

Zusammenfassung Insgesamt wurde im Laufe dieser Praxisphase somit ein bereits bestehendes Projekt um verschiedene Features und Funktionen erweiter sowie der bestehende Programmaufbau verbessert. Hierbei wurde ein neuartiger Retrieval-Algorithmus implementiert, verschiedene interne Methoden verbessert und schließlich der Prototyp eines PDF-zu-Markdown-Parsers implementiert. Das Projekt kann an verschiedenen Stellen noch verbessert und erweitert werden. Der implementierte Retrieval-Algorithmus ist schon jetzt für eine allgemeine Verwendung geeignet.

8 Literaturverzeichnis

[1] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, and H. Wang, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2024.

[2] Vodafone GmbH, "Vodafone Kundenservice." `https://www.vodafone.de/hilfe/kundenservice.html`, o. J. Einsichtnahme: 18.02.2025.

[3] Vattenfall Europe Sales GmbH, "Digitaler Assistent." `https://www.vattenfall.de/`, o. J. Einsichtnahme: 18.02.2025.

[4] BMW AG, "ONLINE GENIUS (FAQ). FRAGEN UND ANTWORTEN DER BMW KUNDENBETREUUNG." `https://faq.bmw.de/`, o. J. Einsichtnahme: 18.02.2025.

[5] Telekom Deutschland GmbH, "Frag Magenta." `https://www.telekom.de`, o. J. Einsichtnahme: 18.02.2025.

[6] Telefónica Germany GmbH & Co. OHG, "Virtueller Assistent Aura." `https://www.o2online.de/service/aura/?open_chat=true`, o. J. Einsichtnahme: 18.02.2025.

[7] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, D. Metropolitansky, R. O. Ness, and J. Larson, "From local to global: A graph rag approach to query-focused summarization," *arXiv preprint arxiv:2404.16130v2*, no. 2, 2025.

[8] ArangoDB, "Container Support." `https://arangodb.com/container-support/`, o. J. Einsichtnahme: 20.02.2025.

[9] Elasticsearch B.V., "Production-ready billion scale vector database - Elasticsearch." `https://www.elastic.co/elasticsearch/vector-database`, o. J. Einsichtnahme: 20.02.2025.

[10] o. V., "Install Elasticsearch with Docker." `https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html`, o. J. Einsichtnahme: 20.02.2025.

[11] o. V., “Ds4sd/docling.” https://trendshift.io/repositories/12132, o. J. Einsichtnahme: 28.07.2025.

[12] Christoph Auer, et. al., “Docling technical report,” tech. rep., Deep Search Team, 8 2024.

[13] Z. Xu, S. Jain, and M. Kankanhalli, “Hallucination is inevitable: An innate limitation of large language models,” *arXiv preprint arxiv:2404.11817v2*, no. 2, 2025.

[14] o. V., “Dateiformate zum Speichern von Dokumenten.” https://support.microsoft.com/de-de/topic/dateiformate-zum-speichern-von-dokumenten-88de3863-c9e5-4f89-be60-906f9065e43c, o. J. Einsichtnahme: 17.07.2025.

[15] o. V., “Dateiformate, die in PowerPoint unterstützt werden.” https://support.microsoft.com/de-de/office/dateiformate-die-in-powerpoint-unterst%C3%BCtzt-werden-252c6fa0-a4bc-41be-ac82-b77c9773f9dc, o. J. Einsichtnahme: 25.07.2025.

[16] Mathieu Fenniak and pypdf contributors, “Welcome to pypdf.” https://pypdf.readthedocs.io/en/stable/, o. J. Einsichtnahme: 17.07.2025.

[17] V. A. Traag, L. Waltman, and N. J. van Eck, “From louvain to leiden: guaranteeing well-connected communities,” *Scientific Reports*, vol. 9, Mar. 2019.

[18] D. Nagar, “Llms love structure: Using markdown for better pdf analysis.” https://www.appgambit.com/blog/llms-love-structure-using-markdown-for-pdf-analysis, Mai 2025.

[19] o.V., “Run a search.” https://www.elastic.co/docs/api/doc/elasticsearch/operation/operation-search#operation-search-200, 8 2025. Einsichtnahme: 07.08.2025.

[20] o. V., “PyPDF2 3.0.1.” https://pypi.org/project/PyPDF2/, o. J. Einsichtnahme: 08.08.2025.

[21] Free Software Foundation, Inc. https://fsf.org/, “GNU GENERAL PUBLIC LICENSE,” 2007. Einsichtnahme: 11.08.2025.

[22] John MacFarlane, “Pandoc a universal document converter.” https://pandoc.org/, o. J. Einsichtnahme: 08.08.2025.

[23] Yusuke Shinyama, “pdfminer 20191125.” https://pypi.org/project/pdfminer/, o. J. Einsichtnahme: 08.08.2025.

[24] Adobe, “What does PDF mean?.” https://www.adobe.com/acrobat/about-adobe-pdf.html, o. J. Einsichtnahme: 12.08.2025.

[25] ISO Central Secretary, “Document management - Portable document format - Part 2: PDF 2.0,” Standard, International Organization for Standardization, Geneva, CH, Dez. 2020.

[26] Adobe Developer Support, “Adobe CMap and CIDFont Files Specification,” Technical Note, Adobe Systems, Incorporated, Jun. 1993.