

# **Einsatz von Retrieval-Augmented LLMs zur Unterstützung von Tutoren in Programmier-Einsteigerkursen**

Bachelor's Thesis von

Niklas Kerkhoff

An der KIT-Fakultät für Informatik  
KASTEL – Institut für Informationssicherheit und Verlässlichkeit

- |                         |                             |
|-------------------------|-----------------------------|
| 1. Prüfer/Prüferin:     | Prof. Dr.-Ing. Anne Koziolk |
| 2. Prüfer/Prüferin:     | Prof. Dr. Ralf Reussner     |
| 1. Betreuer/Betreuerin: | M.Sc. Dominik Fuchß         |
| 2. Betreuer/Betreuerin: | M.Sc. Yves Kirschner        |

12. August 2024 – 12. Dezember 2024

Karlsruher Institut für Technologie  
Fakultät für Informatik  
Postfach 6980  
76128 Karlsruhe

---

*Einsatz von Retrieval-Augmented LLMs zur Unterstützung von Tutoren in Programmier-Einsteigerkursen (Bachelor's Thesis)*

Ich versichere wahrheitsgemäß, die Arbeit selbständig verfasst, alle benutzten Quellen und Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

**Karlsruhe, 12. Dezember 2024**

.....  
(Niklas Kerkhoff)



# Zusammenfassung

Zur Unterstützung von Tutoren der Programmierenvorlesung am KIT ist ein Assistent entwickelt worden. Dieser basiert auf großen Sprachmodellen mit Retrieval-Augmented-Generation. Er stellt den Tutoren Informationen zu relevanten Inhalten der Vorlesung schnell und zuverlässig zur Verfügung. Ausgehend von einer Anforderungsanalyse ist der Assistent speziell für diese Anwendung entworfen und optimiert worden. Insbesondere die eigens entwickelten Retriever führen zu einem verbesserten Abruf der Dokumente. Der Nutzen und die Zuverlässigkeit des Assistenten sind in einer Evaluation geprüft worden. Diese zeigt, dass er meistens korrekte und zuverlässige Informationen bereitstellen kann, deren Nutzen für die Tutoren jedoch noch gesteigert werden könnte.



# Inhaltsverzeichnis

<b>Zusammenfassung</b>	<b>i</b>
<b>1. Einleitung</b>	<b>1</b>
<b>2. Grundlagen</b>	<b>3</b>
2.1. Natürliche Sprachverarbeitung . . . . .	3
2.2. Einbettung . . . . .	3
2.3. Transformer-basierte große Sprachmodelle . . . . .	4
2.4. Vectorstores . . . . .	5
2.5. Retrieval-Augmented-Generation . . . . .	6
<b>3. Verwandte Arbeiten</b>	<b>9</b>
3.1. Assistenten in der Lehre . . . . .	9
3.1.1. Beschreibung einiger Assistenten . . . . .	9
3.1.2. Schlussfolgerung und Abgrenzung . . . . .	10
3.2. Methoden für Retrieval-Augmented-Generation . . . . .	10
3.3. Prompt-Engineering . . . . .	11
<b>4. Anforderungsanalyse</b>	<b>13</b>
4.1. Analyse der Tätigkeiten der Tutoren . . . . .	13
4.2. Studie mit ehemaligen Tutoren . . . . .	14
4.2.1. Bedarf nach vereinfachtem Informationsgewinn . . . . .	14
4.2.2. Vertrauen in den Assistenten . . . . .	15
4.2.3. Relevante Dokumente . . . . .	16
4.2.4. Benutzerschnittstelle . . . . .	17
4.2.5. Funktionen des Assistenten . . . . .	17
4.3. Liste der Anforderungen . . . . .	18
<b>5. Entwicklung des Assistenten</b>	<b>21</b>
5.1. Entwurf . . . . .	21
5.1.1. Architektur . . . . .	21
5.1.2. Abläufe . . . . .	22
5.2. Web-Frontend . . . . .	23
5.2.1. Liste der Funktionen . . . . .	24
5.2.2. Beschreibung der Funktionen und Benutzerschnittstelle . . . . .	25
5.3. App-Service . . . . .	27

5.4.	RAG-Service . . . . .	28
5.4.1.	Vorgehen . . . . .	28
5.4.2.	LangChain . . . . .	29
5.4.3.	Einbettung von Inhalten . . . . .	30
5.4.4.	Abruf der Dokumente für die Chat-Funktion . . . . .	31
5.4.5.	Prompt-Engineering . . . . .	40
5.4.6.	Vergleich von Einbettungsmodellen . . . . .	41
5.4.7.	Vergleich von Sprachmodellen . . . . .	42
5.4.8.	Kalender . . . . .	44
<b>6.</b>	<b>Evaluation des Assistenten</b>	<b>47</b>
6.1.	Auswertung der Chat-Verläufe . . . . .	47
6.2.	Auswertung der Umfrage . . . . .	49
6.2.1.	Schlussfolgerung . . . . .	52
<b>7.</b>	<b>Fazit</b>	<b>53</b>
	<b>Literatur</b>	<b>55</b>
<b>A.</b>	<b>Anhang</b>	<b>61</b>
A.1.	Studie zur Anforderungsanalyse . . . . .	61
A.2.	Code in Anfrage . . . . .	63
A.3.	Beispiel für einen Chat-Verlauf . . . . .	64

# Abbildungsverzeichnis

2.1.	Grundprinzip von RAG . . . . .	7
4.1.	Bedarf nach einfachem Informationsgewinn . . . . .	15
4.2.	Vertrauen in den Assistenten . . . . .	16
5.1.	Komponenten-Diagramm des Assistenten . . . . .	21
5.2.	Bedarf nach einfachem Informationsgewinn . . . . .	25
5.3.	Bedarf nach einfachem Informationsgewinn . . . . .	26
5.4.	Bedarf nach einfachem Informationsgewinn . . . . .	27
5.5.	Aktivitätsdiagramm vom MultiStepsRetriever . . . . .	39
6.1.	Wahrgenommener Nutzern des Assistenten durch die Tutoren. . . . .	51
6.2.	Einstellung der Tutoren gegenüber dem Assistenten . . . . .	51
A.1.	Beispiel für Chat-Verlauf. Die Nachrichten haben einen guten Bezug zur Vorlesung. . . . .	64



# Tabellenverzeichnis

4.1. Liste der Anforderungen . . . . .	19
5.1. Ergebnisse des Prototypen. . . . .	43
5.2. Auswertung . . . . .	45
6.1. Fragebogen der Umfrage . . . . .	50
6.2. Cronbachsches Alpha für jede Gruppe . . . . .	50
6.3. Kennzahlen zur Bewertung des Nutzen (PU) und der Einstellung (AT) . . .	52
A.1. Fragen der Studie zur Anforderungsanalyse . . . . .	62



# 1. Einleitung

Tutoren der Programmierenvorlesung am KIT sind häufig mit einer Vielzahl an Informationen konfrontiert, die es zu überblicken gilt. Dabei handelt es sich um organisatorische Daten wie Terminen, sowie Richtlinien zu Programmierstilen, Vorlesungsinhalten, Informationen zu Übungen und weitere. Die Materialien, die diese Informationen beinhalten, sind über mehrere Websites und Dateien verteilt. Letztere sind über mehrere Online-Portale verteilt. Zudem werden einige dieser Materialien regelmäßig aktualisiert und erweitert.

**Ziel der Arbeit** Während der Arbeit als Tutor stets den Überblick über diese Informationen zu behalten kann eine Herausforderung sein. Diese wird in dieser Arbeit adressiert. Dazu wird ein Assistent entwickelt, welcher den Tutoren ermöglicht, schnell notwendige Informationen abzurufen. In einem Chat mit dem Assistenten können Tutoren nach Informationen fragen. Der Assistent hat Zugriff auf verschiedene Materialien der Vorlesung und generiert basierend auf diesen Inhalten Antworten. Zudem gibt es einen tabellarischen Kalender im Assistenten, der automatisch wichtige Termine und Fristen anzeigt, welche in den Materialien gefunden werden. Im Anschluss zur Implementierung des Assistenten wird dieser mithilfe der aktuellen Tutoren evaluiert. Dabei wird untersucht, wie zuverlässig und hilfreich seine Antworten sind und wie die Tutoren dem Assistenten gegenüber eingestellt sind.

**Technik** Damit die Tutoren in natürlicher, menschlicher Sprache mit dem Assistenten kommunizieren können, erfolgt die Umsetzung des Assistenten mithilfe großer Sprachmodelle [2]. Diese zeigen herausragenden Fähigkeiten im Bereich der natürlichen Sprachverarbeitung [38] und stellen eine bedeutende Technologie im Bereich der künstlichen Intelligenz dar [19]. Nur wenige Jahre nach ihrer Einführung haben sie sich in zahlreichen Applikationen etabliert [18]. Assistenten sind dabei ein verbreiteter Anwendungsbereich. Ein Beispiel dafür ist ChatGPT. Damit das Sprachmodell nicht nur allgemeine Fragen sondern auch Fragen zu Materialien der Vorlesung beantworten kann, müssen ihm diese zur Verfügung gestellt werden. Dazu wird Retrieval-Augmented-Generation [12] eingesetzt. Diese Technik ermöglicht es, Sprachmodellen, die bereits ein breites Allgemeinwissen besitzen, weitere Inhalte zum Beantworten von Benutzeranfragen zur Verfügung zu stellen.

**Vorgehen** In dieser Arbeit wird der Assistent entwickelt und optimiert. Dazu wird zuerst eine Anforderungsanalyse durchgeführt. Basierend darauf wird der Assistent entworfen und implementiert. Als Benutzerschnittstelle wird ein Web-Frontend erstellt. Das Backend ist auf mehrere Services verteilt. Der RAG-Service ist dafür verantwortlich, die notwendigen Informationen abzurufen und zur Verfügung zu stellen. Um möglichst gute Antworten zu

erzeugen, liegt der Fokus der Arbeit auf der Optimierung dieses Services. Dazu werden verschiedene Techniken zum Abrufen der angebundenen Informationen untersucht und eigene Komponenten entwickelt, die diesen Prozess verbessern. Zudem wird untersucht, wie Antworten des Sprachmodells basierend auf diesen Informationen verbessert werden können. Im Anschluss an die Implementierung wird der Assistent evaluiert. Dazu wird eine Nutzerstudie mit aktuellen Tutoren der Programmierenvorlesung durchgeführt.

## 2. Grundlagen

In diesem Kapitel werden die Techniken erläutert, die im Assistenten für die natürliche Sprachverarbeitung und für das Abrufen vorlesungsspezifischer Informationen verwendet werden.

### 2.1. Natürliche Sprachverarbeitung

Ziel der Arbeit ist die Umsetzung eines Assistenten, der Fließtexte versteht und mit dem auf natürliche Weise kommuniziert werden kann. Dazu werden Konzepte der natürlichen Sprachverarbeitung [20], engl. Natural Language Processing (NLP), verwendet.

NLP bezeichnet im Allgemeinen den Prozess, natürliche, menschliche Sprache zu verstehen und zu generieren. Während es sich dabei für Menschen um eine leichte Aufgabe handelt, stellt sie Computer vor große Herausforderungen [38]. Dies hat mehrere Gründe [5]. Zum einen ist natürliche Sprache sehr komplex, da sie durch viele Vokabeln und grammatikalische Regeln definiert ist. Dies erlaubt eine unendliche Anzahl an korrekten Sätzen. Des Weiteren neigt natürliche Sprache zu Ambiguitäten. Diese können häufig nur durch den Kontext und Hintergrundinformationen aufgelöst werden. Darüber hinaus ist es schwierig, Texte variabler Länge so zu verarbeiten, dass jedes neue Wort in den richtigen Zusammenhang mit den anderen Wörtern der Eingabe gebracht wird. Dies ist notwendig, um die Bedeutung des Textes erfassen.

NLP ist bereits seit Jahrzehnten ein Forschungsgebiet [29]. Unter die klassischen Anwendungsgebiete fallen das maschinelle Kategorisieren oder Übersetzung von Texten. Die Menge der Wörter, mit der ein NLP-System umgehen kann, wird der Korpus genannt. Traditionelle Systeme sind häufig auf diesen Korpus beschränkt und haben Schwierigkeiten damit, unbekannte Wörter zu erkennen [17]. Zudem bringen sie die Wörter eines Textes oft nicht in den richtigen Zusammenhang [17]. Dadurch können sie Ambiguitäten nicht auflösen und die Semantik des Textes nicht erfassen. Aktuelle Systeme verwenden große Sprachmodellen, die auf Transformern basieren [38, 36]. Diese ermöglichen es, beliebige Wörter zu verarbeiten und in den richtigen Zusammenhang zu bringen.

### 2.2. Einbettung

Texte, die im Assistenten maschinell verarbeitet werden, werden eingebettet [27]. Dies ist die Grundlage zur Erfassung der Bedeutung dieser Texte.

Vor der eigentlichen Einbettung wird der Text in einzelne Tokens unterteilt. Dies wird als Tokenisierung bezeichnet [28]. Tokens können dabei Wörter, Teilwörter oder auch einzelne Buchstaben sein. Die konkrete Art der Zerlegung hängt von der gewählten Strategie ab. Der Satz „Ich melde mich rechtzeitig zur Präsenzübung an.“ wird von OpenAIs Einbettungsmodell text-embedding-3-large [25] zu „Ich| mel|de| mich| recht|zeitig| zur| Pr|ä|sen|z|ü|bung| an.“ tokenisiert, wobei | die einzelnen Tokens trennt. Es ist erkennbar, dass zum Teil ganze Wörter als Tokens verwendet werden, zum Teil jedoch nur Teilwörter oder einzelne Buchstaben. In der resultierenden Zerteilung muss jeder Token dem System bekannt sein, um im nächsten Schritt eine Einbettung zu ermöglichen. Wenn bspw. das Wort "Präsenzübung" dem System unbekannt ist, muss es entsprechend zerteilt werden.

Nach der Tokenisierung werden die einzelnen Tokens eingebettet [27]. Dabei werden sie in numerische Vektoren übersetzt, welche die Semantik jeweils kodieren sollen. Aktuelle Modelle wie die von OpenAI generieren technologisch führende Einbettungen [27]. Sie betten Tokens abhängig vom Kontext unterschiedlich ein [13]. Die resultierenden Vektoren haben oft mehrere Hundert oder mehrere Tausend Dimensionen [25]. Ihre Werte werden in Trainingsprozessen gelernt. Das Grundprinzip ist, dass ähnliche Tokens auch ähnlichen Vektoren zugeordnet werden sollen. Ein häufig zitiertes Beispiel ist, dass die Gleichung König – Mann + Frau  $\approx$  Königin für die Einbettungen der in vorkommenden Wörter gelten sollte [10].

### 2.3. Transformer-basierte große Sprachmodelle

Aktuelle große Sprachmodell [2], engl. Large Language Models (LLMs), basieren auf Transformern [38, 16]. Sie stellen eine bedeutende Entwicklung in der natürlichen Sprachverarbeitung dar [16]. Sie zeichnen sich darin aus, eine passende textuelle Ausgabe auf beliebige textuelle Eingaben zu generieren. Transformer [34] verwenden Einbettungen und tiefe neuronale Netze [37, 6] kombiniert mit weiteren Mechanismen wie dem sogenannten Attention-Mechanismus [34]. Diese Mechanismen zielen insbesondere darauf ab, die Tokens der Ein- und Ausgabe in den richtigen Kontext zu bringen. Aktuelle große Sprachmodelle haben oft viele Milliarden Parameter, die in einem aufwendigen Prozess trainiert werden [22].

Bei der Generierung von Antworten ist die Einbettung der Eingabe ein wichtiger Schritt. Dies ermöglicht es dem Sprachmodell, die Semantik des Textes zu erfassen. Die Tokens der Ausgabe werden dann basierend auf Wahrscheinlichkeiten ausgegeben. Im Trainingsprozess lernt das Modell, welcher Token in welchem Kontext wahrscheinlich als nächstes ausgegeben werden soll. Der Kontext ergibt sich aus der Eingabe und den bisher ausgegebenen Tokens. Das Modell gibt so lange den nächsten Token aus, bis es einen Token generiert, der das Ende der Ausgabe signalisiert. Viele Modelle geben im selben Kontext nicht immer den wahrscheinlichsten Token aus, sondern bringen Variation mit ein [26]. Somit können sich Antworten trotz des selben Kontexts unterscheiden. Dieses Verhalten ist in manchen Anwendungsfällen erwünscht, bspw. für mehr Kreativität. In vielen Anwendungsfällen führt der mangelnde Determinismus jedoch zu Problemen. Das Verhalten kann über den

sogenannten Temperaturwert des Modells gesteuert werden. Dieser gibt an, wie deterministisch die Antworten sind. Er kann bei der Verwendung des Modells spezifiziert werden. Absoluter Determinismus lässt sich darüber jedoch nicht unbedingt konfigurieren.

Beim Training eines Sprachmodells wird zwischen Vortraining und Feintuning unterschieden [33].

Im Vortraining gilt es, dem Modell grundlegendes Wissen über Sprache und allgemeine Sachverhalte zu vermitteln. Das Modell soll danach in der Lage sein, Eingaben zu allgemeinen Themen zu verstehen und adäquate Antworten zu liefern. Dazu werden große Datenmengen an Texten herangezogen (bspw. aus dem Internet). Das Modell bekommt nun iterativ Teile der Texte übergeben und soll basierend darauf den nächsten Token wählen. Da der korrekte Text vorhanden ist, kann der Token automatisiert überprüft werden. Eine Loss-Funktion kann die Abweichung des gewählten Tokens und des korrekten Tokens basierend auf deren Einbettung berechnen. Die Parameter des Modells können auf diese Weise in einem iterativen Prozess automatisch angepasst werden. Somit handelt es sich dabei um ein selbstüberwachtes Lernen.

Einige Modelle werden nach dem Vortraining einer Feintuning unterzogen. Dies ist ein auf dem Vortraining aufbauender Schritt, um ein Modell auf bestimmte Sachverhalte zu spezialisieren. Das Training verläuft dabei ähnlich wie beim Vortraining, jedoch werden nur Texte, die für den Sachverhalt relevant sind, für das Training verwendet. Durch ein umfangreiches Vortraining genügt i. d. R. ein relativ kleiner Datensatz zum Feintuning.

Bei der Anwendung von Sprachmodellen werden typischerweise neben der Anfrage des Benutzers noch weitere Texte der Anfrage mitgegeben. Dies können der Chat-Verlauf des Benutzers mit dem Modell sein oder Instruktionen, wie das Sprachmodell sich verhalten soll. Bspw. kann einem Sprachmodell in einem Assistenten zur Verbesserung von Texten auf diese Weise mitgeteilt werden, in welchem Stil der Text verfasst werden soll. Der Text selber wäre dabei die Anfrage des Benutzers. Des Weiteren können dadurch weitere relevante Informationen dem Sprachmodell übergeben werden, auf deren Basis es seine Antwort generieren soll. Abschnitt 2.5 stellt eine verbreitete Technik vor, die in dieser Arbeit dafür verwendet wird.

## 2.4. Vectorstores

Vectorstores sind Datenbanken, die Dokumente einbetten, speichern und durchsuchen können [1, 30]. Dazu muss eine Komponente spezifiziert werden, die diese Einbettung durchführt, bspw. OpenAIs text-embedding-3-large. Die generierten Einbettungen werden gespeichert und indiziert. Letzteres ermöglicht einen effizienten Zugriff.

Eine zentrale Funktion von Vectorstores ist, dass sie zu einem gegebenen Text ähnliche Dokumente finden und zurückgeben können. Dabei wird eine textuelle Anfrage an den Vectorstore gestellt. Dieser erstellt daraus eine Einbettung und führt mit ihr und den gespeicherten Dokumenten eine Ähnlichkeitssuche durch. Da moderne Einbettungen nicht nur

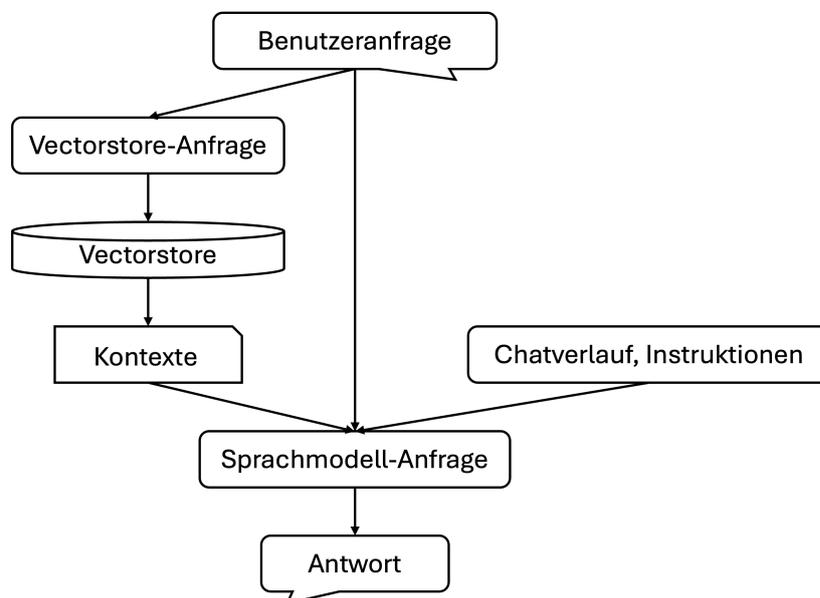
die Syntax sondern auch die Semantik kodieren, werden dabei Dokumente gefunden, die bedeutungsähnlich zur ursprünglichen Anfrage sind.

### **2.5. Retrieval-Augmented-Generation**

Sprachmodelle sind gut darin, die Anfragen eines Benutzers zu verstehen und passende Antworten zu generieren. Ihr breites Allgemeinwissen sowie ihre herausragenden Fähigkeiten in manchen Fachgebieten wie der Programmierung machen sie damit bereits zu hilfreichen Assistenten. In vielen Anwendungsfeldern möchte man dieses Wissen jedoch um weiteres fachspezifisches Wissen erweitern. Eine Möglichkeit wäre ein Feintuning auf diesem Gebiet. Dies bringt jedoch entscheidende Nachteile:

1. Es müssen genug Ressourcen zur Verfügung stehen (Rechenkapazität, Zeit, Geld).
2. Wenn Daten hinzukommen oder sich ändern muss der Prozess erneut durchgeführt werden.

Eine sehr verbreitete Alternative ist Retrieval-Augmented-Generation (RAG) [12]. Dies ist eine Technik, die das interne Wissen von Sprachmodellen durch externes Wissen erweitert. Abbildung 2.1 zeigt den Prozess vom Eingang einer Benutzeranfrage bis zur Antwortgenerierung mit einfachem RAG. Zunächst werden vor jeder Anfrage an das Sprachmodell relevante Dokumente als sogenannte Kontexte aus einem Vectorstore abgerufen. Die Anfrage, die an den Vectorstore gesendet wird kann sich dabei von der Benutzeranfrage unterscheiden. Bspw. kann ein Sprachmodell ohne RAG aufgefordert werden, die Benutzeranfrage für die Suche in einem Vectorstore zu optimieren. Die Kontexte werden zusammen mit der Benutzeranfrage und möglicherweise weiteren Texten wie Chat-Verlauf und Instruktionen an das Sprachmodell gesendet und werden somit bei der Generierung der Antwort miteinbezogen. Während das Wissen, das beim Vortraining und Feintuning dem Modell vermittelt wird, sehr statisch ist, können die Dokumente in einem Vectorstore schnell hinzugefügt oder entfernt werden. Dadurch kann das Modell für viele verschiedene Bereiche eingesetzt werden. In dieser Arbeit werden bereits vollständig trainierte Modelle betrachtet und mithilfe von RAG für den spezifischen Sachverhalt verwendbar gemacht.



**Abbildung 2.1.:** Grundprinzip von RAG



## 3. Verwandte Arbeiten

In diesem Kapitel werden ähnliche Assistenten beschrieben und wie sie sich von meiner Lösung unterscheiden. Des Weiteren werden Techniken zur Optimierung von Anfragen an Vectorstores und Sprachmodelle vorgestellt.

### 3.1. Assistenten in der Lehre

Einige ähnliche Arbeiten befassen sich mit der Entwicklung eines Assistenten, welcher Lernenden Rückmeldung und Unterstützung bei Aufgaben in der Programmierung bieten soll.

#### 3.1.1. Beschreibung einiger Assistenten

Dai u. a. [8] untersuchen, inwiefern ChatGPT in der Lage ist, Lernenden Rückmeldung zu ihren Lösungen von Aufgaben zu geben. Dabei wird auf drei Aspekte eingegangen: (1) Die Verständlichkeit der von ChatGPT generierten Rückmeldung, (2) die Übereinstimmung mit der Rückmeldung von Lehrenden und (3) die Effektivität der Rückmeldung hinsichtlich des Lernens. Die Ergebnisse zeigen, dass ChatGPT zwar in der Lage ist verständliche und effektive Rückmeldung zu geben, diese jedoch häufig von der Rückmeldung abweicht, die die eigentlichen Lehrkräfte geben würden.

Hellas u. a. [14] thematisieren die Leistungsfähigkeit von Sprachmodellen, insbesondere OpenAI's GPT-3.5, Rückmeldung zu Code von Lernenden zu geben. Ein zentrales Ergebnis ist, dass nur selten Antworten gegeben werden, die ohne Bearbeitung durch Lehrkräfte an die Lernenden weitergegeben werden konnten, was auf die Notwendigkeit von Anpassungen hinweist. Zudem gibt es häufig falsche Positivmeldungen bei den erkannten Problemen, was die Zuverlässigkeit der Modelle einschränkt.

Frankford u. a. [11] präsentieren einen AI-Tutor, welcher Studenten Fragen zu ihren Abgaben im Bereich Programmierung beantworten kann. Ziel ist es zum einen, die eigentlichen Tutoren zu entlasten, sodass sie sich auf andere Aufgaben konzentrieren können, welche nicht automatisiert erledigt werden können. Zum anderen ermöglicht der AI-Tutor eine deutlich schnellere Rückmeldung an die Studenten, wovon diese ebenfalls profitieren. Die Entwickler des AI-Tutors evaluieren in einer Studie, wie hilfreich der Assistent für die Studenten ist. Diese zeigt, dass er manche Studenten gut unterstützen kann, andere nicht.

Im Mittel wird er als neutral bewertet. Einige Studenten kritisieren besonders zu generelle Antworten des AI-Tutors, welche keine Hilfe darstellen.

#### 3.1.2. Schlussfolgerung und Abgrenzung

In allen Fällen konnten Assistenten entwickelt werden, die Lernenden schnelle und korrekte Rückmeldung liefern können. Jedoch weicht diese Rückmeldung häufig von der tatsächlich gewünschten Rückmeldung ab. Dies ist möglicherweise darauf zurückzuführen, dass der Assistent weder zu konkrete noch zu allgemeine Lösungen bereitstellen darf und dass jede Lehrveranstaltung besondere Anforderungen und Richtlinien hat, die nur schwierig durch ein allgemein trainiertes Modell erfüllt werden können. Um diese Herausforderungen anzugehen, soll in der Arbeit ein ähnlicher Assistent umgesetzt werden, welcher jedoch nicht von den Lernenden selber sondern von Tutoren bedient werden soll. Sie können ihre Arbeit durch schnelles Erfragen von Informationen beschleunigen, jedoch können sie die Antworten des Assistenten überprüfen, bevor sie an Studenten weitergegeben wird.

## 3.2. Methoden für Retrieval-Augmented-Generation

In diesem Abschnitt werden verschiedene RAG-Methoden vorgestellt. In Unterabschnitt 5.4.4 werden Varianten von ihnen jeweils erörtert und gegeneinander abgewogen, um möglichst gute Ergebnisse zu erzielen.

**Stuff-Methode** Bei der Stuff-Methode [30] handelt es sich um die einfachste RAG-Methode. Dabei wird mit der Benutzeranfrage in einem Vectorstore nach den am besten passenden Dokumenten gesucht. Dabei wird die Anzahl der abzurufenden Dokumente typischerweise beschränkt. Die resultierenden Kontexte werden zusammen mit der Benutzeranfrage und gegebenenfalls weiteren Texten wie Chatverlauf und Instruktionen an das Sprachmodell gesendet. Diese Methode zeichnet sich durch eine hohe Geschwindigkeit beim Abfragen der Dokumente aus.

**Unterteilen der Dokumente** Eine einfache Methode, die sich mit den anderen Methoden kombinieren lässt, ist das Unterteilen der Dokumente vor der Einbettung in sogenannte Chunks [23]. Anschließend werden statt der gesamten Dokumente die einzelnen Chunks eingebettet. Die jeweiligen Einbettungen müssen somit weniger Inhalt und Kontext repräsentieren, was die Ähnlichkeit zu Benutzeranfragen erhöhen kann. Eine häufig verwendete Strategie zur Unterteilung von Dokumenten ist das Zerlegen in gleich lange Chunks gemessen an den Zeilenlängen, bspw. 1000 Zeilen. Um möglichst keinen Kontext zu verlieren, lässt man dabei typischerweise Überschneidungen der Chunks über bspw. über 200 Zeilen zu.

**Query-Step-Down-Methode** Bei der Query-Step-Down-Methode [30] werden aus der Benutzeranfrage unterschiedliche Vectorstore-Anfragen generiert. Dazu kann ein Sprachmodell verwendet werden. Anschließend wird für jede der  $n$  generierten Anfragen der Prozess der Stuff Methode durchgeführt. Die dabei resultierenden  $n$  Antworten werden anschließend kombiniert und an den Benutzer gegeben. Diese Methode eignet sich besonders, um Ambiguitäten in der Benutzeranfrage aufzulösen. Durch die zusätzlichen Anfrage an ein Modell zur Generierung von Vectorstore-Anfragen und die mehrfachen Vectorstore-Anfragen kann sich die Generierung der Antwort jedoch verzögern.

**Parent-Document-Retriever** Für den Abrufprozess sind häufig kleine Chunks der Dokumente vorteilhaft. Für die tatsächliche Antwortgenerierung eignen sich jedoch oft größere Chunks oder sogar das gesamte Dokument besser. Der Parent-Document-Retriever [15] adressiert diese Herausforderung. Für eine präzisere Suche bettet er die Dokumente als kleine Chunks ein. Nach dem Abrufen der kleinen Chunks führt er diese jedoch wieder zu größeren Dokumenten zusammen und gibt diese als Resultat aus.

**Active RAG** Active RAG gibt es in verschiedene Varianten [4]. Die Grundidee ist es, das Sprachmodell entscheiden zu lassen, ob es die Anfrage des Benutzers beantworten kann oder ob es zusätzliche Informationen benötigt. Abhängig davon kann es eine Antwort ausgeben oder einen Kontext-Abruf veranlassen. Damit kann das Sprachmodell entscheiden, welche Informationen es von dem Vectorstore abrufen möchte, um die Anfrage zu beantworten.

### 3.3. Prompt-Engineering

Als Prompt wird die Anfrage an das Sprachmodell bezeichnet. Ein wichtiger Teil der Anfragen sind die Instruktionen, die das Modell anweisen, nach welchen Regeln es auf die eigentliche Anfrage des Benutzers antworten soll [35]. Das Optimieren der Anfragen, insbesondere der Instruktionen wird Prompt-Engineering genannt. Im Folgenden werden fünf Techniken vorgestellt.

**Grundlegende Informationen** In vielen Fällen ist es wichtig, dem Sprachmodell grundlegende Informationen über den Sachverhalt mitzuteilen. Darüber kann das Modell angewiesen werden, was es mit der Anfrage des Benutzers machen soll (bspw. den Text verbessern oder übersetzen) oder wie es mit dem Benutzer interagieren soll (bspw. per Du ansprechen). Zudem können Informationen wie das aktuelle Datum beigefügt werden.

**Few-Shot-Prompting** Zero-, One- und Few-Shot bezeichnet die Häufigkeit, die das Sprachmodell einen bestimmten Sachverhalt bereits gesehen hat [2]. Wenn für die Antwort bspw. ein Ausgabeformat gefordert ist, welches dem Modell unbekannt ist und das nur mit Worten beschrieben wird, handelt es sich um ein Zero-Shot-Setting. Werden den Instruktionen jedoch Beispiele für mögliche Ausgaben hinzugefügt, entsteht daraus ein Few-Shot-Setting. Dies kann die Ausgabe verbessern [2].

**Chain-of-Thought** Beim Chain-of-Thought [3] wird das Sprachmodell angewiesen, bei der Generierung seiner Antwort Schritt für Schritt zu denken. Es soll nicht nur seine finale Antwort ausgeben, sondern vorher erklären, wie es auf diese kommt. Dadurch erstellt es bei der Generierung der Antwort Zusammenfassungen der verschiedenen Teilprobleme. Diese stellen für das Modell einen Kontext dar, den es für die Ausgabe der finalen Antwort mit einbeziehen kann.

**Emotion-Prompting** Beim Emotion-Prompting wird dem Sprachmodell mitgeteilt, dass etwas besonders Grausames passiert, wenn es falsche Antworten generiert und dass es dies auf keinen Fall zulassen darf [31]. Die Idee ist, dass das Modell gelernt hat, dass grausame Ereignisse zu vermeiden sind und somit besonders vorsichtig und bedacht antwortet.

**Sampling** Sampling bezeichnet die Technik, dem Sprachmodell mehrmals die Anfrage des Benutzers zu stellen [31]. Dies kann mit anderen Techniken wie Chain-of-Thought kombiniert werden. Aus den generierten Antworten kann dann die beste (bspw. mithilfe eines Sprachmodells) ausgesucht werden.

## 4. Anforderungsanalyse

Die Grundidee der Bachelorarbeit ist es, einen Assistenten zu entwickeln, der möglichst hilfreich für die Tutoren der Programmierenvorlesung ist. Sie sollen ihm Fragen stellen können, die er zuverlässig beantwortet. Dies können bspw. Fragen zu Inhalten der Vorlesung sein. Dazu sollen Sprachmodelle und RAG verwendet werden. Somit ergeben sich mehrere Fragen, die in der Anforderungsanalyse geklärt werden sollen:

1. Welche Art von Anfragen werden die Tutoren stellen?
2. Welche Dokumente müssen eingebettet werden?
3. Welche Funktionen soll der Assistent bereitstellen?
4. Wie soll die Benutzerschnittstelle gestaltet sein?

Um diese Fragen zu beantworten wurde zunächst die Tätigkeit der Tutoren analysiert, anschließend eine explorative Studie durchgeführt und zuletzt die resultierenden Anforderungen zusammengefasst.

### 4.1. Analyse der Tätigkeiten der Tutoren

Die Tutoren der Programmierenvorlesung haben vor allem die Aufgabe Tutorien zu halten sowie Abgaben der Studenten zu korrigieren.

**Halten von Tutorien** Beim Halten von Tutorien werden Inhalte der Vorlesungen wiederholt und vertieft. Zu diesen Zwecken werden Vorträge gehalten, Fragen der Studenten beantwortet und Übungsaufgaben gegeben und besprochen. Ein Tutorium muss daher sorgfältig vorbereitet werden. Bei dieser Vorbereitung kann ein Assistent helfen, um Fragen zu Inhalten zu beantworten oder Übungsaufgaben zu generieren, die dem Fortschritt der Vorlesung angemessen sind. Während der Tutorien kann der Assistent ebenfalls beim Beantworten der Fragen der Studenten den Tutor unterstützen. Dadurch könnte die Antwortzeit verkürzt werden.

**Korrigieren der Abgaben** Studenten müssen regelmäßig Übungsblätter mit Programmieraufgaben lösen. Diese Abgaben werden von den Tutoren korrigiert. Bei der Korrektur sind genaue Richtlinien einzuhalten, die aus verschiedenen Quellen hervorgehen. Ein Assistent kann dabei helfen, Fragen zu diesen Richtlinien schnell zu beantworten, um die Korrektur zu beschleunigen.

## 4.2. Studie mit ehemaligen Tutoren

Es wird eine explorative Studie mit ehemaligen Tutoren durchgeführt. Ihre Erfahrungen und Perspektiven sowie die Einblicke in ihre Arbeitsweise sollen helfen, die oben beschriebenen Fragen bezüglich der Anforderungen zu beantworten. Zudem soll die Studie Aufschluss darüber geben, wie hoch der Bedarf nach einem solchen Assistenten ist und für wie vertrauenswürdig dieser eingeschätzt wird. Damit können die Funktionen des Assistenten entsprechend angepasst werden.

Die Studie ist von 12 Teilnehmern vollständig ausgefüllt worden. Diese Antworten werden bei der Auswertung berücksichtigt. Es ist möglich, dass sich die Tutoren je nach Erfahrung unterschiedliche Anforderungen und Einstellungen bezüglich des Assistenten haben. Daher ist die Häufigkeit, mit der die Teilnehmer der Studie bereits Tutor waren, erfasst worden. Drei der Teilnehmer sind einmal, vier von ihnen zweimal und fünf mehr als zweimal Tutor gewesen. Um die Zahlen zu anonymisieren, ist die genaue Häufigkeiten für Teilnehmer, die mehr als zweimal Tutor gewesen sind, nicht erfasst worden.

Die Fragen der Studie sind in Tabelle A.1 aufgelistet. Im Folgenden wird basierend auf den Antworten der Teilnehmer der Bedarf nach einem Assistenten für Tutoren sowie das Vertrauen, dass die Tutoren in ihn hätten, untersucht. Daraus werden Anforderungen an den Assistenten abgeleitet. Einige Fragen werden dabei nicht näher beleuchtet, da aus ihren Ergebnissen keine weiteren Anforderungen hervorgehen.

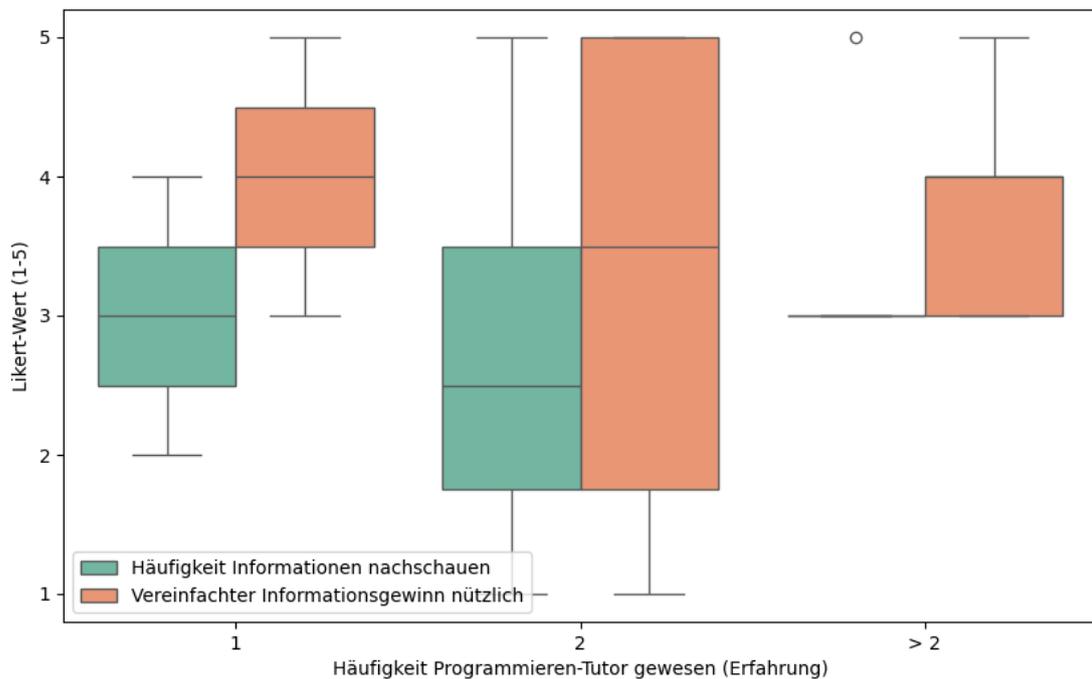
### 4.2.1. Bedarf nach vereinfachtem Informationsgewinn

Die Grundidee des Assistenten ist die Abfrage von Informationen. Dazu wird untersucht, wie häufig die Tutoren Informationen abfragen und ob sie einen vereinfachten Informationsgewinn als nützlich empfinden. Diese Werte werden als Hinweise für den Bedarf nach einem Assistenten interpretiert. Sie sind in Abbildung 4.1 dargestellt. Es ist denkbar, dass sie sich abhängig von der Erfahrung des jeweiligen Tutors unterscheiden. Daher werden sie abhängig von der Häufigkeit, die ein Teilnehmer der Studie Tutor gewesen ist, dargestellt.

Die Antworten der Teilnehmer der Studie variieren stark. Dabei ist kein eindeutiger Zusammenhang zwischen der Erfahrung und den gemessenen Werten herzustellen. Es ist jedoch erkennbar, dass einige Teilnehmer regelmäßig Informationen nachschauen und noch mehr einen vereinfachten Informationsgewinn als nützlich betrachten.

Aus der Analyse kann geschlossen werden, dass der Bedarf nach einem Assistenten für vereinfachten Informationsgewinn nicht enorm ist. Um dennoch zu überzeugen, werden daraus folgende Anforderung an den Assistenten abgeleitet:

1. Die Antworten des Assistenten sollen möglichst hilfreich sein.
2. Die Bedienung des Assistenten soll möglichst einfach sein.
3. Die Abfrage von Informationen soll möglichst schnell durchgeführt werden.



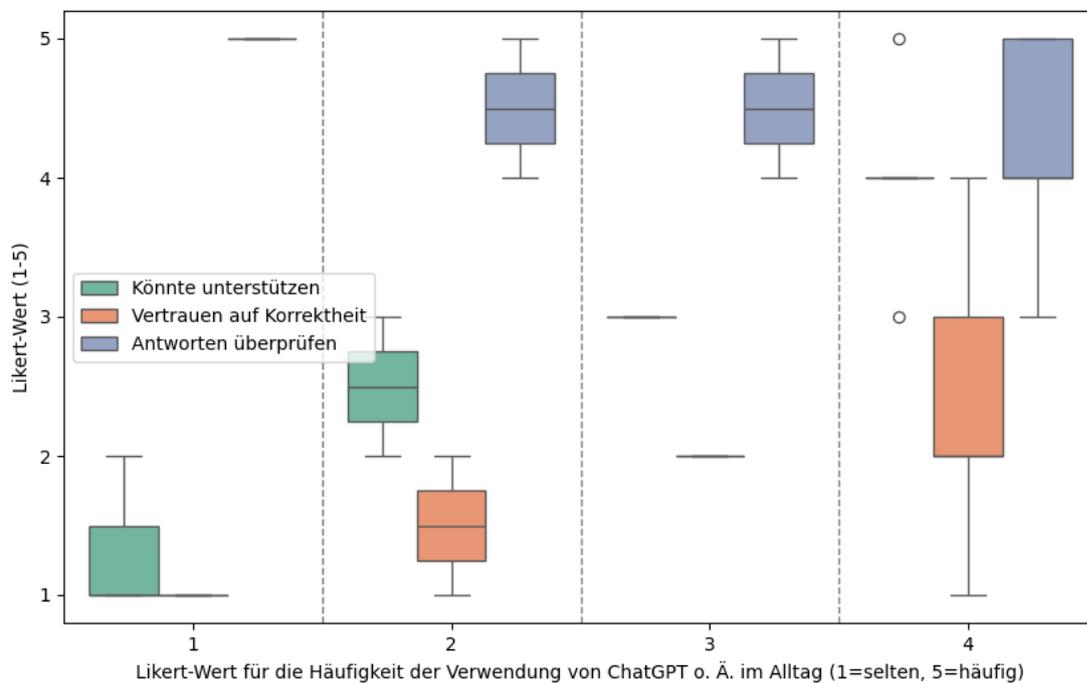
**Abbildung 4.1.:** Bedarf nach einfachem Informationsgewinn

#### 4.2.2. Vertrauen in den Assistenten

Die Teilnehmer sind gefragt worden, ob sie glauben, dass ein solcher Assistent sie unterstützen könnte, ob sie seinen Antworten vertrauen würden und ob sie diese in jedem Fall überprüfen wollen würden. Diese Antworten werden in Abhängigkeit zu ihrer Nutzung von ChatGPT oder ähnlichen Assistenten im Alltag abgebildet. Dies folgt aus der Überlegung, dass Personen, die solche Assistenten im Alltag verwenden, möglicherweise ein allgemein höheres Vertrauen in diese Technologie haben und umgekehrt. Abbildung 4.2 stellt die Ergebnisse da.

Die Antworten variieren weniger stark als in Abbildung 4.1. Die Werte von „Könnte unterstützen“ steigen mit der Häufigkeit der Nutzung im Alltag. Die Werte von „Vertrauen auf Korrektheit“ zeigen eine ähnliche Tendenz. Die Werte beider Mengen sind jedoch eher klein. Fast alle Teilnehmer wiederum zeigen einen großen Bedarf, die Antworten des Assistenten zu überprüfen.

Insgesamt lässt sich schlussfolgern, dass die meisten Teilnehmer bezweifeln, dass der Assistent zuverlässige Antworten generieren kann. Selbst Teilnehmer, die Assistenten wie ChatGPT häufig im Alltag einsetzen, schätzen ihr Vertrauen in einen solchen Assistenten als eher gering ein.



**Abbildung 4.2.:** Vertrauen in den Assistenten

Um diese Bedenken zu adressieren, werden folgende Anforderungen an den Assistenten gestellt:

1. Die Antworten des Assistenten sollen möglichst korrekt sein.
2. Die Antworten des Assistenten sollen möglichst nachvollziehbar sein.
3. Die Informationsquelle, auf der die Antworten basieren, soll aus der Applikation heraus einsehbar und aufrufbar sein.

#### 4.2.3. Relevante Dokumente

Die Teilnehmer sind konkret gefragt worden, für welche Dokumente sie den Assistenten gerne verwenden würden. Dabei sind in der Fragestellung die Vorlesungswebsite, Übungsblätter, Protokolle und der Chat-Verlauf zwischen den Tutoren und Übungsleitern als Vorschläge gegeben worden.

Fünf Teilnehmer schlagen die Übungsblätter vor, drei die Vorlesungswebsite, drei den Chat-Verlauf, zwei die Vorlesungsfolien, zwei die Protokolle, zwei die Java Dokumentation und einer das Programmieren-Wiki.

Die Übungsblätter haben die meiste Zustimmung, weshalb diese vom Assistenten besonders unterstützt und bei optimiert werden sollen. Zudem sollen auch andere PDFs wie die Vorlesungsfolien unterstützt werden, auch wenn diese bei der Optimierung nicht im Fokus liegen. Des Weiteren sollen Websites, Markdown-Inhalte und Media-Wiki-Inhalte unterstützt werden. Diese können auf ähnliche Weise verarbeitet und optimiert werden, sodass eine große Anzahl an Dokumenten damit unterstützt und optimiert werden kann. Dazu gehören die Vorlesungswebsite, das Programmieren-Wiki, die Protokolle und weitere Notizen. Das Format des Chat-Verlaufs unterscheidet sich stark von dem der anderen genannten Dokumente. Aufgrund der zeitlichen Begrenzung der Arbeit wird er daher nicht weiter betrachtet. Die Java-Dokumentation ist besonders umfangreich. Zudem sind die verwendeten Sprachmodelle bereits ohne weitere Kontexte in der Lage, tiefgehende Fragen zur Programmiersprache Java zu beantworten. Sie wird ebenfalls nicht an den Assistenten angebunden.

#### 4.2.4. Benutzerschnittstelle

Für die Wahl der Benutzerschnittstelle haben die Teilnehmer der Studie zwischen Web-Applikation, einem Chatbot im Element<sup>1</sup>-Chat oder einem eigenen Vorschlag wählen können. Da nur eine Benutzerschnittstelle entwickelt wird, ist nur eine Option als Antwort möglich gewesen.

Die vordefinierten Optionen Web-Applikation und Chatbot sind aufgrund folgender Überlegungen vorgeschlagen worden. Eine Web-Applikation kann flexibel gestaltet werden und kann neben einer Chat-Funktion weitere Funktionen anbieten. So ist es bspw. einfach, in einer Web-Applikation automatisch als relevant eingestufte Inhalte anzuzeigen. Der Chatbot wäre hingegen direkt im Element-Chat, über den die Tutoren miteinander kommunizieren, verfügbar. Hier suchen die Tutoren ohnehin bereits nach Antworten auf ihre Fragen, weswegen dies einen besonders schnellen Zugriff auf den Assistenten ermöglichen könnte.

In der Studie haben neun der Teilnehmer die Web-Applikation und drei den Chatbot gewählt. Weitere Vorschläge sind nicht genannt worden. Folglich wird entschieden, eine Web-Applikation zu entwickeln.

#### 4.2.5. Funktionen des Assistenten

Vor der Durchführung der Studie sind drei Funktionen überlegt worden, die der Assistent anbieten könnte:

1. Antworten auf Anfragen geben (Chat-Funktion)
2. Automatisch als relevant eingestufte Inhalte anzeigen

---

<sup>1</sup><https://element.io/>

### 3. Automatisch häufig gestellte Fragen beantworten

Die Teilnehmer sind in der Studie gefragt worden, welche dieser Funktionen sie hilfreich finden würden. Da mehrere der Funktionen implementiert werden können, ist eine Mehrfachauswahl möglich gewesen. Zudem ist es angeboten worden, eigene Vorschläge anzugeben.

Sechs der Teilnehmer haben für eine Chat-Funktion gestimmt, neun für das automatische Anzeigen als relevant eingestufte Informationen und vier für das automatische Anzeigen häufig gestellter Fragen. Weitere Vorschläge wurden nicht genannt. Da die Zustimmung bei den ersten beiden Funktionen bei mindestens der Hälfte der Teilnehmer liegt, die Zustimmung bei der dritten Funktion jedoch nur bei einem Drittel, wird entschieden, die Chat-Funktion und das automatische Anzeigen als relevant eingestufte Inhalte zu implementieren.

**Auswahl der automatisch angezeigten Informationen** Es ist wichtig, dass die Informationen übersichtlich dargestellt werden können. Dadurch sollen die Benutzer unmittelbar erkennen, was sie bei ihrer Arbeit beachten müssen. Es soll sich daher auf Informationen beschränkt werden, die von allgemeiner Relevanz für alle Tutoren sind. Zudem soll es sich um Informationen handeln, die Erinnerungen an Aufgaben oder Ereignisse darstellen, statt inhaltliche Fragen vorweg zu nehmen. Daher wird entschieden, Kalendereinträge bestehend aus Terminen und Fristen darzustellen. Dabei handelt es sich um Informationen, bei denen eine zeitnahe Erinnerung besonders wichtig ist, um die Arbeit zuverlässig zu erledigen. Zudem können diese Informationen über verschiedene Datenquellen verteilt sein. Dadurch steigt der Mehrwert dieser Funktion für die Benutzer.

### 4.3. Liste der Anforderungen

Die zum Ende der Anforderungsanalyse festgestellten Anforderungen sind in Tabelle 4.1 zusammengetragen.

Diese Anforderungen werden aus der Analyse der Tätigkeiten der Tutoren und den Ergebnissen der Studie abgeleitet. Sie sollen sowohl die Funktionalität als auch eine positive Benutzererfahrung sicherstellen. Einige der Anforderungen sind nur vage formuliert worden, da sie nicht im Fokus der Arbeit stehen und eine Konkretisierung besonders schwierig ist.

Die Anforderungen 2.10, 3.10, 3.20 und 4.10 beschreiben die Hauptfunktionen, die der Assistent erfüllen soll. Die Anforderungen 1.\*, 2.20, 2.30 und 2.50 beschreiben, wie der Benutzer mit dem Assistenten interagieren können soll, um diese Hauptfunktionen einfach und schnell zu verwenden. Die Anforderungen 2.40, 3.30 und 3.40 sollen ein möglichst hohes Vertrauen der Benutzer in den Assistenten sicherstellen.

ID	Beschreibung
1.10	Es soll eine Web-Applikation als Benutzerschnittstelle implementiert werden.
1.20	Die Web-Applikation soll einfach zu bedienen sein.
1.30	Die Benutzer sollen unabhängige, durch einen Login gesicherte Bereiche haben.
2.10	Die Benutzer sollen Antworten auf Fragen bekommen (Chat-Funktion).
2.20	Die Antworten sollen schnell angezeigt werden.
2.30	Beim Beantworten der Fragen soll die bisherige Konversation berücksichtigt werden.
2.40	Es sollen nur Antworten ausgegeben werden, bei denen der Assistent zuversichtlich ist, dass diese korrekt sind.
2.50	Die Benutzer sollen mehrere unabhängige Konversationen führen können.
3.10	Zur Beantwortung von Fragen sollen relevante Dokumente hinzugezogen werden.
3.20	Die Anbindung und Abfrage der Dokumente soll möglichst optimiert werden, um zuverlässige und hilfreiche Antworten zu liefern.
3.30	Es soll einsehbar sein, auf welchen Dokumenten die Antworten basieren.
3.40	Der Benutzer soll auf diese Dokumente aus der Web-Applikation aus zugreifen können.
4.10	Der Assistent soll automatisch Termine und Fristen anzeigen (Kalender-Funktion).

**Tabelle 4.1.:** Liste der Anforderungen



# 5. Entwicklung des Assistenten

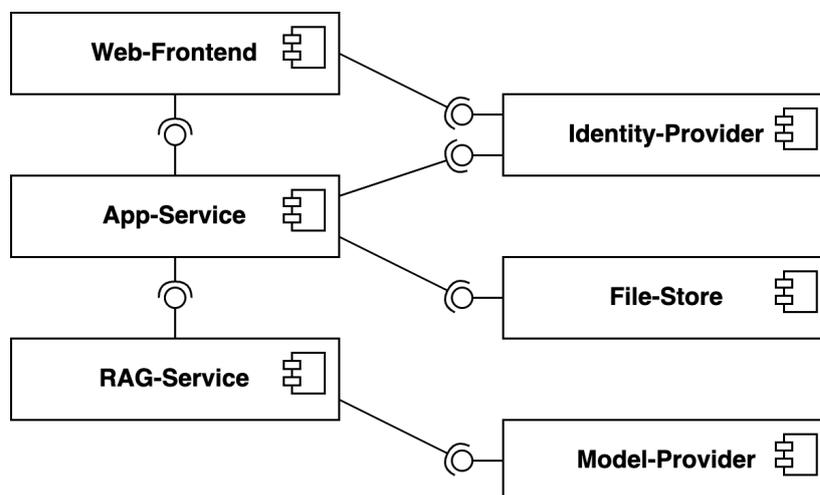
Der Assistent besteht aus mehreren Komponenten, bei denen es sich zum Teil um eigene Entwicklungen und zum Teil um wiederverwendete Komponenten handelt. In Abschnitt 5.1 wird zunächst ein Überblick über den Assistenten vermittelt. In den anschließenden Abschnitten werden die eigens entwickelten Komponenten einzeln vorgestellt.

## 5.1. Entwurf

In diesem Abschnitt werden die grundlegende Architektur und die zentralen Abläufe des Assistenten vorgestellt.

### 5.1.1. Architektur

Es wurde eine serviceorientierte Architektur gewählt. Abbildung 5.1 stellt ihre Zusammenhänge in einem Komponenten-Diagramm dar.



**Abbildung 5.1.:** Komponenten-Diagramm des Assistenten

Bei den Komponenten RAG-Service, App-Service und Web-Frontend handelt es sich um eigene Entwicklungen, während für Model-Provider, File-Store und Identity-Provider bestehende Lösungen von externen Anbietern verwendet worden sind. Die Aufteilung des Backends in fünf Komponenten hat mehrere Vorteile:

1. Wiederverwendung von Komponenten und Wissen
2. Verwendung unterschiedlicher Technologien
3. Besserer Fokus auf die jeweiligen Aspekte

Als Identity-Provider kommt beim Assistenten Keycloak<sup>1</sup> zum Einsatz. Er stellt die Benutzerverwaltung und Authentifizierung zur Verfügung.

Für den File-Store wird SeaweedFS<sup>2</sup> verwendet. Er erleichtert das Speichern von Dateien sowie den Zugriff auf gespeicherte Dateien durch verschiedenen Services des Assistenten.

Als Model-Provider werden hier Server bezeichnet, die Sprach- und Einbettungsmodelle zur Verfügung stellen. Insbesondere werden Modelle von OpenAI<sup>3</sup> verwendet. In manchen Versuchen kommen auch Modelle zum Einsatz, die von einem KIT-Server mittels Ollama<sup>4</sup> zur Verfügung gestellt werden.

Das Web-Frontend kommuniziert ausschließlich mit dem Identity-Provider und dem App-Service. Über den Identity-Provider können sich Benutzer registrieren und einloggen. Der App-Service stellt anwendungsspezifische Funktionalität zur Verfügung und agiert zudem als Zwischenschicht zur Kommunikation mit anderen Backend-Services. Insbesondere kümmert er sich um die Authentifizierung und Autorisierung von Backend-Anfragen mithilfe des Identity-Providers. Er greift auf einen File-Store zu, um Dateien zu speichern. Des Weiteren agiert er mit dem RAG-Service, welcher den Kern der Funktionalität des Assistenten implementiert. Durch das Auslagern der anwendungsspezifischen Funktionalität in den App-Service fokussiert sich der RAG-Service ausschließlich auf die Anbindung an Sprachmodelle unter Verwendung von RAG. Zudem ist er weitgehend zustandslos, was die Entwicklung vereinfacht und den Service robuster macht.

Die Backend-Services werden zusammen mit einem Reverse-Proxy in einem Docker-Netzwerk deployed. Der Reverse-Proxy ist dabei der einzige Service, welcher eine Verbindung nach außen herstellt. Er leitet alle Anfragen an den App-Service oder den Identity-Provider. Die anderen Services sind somit nur innerhalb des Docker-Netzwerk erreichbar.

### 5.1.2. Abläufe

Es werden vier Abläufe definiert, die die Hauptfunktionen des Assistenten bereitstellen.

---

<sup>1</sup><https://www.keycloak.org>

<sup>2</sup><https://seaweedfs.com>

<sup>3</sup><https://openai.com>

<sup>4</sup><https://ollama.com>

**Einbettung von Ressourcen** Der Assistent soll die Einbettung von Websites und Dateien unterstützen. Diese werden im Folgenden als Ressourcen bezeichnet. Ein Administrator muss definieren welche Ressourcen eingebettet werden sollen und auf welche Art und Weise diese eingebettet werden sollen. Dies geschieht über Konfigurationsdateien. Während der Assistent auf Website-Ressourcen über URLs zugreifen kann, müssen Datei-Ressourcen und die Konfigurationsdateien erst über das Web-Frontend hochgeladen und vom App-Service verwaltet werden. Anschließend kann der Administrator die Einbettung ausführen. Dabei werden die Konfigurationsdateien vom App-Service ausgewertet und die notwendigen Informationen an den RAG-Service übertragen. Dieser führt mithilfe von einem Vectorstore und einem Einbettungsmodell die Einbettung durch.

**Laden des Kalenders** Ein Administrator kann über das Web-Frontend einen neuen Kalender generieren. Diese Anfrage wird über den App-Service an den RAG-Service delegiert. Dieser generiert basierend auf den eingebetteten Dokumenten und einem Sprachmodell einen neuen Kalender im JSON-Format und gibt diesen an den App-Service zurück. Der App-Service speichert den Kalender, sodass er nicht ständig neu erstellt werden muss, und gibt ihn an das Web-Frontend weiter.

**Nachrichten senden und empfangen** Benutzer können über das Web-Frontend Nachrichten in separaten Chats mit dem Assistenten austauschen. Der App-Service nimmt vom Benutzer gesendete Nachrichten entgegen, speichert sie und sendet sie als Anfrage zusammen mit dem bisherigen Chat-Verlauf an den RAG-Service. Dieser generiert basierend auf den eingebetteten Dokumenten und einem Sprachmodell eine Antwort und gibt diese als Event-Stream zurück. Dabei wird die Antwort nicht erst zurückgegeben, nachdem sie vollständig generiert wurde. Stattdessen wird jeder Token der Nachricht, sobald er generiert worden ist, als Event ausgegeben. Der Assistent leitet diesen Stream an das Web-Frontend weiter und speichert die Antwort ab, sobald sie vollständig ist.

**Zusammenfassungen der Konversationen laden** Nachdem der App-Service eine vollständige Nachricht basierend auf einer Nachricht eines Benutzers vom RAG-Service erhalten hat, veranlasst er diesen, eine neue Zusammenfassung des Chats zu generieren. Dazu übergibt er den neuen Chat-Verlauf an den RAG-Service, welcher eine Antwort mithilfe eines Sprachmodells generiert. Anschließend gibt er, statt einen Event-Stream zu verwenden, die gesamte Zusammenfassung zurück, welche vom App-Service gespeichert wird und im Event-Stream der Antwort auf die ursprüngliche Nachricht des Benutzers an das Web-Frontend weitergegeben wird.

## 5.2. Web-Frontend

Das Web-Frontend wird mit den Technologien React und TypeScript umgesetzt. React gehört zu den verbreitetsten Bibliotheken für das Erstellen dynamischer Web-Applikationen. Sie ist aufgrund des großen Community-Supports, vieler kompatibler Bibliotheken und meiner bisherigen Erfahrung mit ihr ausgewählt worden. TypeScript ermöglicht eine typsichere

Implementierung wodurch im Vergleich zu JavaScript besonders die Wartbarkeit erhöht und die Fehleranfälligkeit gesenkt wird.

### 5.2.1. Liste der Funktionen

Folgende Funktionen stehen den Tutoren im Frontend zur Verfügung:

1. Kalender einsehen
2. Starten eines Chats mit einem Sprachmodell
3. Überblick über bisherige Chats mit Titel und Zusammenfassung anzeigen lassen
4. Öffnen bisheriger Chats
5. Nachrichten schreiben in einem geöffneten Chat
6. Löschen bisheriger Chats
7. Nachrichten des Sprachmodells in einem geöffneten Chat auswählen
8. Feedback zur ausgewählten Nachricht geben
9. Quellen der Information der ausgewählten Nachricht anzeigen lassen
10. Quellen öffnen
11. Zusammenfassung von geöffnetem Chat anzeigen lassen
12. Dokumente anzeigen lassen, auf die der Assistent zugreifen kann

Benutzer mit der Rolle „document-manager“ können zusätzlich

13. Dateien hochladen
14. Dateien löschen
15. Dokument-Einstellungen hochladen
16. Dokument-Einstellungen löschen
17. den Einbettungsprozess starten
18. Einbettungen löschen
19. die Einbettung für bestimmte Dokumente wiederholen
20. einen neuen Kalender erstellen

Diese Benutzer werden im Folgenden als Document-Manager bezeichnet.

## 5.2.2. Beschreibung der Funktionen und Benutzerschnittstelle

Das Web-Frontend besteht aus drei Seiten, mit denen die Funktionen aus Unterabschnitt 5.2.1 zur Verfügung gestellt werden.

Der Kalender dient als Erinnerung für wichtige Termine und Fristen und wird daher auf allen drei Seiten auf der links angezeigt. Er wird, wie in Unterabschnitt 5.4.8 beschrieben, basierend auf den eingebetteten Dokumenten generiert. Generierte Kalender werden im App-Service gespeichert, um nicht ständig neu generiert werden zu müssen. Document-Manager bekommen den Button mit dem Neu-Laden-Icon oben rechts angezeigt. Damit wird ein neuer Kalender generiert. Dies ist vor allem dann sinnvoll, wenn neue Dokumente eingebettet worden sind oder der generierte Kalender aufgrund der nicht-deterministischen Antworten des Sprachmodells nicht zufriedenstellend ist.

Abbildung 5.2 zeigt eine Übersicht über alle bisherigen Chats. Dazu wird für jeden Chat eine Karte mit Titel, Untertitel und Zusammenfassung angezeigt. Die Idee ist, dass ein Sprachmodell einen aussagekräftigen Titel generiert sowie einen Untertitel, der das Informationsbedürfnis des Benutzers beschreibt und eine Zusammenfassung der Ergebnisse. Durch das Schreiben und Senden einer Nachricht mit dem Textfeld unten wird ein neuer Chat angelegt und geöffnet.

The screenshot displays a web interface with three main sections:

- Kalender (30):** A table with columns 'Datum' and 'Ereignis'. It lists dates from 01.10.2024 to 11.12.2024 with corresponding events like 'Start Anmeldung im Campus-Management-Portal' and 'Übungsblatt 4 Abgabe'.
- Chats:** A list of chat cards, each with a title, subtitle, and summary. Examples include 'Bewertung von JavaDoc-Kommentaren', 'Empfohlene Verschachtelungstiefe in der Programmierung', and 'Übungsblätter und Termine'.
- Dokumente:** A section containing a message input field labeled 'Nachricht' and a 'Senden' button.

Abbildung 5.2.: Bedarf nach einfachem Informationsgewinn

Abbildung 5.3 zeigt einen geöffneten Chat an. Der Hauptinhalt dieser Seite ist der Chat-Verlauf des Benutzers mit dem Sprachmodell sowie die Eingabe für neue Nachrichten an das Sprachmodell. Während das Sprachmodell eine Antwort und die neue Zusammenfassung generiert, können keine neuen Nachrichten geschrieben und gesendet werden. Dies soll eine bessere Struktur in den Chat bringen und Fehler vorbeugen. Somit wird für jede Nachricht des Benutzers eine Nachricht des Sprachmodells generiert. Durch das Klicken

auf eine Nachricht des Sprachmodells wird diese ausgewählt. Dadurch wird die Eingabe zum Feedback eingeblendet. Darüber können Tutoren insbesondere in der Nutzerstudie die Evaluierung von Chat-Verläufen unterstützen. Des Weiteren werden für die ausgewählte Nachricht in der rechten Seitenleiste die Quellen der Nachricht angezeigt. Dabei handelt es sich um die Dokumente, die der Nachricht des Benutzers an das Sprachmodell als Kontexte angehängt worden sind. Die Quellen können direkt über das Web-Frontend geöffnet werden. So können Tutoren besonders schnell Informationen verifizieren oder weitere Informationen erlangen.

The screenshot shows a web interface titled "Bewertung von JavaDoc-Kommentaren". On the left is a calendar for 2024. The main area contains a question: "Der Student hat setAge neben der Beschreibung des Parameters nur mit 'Sets age of the person' im JavaDoc. Ab welchem Blatt muss ich dafür Punkte abziehen? Sollte ich überhaupt dafür Punkte abziehen, oder findest du, das reicht?". Below the question are evaluation criteria and recommendations. On the right, a "Quellen (8)" panel lists sources like "Programmierenwiki: JavaDoc: Trivial" with relevance scores. At the bottom, there is a "Feedback" section with a star rating and a "Nachricht" input field.

**Abbildung 5.3.:** Bedarf nach einfachem Informationsgewinn

Abbildung 5.4 zeigt die Seite zum Einsehen und Verwalten von Dokumenten. Tutoren wird nur die Liste der eingebetteten Dokumente ohne weitere Buttons angezeigt. Document-Manager können zusätzlich die Leiste mit den Uploads sowie alle Buttons sehen und damit die Dokumente verwalten. Um Inhalte einzubetten, müssen sie über Settings-Dateien konfiguriert werden. Dabei wird zwischen der Main-Setting und den Value-Settings unterschieden. Die Main-Setting beinhaltet die eigentliche Konfiguration, während die Value-Settings Werte für Variablen in der Main-Setting bereitstellen. Es kann mehrere Dateien für Value-Settings geben, jedoch nur eine Main-Setting. In der Main-Setting werden entweder Websites oder Dateien referenziert, die eingebettet werden sollen. Referenzierte Dateien müssen erst als Ressourcen hochgeladen werden. Über den Einbettungs-Button wird die Konfiguration angewandt. Dabei werden nur solche Dokumente eingebettet, die noch nicht eingebettet worden sind. Dies wird anhand des Titels erkannt, welcher für jeden Inhalt in der Main-Setting hinterlegt ist. Über die Buttons in der Liste der eingebetteten Dokumente kann ein bestehendes Dokument neu eingebettet oder gelöscht werden.

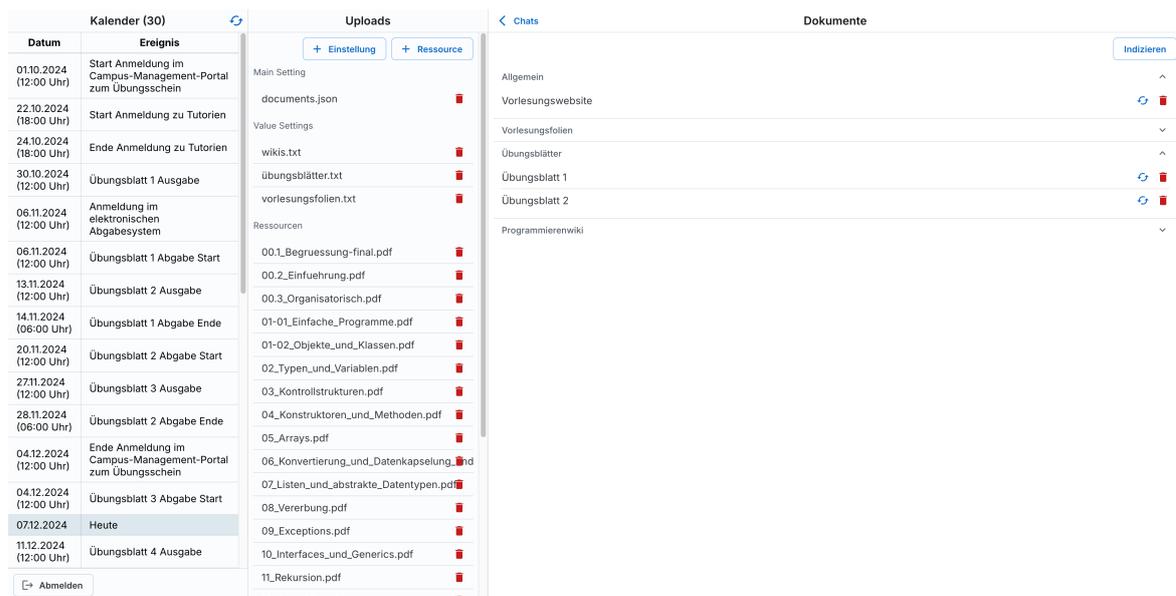


Abbildung 5.4.: Bedarf nach einfachem Informationsgewinn

## 5.3. App-Service

Der App-Service wird mit den Technologien Spring Boot und Kotlin entwickelt. Damit lassen sich in kurzer Zeit robuste Applikationen auf einer modernen Code-Basis entwickeln. Zu seinen Aufgaben gehören

1. Authentifizieren und Autorisieren von Backend-Anfragen
2. Interaktion mit dem RAG-Service
3. Speichern von Chats
4. Speichern des Kalenders
5. Verwalten von Document-Settings
6. Verwalten von Datei-Ressourcen

Chats bestehen aus einem Verlauf und einer Zusammenfassung. Der Verlauf besteht aus Nachrichten, die wiederum aus einem Text, einem Verfasser (Der Benutzer oder das Sprachmodell), Kontexte und Feedback bestehen. Der App-Service sendet Nachrichten des Benutzers zusammen mit dem Verlauf an den RAG-Service. Er speichert die Antwort des RAG-Service und lädt und speichert eine neue Zusammenfassung des neuen Verlaufs vom RAG-Service.

Die Berechnung des Kalenders ist nicht deterministisch und kann aufwendig sein. Daher soll der generierte Kalender im App-Service gespeichert werden. Der App-Service lädt auf Anfrage einen neuen Kalender vom RAG-Service und speichert ihn und stellt ihn so dem Web-Frontend zur Verfügung.

Die Settings-Dateien, insbesondere die Main-Setting muss ausgelesen und ausgewertet werden. Bei Anfrage zum Einbetten der Ressourcen müssen die aus ihr resultierenden Anfragen an den RAG-Service gestellt werden.

Datei-Ressourcen werden mithilfe des File-Stores gespeichert und im App-Service verwaltet. Ein großer Vorteil der Verwendung des File-Stores ist, dass die Dateien zum Einbetten dem RAG-Service einfach zur Verfügung gestellt werden können. Dazu wird dem RAG-Service die jeweilige URL der Datei übergeben. Dieser Mechanismus ist nicht spezifisch für den File-Store, sondern würde mit jeder validen URL funktionieren, die zu einer Datei führt. Somit besteht auf Komponenten-Ebene keine Abhängigkeit zwischen dem RAG-Service und dem File-Store.

### 5.4. RAG-Service

Der RAG-Service ist für das Einbetten von Dokumenten, Beantworten von Fragen, sowie das Laden des Kalenders und Chat-Zusammenfassungen mithilfe von Sprachmodellen und RAG zuständig. In diesem Kapitel werden zuerst das Vorgehen bei der Entwicklung sowie LangChain, das verwendete Framework, beschrieben. Anschließend werden die Versuche und Artefakte der Optimierung des RAG-Services erläutert.

#### 5.4.1. Vorgehen

In dieser Arbeit wird das Framework LangChain (siehe Unterabschnitt 5.4.2) verwendet, um mit Sprachmodellen und Vectorstores zu interagieren sowie RAG-Methoden umzusetzen.

Zuerst wird damit ein Prototyp gebaut, an dem erste Versuche durchgeführt werden können. Dieser verwendet bereits RAG, um Fragen zu Inhalten zu beantworten. Er wird durch eine einfache Benutzeroberfläche basierend auf der Bibliothek `gradio`<sup>5</sup> zur Verfügung gestellt. Der Prototyp ermöglicht es, frühzeitig die Möglichkeiten von Sprachmodellen mit RAG zu untersuchen. Eine wichtige Erkenntnis ist, dass führende Modelle die meisten Fragen korrekt beantworten können, auch wenn viele Kontexte gegeben sind, die zur Beantwortung der Frage irrelevant sind. Das Abrufen von Dokumenten aus dem Vectorstore stellt jedoch eine besondere Herausforderung dar. Ohne Optimierung werden bei einigen Anfragen an den Vectorstore die falschen Dokumente abgerufen, obwohl es scheinbar offensichtlich ist, welches die richtigen Dokumente sind (siehe Unterabschnitt 5.4.4.1).

Anschließend an die ersten Versuche mit dem Prototypen wird der eigentliche Assistent und damit auch der RAG-Service entwickelt. Die Entwicklung profitiert dabei von den ersten Erfahrungen mit LangChain sowie den Erkenntnissen der Versuche. Bei der Architektur des RAG-Services ist die Austauschbarkeit der Komponenten ein wichtiger Aspekt. Insbesondere lassen sich dadurch das Sprachmodell, das Einbettungsmodell, der Vectorstore, sowie die Einbettungs- und die Abrufstrategie leicht konfigurieren.

---

<sup>5</sup><https://www.gradio.app>

Während der Entwicklung und Optimierung des RAG-Services werden diverse Techniken implementiert und evaluiert. Dies wird in späteren Unterabschnitten beschrieben. Die einfache Konfiguration des RAG-Services ermöglicht es, auch im finalen Produkt alle Techniken zu verwenden. Somit kann das Vorgehen bei der Evaluationen der einzelnen Techniken wiederholt werden. Die dafür notwendige Konfiguration bezieht sich dabei meistens auf die verwendete Einbettungs- und die Abrufstrategie. Wenn nicht anders angegeben, sind die anderen Parameter der Konfiguration bei den Evaluationen der Techniken immer wie folgt gesetzt worden:

- Sprachmodell: OpenAIs GPT-4o mit Temperaturwert von 0
- Einbettungsmodell: OpenAIs text-embedding-3-large
- Vectorstore: Chroma

Aufgrund des nicht-deterministischen Verhaltens von Sprachmodellen und Suchen in Vectorstores können sich Ergebnisse trotz identischer Konfiguration leicht unterscheiden. Um dennoch möglichst zuverlässige Ergebnisse zu erhalten, werden Versuche stets mehrfach durchgeführt.

### 5.4.2. LangChain

LangChain ist eine aktuelle und verbreitete Bibliothek, um mit Sprachmodellen zu interagieren. Sie bietet diverse Funktionen für verschiedene Anwendungen, wie bspw. RAG. Des Weiteren stellt sie einige Grundfunktionalitäten zur Verfügung, mit denen man den gesamten Prozess einer Anfrage an ein Sprachmodell implementieren kann.

Eine Anfrage an ein Sprachmodell beginnt typischerweise mit ihrer Erstellung. Im einfachsten Falle handelt es sich dabei nur um die Anfrage, die der Benutzer im System eingegeben hat. In vielen Fällen werden jedoch noch weitere Instruktionen und möglicherweise die bisherige Konversation in die Anfrage mit aufgenommen. LangChain bietet Möglichkeiten, solche Anfragen basierend auf sogenannten Prompt-Templates oder bisherigen Nachrichten zu erstellen.

Im Falle einer RAG-basierten Anwendung wie diesem Assistenten müssen vor der Anfrage an das Sprachmodell relevante Kontexte abgefragt werden. Dazu bietet LangChain Integrationen für diverse Vectorstores wie Chroma an. Für die Einbettung von Inhalten in den Vectorstore stellt LangChain sogenannte Document-Loader zur Verfügung. Für verschiedene Formate wie bspw. Websites, PDFs, Markdown, etc. gibt es unterschiedliche Document-Loader. Zuerst laden sie die Inhalte, bspw. einer Website oder Datei. Manche unterteilen diesen anschließend in mehrere Chunks. Das Resultat sind ein oder mehrere Dokumente, die von einem Vectorstore eingebettet werden können. Zudem stellt es verschiedene sogenannte Retriever zur Verfügung, mit denen die Vectorstores auf unterschiedliche Weise nach den gewünschten Kontexten durchsucht werden können. Die abgerufenen Kontexte können über Platzhalter in den Prompt-Templates den Sprachmodell-Anfragen angehängt werden.

### 5.4.3. Einbettung von Inhalten

Damit der Assistent Fragen zu Inhalten beantworten kann, müssen diese erst als Dokumente eingebettet werden. Diese Funktionalität stellt der RAG-Service über seine API zur Verfügung.

Das Erstellen von Dokumenten, die eingebettet werden können, erfolgt basierend auf dem Document-Loader Prinzip aus dem LangChain Framework. Diese Document-Loader laden zunächst die Inhalte und wandeln sie daraufhin in Dokumente um. Diese werden anschließend an den Vectorstore übergeben. Mithilfe eines Einbettungsmodells erzeugt dieser aus ihnen Einbettungen und speichert diese ab. Welcher Document-Loader verwendet werden soll, muss der API-Anfrage übergeben werden.

Die Document-Loader des Assistenten laden entweder den Inhalt einer Datei oder einer Website. Document-Loader, die Inhalte aus Dateien laden, bekommen eine URL übergeben, von der sie die Datei laden. Diese wird der API-Anfrage übergeben. Document-Loader, die Inhalte einer Website laden, bekommen neben der URL noch weitere Parameter übergeben, die das HTML-Element spezifizieren, dessen Inhalt geladen werden soll. Diese Informationen müssen ebenfalls der API-Anfrage übergeben werden und müssen letztlich vom Administrator des Assistenten spezifiziert werden.

Die geladenen Inhalte werden abhängig vom Document-Loader auf unterschiedliche Weise in Dokumente umgewandelt. Eine einfache Möglichkeit ist, den geladenen Inhalt unverändert als den Inhalt des Dokuments zu verwenden. Der Prototyp zeigt jedoch, dass bei Verwendung einfacher Methoden häufig die falschen Dokumente bei einem Abruf zurückgegeben werden (siehe Unterabschnitt 5.4.1). Das zugrunde liegende Problem ist, dass bei einer Ähnlichkeitssuche zwischen den eingebetteten Dokumenten und den Anfragen an den Vectorstore die Ähnlichkeit zu den falschen Dokumenten oft höher ist als zu den richtigen. Im folgenden Unterabschnitt wird dies zunächst demonstriert. Anschließend werden verschiedene Methoden entwickelt, um dieses Problem zu lösen. Ein wichtiger Ansatz dabei ist, die Inhalte nach Überschriften zu zerteilen. Die dabei resultierenden Chunks werden dann in Dokumente umgewandelt und eingebettet. Durch die thematische Unterteilung soll die Ähnlichkeit zwischen der Anfrage und einem bestimmten Thema erhöht werden.

Um Inhalte nach Überschriften zu zerteilen muss für jede vom Assistenten unterstützte Art des Inhalts eine Strategie implementiert sein, die den Inhalt zerteilt. Der Assistent unterstützt die Unterteilung nach Überschriften für folgende Inhalte:

1. Übungsblätter als PDFs
2. HTML
3. Media-Wiki
4. Markdown

Zudem unterstützt er einfache, generische Document-Loader für Websites und PDFs, den WebBaseLoader und den PyPDFLoader von LangChain. Diese werden wie auch die Strategien zur Unterteilung nach Überschriften im folgenden Unterabschnitt erläutert.

Für die Tests, die bei den Optimierungen in den folgenden Unterabschnitten am RAG-Service durchgeführt werden, werden immer die Vorlesungsmaterialien aus dem Wintersemester 2023/2024 verwendet. Das kommt daher, dass die Materialien zum Zeitpunkt der Arbeit vollständig vorhanden sind. Zudem wird die anschließende Evaluation (siehe Kapitel 6) auf dem Stand vom aktuellen Wintersemester 2024/2025 durchgeführt. So kann überprüft werden, ob sich die Optimierungen auf andere Daten übertragen lassen.

#### 5.4.4. Abruf der Dokumente für die Chat-Funktion

Für die Optimierung der Chat-Funktion werden verschiedene Strategien zum Abruf der Dokumente aus dem Vectorstore implementiert und evaluiert. Dies stellt einen besonderen Schwerpunkt dieser Arbeit dar.

##### 5.4.4.1. Einfaches Laden und Abrufen

Für erste Versuche soll ein einfacher RAG-Prozess implementiert werden.

**RAG mit einem Dokument** Zuerst soll nur die Vorlesungswebsite eingebettet werden. Für diese Einbettung wird der WebBaseLoader von LangChain verwendet. Diesem Document-Loader wird die URL der einzubettenden Website übergeben. Er lädt den Inhalt und erstellt aus ihm ohne weitere Verarbeitung ein Dokument, welches anschließend eingebettet werden kann. Somit befindet sich in diesem ersten Versuch nur ein Dokument im Vectorstore. Der Retriever ist so konfiguriert, dass er bei einer Vectorstore-Anfrage die fünf ähnlichsten Dokumente abrufen. Somit wird bei einem Abruf die Vorlesungswebsite immer unabhängig von ihrer Ähnlichkeit zur Anfrage zurückgegeben.

Es werden nun folgende Fragen an den Assistenten gestellt:

1. Wie viele Übungsblätter gibt es?
2. Wann findet die Präsenzübung statt?
3. Wann findet die Vorlesung statt?
4. Wann findet die erste reguläre Vorlesung statt?
5. Lernen die Studierenden Lambda-Ausdrücke in der Vorlesung?

Diese Fragen decken einen großen Teil des Inhalts der Vorlesungswebsite ab. Zudem eignen sich diese Fragen neben dem Testen des Abrufs auch zum Testen der Antworten des Sprachmodells. Die zweite Frage lässt sich unmittelbar aus dem Inhalt beantworten. Für Frage eins, drei und vier muss das Sprachmodell mehrere Informationen zusammenführen. Frage fünf lässt sich aus dem Inhalt nicht sicher beantworten. In solchen Fällen soll der Assistenten den Benutzern dies mitteilen, statt eine falsche Antwort zu geben.

Diese Fragen werden jeweils erst an den Vectorstore und anschließend mitsamt der gefundenen Vorlesungswebsite als Kontext an das Sprachmodell gerichtet. Dabei handelt es sich

um einen einfachen RAG-Prozess (siehe Absatz 3.2). Der Retriever, der dabei verwendet wird, wird im Folgenden BaseRetriever genannt. Da die Antworten von Vectorstores und Sprachmodellen häufig nicht deterministisch sind, wird dies für jede Frage zehn Mal durchgeführt. In jedem Fall kann das Sprachmodell die Frage korrekt beantworten. Dies zeigt, dass der Abruf-Prozess funktioniert und das Sprachmodell in der Lage ist, die richtigen Informationen aus dem Kontext zu extrahieren.

**RAG mit mehreren Dokumenten** Im nächsten Schritt werden neben der Vorlesungswebsite auch die sechs Übungsblätter eingebettet. Für diese Einbettung wird der PyPDFLoader von LangChain verwendet. Dieser lädt die PDF-Dateien über eine URL. Anschließend unterteilt er die Dateien seitenweise in einzelne Chunks. Somit werden pro PDF-Datei mehrere Dokumente erzeugt und eingebettet. Jedoch werden weiterhin nur maximal fünf Dokumente abgerufen. Es werden zunächst dieselben fünf Fragen wieder jeweils zehn Mal gestellt. Um die Fragen diesmal richtig zu beantworten, muss in jedem Fall die Vorlesungswebsite abgerufen werden. Sie ist das einzige Dokument, welche dem Sprachmodell die notwendigen Informationen zur Verfügung stellen kann.

Bei der ersten und zweiten Fragen werden beim Abruf in jedem Fall ausschließlich Seiten der PDFs gefunden. Damit ist es dem Sprachmodell nicht möglich, die gewünschten Antworten zu generieren. Es antwortet insofern korrekt, dass es angibt, die Frage nicht beantworten zu können oder erklärt, welche Informationen es im Kontext findet. Bei den restlichen Fragen wird die Vorlesungswebsite abgerufen. Das Sprachmodell antwortet somit korrekt.

Anschließend an die Fragen zur Vorlesungswebsite werden nun folgende Fragen jeweils zehn Mal zu den Übungsblättern gestellt:

6. In welcher Aufgabe sollen die Studierenden Hilfsmethoden für Strings schreiben?
7. Worin unterscheiden sich die Aufgaben Adventskalender und Adventskalender II?
8. Welche Bibliotheken sind auf Übungsblatt 3 erlaubt?
9. Nenne mir alle Unterüberschriften von Vier bis zwei gewinnt.
10. Worum geht es in Übungsblatt 2 Aufgabe A?

Diese Fragen testen vor allem, ob die richtigen Dokumente gefunden werden können. Frage eins und zwei prüfen, ob die richtigen Dokumente basierend auf inhaltlichen Fragen gefunden werden. Frage vier ist eine künstliche Frage, die prüft, ob die gesamte Aufgabe „Vier bis zwei“ erfasst werden kann. Die fünfte Frage ist besonders schwer zu beantworten, da der relevante Teil der Anfrage „Übungsblatt 2 Aufgabe A“ sehr ähnlich ist zu beliebigen „Übungsblatt  $x$  Aufgabe  $y$ “. Es ist jedoch denkbar, dass die Tutoren diese eigentlich sehr präzise Schreibweise im Chat mit dem Assistenten verwenden werden. Frage drei stellt dieselbe Herausforderung, bringt mit dem Begriff „Bibliotheken“ jedoch noch weiteren Kontext mit ein.

Bei den Fragen eins, zwei und drei werden in jedem Fall die richtigen Dokumente abgerufen und die Fragen korrekt beantwortet. Für Frage vier wird immer eine relevante Seite des Übungsblatts nicht abgerufen. Die Antwort des Sprachmodells ist daher unvollständig. Bei

Frage fünf wird keines der relevanten Dokumente abgerufen. Das Sprachmodell kann die Frage daher nicht beantworten, teilt dies jedoch korrekt mit.

**Schlussfolgerung** Für manche Fragen liefert der RAG-Service bereits gute Ergebnisse. Es ist jedoch gezeigt worden, dass diese insbesondere bei Einbettung mehrerer Dokumente noch verbessert werden können. Dazu muss untersucht werden, wie die Ähnlichkeit zu den gewünschten Dokumenten erhöht bzw. die Ähnlichkeit zu unerwünschten Dokumenten gesenkt werden kann. Dies kann vor allem über die Einbettung der Dokumente und die Art und Weise des Abrufs gesteuert werden.

#### 5.4.4.2. Unterteilen der Dokumente nach Überschriften

Als erste Verbesserung zum einfachen Einbetten der Inhalte wird das geschickte Unterteilen der Inhalte vor dem Einbetten untersucht. Eine häufige Methode dabei ist das Unterteilen in gleich große Chunks mit Überschneidung (siehe Absatz 3.2). Beim RAG-Service wird eine andere Strategie implementiert. Dabei werden die Inhalte nach Überschriften zergliedert. Die Idee ist, dass dies sicherstellt, dass der Kontext eines Abschnitts im Dokument stets erhalten bleibt. Gleichzeitig wird nicht mehr Kontext als der des jeweiligen Abschnitts in das Dokument aufgenommen. Dies ist vor allem dann sinnvoll, wenn die Längen der Abschnitte stark variieren, wie es bei den Inhalten der Programmierenvorlesung der Fall ist. Des Weiteren muss beim Unterteilen nach Überschriften keine optimale Länge der Chunks sowie Länge der Überschneidung ermittelt werden. Diese könnten sich zudem ändern, wenn andere Inhalte verwendet werden als beim Testen.

Für das Unterteilen in Überschriften von HTML, Markdown und Media-Wiki wird eine generische Funktion implementiert. Dieser wird ein regulärer Ausdruck übergeben, über den sie Überschriften im ebenfalls übergebenen Inhalt erkennt. Zudem erkennt sie den Grad der Überschrift (1. Überschrift, 2. Unterüberschrift, 3. Unterunterüberschrift, etc.). Beim RAG-Service werden nur nach Überschriften bis Grad drei unterteilt, um nicht zu kleine Chunks zu erstellen, die womöglich wichtigen Kontext wieder verlieren. Beim Erstellen eines Chunks wird der Inhalt der jeweiligen Überschrift als Inhalt des Chunks verwendet. Zusätzlich werden zu Beginn jedes Chunks die Überschrift sowie alle übergeordneten Überschriften in den Chunk geschrieben. Bspw. wird

```
# Überschrift
Einleitung
## Unterüberschrift
Text
```

umgewandelt in

```
Dokument 1:
# Überschrift
```

### Einleitung

Dokument 2:

# Überschrift

## Unterüberschrift

Text

Durch das Übernehmen der übergeordnete Überschriften bleibt ein Teil des Kontexts des restlichen Dokuments in jedem Chunk erhalten.

Die Unterteilung der Übungsblätter basiert auf dem PyPDFLoader. Dieser ist zuständig für das Laden der Inhalte und erstellt wie gewohnt Dokumente aus den einzelnen Seiten. Daraufhin wird in jedes Dokument die Überschrift der Aufgabe, zu der es gehört, geschrieben. Dabei wird ausgenutzt, dass diese Überschrift stets einem einheitlichen Schema entsprechen, welches über einen regulären Ausdruck erkannt werden kann. Durch das Hinzufügen der Überschrift erhalten die Seiten zusätzlichen Kontext, was die Ähnlichkeitssuche verbessern soll.

Zum Testen der Methode werden die Vorlesungswebsite und die sechs Übungsblätter mit Unterteilung nach Überschriften eingebettet. Es werden dieselben zehn Fragen wie in Unterunterabschnitt 5.4.4.1 jeweils zehn Mal gestellt. Auch in diesem Fall werden maximal fünf Dokumente abgerufen.

Bei den ersten fünf Fragen wird in jedem Fall das richtige Dokument zurückgegeben. Anders als in Unterunterabschnitt 5.4.4.1 handelt es sich dabei nicht um die gesamte Vorlesungswebsite, sondern um einen bestimmten Abschnitt auf der Website, mit dem die Frage beantwortet werden kann. Die Antworten des Sprachmodells sind somit korrekt. Von den fünf Fragen zu den Übungsblättern werden nun bei den ersten vier alle notwendigen Dokumente abgerufen. Das Sprachmodell antwortet auch hier korrekt. Bei der letzten Frage werden jedoch die falschen Dokumente abgerufen. Das Sprachmodell gibt korrekt aus, dass es die notwendigen Informationen nicht finden kann.

Die bisherigen Versuche beruhen auf wenigen eingebetteten Dokumenten. Ab jetzt werden neben der Vorlesungswebsite und den Übungsblätter auch die PDFs der Vorlesungsfolien und die Websites des Programmieren-Wikis eingebettet. Bei den Folien handelt es sich ohnehin um kurze Seiten mit wenig Kontext. Daher wird für sie der PyPDFLoader verwendet. Das Programmieren-Wiki wird nach Überschriften von Grad eins, zwei und drei unterteilt. Die Vorlesungswebsite wird von nun an nicht mehr als HTML sondern als Media-Wiki eingebettet. Dies ist das Markup, in dem ihr Inhalt verfasst wird. Es ist öffentlich einsehbar. Das Verarbeiten und Einbetten des Media-Wikis hat zwei Vorteile. Zum einen sind die Ergebnisse mit Media-Wiki etwas besser ausgefallen. Zum anderen lassen sich die im Web-Frontend angegebenen Texte besser lesen, wenn diese in Media-Wiki dargestellt werden. Die Seiten des Programmieren-Wikis sind ebenfalls mit Media-Wiki geschrieben. Diese basieren jedoch stark auf Schablonen, die dynamisch eingebunden werden und es erschweren, manche Informationen auszulesen. Um keinen Inhalt zu verlieren, wird das aus dem Media-Wiki resultierende HTML für die Verarbeitung und Einbettung verwendet.

Um den RAG-Service mit der hohen Anzahl an Dokumenten zu testen werden wieder die bekannten zehn Fragen gestellt. Dabei werden wie im vorherigen Versuch die fünf Fragen zur Vorlesungswebsite und die ersten vier Fragen zu den Übungsblätter korrekt beantwortet. Frage sechs stellt dabei eine Ausnahme dar. Sie wird nicht mehr richtig beantwortet. Das könnte daher kommen, dass signifikante Begriffe der Anfrage wie Aufgabe, Studierende, Hilfsmethoden und Strings im neuen Kontext enthalten sind und hier eine höhere Ähnlichkeit besteht. Bei der Auswertung der Korrektheit der Antworten ist zu beachten, dass die neuen Inhalte weiteren Kontext zur Beantwortung mancher Fragen liefern, sodass sich die abgerufenen Dokumente sowie die Antworten des Sprachmodells in diesen Fällen korrekterweise ändern.

Um den Abruf der neuen Inhalte sowie die Leistungen der Retriever zu testen, kommen folgende Fragen hinzu:

11. Ein Student hat alles in die main Methode geschrieben, ab wann gibt es dafür Abzug?
12. Was bedeutet Exited Prematurely?
13. Werden LinkedLists in der Vorlesung behandelt?
14. Code aus Abschnitt A.2. Das mit dem type als String ist nicht gut. Ab welchem Blatt gibt es dafür Abzug?
15. Der Student hat setAge neben der Beschreibung des Parameters nur mit „Sets age of the person“ im JavaDoc. Ab welchem Blatt muss ich dafür Punkte abziehen? Sollte ich überhaupt dafür Punkte abziehen, oder findest du, das reicht?

Die ersten drei Fragen richten sich an die neuen Inhalte. Bei Frage vier und fünf handelt es sich um komplexere Fragen. Insbesondere bei Frage vier muss die Problematik des Codes verstanden und für die Ähnlichkeitssuche verwendet werden und nicht der Inhalt.

Von den neuen Fragen werden für die ersten drei Fragen die richtigen Dokumente abgerufen und richtige Antworten generiert. Dies entspricht den Erwartungen, da analoge Fragen bereits erfolgreich an andere Inhalte gestellt worden sind. Frage vier wird in keinem Fall richtig beantwortet. Das kommt daher, dass die Einbettung den Inhalt einbettet und nicht die Problematik. Frage fünf wird in vier von zehn Fällen richtig beantwortet. Zum einen ist die Fragestellung komplexer als bei den ersten drei Fragen. Zum anderen gibt es zur Beantwortung dieser Frage mehrere sehr ähnliche Dokumente (JavaDoc, JavaDoc trivial, Dokumentation, Kommentare), die sich teilweise widersprechen.

**Schlussfolgerung** Das Unterteilen nach Überschriften hat bei einigen Fragen zu einer signifikanten Verbesserung der Ergebnisse beigetragen. Dies ist auf die erhöhte Ähnlichkeit der einzelnen Chunks mit den Vectorstore-Anfragen zurückzuführen. Dennoch konnten bisher nicht alle Fragen beantwortet werden, sodass eine weitere Optimierung sinnvoll ist.

### 5.4.4.3. SelfQueryRetriever und GeneratedQueriesRetriever

Um den Abruf von Dokumenten weiter zu verbessern, wird die Verwendung des SelfQueryRetriever von LangChain im RAG-Service untersucht. Anders als der bisherige Retriever stellt dieser die Anfragen des Benutzers nicht an den Vectorstore. Stattdessen fordert er ein Sprachmodell auf, eine Anfrage zu aus der Benutzeranfrage zu generieren. Zudem unterstützt er die Generierung von Filtern, was jedoch einen hohen manuellen Aufwand bei der Einbettung der Inhalte erfordert. Daher wird diese Funktion hier nicht verwendet.

Um den SelfQueryRetriever zu testen, werden die bisherigen 15 Fragen gestellt. Bei den ersten zehn Fragen werden für die ersten neun die richtigen Dokumente abgerufen, mit Ausnahme von Frage zwei zur Präsenzübung. Hier werden zwar Dokumente zur Präsenzübung abgerufen, jedoch nicht solche die ihr Datum und ihre Uhrzeit spezifizieren. Dies kommt daher, dass die vom Sprachmodell generierte Anfrage an den Vectorstore zu sehr von der ursprünglichen Anfrage abstrahiert. Frage zehn wird wie bisher und aus den bisherigen Gründen falsch beantwortet. Bei Frage elf wird in jedem Fall das notwendige Dokument abgerufen, um die Frage zu beantworten. Jedoch wird nur in manchen Fällen die richtige Antwort gegeben. Dies ist nicht ganz nachvollziehbar, da das Sprachmodell im Falle des BaseRetrievers ähnliche Kontexte erhalten hat und jedes Mal richtig geantwortet hat. Frage 12, 13 und 15 werden in jedem Fall richtig beantwortet. Bei Frage 15 ist das Umwandeln der komplexen Anfrage in eine einfache kurze Anfrage ausschlaggebend für die Verbesserung der Ergebnisse gegenüber dem BaseRetriever. Frage 14 wird in keinem Fall richtig beantwortet. Der SelfQueryRetriever generiert keine Anfrage basierend auf der Problematik sondern basierend auf dem Inhalt.

Der SelfQueryRetriever generiert nur eine Anfrage aus der Benutzeranfrage. Diese ist häufig sehr kurz und verliert dadurch zu viel vom Kontext der originalen Anfrage. Dies sind entscheidende Nachteile. Es wird daher der GeneratedQueriesRetriever entwickelt, der auf der Query-Step-Down-Methode aus Absatz 3.2 basiert. Ein großer Vorteil der eigenen Implementierung ist, dass die Anfrage an das Sprachmodell frei implementiert werden kann. Statt nur die Benutzeranfrage zu senden, wird beim GeneratedQueriesRetriever die gesamte bisherige Konversation gesendet. Das Modell wird aufgefordert, wenige, kurze Anfragen an einen Vectorstore zu generieren, die helfen sollen, die letzte Nachricht des Benutzers zu beantworten. Die Anfragen sollen dennoch alle Themenbereiche der letzten Nachricht abdecken. Die bisherige Konversation soll dazu verwendet werden, um den Kontext zu verstehen. Nachdem die Anfragen vom Sprachmodell generiert worden sind, werden sie nacheinander mithilfe des BaseRetriever an den Vectorstore gestellt. Für jede Anfrage werden Dokumente abgerufen. Anschließend werden Duplikate in den Dokumenten entfernt.

Der GeneratedQueriesRetriever beantwortet alle Fragen außer Frage zehn und elf in jedem Fall korrekt. Insbesondere generiert er für Frage 14 Anfragen, die die Problematik beschreiben. Dies kommt daher, dass er aufgefordert wird, die Thematik der Anfrage und nicht ihren Inhalt mit den ausgegebenen Vectorstore-Anfragen zu beschreiben. Für Frage zehn stellt auch er jedoch keine Optimierung dar. Für Frage elf werden Anfragen generiert, die auch zu anderen Dokumenten eine hohe Ähnlichkeit haben. In manchen Fällen werden

dabei die richtigen Dokumente abgerufen. Das Sprachmodell generiert in diesem Fall jedoch nur manchmal eine richtige Antwort. Das ist insofern nachvollziehbar, dass sehr viele Kontexte der Anfrage an das Sprachmodell angehängt werden und damit die Information möglicherweise übersehen wird.

**Schlussfolgerung** Der `GeneratedQueriesRetriever` ist ein großer Fortschritt für die Arbeit. Er ist in der Lage, Anfragen basierend auf dem Chat-Verlauf zu generieren. Zudem kann er die eigentlichen Informationsbedürfnisse aus komplexen Anfragen extrahieren. Dabei kann es jedoch passieren, dass von ihm generierte Anfragen zu stark von der originalen Anfrage abweichen und viele irrelevante Dokumente extrahiert werden.

#### 5.4.4.4. `WithReferencesRetriever`

Im Folgenden wird der `WithReferencesRetriever` entwickelt. Er adressiert zwei Aspekte, die die bisherigen Retriever nicht behandelt haben. Zum einen ist die Frage nach Übungsblatt  $x$  Aufgabe  $y$  noch nicht im Allgemeinen beantwortet worden. Zum anderen wird bisher nur wenig ausgenutzt, dass die Überschriften in vielen Dokumenten bereits einen großen Teil des Themas und Inhalts des Abschnitts.

Die Idee des `WithReferencesRetriever` beruht auf dem `Parent-Document-Retriever` aus Absatz 3.2. Der Retriever unterscheidet zwischen bisherigen Dokumenten und solchen, die bisherige Dokumente referenzieren. Letztere werden im Folgenden als Meta-Dokumente bezeichnet. Diese Meta-Dokumente werden beim Einbetten der Inhalte erzeugt. Im RAG-Service wird für jedes Dokument, das eingebettet werden soll, ein weiteres Dokument erzeugt. Dieses enthält ausschließlich die Überschriften des einzubettenden Dokuments. Dieses Meta-Dokument wird ebenfalls eingebettet. Das Abrufen der Dokumente geschieht über den `BaseRetriever`. Wird ein Meta-Dokument abgerufen, wird automatisch das originale Dokument statt dem Meta-Dokument zurückgegeben. Duplikate werden entfernt. Die Idee ist, dass eine Benutzeranfrage sehr ähnlich zu Überschriften sein könnte, wenn diese genau zu der Thematik der Anfrage passen. Der größere Kontext des originalen Dokuments könnte diese Ähnlichkeit wiederum beeinträchtigen. Um flexibler eingesetzt werden zu können, nimmt der `WithReferencesRetriever` statt einer einzelnen Anfrage eine Liste von Anfragen entgegen und ruft die jeweiligen Dokumente ab, wobei entfernt Duplikate.

Der `WithReferencesRetriever` liefert ähnliche Ergebnisse wie der `BaseRetriever`. Die Unterschiede liegen in der Beantwortung von Frage 11 und Frage 15. Frage 11 wird vom `BaseRetriever` richtig beantwortet, vom `WithReferencesRetriever` jedoch nicht. Da der `WithReferencesRetriever` das richtige Dokument abrufen und auch die anderen abgerufenen Dokumente ähnlich sind, ist dies eher auf die nicht-deterministischen Antworten des Sprachmodells zurückzuführen. Bei Frage 15 antwortet der `WithReferencesRetriever` jedoch zuverlässiger als der `BaseRetriever`. Dies könnte darauf zurückgeführt werden, dass der `WithReferencesRetriever` in diesem wie auch in anderen Fällen dazu neigt, mehr Dokumente zurückzugeben, die dem Thema entsprechen. Dadurch erhält das Sprachmodell weniger irrelevanten Kontext, was die Antwort erleichtern könnte. Eine Vermutung ist gewesen,

dass das Einführen der Meta-Dokumente mit dem WithReferencesRetriever dazu führen könnte, dass Frage zehn richtig beantwortet wird. Das hat sich nicht bewahrheitet.

**Schlussfolgerung** Der WithReferencesRetriever fügt keine signifikante Steigerung der Leistung hinzu. Die Dokumente, die von ihm abgerufen werden, sind jedoch häufig thematisch weniger gestreut als beim BaseRetriever. Dies kann bei der Generierung von Antworten ein Vorteil sein. Zudem könnten in zukünftigen Arbeiten weitere Strategien zur Erzeugung von Meta-Dokumenten implementiert werden. Diese könnten möglicherweise einen größeren Einfluss haben als die bisherige Strategie. Aus diesen Gründen wird der WithReferencesRetriever in den nächsten Schritten der Optimierung weiterhin betrachtet.

### 5.4.4.5. CombinedRetriever

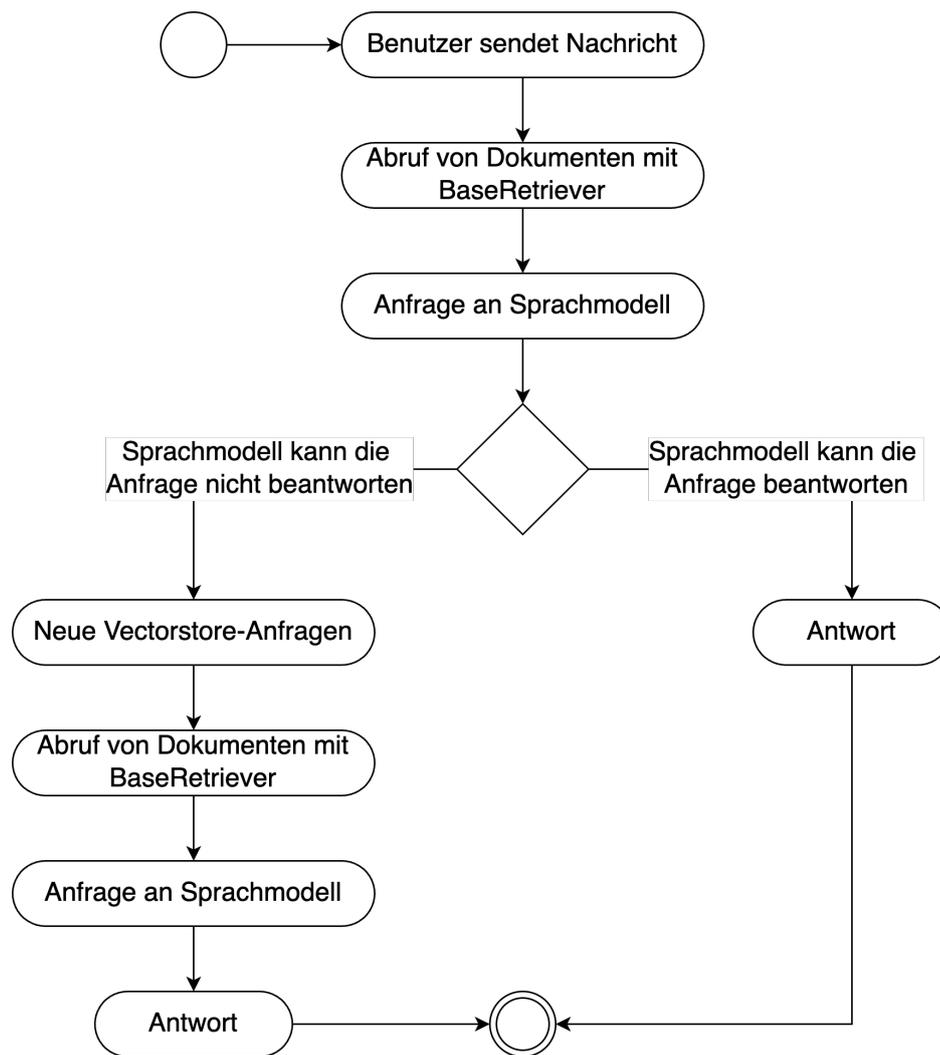
Die unterschiedlichen Herangehensweisen und Stärken des GeneratedQueriesRetrievers und WithReferencesRetrievers legen nahe, dass eine Kombination beider Retriever den Abruf verbessern kann. Dies wird im Folgenden untersucht. Dazu wird der CombinedRetriever entwickelt. Er generiert zunächst auf dieselbe Weise wie der GeneratedQueriesRetriever mehrere Anfragen und verwendet den WithReferencesRetriever, um die Dokumente abzurufen (der GeneratedQueriesRetriever verwendet den BaseRetriever, um die Dokumente abzurufen).

Der CombinedRetriever wird mit den bisherigen 15 Fragen evaluiert. Die Ergebnisse unterscheiden sich nicht signifikant von denen des GeneratedQueriesRetrievers. Das kommt daher, dass der WithReferencesRetriever und der BaseRetriever ähnliche Ergebnisse liefern.

### 5.4.4.6. MultiStepsRetriever

Die bisherigen Ergebnisse zeigen, dass die verschiedenen Methoden den Abruf in manchen Fällen verbessern, in manchen aber auch verschlechtern. Zudem wird durch die zusätzliche Anfrage an ein Sprachmodell, die vermehrten Anfragen an den Vectorstore und die Suche der Referenzen zu den originalen Dokumenten die Antwortzeit des Assistenten spürbar verlängert. Für einige Anfragen liefert der BaseRetriever bereits gute Ergebnisse. In der folgenden, letzten Technik zur Optimierung der Anfragen werden die Vorteile sowohl von dem einfachen als auch den komplexen Retrievern kombiniert. Dazu wird der MultiStepsRetriever entwickelt, welcher auf Active RAG (siehe Absatz 3.2) basiert. Seine Funktionsweise ist in Abbildung 5.5 dargestellt.

Zunächst werden mit der Benutzeranfrage Dokumente aus dem Vectorstore abgerufen. Diese werden zusammen mit der Konversation an das Sprachmodell geschickt. Das Sprachmodell wird aufgefordert, eine Antwort zu generieren, wenn es die dafür notwendigen Informationen besitzt. Diese wird dann direkt an den Benutzer ausgegeben und der Prozess terminiert. Andernfalls soll es wie beim GeneratedQueriesRetriever neue Vectorstore-Anfragen ausgeben. Diese werden für den Abruf neuer Dokumente verwendet. Anschließend folgt eine



**Abbildung 5.5.:** Aktivitätsdiagramm vom MultiStepsRetriever

erneute Anfrage an das Sprachmodell mit den neuen Dokumenten. Der Prozess ist so implementiert, dass dies iterativ fortgesetzt werden kann. Dabei muss für jede Iteration ein Prompt-Template angegeben werden. So kann dem Sprachmodell mitgeteilt werden, wie es in der jeweiligen Iteration antworten soll, um nicht immer dasselbe zu tun. Im RAG-Service ist die Anzahl der Iterationen auf zwei beschränkt. Anders als im ersten Durchlauf wird das Sprachmodell im zweiten Durchlauf nicht mehr aufgefordert, neue Anfragen auszugeben. Stattdessen soll es in jedem Fall eine Antwort geben, wenn es die Antwort nicht weiß, soll es dies mitteilen. Die Beschränkung auf zwei Iterationen soll die Antwortzeit des Assistenten verkürzen.

Der MultiStepsRetriever wird mit den bisherigen 15 Fragen evaluiert. Seine Ergebnisse sind ähnlich zu denen des GeneratedQueriesRetrievers und CombinedRetrievers. Der einzige signifikante Unterschied liegt in der Beantwortung von Frage 14. Diese wird vom MultiStepsRetriever stets falsch beantwortet. Eine weitere Schwäche ist, dass die Ergebnisse des BaseRetrievers häufig nicht genutzt werden, obwohl sie bereits ausreichen würden, um die

Anfrage sicher zu beantworten. Diese Probleme lassen sich auf das Prompt-Engineering zurückführen. Damit der MultiStepsRetriever funktioniert, ist das Prompt-Template angepasst worden. Dies führt dazu, dass andere Anfragen ausgegeben werden als beim Generated-QueriesRetriever. Zudem kann über das Prompt-Template gesteuert werden, wann neue Anfragen ausgegeben werden sollen und wann eine Antwort basierend auf den Kontexten des BaseRetrievers ausgegeben werden soll. Aufgrund der begrenzten Zeit der Arbeit werden diese Optimierungen nicht mehr untersucht.

**Schlussfolgerung** Der MultiStepsRetriever kombiniert die bisherigen Retriever in einer Weise, dass die Stärken jeder dieser Retriever in den MultiStepsRetriever mit einfließen. Aus technischen Gründen sind Anpassungen vorgenommen worden, die die Ergebnisse leicht verschlechtern. Zudem fehlt eine Form der Kalibrierung, welche optimiert, wann welcher Retriever verwendet werden soll.

### 5.4.5. Prompt-Engineering

Der Prototyp hat gezeigt, dass die Antworten vom Sprachmodell bereits ohne Prompt-Engineering sehr gut sind, während der Abruf der Dokumente noch stark verbessert werden konnte. Daher liegt der Fokus dieser Arbeit mehr auf der Optimierung der Anfrage als auf dem Prompt-Engineering. Es werden dennoch einige Techniken untersucht und eingesetzt.

Eine Herausforderung, der in dieser Arbeit aufgrund der zeitlichen Begrenzung nicht weiter nachgegangen wird, ist, dass die Instruktionen aus den Prompt-Templates häufig nicht bei der Generierung der Antworten beachtet werden. Dies wird im Folgenden für die jeweiligen Methoden, bei denen das Problem auftritt, näher erläutert.

**Grundlegende Informationen** Die Prompt-Templates im RAG-Service beinhalten diverse Informationen, die die generierten Antworten verbessern sollen. Dazu gehören insbesondere:

1. Das aktuelle Datum
2. Die Tätigkeit der Benutzer
3. Wie Code geschrieben werden soll

Außerdem gibt es Anweisungen, wie die Ausgabe erfolgen soll. Dies bezieht sich zum einen auf optische Aspekte wie das Markieren wichtiger Informationen mit Markdown. Für den MultiStepsRetriever sind auch technische Ausgaben notwendig, um zwischen der Ausgabe einer Antwort oder neuen Anfragen unterscheiden zu können.

Häufig werden Informationen jedoch nicht beachtet. Dies wird besonders in der Evaluation deutlich. Insbesondere wird dort festgestellt, dass das Sprachmodell sich der Tutor-Rolle des Benutzers nicht bewusst ist. Zudem verwendet es die technisch notwendigen Ausgaben gelegentlich falsch.

**Few-Shot-Prompting** Diese Technik wird bei der Erstellung des Kalenders eingesetzt. Dieser soll in einem fest vorgegebenen JSON-Format ausgegeben werden, weswegen sich das Angeben von Beispielen besonders eignet. Anders als vorher sind seit dem Mitliefern der Beispiele keine Fehler im Format der Ausgabe aufgetreten.

**Chain-of-Thought** Die Ausgabe des Prototypen verändert sich stark, wenn das Sprachmodell angewiesen wird, sein Ergebnis vor der Ausgabe zu erklären. Selbst einfache Fragen, bspw. nach der Anzahl der Übungsblätter, resultieren in detailreiche Erklärungen. In manchen Fällen beinhalten diese Erklärungen weitere relevante Informationen, die im finalen Ergebnis nicht enthalten sind. Auch im Assistenten wird das Modell aufgefordert, seine Antwort genau zu erklären. Bei vielen Anfragen wird dies jedoch nicht beachtet. Dieselben Fragen, auf die der Prototyp ausführliche Antworten liefert, werden vom Assistenten meist knapp beantwortet. Die Antworten sind dennoch in den meisten Fällen korrekt.

**Emotion-Prompting** Das Emotion-Prompting ist im Prototypen getestet worden. Zu Beginn der Arbeit sind Fälle produziert worden, in denen das Modell unter Anwendung des Emotion-Promptings ausgegeben hat, die Antwort nicht zu kennen, obwohl es ohne Emotion-Prompting stets korrekt geantwortet hat. Dies konnte zuletzt nicht reproduziert werden. Im Assistenten wird dem Sprachmodell mit schlechterer Benutzererfahrung gedroht, wenn es bestimmte Instruktionen nicht befolgt. Dies scheint jedoch keinen Einfluss auf die Antworten des Sprachmodells zu haben.

**Sampling** In Unterabschnitt 5.4.4 wird gezeigt, dass sich in vereinzelt Fällen die Antworten des Assistenten bei gleicher Anfrage unterscheiden können. Es ist naheliegend, Sampling einzusetzen, um die beste Antwort aus einer Vielzahl generierter Antworten zu wählen. Ein großer Nachteil dieser Methode ist, dass die Generierung der finalen Antworten unter Verwendung von Sampling deutlich länger dauert. Das Generieren der finalen Antwort des Assistenten dauert zum Teil ohnehin schon recht lange. Zudem sind die meisten falschen Antworten eher auf Fehler beim Abruf der Dokumente zurückzuführen. Daher wird in dieser Arbeit kein Sampling eingesetzt und stattdessen der Fokus auf den Abruf der Dokumente gelegt.

#### 5.4.6. Vergleich von Einbettungsmodellen

In der bisherigen Arbeit ist das Einbettungsmodell text-embedding-3-large von OpenAI verwendet worden. Dieses wird von OpenAI-Servern zur Verfügung gestellt. Insbesondere aus Datenschutzgründen wäre eine selbst-gehostete Alternative sehr vorteilhaft. Im Folgenden wird text-embedding-3-large mit den Modellen nomic-embed-text:v1.5, llama3.1:8b und llama3.1:70b verglichen. Diese können über Ollama von einem Server des KITs bereitgestellt werden. Das Modell nomic-embed-text:v1.5 ist das verbreitetste Einbettungsmodell, welches über Ollama verwendet werden kann. Bei den LLaMA-Modellen handelt es sich um Sprachmodelle. Da Sprachmodelle ebenfalls Einbettungen erzeugen, können sie auch als Einbettungsmodelle fungieren. Sie werden als mögliche Alternative ebenfalls in den Vergleich mit aufgenommen, da nomic-embed-text:v1.5 deutlich schlechtere Leistungen im

Vergleich zu text-embedding-3-large zeigt. Die Leistungen der LLaMA-Modelle entsprechen jedoch eher denen von nomic-embed-text:v1.5.

In dem Vergleich werden die Vorlesungswebsite und alle sechs Übungsblätter jeweils für jedes Einbettungsmodell in einem separaten Vectorstore eingebettet. Dazu werden der WebBaseLoader und der PyPDFLoader verwendet. Anschließend werden Ähnlichkeitssuchen auf den Vectorstores mit folgenden Anfragen durchgeführt:

1. Zu welchen Zeiten findet die Vorlesung statt?
2. Worum geht es in Aufgabe A auf Übungsblatt 2?
3. Welche Bibliotheken sind auf Übungsblatt 4 erlaubt?
4. Wo gibt es Informationen zur Fachschaft?
5. Welche Inhalt und Lernziele gibt es in der Vorlesung?

Bei jeder der fünf Anfragen pro Vectorstore wird die Anzahl der abzurufenden Dokumente auf vier beschränkt. Dies entspricht dem Standard bei LangChain. Jede der Anfragen kann mit genau einem Dokument im Vectorstore beantwortet werden. Es wird daher für jede Anfrage untersucht, ob sich das notwendige Dokument unter den vier abgerufenen Dokumenten befindet. Zudem wird sein Rang bezüglich der Ähnlichkeit unter den vier Dokumenten bewertet. Da die Antworten von Vectorstores nicht unbedingt deterministisch sind, wird jede Frage jedem Modell jeweils zehn Mal gestellt.

Das Modell text-embedding-3-large gibt für die erste, dritte und fünfte Frage in jedem Fall das richtige Dokument aus. Dieses ist in jedem Fall auch das ähnlichste Dokument zur Anfrage von den vier ausgegebenen Dokumenten. Die falsche Aussage zu Dokument zwei stimmt mit den Untersuchungen bezüglich der Abruf-Strategien überein (siehe Unterunterabschnitt 5.4.4.1). Alle anderen Modelle beantworten jede Frage in jedem Fall falsch.

**Schlussfolgerung** Die Einbettungsmodelle, die mit text-embedding-3-large verglichen worden sind, eignen sich nicht als vergleichbare Alternativen. Anbieter wie Hugging Face<sup>6</sup> stellen noch weitere Modelle zur Verfügung, die in dieser Arbeit jedoch nicht untersucht werden.

### 5.4.7. Vergleich von Sprachmodellen

In der bisherigen Arbeit ist das Sprachmodell GPT-4o von OpenAI verwendet worden. Wie auch bei den Einbettungsmodellen ist eine selbst-gehostete Alternative aus Datenschutzgründen vorteilhaft. Im Folgenden werden GPT-4o und das kleinere GPT-4o mini mit den von einem Server des KIT über Ollama bereitgestellten Modellen LLaMA 3.1:8b und LLaMA 3.1:70b verglichen. Dazu wird die Vorlesungswebsite über den WebBaseLoader eingebettet. Anschließend werden jedem Modell die folgenden Fragen jeweils zehn Mal gestellt:

---

<sup>6</sup><https://huggingface.co>

1. Wie viele Übungsblätter gibt es?
2. Wann findet die Präsenzübung statt?
3. Wann findet die Vorlesung statt?

Der Test bezieht sich nicht auf die Optimierung des RAG-Prozesses, sondern soll die Leistungen der verglichenen Sprachmodelle darstellen. Untersucht wird dabei vor allem die Korrektheit der Antworten. Zudem werden die Zeiten der Antwort-Generierung gemessen und jeweils der Median berechnet. Die Ergebnisse sind in Tabelle 5.1 dargestellt.

	gpt-4o-mini	gpt-4o	llama3.1:8b	llama3.1:70b
1. Frage: Korrekt	Alle, vollst.	Alle, vollst.	Keine	Keine
1. Frage: Zeit	1,2s	1,0s	1,4s	1,6s
2. Frage: Korrekt	Alle, vollst.	Alle, vollst.	Alle, vollst.	Alle, vollst.
2. Frage: Zeit	1,6s	1,4s	1,2s	20,6s
3. Frage: Korrekt	Alle, unvollst.	Alle, unvollst.	Keine	Keine
3. Frage: Zeit	2,3s	2,1s	1,3s	24,1s

**Tabelle 5.1.:** Ergebnisse des Prototypen.

Es sind immer jede der zehn Antworten pro Frage korrekt und vollständig, oder korrekt aber unvollständig oder inkorrekt gewesen.

Die gemessenen Zeiten geben keine Auskunft über die Güte des Modells, da insbesondere die Geschwindigkeiten der vom KIT-Server bereitgestellten Modelle stark von der Leistung des Servers abhängt. Sie ist jedoch ein Maß dafür, inwiefern das Modell in diesem konkreten Kontext in einem Produktiv-System praktikabel ist. Es ist erkennbar, dass die Benutzer bei der Verwendung von LLaMA 3.1:70b bei den meisten Fragen besonders lange auf eine Antwort warten müssten.

Interessanter als die gemessenen Zeiten sind jedoch die Ergebnisse bezüglich der Korrektheit der Antworten. Dabei schneiden die GPT-Modelle in jedem Fall besser ab als die LLaMA-Modelle.

Die erste Frage bezieht sich auf die Anzahl der Übungsblätter. Diese sind auf der Website aufgelistet als Übungsblatt 0 bis Übungsblatt 5. Während die GPT-Modelle die richtige Antwort „sechs“ ausgeben, geben die LLaMA Modell „fünf“ aus. Letztere Ergebnisse sind aufgrund der Benennung der Übungsblätter nachvollziehbar, jedoch falsch. Es ist jedoch möglich, dass dieser Fehler in anderen Kontexten weniger schwer zum Tragen kommt.

Die Antwort nach der Präsenzübung steht unmittelbar auf der Website und wird von den Modellen (nahezu) original wiedergegeben. Somit sind alle Antworten korrekt und vollständig.

Die Vorlesungszeiten stehen über drei Stichpunkte verteilt auf der Website. Es ist daher nachvollziehbar, dass die Antworten nicht vollständig sind. Dennoch stehen verschiedene Zeiten zu Vorlesungen unmittelbar auf der Website. Es ist daher weniger nachvollziehbar, weswegen die LLaMA-Modelle in jedem Fall falsche Antworten geben. LLaMA 3.1:8b gibt

an, keine Informationen zu besitzen. LLaMA 3.1:70b gibt falsche Zeiten an, die auch in keinem anderen Kontext auf der Website auftreten.

**Schlussfolgerung** Es gibt signifikante Qualitätsunterschiede zwischen den getesteten LLaMA-Modellen und den GPT-Modellen. Diese Unterschiede haben sich auch in anderen Tests bestätigt. Aus diesem Grund fokussiert sich die Arbeit auf eine Implementierung des Assistenten mit OpenAIs GPT-Modellen. Es ist dennoch darauf geachtet worden, die Modell-Anbindung austauschbar zu gestalten. So können verschiedene Modelle zu einem späteren Zeitpunkt nochmal erörtert werden. Selbst-gehostete Modelle sind insbesondere wegen der Datenschutz-Vorteile weiterhin von Interesse.

### 5.4.8. Kalender

Die bisherige Optimierung setzt sich mit RAG-Methoden zur Optimierung der Chat-Funktion auseinander. Diese sind evaluiert und für diesen Anwendungsfall optimiert worden. Neben dieser Chat-Funktion soll der RAG-Service aus den eingebetteten Dokumenten einen Kalender generieren können (siehe Tabelle 4.1). So sollen die Benutzer direkt beim Starten der Anwendung Termine und Fristen angezeigt bekommen.

**Verfahren** Für das Finden von Kalendereinträge werden grundsätzlich dieselben Dokumente herangezogen wie für das Beantworten von Fragen. Dies führt jedoch zu neuen technischen Herausforderungen. Während die Chat-Funktion dafür optimiert ist, die richtigen Antworten in wenigen abgerufenen Dokumenten zu finden, ist dies für den Kalender nicht möglich. Das kommt daher, dass relevante Ereignisse in beliebig vielen Dokumenten zu finden sein könnten. Des Weiteren enthalten viele Dokumente, bspw. die Übungsblätter, Termine und Fristen, die auch in anderen Dokumenten zu finden sind, wie der Vorlesungswebsite. Sie müssten in diesem Fall nicht abgerufen werden. Um dieses Problem zu lösen wird eine manuelle Konfiguration eingeführt. In der Main-Setting muss für jeden Inhalt (bspw. Vorlesungswebsite) oder jede Gruppe von Inhalten (bspw. Übungsblätter) konfiguriert werden, ob darin nach Kalender-Einträgen gesucht werden soll. Der Aufwand dafür ist nur sehr gering, hat jedoch einen großen Einfluss auf die Effizienz der Implementierung. Der Prozess basiert somit nicht mehr auf klassischem RAG mit Ähnlichkeitssuche. Stattdessen werden genau die Dokumente, die für die Suche nach Kalender-Einträgen konfiguriert worden sind, an das Sprachmodell gesendet. Dieses wird aufgefordert alle Ereignisse, die in einem Kalender darstellbar sind, zu extrahieren und auszugeben.

**Antwort des Sprachmodells** Zuerst wird das Sprachmodell aufgefordert, aus den gefundenen Ereignissen eine Tabelle mithilfe von Markdown zu erstellen. Dadurch ist nur wenige Code im Web-Frontend notwendig, um die Antwort des RAG-Service darzustellen. Zudem kann das Sprachmodell das Design an den Inhalt dynamisch anpassen. Dieses Format eignet sich jedoch nicht, um die Daten weiterzuverarbeiten. Dies ist bspw. dann notwendig, wenn das Sprachmodell Fehler beim Sortieren der Ereignisse macht. Daher wird das Sprachmodell im nächsten Schritt aufgefordert, seine Ausgabe als JSON zu generieren. Dabei soll es sich um ein Array von Objekten handeln, die jeweils ein Ereignis darstellen. Die Attribute dieser

Objekte sind Titel, Datum und Uhrzeit des jeweiligen Ereignisses. Dieses Format lässt sich leicht weiterverarbeiten und damit auch in darstellbare Objekte umwandeln. Zudem liefert es bessere Ergebnisse als die Antworten mit Markdown. Diese sind in Tabelle 5.2 dargestellt. Die jeweiligen Konfigurationen aus Modell und Format sind jeweils zehn Mal getestet worden. Die dargestellten Werte sind die Mediane aus den Versuchen

Modell	Format	Anzahl	Fehlerhaft
GPT-4o mini	Markdown	19	10
GPT-4o	Markdown	30	2
GPT4-o mini	JSON	24	9
GPT-4o	JSON	32	0

**Tabelle 5.2.:** Auswertung

Anzahl: Anzahl der Ereignisse, die gefunden worden sind. Fehlerhaft: Anzahl der Fehler, die in den Ereignissen vorhanden gewesen sind.

Beide Modelle generieren bessere Antworten im JSON-Format als im Markdown-Format. Der Kalender wird daher mit JSON umgesetzt.

Zudem ist erkennbar, dass GPT-4o über 30% mehr Ereignisse findet als GPT-4o mini. Zudem treten bei der Generierung von Kalendern mit GPT-4o deutlich weniger Fehler auf, als mit GPT-4o mini. Anders als bei den Antworten auf die einfachen Anfragen in Unterabschnitt 5.4.7 übertrifft GPT-4o das kleinere GPT-4o mini hier signifikant.



## 6. Evaluation des Assistenten

In diesem Kapitel wird der implementierte Assistent evaluiert. Dabei sollen folgende Fragen beantwortet werden:

1. Wie zuverlässig sind die Antworten des Assistenten?
2. Wie hilfreich ist der Assistenten?
3. Wie sind die Tutoren gegenüber dem Assistenten eingestellt?

Zu diesem Zweck ist der Assistent den Tutoren des Wintersemesters 2024/2025 zur Verfügung gestellt worden. Sie sind gebeten worden, ihn bei ihrer Arbeit einzusetzen und anschließend eine Umfrage auszufüllen. Bei der folgenden Evaluation werden zum einen die Chat-Verläufe der Tutoren mit dem Assistenten untersucht und zum anderen die Umfrage ausgewertet.

### 6.1. Auswertung der Chat-Verläufe

Insgesamt sind 21 Chats<sup>1</sup> von Tutoren erstellt worden. Davon werden drei Chats nicht betrachtet, da sie keinen Bezug zur Programmierenvorlesung darstellen. Die 18 betrachteten Chats werden nach folgenden Kriterien untersucht:

1. Werden die Anfragen der Tutoren vom Assistenten verstanden?
2. Findet der Assistent die richtigen Quellen?
3. Sind die Antworten des Assistenten korrekt?
4. Sind die Antworten des Assistenten hilfreich im Bezug auf die Arbeit der Tutoren?

Die Feedback-Funktion für Nachrichten ist nur in wenigen Fällen verwendet worden, sodass es nur wenig Rückmeldung von den Tutoren zu den Nachrichten des Assistenten gibt. Daher werden die oben genannten Kriterien in den Chat-Verläufen hauptsächlich von mir eingeschätzt. Es ist möglich, dass meine Einschätzung leicht von der des jeweiligen Tutors abweicht. Die Verläufe zeigen insgesamt jedoch eindeutig die Stärken und Schwächen des Assistenten, weshalb diese Abweichungen zu keinen signifikante Änderungen der Ergebnisse dieser Evaluation führen würden.

---

<sup>1</sup>Die erstellten Chats können von Benutzern mit entsprechender Berechtigung eingesehen werden

Der Assistent versteht grundsätzlich die Anfragen des Tutors. Ein Problem ist jedoch, dass etwa bei der Hälfte der Verläufe der Rahmen der Vorlesung sowie der Zweck, Tutoren bei ihrer Arbeit zu unterstützen, nicht mit einbezogen wird. In manchen Verläufen ist es notwendig gewesen, zu konkretisieren, dass die vorherige Frage in Bezug auf die Vorlesung gestellt worden ist oder dass der Benutzer ein Tutor ist. Diese Informationen werden bei Chat-Nachrichten an das Sprachmodell mitgegeben. Daher ist nicht direkt nachvollziehbar, weshalb dies bei der Generierung der Antworten in manchen Fällen nicht berücksichtigt wird (siehe Unterabschnitt 5.4.5).

Die abgerufenen Dokumente beinhalten in den meisten Fällen alle notwendigen Dokumente zur Beantwortung der Frage. Es gibt jedoch drei Probleme beim Abruf von Dokumenten, die in bestimmten Fällen auftreten. Manche Anfragen benötigen besonders viele Dokumente, um beantwortet werden zu können, bspw. Zusammenfassungen großer PDFs. Die Anzahl der abgerufenen Dokumente ist dafür zu sehr in der Anzahl beschränkt. Zudem werden die abgerufenen Dokumente auch darüber gefiltert, wie ähnlich sie tatsächlich zur Anfrage sind. Dieses Limit ist für manche Fälle zu restriktiv eingestellt. Diese Restriktion ist vorgenommen worden, um nicht zu viele irrelevante Dokumente abzurufen. Dies ist damit jedoch nicht vollständig unterbunden worden. Nach Leng u. a. [21] beeinträchtigen lange Kontexte zwar nicht die Qualität der Antworten, jedoch würden weniger irrelevante Kontexte die Effizienz erhöhen und weitere relevante Kontexte die Antworten verbessern.

Wie die meisten Dokumente sind auch die Antworten meistens korrekt. Insgesamt können ungefähr 14 der 18 Chats als korrekt eingestuft werden. Zudem zeigen mehrere Verläufe, dass der Assistent in der Lage ist, eine Verknüpfung seines internen Wissens, bspw. zu Konzepten in Java, mit den Vorlesungsinhalten in seinen Antworten herzustellen. Die generierten Antworten beschreiben die Thematik dadurch differenziert, das Ergebnis entspricht dabei den Vorlesungsinhalten. Abbildung A.1 zeigt ein Beispiel für einen solchen Chat-Verlauf. Falsche Antworten werden in der Regel nur wegen mangelndem Kontext gegeben.

Auch wenn die meisten Antworten des Assistenten korrekt sind, fehlt bei einigen ein unaufgeforderter Bezug zur Vorlesung oder zur Arbeit der Tutoren. Bspw. werden Fragen zu Richtlinien und Konzepten häufig mit einer Beschreibung des jeweiligen beantwortet. Für Tutoren wäre besonders hilfreich zu wissen, was daraus für sie folgt. Das könnten Korrekturhinweise sein oder Angaben bis wann sie eine Richtlinie oder ein Konzept den Studierenden mitteilen sollten. Im aktuellen Assistenten müssen solche Informationen häufig explizit erfragt werden.

**Schlussfolgerung** Die Antworten des Assistenten sind meistens korrekt und zuverlässig. Dies kann zum einen auf die ausgezeichneten Leistungen des Sprachmodells zurückgeführt werden, Anfragen zu verstehen und Antworten zu generieren. Zum anderen ruft der Retriever meistens die richtigen Dokumente ab, wodurch das Sprachmodell den notwendigen Kontext erhält. Um dies weiter zu optimieren, wären Verbesserungen im Retriever sinnvoll, um möglichst alle relevanten und keine irrelevanten Dokumente zu finden. Durch seine zuverlässigen und korrekten Antworten kann der Assistent bereits in vielen Situationen hilfreiche Antworten geben. Verbesserungen könnten vor allem über Prompt-Engineering erreicht werden. Dem Sprachmodell kann darüber mitgeteilt werden, wie es Antworten

generieren soll. Zudem kann weiteres Hintergrundwissen über die Vorlesung und Tutoren angehängt werden, die dem Sprachmodell weiteren Kontext geben und es darüber informieren, worauf es den Fokus legen soll. Ein wesentlicher Aspekt dabei ist jedoch das Prompt-Engineering zu verbessern, sodass die Aufforderungen auch umgesetzt werden.

## 6.2. Auswertung der Umfrage

Die Umfrage basiert auf dem Technology Acceptance Model (TAM) [9]. Dabei handelt es sich um ein Modell, mit dem der Nutzen neuer Technologien sowie ihre Annahme bei den Benutzern untersucht werden können. Während sich der ebenfalls sehr verbreitete System Usability Scale (SUS) Questionnaire sehr auf die Interaktion des Benutzers mit dem System fokussiert, steht beim TAM neben der Bedienbarkeit auch der Nutzen sehr im Vordergrund. Daher wird dieser verwendet.

Die wesentlichen Gruppen, die sich in vielen Beispielen von TAMs wiederfinden, sind die vom Benutzer wahrgenommene Bedienbarkeit und der wahrgenommene Nutzen. Darüber hinaus untersucht er die Einstellung des Benutzers gegenüber dem System sowie seine Intention bezüglich der Nutzung. Zudem wird die sogenannte Selbstwirksamkeitserwartung gemessen und ob der Benutzer Schwierigkeiten beim Zugriff auf das System hat. Die Selbstwirksamkeitserwartung gibt das Vertrauen einer Person in die eigene Kompetenz, eine bestimmte Handlung ausführen zu können, an. Zusammen mit Problemen beim Zugriff auf das System kann sie die Wahrscheinlichkeit, dass der Benutzer das System benutzen wird, negativ beeinflussen [24]. Aufgrund der begrenzten, aus anderen Systemen jedoch bekannten Funktionen, die der Assistent bietet, ist davon auszugehen, dass die Tutoren die Fähigkeit haben werden, das System zu bedienen und keine Zweifel an diesen Fähigkeiten haben werden. Stattdessen könnten andere Faktoren eine ähnliche Wirkung haben. Statt Fragen zur Selbstwirksamkeitserwartung werden hier verallgemeinert Fragen zu Bedenken gestellt. Zudem werden die Einstellung und die Intention bei der Auswertung gemeinsam betrachtet, da sie sich sehr ähneln. Das im Folgenden berechnete hohe Cronbachsche Alpha der Gruppe Einstellung zeigt, dass dies die Gruppen Einstellung und Intention ähnliche Größen messen. Der resultierende Fragebogen ist in Tabelle 6.1 dargestellt.

Da die Umfrage mit aktuellen Tutoren durchgeführt wird, ist die Anzahl der potentiellen Teilnehmer klein. Insgesamt haben nur trotz mehrfacher Aufforderung nur sechs Tutoren an der Umfrage teilgenommen. Die Ergebnisse sind somit nicht zuverlässig. Es lassen sich dennoch Tendenzen in der Auswertung erkennen, die Mutmaßungen bezüglich dem Nutzen und der Einstellung der Tutoren ermöglichen.

Zunächst wird das Cronbachsche Alpha [7] für die Gruppen Bedienbarkeit, Nutzen, Einstellung und Bedenken berechnet. Dieses gibt die interne Konsistenz der Antworten innerhalb der jeweiligen Gruppe an. Darüber kann entschieden werden, ob die Antworten der Fragen der jeweiligen Gruppe gemeinsam eine Größe messen und somit eine gemeinsame Aussage ermöglichen. Tabelle 6.2 listet das Cronbachsche Alpha für jede Gruppe auf.

ID	Frage
PEOU_1	Ich habe die Funktionen des Tutor Assistants schnell verstanden
PEOU_2	Es war umständlich, mit dem Tutor Assistant zu interagieren
PEOU_3	Die Bedienung des Tutor Assistants war intuitiv
PU_1	Der Tutor Assistant half mir, die gewünschten Informationen abzurufen
PU_2	Der Tutor Assistant verstand meine Anfragen richtig
PU_3	Die Antworten waren hilfreich und informativ
PU_4	Der Tutor Assistant konnte mir bei unterschiedlichen Anfragen helfen
PU_5	Der Tutor Assistant hat meine Produktivität erhöht
PU_6	Der Tutor Assistant hat die Qualität meiner Arbeit verbessert
PU_7	Der Tutor Assistant vereinfachte meine Arbeit
AT_1	Ich bin positiv eingestellt gegenüber dem Tutor Assistant
AT_2	Die Zusammenarbeit mit dem Tutor Assistant ist nicht zielführend
AT_3	Ich würde den Tutor Assistant anderen weiterempfehlen
AT_4	Ich möchte den Tutor Assistant weiterhin verwenden
CO_1	Ich war zuversichtlich, den Tutor Assistant bei meiner Arbeit einzusetzen
CO_2	Ich hatte die notwendigen Fähigkeiten für die Bedienung des Tutor Assistants
CO_3	Ich hatte Bedenken bezüglich Datenschutz
SA_1	Ich hatte keine Schwierigkeiten auf den Tutor Assistant zuzugreifen

**Tabelle 6.1.:** Fragebogen der Umfrage

PEOU: Bedienbarkeit, PU: Nutzen, AT: Einstellung, CO: Bedenken, SA: Zugriff.

AT\_4 ist ursprünglich als Frage zur Intention gestellt worden.

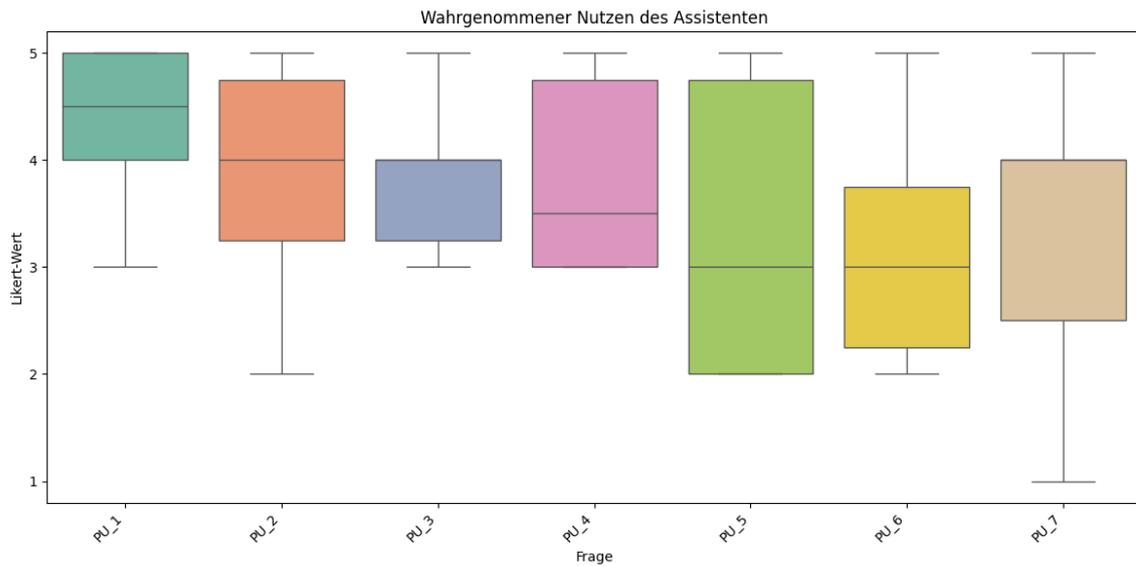
Gruppe	Cronbachsches Alpha
Bedienbarkeit	-1, 1
Nutzen	0.75
Einstellung	0.86
Bedenken	-0.34

**Tabelle 6.2.:** Cronbachsches Alpha für jede Gruppe

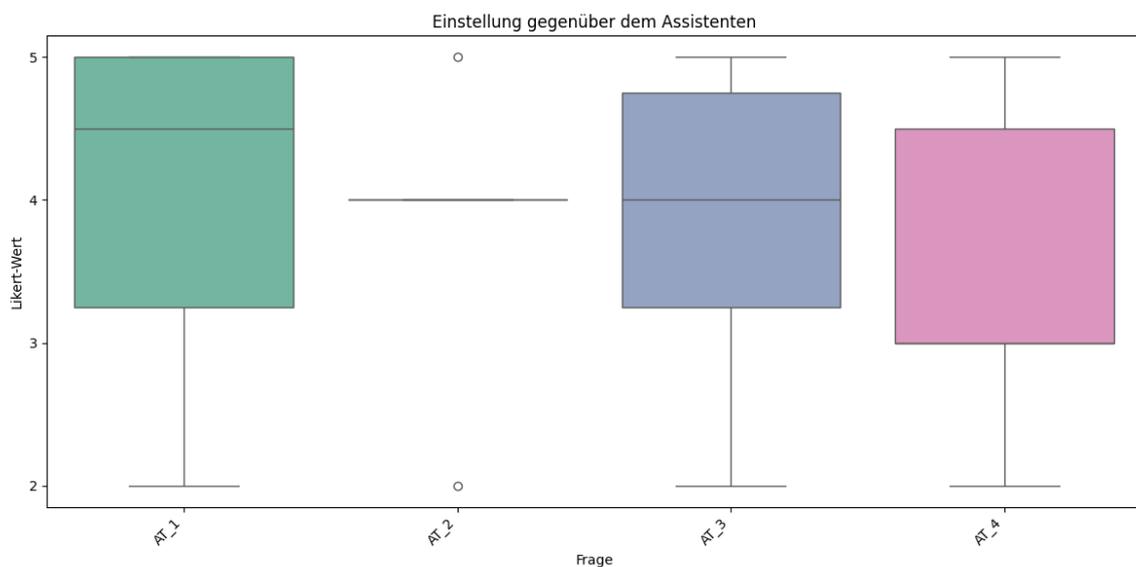
Damit eine Gruppe eine verlässliche Aussage liefert muss das Cronbachschen Alpha je nach Quelle bei mindestens 0,6 oder 0,7 liegen [32]. Tabelle 6.2 zeigt, dass dies für die Gruppen Nutzen und Einstellung gilt. Diese werden im Folgenden näher untersucht.

Um einen Überblick über die Gruppen Nutzen und Einstellung zu erhalten, werden sie in Abbildung 6.1 und Abbildung 6.2 als Boxplots dargestellt. Es ist erkennbar, dass die Daten zum Teil stark gestreut sind. Das deutet darauf hin, dass die Meinungen der Tutoren stark voneinander abweichen. Die meisten Werte liegen jedoch im neutralen bis positiven Bereich. Es kann somit geschlussfolgert werden, dass die meisten Tutoren einen gewissen Nutzen in dem Assistenten sehen und ihm gegenüber grundsätzlich positiv eingestellt sind.

Die Daten, auf denen die Boxplots basieren, sind in Tabelle Tabelle 6.3 dargestellt. Sie zeigt Median, arithmetisches Mittel (Mittelwert) und Varianz für jede Frage für die Gruppe Nutzen (PU) und Einstellung (AT) an. Zudem zeigt sie jeweils den Median der Medians sowie das



**Abbildung 6.1.:** Wahrgenommener Nutzern des Assistenten durch die Tutoren.



**Abbildung 6.2.:** Einstellung der Tutoren gegenüber dem Assistenten

arithmetische Mittel der Mittelwerte und der Varianzen als resultierende Kennzahlen an. Diese bestätigen den Eindruck, den die Boxplots geben. Insgesamt ist für beide Gruppen eine positive Tendenz erkennbar, die Varianz ist jedoch groß.

Neben den TAM-basierten Fragen ist in der Umfrage auch danach gefragt worden, was den Tutoren am Assistenten gefallen hat und was nicht. Positiv wird mehrfach empfunden, dass der Assistent nützliche und verlässliche Informationen liefert. Zudem sei die Angabe der Quellen hilfreich. Als negative Aspekte werden genannt, dass zum Teil nur eine Quelle ange-

Frage	Median	Mittelwert	Varianz
PU_1	4,5	4,3	0,7
PU_2	4,0	3,8	1,4
PU_3	4,0	3,8	0,6
PU_4	3,5	3,8	1,0
PU_5	3,0	3,3	2,3
PU_6	3,0	3,2	1,4
PU_7	4,0	3,3	2,3
PU_ges	4,0	3,7	1,4
AT_1	4,5	4,0	1,6
AT_2	4,0	3,8	1,0
AT_3	4,0	3,8	1,4
AT_4	3,0	3,5	1,5
AT_ges	4,0	3,8	1,4

**Tabelle 6.3.:** Kennzahlen zur Bewertung des Nutzen (PU) und der Einstellung (AT)  
 Es werden die Werte pro Frage (1..\*) aufgelistet sowie die Gesamt-Mediane / -Mittelwerte /  
 -Varianzen (ges).

geben wird und der Assistent häufig die Tutor-Rolle des Benutzers nicht in den Antworten berücksichtigt.

### 6.2.1. Schlussfolgerung

Die Ergebnisse der verschiedenen Teile der Evaluation stimmen weitgehend miteinander überein. Der Assistent ist in der Lage, zuverlässige und nützliche Antworten zu generieren. Das Anzeigen der Quellen ist dabei ein großer Vorteil. Noch hilfreicher wären die Antworten, wenn der Assistent Antworten mit noch mehr Bezug zur Vorlesung und die Arbeit der Tutoren geben würde. Insgesamt sind die Tutoren dem Assistenten gegenüber eher positiv eingestellt.

## 7. Fazit

Es ist ein Assistent entwickelt worden, der Tutoren relevante Informationen zur Vorlesung und ihrer Arbeit schnell und einfach zur Verfügung stellen kann. Dazu ist ein RAG-Prozess implementiert worden, welcher einem Sprachmodell die gefragten Inhalte zur Verfügung stellt. Dabei ist entscheidend, ob die richtigen Dokumente vom Retriever abgerufen werden. Um dies zu optimieren sind mehrere Retriever entwickelt und evaluiert worden. Zudem sind mit jeder Anfrage an das Sprachmodell grundlegende Informationen der Anfrage beigefügt worden, die die Antworten verbessern sollten.

Der entwickelte Assistent ist im Anschluss mithilfe der aktuellen Tutoren evaluiert worden. Die Evaluation hat ergeben, dass der Assistent in der Lage ist, Fragen zuverlässig zu beantworten. Auch wenn die meisten Antworten korrekt sind, könnten sie jedoch häufig einen stärkeren Bezug zur Vorlesung und der Arbeit der Tutoren herstellen. Für eine verlässliche Aussage bezüglich der Meinung der Tutoren war die Stichprobe der Tutoren, die an der Umfrage teilgenommen haben zu klein. Es scheint jedoch, dass die Tutoren eher positiv gegenüber dem Assistenten eingestellt sind und einen Nutzen in ihm erkennen.

Besondere Stärken des Assistenten folgen aus der Optimierung des Retriever. Der `GeneratedQueriesRetriever` ist dabei ein wichtiger Fortschritt gewesen, um auch für komplexe Anfragen die richtigen Dokumente aus dem Vectorstore abzurufen. Zudem ermöglicht dieser Retriever, dass der Kontext des gesamten Chat-Verlaufs beim Abrufen der Dokumente berücksichtigt werden kann. Dies wird erreicht, indem ein Sprachmodell basierend auf der Anfrage im Kontext des Chat-Verlaufs Anfragen an den Vectorstore generiert. Weitere Retriever haben bei der Untersuchung keine signifikanten Verbesserungen gezeigt. Ein Beispiel dafür ist der `WithReferencesRetriever`. Es ist jedoch denkbar, dass dieser in anderen Kontexten oder mit weiteren Optimierungen die Ergebnisse des Abrufs verbessern kann.

Als zentrale Herausforderung bleibt, dem Assistenten beizubringen, im Kontext der Vorlesung und der Arbeit der Tutoren Antworten zu generieren. Einige dafür notwendige Informationen werden bereits in jeder Anfrage dem Sprachmodell mitgegeben, jedoch nicht in die Antwort mit einbezogen. Dies könnte über weiteres Prompt-Engineering optimiert werden. Weitere Optimierungen wären beim Retriever möglich, um insgesamt weniger irrelevante und für manche Anfragen mehr relevante Dokumente abzurufen. Dies würde die Effizienz der Anfragen steigern und Antworten ermöglichen, die besonders viele Dokumente benötigen.



# Literatur

- [1] Ryan C. Barron u. a. *Domain-Specific Retrieval-Augmented Generation Using Vector Stores, Knowledge Graphs, and Tensor Factorization*. arXiv:2410.02721. Okt. 2024. DOI: 10.48550/arXiv.2410.02721. URL: <http://arxiv.org/abs/2410.02721> (besucht am 28. 11. 2024).
- [2] Tom Brown u. a. „Language Models are Few-Shot Learners“. In: *Advances in Neural Information Processing Systems*. Bd. 33. Curran Associates, Inc., 2020, S. 1877–1901. URL: <https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfc4967418bfb8ac142f64a-Abstract.html> (besucht am 11. 08. 2024).
- [3] Deborah Carlander u. a. „Controlled Chain of Thought: Eliciting Role-Play Understanding in LLM Through Prompts“. In: *2024 IEEE Conference on Games (CoG)*. ISSN: 2325-4289. Aug. 2024, S. 1–4. DOI: 10.1109/CoG60054.2024.10645667. URL: [https://ieeexplore.ieee.org/abstract/document/10645667?casa\\_token=7LUm\\_bIUptUAAAAA:T-tHh43wbF\\_GM3eF-dSG8xY8rKjGgdhqzTS9IoSvKQv51pHm8MZy1Tf0uEVzPA7Epoj0K0zU](https://ieeexplore.ieee.org/abstract/document/10645667?casa_token=7LUm_bIUptUAAAAA:T-tHh43wbF_GM3eF-dSG8xY8rKjGgdhqzTS9IoSvKQv51pHm8MZy1Tf0uEVzPA7Epoj0K0zU) (besucht am 10. 12. 2024).
- [4] Qinyuan Cheng u. a. *Unified Active Retrieval for Retrieval Augmented Generation*. arXiv:2406.12534 [cs]. Juni 2024. DOI: 10.48550/arXiv.2406.12534. URL: <http://arxiv.org/abs/2406.12534> (besucht am 11. 08. 2024).
- [5] K. R. Chowdhary. „Natural Language Processing“. en. In: *Fundamentals of Artificial Intelligence*. Hrsg. von K.R. Chowdhary. New Delhi: Springer India, 2020, S. 603–649. ISBN: 978-81-322-3972-7. DOI: 10.1007/978-81-322-3972-7\_19. URL: [https://doi.org/10.1007/978-81-322-3972-7\\_19](https://doi.org/10.1007/978-81-322-3972-7_19) (besucht am 04. 06. 2024).
- [6] Radoslaw M. Cichy und Daniel Kaiser. „Deep Neural Networks as Scientific Models“. English. In: *Trends in Cognitive Sciences* 23.4 (Apr. 2019). Publisher: Elsevier, S. 305–317. ISSN: 1364-6613, 1879-307X. DOI: 10.1016/j.tics.2019.01.009. URL: [https://www.cell.com/trends/cognitive-sciences/abstract/S1364-6613\(19\)30034-8](https://www.cell.com/trends/cognitive-sciences/abstract/S1364-6613(19)30034-8) (besucht am 28. 11. 2024).
- [7] Lee J. Cronbach. „Coefficient alpha and the internal structure of tests“. en. In: *Psychometrika* 16.3 (Sep. 1951), S. 297–334. ISSN: 1860-0980. DOI: 10.1007/BF02310555. URL: <https://doi.org/10.1007/BF02310555> (besucht am 09. 12. 2024).
- [8] Wei Dai u. a. „Can Large Language Models Provide Feedback to Students? A Case Study on ChatGPT“. In: *2023 IEEE International Conference on Advanced Learning Technologies (ICALT)*. ISSN: 2161-377X. Juli 2023, S. 323–325. DOI: 10.1109/ICALT58122.2023.00100. URL: <https://ieeexplore.ieee.org/abstract/document/10260740> (besucht am 04. 08. 2024).

- [9] Fred D. Davis. „Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology“. In: *MIS Quarterly* 13.3 (1989). Publisher: Management Information Systems Research Center, University of Minnesota, S. 319–340. ISSN: 0276-7783. DOI: 10.2307/249008. URL: <https://www.jstor.org/stable/249008> (besucht am 11. 12. 2024).
- [10] Kawin Ethayarajh. *Rotate King to get Queen: Word Relationships as Orthogonal Transformations in Embedding Space*. arXiv:1909.00504 [cs]. Sep. 2019. DOI: 10.48550/arXiv.1909.00504. URL: <http://arxiv.org/abs/1909.00504> (besucht am 11. 08. 2024).
- [11] Eduard Frankford u. a. „AI-Tutoring in Software Engineering Education“. In: *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*. arXiv:2404.02548 [cs]. Apr. 2024, S. 309–319. DOI: 10.1145/3639474.3640061. URL: <http://arxiv.org/abs/2404.02548> (besucht am 12. 06. 2024).
- [12] Yunfan Gao u. a. *Retrieval-Augmented Generation for Large Language Models: A Survey*. arXiv:2312.10997 [cs]. März 2024. DOI: 10.48550/arXiv.2312.10997. URL: <http://arxiv.org/abs/2312.10997> (besucht am 12. 08. 2024).
- [13] Jose Manuel Gomez-Perez, Ronald Denaux und Andres Garcia-Silva. „Understanding Word Embeddings and Language Models“. en. In: *A Practical Guide to Hybrid Natural Language Processing: Combining Neural Models and Knowledge Graphs for NLP*. Hrsg. von Jose Manuel Gomez-Perez, Ronald Denaux und Andres Garcia-Silva. Cham: Springer International Publishing, 2020, S. 17–31. ISBN: 978-3-030-44830-1. DOI: 10.1007/978-3-030-44830-1\_3. URL: [https://doi.org/10.1007/978-3-030-44830-1\\_3](https://doi.org/10.1007/978-3-030-44830-1_3) (besucht am 28. 11. 2024).
- [14] Arto Hellas u. a. „Exploring the Responses of Large Language Models to Beginner Programmers’ Help Requests“. In: *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*. Bd. 1. ICER ’23. New York, NY, USA: Association for Computing Machinery, Sep. 2023, S. 93–105. ISBN: 978-1-4503-9976-0. DOI: 10.1145/3568813.3600139. URL: <https://dl.acm.org/doi/10.1145/3568813.3600139> (besucht am 04. 08. 2024).
- [15] Luis-Bernardo Hernandez-Salinas u. a. „IDAS: Intelligent Driving Assistance System Using RAG“. In: *IEEE Open Journal of Vehicular Technology* 5 (2024). Conference Name: IEEE Open Journal of Vehicular Technology, S. 1139–1165. ISSN: 2644-1330. DOI: 10.1109/OJVT.2024.3447449. URL: <https://ieeexplore.ieee.org/abstract/document/10643289> (besucht am 27. 11. 2024).
- [16] Yunpeng Huang u. a. *Advancing Transformer Architecture in Long-Context Large Language Models: A Comprehensive Survey*. arXiv:2311.12351. Feb. 2024. DOI: 10.48550/arXiv.2311.12351. URL: <http://arxiv.org/abs/2311.12351> (besucht am 28. 11. 2024).
- [17] S. Joshua Johnson, M. Ramakrishna Murty und I. Navakanth. „A detailed review on word embedding techniques with emphasis on word2vec“. en. In: *Multimedia Tools and Applications* 83.13 (Apr. 2024), S. 37979–38007. ISSN: 1573-7721. DOI: 10.1007/s11042-

- 
- 023-17007-z. URL: <https://doi.org/10.1007/s11042-023-17007-z> (besucht am 28. 11. 2024).
- [18] Jean Kaddour u. a. *Challenges and Applications of Large Language Models*. arXiv:2307.10169 [cs]. Juli 2023. DOI: 10.48550/arXiv.2307.10169. URL: <http://arxiv.org/abs/2307.10169> (besucht am 18. 06. 2024).
- [19] Enkelejda Kasneci u. a. „ChatGPT for good? On opportunities and challenges of large language models for education“. In: *Learning and Individual Differences* 103 (Apr. 2023), S. 102274. ISSN: 1041-6080. DOI: 10.1016/j.lindif.2023.102274. URL: <https://www.sciencedirect.com/science/article/pii/S1041608023000195> (besucht am 18. 06. 2024).
- [20] Diksha Khurana u. a. „Natural language processing: state of the art, current trends and challenges“. en. In: *Multimedia Tools and Applications* 82.3 (Jan. 2023), S. 3713–3744. ISSN: 1573-7721. DOI: 10.1007/s11042-022-13428-4. URL: <https://doi.org/10.1007/s11042-022-13428-4> (besucht am 28. 11. 2024).
- [21] Quinn Leng u. a. *Long Context RAG Performance of Large Language Models*. arXiv:2411.03538 [cs]. Nov. 2024. DOI: 10.48550/arXiv.2411.03538. URL: <http://arxiv.org/abs/2411.03538> (besucht am 09. 12. 2024).
- [22] Ron Li, Andre Kumar und Jonathan H. Chen. „How Chatbots and Large Language Model Artificial Intelligence Systems Will Reshape Modern Medicine: Fountain of Creativity or Pandora’s Box?“ In: *JAMA Internal Medicine* 183.6 (Juni 2023), S. 596–597. ISSN: 2168-6106. DOI: 10.1001/jamainternmed.2023.1835. URL: <https://doi.org/10.1001/jamainternmed.2023.1835> (besucht am 27. 11. 2024).
- [23] Esmaeil Narimissa und David Raithel. *Exploring Information Retrieval Landscapes: An Investigation of a Novel Evaluation Techniques and Comparative Document Splitting Methods*. arXiv:2409.08479. Sep. 2024. DOI: 10.48550/arXiv.2409.08479. URL: <http://arxiv.org/abs/2409.08479> (besucht am 28. 11. 2024).
- [24] Alexander Tobias Neumann u. a. „An LLM-Driven Chatbot in Higher Education for Databases and Information Systems“. In: *IEEE Transactions on Education* (2024). Conference Name: IEEE Transactions on Education, S. 1–14. ISSN: 1557-9638. DOI: 10.1109/TE.2024.3467912. URL: <https://ieeexplore.ieee.org/abstract/document/10706931> (besucht am 22. 11. 2024).
- [25] *OpenAI Vector embeddings*. URL: <https://platform.openai.com/docs/guides/embeddings/what-are-embeddings?lang=node> (besucht am 28. 11. 2024).
- [26] Dhavalkumar Patel u. a. *Exploring Temperature Effects on Large Language Models Across Various Clinical Tasks*. en. ISSN: 2431-0824 Pages: 2024.07.22.24310824. Juli 2024. DOI: 10.1101/2024.07.22.24310824. URL: <https://www.medrxiv.org/content/10.1101/2024.07.22.24310824v1> (besucht am 27. 11. 2024).
- [27] Alina Petukhova, João P. Matos-Carvalho und Nuno Fachada. *Text clustering with LLM embeddings*. arXiv:2403.15112 [cs]. Mai 2024. DOI: 10.48550/arXiv.2403.15112. URL: <http://arxiv.org/abs/2403.15112> (besucht am 11. 08. 2024).

- [28] Nived Rajaraman, Jiantao Jiao und Kannan Ramchandran. *Toward a Theory of Tokenization in LLMs*. arXiv:2404.08335 [cs]. Apr. 2024. DOI: 10.48550/arXiv.2404.08335. URL: <http://arxiv.org/abs/2404.08335> (besucht am 11. 08. 2024).
- [29] R. Rosenfeld. „Two decades of statistical language modeling: where do we go from here?“ In: *Proceedings of the IEEE* 88.8 (Aug. 2000). Conference Name: Proceedings of the IEEE, S. 1270–1278. ISSN: 1558-2256. DOI: 10.1109/5.880083. URL: <https://ieeexplore.ieee.org/abstract/document/880083> (besucht am 27. 11. 2024).
- [30] Tolga Şakar und Hakan Emekci. „Maximizing RAG efficiency: A comparative analysis of RAG methods“. en. In: *Natural Language Processing* (Okt. 2024), S. 1–25. ISSN: 2977-0424. DOI: 10.1017/nlp.2024.53. URL: <https://www.cambridge.org/core/journals/natural-language-processing/article/maximizing-rag-efficiency-a-comparative-analysis-of-rag-methods/D7B259BCD35586E04358DF06006E0A85> (besucht am 28. 11. 2024).
- [31] Julien Siebert, Thorsten Honroth und Patricia Kelbert. *Prompt Engineering: wie man mit großen Sprachmodellen kommuniziert*. Mai 2024. URL: <https://www.iese.fraunhofer.de/blog/was-ist-prompt-engineering/> (besucht am 12. 08. 2024).
- [32] Keith S. Taber. „The Use of Cronbach’s Alpha When Developing and Reporting Research Instruments in Science Education“. en. In: *Research in Science Education* 48.6 (Dez. 2018), S. 1273–1296. ISSN: 1573-1898. DOI: 10.1007/s11165-016-9602-2. URL: <https://doi.org/10.1007/s11165-016-9602-2> (besucht am 09. 12. 2024).
- [33] Vithursan Thangarasa u. a. „SPDF: Sparse Pre-training and Dense Fine-tuning for Large Language Models“. en. In: *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*. ISSN: 2640-3498. PMLR, Juli 2023, S. 2134–2146. URL: <https://proceedings.mlr.press/v216/thangarasa23a.html> (besucht am 12. 08. 2024).
- [34] Ashish Vaswani u. a. *Attention Is All You Need*. arXiv:1706.03762. Aug. 2023. DOI: 10.48550/arXiv.1706.03762. URL: <http://arxiv.org/abs/1706.03762> (besucht am 28. 11. 2024).
- [35] Jules White u. a. *A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT*. arXiv:2302.11382 [cs]. Feb. 2023. DOI: 10.48550/arXiv.2302.11382. URL: <http://arxiv.org/abs/2302.11382> (besucht am 10. 12. 2024).
- [36] Thomas Wolf u. a. „Transformers: State-of-the-Art Natural Language Processing“. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Hrsg. von Qun Liu und David Schlangen. Online: Association for Computational Linguistics, Okt. 2020, S. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6. URL: <https://aclanthology.org/2020.emnlp-demos.6> (besucht am 28. 11. 2024).
- [37] Yu-chen Wu und Jun-wen Feng. „Development and Application of Artificial Neural Network“. en. In: *Wireless Personal Communications* 102.2 (Sep. 2018), S. 1645–1656. ISSN: 1572-834X. DOI: 10.1007/s11277-017-5224-x. URL: <https://doi.org/10.1007/s11277-017-5224-x> (besucht am 28. 11. 2024).

- 
- [38] Wayne Xin Zhao u. a. *A Survey of Large Language Models*. arXiv:2303.18223 [cs]. Nov. 2023. DOI: 10.48550/arXiv.2303.18223. URL: <http://arxiv.org/abs/2303.18223> (besucht am 11. 08. 2024).



# A. Anhang

## A.1. Studie zur Anforderungsanalyse

Frage	Antwort-Typ
Wie oft warst du schon Tutor:in für Programmieren?	Häufigkeit (1, 2, >2)
Wie oft warst du schon Tutor:in allgemein?	Häufigkeit (1, 2, >2)
Wie gut schätzt du deine Kenntnisse in der Java-Programmierung ein? (1 = schlecht, 5 = herausragend)	Likert (1-5)
Während meiner Arbeit bin ich sehr gut mit den relevanten Materialien (Vorlesungswebsites, Protokollen, Chats, Übungsblättern, etc.) vertraut. (1 = trifft nicht zu, 5 = trifft voll zu)	Likert (1-5)
Wie viel Zeit (1 = wenig, 5 = viel) investierst du in [Korrigieren]	Likert (1-5)
Wie viel Zeit (1 = wenig, 5 = viel) investierst du in [Vorbereitung von Tutorien]	Likert (1-5)
Wie viel Zeit (1 = wenig, 5 = viel) investierst du in [Beantwortung von Fragen der Studierenden]	Likert (1-5)
Wie viel Zeit (1 = wenig, 5 = viel) investierst du in [Andere Tätigkeiten]	Likert (1-5)
Welche anderen Tätigkeiten?	Freitext
Für meine Tätigkeiten muss ich oft Informationen nachschauen. (1 = trifft nicht zu, 5 = trifft voll zu)	Likert (1-5)
Für welche Tätigkeiten musst du wie oft Informationen nachschauen? (1 = selten, 5 = oft) [Korrigieren]	Likert (1-5)
Für welche Tätigkeiten musst du wie oft Informationen nachschauen? (1 = selten, 5 = oft) [Vorbereiten von Tutorien]	Likert (1-5)
Für welche Tätigkeiten musst du wie oft Informationen nachschauen? (1 = selten, 5 = oft) [Beantwortung von Fragen der Studierenden]	Likert (1-5)
Für welche Tätigkeiten musst du wie oft Informationen nachschauen? (1 = selten, 5 = oft) [Andere Tätigkeiten]	Likert (1-5)
Welche Informationen musst du typischerweise nachschauen?	Freitext
Ich habe häufig das Gefühl, dass ich besser informiert sein sollte. (1 = trifft nicht zu, 5 = trifft voll zu)	Likert (1-5)
Ich denke, dass ein vereinfachter Informationsgewinn nützlich sein kann. (1 = trifft nicht zu, 5 = trifft voll zu)	Likert (1-5)

Ich denke, dass eine KI wie ChatGPT mich dabei unterstützen kann. (1 = trifft nicht zu, 5 = trifft voll zu)	Likert (1-5)
Ich würde einer KI wie ChatGPT vertrauen, mir die korrekte Antwort zu liefern. (1 = trifft nicht zu, 5 = trifft voll zu)	Likert (1-5)
Ich würde jede Antwort der KI wie ChatGPT überprüfen wollen. (1 = trifft nicht zu, 5 = trifft voll zu)	Likert (1-5)
Ich verwende ChatGPT oder ähnliche Assistenten häufig im Alltag. (1 = trifft nicht zu, 5 = trifft voll zu)	Likert (1-5)
Wenn nicht ChatGPT, was dann?	Freitext
Ich verwende die Antworten von ChatGPT oder ähnlichen Assistenten häufig ohne sie zu überprüfen. (1 = trifft nicht zu, 5 = trifft voll zu)	Likert (1-5)
Für welche Datenquellen würdest du besonders gerne den Assistenten verwenden (bspw. Vorlesungswebsite, Übungsblätter, Protokolle, Chat-Verlauf, etc.)?	Freitext
Warum würdest du den Assistenten gerade für diese Datenquellen besonders gerne verwenden?	Freitext
Welche Art von Benutzerschnittstelle würdest du bevorzugen?	Website, Chat
Welche Art von Benutzerschnittstelle würdest du bevorzugen? [Sonstiges]	Freitext
Welche Funktionalitäten fändest du hilfreich? [Antworten auf Fragen bekommen]	ja/nein
Welche Funktionalitäten fändest du hilfreich? [Relevante Informationen automatisch angezeigt bekommen]	ja/nein
Welche Funktionalitäten fändest du hilfreich? [Antworten zu häufig gestellten Fragen automatisch angezeigt bekommen]	ja/nein
Welche Funktionalitäten fändest du hilfreich? [Sonstiges]	Freitext
Welche weiteren Ideen hättest du noch allgemein?	Freitext

**Tabelle A.1.:** Fragen der Studie zur Anforderungsanalyse

## A.2. Code in Anfrage

```
public void printVehicle(String type) {
    switch (type) {
        case "CAR":
            System.out.println("This is a car.");
            break;
        case "TRUCK":
            System.out.println("This is a truck.");
            break;
        case "MOTORCYCLE":
            System.out.println("This is a motorcycle.");
            break;
        default:
            System.out.println("Unknown vehicle type.");
    }
}
```

### A.3. Beispiel für einen Chat-Verlauf

Was hältst du davon Getter für eine Klasse zu schreiben, die nicht verwendet werden?

1. Ist das im allgemeinen eher gut oder schlecht? Warum?

2. Wie sind die regeln in unserem Programmieren Modul?

- 1. Im Allgemeinen:** Es ist im Allgemeinen **nicht empfehlenswert**, Getter (oder Setter) für eine Klasse zu schreiben, die nicht verwendet werden. Der Grund dafür ist, dass ungenutzte Methoden den Code unnötig aufblähen und die Lesbarkeit sowie Wartbarkeit des Codes erschweren können. Sie können auch zu einer größeren Programmgröße führen und das Debugging erschweren. Wenn eine Methode in der Zukunft benötigt wird, kann sie dann hinzugefügt werden.
- 2. Regeln in eurem Programmieren Modul:** In eurem Programmieren Modul wird empfohlen, unbenutzte Elemente wie Klassen, Methoden und Zeilen im finalen Produkt zu entfernen, da sie keinen weiteren Nutzen haben und den Lesefluss stören. Auch wenn der Code im aktuellen Zustand einen überschaubaren Umfang hat, sollte darauf geachtet werden, dass im finalen Release keine unbenutzten Elemente vorhanden sind. Moderne IDEs weisen auf unbenutzte Elemente hin, und es wird als leichtes Vergehen angesehen, solche Elemente im Code zu belassen.



Sehr guter Bezug auf das Programmieren Wiki.

Senden

Kann man nicht auch argumentieren, dass das Open-Closed-Principle dafür spricht getter zu schreiben, die aktuell noch nicht, aber später vielleicht schon benötigt werden?

Das **Open-Closed-Principle** besagt, dass Software-Komponenten **offen für Erweiterung, aber geschlossen für Änderung** sein sollten. Das bedeutet, dass man das Verhalten einer Klasse erweitern können sollte, ohne den bestehenden Code zu ändern.

#### Argumentation für das Schreiben von Gettern im Voraus:

- Man könnte argumentieren, dass das Vorhandensein von Gettern die Klasse für zukünftige Erweiterungen vorbereitet, ohne dass bestehender Code geändert werden muss. Dies könnte theoretisch im Einklang mit dem Open-Closed-Principle stehen, da die Klasse bereits die Schnittstellen bereitstellt, die für zukünftige Erweiterungen benötigt werden könnten.

#### Gegenargumente:

- Das Vorhandensein von ungenutzten Gettern kann den Code unnötig aufblähen und die Wartbarkeit erschweren, was im Widerspruch zu guten Designpraktiken steht.
- Das Open-Closed-Principle bezieht sich eher auf die Fähigkeit, neue Funktionalitäten hinzuzufügen, ohne den bestehenden Code zu ändern, als auf das Vorhandensein ungenutzter Methoden.
- Moderne Entwicklungspraktiken und Werkzeuge ermöglichen es, Methoden schnell hinzuzufügen, wenn sie benötigt werden, ohne dass dies als signifikante Änderung betrachtet wird.

Insgesamt ist es in der Praxis oft besser, nur die Methoden zu implementieren, die tatsächlich benötigt werden, und den Code bei Bedarf zu erweitern. Dies hält den Code sauber und wartbar.

Abbildung A.1.: Beispiel für Chat-Verlauf. Die Nachrichten haben einen guten Bezug zur Vorlesung.