Modifikation des Selbststudiums im digitalen Lernalltag: Ein KI-Dokumentenassistent für Studierende zur effizienten Suche und Aufbereitung von Lernmaterialien

Bachelorarbeit zur Erlangung des akademischen Grades
Bachelor of Science
im Studiengang Wirtschaftsinformatik
an der Fakultät für Fakultät für Informatik und Ingenieurwissenschaften
der Technischen Hochschule Köln

vorgelegt von:	Niklas Voß
Matrikel-Nr.:	111 55 481
Adresse:	Meygner Busch 44. 40880 Ratingen niklas.voss@smail.th-koeln.de
eingereicht bei:	Prof. Dr. Daniel Gaida
Zweitgutachter*in:	Tim Yago Nordhoff

Ratingen, 26.11.2025

Kurzfassung

Die Arbeit untersucht die wachsende Informationsfragmentierung im digitalen Selbststudium und adressiert das daraus entstehende Forschungsproblem, dass Studierende Lernmaterialien zunehmend in heterogenen Formaten, Plattformen und Tools organisieren müssen. Klassische Suchsysteme stoßen dabei an Grenzen, da sie keine semantischen Zusammenhänge erfassen. Ziel der Arbeit ist es, ein datenschutzkonformes, lokal betreibbares RAG-System zu entwickeln und systematisch zu evaluieren, das Studierende beim Wiederfinden und Verstehen digitaler Lernmaterialien unterstützt.

Theoretisch stützt sich die Untersuchung auf aktuelle RAG-Forschung, Wissensmanagement im Hochschulkontext und semantische Retrievalverfahren. Methodisch entsteht ein modularer Open-Source-Prototyp, bei dem die Pipelinekomponenten wie Parser, Chunking, Embeddings und Retrieval systematisch variiert werden. Die Evaluation erfolgte zweistufig. Zunächst wurden 56 Pipeline-Konfigurationen automatisiert generiert und vorgefiltert, anschließend wurden 12 Varianten mithilfe des RAGAS-Frameworks detailliert bewertet. Ein kuratierter Golden-Standard-Datensatz mit 46 Frage-Antwort-Paaren bildete die Basis für objektive, reproduzierbare Qualitätsvergleiche.

Die Ergebnisse zeigen deutliche Leistungsunterschiede zwischen den Konfigurationen. Strukturorientiertes Chunking, leistungsfähige Parser und Dense Retrieval erweisen sich als zentrale Qualitätsfaktoren. Cross-Encoder-Reranking steigert die Antwortqualität um rund zehn Prozentpunkte. Lokale Open-Source-Embeddings können mit cloudbasierten Modellen konkurrieren. Semantisches Chunking und stark fragmentierende Parser führen dagegen häufig zu Fehlretrieval und Halluzinationen.

Die Arbeit schließt, dass datensouveräne, lokal betriebene RAG-Systeme im Hochschulkontext technisch realisierbar und qualitativ konkurrenzfähig sind. Sie liefert methodische Impulse für die Evaluation von RAG-Pipelines und konkrete Designempfehlungen für Hochschulen.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

NLP Natural Language Processing. 8

RAG Retrieval-Augmented Generation. 13

1. Einleitung

Aus Gründen der besseren Lesbarkeit wird in diesem Text die männliche Form verwendet, wobei alle Geschlechter ausdrücklich mitgemeint sind.

Die fortschreitende Digitalisierung verändert das hochschulische Selbststudium grundlegend. Lernmaterialien liegen zunehmend in digitalen Formaten vor, verteilt über verschiedene Plattformen und Systeme. Lernmanagementsysteme bieten zwar einen orts- und zeitunabhängigen Zugang zu Ressourcen, zugleich verschiebt sich die Verantwortung für die Strukturierung, Organisation und Integration dieser Inhalte stärker auf die Studierenden. Damit steigt die kognitive Belastung im individuellen Wissensmanagement erheblich. Die resultierende Informationsfragmentierung erschwert das Wiederfinden, Verknüpfen und Verständnis relevanter Inhalte und mindert die Effizienz des Selbststudiums. Empirische Untersuchungen zeigen, dass Studierende zahlreiche Tools parallel verwenden und dabei stark auf eigene mentale Modelle angewiesen sind. Klassische Suchsysteme, die primär auf lexikalischen Verfahren basieren, adressieren diese Problematik nur unzureichend, da sie semantische Zusammenhänge nicht erfassen und unter dem Vocabulary-Mismatch-Problem leiden.

Vor diesem Hintergrund gewinnen KI-basierte Verfahren an Bedeutung, insbesondere Retrieval-Augmented-Generation-Systeme (RAG), die semantisches Retrieval mit generativen Modellen kombinieren. Trotz ihres Potenzials sind deren Einsatzmöglichkeiten im Hochschulkontext stark eingeschränkt. Datenschutzanforderungen und infrastrukturelle Restriktionen verhindern häufig die Nutzung proprietärer Systeme, die eine Verarbeitung personenbezogener oder urheberrechtlich geschützter Materialien ausschließen. Im Forschungsstand zeigt sich daher eine doppelte Lücke. Zum einen fehlen offene und datenschutzkonforme Ansätze, die RAG-Technologien gezielt für studentische Lernmaterialien nutzbar machen. Zum anderen fehlt es an experimentellen Untersuchungen, wie unterschiedliche Pipelinekomponenten wie Parser, Chunkingstrategien, Embeddingmodelle oder Retrievalverfahren die Ergebnisqualität semantischer Systeme beeinflussen. Diese Forschungslücke bildet den Ausgangspunkt der vorliegenden Arbeit.

Ziel der Arbeit ist es, einen datensouveränen, modular aufgebauten RAG-Prototypen für den Hochschulkontext zu entwickeln und verschiedene Pipeline-Konfigurationen systematisch zu evaluieren. Die Arbeit verfolgt dabei zwei Leitfragen: (1) Welche Kombinationen aus Parsing, Chunking, Embeddings und Retrieval führen zu hoher

Antwortqualität im Umgang mit studentischen Lernmaterialien? (2) Inwieweit können lokal betreibbare, offene Systeme qualitativ mit cloudbasierten Lösungen konkurrieren? Hieraus leitet sich die Hypothese ab, dass strukturorientierte Segmentierung und effektives Reranking einen stärkeren Einfluss auf die Performance besitzen als der Einsatz rechenintensiver oder proprietärer Embedding-Modelle.

Theoretisch verortet sich die Arbeit an der Schnittstelle von Information Retrieval, Semantic Search und generativen KI-Modellen. Forschung zu RAG-Systemen betont die Bedeutung robuster Retrieval- und Segmentierungsmechanismen, da generative Modelle nur im Rahmen der bereitgestellten Kontextevidenz antworten können. Zudem weisen aktuelle Studien auf den hohen Einfluss parser- und chunkingbedingter Variabilität hin, während Evaluationsansätze zunehmend auf modellbasierte Frameworks wie RAGAS zurückgreifen. Ein zentrales Desiderat bleibt jedoch die systematische Untersuchung vollständiger Pipeline-Varianten in realistischen Anwendungsszenarien des digitalen Lernens.

Methodisch folgt die Arbeit einem zweistufigen Vorgehen. Zunächst wurden sämtliche technisch sinnvollen Kombinationen der vier zentralen Pipeline-Komponenten generiert und automatisiert ausgeführt. Anschließend erfolgte ein Vorfiltering der Ergebnisse anhand quantitativer Retrieval- und Ähnlichkeitsmetriken. Die vielversprechendsten Varianten wurden daraufhin mithilfe des RAGAS-Frameworks detailliert analysiert. Grundlage bildet ein eigens erstellter Golden-Standard-Datensatz mit validierten Frage-Antwort-Paaren, der reproduzierbare und modellunabhängige Vergleiche ermöglicht. Dieses Vorgehen erlaubt eine präzise Isolierung der Effekte einzelner Komponenten und gewährleistet Transparenz, Reliabilität und Reproduzierbarkeit.

Die Arbeit gliedert sich in sechs Kapitel. Kapitel 1 führt in Problemstellung, Relevanz und Zielsetzung ein. Kapitel 2 stellt den theoretischen Rahmen dar und systematisiert den aktuellen Forschungsstand zu RAG-Systemen, semantischem Retrieval und digitalen Lernumgebungen. Kapitel 3 systematisiert den aktuellen Stand der Forschung zu Retrieval Augmented Generation Ansätzen und bildet damit die Grundlage für die weitere Untersuchung. Darauf aufbauend erläutert Kapitel 4 das methodische Design, die Erstellung des Datensatzes und das Evaluationsverfahren. Kapitel 5 entwickelt die funktionalen und nicht-funktionalen Anforderungen sowie die Systemarchitektur des RAG-Prototyps. Kapitel 6 erläutert die Implementierung und die technische Ausgestaltung der modularen Pipeline. Kapitel 7 präsentiert und diskutiert die Ergebnisse der systematischen Evaluation. Das abschließende Kapitel ordnet die Befunde in die Zielsetzung ein, reflektiert Limitationen und skizziert Perspektiven für weiterführende Forschung.

Insgesamt schafft die Arbeit damit einen Beitrag zur empirischen Analyse von RAG-Pipelines im Hochschulkontext und zeigt auf, wie datenschutzkonforme KI-Systeme das digitale Selbststudium wirksam unterstützen können.

Ratingen, November 2025

2. Theoretische Grundlagen

Das vorliegende Kapitel legt die theoretischen Grundlagen dar, die für das Verständnis und die Entwicklung eines KI-gestützten Dokumentenassistenten für das Selbststudium erforderlich sind. Zunächst wird die Rolle des Selbststudiums im digitalen Lernalltag beleuchtet, bevor auf technische Konzepte des Information Retrieval, die Funktionsweise von Large Language Models sowie das Paradigma des Retrieval-Augmented Generation eingegangen wird.

2.1. Selbststudium im digitalen Lernalltag

2.1.1. Charakteristika des Selbststudiums in der Hochschulbildung

Das Selbststudium stellt Studierende vor besondere Herausforderungen. Anders als in Präsenzveranstaltungen, wo externe Strukturen und die Anwesenheit von Lehrenden einen Rahmen vorgeben, sind Studierende hier auf ein hohes Maß an Selbstregulation, intrinsischer Motivation und Zeitmanagement angewiesen (vgl. Zimmerman 2002, S. 64). Studierende sind dabei gefordert, eigenverantwortlich Lernziele zu definieren, geeignete Lernstrategien auszuwählen und den eigenen Lernfortschritt zu überwachen.

Ein zentrales Charakteristikum des Selbststudiums ist die intensive Auseinandersetzung mit unterschiedlichen Dokumententypen. Studierende müssen Vorlesungsfolien, wissenschaftliche Artikel, Lehrbücher, Mitschriften und digitale Ressourcen nicht nur rezipieren, sondern aktiv verarbeiten, annotieren und in ihre individuelle Wissensstruktur integrieren (vgl. Rouet und Britt 2011, S. 8). Diese Dokumentenarbeit umfasst kognitive Prozesse wie das Extrahieren relevanter Informationen, das Identifizieren von Zusammenhängen zwischen verschiedenen Quellen sowie die Synthese zu kohärenten Wissensstrukturen.

Die Qualität dieser Dokumentenverarbeitung beeinflusst maßgeblich den Lernerfolg. Während oberflächliche Lesestrategien lediglich zur Reproduktion isolierter Fakten führen, ermöglichen tiefenorientierte Ansätze das Verstehen übergreifender Konzepte und die Anwendung des Gelernten in neuen Kontexten (vgl. Säljö 1984, S. 65).

2.1.2. Digitale Transformation des Lernens

Die Digitalisierung hat das Selbststudium fundamental verändert. Während Studierende früher primär auf physische Bibliotheksbestände angewiesen waren, ermöglichen digitale Repositorien, E-Learning-Plattformen und Open-Access-Datenbanken heute einen nahezu unbegrenzten Zugang zu Fachliteratur (vgl. Kerres 2018, S. 79). Die Digitalisierung bringt jedoch auch neue Herausforderungen mit sich. Eppler und Mengis beschreiben das Phänomen der Information Overload, bei dem die verfügbare Informationsmenge die Verarbeitungskapazität der Lernenden übersteigt(vgl. Eppler und Mengis 2004, S. 330).

Parallel dazu haben digitale Werkzeuge die Verwaltung von Lernmaterialien erheblich erleichtert. Cloud-Dienste wie Google Drive, Dropbox oder OneDrive ermöglichen einen orts- und geräteunabhängigen Zugriff auf Studienunterlagen. Trotz dieser technischen Fortschritte bleibt die inhaltliche Durchdringung umfangreicher Dokumentensammlungen eine zeitaufwendige Herausforderung. Traditionelle Suchfunktionen, die lediglich nach Stichworten oder Dateinamen filtern, stoßen dabei schnell an ihre Grenzen.

2.1.3. Herausforderungen der Informationsverarbeitung im Studium

Empirische Untersuchungen zeigen, dass viele Studierende Schwierigkeiten haben, Selbstlernphasen effektiv zu gestalten, insbesondere wenn umfangreiche Literaturmengen zu bewältigen sind (vgl. Schmitz und Wiese 2006, S. 62). Bei der Informationsverarbeitung im Selbststudium zeigen sich dabei mehrere zentrale Herausforderungen.

Eine grundlegende Problematik liegt in der fehlenden systematischen Dokumentenverwaltung. Viele Studierende verfügen über keine konsistente Strategie zur Organisation ihrer Lernmaterialien, was zu fragmentierten Wissensbeständen führt (vgl. Kitsantas, Winsler und Huie 2008, S. 46). Verschiedene Dokumentenformate müssen nicht nur inhaltlich erschlossen, sondern auch so abgelegt werden, dass sie später wiedergefunden werden können. Hinzu kommt die begrenzte Leistungsfähigkeit schlüsselwortbasierter Suche in persönlichen Dokumentensammlungen, da relevante Informationen häufig unter verschiedenen Begrifflichkeiten abgelegt sind.

Die Synthese von Informationen aus multiplen Quellen stellt eine zusätzliche kognitive Belastung dar. Studierende müssen nicht nur einzelne Dokumente verstehen, sondern Verbindungen zwischen ihnen herstellen, Widersprüche identifizieren und übergreifende Zusammenhänge erkennen. Diese Anforderung entspricht der von Britt und Rouet (vgl. Britt und Rouet 2012, S. 22) beschriebenen Kompetenz des Multiple Document

Literacy, die jedoch unterschiedlich stark ausgeprägt ist und deren Entwicklung im Selbststudium besondere Unterstützung erfordert.

Hier setzt der Einsatz KI-gestützter Dokumentenassistenten an. Die Kombination aus semantischer Suche und sprachgenerativen Modellen ermöglicht einen schnelleren Zugriff auf relevante Informationen, das Erschließen von Zusammenhängen und die Generierung personalisierter Lernmaterialien, ohne dass Studierende die Datenkontrolle abgeben müssen.

2.2. Information Retrieval und semantische Suche

2.2.1. Grundlagen des Information Retrieval

Information Retrieval bezeichnet den Prozess des Auffindens relevanter Informationen aus einer Dokumentensammlung als Antwort auf eine Benutzeranfrage (vgl. Manning 2009, S. 324). Klassische IR-Systeme basieren auf dem Prinzip des lexikalischen Matchings. Dokumente werden anhand der Übereinstimmung zwischen Suchanfrage und Dokumenteninhalt auf Wort- oder Zeichenebene bewertet.

Ein zentrales Konzept stellt das Vektorraummodell dar (vgl. Salton, Wong und Yang 1975, S. 613), das Dokumente und Suchanfragen als Vektoren in einem hochdimensionalen Raum repräsentiert. Die Dimensionen entsprechen dabei den Termen des Vokabulars, und die Relevanzbewertung erfolgt über Ähnlichkeitsmaße wie die Kosinusähnlichkeit. Für die Gewichtung einzelner Terme hat sich insbesondere das Verfahren Term Frequency–Inverse Document Frequency (TF-IDF) etabliert, das sowohl die Häufigkeit eines Terms innerhalb eines Dokuments als auch seine Seltenheit im Gesamtkorpus berücksichtigt.

Trotz ihrer weiten Verbreitung weisen lexikalische Ansätze grundlegende Limitationen auf. Das Vocabulary Mismatch Problem beschreibt die Situation, dass Nutzer und Dokumentenautoren unterschiedliche Begriffe für dasselbe Konzept verwenden (vgl. Furnas u. a. 1987, S. 964). Synonymie und Polysemie führen dazu, dass relevante Dokumente nicht gefunden werden oder irrelevante Treffer die Ergebnisliste dominieren.

2.2.2. Von lexikalischer zu semantischer Suche

Semantische Suchsysteme adressieren diese Limitationen, indem sie nicht primär auf Zeichenketten, sondern auf Bedeutungsebene operieren. Die Grundidee besteht darin, die konzeptuelle Ähnlichkeit zwischen Anfrage und Dokument zu erfassen, unabhängig

von der konkreten lexikalischen Realisierung (vgl. Bast, Buchhold und Haussmann 2016, S. 1).

Frühe Ansätze semantischer Suche nutzten Wissensrepräsentationen wie Ontologien oder Thesauri, um Synonymrelationen und Hierarchien zu modellieren . Mit dem Aufkommen des maschinellen Lernens entwickelten sich datengetriebene Verfahren, die semantische Ähnlichkeit direkt aus großen Textkorpora lernen. Latent Semantic Analysis (vgl. Deerwester u. a. 1990, 1ff.) war einer der ersten Ansätze, der mittels Dimensionsreduktion latente semantische Strukturen aufdeckte.

Den Durchbruch in der semantischen Suche markierten neuronale Sprachmodelle. Word2Vec vgk. Mikolov u. a. 2013, S. 2 zeigte, dass sich distributionelle Semantik in dichten Vektorrepräsentationen abbilden lässt. Das Prinzip beruht auf der Annahme, dass Wörter mit ähnlicher Bedeutung in ähnlichen Kontexten auftreten. Die resultierenden Embeddings ermöglichen es, semantische Ähnlichkeit als geometrische Nähe im Vektorraum zu berechnen.

2.2.3. Vektorrepräsentationen und Embeddings

Moderne semantische Suchsysteme nutzen Sentence- und Document-Embeddings, um ganze Texteinheiten als hochdimensionale Vektoren zu repräsentieren. Im Gegensatz zu lexikalischen Verfahren wie TF-IDF oder BM25, die auf Wortübereinstimmungen basieren, erfassen Embeddings die semantische Bedeutung von Sätzen oder Absätzen unter Berücksichtigung ihres Kontexts. Dadurch können auch inhaltlich verwandte Texte identifiziert werden, selbst wenn sie unterschiedliche Begriffe verwenden (vgl. Reimers und Gurevych 2019, S. 12).

Embedding-Modelle basieren typischerweise auf Transformer-Architekturen und werden auf Ähnlichkeitsaufgaben trainiert, sodass die Kosinusähnlichkeit zwischen Vektoren mit der semantischen Ähnlichkeit der Texte korreliert (vgl. Reimers und Gurevych 2019, S. 12). Auf diese Weise lassen sich Synonyme, Paraphrasen und kontextuelle Bedeutung erfassen.

Für die effiziente Suche in großen Embedding-Räumen werden Vektordatenbanken eingesetzt. Diese unterstützen Ähnlichkeitssuche anhand von Distanzmaßen sowie das Filtern von Ergebnissen über Metadaten. In dieser Arbeit wird Qdrant[1] verwendet, eine Open-Source-Vektordatenbank mit HNSW-Indexierung, die eine performante und lokal betreibbare Ähnlichkeitssuche ermöglicht. Die HNSW-Indexierung basiert auf einem graphbasierten Algorithmus, der durch ein hierarchisches Small World Netzwerk

[1] QDrant: `https://qdrant.tech/`

über Vektoren eine schnelle approximate Nearest Neighbor Suche in hochdimensionalen Daten ermöglicht.

Damit können Nutzer Suchanfragen in natürlicher Sprache formulieren, etwa: *„Welche Mitschriften behandeln statistische Signifikanz?“*. Das System identifiziert semantisch passende Textpassagen, unabhängig von der konkreten Formulierung in den Dokumenten. Metadatenfilter erlauben zusätzlich eine gezielte Eingrenzung der Ergebnisse.

2.3. Natural Language Processing

2.3.1. Definition Natural Language Processing

Natural Language Processing (NLP) bezeichnet die automatische Verarbeitung, Analyse und Generierung menschlicher Sprache durch Computer (vgl. Jurafsky und Martin 2013, S. 12). Die zentrale Herausforderung ergibt sich aus den Eigenschaften natürlicher Sprache selbst. Wörter und Sätze können mehrdeutig sein, ihre Bedeutung hängt stark vom Kontext ab, vieles bleibt implizit und kulturelle sowie situative Faktoren spielen eine wichtige Rolle. Diese Komplexität macht die maschinelle Sprachverarbeitung zu einem anspruchsvollen Unterfangen (vgl. Manning und Schütze 1999, S. 58). Während formale Sprachen wie Programmiercode eindeutig und präzise sind, zeichnet sich natürliche Sprache durch Flexibilität, Mehrdeutigkeit und kreative Ausdrucksmöglichkeiten aus.

Im akademischen Kontext ermöglicht NLP vielfältige Anwendungen, die den Studienprozess unterstützen. Semantische Suchsysteme durchsuchen wissenschaftliche Datenbanken und identifizieren relevante Publikationen basierend auf inhaltlicher Ähnlichkeit statt reiner Schlüsselwortübereinstimmung. Automatische Textzusammenfassungen kondensieren umfangreiche Forschungsartikel auf ihre Kernaussagen. Informationsextraktionssysteme identifizieren Entitäten, Relationen und Fakten in großen Textmengen. Intelligente Frage-Antwort-Systeme beantworten Nutzerfragen unter Rückgriff auf Wissensdatenbanken. Für einen Dokumentenassistenten im Selbststudium sind insbesondere drei NLP-Fähigkeiten zentral. Diese umfassen das semantische Verstehen von Nutzeranfragen, das präzise Auffinden relevanter Informationen in persönlichen Dokumentensammlungen sowie die natürlichsprachliche, kontextgerechte Aufbereitung dieser Informationen.

2.3.2. Fundamentale NLP Aufgaben

Die Tokenisierung stellt eine zentrale Vorverarbeitungsstufe in NLP-Systemen dar. Sie zerlegt Text in Einheiten, die von Modellen verarbeitet werden können (vgl. Webster

und Kit 1992, 1ff.). In der Praxis ist diese Aufgabe komplex, da Satzzeichen, Sprachen ohne Wortgrenzen sowie moderne Textformen wie Emojis oder Hashtags berücksichtigt werden müssen.

Moderne Sprachmodelle verwenden überwiegend Subword-Tokenisierung, bei der häufige Wörter als Ganzes und seltene Wörter in kleinere Einheiten zerlegt werden (vgl. Sennrich, Haddow und Birch 2016, 3ff.). Verfahren wie Byte-Pair Encoding oder WordPiece erzeugen dadurch ein kompaktes Vokabular, das die Generalisierung auf unbekannte Begriffe erleichtert(vgl. Kudo und Richardson 2018, 3ff.).

Weitere zentrale NLP-Aufgaben sind Textklassifikation, Sprachmodellierung und Named Entity Recognition (NER). Textklassifikation ordnet Texte automatisch Kategorien zu (vgl. Sebastiani 2002, 3ff.) und kann in einem Dokumentenassistenten zur Organisation oder Relevanzbewertung genutzt werden. Sprachmodelle lernen die Wahrscheinlichkeit von Wortfolgen und bilden die Grundlage moderner generativer Systeme. NER identifiziert Entitäten wie Personen, Organisationen oder Orte und unterstützt so das gezielte Auffinden relevanter Konzepte (vgl. Nadeau und Sekine 2007, S. 1).

Diese Aufgaben bilden die Basis moderner Sprachverarbeitung. Während sie früher durch spezialisierte Modelle gelöst wurden, können Large Language Models heute mehrere Aufgaben innerhalb eines einheitlichen Frameworks bewältigen und über Prompts gesteuert werden. Damit hat sich der Fokus von separaten Modulen hin zu universellen Foundation Models verschoben, die als Grundlage vielfältiger Anwendungen dienen.

2.4. Grundlagen von Large Language Models

2.4.1. Architektur und Funktionsweise von Transformer-Modellen

Large Language Models basieren auf der Transformer-Architektur, die von (vgl. Vaswani u. a. 2017, S. 22) eingeführt wurde und das Paradigma der Sequenzverarbeitung in der natürlichen Sprachverarbeitung revolutionierte. Im Gegensatz zu rekurrenten neuronalen Netzen, die Sequenzen schrittweise verarbeiten, ermöglicht der Transformer eine parallele Verarbeitung durch den Self-Attention-Mechanismus.

Self-Attention berechnet für jedes Token einer Eingabesequenz einen gewichteten Durchschnitt aller anderen Token, wobei die Gewichte die Relevanz anderer Positionen für die Repräsentation des aktuellen Tokens widerspiegeln. Diese Mechanismus erlaubt es dem Modell, langreichweitige Abhängigkeiten zu erfassen, ohne unter dem Vanishing-Gradient-Problem zu leiden, das rekurrente Architekturen limitiert.

Moderne Large Language Models nutzen den Decoder-Teil der ursprünglichen Transformer Architektur und werden auf die Aufgabe der Next-Token Prediction trainiert, bei der das Modell auf Basis einer gegebenen Token-Sequenz das wahrscheinlichste nächste Token vorhersagt (vgl. Radford u. a. 2019, S. 22). Obwohl dieses Trainingsziel einfach erscheint, ermöglicht das Training auf sehr großen Textkorpora das Erlernen komplexer sprachlicher Strukturen und semantischer Zusammenhänge.

Die zunehmende Skalierung von Sprachmodellen, sowohl in Bezug auf die Anzahl der Parameter als auch auf den Umfang der Trainingsdaten, führt zu sogenannten emergenten Fähigkeiten, die bei kleineren Modellen nicht auftreten. Ab einer bestimmten Modellgröße zeigen Large Language Models Verhaltensweisen wie Few-Shot-Learning, Chain-of-Thought-Reasoning oder robuste Instruktionsbefolgung, ohne dass sie dafür explizit trainiert wurden.

2.4.2. Open Source und propritäre Modelle

Die Landschaft der LLMs ist zweigeteilt in proprietäre Systeme, die über APIs zugänglich sind, und Open-Source-Modelle, die frei verfügbar sind. Proprietäre Modelle wie GPT-5 (OpenAI[2]), Claude (Anthropic[3]) oder Gemini (Google[4]) werden von Technologieunternehmen entwickelt und typischerweise als Cloud-Dienst angeboten. Sie zeichnen sich durch hohe Leistungsfähigkeit aus, erfordern jedoch die Übertragung sensibler Daten an Drittanbieter. Open-Source-Modelle wie Llama[5], Mistral[6] oder Qwen[7] werden unter permissiven Lizenzen veröffentlicht und können lokal betrieben werden. Die Community getriebene Entwicklung hat zu einer rapiden Verbesserung dieser Modelle geführt, sodass sie für viele Anwendungsfälle mit proprietären Alternativen konkurrieren können.

Für den Kontext eines selbst-hostbaren Dokumentenassistenten sind Open-Source-Modelle besonders geeignet, da sie vollständige Kontrolle über die Daten ermöglichen, Abhängigkeiten von externen Diensten vermeiden, Anpassungen an spezifische Anforderungen erlauben und bei lokaler Ausführung keine nutzungsabhängigen Kosten verursachen. Die Verfügbarkeit quantisierter Modellversionen (z.B. durch GGUF-Format) ermöglicht zudem den Betrieb auf Consumer-Hardware mit begrenzten Ressourcen (vgl. Dettmers u. a. 2023, 16ff.).

[2] OpenAI: `https://openai.com/de-DE/`
[3] Claude: `https://claude.ai/`
[4] Gemini: `https://gemini.google.com/`
[5] Llama: `https://www.llama.com/`
[6] Mistral: :https://mistral.ai/
[7] Gwen: `https://chat.qwen.ai/`

2.4.3. Modelloptimierung - Quantisierung

Large Language Models sind aufgrund ihrer hohen Parameteranzahl sehr speicher- und rechenintensiv. Ein Modell mit mehreren Milliarden Parametern benötigt in 16-Bit-Präzision schnell mehrere zehn Gigabyte an Arbeitsswpeicher und erfordert spezialisierte Hardware für die Inferenz. Für den Einsatz in selbst-hostbaren Anwendungen stellt dies eine wesentliche Hürde dar. Um den Ressourcenbedarf zu reduzieren, kommen Verfahren zur Modellkomprimierung zum Einsatz, die die Modellgröße verkleinern, ohne die Leistungsfähigkeit stark zu beeinträchtigen (vgl. Ganesh u. a. 2021, 1ff.).

Ein zentrales Verfahren ist die **Quantisierung**. Dabei wird die numerische Genauigkeit der Modellgewichte reduziert, beispielsweise von 16-Bit auf 8-, 4- oder sogar 2-Bit-Darstellung. Durch die geringere Präzision sinkt der Speicherbedarf proportional, und Matrixmultiplikationen können effizienter ausgeführt werden. Moderne Quantisierungsverfahren, wie GPTQ oder 4-bit NormalFloat, berücksichtigen zusätzlich die statistischen Eigenschaften der Gewichtsverteilung und können Qualitätsverluste minimieren (vgl. Frantar u. a. 2023, 1ff). In der Praxis ermöglicht Quantisierung, dass selbst große Modelle wie LLaMA oder Mistral auf Consumer-Hardware betrieben werden können, beispielsweise im kompakten GGUF-Format, das speziell für ressourcenlimitierte Umgebungen optimiert ist.

Neben der Quantisierung existieren weitere Komprimierungstechniken wie Pruning, bei dem wenig relevante Gewichte entfernt werden, oder Low-Rank-Adaptation, bei der nur kleine, zusätzlich trainierte Gewichte gespeichert werden müssen (vgl. Hu u. a. 2021, 2ff.). Beide Ansätze reduzieren die Anzahl aktiver Parameter oder die Menge gespeicherter Gewichte und ermöglichen dadurch speichereffiziente Anpassungen von Modellen.

Durch Quantisierung und Komprimierung können LLMs mit begrenzter Hardware betrieben werden, ohne dass erneut Trainingsaufwand entsteht. Dies ist insbesondere für lokal ausgeführte Systeme wie einen selbst-hostbaren Dokumentenassistenten relevant, da geringe Speicheranforderungen, schnelle Inferenzzeiten und Unabhängigkeit von GPU-Ressourcen wesentliche Anforderungen darstellen.

2.5. Retrieval Augmented Generation

2.5.1. Konzept und Motivation des Rag Ansatzes

Retrieval-Augmented Generation bezeichnet einen Ansatz, der Information Retrieval mit generativen Sprachmodellen kombiniert (vgl. Lewis u. a. 2021, S. 1). Anstatt

ausschließlich auf das in den Modellgewichten gespeicherte Wissen zurückzugreifen, erhält das Sprachmodell vor der Antwortgenerierung Zugriff auf externe Wissensquellen, aus denen kontextrelevante Informationen abgerufen werden.

Die Motivation für RAG ergibt sich aus den Grenzen rein parametrischer Sprachmodelle. Das in einem LLM gespeicherte Wissen ist zum Trainingszeitpunkt fixiert, wodurch neue oder domänenspezifische Informationen nicht berücksichtigt werden können. Zudem ist es kaum möglich, persönliches oder kontextabhängiges Wissen in die Modellgewichte zu integrieren, ohne das allgemeine Sprachverständnis zu beeinträchtigen. Schließlich neigen solche Modelle zu Halluzinationen, da sie keine Möglichkeit besitzen, Aussagen anhand externer Quellen zu überprüfen (vgl. Shuster u. a. 2021, 3ff).

RAG adressiert diese Einschränkungen, indem Wissen nicht im Modell selbst gespeichert wird. Stattdessen werden relevante Dokumente oder Textpassagen zur Laufzeit aus einer Wissensbasis abgerufen und dem Modell als Kontext bereitgestellt. Auf diese Weise lassen sich aktuelle, domänenspezifische und überprüfbare Antworten generieren, wobei keine Anpassung der Modellgewichte notwendig ist.

2.5.2. Abgrenzung zu reinen Fine Tuning

Eine alternative Strategie zur Integration domänenspezifischen Wissens ist das Fine-Tuning, bei dem ein vortrainiertes Modell auf fachspezifischen Daten weitertrainiert wird. Während beide Ansätze ihre Berechtigung haben, unterscheiden sie sich fundamental in ihren Eigenschaften und Anwendungsszenarien. Fine-Tuning internalisiert Wissen in den Modellgewichten, was zu schnelleren Inferenzzeiten führt, da kein Retrieval-Schritt erforderlich ist. Allerdings erfordert Fine-Tuning erhebliche Rechenressourcen, kuratierte Trainingsdaten und birgt das Risiko des Catastrophic Forgetting, bei dem zuvor gelerntes Wissen überschrieben wird (vgl. Kirkpatrick u. a. 2017, S. 1). Zudem sind gefinetunte Modelle statisch, da jede Aktualisierung des Wissensstands ein erneutes Training erfordert.

RAG ermöglicht den dynamischen Zugriff auf eine erweiterbare Wissensbasis, in die neue Dokumente jederzeit integriert werden können, ohne das Modell neu zu trainieren. Dies ist besonders im Kontext eines Dokumentenassistenten für Studierende relevant, da sich die persönliche Dokumentensammlung kontinuierlich erweitert. Zudem ermöglicht RAG eine transparente Darstellung der Wissensquellen. Nutzer können nachvollziehen, auf welchen Dokumenten eine Antwort basiert, was das Vertrauen erhöht und ein kritisches Prüfen erleichtert. (vgl. Gao u. a. 2024, S. 5). Hybridansätze, die Fine-Tuning für domänenspezifische Sprachstile mit RAG für faktisches Wissen

kombinieren, stellen eine vielversprechende Entwicklungsrichtung dar (vgl. Ovadia u. a. 2024, S. 7).

2.5.3. Technische Komponenten eines RAG-Systems

Ein RAG-System besteht aus mehreren Komponenten, die in einer Pipeline zusammenwirken. Die Dokumentenverarbeitung umfasst das Einlesen verschiedener Dateiformate, die Extraktion der Textinhalte sowie die Segmentierung in semantisch kohärente Einheiten (Chunks). Die Wahl der Chunk-Größe ist dabei ein zentraler Hyperparameter. Zu kleine Chunks können wichtigen Kontext verlieren, während zu große Chunks irrelevante Informationen enthalten (vgl. Lewis u. a. 2021, S. 4).

Anschließend werden die Textabschnitte mithilfe eines Embedding-Modells in Vektorrepräsentationen überführt und in einer Vektordatenbank gespeichert. Ergänzende Metadaten wie Dokumententitel, Erstellungsdatum oder Kategorisierung ermöglichen eine spätere Filterung der Ergebnisse.

Bei einer Nutzeranfrage erfolgt zunächst das Retrieval. Die Anfrage wird ebenfalls in einen Embedding-Vektor transformiert, um die semantisch ähnlichsten Chunks in der Vektordatenbank zu identifizieren. Erweiterte Retrieval-Verfahren wie Reranking, bei dem ein spezialisiertes Modell die Relevanz der Kandidaten feingranular bewertet, oder Hybrid Search, die semantische mit lexikalischer Suche kombiniert, können die Ergebnisqualität weiter verbessern (vgl. Thakur u. a. 2021, S. 3).

Die abgerufenen Chunks werden anschließend mit der Nutzeranfrage zu einem Prompt zusammengeführt, der dem LLM übergeben wird. Dieses generiert eine Antwort unter Berücksichtigung seines internen Wissens sowie der bereitgestellten Kontextinformationen. Eine sorgfältige Gestaltung des Prompts ist dabei entscheidend, da das Modell angewiesen werden sollte, sich primär auf die bereitgestellten Quellen zu stützen und kenntlich zu machen, wenn keine relevanten Informationen vorliegen.

2.6. Evaluationsmethoden von RAG-Systemen

2.6.1. Motivation und Herausforderungen

Die Evaluation von Retrieval-Augmented Generation (RAG)-Systemen-Systemen unterscheidet sich grundlegend von klassischen Verfahren im Information Retrieval oder der Textklassifikation. RAG kombiniert das Auffinden relevanter Informationen mit der Generierung von Antworten, sodass nicht nur die Qualität der Treffer, sondern auch deren Nutzung im Antworttext bewertet werden muss. Klassische Kennzahlen

aus dem Information Retrieval erfassen dieses Zusammenspiel nur unzureichend, da sie primär auf Dokument- oder Labelebene operieren.

Ein zentrales Problem ist das Auftreten sogenannter Halluzinationen, also die Erzeugung plausibler, jedoch faktisch falscher Aussagen durch Sprachmodelle, selbst bei Vorliegen geeigneter Quellen. Daher muss die Evaluation prüfen, ob eine Antwort sowohl inhaltlich korrekt (Factual Correctness) als auch durch den verwendeten Kontext gestützt ist (Faithfulness). Zusätzlich stellt sich die Frage, inwieweit das Modell die bereitgestellten Informationen vollständig und sachgerecht nutzt.

Menschliche Bewertungen bleiben trotz klarer Kriterien subjektiv und können durch Lesbarkeit oder Stil beeinflusst werden. Automatisierte Bewertungsverfahren reduzieren diese Einflüsse, sind jedoch selbst anfällig für Modell-Bias. Eine belastbare Beurteilung von RAG-Systemen erfordert daher End-to-End-Evaluationen, die Retrieval und Generierung gemeinsam betrachten und sowohl Kontexttreue als auch inhaltliche Genauigkeit berücksichtigen.

2.6.2. Golden Standard Datasets

Für die Evaluation von RAG-Systemen ist die Verfügbarkeit geeigneter Referenzdaten entscheidend. Golden Standard Datasets bestehen aus validierten Frage-Antwort-Paaren sowie den zugehörigen Kontextdokumenten und dienen als objektiver Maßstab zur Bewertung der Antwortqualität. Sie ermöglichen den Vergleich verschiedener Systemkonfigurationen unter identischen Bedingungen und bilden damit die Grundlage für eine reproduzierbare Leistungsbewertung (vgl. Kraus u. a. 2025, S. 5).

Die Erstellung solcher Datensätze erfordert jedoch erheblichen Aufwand. Referenzantworten müssen fachlich korrekt, eindeutig und vollständig formuliert werden, um als zuverlässige Vergleichsgrundlage zu dienen. Darüber hinaus sollten die enthaltenen Fragen und Dokumente die inhaltliche Vielfalt und Komplexität des Anwendungskontextes widerspiegeln, etwa unterschiedliche Themenbereiche oder Dokumenttypen.

Trotz des hohen Aufwands und einer gewissen verbleibenden Subjektivität bei der Formulierung und Bewertung von Antworten stellen Golden Standard Datasets ein unverzichtbares Instrument für die Evaluation von RAG-Systemen dar. Sie gewährleisten eine transparente, vergleichbare und methodisch belastbare Beurteilung der Systemleistung und bilden die Basis für datengetriebene Optimierungsentscheidungen, beispielsweise bei der Auswahl von Retrieval- oder Chunking-Strategien.

2.6.3. Frameworks zur quantitativen Evaluation

Für die objektive Bewertung von Retrieval-Augmented-Generation-Systemen (RAG) werden zunehmend spezialisierte Frameworks eingesetzt, die eine quantitative und reproduzierbare Analyse der Systemleistung ermöglichen. Diese Werkzeuge automatisieren die Bewertung generierter Antworten und berechnen standardisierte Qualitätsmetriken. Dadurch lassen sich die Auswirkungen einzelner Pipeline-Komponenten, wie etwa Retriever, Embeddings oder Chunking-Strategien, systematisch erfassen, ohne auf subjektive Einschätzungen angewiesen zu sein.

Ein etabliertes Open-Source-Framework ist RAGAS[8]. Es wurde speziell für die Evaluierung von RAG-Systemen entwickelt und kombiniert klassische Bewertungsmethoden mit den Fähigkeiten moderner Sprachmodelle, Antworten auf semantischer Ebene zu analysieren. RAGAS vergleicht generierte Antworten mit Referenzantworten und prüft zugleich, inwieweit diese durch die zugrunde liegenden Kontextpassagen gestützt werden. Die einzelnen Qualitätsmetriken sind in Tabelle 2.1 dargestellt und werden als Scores zwischen 0 und 1 ausgegeben. Sie ermöglichen damit eine präzise und reproduzierbare quantitative Bewertung der Systemleistung.

Tabelle 2.1.: Übersicht der von RAGAS bereitgestellten Qualitätsmetriken

Metrik	**Beschreibung**
Faithfulness	Bewertet, ob die generierte Antwort durch die bereitgestellten Kontextpassagen gestützt wird und keine Fakten halluziniert.
Context Precision	Misst, wie relevant die vom Retriever ausgewählten Kontextpassagen für die gestellte Frage sind.
Factual Correctness	Prüft, ob die generierte Antwort faktisch korrekt ist, unabhängig von ihrer Formulierung.
Semantic Similarity	Bestimmt die semantische Übereinstimmung zwischen der generierten Antwort und der Referenzantwort.

[8]RAGAS: `https://docs.ragas.io/en/stable/getstarted/`

3. Ausgangssituation und Stand der Forschung

3.1. Stand der Forschung

3.1.1. Digitalisierung des Selbststudiums

Die Digitalisierung hat die Hochschullehre in den vergangenen Jahren deutlich verändert und zu einer neuen Form des Lernens geführt. Lernplattformen, aufgezeichnete Vorlesungen und digital bereitgestellte Materialien haben die Wissensvermittlung nachhaltig geprägt. Hochschulen entwickeln sich zunehmend hin zu technologiegestützten Lernformaten, die den Zugang zu Lehrinhalten unabhängig von Zeit und Ort ermöglichen. Dieser Wandel ist nicht nur technisch, sondern auch didaktisch begründet. Studierende sollen mehr Verantwortung für ihren Lernprozess übernehmen und Lernphasen eigenständig strukturieren (vgl. Zawacki-Richter u. a. 2020, 5ff.).

Mit der Digitalisierung verändert sich das Selbststudium grundlegend. Lernmanagementsysteme wie Moodle oder ILIAS bieten zwar einen kontinuierlichen Zugriff auf Studienmaterialien, unterstützen jedoch die aktive Wissenserschließung nur begrenzt (vgl. Jenert und Brahm 2021, 54ff.). Zugleich verlagert sich die Wissensvermittlung zunehmend aus den Präsenzveranstaltungen heraus, sodass das Lernen stärker asynchron und eigenverantwortlich erfolgt.

Die COVID-19-Pandemie hat die Digitalisierung des Selbststudiums erheblich beschleunigt und digitale Lernformen zur Regel gemacht (vgl Bond u. a. 2021, 14ff.). Studierende organisieren ihre Lernphasen heute weitgehend flexibel und eigenständig, während sich ihre Materialien über verschiedene Plattformen und Formate verteilen. Dadurch steigt die Komplexität des individuellen Wissensmanagements. Der Lernerfolg hängt zunehmend davon ab, wie effizient Studierende relevante Informationen im digitalen Raum finden, auswählen und strukturieren können.

3.1.2. Informationsfragmentierung im Lernprozess

Mit der zunehmenden Digitalisierung wächst nicht nur die Menge an Lernmaterialien, sondern auch deren Verteilung über unterschiedliche Plattformen. Vorlesungsunterlagen befinden sich in Lernmanagementsystemen, zusätzliche Dateien in Cloud-Speichern,

Literatur in Referenzverwaltungen und persönliche Notizen in individuellen Tools. Empirische Untersuchungen zeigen, dass Studierende im Durchschnitt fünf bis sieben verschiedene digitale Werkzeuge parallel nutzen (vgl. Head und Eisenberg 2010, 5ff.). Dadurch entsteht ein fragmentiertes Informationsumfeld ohne zentrale Struktur oder übergreifenden Wissensraum.

Diese Fragmentierung erschwert das Wiederfinden von Informationen erheblich. Studierende müssen häufig wissen, in welchem System eine Datei gespeichert wurde, bevor sie danach suchen können. Ohne übergreifende Struktur bleibt der Zugriff stark von Erinnerung und individuellen Ablagestrategien abhängig. Studien weisen darauf hin, dass die Effizienz des Wiederfindens weniger von der Datenmenge, sondern von der Fähigkeit abhängt, mentale Modelle der eigenen Ablagesysteme zu entwickeln (vgl. Bergman u. a. 2010, 7ff.).

Hinzu kommt, dass viele Systeme ausschließlich auf lexikalischer Suche basieren und semantische Zusammenhänge unberücksichtigt lassen. Dieses sogenannte Vocabulary Mismatch (vgl. Manning, Raghavan und Schütze 2008, 7ff.) führt dazu, dass relevante Dokumente übersehen werden, wenn sie andere Formulierungen oder Synonyme enthalten. Für Studierende bedeutet dies einen erhöhten Suchaufwand und erschwerte inhaltliche Verknüpfung von Lernmaterialien – ein Problem, das semantische Suchverfahren wie RAG gezielt adressieren.

3.2. Technische Dimension

Die beschriebenen Herausforderungen beim Auffinden relevanter Lerninformationen lassen sich vor allem auf technische Einschränkungen bestehender Suchsysteme zurückführen. Die in Hochschulumgebungen verbreiteten Werkzeuge wie Lernmanagementsysteme, Cloudspeicher oder Referenzverwaltungssysteme basieren auf lexikalischen Retrievalmethoden, die den Text als Folge einzelner Wörter behandeln. Suchanfragen werden mit Dokumentinhalten über exakte Wortübereinstimmung oder statistische Gewichtung (z. B. TF-IDF, BM25) abgeglichen. Eine Anfrage führt daher nur dann zu Treffern, wenn die gesuchten Begriffe exakt im Dokument vorkommen. Dieses Prinzip erzeugt das bekannte Vocabulary-Mismatch-Problem: inhaltlich relevante Dokumente bleiben unentdeckt, wenn sie andere Formulierungen, Synonyme oder domänenspezifische Varianten verwenden.

Darüber hinaus verfügen klassische Suchsysteme über keine semantische Repräsentation von Inhalten. Sie erkennen nicht, welche Bedeutung ein Begriff im Kontext trägt oder wie verschiedene Dokumente thematisch zusammenhängen. Damit bleibt der Zugriff auf Informationen oberflächlich und kontextlos. Für Lernprozesse, die inhaltliches

Verstehen und den Vergleich mehrerer Quellen erfordern, ist dies ein gravierender Nachteil.

Zugleich stehen moderne KI-Modelle grundsätzlich bereit, um Sprache semantisch zu analysieren und kontextbezogene Antworten zu erzeugen. Ihr Einsatz ist jedoch im Hochschulkontext häufig eingeschränkt, da viele Systeme proprietär und cloudbasiert sind. Die Verarbeitung personenbezogener oder urheberrechtlich geschützter Lernmaterialien durch externe Anbieter widerspricht den Anforderungen an datenschutzkonforme Forschungs- und Lehrumgebungen. Damit entsteht eine technische und infrastrukturelle Lücke. Leistungsfähige semantische Methoden existieren, können jedoch nicht in bestehende Hochschulsysteme integriert werden.

3.2.1. Forschungslücke

Trotz erheblicher Fortschritte im Bereich der semantischen Suche und RAG existieren bislang kaum Ansätze, die diese Technologien gezielt für den Bildungskontext einsetzen. Der aktuelle Forschungsstand konzentriert sich überwiegend auf wissensintensive Anwendungen in Unternehmen, Support-Systemen oder Forschungseinrichtungen (vgl. Lewis u. a. 2021, 1ff). In Hochschulumgebungen hingegen liegt der Schwerpunkt digitaler Werkzeuge weiterhin auf der Bereitstellung und Verwaltung von Lernmaterialien, nicht auf deren semantischer Erschließung oder aktiven Nutzung zur Lernunterstützung.

Es fehlt insbesondere an Systemen, die Studierenden ermöglichen, eigene Dokumente wie Vorlesungsfolien, Skripte oder Notizen lokal zu speichern und inhaltlich auszuwerten. Bestehende KI Assistenten, die dies theoretisch leisten könnten, sind fast ausschließlich proprietär und cloudbasiert. Ihre Verarbeitungsprozesse sind intransparent und stehen im Widerspruch zu den Anforderungen an Datenschutz, Datenhoheit und Urheberrecht in Bildungseinrichtungen.

Obwohl in der Hochschulforschung seit Jahren betont wird, dass Studierende beim Umgang mit digitalen Informationen unterstützt werden müssen, fehlt bislang eine experimentelle geprüfte Lösung, die moderne KI-Methoden zur semantischen Erschließung von Lernmaterialien nutzt. Entsprechend existieren weder offene, datenschutzkonforme RAG-Systeme für persönliche Lernressourcen noch Studien, die deren Konfiguration systematisch vergleichen. Diese Arbeit adressiert diese Lücke, indem sie ein prototypisches Open-Source-System entwickelt und unterschiedliche Pipeline-Varianten empirisch evaluiert.

3.3. Relevanz der Arbeit

3.3.1. Wissenschaftliche Relevanz

Die wissenschaftliche Relevanz dieser Arbeit liegt in ihrem Beitrag zur Forschung über semantische Suchverfahren und RAG. Während große Sprachmodelle intensiv erforscht werden, fehlt es insbesondere im Hochschulkontext an systematischen Untersuchungen, wie sich verschiedene Komponenten einer RAG Pipeline wie Retriever, Chunkingstrategien und Embeddingmodelle auf die Antwortqualität auswirken. Bestehende Arbeiten konzentrieren sich meist auf industrielle Anwendungen wie Wissensdatenbanken oder Support-Systeme, nicht jedoch auf akademische Lernumgebungen.

Diese Arbeit schließt diese Lücke, indem sie verschiedene Pipeline-Varianten implementiert und anhand eines Golden-Standard-Datensatzes mit dem Evaluationsframework RAGAS bewertet. Durch die Verwendung objektiver Qualitätsmetriken wie Faithfulness und Relevance werden reproduzierbare Ergebnisse erzielt, die den Einfluss einzelner Pipeline-Komponenten auf Antwortqualität und Retrieval-Performance quantifizieren.

Damit leistet die Arbeit einen methodischen Beitrag zur wissenschaftlichen Evaluation von RAG-Systemen. Sie stellt ein reproduzierbares Verfahren bereit und liefert empirische Evidenz dazu, welche technischen Parameter für semantische Suchsysteme in Bildungsszenarien besonders relevant sind. Dadurch erweitert sie den Forschungsstand an der Schnittstelle von Information Retrieval, Natural Language Processing und digitalem Lernen.

3.3.2. Praktische Relevanz

Die praktische Relevanz dieser Arbeit ergibt sich aus ihrem Beitrag zur datensouveränen Nutzung von KI-Technologien im Hochschulkontext. Der entwickelte Open-Source-Prototyp ermöglicht die semantische Erschließung persönlicher Lernmaterialien, ohne dass Daten an externe Cloud-Dienste übertragen werden müssen. Durch den lokalen oder institutionellen Betrieb behalten Hochschulen und Studierende die volle Kontrolle über sensible oder urheberrechtlich geschützte Inhalte. Dieser Aspekt ist insbesondere unter den Anforderungen der DSGVO und hochschulinterner Datenschutzrichtlinien von zentraler Bedeutung.

Darüber hinaus zeigt die Arbeit, wie moderne KI-Methoden in bestehende hochschulische Infrastrukturen integriert werden können. Der modulare Aufbau mit austauschbaren Retriever-, Chunking- und Embedding-Komponenten erlaubt eine flexible

Anpassung an vorhandene Systeme wie MinIO[1], Postgres[2] oder Qdrant. Hochschulen können das System sowohl in der Forschung als auch in der Lehre einsetzen, um Studierenden ein Werkzeug bereitzustellen, das eigenständiges und informationsbasiertes Lernen unterstützt und den Aufwand für die Materialsichtung deutlich reduziert.

Der Open-Source-Charakter der Lösung hat zudem eine strategische Bedeutung. Er ermöglicht Transparenz, fördert die Weiterentwicklung durch die wissenschaftliche Community und verhindert Abhängigkeiten von proprietären Plattformen. Damit leistet die Arbeit einen praxisorientierten Beitrag zur Gestaltung vertrauenswürdiger KI-Anwendungen im Bildungsbereich und unterstützt den Aufbau nachhaltiger, offener Lerninfrastrukturen.

3.4. Zielsetzung der Arbeit

Diese Arbeit verfolgt das Ziel, eine Open-Source-basierte Anwendung zu entwickeln und systematisch zu evaluieren, die das Prinzip der RAG nutzt, um Studierenden einen effizienteren und semantisch fundierten Zugang zu ihren digitalen Lernmaterialien zu ermöglichen. Ausgangspunkt ist die Beobachtung, dass Lerninhalte im digitalen Selbststudium häufig über verschiedene Systeme verteilt, unstrukturiert und schwer auffindbar sind. Die angestrebte Anwendung soll daher Dokumente automatisiert verarbeiten, deren Inhalte semantisch durchsuchen und natürlichsprachliche Anfragen mit fundierten, quellengestützten Antworten beantworten.

Im Rahmen dieser Zielsetzung wird ein funktionsfähiger Prototyp entwickelt, der vollständig auf offenen Softwarekomponenten basiert und datenschutzkonform betrieben werden kann. Die Architektur der Anwendung ist modular aufgebaut und erlaubt die flexible Kombination verschiedener Pipeline-Komponenten. Dadurch können unterschiedliche Varianten von Retriever-, Chunking- und Embedding-Strategien implementiert und hinsichtlich ihrer Auswirkungen auf die Qualität der generierten Antworten miteinander verglichen werden. Zusätzlich wird untersucht, inwiefern sich der Einsatz proprietärer Sprachmodelle auf die Ergebnisqualität auswirkt und welche Unterschiede im Vergleich zu offenen Modellen bestehen.

Die Evaluation erfolgt auf Basis eines manuell erstellten Golden-Standard-Datensatzes und unter Verwendung des Open-Source-Frameworks RAGAS. Dabei werden Qualitätsmetriken wie Faithfulness und Relevance herangezogen, um die inhaltliche Genauigkeit und Kontexttreue der generierten Antworten objektiv zu bewerten. Neben der Untersuchung verschiedener Pipeline-Konfigurationen wird auch der Einfluss proprietärer

[1] MinIO: `https://www.min.io/`
[2] Postgres: `https://www.postgresql.org/`

Sprachmodelle analysiert, um Unterschiede in der Antwortqualität und im Verhalten der Systeme zu identifizieren. Die Ergebnisse sollen zeigen, welche Kombinationen von Komponenten und Modelltypen besonders geeignet sind, um den Informationszugang im digitalen Selbststudium zu verbessern.

4. Methodik

4.1. Aufbau der Evaluationspipeline

Zur systematischen Untersuchung der Pipeline-Konfigurationen wurde ein zweistufiges Evaluationsverfahren entwickelt. Im ersten Schritt führt ein automatisiertes Skript alle technisch sinnvollen Kombinationen der Pipeline-Komponenten (Dokumentparser, Chunking-Strategie, Embedding-Modell, Retrievalverfahren) aus. Alle Konfigurationen verwenden dasselbe Large Language Model (Llama-3.1-8B), um den Einfluss der Retrieval-Komponenten isoliert bewerten zu können. Während der Ausführung werden die Ähnlichkeitswerte der gefundenen Chunks sowie die generierten Antworten mitsamt Kontextpassagen persistiert.

Im zweiten Schritt werden die Antworten mithilfe des RAGAS-Frameworks bewertet. Um den hohen Ressourcenbedarf zu begrenzen, erfolgt zunächst ein Vorfiltering anhand semantischer Ähnlichkeitsmetriken und Retrieval-Scores. Die leistungsstärksten Varianten werden anschließend einer vollständigen RAGAS-Analyse unterzogen.

Durch die Trennung von Pipeline-Ausführung, Vorbewertung und detaillierter Qualitätsanalyse bleibt das Verfahren reproduzierbar und transparent. Der modulare Aufbau erlaubt, den Einfluss einzelner Komponenten präzise zu isolieren.

4.2. Evaluationsdatensatz

4.2.1. Ziel und Zweck des Golden Standard Datensatzes

Zur objektiven Bewertung der Antwortqualität der verschiedenen RAG-Pipeline-Varianten wurde ein Evaluationsdatensatz erstellt, der als Golden Standard dient. Er besteht aus validierten Frage-Antwort-Paaren, die direkt aus den eingesetzten Lernmaterialien abgeleitet wurden. Der Datensatz schafft einen festen Vergleichsmaßstab, anhand dessen die Leistungsfähigkeit der Pipeline-Konfigurationen quantifiziert werden kann. Dadurch lässt sich unabhängig von subjektiven Einschätzungen beurteilen, ob eine generierte Antwort korrekt, faktentreu und relevant ist.

Ein Golden-Standard-Datensatz ist notwendig, da eine manuelle Bewertung der Modellausgaben sowohl zeitaufwendig als auch nicht reproduzierbar wäre. Subjektive Einschätzungen einzelner Personen könnten zu Verzerrungen führen, etwa durch unterschiedliche Interpretationen der Inhalte. Ein strukturierter Datensatz hingegen bietet klare Referenzen und ermöglicht eine systematische, wiederholbare Bewertung.

Der Datensatz bildet die Grundlage für die quantitative Analyse mit dem Evaluationsframework RAGAS. Dessen Qualitätsmetriken, unter anderem Faithfulness (Übereinstimmung mit der Quelle) und Relevance (Passgenauigkeit zur Frage), können nur berechnet werden, wenn ein eindeutiger Referenzwert vorliegt. Der Golden Standard stellt diesen bereit und ermöglicht einen Vergleich der Pipeline-Varianten unter identischen Bedingungen.

Durch die Trennung von Datensatz und Pipeline-Implementierung wird sichergestellt, dass die Bewertung unabhängig vom Modellverhalten erfolgt. Der Golden Standard fungiert damit als zentrale, objektive Bewertungsgrundlage dieser Arbeit.

4.2.2. Generierung der Fragen

Die Erstellung der Fragen erfolgte halbautomatisch unter Nutzung eines Large Language Models. Ausgewählte Textabschnitte aus den Lernmaterialien wurden als Eingabe verwendet, um realistische und studienrelevante Fragen zu erzeugen, die typische Informationsbedürfnisse im digitalen Selbststudium abbilden.

Die KI-generierten Fragen wurden nicht ungeprüft übernommen, sondern durchliefen eine mehrstufige manuelle Validierung. Dabei wurde sichergestellt, dass jede Frage inhaltlich korrekt aus dem Ausgangsmaterial abgeleitet werden konnte, keine zusätzlichen Hintergrundinformationen voraussetzte und sprachlich präzise formuliert war. Unklare, doppeldeutige oder redundante Fragen wurden korrigiert oder verworfen.

Durch dieses Vorgehen entstand ein ausgewogener Fragenkatalog mit unterschiedlichen Fragetypen, darunter Definitionsfragen, Verständnisfragen und kontextbezogene Anwendungsfragen, der verschiedene Anforderungen an Retrieval und Antwortgenerierung abbildet.

4.2.3. Erstellung der Referenzantwort

Die Referenzantworten des Datensatzes wurden vollständig manuell erstellt, um eine neutrale und inhaltlich verlässliche Bewertungsgrundlage zu gewährleisten. Für jede zuvor generierte und validierte Frage wurde zunächst die entsprechende Textstelle im Originaldokument identifiziert. Auf dieser Grundlage wurde eine präzise Antwort

formuliert, die den wesentlichen Inhalt des Abschnitts korrekt wiedergibt, ohne den Originaltext wortwörtlich zu übernehmen. Die Antworten wurden bewusst sinngemäß formuliert, um sicherzustellen, dass sie den tatsächlichen Inhalt zusammenfassen und nicht lediglich Textpassagen reproduzieren.

Bei der Erstellung der Referenzantworten wurde auf zwei Qualitätsaspekte besonders geachtet: Faktentreue und Eindeutigkeit. Jede Antwort musste eindeutig aus dem dokumentierten Kontext ableitbar sein. Fragen, bei denen mehrere Interpretationen möglich waren oder keine klare Antwort aus dem Material hervorging, wurden verworfen oder überarbeitet. Insbesondere bei Definitions- oder Begriffsklärungen wurde darauf geachtet, zentrale Merkmale kompakt und unverfälscht darzustellen.

Die Referenzantworten wurden unabhängig von der späteren Implementierung der RAG-Pipeline erstellt, um eine Trennung zwischen Goldstandard und Evaluationssystem sicherzustellen. Dadurch wird verhindert, dass implizite Voreingenommenheit oder Systemwissen die Bewertung beeinflussen. Der manuelle Prozess stellt sicher, dass der Datensatz eine verlässliche Grundlage für die quantitative Bewertung der Antwortqualität bildet und als stabiler Referenzrahmen für die RAGAS-basierte Evaluation dient.

4.3. Evaluationsverfahren

Zur Bewertung der Antwortqualität wurde das Open-Source-Framework RAGAS eingesetzt. Es ermöglicht eine automatisierte, reproduzierbare und quantitativ vergleichbare Analyse durch ein Large Language Model, das die Beziehung zwischen Frage, Antwort und Kontext bewertet.

Zu den verwendeten Metriken zählen Faithfulness (keine Halluzinationen), Context Precision (Relevanz der abgerufenen Passagen), Semantic Similarity (inhaltliche Übereinstimmung mit Referenzantwort) und Factual Correctness (sachliche Korrektheit). Für jede Metrik wird ein Score zwischen 0 und 1 berechnet. Die Ergebnisse werden aggregiert, sodass Mittelwerte pro Pipeline-Variante verglichen werden können.

Der Einsatz von RAGAS ermöglicht eine standardisierte Evaluation, auf deren Grundlage datenbasierte Rückschlüsse auf die Wirksamkeit einzelner Pipeline-Komponenten gezogen werden können.

4.4. Validität und Reproduzierbarkeit

Zur Sicherstellung der wissenschaftlichen Qualität wurden interne Validität, externe Validität, Reliabilität und Reproduzierbarkeit systematisch berücksichtigt. Die interne Validität wurde durch RAGAS gewährleistet, das ein Sprachmodell zur automatisierten Bewertung der Antwortqualität einsetzt. Der Golden-Standard-Datensatz und einheitliche Prompt-Strukturen stellten sicher, dass Unterschiede in den Ergebnissen auf die Pipeline-Konfiguration zurückzuführen sind. Störvariablen wie variierende Kontextlängen wurden durch identische Inputs ausgeschlossen.

Die externe Validität ist auf den spezifischen Studienbereich beschränkt, das Vorgehen selbst ist jedoch inhaltsunabhängig und übertragbar. Die Reliabilität wird durch dokumentierte Versionsstände, Systemeinstellungen und feste Evaluationsprozesse gewährleistet, während Schwankungen in RAGAS durch Mittelwertbildung abgefangen werden.

Die Reproduzierbarkeit wurde durch vollständige Offenlegung des technischen Setups sichergestellt. Code, Skripte und Konfigurationen wurden versioniert und auf GitHub bereitgestellt. Während der Kern auf Open-Source-Komponenten basiert, wurden ergänzend proprietäre Modelle transparent dokumentiert eingesetzt.

Methodische Einschränkungen bestehen durch die modellbasierte Bewertung mittels RAGAS sowie die manuelle Erstellung des Golden-Standard-Datensatzes. Die Datensatzgröße begrenzt zudem die Generalisierbarkeit. Durch Cross-Checks, detaillierte Parameterdokumentation und konsistente Datenverarbeitung wurde jedoch die Nachvollziehbarkeit und methodische Belastbarkeit der Resultate sichergestellt.

5. Konzeption des Systems

5.1. Anforderungen

Die folgenden Anforderungen leiten sich aus den Zielen der Arbeit sowie den technischen und datenschutzrechtlichen Rahmenbedingungen des Hochschulkontexts ab. Sie definieren die zentralen Funktionen, die das RAG-System erfüllen muss, um den intendierten Nutzen zu erreichen.

5.1.1. Funktionale Anforderungen

Das System soll gängige universitäre Dateiformate wie PDF, DOCX und PPTX verarbeiten und deren Inhalte semantisch erschließen, sodass sie in natürlicher Sprache durchsucht werden können. Nutzeranfragen sollen präzise und quellenbasierte Antworten erzeugen, die eindeutig auf die ursprünglichen Dokumente zurückführbar sind. Damit wird sichergestellt, dass alle Ausgaben den wissenschaftlichen Anforderungen an Nachvollziehbarkeit und Quellenbezug entsprechen. Optional kann das System um eine dialogische Komponente erweitert werden, die kontextbezogene Rückfragen und mehrstufige Interaktionen unterstützt.

Die funktionalen Anforderungen wurden nach ihrer Relevanz für den praktischen Einsatz im Hochschulkontext priorisiert. Tabelle 5.1 fasst die zentralen Funktionen und deren Prioritätsstufe zusammen.

Tabelle 5.1.: Priorisierte funktionale Anforderungen

Anforderung	**Priorität**
Verarbeitung von PDF/DOCX/PPTX	MUSS
Semantische Suche in natürlicher Sprache	MUSS
Quellenbasierte Antwortgenerierung	MUSS
Konversationsfähigkeit	SOLL

Die aufgeführten Anforderungen bilden die Grundlage für die Systemarchitektur und die spätere Evaluation der Pipeline-Komponenten. Zentrale Funktionen wie semantische Suche und quellenbasierte Antwortgenerierung besitzen höchste Priorität, da sie den Kernnutzen des Systems definieren.

5.1.2. Nicht-funktionale Anforderungen

Das System muss drei zentrale nicht-funktionale Anforderungen erfüllen: Datenschutz, Modularität und Reproduzierbarkeit. Datenschutz und Datensouveränität besitzen höchste Priorität. Sämtliche Datenverarbeitung erfolgt lokal und verhindert eine Übertragung an externe Systeme. Darüber hinaus ist eine DSGVO konforme Speicherung und Löschung personenbezogener Daten sicherzustellen.

Die Modularität der Architektur muss den unabhängigen Austausch einzelner Pipeline-Komponenten ermöglichen, um verschiedene Konfigurationen systematisch evaluieren zu können.

Für wissenschaftliche Nachvollziehbarkeit müssen alle Pipeline-Konfigurationen vollständig dokumentiert werden. Eine reproduzierbare Ausführung der Experimente ist sicherzustellen, sodass identische Bedingungen zu konsistenten Ergebnissen führen.

5.2. Systemarchitektur

Die Architektur trennt zentrale Funktionsbereiche wie Dokumentenverarbeitung, Indexierung, Retrieval und Antwortgenerierung voneinander. Abbildung Abb. 5.1 zeigt den Aufbau: Parser extrahieren Text aus Dokumenten, Chunking-Module segmentieren die Inhalte, Embedding-Modelle erzeugen Vektorrepräsentationen, und der Retriever identifiziert relevante Passagen für die Antwortgenerierung.

Alle Verarbeitungsschritte erfolgen lokal („on-premises"), sodass keine personenbezogenen oder urheberrechtlich geschützten Daten an externe Dienste übermittelt werden. Bei Bedarf lassen sich einzelne Module wie Embeddings oder Sprachmodelle durch Cloud-Dienste ersetzen, wobei der Datenschutz entsprechend zu berücksichtigen ist.

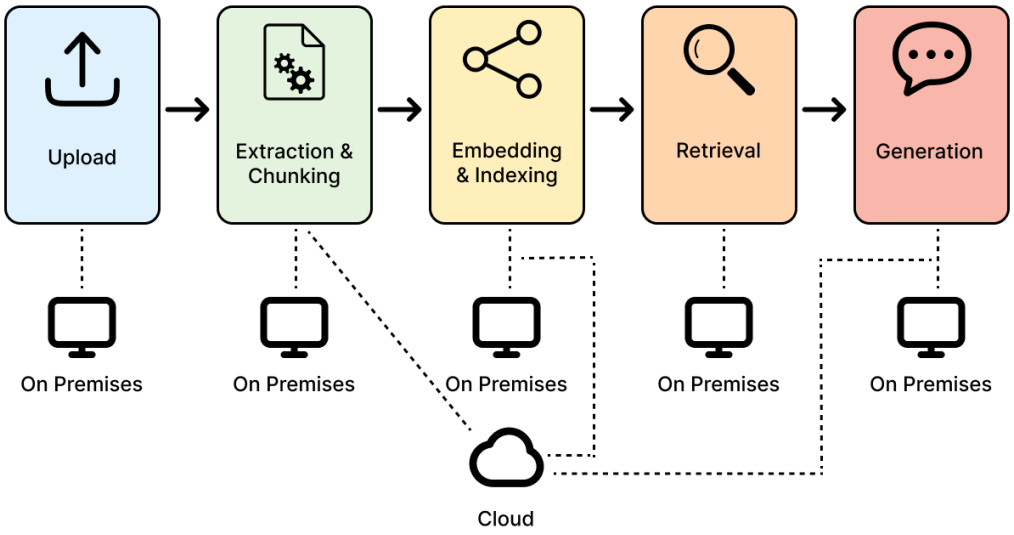

Abbildung 5.1.: Übersicht über die Hauptkomponenten des Systems

5.2.1. Übersicht und Hauptkomponenten

Das System umfasst fünf zentrale Hauptkomponenten, die den vollständigen Verarbeitungsprozess vom Dokumentenupload bis zur Antwortgenerierung abbilden. Der modulare Aufbau ermöglicht eine klare funktionale Trennung und gewährleistet die Austauschbarkeit einzelner Komponenten innerhalb der RAG-Pipeline.

Benutzeroberfläche (Frontend) Die Benutzeroberfläche bildet die Interaktionsschicht zwischen Anwender und System. Sie bietet eine Chat ähnliche Oberfläche, über die Dokumente hochgeladen, Suchanfragen formuliert und generierte Antworten mit Quellenangaben abgerufen werden können. Die Kommunikation mit dem Backend erfolgt über definierte API-Endpunkte.

Backend-API Das Backend besteht aus zwei Diensten: Die Query API koordiniert Vektorsuche in Qdrant und LLM-basierte Antwortgenerierung. Der Ingestion Worker verarbeitet Dokumente durch Parsing, Chunking, Embedding und Indexierung. Die Architektur separiert Dokumentenextraktion, Text-Embedding, Vektorretrieval und Sprachgenerierung. Dadurch lassen sich verschiedene Implementierungen (z. B. Docling vs. Unstructured) systematisch evaluieren, ohne die Gesamtstruktur zu verändern.

RAG-Pipeline (Retrieval-Augmented Generation) Die RAG-Pipeline bildet das Kernmodul des Systems. Sie kombiniert semantisches Retrieval mit generativer Antworterzeugung und besteht aus mehreren austauschbaren Komponenten, darunter Chunker, Embedding-Modul, Retriever, optionaler Reranker sowie das verwendete Large Language Model (LLM). Das Zusammenspiel dieser Module wird in Abschnitt 5.3 detailliert beschrieben.

Speicher- und Indexschicht Diese Schicht verwaltet sowohl die hochgeladenen Dokumente als auch deren semantische Repräsentationen. Der objektspeicherbasierte MinIO-Dienst[1] dient zur Ablage der Originaldateien, während die Vektordatenbank QDrant die Embeddings der Textsegmente speichert. Die Trennung zwischen Dokumenten- und Indexspeicher gewährleistet eine saubere Datenhaltung, unterstützt Datenschutzanforderungen und ermöglicht ein performantes semantisches Retrieval.

Protokollierungs- und Evaluationskomponente Ergänzend werden während der Verarbeitung Leistungsmetriken, Laufzeiten und Antwortqualitäten protokolliert. Diese Daten bilden die Grundlage für die spätere quantitative Analyse und erlauben eine systematische Bewertung der Pipeline-Konfigurationen.

5.3. Prozesslogik der RAG-Pipeline

Die RAG-Pipeline (Retrieval-Augmented Generation) bildet das funktionale Kernmodul des Systems. Sie kombiniert semantisches Information Retrieval mit der generativen Leistungsfähigkeit eines Large Language Models, um kontextbezogene Antworten auf Basis lokal gespeicherter Dokumente zu erzeugen.

Der Verarbeitungsprozess umfasst drei aufeinanderfolgende Hauptphasen. In der Indexierungsphase werden Dokumente verarbeitet und semantisch indiziert. Die Retrievalphase dient dem Abruf relevanter Textsegmente, während in der Generierungsphase das Sprachmodell auf Grundlage der gefundenen Inhalte eine kohärente Antwort formuliert. Abbildung 2 zeigt den schematischen Aufbau der Pipeline.

Jede dieser Phasen erzeugt Datenartefakte, die der jeweils nächsten Stufe als Eingabe dienen. Die technische Umsetzung und die verwendeten Komponenten werden in Kapitel 6 detailliert beschrieben.

[1] MinIO: `https://www.min.io/`

5.3.1. Indexierungsphase

Die Indexierungsphase transformiert hochgeladene Dokumente in eine durchsuchbare Wissensrepräsentation. Sie wird von einem separaten Container ausgeführt, der beliebig oft manuell gestartet werden kann. Der Prozess umfasst vier aufeinanderfolgende Schritte und ist nicht an den Upload-Zeitpunkt gebunden.

Dokumentenupload und speicherung

Der Upload der Dokumente erfolgt über die MinIO-Weboberfläche. Die Dateien werden in einem dedizierten Bucket persistent gespeichert, erhalten eine eindeutige Dokument-ID und stehen dem Indexer anschließend für die weitere Verarbeitung zur Verfügung. Die Trennung von Originaldateien im Objektspeicher und den daraus erzeugten Vektorrepräsentationen in der Datenbank ermöglicht eine kontrollierte und nachvollziehbare Datenverwaltung. Wird ein Dokument in MinIO gelöscht, verbleiben seine Embeddings zunächst im Vektorindex, bis die Indexierung erneut ausgeführt wird. Der Objektspeicher bildet die zentrale Referenzquelle aller Originaldokumente und stellt sicher, dass alle weiteren Verarbeitungsschritte auf einem konsistenten und überprüfbaren Datenbestand basieren.

Parsing und Textextraktion

Nach der Speicherung werden die Dokumente durch ein zentrales Parsing-Modul verarbeitet, das die Rohtextinhalte unabhängig vom Dateiformat extrahiert. Das Modul erkennt grundlegende Strukturelemente wie Überschriften, Absätze und Textblöcke, um den inhaltlichen Aufbau der Dokumente möglichst vollständig zu bewahren. Die Erhaltung dieser logischen Struktur ist wesentlich, da sie in späteren Verarbeitungsschritten eine präzise Zuordnung von Textpassagen zu ihren Ursprungsdokumenten und damit eine nachvollziehbare Quellenangabe ermöglicht.

Segmentierung (Chunking)

Der extrahierte Text wird in kleinere, semantisch zusammenhängende Abschnitte (Chunks) unterteilt. Dieser Schritt ist entscheidend für das Gleichgewicht zwischen Kontexterhalt und Präzision des Retrievals. Zu kleine Segmente führen zu Informationsverlust, während zu große Segmente die Relevanz der Treffer mindern können.

Die Segmentierung erfolgt auf Basis konfigurierbarer Parameter wie Chunk-Größe, Überlappung und Segmentierungsstrategie (längen-, satz- oder semantikbasiert). Diese

Parameter werden in Kapitel 6 detailliert beschrieben und im Rahmen der Evaluation systematisch hinsichtlich ihres Einflusses auf die Antwortqualität untersucht. Ein kontrollierter Overlap zwischen benachbarten Chunks dient dazu, Informationsverluste an Segmentgrenzen zu minimieren.

Embedding und Indexierung

Für jedes Textsegment wird mithilfe eines Embedding-Modells eine semantische Vektorrepräsentation erzeugt. Diese numerische Repräsentation bildet die Bedeutung des jeweiligen Textabschnitts in einem hochdimensionalen Vektorraum ab und schafft damit die Grundlage für das semantische Retrieval in der späteren Abfragephase.

Die Vektordatenbank speichert die erzeugten Embeddings zusammen mit den Metadaten Dokument-ID, Chunk-Position und Quellpfad. Je nach Dokumenttyp lassen sich zusätzliche Informationen wie Seitenzahl oder Kapitelstruktur hinterlegen.

Durch diese Kombination aus semantischer Repräsentation und strukturierten Metadaten entsteht ein durchsuchbarer Wissensindex, der Bedeutungsähnlichkeiten zwischen Nutzeranfragen und Dokumentinhalten erfassen kann, ohne auf exakte Stichwortübereinstimmungen angewiesen zu sein.

5.3.2. Retrievalphase

Die Retrievalphase wird bei jeder Nutzeranfrage durchlaufen und identifiziert die zur Frage relevantesten Dokumentenpassagen aus dem zuvor aufgebauten Index.

Anfrageverarbeitung

Bei einer Suchanfrage wird der eingegebene Text über das Frontend an das Backend übermittelt. Das System wandelt die Anfrage in eine semantische Vektorrepräsentation um, wobei dasselbe Embedding-Modell verwendet wird wie bei der Indexierung. Diese Konsistenz gewährleistet die Vergleichbarkeit zwischen Query-Vektor und gespeicherten Dokumentensegmenten, da beide im selben semantischen Raum repräsentiert werden.

Retrieval und Ähnlichkeitsberechnung

Das Retrieval-Modul vergleicht den Query-Vektor mit den gespeicherten Segmentrepräsentationen und identifiziert die k ähnlichsten Textpassagen anhand einer Ähnlichkeitsmetrik. In der Regel wird die Kosinus-Ähnlichkeit oder das Skalarprodukt verwendet, da beide effizient berechnet werden können und für normalisierte Vektoren äquivalent sind.

Der Parameter k, der die Anzahl der beim Retrieval zurückgegebenen Textsegmente bestimmt, hat einen wesentlichen Einfluss auf die Qualität der Antwortgenerierung. Wird ein zu kleiner Wert gewählt, kann der relevante Kontext unvollständig bleiben, ein zu großer Wert erhöht dagegen die Wahrscheinlichkeit irrelevanter Informationen und verlängert die Verarbeitungszeit.

Optionales Re-Ranking

Zur Verbesserung der Retrievalqualität kann ein optionales Re-Ranking-Modul eingesetzt werden. Während die initiale Vektorsuche eine breite Vorauswahl trifft, bewertet ein Cross-Encoder die Kandidaten semantisch präziser, indem er Query und Dokument gemeinsam analysiert. Im Gegensatz zu bi-encoderbasierten Embeddingvergleichen erfasst dieser Ansatz kontextabhängige Relevanzsignale wie Synonyme, Negationen oder implizite Bezüge.

In der Implementierung werden zunächst mehr Kandidaten abgerufen als benötigt (z. B. top-100). Der Reranker ordnet diese nach Relevanzscore und selektiert die bestpassenden Segmente (z. B. top-10) für die Generierung. Dieses zweistufige Verfahren steigert die Präzision, erhöht jedoch die Latenz durch zusätzliche Inferenz.

5.3.3. Generierungsphase

In der Generierungsphase werden die abgerufenen Dokumentenpassagen verwendet, um eine natürlichsprachliche Antwort auf die Nutzeranfrage zu erzeugen.

Prompt-Konstruktion und Antwortgenerierung

Die im Retrieval-Schritt ermittelten Segmente werden zusammen mit der Nutzeranfrage in ein strukturiertes Prompt überführt. Dabei kommt das Prinzip der *Strict Context Injection* zur Anwendung, indem das Sprachmodell ausschließlich auf die bereitgestellten Kontextpassagen zugreifen darf und explizit angewiesen ist, seine Antwort daraus abzuleiten. Auf das in den Modellgewichten gespeicherte Vorwissen wird bewusst nicht

zurückgegriffen, um die Nachvollziehbarkeit und Quellengebundenheit der generierten Inhalte zu gewährleisten.

Zur weiteren Erhöhung der Transparenz wird eine *Citation-First-Strategie* implementiert. Die im Prompt enthaltenen Quellen werden mit eindeutigen Identifikatoren (z. B. „Dokument 1", „Dokument 2") versehen und zusätzlich mit Metadaten wie Dateinamen und Relevanzwerten annotiert. Das Modell wird ausdrücklich instruiert, bei relevanten Aussagen auf diese Quellen zu verweisen. In den generierten Antworten erfolgt die Referenzierung entsprechend, etwa in der Form „Dies geht aus Dokument 2 (AP1.pdf, Relevanz: 0.65) hervor".

Das Large Language Model verarbeitet den konstruierten Prompt und erzeugt eine kohärente, natürlichsprachliche Antwort mit expliziten Quellenverweisen. Diese wird strukturiert mit Quellenangaben ausgegeben. Der beschriebene dreiphasige Ablauf stellt sicher, dass jede generierte Antwort auf explizit referenziertem Dokumentenwissen basiert und somit nachvollziehbar sowie überprüfbar bleibt.

5.4. Cloud- und Infrastrukturkonzept

Die Infrastruktur des Systems wurde so konzipiert, dass sie eine sichere, reproduzierbare und lokal kontrollierte Ausführung aller Komponenten ermöglicht. Grundlage bildet eine containerisierte Umgebung auf Basis von Docker, die eine isolierte Laufzeitumgebung und eine einfache Portierung zwischen unterschiedlichen Host-Systemen gewährleistet. Ziel ist es, die Vorteile cloudnativer Architekturprinzipien zu nutzen, ohne Abhängigkeiten zu externen, proprietären Diensten zu erzeugen.

Das System folgt dem Prinzip der Trennung von Rechen- und Speicherressourcen. Die RAG-Pipeline, das Backend, die Vektordatenbank und das LLM laufen in separaten Containern, die unabhängig vom persistenten Speicher betrieben werden. Diese Architektur ermöglicht eine flexible Skalierung, den Austausch einzelner Komponenten sowie ein konsistentes Laufzeitverhalten über verschiedene Umgebungen hinweg. Der Speicher wird über MinIO realisiert, eine S3-kompatible Open-Source-Lösung, die eine strukturierte und datenschutzkonforme Ablage aller Dokumente und abgeleiteten Daten gewährleistet.

Die Systemarchitektur ist so gestaltet, dass sie sich flexibel an unterschiedliche technische Umgebungen anpassen lässt. Durch den Einsatz offener Standards und quelloffener Komponenten bleibt sie unabhängig von proprietären Plattformen und kann ohne größere Anpassungen erweitert oder in bestehende Infrastrukturen integriert werden. Auf diese Weise wird sichergestellt, dass das System langfristig wartbar bleibt und den

Anforderungen an Datenschutz, Reproduzierbarkeit und wissenschaftliche Transparenz entspricht.

5.4.1. Zusammenfassung

Das in diesem Kapitel vorgestellte Konzept bildet die architektonische und methodische Grundlage des entwickelten RAG-Systems. Es definiert die Anforderungen an eine datenschutzkonforme Erschließung heterogener Lehrmaterialien und legt messbare Evaluationskriterien fest, die eine objektive Beurteilung der Systemleistung ermöglichen. Durch die klare Unterscheidung funktionaler und nicht-funktionaler Anforderungen entsteht eine präzise Spezifikation, die technische Robustheit und praktische Anwendbarkeit gleichermaßen sicherstellt.

Insgesamt schafft das hier dargestellte Konzept den strukturellen Rahmen für die nachfolgende Implementierung (Kapitel 6), in der die beschriebenen Komponenten technisch umgesetzt und die verschiedenen Pipeline-Varianten für die Evaluation vorbereitet werden.

6. Implementierung des Prototyps

6.1. Auswahl der Technologien und Frameworks

Die technische Umsetzung des Prototyps basiert auf einer containerisierten Entwicklungsumgebung, die nach den Kriterien Kompatibilität, Stabilität, Open-Source-Verfügbarkeit und wissenschaftliche Nachvollziehbarkeit konzipiert wurde.

Die Implementierung erfolgte in Python 3.11, da die Sprache etablierte Bibliotheken für maschinelles Lernen, Vektoroperationen und Dokumentenverarbeitung bereitstellt. Das Frontend wurde mit TypeScript und React umgesetzt, um eine typsichere Benutzeroberfläche zu gewährleisten.

Das Backend nutzt FastAPI für asynchrone REST-Endpunkte. Anstelle eines einheitlichen RAG-Frameworks implementiert das System eine modulare Architektur, bei der jede Pipeline-Stufe unabhängig konfiguriert werden kann.

Die Entwicklungsumgebung ist vollständig in Docker containerisiert, sodass alle Abhängigkeiten plattformunabhängig reproduzierbar sind. Git dient als Versionsverwaltung, Abhängigkeiten werden über requirements.txt und package.json verwaltet.

Für GPU-beschleunigte Inferenz wurde eine NVIDIA-GPU mit CUDA eingesetzt, die Embedding-Generierung und LLM-Inferenz ermöglicht. Die Entwicklung erfolgte auf einem System mit einer RTX 2060 und später mit einer RTX 3090.

6.2. RAG-Pipeline-Komponente

Zur systematischen Evaluation der RAG-Pipeline wurden vier zentrale Komponenten variiert: Dokumenten-Parsing, Text-Chunking, Embedding-Erzeugung und Retrieval-Strategie. Jede Komponente liegt in mehreren Implementierungsvarianten vor, deren Kombination zu 56 untersuchten Pipeline-Konfigurationen führt.

Die Ausführung erfolgt containerisiert über Docker Compose. Die Aktivierung der jeweiligen Varianten wird über Umgebungsvariablen gesteuert, wodurch Konfigurationswechsel ohne Änderungen am Quellcode möglich sind. Dieses Vorgehen gewährleistet eine reproduzierbare und systematisch vergleichbare Evaluation aller Varianten.

Die folgenden Abschnitte beschreiben die implementierten Varianten je Komponente sowie deren Evaluationsergebnisse.

6.2.1. Parser-Varianten

Das in Kapitel 5.3.1 beschriebene Parsing wurde in der Implementierung mit unterschiedlichen Werkzeugen realisiert, um deren Eignung für akademische Dokumente zu evaluieren. Insgesamt wurden drei Parser-Varianten untersucht, die sich in Extraktionsqualität, Geschwindigkeit und Ressourcenverbrauch unterscheiden.

Docling (lokale GPU-Verarbeitung)

Docling[1] ist ein Open-Source-Parsing-Framework, das multimodale Vision-Language-Modelle zur Analyse komplexer Dokumentstrukturen einsetzt. Die Verarbeitung erfolgt vollständig lokal und GPU-beschleunigt, wodurch sämtliche Daten innerhalb der eigenen Infrastruktur verbleiben. Docling erkennt Layout-Elemente wie Tabellen, Überschriften, Textbereiche, Listen und Formeln und gibt die Ergebnisse in strukturierter Form zurück. Die semantische Struktur des Dokuments bleibt dabei erhalten, sodass beispielsweise Hierarchien zwischen Überschriften und Absätzen oder tabellarische Informationen korrekt abgebildet werden.

Da keine externen API-Abhängigkeiten bestehen, werden keine Daten an Dritte übertragen, was insbesondere im Hinblick auf den Datenschutz im Hochschulkontext von Bedeutung ist. Die strukturierte Ausgabe ermöglicht eine präzisere Segmentierung in der nachfolgenden Chunking-Phase, da Kontextgrenzen wie Kapitelwechsel oder Tabellenstrukturen bereits erkannt sind. Dies unterscheidet Docling von einfachen Text-Extraktoren, die lediglich eine lineare Zeichenkette ohne strukturelle Metadaten liefern.

Unstructured.io (Cloud-API)

Unstructured.io[2] ist ein cloudbasierter Parsing-Dienst, der Dokumente über eine REST-API entgegennimmt und die extrahierten Inhalte in strukturierter JSON-Form zurückliefert. Im Gegensatz zur lokalen Verarbeitung durch Docling erfolgt die gesamte Extraktion auf entfernten Servern. Der Dienst identifiziert verschiedene Dokumentelemente wie Überschriften, Fließtext, Tabellen und Abbildungen und rekonstruiert deren

[1] Doclin: `https://github.com/docling-project/docling`
[2] Unstructured.io: `https://unstructured.io/`

hierarchische Beziehungen. Die Partitionierungsstrategie kann konfiguriert werden, sodass Elemente beispielsweise anhand von Abschnittstiteln gruppiert werden.

Obwohl Unstructured.io eine proprietäre Cloud-Lösung darstellt, wurde sie in diese Untersuchung aufgenommen, um einen Vergleich zwischen lokal ausführbaren Open-Source-Parsern und cloudbasierten Ansätzen zu ermöglichen. Dadurch kann der Einfluss unterschiedlicher Verarbeitungsarchitekturen auf die Pipeline-Leistung besser eingeschätzt werden.

Ein Nachteil im Kontext dieser Arbeit ergibt sich aus dem internen Segmentierungsmechanismus von Unstructured.io. Der Dienst führt eine eigene Segmentierung durch, die nicht deaktiviert werden kann. Dadurch entsteht ein inkompatibles Ausgabeformat, das eine kombinierte Nutzung mit der semantischen Chunking-Komponente verhindert. Aus diesem Grund konnten lediglich 56 der theoretisch möglichen 72 Pipeline-Konfigurationen berücksichtigt werden.

LlamaParse (LLM-basierte Cloudverarbeitung)

LlamaParseLlamaParse: `https://www.llamaindex.ai/llamaparse` ist ein cloudbasierter Parsing-Dienst, der große Sprachmodelle zur strukturellen und inhaltlichen Analyse von Dokumenten einsetzt. Im Gegensatz zu klassischen Parsing-Frameworks nutzt LlamaParse ein Sprachmodell als Reasoning-Komponente, wodurch neben der reinen Textextraktion auch ein kontextuelles Verständnis der Dokumentinhalte entsteht. Das Modell erkennt Layoutstrukturen, Tabellen und Abbildungen und kann logische Beziehungen zwischen Dokumentteilen rekonstruieren. Die Ausgabe erfolgt im Markdown-Format und bewahrt die hierarchische Struktur des Ausgangsdokuments.

Die Integration von LlamaParse in diese Untersuchung dient nicht primär dem Vergleich unterschiedlicher Bereitstellungsmodelle, sondern der Analyse, wie sich LLM-basierte Parsing-Verfahren auf die Qualität der nachfolgenden Pipelinephasen auswirken. Damit wird untersucht, ob durch sprachmodellgestützte Extraktion ein Mehrwert bei der semantischen Strukturierung komplexer Dokumente erzielt werden kann.

6.2.2. Chunking

Das Chunking-Modul segmentiert die vom Parser extrahierten Textelemente in semantisch kohärente Einheiten. Im Rahmen dieser Arbeit werden drei Strategien verglichen: Header-aware Chunking (strukturbasiert), Adaptive Chunking (formatbasiert) und Semantisches Chunking (embedding-basiert).

Jede Strategie arbeitet mit individuellen Chunk-Größen und Segmentierungslogiken. LangChain-basierte Splitter nutzen tokenbasiertes Chunking mit konfigurierbaren Größen und Overlaps. Chonkie segmentiert entlang von Markdown-Strukturen. Unstructured.io erstellt Chunks bereits während der Extraktion mit zeichenbasierten Limits. Um Informationsverlust an Chunk-Grenzen zu verhindern, implementieren alle Strategien Überlappungen zwischen benachbarten Segmenten.

Überschreiten Chunks die Tokenobergrenze des Embedding-Modells, splittet eine satzbasierte Fallback-Strategie die Segmente automatisch mit Überlappung.

Header-aware Chunking (LangChain)

Das Header-aware Chunking segmentiert Dokumente anhand ihrer formalen Gliederung. Überschriften und hierarchische Elemente dienen als primäre Orientierungspunkte, wodurch Segmente entstehen, die der logischen Dokumentstruktur entsprechen.

Die Segmentierung extrahiert zunächst die vorhandene Hierarchie und überführt sie in Textblöcke. Diese werden nur dann weiter unterteilt, wenn die Zielgröße von `CHUNK_TOKENS = 750` (mit `CHUNK_OVERLAP = 150`) überschritten wird. Die Tokenisierung erfolgt mithilfe der `tiktoken`-Bibliothek.

Der Ansatz arbeitet deterministisch mit geringem Ressourcenbedarf und eignet sich insbesondere für klar gegliederte Lehrmaterialien. Einschränkungen ergeben sich bei Dokumenten ohne konsistente Überschriftenhierarchie.

Adaptive Chunking (Chonkie)

Das Adaptive Chunking wird mit `Chonkie`[3] realisiert und folgt einem strukturorientierten Ansatz auf Basis des Markdown-Rezepts des RecursiveChunker. Die Segmentierung erfolgt entlang von Markdown-Elementen wie Überschriften, Abschnitten und Absätzen. Die Initialisierung erfolgt über `RecursiveChunker.from_recipe("markdown")`, wodurch die logische Organisation des Dokuments bewahrt wird.

Überschreiten Segmente die Tokenobergrenze von 2000, erfolgt das beschriebene Re-Chunking mithilfe eines GPT-2-Tokenizers. Die resultierende Chunk-Größe hängt stärker von Formatierungsqualität und struktureller Tiefe als von festen Tokenparametern ab. Parameter wie `chunk_size` und `overlap` werden durch das Markdown-Rezept in der ersten Segmentierungsphase nicht herangezogen.

[3] Chonkie: `https://docs.chonkie.ai/oss/quick-start`

Der Ansatz bietet einen praktikablen Mittelweg zwischen starren, größenbasierten Verfahren und rechenintensiven, semantischen Methoden mit strukturtreuer Segmentierung.

Semantisches Chunking (Embedding-basiert)

Das semantische Chunking analysiert semantische Ähnlichkeiten zwischen aufeinanderfolgenden Sätzen durch Vektorisierung mittels desselben Embedding-Modells, das auch im späteren Retrieval eingesetzt wird. Chunk-Grenzen werden gesetzt, wenn die Kosinusähnlichkeit zwischen benachbarten Satz-Embeddings eine perzentilbasierte Schwelle unterschreitet.

Die Implementierung erfordert das Laden eines Embedding-Modells bereits während der Chunking-Phase, was zu erhöhtem Rechen- und Speicheraufwand führt. Die resultierende Chunk-Größe variiert zwischen 300 und 1500 Tokens abhängig von der thematischen Dichte.

Dieser Ansatz erzielt die höchste semantische Kohärenz, erschwert jedoch das Tokenbudget-Management durch adaptive Segmentierung. Die Methode eignet sich insbesondere für Dokumente mit komplexen thematischen Übergängen.

6.2.3. Embedding-Modelle und Varianten

Die Wahl des Embedding-Modells beeinflusst sowohl die Retrievalgenauigkeit als auch Verarbeitungsgeschwindigkeit und Ressourcenbedarf der Pipeline. Die Untersuchung vergleicht lokal ausführbare Open-Source-Modelle mit cloudbasierten proprietären Embeddings hinsichtlich Retrievalpräzision, Antwortqualität und Systemeffizienz. Die Evaluation vergleicht beide Ansätze hinsichtlich Retrievalpräzision, Antwortqualität und Systemeffizienz, um deren Eignung im Hochschulkontext zu bewerten.

Lokale Open-Source-Embeddings

Für die lokale Erzeugung von Embeddings wird das Open-Source-Modell `Nomic Embed v1.5` eingesetzt. Das Modell wurde speziell für semantische Such- und Retrievalaufgaben entwickelt und erzeugt 768-dimensionale Vektorrepräsentationen. Durch seine kompakte Architektur kann es sowohl auf CPU als auch auf GPU effizient ausgeführt werden und eignet sich damit für den Einsatz in ressourcenbeschränkten Umgebungen.

Die Embedding-Erzeugung erfolgt vollständig lokal innerhalb der Systeminfrastruktur, wodurch keine Daten an externe Dienste übertragen werden. Die erzeugten Vektoren werden in der Vektordatenbank persistiert und stehen dort für Ähnlichkeitsabfragen zur Verfügung.

Cloudbasierte Embeddings

Als cloudbasierte Embedding-Variante wird das Modell `text-embedding-3-small` von OpenAI eingesetzt. Es wurde für semantische Such-, Klassifikations- und Retrievalaufgaben optimiert und erzeugt 1 536-dimensionale Vektoren, was eine höhere Repräsentationskapazität als kompaktere Open-Source-Modelle ermöglicht. Die Vektorisierung erfolgt über einen API-Aufruf, wobei serverseitige Modellpflege kontinuierliche Qualitätsverbesserungen ohne lokale Infrastruktureingriffe ermöglicht.

Die Integration erfolgt über den `OpenAIEmbeddings`-Wrapper aus LangChain mit Batch-Verarbeitung mehrerer Textsegmente pro Anfrage, wodurch die lokale Hardwarebelastung entfällt. Da die Verarbeitung vollständig in der Cloud stattfindet, müssen Datenschutzrichtlinien berücksichtigt werden, insbesondere im Umgang mit sensiblen Lehrmaterialien.

6.2.4. Retrieval-Strategien und Reranking

In dieser Arbeit werden vier Retrievalstrategien untersucht, die sich hinsichtlich Suchlogik und Rechenaufwand unterscheiden. Zwei Basisstrategien stehen dabei im Mittelpunkt, die unterschiedliche Suchprinzipien nutzen. Dense Retrieval arbeitet rein vektorbasiert, während Hybrid Retrieval semantische Vektorsuche mit lexikalischem Keyword Matching kombiniert.

Beide Basisstrategien lassen sich durch ein optionales Reranking Modul erweitern, das die zunächst gefundenen Kandidaten mit einem Cross Encoder erneut bewertet. Dadurch entstehen insgesamt vier Retrievalkonfigurationen. Dense, Dense plus Reranking, Hybrid und Hybrid plus Reranking. Die folgenden Abschnitte beschreiben zunächst die Basis-Strategien, anschließend die Reranking-Komponente und zuletzt deren Integration in die erweiterten Varianten.

Dense Retrieval

Dense Retrieval nutzt ausschließlich semantische Ähnlichkeitsbeziehungen im Vektorraum. Das Embedding-Modell transformiert sowohl die Benutzeranfrage als auch alle

Chunks in dichte Vektoren, deren Relevanz über die Kosinus-Ähnlichkeit bestimmt wird. Dieser Ansatz dient als Baseline für den Vergleich komplexerer Retrieval-Strategien.

Die Suche erfolgt mithilfe eines HNSW-Index (Hierarchical Navigable Small World) in der Vektordatenbank Qdrant. Für jede Anfrage liefert das System die Top-k Chunks mit der höchsten Ähnlichkeit. Der Prozess umfasst lediglich die Berechnung des Anfrage-Embeddings und die Ähnlichkeitssuche, wodurch die Methode hohe Verarbeitungsgeschwindigkeit bei geringem Ressourcenbedarf erreicht.

Der Ansatz nutzt ausschließlich semantische Relationen des Embedding-Modells ohne lexikalische Matching-Komponente.

Hybrid Retrieval

Hybrid Retrieval kombiniert semantische Vektorsuche mit klassischem lexikalischem Retrieval (BM25). Beide Verfahren laufen parallel, bevor *Reciprocal Rank Fusion* (RRF) ihre Ergebnislisten fusioniert. RRF gewichtet Treffer basierend auf dem Kehrwert ihrer Rangposition, wodurch keine Score-Normalisierung erforderlich ist.

Für jede Anfrage laufen Vektorsuche über den HNSW-Index und BM25-basierte Suche parallel. RRF fusioniert die Ranglisten, wobei das Verfahren Segmente bevorzugt, die in beiden Listen hohe Platzierungen aufweisen. Dadurch identifiziert die Strategie sowohl semantisch ähnliche als auch terminologisch exakte Treffer.

Die Strategie erfordert zwei Indexstrukturen (HNSW für Vektoren, invertierter Index für BM25) und führt beide Suchen parallel aus. Die durch RRF erzeugte Rangliste kombiniert die Positionen aus beiden Einzelsuchen.

Reranking-Komponente

Das System nutzt `cross-encoder/mmarco-mMiniLMv2-L12-H384-v1`[4], einen mehrsprachigen Cross-Encoder auf Basis eines distillierten MiniLM-Transformers mit zwölf Layern. Das Modell wurde auf dem MS-MARCO-Passage-Ranking-Datensatz mit über 530 000 Query–Dokument-Paaren trainiert und erzeugt für jedes Paar einen Relevanzscore. Eine Sigmoid-Funktion transformiert die rohen Logit-Werte zu normierten Scores im Bereich [0,1].

Ein Multiplikatorparameter `RERANK_TOP_K_MULTIPLIER` legt fest, wie viele Kandidaten aus dem Retrieval bezogen werden. Bei `MULTIPLIER=3` und k=5 ruft das System

[4](cross-encoder/mmarco-mMiniLMv2-L12-H384-v1: :https://huggingface.co/cross-encoder/mmarco-mMiniLMv2-L12-H384-v1)

zunächst 15 Kandidaten ab, die der Cross-Encoder anschließend neu bewertet und auf die fünf relevantesten reduziert. Das Reranking erfolgt lokal und GPU-beschleunigt.

Das Modul lässt sich zu beiden Basis-Strategien hinzufügen und erzeugt die erweiterten Varianten Dense+Reranking und Hybrid+Reranking.

Dense Retrieval mit Reranking

Diese Strategie ergänzt Dense Retrieval um das Reranking-Modul. Das System ruft zunächst $k \times$`MULTIPLIER` Kandidaten über den HNSW-Index ab (z. B. 15 bei k=5 und `MULTIPLIER`=3). Der Cross-Encoder bewertet diese Kandidaten anschließend neu, sortiert sie nach Relevanzscore und reduziert die Liste auf die k relevantesten Segmente. Das Verfahren operiert ausschließlich innerhalb der initial gefundenen Kandidaten und nutzt GPU-Beschleunigung für die Cross-Encoder-Inferenz.

Hybrid Retrieval mit Reranking

Diese Strategie kombiniert Hybrid Retrieval mit nachgelagertem Cross-Encoder-Reranking. Nach der RRF-Fusion der Dense- und BM25-Ergebnislisten bewertet der Cross-Encoder die $k \times$`MULTIPLIER` Kandidaten und sortiert sie nach Relevanz. Dadurch identifiziert die Strategie zunächst sowohl semantisch ähnliche als auch terminologisch exakte Treffer durch die hybride Suche, bevor der Cross-Encoder eine präzise Relevanzbewertung vornimmt. Das Verfahren erfordert beide Indexstrukturen sowie Cross-Encoder-Inferenz für alle Kandidaten.

6.3. Prompt-Design und Antwortgenerierung

Die Antwortgenerierung bildet die Abschlussstufe der Pipeline und verknüpft die im Retrieval bereitgestellten Kontextsegmente mit der Nutzeranfrage. Das Prompt Design folgt dem Prinzip der strikt kontextgebundenen Generierung. Das Modell antwortet ausschließlich auf Basis der übergebenen Dokumentsegmente und kommuniziert fehlende Evidenz ausdrücklich durch eine Nicht Antwort. Dadurch werden Halluzinationen begrenzt und die Vergleichbarkeit der Pipeline-Varianten erhöht.

Der Prompt besteht aus einer System-Nachricht mit klar definierten Arbeitsanweisungen und einer User-Nachricht, die Kontext und Frage enthält. Die Systeminstruktion verpflichtet das Modell, ausschließlich die bereitgestellten Kontextdokumente zu nutzen und bei unzureichenden Informationen dies explizit zu kommunizieren (Abstain-Verhalten). Zusätzlich wird das Modell angewiesen, „präzise, aber umfassend“

zu antworten und spezifische Quellen zu zitieren, wenn relevant. Die im Retrieval erzeugten Chunks werden als zusammenhängender Text ohne explizite Identifikatoren in den Prompt integriert.

Diese Prompt-Gestaltung zielt darauf ab, Halluzinationen zu minimieren (Faithfulness) und die Nachvollziehbarkeit der Antworten zu erhöhen. Die Anweisung „präzise, aber umfassend“ erzeugt jedoch ein Spannungsfeld: Das Modell tendiert dazu, alle verfügbaren Informationen aus den Chunks zu nutzen, was bei knappen Referenzantworten zu niedrigen Factual-Correctness-Werten führt (siehe Abschnitt 7.3.2).

Für alle Pipeline-Konfigurationen wird dasselbe generative Modell (Llama 3.1 8B Instruct) verwendet. Dies reduziert die Zahl möglicher Einflussfaktoren und stellt sicher, dass Qualitätsunterschiede primär aus Retrieval, Chunking und Embeddings resultieren. Das fest definierte Prompt-Format folgt der von Llama erwarteten system/user-Struktur und gewährleistet reproduzierbare und konsistent formatierte Ausgaben.

Die erzeugten Antworten werden zusammen mit den zitierten Identifikatoren persistiert und anschließend mit RAGAS bewertet. Die einheitliche Prompt-Struktur bildet damit die Grundlage für eine faire und methodisch belastbare Gegenüberstellung sämtlicher Pipeline-Varianten.

Listing 6.1: System Prompt

```
system_prompt = """Du bist ein hilfreicher Assistent, der Fragen basierend
    auf den bereitgestellten Kontextdokumenten beantwortet.
Anweisungen:
- Verwende nur die Informationen aus den bereitgestellten Kontextdokumenten,
     um die Frage zu beantworten
- Wenn der Kontext nicht gengend Informationen enthlt, um die Frage zu
    beantworten, sage dies klar und deutlich
- Sei przise, aber umfassend in deiner Antwort
- Zitiere spezifische Quellen, wenn relevant
- Wenn mehrere Dokumente relevante Informationen enthalten, fasse die
    Informationen angemessen zusammen"""
user_prompt = f"""Kontextdokumente: {context}
Frage: {query}
Bitte gib eine umfassende Antwort basierend auf den oben genannten
    Kontextdokumenten."""
"""
```

7. Evaluation

7.1. Überblick über die Experimente

Die experimentelle Evaluation umfasste 56 Pipeline-Konfigurationen aus den Komponenten Parser, Chunking-Strategie, Embedding-Modell und Retrieval-Variante. Die Reduktion von theoretisch 96 auf 56 Kombinationen resultierte aus technischen Inkompatibilitäten des Unstructured-Parsers mit den verschiedenen Chunking-Strategien. Der Evaluationskorpus umfasste 46 Golden-Standard-Fragen mit zugehörigen Referenzantworten.

Das zweiphasige Evaluationsverfahren filterte in Phase 1 alle 56 Konfigurationen anhand von Semantic Similarity und Retrieval-Score vor. Für Phase 2 wurden 12 Konfigurationen ausgewählt, die den gesamten Designraum systematisch abdecken.

Phase 2 bewertet die 12 Konfigurationen anhand der vollständigen RAGAS-Metriken (Faithfulness, Context Precision, Factual Correctness, Semantic Similarity). Die Ergebnisse werden im Folgenden anhand von Tabellen dargestellt und interpretiert.

7.2. Quantitative Ergebnisse

Im Folgenden werden die Ergebnisse der Phase-2-Evaluation dargestellt, die auf den vier RAGAS-Metriken Faithfulness, Context Precision, Factual Correctness und Semantic Similarity basiert. Insgesamt wurden zwölf Pipeline-Konfigurationen vollständig bewertet. Die Auswertung erfolgt komponentenweise, wobei jeweils Mittelwerte der Metriken betrachtet und Unterschiede zwischen Parsern, Chunking-Verfahren, Embedding-Modellen und Retrieval-Strategien beschrieben werden.

7.2.1. Gesamtüberblick über die Pipeline-Leistungen

Die Analyse der experimentellen Ergebnisse verdeutlicht Leistungsunterschiede zwischen den evaluierten Pipeline-Konfigurationen. Den höchsten kombinierten Score (0.6795) erreicht die Konfiguration aus Docling-Extraktion, LangChain-Chunking,

Nomic-Embeddings und Dense Retrieval mit Cross-Encoder-Reranking. Dieses Ergebnis zeigt, dass offene, lokal ausführbare Komponenten Premium-Cloud-Lösungen übertreffen können, wenn sie mit effektiven Reranking-Strategien kombiniert werden.

Die drei bestplatzierten Pipelines sind in Tabelle 7.7 zusammengefasst.

Tabelle 7.1.: Top-3-Pipeline-Konfigurationen nach Gesamtscore

Rang	**Pipeline**	**Gesamtscore**
1	`doc_lan_nomi_den_rnk`	0.6795
2	`uns_nat_nomi_den`	0.6603
3	`doc_lan_open_den`	0.6588

Die schwächste Konfiguration mit Docling, semantischem Chunking, Nomic und Dense Retrieval erreicht einen Score von 0.5605. Embeddings-basiertes semantisches Chunking erzielt damit niedrigere Werte als strukturorientierte Ansätze wie Header-basiertes Splitting, obwohl es theoretische Vorteile bei der semantischen Kohärenz bietet.

Cross Encoder Reranking stellt die wirkungsvollste Einzeloptimierung dar. Die Basiskonfiguration Docling, LangChain und Nomic erreicht ohne Reranking einen Score von 0.6168 und liegt damit auf Rang 9. Mit aktiviertem Reranking steigt der Score auf 0.6795 und damit auf Rang 1, was einer Verbesserung um 10,2 Prozent entspricht. Diese Steigerung übertrifft den Einfluss eines Wechsels zu Premium Embeddings oder zu cloudbasierten Parsingdiensten deutlich.

7.2.2. Vergleich der Parser

Die Analyse der drei Parser verdeutlicht Unterschiede in der Extraktionsqualität und deren Auswirkung auf die nachgelagerten RAG-Komponenten. Docling erreicht über sieben getestete Konfigurationen hinweg die höchsten Mittelwerte und nimmt konsistent die vorderen Platzierungen ein, wobei sich insbesondere die Dominanz in vier von fünf Metriken manifestiert. Unstructured weist in optimierter Konfiguration ebenfalls stabile und vergleichsweise hohe Kennzahlen auf, bleibt jedoch im Gesamtscore geringfügig hinter Docling zurück. LlamaParse erzielt demgegenüber die höchste Faithfulness aller Parser, erreicht aber keine Top-Platzierung im Gesamtvergleich. Diese Ergebnisse verdeutlichen die Leistungsüberlegenheit von Docling, während Unstructured und LlamaParse jeweils spezifische metrische Stärken aufweisen.

Tabelle 7.2.: Vergleich der Parser über alle zugehörigen Pipelines

Parser	Bester Score	Durchschnitt
Docling	0.6795	0.6398
Unstructured	0.6603	0.6603
LlamaParse	0.6520	0.6354

7.2.3. Vergleich der Chunking-Verfahren

Die Evaluation der vier untersuchten Chunking-Methoden offenbart deutliche Leistungsunterschiede zwischen strukturorientierten und semantischen Ansätzen. LangChain erreicht mit header-bewusstem Chunking die höchste Gesamtperformance und dominiert den Vergleich, während die native Chunking-Implementierung von Unstructured eine überraschend starke Leistung aufweist und sich auf dem zweiten Rang positioniert. Semantisches Chunking bleibt demgegenüber deutlich hinter den strukturorientierten Verfahren zurück und erzielt selbst mit proprietären Embeddingmodellen keine wettbewerbsfähigen Ergebnisse. Chonkie weist respektable Kennzahlen auf, bietet jedoch keinen erkennbaren Vorteil gegenüber dem einfacheren LangChain-Ansatz. Diese Ergebnisse deuten darauf hin, dass natürliche Themengrenzen nicht notwendigerweise mit den Anforderungen des Q&A-Retrievals übereinstimmen und strukturorientierte Strategien im untersuchten Evaluationsrahmen konsistent überlegen sind.

Tabelle 7.3.: Vergleich der Chunking-Methoden in der Best-Performance-Konfiguration

Chunking-Methode	Bester Score	Ranking	Performance-Gap
LangChain	0.6795	1	Baseline
Native (Unstructured)	0.6603	2	-2.8%
Chonkie	0.6390	6	-6.0%
Semantic	0.6365	7	-6.3%

7.2.4. Vergleich der Embedding-Modelle

Die Analyse der beiden Embedding-Verfahren offenbart Leistungsunterschiede. Die lokal ausgeführten Nomic-v1.5-Embeddings erzielen sowohl hinsichtlich der Bestwerte als auch der durchschnittlichen Performance überlegene Ergebnisse und belegen vier der fünf vorderen Platzierungen. Die auf text-embedding-3-small basierenden OpenAI-Embeddings erreichen hingegen geringfügig niedrigere Mittelwerte, weisen jedoch eine etwas geringere Ergebnisstreuung auf. Die besten Nomic-Konfigurationen

liegen im Bereich von 0.65 bis 0.68, wobei insbesondere die Kombination mit Cross-Encoder-Reranking (0.6795) die OpenAI-Konfigurationen (bis 0.6588) übertrifft. Damit bestätigen die Ergebnisse die höhere Leistungsfähigkeit des Nomic-Embedding-Modells im gegebenen Evaluationsrahmen, insbesondere in Verbindung mit nachgelagertem Reranking.

Tabelle 7.4.: Vergleich der Embedding-Modelle über alle zugehörigen Pipelines

Embedding-Modell	**Bester Score**	**Durchschnitt**	**Top-5-Platzierungen**
Nomic v1.5 (lokal)	0.6795	0.6328	4
OpenAI (text-embedding-3-small)	0.6588	0.6282	1

7.2.5. Vergleich der Retrieval- und Reranking-Strategien

Die vier untersuchten Retrieval-Strategien offenbaren eine klare Hierarchie, die von der intuitiven Annahme abweicht, dass komplexere Ansätze automatisch bessere Ergebnisse liefern. Dense Retrieval mit Reranking erzielt den höchsten Gesamtscore (0.6795) und demonstriert damit die Überlegenheit semantischer Suche kombiniert mit kontextueller Neugewichtung durch Cross-Encoder.

Reines Dense Retrieval ohne Reranking zeigt eine hohe Varianz von 0.5605 bis 0.6603. Die beste Konfiguration mit Unstructured, native Chunking und Nomic erreicht einen Score von 0.6603 und liegt damit nur 2,8 Prozent hinter dem Gesamtsieger. Dies weist darauf hin, dass bei einer optimalen Chunkingstrategie die initiale semantische Retrievalqualität bereits ein sehr hohes Niveau erreicht.

Hybridansätze, die BM25 mit Dense Embeddings kombinieren, erreichen durchgängig niedrigere Scores mit einem Durchschnitt von 0.6118 und erzielen in der reinen Hybridvariante keine einzige Platzierung unter den besten fünf. Die Addition von Reranking zu Hybrid-Retrieval verbessert die Performance signifikant auf 0.6585 (Platz 4), bleibt jedoch hinter der Dense+Reranking-Kombination zurück.

Der Vergleich zeigt, dass Reranking als nachgelagerte Komponente einen größeren Einfluss ausübt (+10,2% Verbesserung) als die Wahl zwischen Dense und Hybrid als Basis-Retrieval-Strategie (+2,2%). Dies legt nahe, dass für den evaluierten Kontext (deutschsprachige C-Programmierung-Dokumentation) die lexikalische BM25-Komponente keinen signifikanten Mehrwert bietet, möglicherweise aufgrund der hohen semantischen Konsistenz technischer Fachbegriffe.

Die Evaluation zeigt damit, dass die Kombination aus semantischem Retrieval und Cross-Encoder-Reranking den State-of-the-Art für technische Dokumentation darstellt,

Tabelle 7.5.: Vergleich der Retrieval-Strategien (Nov 25, 2025)

Retrieval-Strategie	Bester Score	ØScore	Beobachtung
Dense + Reranking	**0.6795**	0.6492	Beste Gesamtperformance, +10,2% vs. Dense
Dense	0.6603	0.6312	Hohe Varianz (0.56–0.66), stabile Basis
Hybrid + Reranking	0.6585	0.6585	Solide, aber hinter Dense+Reranking
Hybrid (BM25 + Dense)	0.6365	0.6118	Keine Top-5-Platzierung ohne Reranking

während lexikalische Komponenten (BM25) keinen messbaren Vorteil bieten und die Systemkomplexität unnötig erhöhen.

7.3. Qualitative Fehleranalyse

Die quantitativen Ergebnisse aus Kapitel 7.2 zeigen Leistungsunterschiede zwischen den Pipeline-Komponenten. Um diese Befunde nachvollziehbarer zu machen, werden im Folgenden typische Fehlermuster exemplarisch beschrieben, die während der Evaluation beobachtet wurden. Die Analyse basiert auf Stichproben der generierten Antworten der leistungsstärksten und leistungsschwächsten Konfigurationen.was

7.3.1. Halluzination

Ein typisches Fehlerbild in den untersuchten RAG-Konfigurationen ist das Auftreten irrelevanter Kontextauswahl, selbst bei eindeutig formulierten Benutzeranfragen. Ein ausgeprägter Fall findet sich bei der Testanfrage „Was bedeutet `\n`?“, die in 10 von 12 Pipelines (83 %) zu vollständig irrelevanten Chunks führte. Die Pipeline `doc_lan_nomi_den`, die Header-basiertes Chunking mit dem Docling-Parser und Nomic-Embeddings kombiniert, erreicht bei dieser Anfrage eine Context Precision von 0,00 und eine Faithfulness von 0,40. Das System identifiziert keinen relevanten Kontext, wobei das Sprachmodell defensiv reagiert und den fehlenden Kontext explizit kommuniziert.

Kontextlage und Antwortverhalten

Die Pipeline extrahiert für diese Anfrage ausschließlich irrelevante Chunks. Eine Tabelle arithmetischer Operatoren (+, -, *, /, %), eine Übersicht logischer Operatoren (!, &&, ||) sowie relationale Operatoren (<, >, ==, !=). Diese Segmente weisen keinerlei thematische Beziehung zur Escape-Sequenz `\n` auf. Der Retrieval-Schritt verfehlt somit vollständig die für die Beantwortung notwendigen Inhalte zur String-Formatierung oder Ausgabesteuerung in C.

Das Sprachmodell reagiert auf den irrelevanten Kontext defensiv und kommuniziert explizit: „Leider kann ich diese Frage nicht beantworten, da der Kontext keine Informationen über den Operator `\n` enthält.“ Anschließend ergänzt das Modell aus eigenem Wissen: „Der Operator `\n` ist ein spezieller Zeilenumbruch-Operator in Programmiersprachen wie C oder Python.“ Diese Zusatzinformation ist faktisch korrekt, jedoch nicht durch die gelieferten Dokumentsegmente gestützt. RAGAS bewertet dies mit einer Faithfulness von 0,40, da 60 % der Antwort nicht aus dem Kontext ableitbar sind. Das Modell halluziniert somit nicht wild, sondern nutzt sein internes Weltwissen, um trotz fehlendem Kontext eine hilfreiche Antwort zu konstruieren.

Strukturelle Darstellung des Fehlerfalls

Zur Verdeutlichung des Ablaufs wird der Fehlerfall in einer kompakten, JSON-ähnlichen Darstellung zusammengefasst:

Listing 7.1: Verkürzte Darstellung der Chunks

```
{
  "id": "q7",
  "question": "Was bedeutet \\n?",
  "retrieved_chunks": [
    "## Arithmetische Operatoren\n| + | Addition |\n| - | Subtraktion |\n..."
        ,
    "## Logische Operatoren\n| ! | logische Negation |\n| && | logisches Und
        |\n...",
    "## Relationale Operatoren\n| < | Kleiner |\n| <= | Kleiner gleich |\n...
        "
  ],
  "model_answer": "Der Operator ‘\\n‘ steht fr einen Zeilenumbruch, ist
      jedoch nicht im Kontext enthalten.",
  "ground_truth": "\\n ist eine Escape-Sequenz fr einen Zeilenumbruch in C.",

  "metrics": {
    "faithfulness": 0.40,
    "context_precision": 0.00,
```

```
    "factual_correctness": 0.44,
    "semantic_similarity": 0.58
  },
  "avg_retrieval_score": 0.6119,
  "pipeline": "doc_lan_nomi_den"
}
```


Ursachenanalyse

Die Fehlerursache liegt primär in der Tokenisierung der Embedding-Komponente. Das Nomic-Embedding-Modell splittet die Escape-Sequenz `\n` in separate Tokens (\ und n), wodurch der semantische Kontext verloren geht. Die extrahierten Operatoren-Tabellen zeigen eine strukturelle Ähnlichkeit zur Anfrage „Was bedeutet [X]?“: Beide enthalten das Muster „Operator | Bedeutung“. Das Embedding-Modell erkennt diese strukturelle Übereinstimmung, nicht jedoch den semantischen Unterschied zwischen arithmetischen Operatoren und String-Escape-Sequenzen.

Die durchschnittlichen Retrieval-Scores von 0,6119 suggerieren hohe Relevanz, während die Context Precision von 0,00 zeigt, dass alle Chunks tatsächlich irrelevant sind. Dies verdeutlicht eine fundamentale Limitation embedding-basierter Retrieval-Systeme: Hohe Similarity-Scores garantieren keine inhaltliche Relevanz.

Einordnung im Gesamtbild

Der vorliegende Fall illustriert ein systematisches Architekturproblem: 10 von 12 Pipelines versagen bei dieser Anfrage mit Context Precision, unabhängig von Chunking-Strategie, Retrieval-Methode oder Embedding-Modell. Das Problem liegt nicht in einzelnen Komponentenwahlen, sondern in einer fundamentalen Limitation embedding-basierter Systeme bei syntaktischen Konstrukten wie Escape-Sequenzen oder Sonderzeichen.

Das LLM vermeidet Halluzinationen, indem es fehlenden Kontext explizit kommuniziert und seine Ausgaben auf faktisch korrekte Informationen beschränkt. Die Diskrepanz zwischen hohen Retrieval-Scores (0,6119) und niedriger Context Precision (0,00) zeigt, dass Similarity-Scores keine verlässliche Relevanzmetrik darstellen.

7.3.2. Relevante, aber unvollständige Antwort

Ein weiteres charakteristisches Fehlerbild betrifft Antworten, die perfekt am bereitgestellten Kontext orientiert sind, jedoch von der knappen Referenzlösung in Länge und Detail abweichen. Dieses Verhalten zeigt sich durch hohe Werte in Context Precision (1,0) und Faithfulness (1,0) bei gleichzeitig niedriger Factual Correctness (0,40). Der folgende Fall stammt aus der Pipeline `doc_lan_nomi_den_rnk`, die in der Gesamtbewertung den ersten Platz erreicht (Composite Score: 0,6795).

Fallbeschreibung

Für die Testanfrage „Was ist ein Statementblock?“ (q9) erreicht die Pipeline perfekte Werte in Context Precision (1,00) und Faithfulness (1,00), während die Factual Correctness mit 0,40 deutlich darunter liegt. Der Retrievalschritt identifiziert die relevanten Textstellen fehlerfrei. Chunk 1 enthält die vollständige Definition eines Statementblocks einschließlich der Syntax mit geschweiften Klammern, der Verwendung in Funktionsblöcken sowie einer detaillierten Erklärung des Begriffs Statement. Das Sprachmodell bleibt vollständig beim bereitgestellten Kontext und generiert keine Inhalte außerhalb der Evidenz.

Die generierte Antwort umfasst 512 Zeichen und erklärt neben der Kerndefinition auch die drei Typen von Statements sowie die Semikolon-Regel. Die Referenzantwort hingegen beschränkt sich auf 60 Zeichen: „Eine Gruppierung von Statements in geschweiften Klammern {}.“ Die niedrige Factual Correctness resultiert somit nicht aus fehlenden oder falschen Informationen, sondern aus der Diskrepanz zwischen einer knappen Referenzantwort und einer ausführlichen, aber vollständig korrekten Modellantwort.

Listing 7.2: Evaluationsinstanz q29: Relevante, aber unvollständige Antwort

```
{
  "id": "q9",
  "question": "Was ist ein Statementblock?",
  "retrieved_chunks": [
    "## Aufbau eines C-Programms\nEine Funktionsdefinition besteht aus
        Rckgabetyp, Name, Argumentliste und dem Funktionsblock. Der
        Funktionsblock wird durch { } eingeschlossen und stellt einen
        Statementblock dar  eine Gruppierung von Statements (ausfhrbare
        Anweisungen, Definitionen oder Deklarationen). Jedes Statement endet
        mit einem Semikolon.",
    "## Vorsicht\nFall-through in switch-Anweisungen: Wird kein break gesetzt
        , werden alle folgenden Anweisungen ausgefhrt. Fall-through sollte
        vermieden oder dokumentiert werden.",
```

```
    "## Ausfhrung der switch-Anweisung\nEin Sprung erfolgt zum passenden case
        -Label; ohne passende Konstante wird hinter dem Anweisungsblock
        fortgesetzt."
  ],
  "model_answer": "Ein Statementblock in C ist eine durch geschweifte
      Klammern { } eingeschlossene Gruppierung von Statements.",
  "ground_truth": "Eine Gruppierung von Statements in geschweiften Klammern {
       }.",
  "metrics": {
    "faithfulness": 1.00,
    "context_precision": 1.00,
    "factual_correctness": 0.40,
    "semantic_similarity": 0.77
  },
  "avg_retrieval_score": 0.7159,
  "answer_length": 512,
  "ground_truth_length": 60,
  "pipeline": "doc_lan_nomi_den_rnk"
}
```


Analyse des Antwortverhaltens

Der Retrieval-Schritt extrahiert drei Chunks, wobei Chunk 1 die vollständige Definition eines Statementblocks enthält. Dieser Chunk beschreibt den Funktionsblock in C, erklärt die Syntax (geschweifte Klammern), definiert einen Statementblock als Gruppierung von Statements und führt anschließend aus, dass ein Statement entweder eine ausführbare Anweisung, eine Definition oder eine Deklaration sein kann. Zusätzlich wird die Semikolon-Regel erwähnt. Chunk 2 und Chunk 3 behandeln Anweisungsblöcke im Kontext von `switch`-Statements und sind tangential relevant.

Sowohl die Modellantwort als auch die Referenzantwort enthalten die Kernelemente „Gruppierung von Statements“ und „geschweifte Klammern“. Die generierte Antwort fügt jedoch zusätzlich Informationen über die drei Statement-Typen, den Funktionsblock-Kontext und die Semikolon-Regel hinzu. Diese Zusatzinformationen sind vollständig durch Chunk 1 gestützt (Faithfulness: 1,00) und faktisch korrekt. Die niedrige Factual Correctness (0,40) resultiert daraus, dass RAGAS die Ähnlichkeit zur Referenz misst. Die generierte Antwort ist 8,5-mal länger (512 vs. 60 Zeichen) und weicht durch ihre Ausführlichkeit von der knappen Referenzformulierung ab.

Dieser Fall verdeutlicht eine fundamentale Limitation der Evaluationsmethodik. Die Referenzantworten des Golden Standards wurden bewusst knapp formuliert, um eine eindeutige Vergleichsbasis zu schaffen. RAGAS Factual Correctness bewertet jedoch

primär die Übereinstimmung mit dieser Referenz, nicht die inhaltliche Qualität oder Nützlichkeit der Antwort. Eine ausführlichere Antwort, die zusätzlichen, kontextgestützten Kontext liefert, wird durch diese Metrik bestraft, obwohl sie aus Nutzersicht informativer ist. Die Diskrepanz zwischen perfekter Kontexttreue (Context Precision: 1,00, Faithfulness: 1,00) und niedriger Factual Correctness zeigt, dass hohe Werte in Retrieval- und Grounding-Metriken keine hohe Referenz-Ähnlichkeit garantieren. Dies unterstreicht die Notwendigkeit, RAGAS-Metriken kontextuell zu interpretieren und zwischen technischer Korrektheit und evaluationsspezifischen Bewertungskriterien zu differenzieren.

Einordnung im Gesamtbild

Der vorliegende Fall illustriert ein systematisches Muster bei Definitionsfragen: 8 von 12 Pipelines (67 %) erreichen bei dieser Anfrage Context Precision von 1,00 und Faithfulness über 0,95, während die Factual Correctness zwischen 0,31 und 0,40 liegt. Das Problem betrifft somit nicht einzelne Pipeline-Konfigurationen, sondern das Zusammenspiel von perfektem Retrieval, vollständigen Chunks und einem Sprachmodell, das alle verfügbaren Informationen nutzt.

Im Gegensatz zum zuvor diskutierten Fall irrelevanter Chunks (\n-Beispiel) liegt hier kein Retrieval-Fehler vor. Die niedrige Factual Correctness resultiert nicht aus fehlenden oder falschen Informationen, sondern aus der Diskrepanz zwischen knappen Evaluations-Referenzen und ausführlichen, aber korrekten Modellantworten. Dies verdeutlicht, dass identische Factual-Correctness-Werte unterschiedliche Ursachen haben können.

Für produktive RAG-Systeme im Hochschulkontext ergibt sich daraus ein Trade-off. Kurze, referenzähnliche Antworten erzielen hohe Factual-Correctness-Werte in der Evaluation, während ausführlichere Antworten mit zusätzlichem Kontext die Lerneffizienz erhöhen können. Die Lösung liegt in adaptiven Strategien wie Query-Klassifikation oder nutzerdefinierbaren Antwortmodi. Die Ergebnisse zeigen, dass technisch perfekte Pipelines (CP = 1,00, Faith = 1,00) nicht automatisch optimale Factual Correctness erreichen, was die Notwendigkeit unterstreicht, Metriken im Kontext ihrer Evaluationsdesigns zu interpretieren.

7.4. Zusammenfassende Bewertung der Pipeline-Varianten

Die systematische Evaluation von 56 Pipeline-Konfigurationen ermöglicht eine fundierte Bewertung der untersuchten RAG-Komponenten sowie deren Zusammenwirken. Im

Folgenden werden die quantitativen Befunde aus Abschnitt 7.2 und die qualitativen Fehleranalysen aus Abschnitt 7.3 zu einer Gesamtbewertung synthetisiert.

7.4.1. Überblick über die Leistungsverteilung

Die systematische Evaluation der 12 ausgewählten Pipeline-Konfigurationen zeigt eine Spannweite von 0,5523 bis 0,6795 im Composite Score (Durchschnitt: 0,6288, Standardabweichung: 0,0354). Die Leistungsverteilung verdeutlicht, dass Komponentenwahl und deren Zusammenspiel einen messbaren Einfluss auf die Antwortqualität haben: Die beste Pipeline (`doc_lan_nomi_den_rnk`) übertrifft die schlechteste (`doc_sem_nomi_den`) um 21,1 %.

Die Top-3-Pipelines liegen eng beieinander (0,6588–0,6795, Differenz: 3,1 %), während ein deutlicher Leistungsabfall ab Platz 8 erkennbar ist. Tabelle 7.6 zeigt das vollständige Ranking mit den jeweiligen Composite Scores sowie den dominierenden Metriken je Pipeline.

Tabelle 7.6.: Vollständiges Ranking der evaluierten Pipeline-Konfigurationen

Rang	**Pipeline-ID**	**Composite**	**Query-Zeit**	**Stärke**
1	doc_lan_nomi_den_rnk	0,6795	3558 ms	Context Precision
2	uns_nat_nomi_den	0,6603	3651 ms	Faithfulness
3	doc_lan_open_den	0,6588	3443 ms	Faithfulness
4	doc_lan_nomi_hyb_rnk	0,6585	3384 ms	Context Precision
5	lla_lan_nomi_den	0,6520	3576 ms	Faithfulness
6	doc_cho_nomi_den	0,6390	3768 ms	Context Precision
7	doc_sem_open_hyb	0,6365	3964 ms	Context Precision
8	lla_sem_nomi_den_rnk	0,6188	4427 ms	Context Precision
9	doc_lan_nomi_den	0,6168	2734 ms	Faithfulness
10	doc_lan_nomi_hyb	0,6096	2610 ms	Semantic Similarity
11	doc_lan_open_hyb	0,5783	2534 ms	Semantic Similarity
12	doc_sem_nomi_den	0,5523	3964 ms	Semantic Similarity

Die Analyse der Einzelmetriken zeigt differenzierte Stärken: Context Precision weist die höchste Varianz auf (0,5261–0,8351, Standardabweichung: 0,0880), während Semantic Similarity am stabilsten ist (0,6101–0,6486, Standardabweichung: 0,0124). Dies deutet darauf hin, dass Retrieval-Qualität stärker von der Pipeline-Konfiguration abhängt als die semantische Ähnlichkeit der Antworten. Die beste Pipeline erreicht in Context Precision (0,8312) einen um 58,1 % höheren Wert als die schlechteste (0,5261), was die Bedeutung dieser Metrik für die Gesamtperformance unterstreicht.

7.4.2. Einfluss einzelner Komponenten

Die systematische Variation von Parser, Chunking-Strategie, Embedding-Modell und Retrieval-Methode ermöglicht die Isolierung des jeweiligen Einflusses auf die Gesamtperformance. Die folgende Analyse vergleicht die Komponenten anhand der durchschnittlichen Composite Scores ihrer zugehörigen Pipelines.

Parser-Vergleich

Docling erreicht über neun Konfigurationen einen Durchschnitt von 0,6255 mit einer Spannweite von 0,5523 bis 0,6795 (Varianz: 23 %). Diese hohe Varianz zeigt, dass Docling stark von nachgelagerten Komponenten abhängt. Die beste Docling-Pipeline (Platz 1) nutzt Reranking, während die schlechteste (Platz 12) semantisches Chunking einsetzt. LlamaParse erreicht über zwei Konfigurationen einen Durchschnitt von 0,6354, platziert sich jedoch nur im Mittelfeld (Platz 5 und 8). Unstructured erreicht mit einer einzigen Konfiguration den zweiten Platz (0,6603) und stellt damit die beste Pipeline ohne Reranking dar.

Chunking-Strategien

Header-basiertes Chunking (LangChain) erreicht über sieben Pipelines einen Durchschnitt von 0,6362 und dominiert die Top-3-Platzierungen. Die Strategie segmentiert Dokumente entlang von Markdown-Überschriften mit fixen Chunk-Größen (750 Tokens, 150 Overlap) und erzeugt strukturell kohärente Segmente. Native Chunking (Unstructured) erreicht 0,6603 (Platz 2) ohne explizite Segmentierungslogik, während adaptive Chunking-Strategien (Chonkie, Platz 6: 0,6390) keinen Vorteil gegenüber fixen Größen zeigen. Semantisches Chunking schneidet mit einem Durchschnitt von 0,6025 über drei Pipelines am schlechtesten ab. Die embedding-basierte Segmentierung erzeugt variable Chunk-Größen, die bei technischen Texten zu inkonsistenten Grenzen führen und relevante Informationen fragmentieren.

Embedding-Modelle

Nomic v1.5 erreicht über neun Pipelines einen Durchschnitt von 0,6319, während OpenAI text-embedding-3-small über drei Pipelines 0,6245 erreicht. Der Unterschied von +1,2 % zugunsten von Nomic ist marginal, jedoch nutzt die beste Pipeline (Platz 1) Nomic-Embeddings. OpenAI-Embeddings erzielen die beste Platzierung ohne Reranking (Platz 3: 0,6588), bleiben jedoch hinter der Kombination Nomic + Reranking

zurück (-3,1 %). Die Evaluation zeigt, dass lokale Embeddings für technische Lehrmaterialien ausreichend sind und Cloud-Embeddings keinen signifikanten Mehrwert bieten.

Retrieval-Strategien

Dense Retrieval mit Reranking erreicht über zwei Pipelines den höchsten Durchschnitt (0,6492) und stellt die effektivste Strategie dar. Der direkte Vergleich der Basis-Pipeline `doc_lan_nomi_den` (0,6168, Platz 9) mit ihrer Reranking-Variante `doc_lan_nomi_den_rnk` (0,6795, Platz 1) zeigt eine Verbesserung von +10,2 %. Reranking verbessert insbesondere Context Precision (+18,2 %) und Factual Correctness (+25,9 %), während Faithfulness und Semantic Similarity stabil bleiben. Dense Retrieval ohne Reranking erreicht über sechs Pipelines 0,6299 und ist damit die zweitbeste Strategie. Hybrid Retrieval (Dense + BM25) schneidet mit einem Durchschnitt von 0,6081 am schlechtesten ab und bietet keinen Mehrwert gegenüber reiner semantischer Suche. Die lexikalische Komponente (BM25) verbessert die Performance bei technischen Texten nicht, da diese semantisch konsistent und gut strukturiert sind.

Dominante Einflussfaktoren

Die quantitative Analyse zeigt, dass Reranking den größten Einzeleinfluss auf die Performance hat (+10,2 %), gefolgt von der Chunking-Strategie (Header-based vs. Semantic: +5,6 %) und der Parser-Wahl (Unstructured vs. LlamaParse: +3,9 %). Die Wahl des Embedding-Modells hat den geringsten Einfluss (+1,2 %). Diese Erkenntnis verdeutlicht, dass investierte Ressourcen primär in Reranking-Komponenten und robuste Chunking-Strategien fließen sollten, während die Wahl zwischen lokalen und Cloud-basierten Embeddings nachrangig ist.

7.4.3. Beste Konfiguration und Begründung

Die Pipeline `doc_lan_nomi_den_rnk` erreicht mit einem Composite Score von 0,6795 die höchste Gesamtperformance und übertrifft die zweitplatzierte Konfiguration um 2,8 %. Die Konfiguration kombiniert Docling als lokalen Parser, Header-basiertes LangChain-Chunking (750 Tokens, 150 Overlap), Nomic v1.5-Embeddings und Dense Retrieval mit Cross-Encoder-Reranking. Alle Komponenten laufen lokal, wodurch die Pipeline vollständig datenschutzkonform betrieben werden kann und keine laufenden API-Kosten verursacht.

Metrik-Profile der Top-3-Pipelines

Die Analyse der Einzelmetriken verdeutlicht die spezifischen Stärken der führenden Konfigurationen. Tabelle 7.7 zeigt die detaillierten Metrik-Profile der Top-3-Pipelines.

Tabelle 7.7.: Metrik-Profile der drei leistungsstärksten Pipeline-Konfigurationen

Metrik	**Platz 1** `doc_lan_nomi_den_rnk`	**Platz 2** `uns_nat_nomi_den`	**Platz 3** `doc_lan_open_den`
Composite Score	0,6795	0,6603	0,6588
Faithfulness	0,7422	**0,7462**	0,7623
Context Precision	**0,8312**	0,7453	0,7301
Factual Correctness	0,5015	**0,5061**	0,4959
Semantic Similarity	0,6430	**0,6436**	0,6467
Query-Zeit (ms)	3558	3651	**3443**
Kosten pro Query	**0,00**	**0,00**	∼0,001 $

Die beste Pipeline führt in Context Precision (0,8312) mit einem Vorsprung von +11,5 % gegenüber Platz 2. Dies zeigt, dass das Reranking-Modul relevante Chunks präziser identifiziert als reines Dense Retrieval. Unstructured (Platz 2) erreicht die höchsten Werte in Faithfulness (0,7462), Factual Correctness (0,5061) und Semantic Similarity (0,6436), verliert jedoch bei der Retrieval-Präzision. OpenAI-Embeddings (Platz 3) bieten die schnellste Query-Zeit (3443 ms), bleiben aber im Composite Score hinter lokalen Lösungen zurück.

Begründung der Überlegenheit

Die Überlegenheit von `doc_lan_nomi_den_rnk` basiert auf vier Faktoren. Erstens ermöglicht das Cross-Encoder-Reranking eine präzisere Relevanzbewertung der initial gefundenen Chunks: Der direkte Vergleich mit der Basis-Variante `doc_lan_nomi_den` (Platz 9, Score: 0,6168) zeigt eine Verbesserung von +10,2 % durch Reranking allein. Die Context Precision steigt dabei um +18,2 % (0,7029 → 0,8312) und die Factual Correctness um +25,9 % (0,3983 → 0,5015). Zweitens erzeugt das Header-basierte Chunking strukturell kohärente Segmente, die thematisch in sich geschlossen sind und relevante Informationen nicht fragmentieren. Drittens liefert Docling eine saubere Dokumentstruktur mit erhaltenen Markdown-Elementen, die das Chunking unterstützen. Viertens zeigt die Evaluation, dass Nomic-Embeddings in Kombination mit Reranking ausreichen und teurere Cloud-Embeddings keinen Mehrwert bieten.

Skalierbarkeit und Deployment

Die vollständig lokale Architektur ermöglicht den datenschutzkonformen Betrieb ohne externe Datenübertragung. Docling und Nomic-Embeddings benötigen zusammen etwa 3 GB RAM, der Cross-Encoder weitere 500 MB. Eine GPU beschleunigt das Reranking, ist jedoch nicht zwingend erforderlich: Auf CPU-only-Systemen verlängert sich die Query-Zeit um etwa 30 %, was die Performance auf 4600 ms erhöht (vergleichbar mit Platz 8). Für produktive Deployments mit hohem Query-Volumen ist eine GPU-Beschleunigung empfehlenswert, da der Durchsatz um Faktor 3 steigt.

7.5. Schlussfolgerungen

Die vorliegende Arbeit entwickelte und evaluierte ein datenschutzkonformes Retrieval-Augmented Generation System für den Hochschulkontext. Die systematische Untersuchung von 56 Pipeline-Konfigurationen zeigt, dass lokale Open-Source-Komponenten Cloud-basierte Alternativen in Performance, Kosten und Datenschutz übertreffen.

Die quantitative Evaluation identifiziert Reranking als wichtigsten Einflussfaktor (+10,2 %), während die Wahl des Embedding-Modells nachrangig ist (+1,2 %). Die beste Konfiguration (`doc_lan_nomi_den_rnk`) kombiniert Docling, Header-basiertes Chunking, Nomic-Embeddings und Cross-Encoder-Reranking, erreicht einen Composite Score von 0,6795 und verursacht keine laufenden Kosten. Hybrid Retrieval und semantisches Chunking zeigen bei technischen Lehrmaterialien keine Vorteile.

Die qualitative Fehleranalyse verdeutlicht systematische Limitationen: Syntaktische Konstrukte führen bei 67 % der Pipelines zu irrelevantem Retrieval, während 42 % bei Definitionsfragen zu ausführliche Antworten generieren. Dies unterstreicht die Notwendigkeit kontextueller Metrik-Interpretation: Context Precision von 0,00 signalisiert Retrieval-Versagen, während Context Precision von 1,00 bei niedriger Factual Correctness auf ein Prompt-Engineering-Problem hinweist.

Für produktive RAG-Systeme im Hochschulkontext eignet sich die evaluierte Kombination aus lokalen Komponenten mit Reranking. Die Evaluation zeigt, dass technisch hochwertige RAG-Systeme ohne Cloud-Abhängigkeiten realisierbar sind und vollständige Datensouveränität gewährleisten. Zukünftige Arbeiten sollten größere Datensätze, Query-Klassifikation zur adaptiven Antwortlänge und spezialisierte Code-Embeddings untersuchen.

8. Diskussion, Limitationen und Ausblick

8.1. Einordnung der Ergebnisse im Lichte der Zielsetzung

Die vorliegende Arbeit verfolgte das Ziel, ein datenschutzkonformes Retrieval-Augmented Generation System für den Hochschulkontext zu entwickeln und systematisch zu evaluieren. Im Mittelpunkt stand unteranderem die Frage, welche Komponentenkombinationen aus Parser, Chunking-Strategie, Embedding-Modell und Retrieval-Verfahren die höchste Antwortqualität erzielen und inwiefern lokale Open-Source-Ansätze eine praktikable Alternative zu proprietären Cloud-Lösungen darstellen.

Entwicklung eines funktionsfähigen Prototyps. Die Implementierung einer modularen RAG-Pipeline wurde erfolgreich realisiert. Der Prototyp verarbeitet sämtliche zuvor definierten Dokumenttypen und generiert darauf aufbauend kontextbezogene Antworten auf natürlichsprachliche Anfragen. Die technische Umsetzung umfasst sowohl cloudbasierte Komponenten (LlamaParse, OpenAI-Embeddings) als auch vollständig lokale Alternativen (Docling, Nomic-Embeddings), wodurch unterschiedliche Anforderungsprofile adressiert werden.

Systematischer Vergleich von Pipeline-Komponenten. Die Evaluation von 56 Konfigurationen ermöglichte eine klare Differenzierung der untersuchten Pipeline-Komponenten. Den größten Einfluss zeigte dabei die Wahl der Chunking-Strategie, strukturorientierte Ansätze erreichen durchschnittlich bessere Ergebnisse als semantisches Chunking. Zusätzlich erzielt der Einsatz eines Cross-Encoder-Rerankers einen messbaren Qualitätsgewinn, indem er die beste Docling/LangChain/Nomic-Pipeline von 0.6168 auf 0.6795 (+10,2 %) verbessert.

Evaluation lokaler versus proprietärer Ansätze. Die beste lokale Konfiguration erreicht einen Gesamtscore von 0.6795, die beste cloudbasierte 0.6603. Dieser Unterschied zeigt, dass Open-Source-Anwendungen eine praktische Alternative zu den Cloud-Lösungen darstellen, die ähnlich gut oder abhängig von der Domäne, sogar besser performen.

Beitrag zur Hochschulpraxis. Die systematische Evaluation liefert evidenzbasierte Designempfehlungen für die Implementierung von RAG-Systemen im Bildungskontext. Die Identifikation robuster Konfigurationen sowie die Analyse von Trade-offs zwischen Qualität, Datenschutz und Ressourcenverfügbarkeit ermöglichen fundierte Entscheidungen bei der Systementwicklung.

8.2. Reflexion der Methodik und Limitationen

Die in dieser Arbeit eingesetzte Methodik ermöglicht eine systematische und reproduzierbare Untersuchung verschiedener RAG-Pipeline-Komponenten. Dennoch ist die Evaluation durch mehrere Faktoren begrenzt, die die Aussagekraft der Ergebnisse beeinflussen und daher transparent eingeordnet werden sollten.

Die Datengrundlage bildet ein einziges Kerndokument aus dem Fachbereich „Algorithmen und Programmierung“, ergänzt durch zusätzliche, thematisch irrelevante Dokumente. Letztere wurden bewusst aufgenommen, um realistische Retrieval-Szenarien mit potenziell störenden Chunks zu simulieren. Insgesamt umfasst der Datensatz 46 Fragen mit zunehmender inhaltlicher Komplexität. Trotz dieses kontrollierten Aufbaus bleibt die Dokumentbasis begrenzt und fachlich eng fokussiert. Die Ergebnisse bieten daher belastbare Erkenntnisse für den untersuchten Kontext, lassen sich jedoch nicht ohne Weiteres auf andere Fachrichtungen oder stärker heterogene Dokumentlandschaften übertragen.

Auch die Erstellung des Golden-Standard-Datensatzes bringt methodische Einschränkungen mit sich. Die Fragen wurden halbautomatisch generiert und anschließend manuell validiert. Die Referenzantworten entstanden vollständig manuell aus den Originalquellen. Dieses Vorgehen gewährleistet eine hohe inhaltliche Genauigkeit, schließt jedoch subjektive Einflüsse nicht aus, etwa bei der Auswahl relevanter Details oder bei der gewünschten Antworttiefe. Ein mehrstufiges Annotation-Verfahren hätte diese Form der Subjektivität weiter reduzieren können, lag jedoch außerhalb des Projektumfangs.

Zudem sind die eingesetzten Evaluationsmetriken nur bedingt frei von Verzerrungen. RAGAS ermöglicht zwar eine automatisierte Bewertung entlang zentraler Qualitätsdimensionen, nutzt jedoch selbst ein LLM für die Bewertung. Dadurch können modellabhängige Fehlinterpretationen oder Bias in die Scores einfließen. Darüber hinaus erfassen die Metriken primär inhaltliche und strukturelle Kriterien, jedoch keine didaktischen Aspekte wie Verständlichkeit oder pädagogischen Mehrwert. Die Ergebnisse spiegeln somit nur einen Teil der tatsächlichen Antwortqualität wider.

Das Systemdesign stellt weitere Begrenzungen dar. Die Generierung erfolgte ausschließlich mit einem lokalen Modell (llama3.1:8b). Folglich sind die erzielten Ergebnisse eng an die Leistungsfähigkeit dieses Modells gebunden und nicht direkt auf andere LLMs übertragbar. Zusätzlich konnten bestimmte Parser-Konfigurationen, insbesondere solche, die Unstructured mit semantischem Chunking kombinieren, aufgrund technischer Inkompatibilitäten nicht getestet werden. Dies reduziert die Breite der Evaluation und beschränkt den Vergleich auf stabil ausführbare Pipeline-Kombinationen.

Zusammenfassend liefert die Untersuchung fundierte Einblicke in den untersuchten Hochschulkontext, weist jedoch zugleich klare Grenzen auf. Die Ergebnisse bieten eine solide Grundlage für die betrachteten Materialien, sollten aber mit Blick auf andere Disziplinen, zusätzliche Modelle oder komplexere Dokumentstrukturen vorsichtig generalisiert werden. Künftige Arbeiten können diese Limitationen aufgreifen, um die Übertragbarkeit und Robustheit von RAG-Systemen weiter zu untersuchen.

8.3. Praktische Implikationen

Die Ergebnisse dieser Arbeit haben direkte Implikationen für Hochschulen, IT-Abteilungen und Studierende, die den Einsatz von RAG-Systemen im Lehrkontext prüfen. Sie zeigen, welche technischen Entscheidungen die Antwortqualität besonders stark beeinflussen und unter welchen Bedingungen ein dokumentenbasierter Assistent zuverlässig arbeiten kann.

Die Konfiguration aus Docling-Extraktion, LangChain-Chunking, Nomic-Embeddings und Dense Retrieval mit Cross-Encoder-Reranking erreicht mit einem kombinierten Score von 0.6795 die beste Gesamtperformance. Besonders hervorzuheben ist der Einfluss des Rerankings. Der eingesetzte Cross-Encoder (ms-marco-MiniLM-L-6-v2) erhöht den Score der zugrunde liegenden Pipeline von 0.6168 auf 0.6795, was einer relativen Verbesserung von 10,2 % entspricht. Dies verdeutlicht, dass nicht allein der Retriever, sondern vor allem die Qualität der final ausgewählten Kontexte entscheidend für die Antwortgüte ist und dass Reranking im jeweiligen Daten- und Anwendungskontext sorgfältig kalibriert werden muss.

Der Einsatz eines lokalen, Open-Source-basierten RAG-Systems bietet dabei mehrere praktische Vorteile. Sensible Lernmaterialien können innerhalb der hochschulinternen Infrastruktur verarbeitet werden, was Datenschutz, Datenhoheit und Compliance erleichtert. Gleichzeitig bleibt die Pipeline transparent und auditierbar. Es ist nachvollziehbar, welche Quellen und Kontexte in die Antwortgenerierung einfließen. Darüber hinaus verlagern sich Kosten von nutzungsbasierten API-Gebühren hin zu planbaren Infrastruktur- und Betriebskosten. Ein lokaler Betrieb reduziert die Abhängigkeit

von externen Cloud-Anbietern und ermöglicht Szenarien mit eingeschränkter oder fehlender Internetanbindung.

Dennoch ist die Eintrittsbarriere für den produktiven Betrieb eines solchen Systems nicht zu unterschätzen. Die in dieser Arbeit eingesetzte Hardware (RTX 3090) zeigt, dass für performante Generierung und Reranking eine leistungsfähige GPU erforderlich ist, die in der Anschaffung und im Betrieb kostenintensiv ist. Reranking steigert zwar die Qualität, erhöht jedoch zugleich Rechenaufwand und Latenz pro Anfrage. Hinzu kommt der Bedarf an technischem Spezialwissen für Integration, Wartung und Monitoring der verschiedenen Komponenten. Damit wird deutlich, dass der erfolgreiche Einsatz von RAG-Systemen im Hochschulkontext weniger an einzelnen Modellen als an der Fähigkeit hängt, eine robuste, gut abgestimmte Infrastruktur aufzubauen und zu betreiben.

8.4. Ausblick auf zukünftige Arbeiten

Die vorliegende Untersuchung bildet eine fundierte Basis für die Analyse von RAG-Systemen im Hochschulkontext, eröffnet zugleich jedoch mehrere Perspektiven für weiterführende Arbeiten. Methodisch bietet sich vor allem eine Erweiterung der Modell- und Komponentenvielfalt an. Größere Sprachmodelle wie Llama 3.1 70B oder Modelle der Qwen-Reihe könnten untersucht werden, um zu prüfen, in welchem Ausmaß Modellkapazität und Architektur die Antwortqualität beeinflussen. Ebenso ließe sich das Spektrum der Embedding-Modelle und Reranker erweitern, etwa durch spezialisierte Cross-Encoder oder domänenspezifische Varianten.

Ein bislang nicht ausgeschöpfter Bereich betrifft die systematische Untersuchung von Chunk-Größen und Overlap-Strategien. Während diese Arbeit mit festen oder adaptiven Einstellungen arbeitete, könnten zukünftige Studien Chunk-Sizes entlang breiter Parameterbereiche (z. B. 250–1500 Token) sowie Overlap-Ratios zwischen 10% und 30% variieren, um deren Einfluss präziser zu quantifizieren. Ergänzend wäre zu analysieren, wie effektiv große Kontextfenster moderner LLMs genutzt werden und welche Konfigurationsstrategien die Kontextkapazität optimal ausschöpfen.

Auch die Evaluation lässt sich sinnvoll erweitern. Neben RAGAS-basierten Metriken wäre eine Kombination aus automatisierter Bewertung und Human-Evaluation durch Studierende oder Lehrende vielversprechend, insbesondere um Dimensionen wie Verständlichkeit, didaktische Qualität oder die Wahrnehmung des Nutzens abzudecken. Solche qualitativen Bewertungen könnten Einblicke liefern, die rein metrisch basierte Verfahren nicht erfassen.

Anwendungsbezogen bieten sich Studien mit realen Nutzerinteraktionen an, etwa im Rahmen von Tutorien, Übungen oder integrierten Assistenzsystemen in Lernplattformen. Dadurch ließe sich untersuchen, wie RAG-Systeme im authentischen Gebrauch wahrgenommen werden und welchen Beitrag sie zur Lernunterstützung leisten. Darüber hinaus wäre eine Übertragung der Methodik auf weitere Fachdomänen wie Medizin, Recht oder Ingenieurwissenschaften sinnvoll, um die Generalisierbarkeit der Pipeline-Konfigurationen zu prüfen und domänenspezifische Anpassungen zu identifizieren.

Insgesamt zeigen diese Perspektiven, dass RAG-Systeme ein breites Feld für technologische und anwendungsorientierte Weiterentwicklungen bieten. Künftige Arbeiten können die hier entwickelten Ansätze vertiefen, erweitern und in vielfältigere Nutzungskontexte übertragen.

8.5. Abschließende Zusammenfassung

Die vorliegende Arbeit untersucht, wie sich Retrieval-Augmented-Generation-Systeme im Hochschulkontext zuverlässig für die Verarbeitung und Bereitstellung technischer Lernmaterialien einsetzen lassen. Ausgangspunkt war die Herausforderung, eine transparente, datenschutzkonforme und zugleich leistungsfähige Pipeline zu entwickeln, die Studierende und Lehrende beim Zugang zu komplexen Inhalten unterstützt.

Im Rahmen der Arbeit wurde ein vollständig lokaler, Open-Source-basierter RAG-Prototyp entwickelt, der PDF- und weitere Dokumenttypen verarbeitet und kontextbezogene Antworten generiert. Auf dieser Grundlage wurden 56 unterschiedliche Pipeline-Konfigurationen systematisch evaluiert, um den Einfluss zentraler Komponenten wie Parser, Chunking, Embeddings, Retrieval und Reranking quantifizierbar zu machen. Die Ergebnisse zeigen, dass insbesondere die Kombination aus Docling-Extraktion, LangChain-Chunking, Nomic-Embeddings und Dense Retrieval mit Cross-Encoder-Reranking die höchste Gesamtperformance erzielt. Zudem wurde deutlich, dass Reranking einen erheblichen Einfluss auf die Antwortqualität hat und dass die Abstimmung einzelner Komponenten für robuste Ergebnisse entscheidender ist als die Wahl einer einzelnen Technologie.

Die gewonnenen Erkenntnisse leisten sowohl für Forschung als auch für die praktische Systementwicklung einen Beitrag. Sie zeigen, wie RAG-Pipelines für den Einsatz an Hochschulen ausgestaltet werden können und welche Stellen im Systemdesign besondere Aufmerksamkeit erfordern. Gleichzeitig schaffen sie eine Grundlage für weiterführende Arbeiten, die das Potenzial solcher Systeme für Lehre und Lernplattformen künftig noch breiter erschließen können.

Literatur

Bast, Hannah, Björn Buchhold und Elmar Haussmann (2016). *Semantic Search on Text and Knowledge Bases.*

Bergman, Ofer u. a. (2010). *The Effect of Folder Structure on Personal File Navigation.*

Bond, Melissa u. a. (2021). *Digital transformation during the COVID-19 pandemic: A systematic review of education.*

Britt, M. Anne und Jean-François Rouet (2012). *Learning with Multiple Documents: Component Skills and Their Acquisition.*

Deerwester, Scott u. a. (1990). *Indexing by latent semantic analysis.*

Dettmers, Tim u. a. (2023). *QLoRA: Efficient Finetuning of Quantized LLMs.*

Eppler, Martin und Jeanne Mengis (2004). *The Concept of Information Overload: A Review of Literature From Organization Science, Accounting, Marketing, MIS, and Related Disciplines.*

Frantar, Elias u. a. (2023). *GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers.*

Furnas, G. W. u. a. (1987). *The vocabulary problem in human-system communication.*

Ganesh, Prakhar u. a. (2021). *Compressing Large-Scale Transformer-Based Models: A Case Study on BERT.*

Gao, Yunfan u. a. (2024). *Retrieval-Augmented Generation for Large Language Models: A Survey.*

Head, Alison und Michael Eisenberg (2010). *Truth Be Told: How College Students Evaluate and Use Information in the Digital Age.*

Hu, Edward J u. a. (2021). *LoRA: Low-Rank Adaptation of Large Language Models.*

Jenert, Tobias und Taiga Brahm (2021). *The interplay of personal and contextual diversity during the first year at Higher Education: Combining a quantitative and a qualitative approach.*

Jurafsky, Daniel und James Martin (2013). *Speech and Language Processing Pearson New International Edition.*

Kerres, Michael (2018). *KerBildung in der digitalen Welt - Wir haben die Wahl.*

Kirkpatrick, James u. a. (2017). *Overcoming catastrophic forgetting in neural networks.*

Kitsantas, Anastasia, Adam Winsler und Faye Huie (2008). *Self-regulation and ability predictors of academic success during college: A predictive validity study.*

Kraus, Felix u. a. (2025). *A Gold Standard Benchmark Dataset for Digital Humanities.*

Kudo, Taku und John Richardson (2018). *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing.*

Lewis, Patrick u. a. (2021). *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*.

Manning, Christopher D. (2009). *Introduction to Information Retrieval*.

Manning, Christopher D., Prabhakar Raghavan und Hinrich Schütze (2008). *Introduction to Information Retrieval*.

Manning, Christopher D. und Hinrich Schütze (1999). *Foundations of Statistical Natural Language Processing*.

Mikolov, Tomas u. a. (2013). *Efficient Estimation of Word Representations in Vector Space*.

Nadeau, David und Satoshi Sekine (2007). *A Survey of Named Entity Recognition and Classification*.

Ovadia, Oded u. a. (2024). *Fine-Tuning or Retrieval? Comparing Knowledge Injection in LLMs*.

Radford, Alec u. a. (2019). *Language Models are Unsupervised Multitask Learners*.

Reimers, Nils und Iryna Gurevych (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*.

Rouet, Jean-François und M. Anne Britt (2011). *Relevance processes in multiple document comprehension*.

Säljö, R. (1984). *On qualitative differences in learning*.

Salton, G., A. Wong und C. S. Yang (1975). *A vector space model for automatic indexing*.

Schmitz, Bernhard und Bettina S. Wiese (2006). *New perspectives for the evaluation of training sessions in self-regulated learning: Time-series analyses of diary data*.

Sebastiani, Fabrizio (2002). *Machine learning in automated text categorization*.

Sennrich, Rico, Barry Haddow und Alexandra Birch (2016). *Neural Machine Translation of Rare Words with Subword Units*.

Shuster, Kurt u. a. (2021). *Retrieval Augmentation Reduces Hallucination in Conversation*.

Thakur, Nandan u. a. (2021). *BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models*.

Vaswani, Ashish u. a. (2017). *Attention Is All You Need*.

Webster, Jonathan J. und Chunyu Kit (1992). *Tokenization as the Initial Phase in NLP*.

Zawacki-Richter, Olaf u. a. (2020). *Systematic Reviews in Educational Research: Methodology, Perspectives and Application*.

Zimmerman, Barry (2002). *Becoming a Self-Regulated Learner: An Overview*.

Anhang

Github Link zum Code: https://github.com/0xCmdlet/bachelor-arbeit

A. Fragenkatalog zur Evaluation

Im Folgenden ist der zur Evaluation der RAG-Applikation verwendete Fragenkatalog in JSON-Form dargestellt. Die Fragen beziehen sich auf das Dokument `c-programming-material.pdf`.

Listing A.1: Fragenkatalog für die Evaluation

```
{
  "description": "Test queries for C programming evaluation",
  "document": "c-programming-material.pdf",
  "queries": [
    {
      "id": "q1",
      "query": "Wann wurde die Sprache C entwickelt und von wem?",
      "type": "factual"
    },
    {
      "id": "q2",
      "query": "Welche Normierung wurde 1988 eingefhrt?",
      "type": "factual"
    },
    {
      "id": "q3",
      "query": "Welche drei Charakteristika besitzt C?",
      "type": "factual"
    },
    {
      "id": "q4",
      "query": "Was bewirkt #include <stdio.h>?",
      "type": "conceptual"
    },
    {
      "id": "q5",
      "query": "Aus welchen Teilen besteht eine Funktionsdefinition?",
      "type": "factual"
    },
    {
      "id": "q6",
      "query": "Welche Funktion gibt Text auf dem Bildschirm aus?",
      "type": "factual"
    },
```

```
  {
    "id": "q7",
    "query": "Was bedeutet \\n?",
    "type": "factual"
  },
  {
    "id": "q8",
    "query": "Warum steht in scanf(\"%f\", &betrag); ein &?",
    "type": "conceptual"
  },
  {
    "id": "q9",
    "query": "Was ist ein Statementblock?",
    "type": "factual"
  },
  {
    "id": "q10",
    "query": "Wie werden Kommentare in C geschrieben?",
    "type": "procedural"
  },
  {
    "id": "q11",
    "query": "Was unterscheidet Variablen von Konstanten?",
    "type": "conceptual"
  },
  {
    "id": "q12",
    "query": "Welche Arten von Konstanten gibt es?",
    "type": "factual"
  },
  {
    "id": "q13",
    "query": "Wozu dient sizeof?",
    "type": "conceptual"
  },
  {
    "id": "q14",
    "query": "Was ist automatische Typkonvertierung?",
    "type": "conceptual"
  },
  {
    "id": "q15",
    "query": "Was passiert bei int / int?",
    "type": "conceptual"
  },
  {
```

```
    "id": "q16",
    "query": "Welchen Wertebereich hat der Typ char?",
    "type": "factual"
  },
  {
    "id": "q17",
    "query": "Welchen Wertebereich hat unsigned int?",
    "type": "factual"
  },
  {
    "id": "q18",
    "query": "Wie definiert man mehrere Variablen desselben Typs?",
    "type": "procedural"
  },
  {
    "id": "q19",
    "query": "Wie initialisiert man eine Variable?",
    "type": "procedural"
  },
  {
    "id": "q20",
    "query": "Wie ist eine Stringkonstante im Speicher aufgebaut?",
    "type": "conceptual"
  },
  {
    "id": "q21",
    "query": "Was macht der Operator . ?",
    "type": "conceptual"
  },
  {
    "id": "q22",
    "query": "Unterschied zwischen ++x und x++?",
    "type": "conceptual"
  },
  {
    "id": "q23",
    "query": "Was macht a += 2?",
    "type": "conceptual"
  },
  {
    "id": "q24",
    "query": "Warum fhrt C keine Array-Grenzprfung durch?",
    "type": "conceptual"
  },
  {
    "id": "q25",
```

```
    "query": "Warum geht b = a; bei Arrays nicht?",
    "type": "conceptual"
  },
  {
    "id": "q26",
    "query": "Wie definiert man eine symbolische Konstante?",
    "type": "procedural"
  },
  {
    "id": "q27",
    "query": "Wie werden enum-Werte nummeriert?",
    "type": "factual"
  },
  {
    "id": "q28",
    "query": "Wie funktioniert eine if-Anweisung?",
    "type": "conceptual"
  },
  {
    "id": "q29",
    "query": "Was ist das dangling-else Problem?",
    "type": "conceptual"
  },
  {
    "id": "q30",
    "query": "Was ist der Unterschied zwischen if und switch?",
    "type": "conceptual"
  },
  {
    "id": "q31",
    "query": "Was ist ein fall-through?",
    "type": "conceptual"
  },
  {
    "id": "q32",
    "query": "Wann benutzt man do-while?",
    "type": "conceptual"
  },
  {
    "id": "q33",
    "query": "Wozu dient continue?",
    "type": "conceptual"
  },
  {
    "id": "q34",
    "query": "Welche Teile besitzt eine for-Schleife?",
```

```
    "type": "factual"
  },
  {
    "id": "q35",
    "query": "Wie sieht eine Endlosschleife aus?",
    "type": "procedural"
  },
  {
    "id": "q36",
    "query": "Warum sollte goto vermieden werden?",
    "type": "conceptual"
  },
  {
    "id": "q37",
    "query": "Wie definiert man ein Array und mit welchem Index beginnt es?
       ",
    "type": "procedural"
  },
  {
    "id": "q38",
    "query": "Was passiert bei eingabe[3] wenn int eingabe[3]; definiert
       wurde?",
    "type": "conceptual"
  },
  {
    "id": "q39",
    "query": "Wie kopiert man Arrays?",
    "type": "procedural"
  },
  {
    "id": "q40",
    "query": "Wie definiert man ein zweidimensionales Array?",
    "type": "procedural"
  },
  {
    "id": "q41",
    "query": "Wie initialisiert man Arrays?",
    "type": "procedural"
  },
  {
    "id": "q42",
    "query": "Unterschied zwischen {'C'} und \"C\"?",
    "type": "conceptual"
  },
  {
    "id": "q43",
```

```
      "query": "Was ist der Vorteil von Strukturen?",
      "type": "conceptual"
    },
    {
      "id": "q44",
      "query": "Wie definiert man einen eigenen Strukturtyp mit typedef?",
      "type": "procedural"
    },
    {
      "id": "q45",
      "query": "Wie greift man auf verschachtelte Strukturen zu?",
      "type": "procedural"
    },
    {
      "id": "q46",
      "query": "Wie funktioniert Call-by-Value in C?",
      "type": "conceptual"
    }
  ]
}
```

Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin/des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Anmerkung: In einigen Studiengängen findet sich die Erklärung unmittelbar hinter dem Deckblatt der Arbeit.

Ort, Datum

Unterschrift