Hochschule für Angewandte Wissenschaften Ansbach

Bachelorthesis

Entwicklung eines Retrieval-Augmented Generation (RAG)-Systems für den mehrsprachigen technischen Informationsabruf aus Produktdatenblättern

Niclas Hart

Matrikelnummer: 00163453

Studiengang: Künstliche Intelligenz und Kognitive Systeme

Abgabedatum: 15.02.2026

Betreuer: Prof. Dr. Sigurd Schacht

Zweitbetreuer: Prof. Dr. Stefan Geißelsöder

Eidesstattliche Erklärung

„Ich versichere, dass ich die Arbeit selbständig angefertigt, nicht anderweitig für Prüfungszwecke vorgelegt, alle benützten Quellen und Hilfsmittel angegeben sowie wörtliche und sinngemäße Zitate gekennzeichnet habe."

Ansbach, 12.02.2026
Ort, Datum


Unterschrift

Abstract

Die Bearbeitung von Ausschreibungen im Technical Consulting Center (TCC) von Bechtle erfordert die manuelle Extraktion technischer Informationen aus einer Vielzahl von Produktdatenblättern unterschiedlicher Hersteller und Formate. Dieser Prozess ist zeitintensiv und fehleranfällig, insbesondere bei engen Ausschreibungsfristen. Diese Bachelorarbeit entwickelt und evaluiert ein Retrieval-Augmented Generation (RAG)-System zur automatisierten Beantwortung technischer Fragen aus Produktdatenblättern, um die Informationsbeschaffung zu beschleunigen und die Konsistenz zu verbessern.

Das entwickelte System kombiniert semantisches Retrieval mit generativen Sprachmodellen, um natürlichsprachliche Fragen zu technischen Spezifikationen von Laptops zu beantworten. Die Pipeline umfasst die Extraktion von Inhalten, die Segmentierung in semantisch kohärente Chunks, die Generierung von Vektor-Repräsentationen mittels Sentence-Transformer-Embeddings sowie deren Speicherung in einer Vektordatenbank. Die Antwortgenerierung erfolgt mit GPT-4o-mini, das durch ein zweistufiges Retrieval-Verfahren mit optionalem Reranking unterstützt wird.

Die systematische Evaluation erfolgte mit einem Gold Standard Datensatz von Fragen zu Laptop-Spezifikationen unter Verwendung des RAGAS-Frameworks (Retrieval-Augmented Generation Assessment). Vier verschiedene Embedding-Modelle wurden evaluiert, wobei das Modell `paraphrase-multilingual-MiniLM-L12-v2` die beste Leistung erzielte. Die Evaluationsergebnisse zeigen, dass das System in der Lage ist, einen Großteil der technischen Fragen korrekt zu beantworten. Das Retrieval-System identifiziert effektiv relevante Informationen, während die generierten Antworten größtenteils durch die abgerufenen Dokumente gestützt werden.

Die Evaluation lieferte wichtige Erkenntnisse für die praktische Anwendung von RAG-Systemen: Größere Modelle führen nicht automatisch zu besseren Ergebnissen und multilinguales Training erwies sich als entscheidend für den vorliegenden Anwendungsfall. Die Fehleranalyse identifizierte drei charakteristische Probleme: fehlende Informationen trotz vorhandener Dokumente, unvollständige Antworten aufgrund von Chunk-Grenzen und Modellverwechslungen bei ähnlich benannten Produkten.

Die Arbeit demonstriert erfolgreich, dass RAG-Architekturen bei entsprechender Modellierung ein vielversprechendes Fundament für den Aufbau zuverlässiger und kontextsensitiver NLP-Systeme bilden können. Das entwickelte System adressiert ein konkretes Problem im Arbeitsalltag des TCC und zeigt, wie moderne KI-Technologien praktische Probleme im Unternehmenskontext lösen können. Durch die Automatisierung der Informationsbeschaffung aus Produktdatenblättern kann das System die Bearbeitungszeit von Ausschreibungen reduzieren, die Konsistenz der Informationsbeschaffung verbessern und Fehler durch manuelle Extraktion minimieren.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

In der IT-Systemlandschaft großer Unternehmen spielt die schnelle und präzise Bereitstellung technischer Informationen eine entscheidende Rolle - insbesondere bei der Bearbeitung von Ausschreibungen. Im Rahmen solcher Ausschreibungen müssen technische Anforderungen unterschiedlicher Kunden exakt mit passenden Produktmerkmalen abgeglichen werden. Dieser Prozess erfordert ein tiefes Verständnis von Produktdatenblättern, die detaillierte Spezifikationen, Kompatibilitäten und Zertifizierungen enthalten.

Beim IT-Dienstleister Bechtle wird diese Aufgabe im Technical Consulting Center (TCC) von Spezialisten übernommen, die regelmäßig große Mengen an Produktdaten - beispielsweise von Laptops, Thin-Clients und Monitoren - prüfen und bewerten. Dabei stehen sie vor der Herausforderung, relevante technische Informationen aus einer Vielzahl an Datenblättern unterschiedlicher Hersteller und Formate manuell zu extrahieren. Dieser Prozess ist nicht nur zeitintensiv, sondern birgt auch ein hohes Risiko für Fehler oder Inkonsistenz, insbesondere bei engen Ausschreibungsfristen.

Mit den Fortschritten im Bereich der Künstlichen Intelligenz und insbesondere großer Sprachmodelle eröffnen sich neue Möglichkeiten, diese Arbeitsschritte zu automatisieren und zu optimieren. Allerdings sind generative Sprachmodelle allein nicht in der Lage auf unternehmensspezifische oder aktuelle Produktinformationen zuzugreifen und neigen zu Halluzinationen. Hier setzt der Ansatz der **Retrieval-Augmented Generation (RAG)** an: Durch die Kombination eines semantischen Informationsabrufs mit einem Sprachmodell können Antworten direkt auf Grundlage der tatsächlichen Produktdatenblätter erzeugt werden.

Ein solches System ermöglicht es, technische Fragen in natürlicher Sprache zu stellen und gleichzeitig präzise, dokumentengestützte Antworten zu erhalten - beispielsweise:

> *"Welche Prozessoren unterstützt das Laptop XY?"*
> *"Welche maximale Bildschirmhelligkeit besitzt das Layptop XY?"*

Das Ziel ist es, die Informationsbeschaffung für Ausschreibungen deutlich zu beschleunigen, die Fehlerquote zu reduzieren und den Experten im TCC ein intelligentes Tool zur Verfügung zu stellen, das Wissen aus verschiedenen Produktdokumenten konsolidiert und verständlich aufbereitet. Die Entwicklung eines solchen Systems verspricht nicht nur eine Effizienzsteigerung im Arbeitsalltag, sondern leistet auch einen Beitrag zur Digitalisierung und Wissensautomatisierung im technischen Beratungsfeld von Bechtle.

1.2 Zielsetzung

Ziel dieser Arbeit ist die Konzeption und prototypische Implementierung eines RAG-Systems, das technische Produktdatenblätter auswertet und nutzerfreundlich zugänglich macht. Das System soll insbesondere im TCC von Bechtle eingesetzt werden, um Ausschreibungsprozesse oder das Pre-Sales effizienter zu gestalten. Im Fokus steht dabei die automatisierte Verarbeitung und Bereitstellung technischer Informationen zu Laptops, die häufig in heterogenen und unstrukturierten Formaten - wie PDF-Dokumenten - vorliegen.
Die Arbeit verfolgt somit zwei übergeordnete Zielrichtungen:

1. **Wissenschaftlich-technisch:** Untersuchung der technischen Machbarkeit und Evaluierung geeigneter Komponeneten für ein RAG-System im Kontext technischer Produktinformationen.
2. **Praktisch-anwendungsorientiert:** Entwicklung eines funktionalen Prototyps, der im Rahmen des TCC als Tool zur Unterstützung von Ausschreibungen eingesetzt werden kann.

Auf dieser Grundlage ergibt sich die zentrale Forschungsfrage:

> **Wie kann ein technisches RAG-System aufgebaut werden, um strukturierte und unstrukturierte technische Produktdaten mutlilingual zugänglich zu machen?**

Diese Frage wird durch mehrere technische Teilfragen konkretisiert:

- Welche Komponenten (z.B. OCR, Chunking, Vektordatenbanken, LLMs) sind für die Pipeline notwendig?
- Wie kann semantisches Retrieval technisch robust umgesetzt werden?
- Wie kann Mehrsprachigkeit technisch integriert werden?
- Welche Herausforderungen ergeben sich bei der strukturellen Aufbereitung technischer PDFs?

Ziel der Arbeit ist es, diese Fragen systematisch zu untersuchen, eine geeignete technische Architektur zu entwerfen und die Funktionalität anhand eines protoypischen Systems zu evaluieren. Dabei liegt der Fokus nicht nur auf der Leistungsfähigkeit des Ansatzes, sondern auch auf der praktischen Anwendbarkeit im realen Arbeitskontext bei Bechtle.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit gliedert sich in sechs Kapitel:

In **Kapitel 1** werden die Motivation, die Zielsetzung und die Forschungsfragen vorgestellt. Dabei wird der praktische Kontext im Technical Consulting Center (TCC) der Bechtle AG erläutert und die Relevanz eines RAG-basierten Systems zur Unterstützung von Ausschreibungen begründet.

Kapitel 2 vermittelt die theoretischen Grundlagen der Arbeit. Es werden zentrale Konzepte aus den Bereichen *Information Retrieval, Large Language Models und Retrieval-Augmented Generation* erläutert. Darüber hinaus werden Evaluierungsmethoden für RAG-Systeme sowie vergleichbare Arbeiten und der State of the Art analysiert.

In **Kapitel 3** erfolgt die Anforderungsanalyse und Konzeption des Systems. Hier werden die funktionalen und nicht-funktionalen Anforderungen definiert, die Datenbasis beschrieben und die Systemarchitektur mit ihren Komponenten konzipiert.

Kapitel 4 beschreibt die Implementierung des entwickelten RAG-Prototyps. Detailliert werden die Schritte zur Datenextraktion, Aufbereitung und Indexierung dargestellt sowie die technische Integration der Retrieval- und Generierungsprozesse erläutert.

In **Kapitel 5** werden die Evaluationsmethoden und die Ergebnisse vorgestellt. Anhand eines Gold Standard Datensatzes von 90 Fragen werden vier verschiedene Embedding-Modelle evaluiert und die Leistungsfähigkeit des Systems analysiert. Die Ergebnisse werden interpretiert und im Hinblick auf die Forschungsfragen diskutiert.

Abschließend fasst **Kapitel 6** die wesentlichen Erkenntnisse zusammen und gibt einen Ausblick auf mögliche Weiterentwicklungen. Dabei werden insbesondere Perspektiven für den produktiven Einsatz im Unternehmenskontext sowie für zukünftige Forschung im Bereich technischer RAG-Systeme aufgezeigt.

2 Theoretische Grundlagen

Produktdatenblätter enthalten technische Spezifikationen sowie Beschreibungen von Produkten und liegen häufig in Dokumentenform (z. B. PDF) vor. In diesem Szenario sollen Anfragen in beliebiger Sprache gestellt werden können, um automatisiert Informationen aus deutsch- und englischsprachigen Produktdatenblättern zu erhalten. Retrieval-Augmented Generation ist ein Ansatz, der große Sprachmodelle mit externem Wissen verknüpft, um fundierte und korrekte Antworten zu generieren. Ziel dieses Kapitels ist es, die theoretischen Grundlagen für die Entwicklung eines solchen RAG-Systems darzulegen.

2.1 Information Retrieval (IR)

Information Retrieval bezeichnet das Auffinden von relevanten Materialien (meist aus Dokumenten mit unstrukturiertem Text), das einem bestimmten Informationsbedürfnis entspricht, innerhalb einer großen Sammlung von Daten. [18] Klassische IR-Systeme - wie etwa Web-Suchmaschinen - durchsuchen Textmengen nach Dokumenten, die zu einer Benutzeranfrage passen. Dabei wurde traditionell auf term-basierte Methoden (beispielsweise Stichwortsuche mit TF-IDF oder BM25) gesetzt. Moderne Entwicklungen integrieren jedoch vermehrt neuronale Methoden, um semantische Zusammenhänge nutzen zu können. So hat sich das Feld von rein wortbasierten Ansätzen hin zu Dense Retrieval mit Vektorrepräsentation entwickelt. Letzteres verwendet Embeddingmodelle, um sowohl Dokumente als auch Anfragen in kontinuierliche Vektoren zu überführen, sodass semantisch ähnliche Inhalte durch Abstandsmaße identifiziert werden können. In vielen aktuellen Systemen werden hybride Suchverfahren eingesetzt, die die Vorteile beider Ansätze kombinieren. [35]

Für die technische Analyse von Produktdatenblättern ist IR ein zentraler Baustein. Die Produktdokumente müssen durchsuchbar gemacht werden, um relevante Textabschnitte zu einer Benutzeranfrage zu finden. Typischerweise wird dabei folgender Workflow eingesetzt:

1. **Dokumentenaufbereitung**: Lange Dokumente werden in kleinere Dokument-Chunks zerlegt (z. B. Absätze oder Sinnabschnitte), damit sie effizient verarbeitet und durchsucht werden können. Gegebenenfalls werden dabei Dokumentformate wie PDF zunächst in reinen Text umgewandelt, da PDF-Dateien oft mehrspaltiges Layout, Tabellen und Grafiken enthalten, was die direkte Verarbeitung erschwert. [21]

2. **Indexierung**: Für jeden Chunk wird eine Repräsentation erzeugt - entweder durch Extraktion von Schlüsselwörtern oder mittels semantischer Embeddings. Bei semantischer Indexierung wandelt ein Embedding-Modell jeden Text-Chunk in einen Vektor im semantischen Raum um. Alle Vektoren werden in einer spezialisierten Datenbank (Vektorspeicher) abgelegt, die Ähnlichkeitssuchen unterstützt.

3. **Abfrage-Verarbeitung und Retrieval**: Kommt eine Nutzeranfrage, erzeugt das System aus der Query ebenfalls eine Vektor-Repräsentation (sofern eine semantische Suche genutzt wird) und ruft die Top-K ähnlichsten Dokumenten-Chunk-Vektoren aus dem Index ab. [4] Alternativ oder ergänzend können klassische Stichwortsuchen durchgeführt werden. Das Resultat dieses Schritts ist eine kleine Menge potentiell relevanter Textausschnitte aus den Produktdatenblättern, welche die benötigten Informationen enthalten.

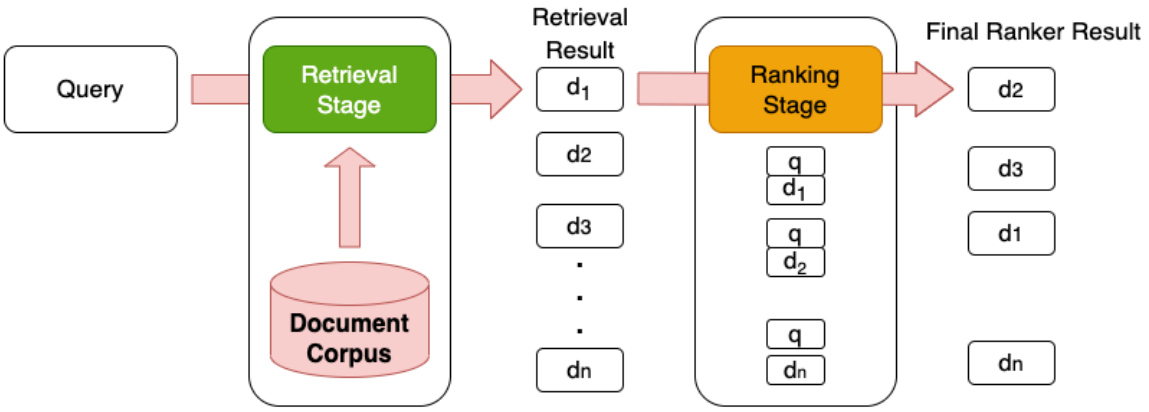

Abbildung 2.1: Überblick über ein zweistufiges Information-Retrieval-System [9]

2.1.1 PDF-Extraktion und Dokumentenanalyse

Die Extraktion von Text aus PDF-Dokumenten stellt eine wesentliche Vorverarbeitungsphase dar. Verschiedene Ansätze unterscheiden sich in ihrer Fähigkeit, strukturierte Elemente zu erhalten:

- **Layout-basierte Extraktion** (z. B. PyMuPDF): Extrahiert Text zeilenweise, kann jedoch bei mehrspaltigen Layouts oder Tabellen die semantische Struktur verlieren.
- **Markdown-Konvertierung** (z. B. pymupdf4llm): Konvertiert PDF-Inhalte in Markdown-Format, wobei Tabellen als Markdown-Tabellen erhalten bleiben.
- **Element-basierte Extraktion** (z. B. Unstructured): Klassifiziert Dokumentelemente (Text, Tabelle, Bild) und erhält deren Typ in den Metadaten.

Für technische Datenblätter mit tabellarischen Spezifikationen ist die Erhaltung der Tabellenstruktur essentiell, da die Zuordnung von Attributen zu Werten sonst verloren geht. Produktdatenblätter enthalten oft komplex formatierte Inhalte (mehrspaltiger Text, eingebettete Tabellen, verschiedene Einheiten und Messwerte), was die direkte Verarbeitung erschwert. [21]

2.1.2 Chunking-Strategien

Die Segmentierung von Dokumenten in Chunks beeinflusst maßgeblich die Retrieval-Qualität. Es existieren verschiedene Ansätze:

1. **Fixed-Size Chunking**: Teilt Dokumente nach fester Zeichenanzahl. Einfach zu implementieren, aber semantische Grenzen werden ignoriert.
2. **Recursive Character Splitting**: Verwendet eine Hierarchie von Trennzeichen (\n\n, \n, Satzzeichen), um Chunks an natürlichen Textgrenzen zu erstellen.
3. **Semantic Chunking**: Nutzt Embedding-Ähnlichkeit zwischen benachbarten Sätzen, um Chunks an semantischen Brüchen zu trennen. Benachbarte Sätze mit hoher Ähnlichkeit bleiben im selben Chunk.
4. **Document-Aware Chunking**: Berücksichtigt Dokumentstruktur wie Überschriften, Tabellen oder Abschnitte explizit bei der Segmentierung. [11]

Die Wahl der Chunk-Größe stellt einen Trade-off dar: Kleine Chunks ermöglichen präziseres Retrieval, verlieren aber Kontext; große Chunks erhalten Kontext, können aber irrelevante Informationen einschließen. [1]

Jeder Chunk kann zusätzlich mit Metadaten versehen werden, etwa dem Produktnamen, der Sprache oder der Abschnittsüberschrift, um bei der Antwortgenerierung Kontext für Quellenangaben zu haben. Diese Chunks bilden die Dokumentensammlung, die in den Vektorindex eingeht.

2.2 Large Language Models (LLMs)

Unter *Large Language Models* (LLMs) versteht man große vortrainierte Sprachmodelle mit oft Milliarden von Parametern, die auf umfangreichen Textkorpora trainiert wurden. Diese Modelle haben bemerkenswerte Fähigkeiten in Sprachverstehen und -generierung, können kontextuelle Bedeutungen erfassen und sogar einfache logische Schlüsse ziehen. [35] Die große Anzahl an Parametern speichert allgemeines Weltwissen und sprachliche Muster. Dadurch sind LLMs in der Lage, zu vielseitigen Eingaben kohärente Texte zu erzeugen und eine breite Palette von NLP-Aufgaben in verschiedensten Domänen zu bewältigen.

Eine Schlüsselinnovation moderner Large Language Models ist die **Transformer-Architektur**, die erstmals 2017 von *Vaswani et al.* vorgestellt wurde und markierte einen Wendepunkt in der sequentiellen Verarbeitung mittels neuronaler Netze. [27] Die Motivation hinter diesem Modell, war die Limitierung rekurrenter Architekturen (Recurrent Neural Networks) zu überwinden, insbesondere deren sequentielle Berechnungen, welche Parallelisierung erschwerte und das Lernen langfristiger Abhängigkeiten behinderte. Traditionelle Sequenztransduktions-Modelle wie Long-Short-Term-Memory (LSTM) verarbeiten Eingaben Schritt für Schritt, was bei langen Sequenzen

die Trainingszeit erhöhte und die Modellierung weit entfernter Beziehungen erschwerte. Der Transformer dagegen verzichtet vollständig auf Rekurrenz und Convolution und setzt stattdessen ausschließlich auf Attention-Mechanismen. [27] Diese vollständige Ablösung der Rekurrenz durch Self-Attention ermöglicht es, Abhängigkeiten zwischen beliebigen Positionen in einer Sequenz zu modellieren - unabhängig von deren Abstand - und dabei alle Token einer Sequenz gleichzeitig (parallel) zu verarbeiten.

2.2.1 Grundlegende Architektur

Das Transformer-Modell folgt dem Encoder-Decoder Prinzip, wie es aus sequenziellen Übersetzungsmodellen bekannt ist.

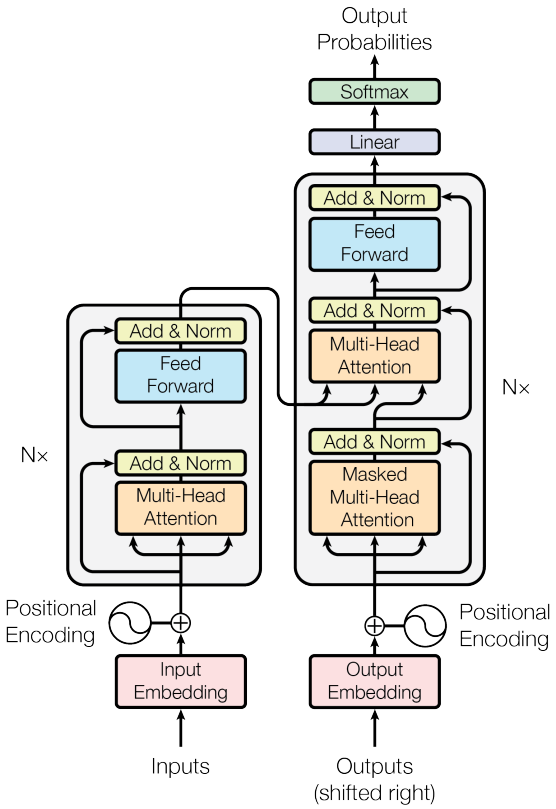

Abbildung 2.2: Schematische Darstellung der Transformer-Architektur. [27]

Ein **Encoder** erhält eine Eingabesequenz und wandelt sie in eine Sequenz kontinuierlicher Repräsentationen um, wohingegen ein **Decoder** autoregressiv Schritt für Schritt eine Ausgabe generiert und dabei jeweils die zuvor erzeugten Symbole als Kontext berücksichtigt. [27] Sowohl der Encoder als auch der Decoder bestehen aus mehreren übereinander gestapelten Schichten, in denen Self-Attention-Mechanismen und einfache Feedforward-Netze kombiniert werden. Jede Encoder-Schicht enthält zunächst eine Multi-Head-Self-Attention-Subschicht, gefolgt von einem positionsweisen fully-connected Feedforward-Netz; zur Stabilisierung des Trainings wird nach jeder Subschicht die Eingabe über eine Residualverbindung hinzuaddiert und anschließend eine Layer Normalization durchgeführt. Der Decoder besitzt eine analoge Schichtenstruktur, ergänzt

um eine zusätzliche **Encoder-Decoder-Attention** in jeder Schicht, welche es dem Decoder ermöglicht, die vom Encoder erzeugten Repräsentationen der Eingabesequenz zu berücksichtigen. Zusätzlich wird in der Self-Attention des Decoder eine Maskierung eingesetzt, die bewirkt, dass jedes Ausgabetoken nur auf frühere (bereits generierte) Tokens aufmerksam sein kann und nicht auf zukünftige. Diese sogenannte kausale Maske stellt sicher, dass der Decoder beim Generieren strikt autoregressiv bleibt. [27]

2.2.2 Limitationen von LLMs

Ihr in den Parametern gespeichertes Wissen ist statisch - es spiegelt den Stand der Trainingsdaten wider und lässt sich nachträglich nur durch erneutes Training oder Fine-Tuning ändern. Dies ist problematisch, wenn aktuelles oder domänenspezifisches Wissen benötigt wird, das nicht in den Trainingsdaten enthalten war. [14]

Zudem neigen LLMs dazu, mit hoher sprachlicher Sicherheit Aussagen zu generieren, die faktisch falsch sind, wenn ihnen das nötige Wissen fehlt. Solche *halluzinierten* Antworten wirken plausibel, sind aber inhaltlich unzuverlässig. [35] Dieses Phänomen - das Erfinden falscher Fakten - ist besonders kritisch bei der technischen Analyse von Dokumenten, wo Genauigkeit oberste Priorität hat.

Ein weiterer Aspekt ist die Mehrsprachigkeit. Viele State-of-the-Art LLMs (z. B. GPT-4) wurden hauptsächlich auf englischsprachigen Texten trainiert, was dazu führt, dass ihre Leistung in anderen Sprachen etwas abfallen kann. [4] Obwohl diese Modelle grundsätzlich über multilinguale Fähigkeiten verfügen, sind bestimmte Sprachen oder Fachjargons unter Umständen weniger gut abgedeckt. Für ein System, das deutsch- und englischsprachige Produktdatenblätter verarbeitet und Anfragen in beliebigen Sprachen versteht, muss daher sichergestellt werden, dass das Sprachmodell die benötigten Sprachen sicher beherrscht oder entsprechende Übersetzungsmechanismen eingebunden werden.

Insgesamt bieten LLMs eine leistungsfähige Grundlage für generative KI-Systeme, müssen aber für wissensintensive Aufgaben durch externe Informationen ergänzt werden, um verlässliche und aktuelle Ausgaben zu gewährleisten. Genau hier setzt das Retrieval-Augmented Generation Konzept an.

2.3 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation bezeichnet ein hybrides Verfahren, bei dem ein generatives Sprachmodell durch eine angebundene Wissensbasis ergänzt wird. Anstatt ausschließlich auf das in den Modellparametern gespeicherte Wissen zurückzugreifen, holt sich das Modell bei RAG aktiv Informationen aus externen Dokumenten, die zum Zeitpunkt der Anfrage relevant sind. In unserem Kontext bedeutet dies: Das System durchsucht die vorhandenen Produktdatenblätter

nach passenden Inhalten zur Nutzeranfrage (*Retrieval*) und verwendet diese Inhalte dann als Kontext, um eine fundierte Antwort zu formulieren (*Generation*).

Formal lässt sich RAG beschreiben als die Kombination eines parametrischen Speichers (dem vortrainierten LLM) mit einem nicht-parametrischen Speicher (der durchsuchbaren Dokumentensammlung). [22] *Lewis et al. (2020)*, die den Begriff RAG prägten, formulierten es als allgemeines Fine-Tuning Rezept: Ein vortrainiertes Sequenz-zu-Sequenz-Modell wird so erweitert, dass es vor der Antwortgenerierung einen differentiellen Zugriff auf eine externe Wissensdatenbank erhält. [14] Konkret werden dazu zu einer Frage zunächst mittels eines neuronalen Retrievers relevante Textpassagen aus der Wissensdatenbank geholt und dem LLM als zusätzlichem Input bereitgestellt. [22] Das Sprachmodell generiert dann konditioniert auf dieser angereicherten Eingabe seine Antwort.

Durch diese Kopplung verbessern sich mehrere Aspekte der generativen Modelle: Die Antworten werden typischerweise faktenbasierter und spezifischer, da sie direkt auf den bereitgestellten Dokumentinhalten beruhen. [14] Zudem lässt sich das Wissensspektrum des Systems dynamisch aktualisieren, indem man die zugrundeliegende Dokumentensammlung (den Index) erweitert oder austauscht, ohne das Modell selbst neu trainieren zu müssen. Für Anwendungsfälle, die kontinuierlich neue Daten produzieren, ist dies ein großer Vorteil gegenüber rein parametrischen Modellen.

Ein weiterer Gewinn von RAG ist die Erklärbarkeit bzw. Nachvollziehbarkeit der Antworten. Da die generierten Ausgaben auf konkreten Quellen basieren, können diese dem Nutzer angezeigt oder zitiert werden. Ähnlich wie Fußnoten in einem Artikel erhöht dies das Vertrauen in die Antworten, weil der Benutzer die Herkunft der Information prüfen kann. Gleichzeitig hilft die Bereitstellung von Quellen dem Modell dabei, Mehrdeutigkeiten in der Anfrage aufzuklären und verringert das Risiko, dass das Modell Halluzinationen produziert (also überzeugend klingende, aber falsche Antworten).

Ein RAG-System durchläuft typischerweise folgende Schritte, um eine Antwort zu erzeugen:

1. **Datensammlung**: Zusammenstellen der domänenspezifischen Wissensbasis. Für den Anwendungsfall werden alle relevante Produktdatenblätter als Textquellen gesammelt.

2. **Vorverarbeitung**: Aufbereitung der Dokumente in ein geeignetes Format. Dies umfasst Bereinigung (Entfernen von Rauschen wie Header, Footer, Formatierungsartefakte) und Normalisierung des Textes sowie das Aufteilen in sinnvolle handhabbare Abschnitte (Chunking). Insbesondere bei sehr langen Dokumenten verbessert das Segmentieren in inhaltlich kohärente Abschnitte die Effizienz und Genauigkeit des späteren Retrievals.

3. **Vektor-Embedding und Indexierung**: Um eine semantische Suche zu ermöglichen, werden alle Textabschnitte durch ein Embedding-Modell (z. B. Sentence-BERT) in Vektoren überführt. Die resultierenden Vektorrepräsentationen werden in einem dafür optimierten Vektordatenbank-Index abgelegt, der schnelle Nearest-Neighbor-Abfragen erlaubt.

4. **Retrieval (Abruf)**: Bei einer eingehenden Benutzeranfrage wandelt der Retriever die Anfrage zunächst ebenfalls in einen Vektor um. Dann werden im Vektorindex die Top-K semantisch ähnlichsten Dokumenten-Embeddings gesucht und die korrespondierenden Textpassagen abgerufen.

5. **Kontext-Augmentation**: Die gefundenen Passagen werden als zusätzlicher Kontext der ursprünglichen Anfrage hinzugefügt. Durch diese Anreicherung mit externem Wissen wird das LLM gezwungen, seine Antwort an die gelieferten Fakten zu binden, anstatt rein auf interne Parametermuster zurückzufallen.

6. **Generierung der Antwort**: Das so kontextualisierte Prompt (Anfrage + Top-K Dokumentinhalte) wird dem generativen Sprachmodell übergeben, das daraufhin eine natürliche Sprachantwort erzeugt. Dank der vorangestellten Referenzinformationen ist diese Antwort nicht nur fließend formuliert, sondern auch inhaltlich begründet und mit den externen Daten konsistent. [10]

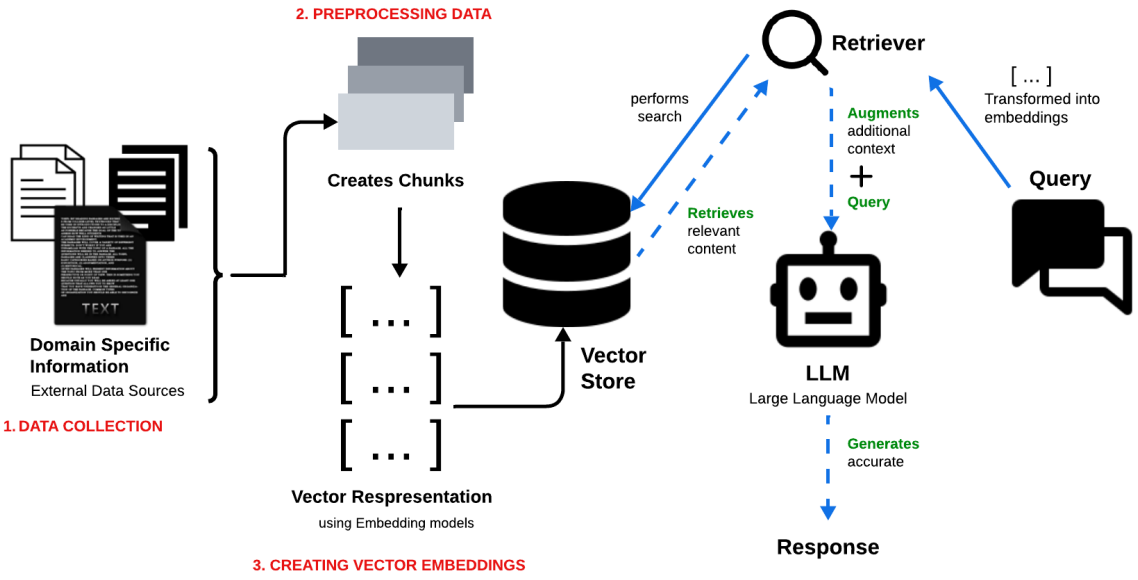

Abbildung 2.3: Architektur eines RAG-Systems [10]

2.3.1 Vector-Store

Ein Vector-Store (auch Vektordatenbank oder Vector Database genannt) ist eine spezialisierte Datenbank, die für die effiziente Speicherung, Indexierung und Abfrage hochdimensionaler Vektoren optimiert ist. Im Kontext von RAG-Systemen dienen Vector-Stores als nicht-parametrischer Speicher, der die Embedding-Repräsentationen der Dokumenten-Chunks persistiert und schnelle Ähnlichkeitssuchen ermöglicht. [7]

Die zentrale Operation eines Vector-Stores ist die **Approximate Nearest Neighbor (ANN)**-Suche, die zu einem Query-Vektor die K ähnlichsten Dokumentenvektoren findet. Da exakte

Nearest-Neighbor-Suchen bei hochdimensionalen Daten und großen Korpora rechenintensiv sind, verwenden moderne Vector-Stores approximative Algorithmen, die einen Trade-off zwischen Suchgeschwindigkeit und Genauigkeit bieten. [35]

Indexierungsstrategien

Zur effizienten Ähnlichkeitssuche werden verschiedene Indexstrukturen eingesetzt:

- **HNSW (Hierarchical Navigable Small World)**: Ein graphbasierter Ansatz, der Vektoren in einer hierarchischen Struktur organisiert. HNSW bietet hohe Suchgeschwindigkeit bei exzellenter Recall-Rate, benötigt jedoch mehr Speicherplatz für die Graphstruktur. [17]
- **IVF (Inverted File Index)**: Partitioniert den Vektorraum mittels Clustering (typischerweise k-Means) in Voronoi-Zellen. Bei der Suche werden nur Vektoren in den vielversprechendsten Partitionen verglichen, was die Suchzeit deutlich reduziert. [32]
- **Flat Index**: Führt eine erschöpfende Suche über alle Vektoren durch. Garantiert exakte Ergebnisse, skaliert jedoch schlecht bei großen Datenmengen. [5]

Ähnlichkeitsmetriken

Die Wahl der Distanz- bzw. Ähnlichkeitsmetrik beeinflusst sowohl die Retrieval-Qualität als auch die Effizienz:

- **Cosinus-Ähnlichkeit**: Misst den Winkel zwischen Vektoren, unabhängig von deren Länge. Besonders geeignet für normalisierte Embeddings.
- **Dot-Product**: Bei normalisierten Vektoren äquivalent zur Cosinus-Ähnlichkeit, jedoch effizienter zu berechnen.
- **Euklidische Distanz**: Misst die geometrische Distanz im Vektorraum. Sinnvoll, wenn die absolute Position im Raum relevant ist.

Implementierungen

Etablierte Vector-Store-Implementierungen für RAG-Systeme umfassen:

- **Chroma**: Eine eingebettete Open-Source-Vektordatenbank, die für Prototyping und kleinere Produktionssysteme optimiert ist. Unterstützt persistente Speicherung und Integration mit gängigen Embedding-Modellen.
- **FAISS** (Facebook AI Similarity Search): Eine hochoptimierte Bibliothek von Meta, die verschiedene Indextypen (IVF, HNSW, PQ) unterstützt und sowohl CPU- als auch GPU-beschleunigte Suchen ermöglicht. [5]

- **Pinecone, Weaviate, Milvus**: Verwaltete bzw. skalierbare Lösungen für produktionsreife Deployments mit Funktionen wie Sharding, Replikation und Filterung.

Für sicherheitskritische Anwendungen adressieren neuere Arbeiten wie FRAG (Federated RAG) die Herausforderungen verteilter Vektordatenbanken, bei denen Daten über mehrere Parteien verteilt sind und Datenschutzanforderungen erfüllt werden müssen. [33]

Die Wahl des Vector-Stores beeinflusst maßgeblich die Latenz und Skalierbarkeit des RAG-Systems. Für technische Datenblätter mit typischerweise einigen hundert bis tausend Chunks sind eingebettete Lösungen wie Chroma ausreichend, während größere Korpora skalierbare Architekturen erfordern. [8]

2.3.2 Embedding-Modelle für semantische Suche

Als Datenformat für den Suchindex dient ein Vektorformat: Jeder Dokument-Chunk wird durch einen hochdimensionalen Zahlenvektor (Embedding) repräsentiert. Embedding-Modelle überführen Text in dichte Vektorrepräsentationen. Für RAG-Systeme sind *Bi-Encoder*-Architekturen etabliert, die Query und Dokument unabhängig enkodieren:

- **Sentence-BERT/Sentence Transformers**: Speziell für Ähnlichkeitsvergleiche trainierte BERT-Varianten mit siamesischen Netzwerk-Architekturen. [23]
- **Multilinguale Embeddings**: Modelle wie `multilingual-e5` oder `paraphrase-multilingual-MiniLM` projizieren Texte verschiedener Sprachen in einen gemeinsamen Vektorraum. Dies ermöglicht Cross-Language Information Retrieval ohne explizite Übersetzung. Eine in Sprache A verfasste Query kann einem in Sprache B verfassten Dokument-Chunk korrekt zugeordnet werden, sofern die eingebettete Bedeutungsähnlichkeit hoch ist. [4]
- **Embedding-Normalisierung**: Normalisierte Embeddings (Einheitslänge) ermöglichen die Verwendung von Dot-Product statt Cosinus-Ähnlichkeit, was effizientere Indexstrukturen erlaubt.

Alternativ könnten alle Dokumente zunächst in eine Zielsprache (etwa Englisch) übersetzt werden, oder eingehende Anfragen vor der Vektorisierung übersetzt werden. Beide Ansätze – Cross-Language IR (CLIR) durch Übersetzung vs. Multilingual IR durch sprachübergreifende Embeddings – sind in der Forschung erprobt. [4] Ein übersetzungsbasierter Ansatz vereinheitlicht die Sprache, birgt aber das Risiko von Übersetzungsfehlern; ein embedding-basierter Ansatz erfordert ein starkes multilinguales Modell, das alle relevanten Sprachen abdeckt.

2.3.3 Retrieval-Strategien

Moderne RAG-Systeme kombinieren verschiedene Retrieval-Ansätze:

Dense Retrieval

Bei der dichten Suche werden Query und Dokumente als Vektoren repräsentiert. Die Ähnlichkeit wird typischerweise über Cosinus-Ähnlichkeit oder Dot-Product berechnet:

$$\text{sim}(q, d) = \frac{\mathbf{e}_q \cdot \mathbf{e}_d}{\|\mathbf{e}_q\| \cdot \|\mathbf{e}_d\|}$$

Embeddingmodelle überführen sowohl Dokumente als auch Anfragen in kontinuierliche Vektoren, sodass semantisch ähnliche Inhalte durch Abstandsmaße identifiziert werden können.

Sparse Retrieval (BM25)

Der BM25-Algorithmus gewichtet Terme nach ihrer Dokumentfrequenz und Termfrequenz:

$$\text{BM25}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})}$$

wobei $f(q_i, D)$ die Termfrequenz und $|D|$ die Dokumentlänge bezeichnet. Klassische IR-Systeme setzen traditionell auf term-basierte Methoden wie BM25. [20]

Hybrid Search

Hybrid-Ansätze kombinieren Dense und Sparse Retrieval durch gewichtete Fusion der Scores:

$$\text{score}_{\text{hybrid}} = \alpha \cdot \text{score}_{\text{BM25}} + (1 - \alpha) \cdot \text{score}_{\text{dense}}$$

Dies nutzt die Stärken beider Methoden: BM25 für exakte Keyword-Matches (z. B. “USB-C”, “DDR5-5600”), Dense Retrieval für semantische Ähnlichkeit. In vielen aktuellen Systemen werden hybride Suchverfahren eingesetzt, die die Vorteile beider Ansätze kombinieren. [35]

Maximal Marginal Relevance (MMR)

MMR balanciert Relevanz und Diversität der abgerufenen Dokumente:

$$\text{MMR} = \arg \max_{d_i \in R \setminus S} \left[\lambda \cdot \text{Sim}(d_i, q) - (1 - \lambda) \cdot \max_{d_j \in S} \text{Sim}(d_i, d_j) \right]$$

wobei λ den Trade-off zwischen Relevanz und Diversität steuert. [3]

2.3.4 Prompt Engineering für RAG-Systeme

Die Gestaltung des Prompts beeinflusst die Qualität der generierten Antworten erheblich:

- **Few-Shot Prompting**: Durch Beispiele im Prompt lernt das Modell das gewünschte Antwortformat. Für technische Datenblätter können Beispiele demonstrieren, wie Spezifikationen formatiert werden sollen.
- **Kontextfenster-Optimierung**: Bei begrenztem Kontextfenster müssen die relevantesten Chunks priorisiert werden. Die Reihenfolge der Chunks kann die Antwortqualität beeinflussen ("Lost in the Middle"-Phänomen). [16]
- **Instruktions-Design**: Explizite Anweisungen zur Quellennutzung, Formatierung und Umgang mit fehlenden Informationen reduzieren Halluzinationen.

Im Training eines RAG-Systems werden Retriever und Generation gemeinsam optimiert (End-to-End-Lernen). Da für die meisten Aufgaben kein direktes Label vorliegt, welches Dokument konkret abgerufen werden sollte, behandelt man die Dokumentauswahl als latente Variable. *Lewis et al.* formulierten zwei Varianten, wie über diese latenten Dokumente marginalisiert wird: [14]

1. **RAG-Sequence**: Hier nimmt das System an, dass ein einziges abgerufenes Dokument für die gesamte Antwort verantwortlich ist. Die Zielwahrscheinlichkeit für eine korrekte Ausgabe y ergibt sich näherungsweise als Summe über die Top-K Dokumente. Formal wird die bedingte Wahrscheinlichkeit folgendermaßen berechnet:

$$P_{\text{RAG-Sequence}}(y \mid x) \approx \sum_{z \in \text{Top-K}(x)} p_\eta(z \mid x)\, p_\theta(y \mid x, z) = \sum_{z \in \text{Top-K}(x)} p_\eta(z \mid x) \prod_{i=1}^{N} p_\theta(y_i \mid x, z, y_{<i}).$$

   wobei $p_\eta(z \mid x)$ die vom Retriever gelieferte Relevanzverteilung über Dokumente z darstellt und $p_\theta(y \mid x, z)$ die vom Generator modellierte Wahrscheinlichkeit ist, y zu erzeugen, wenn Dokument z als Kontext gegeben ist. Die Multiplikation über $i = 1 \ldots N$ entspricht dabei dem sequentiellen Generieren der Token von y. RAG-Sequence nutzt also *ein* Dokument pro Antwort und marginalisiert über verschiedene Kandidaten.

2. **RAG-Token**: Hier wird ein Schritt weiter gegangen: Jedes generierte Token y_i darf auf potenziell *unterschiedlichen* Dokumenten basieren. Für jedes Token wird also über die Top-K Dokumente summiert, was dem Generator erlaubt, Informationen aus mehreren Quellen in einer Antwort zusammenzuführen.

   Mathematisch ergibt sich:

$$P_{\text{RAG-Token}}(y \mid x) \approx \prod_{i=1}^{N} \sum_{z \in \text{Top-K}(x)} p_\eta(z \mid x)\, p_\theta(y_i \mid x, z, y_{<i}).$$

Also ein Produkt über alle Positionswahrscheinlichkeiten, wobei an jeder Stelle i die Beiträge aller Top-Dokumente addiert werden. In der Praxis erlaubt RAG-Token dem Modell, z. B. Satzteile aus verschiedenen Dokumenten zusammenzusetzen. Allerdings ist die Inferenz bei RAG-Token komplexer, da pro Zeitschritt über K Dokumente marginalisiert wird. Optimierte Decoding-Verfahren (z. B. *Thorough vs. Fast Decoding*) sind nötig, um die Generierung effizient zu halten.

Unabhängig von der Variante werden in RAG beide Komponenten iterativ verbessert: Der Generator lernt, die bereitgestellten externen Inhalte effektiv in seine Antworten einzubauen, während der Retriever lernt, jene Dokumente zu liefern, die dem Generator zur Lösung der Aufgabe am meisten nützen. Wichtig ist, dass während des Fine-Tunings typischerweise nur der Anfrage-Encoder des Retrievers und das Generator-Modell angepasst werden - der Dokument-Encoder, und damit der Index, bleibt oft fixiert, um nicht jedem Trainingsschritt den gesamten Korpus neu einbetten zu müssen. Trotzdem erzielt ein solches Setup bereits große Verbesserungen auf wissensintensive Benchmarks, wie *Lewis et al.* zeigten (u.a. State-of-the-Art in Open-Domain-QA). [14]

2.4 Dokumenttypen und Datenformate

Die Wissensbasis für das geplante RAG-System besteht aus einer Sammlung von Produktdatenblättern, die meist als PDF-Dateien vorliegen. PDFs stellen unstrukturierte bzw. halbstrukturierte Daten dar: obwohl sie Text, Tabellen und Bilder für Menschen lesbar anordnen, ist die semantische Struktur für Maschinen nicht unmittelbar zugänglich. Daher muss zunächst eine Extraktion der Inhalte erfolgen.

Für die technische Analyse von Produktdaten empfiehlt sich oft eine Kombination verschiedener Strategien: Wenn etwa die meisten Datenblätter auf Deutsch und Englisch vorliegen, kann man für seltene Anfrage-Sprachen maschinelle Übersetzung einsetzen, während für die Kernsprachen direkt mehrsprachige Embeddings genutzt werden. Wichtig ist außerdem, die Einheiten und Fachbegriffe einheitlich zu handhaben – etwa zu wissen, dass Wi-Fi 6 und 802.11ax im Kontext eines Netzwerktechnik-Datenblatts das Gleiche bedeuten. Solche Synonyme können teilweise im Vektorraum abgebildet werden (wenn das Modell sie semantisch nahe positioniert), müssen aber ggf. auch durch Thesauri oder Normalisierung in der Vorverarbeitung berücksichtigt werden. [21]

2.5 Evaluierung von RAG-Systemen

Die Bewertung von RAG-Systemen erfordert mehrdimensionale Metriken, da sowohl die Retrieval-Komponente als auch die Generierungskomponente evaluiert werden müssen. Grundsätzlich lassen sich zwei Evaluierungsansätze unterscheiden: **Ground Truth-basierte Evaluation** mit Referenzantworten ermöglicht objektive Metriken wie Exact Match, erfordert jedoch aufwendige manuelle Annotation. **Reference-Free Evaluation** benötigt keine Referenzantworten und evaluiert intrinsische Qualitätsaspekte wie Faithfulness oder Answer Relevancy.

2.5.1 Retrieval-Metriken

Die Qualität des Retrievals beeinflusst maßgeblich die finale Antwortqualität. Wichtige Metriken sind:

- **Context Precision**: Misst den Anteil relevanter Chunks unter den abgerufenen Dokumenten. Ein hoher Wert bedeutet, dass der Retriever präzise relevante Dokumente findet. Formal wird Context Precision berechnet als:

$$\text{Context Precision} = \frac{1}{N} \sum_{i=1}^{N} \frac{\text{Relevante Chunks bis Position } i}{i}$$

  wobei N die Anzahl der abgerufenen Chunks bezeichnet und nur relevante Chunks zur Berechnung beitragen.

- **Context Recall**: Misst, welcher Anteil der benötigten Informationen tatsächlich abgerufen wurde. Ein niedriger Recall kann dazu führen, dass wichtige Informationen fehlen. Formal wird Context Recall berechnet als:

$$\text{Context Recall} = \frac{|\text{Relevante Informationen im Kontext}|}{|\text{Gesamtzahl der Aussagen in der Referenz}|}$$

  Diese Metrik erfordert einen Gold Standard, der die vollständigen relevanten Informationen für jede Frage definiert.

- **Context Relevance**: Bewertet die Relevanz jedes einzelnen abgerufenen Chunks für die gegebene Frage, typischerweise durch LLM-as-a-Judge bestimmt. Die Metrik wird für jeden Chunk c_i einzeln bewertet und ergibt sich als Durchschnitt:

$$\text{Context Relevance} = \frac{1}{N} \sum_{i=1}^{N} \text{Relevance}(c_i, q)$$

  wobei $\text{Relevance}(c_i, q) \in [0, 1]$ die Relevanzbewertung des Chunks c_i für die Frage q darstellt. [6, 15]

2.5.2 Generierungs-Metriken

Die Qualität der generierten Antworten wird durch folgende Metriken bewertet:

- **Faithfulness**: Misst, inwieweit die generierte Antwort durch die abgerufenen Dokumente gestützt wird. Eine hohe Faithfulness bedeutet, dass die Antwort faktisch korrekt ist und nicht auf Halluzinationen basiert. Die Metrik wird durch ein LLM bewertet, das prüft, ob alle Aussagen in der Antwort durch den bereitgestellten Kontext belegt werden können. Formal ergibt sich Faithfulness als:

$$\text{Faithfulness} = \frac{|\text{Belegbare Aussagen in der Antwort}|}{|\text{Gesamte Aussagen in der Antwort}|}$$

  wobei eine Aussage als belegbar gilt, wenn sie durch den abgerufenen Kontext gestützt wird.

- **Answer Relevancy**: Bewertet, wie gut die generierte Antwort die gestellte Frage beantwortet, unabhängig von der Korrektheit. Diese Metrik bewertet die semantische Übereinstimmung zwischen Frage und Antwort. Formal wird Answer Relevancy berechnet als:

$$\text{Answer Relevancy} = \frac{1}{M} \sum_{i=1}^{M} \text{Relevancy}(a_i, q)$$

  wobei M die Anzahl der Aussagen in der Antwort bezeichnet und $\text{Relevancy}(a_i, q) \in [0, 1]$ die Relevanz der Aussage a_i für die Frage q misst. Alternativ kann Answer Relevancy auch als durchschnittliche semantische Ähnlichkeit zwischen Frage und Antwort berechnet werden.

- **Answer Correctness**: Kombiniert Faithfulness und Answer Relevancy mit einem Vergleich zur Referenzantwort (nur bei Ground Truth-basierter Evaluation verfügbar). Diese Metrik erfordert Ground Truth-Daten und ermöglicht eine objektive Bewertung der Antwortqualität. Formal ergibt sich Answer Correctness als gewichtete Kombination:

$$\text{Answer Correctness} = \alpha \cdot \text{Faithfulness} + \beta \cdot \text{Answer Relevancy} + \gamma \cdot \text{Similarity}(\text{Answer}, \text{Ground Truth})$$

  wobei α, β und γ Gewichtungsfaktoren darstellen, die typischerweise so gewählt werden, dass $\alpha + \beta + \gamma = 1$ gilt. Die Similarity-Funktion misst die semantische oder lexikalische Ähnlichkeit zwischen der generierten Antwort und der Referenzantwort. [6, 15]

2.5.3 End-to-End-Metriken

Für die Bewertung des gesamten Systems werden Metriken wie Exact Match (EM), F1-Score auf Token-Ebene oder semantische Ähnlichkeit zwischen generierter und Referenzantwort verwendet.

Frameworks wie RAGAS[1] bieten eine automatisierte Evaluation dieser Metriken mittels LLM-as-a-Judge-Ansätzen, bei denen ein LLM die Qualität von Retrieval oder Generierung bewertet. [6] Dies ermöglicht eine skalierbare Evaluierung ohne manuelle Annotation, ist jedoch abhängig von der Qualität des Judge-LLMs und kann Inkonsistenzen zwischen verschiedenen Bewertungen aufweisen.

2.6 Vergleichbare Arbeiten / State of the Art

Der Stand der Forschung zu RAG für technische Dokumente lässt sich in drei Stränge gliedern. Erstens werden Methoden genannt, die Embeddings und Retrieval auf domänenspezifische Terminologie optimieren. Arbeiten wie *Technical-Embeddings* demonstrieren, dass gezielte Query-Expansion und feinjustierte Bi-Encoder die Genauigkeit bei technischen Spezifikationen deutlich steigern. [12] Ergänzend liefern Studien wie *Chunk Twice, Embed Once* und *Mix-of-Granularity* empirische Leitplanken für effektive Segmentierungs- und Auswahlstrategien. [1, 34]

Zweitens fokussiert sich die Community auf multimodale RAG-Varianten. Fallstudien zu Produktdatenblättern zeigen, dass die Einbindung von Tabellen- und Bildinformationen die Antwortqualität steigert, insbesondere bei Spezifikationsfragen. [13] Surveys zu Multimodal-RAG-Systemen fassen Fortschritte und offene Herausforderungen zusammen und unterstreichen die Notwendigkeit gemeinsamer Benchmarks.[19] Frameworks wie VDocRAG und ViDoRAG erweitern den Horizont, indem sie Dokumente als Ganzes verarbeiten und iterative Agenten zur Kontextverfeinerung einsetzen. [25, 29]

Drittens rückt die Perspektive auf Governance und Betrieb stärker in den Vordergrund. Arbeiten zu LLM-spezifischer Nutzenbewertung und föderiertem Retrieval adressieren Nachvollziehbarkeit, Sicherheit und rollenbasierte Zugriffe. [31, 33] Gemeinsam liefern diese Forschungslinien Best Practices, die in die Konzeption eines industrietauglichen RAG-Systems einfließen und das Risiko von Halluzinationen oder Compliance-Verstößen reduzieren.

[1] Retrieval-Augmented Generation Assessment

3 Anforderungsanalyse und Konzeption

3.1 Use Case Definition

Die vorliegende Arbeit befasst sich mit der Konzeption und Implementierung eines Retrieval-Augmented Generation (RAG)-Systems zur kontextbasierten Beantwortung von Fragen auf Grundlage von Dokumentbeständen. Der primäre Anwendungsfall besteht in der Bereitstellung eines intelligenten Assistenzsystems, das Mitarbeitern ermöglicht, natürlichsprachliche Fragen zu technischen Produktdokumenten zu stellen und fundierte, quellenbasierte Antworten zu erhalten.

Im Gegensatz zu konventionellen LLMs, die ausschließlich auf ihrem Trainingskorpus basieren und somit anfällig für Halluzinationen bei domänenspezifischen Fragen sind, kombiniert das System die generative Fähigkeiten moderner Sprachmodelle mit einer präzisen Informationsextraktion aus dem hinterlegten Dokumentenbestand. Diese Hybridarchitektur adressiert das fundamentale Problem der Wissensaktualität und -genauigkeit, da das System seine Antworten stets mit den bereitgestellten Quellen belegt und somit eine nachvollziehbare Informationskette gewährleistet.

Der identifizierte Use Case umfasst dabei mehrere Teilaspekte: Zunächst muss das System in der Lage sein, heterogene Dokumentformate zu verarbeiten, darunter technische Datenblätter im PDF-Format, tabellarische Spezifikationen sowie unstrukturierte Textdokumente. Darüber hinaus soll eine mehrsprachige Unterstützung für primär deutsch- und englischsprachige Dokumente sowie Nutzeranfragen gewährleistet werden. Die Benutzerinteraktion erfolgt über eine webbasierte Oberfläche, die neben der Fragestellung auch das Management von Dokumenten sowie die Visualisierung der Quellverweise ermöglicht.

3.2 Anforderungen

Aus der Use Case Analyse lassen sich sowohl funktionale als auch nicht-funktionale Anforderungen ableiten, die als Grundlage für die nachfolgende Architekturentscheidung dienen.

3.2.1 Funktionale Anforderungen

Die primäre funktionale Anforderung besteht in der Fähigkeit des Systems, natürlichsprachliche Fragen entgegenzunehmen und kontextrelevante Antworten auf Basis der indizierten Dokumente zu generieren. Dabei soll das System explizit die verwendeten Quellabschnitte referenzieren, um die Nachvollziehbarkeit der Antworten sicherzustellen. Eine weitere zentrale Anforderung betrifft die Unterstützung multipler Dokumentformate. Das System muss PDF-Dokumente einschließlich

eingebetteter Tabellen und Bilder mittels optischer Zeichenerkennung (OCR) verarbeiten können, ebenso wie Microsoft Word-Dokumente.

Darüber hinaus erfordert das System die Verarbeitung weiterer Dokumentformate, um eine möglichst breite Abdeckung von Dokumenttypen zu gewährleisten. Konkret werden die Formate TXT, Markdown, CSV, Excel (XLSX), HTML und JSON unterstützt, wobei für jedes Format spezialisierte Loader-Komponenten implementiert wurden. Für PDF-Dokumente wurde eine erweiterte Verarbeitungspipeline entwickelt, die auf PyMuPDF mit Layout-Analyse basiert und insbesondere die Extraktion von Tabellenstrukturen sowie die automatische Erkennung und Entfernung von Kopf- und Fußzeilen ermöglicht.

Eine weitere funktionale Anforderung besteht in der Fähigkeit zur kontextuellen Beantwortung von Fragen durch die Integration einer Chat-Historie. Das System speichert die letzten zehn Nachrichten einer Konversation und nutzt diese als Kontext für nachfolgende Anfragen, um kohärente Mehrfachrunden-Dialoge zu ermöglichen. Zusätzlich werden alle Anfragen und generierten Antworten in einer *QueryHistory* persistiert, um eine Nachvollziehbarkeit der Systemnutzung zu gewährleisten.

Das System implementiert ferner ein zweistufiges Retrieval-Verfahren, bestehend aus einer initialen semantischen Suche auf Basis von Embeddings sowie einem optionalen Reranking-Schritt mittels Cross-Encoder-Modellen. Dieses Verfahren ermöglicht eine präzisere Identifikation relevanter Dokumentenabschnitte, insbesondere bei komplexen Anfragen oder bei der Suche nach spezifischen technischen Spezifikationen.

3.2.2 Nicht-funktionale Anforderungen

Hinsichtlich der nicht-funktionalen Anforderungen steht die Antwortqualität im Vordergrund. Das System soll eine hohe semantische Genauigkeit bei der Dokumentensuche erreichen und Halluzinationen des Sprachmodells durch strikte Kontextbindung minimieren. Die Skalierbarkeit bildet eine weitere wesentliche Anforderung, wobei das System für kleine bis mittlere Dokumentenbestände von bis zu mehreren tausend Dokumenten ausgelegt sein soll. Die Wartbarkeit wird durch eine modulare Architektur mit klar definierten Schnittstellen zwischen den Komponenten sichergestellt. Schließlich muss das System eine angemessene Antwortzeit von wenigen Sekunden pro Anfrage gewährleisten, um eine akzeptable Benutzererfahrung zu bieten.

Die Konfigurierbarkeit des Systems stellt eine weitere wichtige nicht-funktionale Anforderung dar. Alle kritischen Parameter der RAG-Pipeline, wie Chunk-Größe, Overlap-Bereiche, Retrieval-Parameter sowie LLM-Konfigurationen, sind über eine zentrale YAML-Konfigurationsdatei steuerbar. Diese Parametrisierung ermöglicht es, das System ohne Code-Änderungen an verschiedene Anwendungsfälle anzupassen und Optimierungsexperimente durchzuführen.

Die Beobachtbarkeit des Systems wird durch ein umfassendes Logging-Subsystem gewährleistet, das strukturierte Logs im JSON-Format erzeugt. Das Logging umfasst sowohl Anwendungsereignisse als auch detaillierte Protokollierung aller RAG-Interaktionen, einschließlich Zeitmetri-

ken für einzelne Pipeline-Komponenten. Diese Logs werden automatisch rotiert und für einen konfigurierbaren Zeitraum aufbewahrt, um eine spätere Analyse und Fehlerdiagnose zu ermöglichen.

Robustheit und Fehlerbehandlung bilden weitere zentrale Anforderungen. Das System implementiert umfassende Fehlerbehandlungsmechanismen mit Fallback-Strategien, beispielsweise bei fehlgeschlagenen Modellinitialisierungen oder bei der Verarbeitung korrupter Dokumente. Fehlerhafte Verarbeitungsschritte werden detailliert protokolliert, ohne dass das gesamte System beeinträchtigt wird.

3.3 Datenbasis

Die Datenbasis des entwickelten Systems setzt sich aus zwei komplementären Speicherformen zusammen, die unterschiedliche Aspekte der Informationsverwaltung adressieren. Die relationale Datenbankkomponente, realisiert mittels SQLite und dem objektrelationalen Mapper SQLAlchemy, verwaltet die strukturierten Metadaten des Systems. Dies umfasst Dokumentenmetadaten wie Dateinamen, Dateitypen und Verarbeitungsstatus sowie die Chunk-Informationen, die eine Rückverfolgung der generierten Textabschnitte zu ihren Ursprungsdokumenten ermöglichen. Das Datenmodell folgt dabei einem klassischen relationalen Entwurf mit den Entitäten Document, Chunk und *QueryHistory*, wobei Beziehungen über Fremdschlüssel modelliert werden.

Die Chunk-Entität speichert zusätzlich zu den Textinhalten auch Bounding-Box-Informationen für PDF-Dokumente, die eine präzise Positionierung der Textabschnitte innerhalb der ursprünglichen Dokumentseiten ermöglichen. Diese Positionsinformationen werden für die Visualisierung von Quellverweisen in der Benutzeroberfläche genutzt, indem relevante Textpassagen direkt im PDF-Dokument hervorgehoben werden können. Die QueryHistory-Entität erfasst neben den Anfragen und Antworten auch Metadaten wie Zeitstempel und verwendete Quell-Chunks, um eine vollständige Nachvollziehbarkeit der Systemnutzung zu gewährleisten.

Die Vektordatenbank ChromaDB bildet das semantische Rückgrat des Retrieval-Systems. In ihr werden die durch das Embedding-Modell erzeugten hochdimensionalen Vektorrepräsentationen der Textabschnitte persistent gespeichert. ChromaDB wurde aufgrund seiner nativen Python-Integration, der persistenten lokalen Speicherung ohne externe Datenbankserver sowie der effizienten Implementierung von Nearest-Neighbor-Suchen für die semantische Ähnlichkeitsberechnung ausgewählt. Die Speicherung erfolgt in einer Collection, wobei jeder Vektor mit dem zugehörigen Originaltext und Metadaten wie Dokument-ID und Seitenzahl assoziiert wird. Die Collection wird mit dem Embedding-Modell und den Dokumenten-Chunks initialisiert und kann dann dynamisch mit neuen Chunks erweitert werden. Die Abfrage der semantisch ähnlichsten Vektoren erfolgt über die Funktion `similarity_search` von ChromaDB, die die Top-K ähnlichsten Vektoren zurückgibt.

Für die Generierung der Embeddings wird das Modell `sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2` verwendet, das hochdimensionale Vektorrepräsentationen mit

384 Dimensionen erzeugt. Dieses Modell wurde aufgrund seiner guten Balance zwischen Qualität und Performance sowie seiner Unterstützung für multilinguale Texte ausgewählt. Die Embeddings werden sowohl für die Indexierung neuer Dokumente als auch für die Vektorisierung von Nutzeranfragen verwendet, um eine semantische Ähnlichkeitssuche zu ermöglichen.

Zur Verbesserung der Retrieval-Qualität wird zusätzlich ein Reranking-Verfahren implementiert, das auf Cross-Encoder-Modellen basiert. Konkret kommt das Modell `cross-encoder/ms-marco-MiniLM-L-6-v2` zum Einsatz, das die Relevanz der initial durch semantische Suche identifizierten Dokumentenabschnitte bezüglich der Nutzeranfrage neu bewertet. Dieses zweistufige Verfahren ermöglicht es, zunächst eine größere Anzahl von Kandidaten zu identifizieren und anschließend die relevantesten Abschnitte zu selektieren, was insbesondere bei komplexen oder spezifischen Anfragen zu verbesserten Ergebnissen führt.

Für die Generierung der finalen Antworten wird das Sprachmodell GPT-4o-mini von OpenAI verwendet, das durch seine Fähigkeit zur kontextuellen Textgenerierung sowie durch seine Effizienz in Bezug auf Antwortzeit und Kosten ausgewählt wurde. Das Modell wird mit einer niedrigen Temperatur von 0,3 konfiguriert, um Halluzinationen zu minimieren und die Konsistenz der Antworten zu erhöhen. Die maximale Token-Anzahl pro Antwort beträgt 2000 Tokens, um auch detaillierte Antworten zu komplexen Fragen zu ermöglichen.

3.4 Systemarchitektur

Die Gesamtarchitektur des entwickelten RAG-Systems folgt dem Prinzip der Schichtenarchitektur und gliedert sich in vier klar voneinander abgegrenzte Schichten, die durch definierte Schnittstellen miteinander kommunizieren.

3.4.1 Präsentationsschicht

Die Präsentationsschicht bildet die Benutzerschnittstelle des Systems und wurde unter Verwendung des Streamlit-Frameworks implementiert. Streamlit ermöglicht die schnelle Entwicklung datengetriebener Webanwendungen in Python und bietet native Unterstützung für interaktive Elemente wie Chat-Interfaces, Datei-Uploads und Datenvisualisierungen. Die Präsentationsschicht kommuniziert ausschließlich über HTTP-Anfragen mit der darunterliegenden API-Schicht und hält selbst keine Geschäftslogik vor.

Die Streamlit-Anwendung ist in mehrere Seiten unterteilt, die unterschiedliche Funktionalitäten des Systems abdecken. Die Dashboard-Seite bietet eine Übersicht über den aktuellen Dokumentenbestand, zeigt Verarbeitungsstatus der Dokumente an und ermöglicht das Hochladen sowie die Verwaltung von Dokumenten. Die Chat-Seite implementiert ein interaktives Chat-Interface, in dem Nutzer Fragen stellen können und Antworten mit Quellverweisen erhalten.

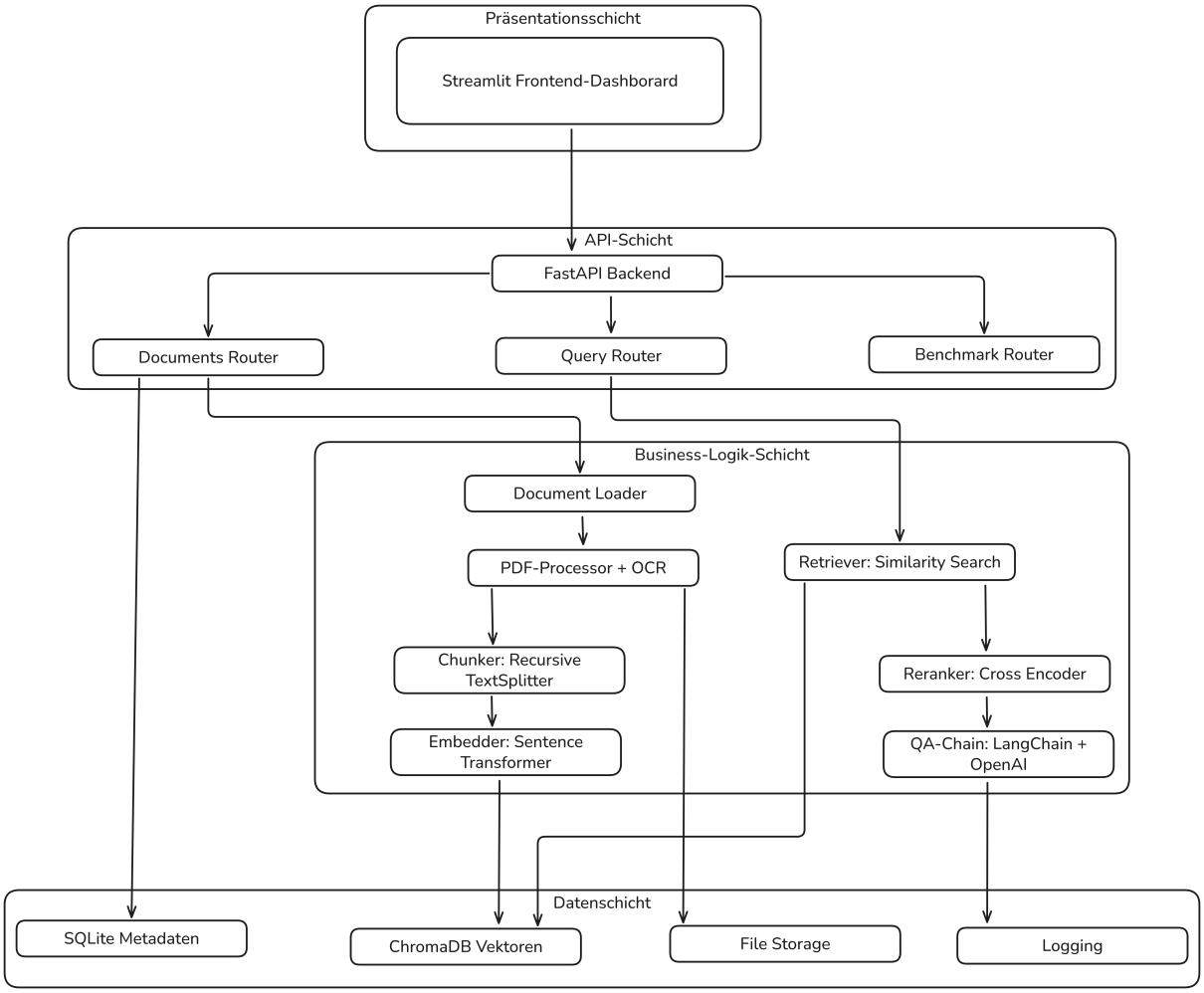

Abbildung 3.1: Systemarchitektur

3.4.2 API-Schicht

Die API-Schicht stellt eine RESTful[1]-Schnittstelle auf Basis des FastAPI-Frameworks bereit. FastAPI wurde aufgrund seiner hohen Performance durch asynchrone Verarbeitung, der automatischen Generierung von OpenAPI-Dokumentation sowie der nativen Unterstützung für Pydantic-Datenvalidierung ausgewählt. Die API-Schicht ist in drei logische Router unterteilt:

- **Documents-Router**: Handhabt den Dokumenten-Upload und die Indexierungsprozesse.
- **Query-Router**: Nimmt RAG-Anfragen entgegen und koordiniert die Antwortgenerierung.
- **Benchmark-Router**: Ermöglicht die Durchführung von RAGAS-Evaluierungen über die API.

Der Documents-Router bietet Endpunkte für das Hochladen von Dokumenten, die Initiierung der Indexierung sowie die Abfrage des Verarbeitungsstatus. Zusätzlich ermöglicht er die Abfrage von Chunk-Informationen für einzelne Dokumente sowie die Löschung von Dokumenten aus dem

[1]Representational State Transfer

System. Der Query-Router verarbeitet RAG-Anfragen, koordiniert die Retrieval- und Generierungsprozesse und gibt strukturierte Antworten mit Quellverweisen zurück. Der Benchmark-Router ermöglicht sowohl die manuelle Durchführung von Evaluierungen mit bereitgestellten Fragen und Antworten als auch die automatische Evaluierung basierend auf einem Gold-Standard-Datensatz.

Die API-Schicht implementiert CORS-Middleware[2], um Cross-Origin-Anfragen von verschiedenen Frontend-Implementierungen zu ermöglichen. Zusätzlich werden alle Anfragen durch Dependency Injection mit Datenbankverbindungen versorgt, wobei SQLAlchemy-Sessions automatisch verwaltet werden. Die API generiert automatisch eine interaktive OpenAPI-Dokumentation, die unter dem Endpunkt `/docs` verfügbar ist und eine direkte Testmöglichkeit aller Endpunkte bietet.

3.4.3 Business-Logik-Schicht

Die Business-Logik-Schicht kapselt die Kernfunktionalität des RAG-Systems in spezialisierten Modulen. Diese Schicht implementiert die gesamte Verarbeitungspipeline von Dokumentenextraktion über die Vektorisierung bis hin zur Antwortgenerierung. Die einzelnen Module sind dabei lose gekoppelt und kommunizieren über definierte Python-Schnittstellen, was die Austauschbarkeit einzelner Komponenten - beispielsweise des Embedding-Modells - ermöglicht.

Die Ingestion-Komponente besteht aus dem `DocumentLoader`, der verschiedene Dokumentformate unterstützt, sowie dem `PDFProcessorAdvanced`, der speziell für die Verarbeitung von PDF-Dokumenten optimiert ist. Der PDF-Prozessor nutzt PyMuPDF und pymupdf4llm mit Layout-Analyse-Funktionalitäten, um Tabellenstrukturen zu erkennen und zu extrahieren sowie Kopf- und Fußzeilen automatisch zu entfernen. Die Extraktion erfolgt wahlweise im Text- oder Markdown-Format, wobei Markdown bevorzugt wird, da es die Struktur von Tabellen besser erhält.

Die Chunking-Komponente verwendet den `RecursiveCharacterTextSplitter` [26] aus dem LangChain-Framework, der Texte anhand konfigurierbarer Separator-Hierarchien in Abschnitte unterteilt. Die Standardkonfiguration verwendet eine Chunk-Größe von 1200 Zeichen mit einem Overlap von 200 Zeichen, um sicherzustellen, dass zusammengehörige Informationen nicht an Chunk-Grenzen getrennt werden. Für PDF-Dokumente werden zusätzlich Seiteninformationen und Bounding-Box-Koordinaten gespeichert, um eine präzise Positionierung der Chunks zu ermöglichen.

Die Embedding-Komponente kapselt das SentenceTransformer-Modell und bietet Methoden zur Vektorisierung einzelner Texte sowie zur Batch-Verarbeitung mehrerer Texte. Die generierten Embeddings werden zusammen mit Metadaten in der ChromaDB-Vektordatenbank gespeichert, wobei jeder Vektor mit einer eindeutigen ID und den zugehörigen Dokument- und Seiteninformationen verknüpft wird.

[2]Cross-Origin Resource Sharing

Die Retrieval-Komponente implementiert die semantische Suche über die Vektordatenbank und bietet spezielle Optimierungen für verschiedene Fragetypen. Insbesondere für Anfragen nach technischen Spezifikationen werden erweiterte Retrieval-Strategien angewendet, die eine größere Anzahl von Kandidaten identifizieren, um sicherzustellen, dass relevante technische Details nicht übersehen werden. Die Retrieval-Komponente kann optional mit einem Reranker kombiniert werden, der die initial identifizierten Kandidaten nach Relevanz neu bewertet.

Die Reranking-Komponente verwendet Cross-Encoder-Modelle, die eine präzisere Bewertung der Relevanz ermöglichen als die initiale semantische Suche, da sie sowohl die Anfrage als auch die Dokumentenabschnitte gemeinsam verarbeiten. Das Reranking erfolgt optional und wird erst bei großen Dokumentbeständen aktiviert, um die Antwortzeit zu optimieren.

Die QA-Chain-Komponente koordiniert die finale Antwortgenerierung durch das Sprachmodell. Sie formatiert die retrieved Dokumentenabschnitte als Kontext, integriert optional die vorhandene Chat-Historie und konstruiert spezifische System-Prompts, die das Sprachmodell anweisen, ausschließlich Informationen aus dem bereitgestellten Kontext zu verwenden. Die Prompts enthalten detaillierte Anweisungen zur Vermeidung von Halluzinationen und zur Priorisierung wichtiger technischer Spezifikationen bei entsprechenden Anfragen.

3.4.4 Datenschicht

Die Datenschicht vereint die persistenten Speicherkomponenten des Systems. Neben der bereits erläuterten relationalen Datenbank und der Vektordatenbank umfasst sie auch das Dateisystem für die Speicherung der originalen Dokumentdateien sowie das Logging-Subsystem für die strukturierte Protokollierung von Anwendungsereignissen und RAG-Interaktionen.

Das Dateisystem wird für die persistente Speicherung der hochgeladenen Originaldokumente verwendet, wobei jedes Dokument mit einer eindeutigen UUID-basierten Kennung versehen wird. Die Dokumente werden in einem konfigurierbaren Upload-Verzeichnis gespeichert, wobei die Pfadinformationen in der relationalen Datenbank referenziert werden. Zusätzlich werden Benchmark-Ergebnisse und Visualisierungen in einem separaten Verzeichnis persistiert, um eine langfristige Nachvollziehbarkeit von Evaluierungsergebnissen zu gewährleisten.

Das Logging-Subsystem verwendet strukturiertes Logging im JSON-Format, um eine maschinelle Verarbeitung der Logs zu ermöglichen. Es werden zwei separate Log-Streams verwaltet: Ein Anwendungslog für allgemeine Systemereignisse und ein Interaktionslog, das detaillierte Informationen über alle RAG-Anfragen und -Antworten erfasst. Die Logs werden automatisch rotiert, wenn eine konfigurierbare Größe erreicht wird, und für einen definierten Zeitraum aufbewahrt, bevor sie automatisch gelöscht werden. Diese Protokollierung ermöglicht sowohl die Fehlerdiagnose als auch die Analyse der Systemnutzung und Performance.

3.5 Verarbeitungspipeline

Die Verarbeitungspipeline des RAG-Systems gliedert sich in zwei Hauptprozesse: die Dokumentenindexierung und die Anfrageverarbeitung.

3.5.1 Dokumentenindexierung

Die Indexierung neuer Dokumente erfolgt in mehreren aufeinander aufbauenden Schritten. Zunächst wird das hochgeladene Dokument durch den entsprechenden Loader basierend auf seinem Dateityp verarbeitet. Für PDF-Dokumente wird der erweiterte PDF-Prozessor verwendet, der Layout-Analysen durchführt und Tabellenstrukturen erkennt, während andere Formate durch spezialisierte Loader-Komponenten verarbeitet werden. Der extrahierte Text wird anschließend durch den Chunker in semantisch sinnvolle Abschnitte unterteilt, wobei Metadaten wie Seitenzahlen und Positionsinformationen erhalten bleiben.

Die generierten Chunks werden zunächst in der relationalen Datenbank persistiert, um eine vollständige Nachvollziehbarkeit zu gewährleisten. Anschließend werden die Textabschnitte durch das Embedding-Modell vektorisiert, wobei die Batch-Verarbeitung genutzt wird, um die Effizienz zu erhöhen. Die generierten Vektoren werden zusammen mit den Originaltexten und Metadaten in der ChromaDB-Vektordatenbank gespeichert, wobei jeder Vektor mit einer eindeutigen ID versehen wird, die auch in der relationalen Datenbank referenziert wird. Während des gesamten Indexierungsprozesses wird der Dokumentenstatus in der Datenbank verwaltet, um den aktuellen Verarbeitungsstand zu verfolgen: zunächst wird der Status auf *processing* gesetzt, bei erfolgreichem Abschluss auf *indexed* und bei Fehlern auf *error*, wodurch eine zuverlässige Fehlerbehandlung und Nachverfolgbarkeit gewährleistet wird. Nach erfolgreicher Indexierung wird der Dokumentenstatus in der Datenbank aktualisiert, um den Abschluss des Prozesses zu signalisieren.

3.5.2 Anfrageverarbeitung

Die Verarbeitung einer Nutzeranfrage durchläuft mehrere Stufen der Informationsverarbeitung. Zunächst wird die Anfrage durch das gleiche Embedding-Modell vektorisiert, das auch für die Indexierung verwendet wurde, um eine konsistente semantische Repräsentation zu gewährleisten. Der Vektor der Anfrage wird anschließend für die semantische Suche in der ChromaDB verwendet, wobei die Top-K ähnlichsten Dokumentenabschnitte identifiziert werden.

Das System implementiert adaptive Retrieval-Strategien, die je nach Fragetyp unterschiedliche Parameter verwenden. Für spezifische Fragetypen wie Prozessor-Spezifikationen oder Screen-to-Body-Ratio-Abfragen werden erweiterte Retrieval-Parameter angewendet, die eine größere Anzahl von Kandidaten identifizieren, um sicherzustellen, dass relevante Informationen nicht

übersehen werden. Diese Strategien berücksichtigen die Besonderheiten verschiedener Fragetypen, beispielsweise die Tatsache, dass Prozessor-Tabellen häufig über mehrere Chunks verteilt sind.

Optional kann ein Reranking-Schritt durchgeführt werden, bei dem die initial identifizierten Kandidaten durch ein Cross-Encoder-Modell neu bewertet werden. Dieser Schritt verbessert die Relevanz der final ausgewählten Dokumentenabschnitte, erfordert jedoch zusätzliche Verarbeitungszeit. Für bestimmte Fragetypen, insbesondere technische Spezifikationsanfragen, wird das Reranking gezielt deaktiviert, um eine breitere Abdeckung relevanter Dokumentenabschnitte zu gewährleisten. Die retrieved Dokumentenabschnitte werden anschließend formatiert und mit Metadaten angereichert, um sie als Kontext für das Sprachmodell aufzubereiten.

Die QA-Chain-Komponente konstruiert einen System-Prompt, der das Sprachmodell anweist, ausschließlich Informationen aus dem bereitgestellten Kontext zu verwenden und Halluzinationen zu vermeiden. Für Spezifikations-Fragen werden spezialisierte Prompts mit detaillierten Anweisungen verwendet, die das Sprachmodell anleiten, systematisch alle bereitgestellten Chunks zu durchsuchen und modellspezifische Informationen zu priorisieren. Optional wird die vorhandene Chat-Historie in den Prompt integriert, wobei die letzten zehn Nachrichten einer Konversation als Kontext verwendet werden, um kontextuelle Mehrfachrunden-Dialoge zu ermöglichen.

Um Token-Limits des Sprachmodells nicht zu überschreiten, implementiert das System eine intelligente Chunk-Limitierung basierend auf Token-Schätzungen. Die Anzahl der an das Sprachmodell übergebenen Chunks wird dynamisch angepasst, wobei wichtige Spezifikations-Chunks priorisiert werden, um eine optimale Balance zwischen Kontextumfang und Antwortqualität zu gewährleisten.

Das Sprachmodell generiert anschließend die finale Antwort, die zusammen mit den Quellverweisen an den Nutzer zurückgegeben wird. Nach der Antwortgenerierung werden nur die tatsächlich in der Antwort referenzierten Quellen extrahiert und zurückgegeben, wodurch die Relevanz der Quellenangaben erhöht und die Antworttransparenz verbessert wird. Die gesamte Anfrage und Antwort werden in der *QueryHistory* gespeichert, um eine Nachvollziehbarkeit zu gewährleisten.

4 Implementierung

4.1 Datenverarbeitung

Die Datenverarbeitung bildet den ersten Schritt der RAG-Pipeline und umfasst die Extraktion, Aufbereitung und Segmentierung von Dokumentinhalten. Die Qualität dieser Vorbereitungsphase hat unmittelbaren Einfluss auf die nachfolgende Retrieval-Genauigkeit und damit auf die Gesamtqualität des Systems.

4.1.1 Multi-Format Document Loader

Die Klasse *DocumentLoader* implementiert eine Fassade für die einheitliche Verarbeitung heterogener Dokumentformate. Der Entwurf folgt dem Strategy-pattern, bei dem für jedes unterstütztes Dateiformat eine spezialisierte Loader-Strategie definiert ist. Die zentrale Methode *load_ document* ermittelt zunächst die Dateierweiterung und delegiert anschließend an die formatspezifische Implementierung.

Für textbasierte Formate wie TXT und Markdown erfolgt die Extraktion durch direktes Einlesen der Dateiinhalte unter Verwendung der UTF-8-Zeichenkodierung. Microsoft Word-Dokumente im DOCX-Format werden mittels der python-docx-Bibliothek verarbeitet, wobei die Absatzstruktur erhalten bleibt und zu einem zusammenhängenden Text konkatiniert wird. Tabellarische Daten in CSV- und Excel-Formaten werden über die Pandas-Bibliothek geladen und in eine textuelle Repräsentation überführt, die die Spaltenstruktur durch Leerzeichen-Trennung bewahrt. HTML-Dokumente durchlaufen eine Bereinigung mittels BeautifulSoup, wobei Markup-Tags entfernt und der reine Textinhalt extrahiert wird. JSON-Dokumente werden geparst und mit Einrückungen re-serialisiert, um eine lesbare Textform zu erzeugen.

Die einheitliche Rückgabestruktur aller Ladestrategien besteht aus einem Dictionary mit dem Schlüssel *text* für den extrahierten Volltext sowie *metadata* für formatspezifische Zusatzinformationen wie Seitenanzahl, Zeilenanzahl oder Spaltenbezeichnungen. Diese Vereinheitlichung ermöglicht die nahtlose Weiterverarbeitung durch die nachfolgenden Pipeline-Stufen unabhängig vom Ursprungsformat.

4.1.2 PDF-Verarbeitung mit optischer Zeichenerkennung (OCR)

Die Verarbeitung von PDF-Dokumenten stellt aufgrund der Komplexität dieses Formats besondere Anforderungen. PDFs können Text in verschiedenen Kodierungen, eingebettete Schriftarten, Tabellen ohne explizite Strukturinformationen sowie Bildbereiche mit relevanten Textinhalten enthalten. Die Klasse *PDFProcessor* adressiert diese Herausforderungen durch eine mehrstufige Extraktionsstrategie.

Die primäre Textextraktion erfolgt über die PyMuPDF-Bibliothek (fitz), die einen low-level-Zugriff auf die PDF-Interna ermöglicht. Für jede Seite wird zunächst die Methode *get_ text("dict")* aufgerufen, die eine strukturierte Repräsentation der Textblöcke mit ihren Bounding-Boxes liefert.

Parallel zur Textextraktion werden mittels *get_ images()* alle eingebetteten Bilder identifiziert. Für jedes Bild wird eine optische Zeichenerkennung (OCR) mittels Tesseract durchgeführt, um potentiell bildbasierte Textinhalte - etwa eingescannte Tabellen oder Diagrammbeschriftungen - zu erschließen. Die Tesseract-Konfiguration unterstützt dabei sowohl deutsche als auch englische Zeichenerkennung zur Gewährleistung mehrsprachiger Dokumentverarbeitung. Der erkannte Text wird dem Seiteninhalt hinzugefügt und entsprechend markiert, um bei Bedarf zwischen nativem PDF-Text und OCR-Text differenzieren zu können.

Die plattformübergreifende Kompatibilität wird durch eine automatische Erkennung des Tesseract-Installationspfades sichergestellt. Das System prüft zunächst die PATH-Umgebungsvariable und durchsucht anschließend typische Installationsverzeichnisse unter Windows sowie Linux / macOS.

4.1.3 Text-Segmentierung mittels Chunking

Die Segmentierung großer Dokumente in kleinere, semantisch kohärente Textabschnitte, sogenannte Chunks, ist ein kritischer Schritt für die Qualität des Retrieval-Prozesses. Zu große Chunks führen zu verdünnter semantischer Repräsentation und können die Präzision der Ähnlichkeitssuche beeinträchtigen, während zu kleine Chunks den notwendigen Kontext für eine sinnvolle Antwortgenerierung verlieren.

Die implementierte Chunking-Strategie basiert auf dem *RecursiveCharacterTextSplitter* [26] aus dem LangChain-Framework. Dieser Splitter versucht, Text zunächst an semantisch sinnvollen Stellen wie Absatzgrenzen zu teilen und greift erst bei Bedarf auf feinere Trennzeichen wie Zeilenumbrüche, Satzenden und schließlich Wortgrenzen zurück. Die rekursive Vorgehensweise gewährleistet, dass Textblöcke möglichst an natürlichen semantischen Grenzen getrennt werden, statt willkürliche Schnitte bei exakt erreichter Zeichenzahl vorzunehmen.

Die Konfigurationsparameter wurden empirisch für den Anwendungsfall technischer Spezifikationsdokumente optimiert. Eine Chunk-Größe von 1200 Zeichen erwies sich als geeigneter Kompromiss zwischen Kontexterhaltung und Retrieval-Präzision. Insbesondere technische Tabellen und Spezifikationslisten bleiben bei dieser Größe häufig als zusammenhängende Einheiten erhalten. Die Überlappung von 200 Zeichen stellt sicher, dass an Chunk-Grenzen keine relevanten Informationen durch Kontextverlust verloren gehen.

Für jedes generierte Chunk-Objekt werden neben dem Textinhalt auch Metadaten wie der Chunk-Index innerhalb des Dokuments und die zugehörige Seitenzahl. Eine eindeutige UUID dient als primärer Identifikator und ermöglicht die Verknüpfung zwischen der relationalen Datenbank und der Vektordatenbank.

4.2 Indexierung & Retrieval

Die Indexierung überführt die textuellen Dokumentfragmente in eine durchsuchbare Vektorrepräsentation, während der Retrieval-Prozess zur Laufzeit die semantisch relevantesten Dokumente für eine gegebene Anfrage identifiziert. Diese Komponenten bilden das informationstheoretische Kernstück des RAG-Systems.

4.2.1 Embedding-Generierung mittels Sentence Transformers

Die Überführung natürlichsprachlicher Texte in dichte Vektorrepräsentationen erfolgt durch ein vortrainiertes Embedding-Modell auf Basis der Sentence-BERT-Architektur. [23] Die Klasse *Embedder* kapselt die Interaktion mit dem SentenceTransformer-Framework und stellt Methoden zur Einzeltext- sowie Batch-Vektorisierung bereit.

Als Standard-Embedding-Modell kommt paraphrase-multilingual-MiniLM-L12-v2 [30] zum Einsatz, ein kompaktes Modell mit 384-dimensionalen Ausgabevektoren. Dieses Modell bietet einen ausgewogenen Kompromiss zwischen Embedding-Qualität und Inferenzgeschwindigkeit. Mit einer Verarbeitungskapazität von 14.000 Sätzen pro Sekunde auf moderner Hardware ermöglicht es die effiziente Indexierung auch umfangreicher Dokumentbestände. Die relativ geringe Dimensionalität der Vektoren reduziert zudem den Speicherbedarf der Vektordatenbank und beschleunigt Ähnlichkeitssuche.

Die Wahl eines Bi-Encoder-Modells gegenüber einem Cross-Encoder für die initiale Embedding-Generierung ist architektonisch begründet: Bi-Encoder berechnen die Embeddings von Query und Dokumenten unabhängig voneinander, wodurch die Dokumenten-Embeddings vorberechnet und persistent gespeichert werden können. Bei einer Suchanfrage muss lediglich der Query-Vektor berechnet und mit den bereits indizierten Dokumentvektoren verglichen werden. Diese Architektur ermöglicht schnelle Antwortzeiten auch bei großen Dokumentbeständen.

Tabelle 4.1: Überblick über die verwendeten Embedding-Modelle

Modell	Dimensionen	Parameter	Basis-Architektur
all-MiniLM-L6-v2	384	22.7 M	MiniLM-L6-H384-uncased
all-mpnet-base-v2	768	0.1 B	MPNet
paraphrase-multilingual-MiniLM-L12-v2	384	0.1 B	MiniLM (Sentence-BERT)
multilingual-e5-large	1024	0.6 B	XLM-RoBERTa-large

4.2.2 Vektorindexierung mit ChromaDB

Die persistente Speicherung der Embeddings erfolgt in ChromaDB, einer eingebetteten Vektordatenbank mit Fokus auf Entwicklerfreundlichkeit und native Python-Integration. Die Klasse

VectorStore abstrahiert die ChromaDB-Operationen hinter einer domänenspezifischen Schnittstelle.

Bei der Initialisierung wird ein PersistentClient instanziiert, der die Vektordaten im Dateisystem ablegt und somit Persistenz über Anwendungsneustarts hinweg gewährleistet. Die Konfiguration deaktiviert explizit die anonymisierte Telemetrie, um datenschutzrechtlichen Anforderungen in Unternehmensumgebungen zu genügen.

Die Dokumentindizierung erfolgt über die Methode *add_ documents()*, die Texte, Embeddings, Metadaten und eindeutige Identifikatoren entgegennimmt. ChromaDB speichert diese Informationen in einer Collection und baut intern einen Index für effiziente Nearest-Neighbor-Suchen auf. Die Abfrageoption *query()* akzeptiert Query-Embeddings und gibt die K ähnlichsten Dokumente mit ihren Distanzwerten, Originaltexten und Metadaten zurück. Die Verwendung der Kosinus-Distanz als Ähnlichkeitsmaß hat sich für textuelle Embeddings als besonders geeignet erwiesen, da sie die Richtung der Vektoren priorisiert und robust gegenüber Längenvariationen ist.

4.2.3 Semantische Suche mit Query Expansion

Der Retrieval-Prozess geht über eine einfache Vektorähnlichkeitssuche hinaus und implementiert mehrere Optimierungsstrategien zur Verbesserung der Ergebnisqualität. Die Klasse *Retriever* koordiniert diese Strategien und bildet die Schnittstelle zwischen der Vektordatenbank und der Antwortgenerierung.

Eine zentrale Optimierung stellt die Query Expansion für Spezifikationsanfragen dar. Das System erkennt anhand von Schlüsselwörtern und Fragepattern, ob eine Anfrage auf technische Spezifikationen abzielt - etwa Fragen nach Prozessoren, Arbeitsspeicher oder Displayeigenschaften. Für solche Anfragen wird der ursprüngliche Query-Text um domänenspezifische Fachbegriffe erweitert, bevor die Embedding-Berechnung erfolgt. Diese Anreicherung verbessert die semantische Übereinstimmung mit technischen Dokumentabschnitten, die zwar relevante Informationen enthalten, aber möglicherweise abweichende Terminologie verwenden.

Darüber hinaus implementiert der Retriever ein differenziertes Similarity-Boosting, das die initiale Ähnlichkeitsbewertung kontextabhängig anpasst. Textabschnitte, die charakteristische Merkmale technischer Spezifikationen aufweisen – etwa Einheitsangaben wie GHz, GB oder mm - erhalten einen Relevanzbonus, der ihre Positionierung im Ranking verbessert. Diese Heuristik adressiert das Problem, dass semantisch hochrelevante technische Abschnitte aufgrund ihrer tabellarischen oder listenartigen Struktur mitunter geringere Cosinus-Ähnlichkeitswerte zu natürlichsprachlichen Anfragen aufweisen.

Für Anfragen, die auf spezifische Produktmodelle referenzieren, aktiviert der Retriever zusätzlich eine Produktfilterung. Mittels regulärer Ausdrücke werden Modellbezeichnungen und Generationsangaben aus der Anfrage extrahiert und zur Einschränkung der Ergebnismenge verwendet.

Diese Filterung verhindert die Vermischung von Spezifikationen verschiedener Produktvarianten und erhöht die Präzision bei modellspezifischen Fragen erheblich.

4.2.4 Reranking mittels Cross-Encoder

Die initiale Retrieval-Phase priorisiert Effizienz durch die Verwendung vorberechneter Embeddings und approximativer Nearest-Neighbor-Suche. Um die Ergebnisqualität weiter zu verbessern, implementiert das System einen zweistufigen Ranking-Ansatz, bei dem die Top-K-Kandidaten durch ein Cross-Encoder-Modell in einer zweiten Phase neu bewertet werden.

Die Klasse *Reranker* nutzt das Cross-Encoder-Modell ms-marco-MiniLM-L-6-v2, das auf dem MS MARCO Passage Ranking Dataset [2] trainiert wurde. Im Gegensatz zu Bi-Encodern, die Query und Dokument unabhängig vektorisieren, verarbeiten Cross-Encoder das Query-Document-Paar gemeinsam durch das Transformer-Netzwerk. Diese gemeinsame Verarbeitung ermöglicht eine feingranularere Erfassung der semantischen Beziehung zwischen Anfrage und Dokumentinhalt, einschließlich lexikalischer Übereinstimmungen und kontextueller Nuancen.

Der Reranking-Prozess konstruiert zunächst Query-Document-Paare aus der Anfrage und den initialen Retrieval-Ergebnissen. Diese Paare werden dem Cross-Encoder als Input übergeben, der für jedes Paar einen Relevanz-Score berechnet. Anschließend werden die Dokumente nach diesem Score neu sortiert und die Top-K-Ergebnisse an die Antwortgenerierung weitergeleitet.

Die Konfiguration sieht vor, dass initial mehr Kandidaten abgerufen werden als final benötigt, um dem Reranker einen ausreichend diversen Kandidatenpool zu bieten. Bei Standardanfragen wird ein Faktor von drei angewandt, bei Spezifikationsanfragen aufgrund deren höherer Komplexität ein Faktor von acht. Nach dem Reranking werden die besten 15 Dokumente für die Kontextgenerierung verwendet, wobei dieser Wert einen Kompromiss zwischen Informationsgehalt und Context-Window-Limits des Sprachmodells darstellt.

4.3 Generativer Teil

Die generative Komponente des Systems nutzt die durch das Retrieval identifizierten Dokumentfragmente als Kontextgrundlage für die Beantwortung von Benutzeranfragen mittels eines Large Language Models. Die technische Umsetzung basiert auf dem LangChain-Framework und der OpenAI API.

4.3.1 Context-Formatierung und Prompt-Konstruktion

Die Klasse *QAChain* orchestriert den generativen Prozess und verantwortet die Transformation der strukturierten Retrieval-Ergebnisse in einen für das Sprachmodell geeigneten Prompt. Die

Methode *format_ context()* bereitet die abgerufenen Dokumente für die Aufnahme in den Prompt auf.

Jedes Dokumentfragment wird mit einer Quellenkennung versehen, die sowohl eine laufende Nummer als auch den internen Chunk-Identifikator enthält. Diese Kennzeichnung ermöglicht dem Sprachmodell die Quellenreferenzierung in seiner Antwort und bildet die Grundlage für die nachgelagerte Quellenvisualisierung im Frontend. Die formatierten Kontextabschnitte werden durch visuelle Trennzeichen separiert, um dem Modell die Unterscheidung verschiedener Dokumentquellen zu erleichtern.

Für Spezifikationsanfragen implementiert das System eine zusätzliche Kontextpriorisierung. Dokumentfragmente, die charakteristische technische Merkmale aufweisen – etwa Prozessorbezeichnungen, Speicherangaben oder Displayspezifikationen - werden im Kontext vorangestellt. Diese Neuordnung stellt sicher, dass die für die Anfrage mutmaßlich relevantesten Informationen im für das Sprachmodell aufmerksamkeitsstärksten Bereich des Kontexts positioniert sind.

4.3.2 Prompt-Engineering für faktentreue Antworten

Ein zentrales Problem bei der Verwendung von Large Language Models für wissensbasierte Fragestellungen besteht in ihrer Tendenz zur Halluzination - der Generierung plausibel klingender, jedoch faktisch unzutreffender Informationen. Das entwickelte System adressiert dieses Problem durch sorgfältig konstruierte System-Prompts, die das Modell zu strenger Quellenorientierung anleiten.

Der System-Prompt etabliert mehrere Verhaltensregeln: Das Modell wird angewiesen, ausschließlich Informationen zu verwenden, die explizit im bereitgestellten Kontext dokumentiert sind. Für nicht dokumentierte Aspekte soll explizit *Nicht spezifiziert* oder eine äquivalente Formulierung verwendet werden, statt auf allgemeines Weltwissen zurückzugreifen. Die Verwendung von Trainingskorpus-Wissen zu ähnlichen Produkten ist explizit untersagt, um Verwechslungen zwischen Modellvarianten zu vermeiden.

Für modellspezifische Anfragen erweitert das System den Prompt um eine dedizierte Warnung zur Modell-Spezifizität. Diese instruiert das Sprachmodell, ausschließlich Informationen zu verwenden, die dem angefragten Produktmodell eindeutig zugeordnet werden können, und Informationen zu anderen Modellen explizit zu ignorieren - selbst wenn diese im Kontext enthalten sind.

Die Prompt-Struktur priorisiert zudem die Hierarchie technischer Informationen. Kernspezifikationen wie Prozessor, Arbeitsspeicher und Speicherkapazität sollen vorrangig behandelt werden, während periphere Details wie Webcam-Spezifikationen oder LED-Indikatoren nur bei expliziter Nachfrage thematisiert werden sollen.

4.3.3 Integration von Konversationshistorie

Das System unterstützt kontextuelle Konversationen, bei denen aufeinander aufbauende Fragen gestellt werden können, ohne dass der Benutzer den Gesprächsgegenstand wiederholt explizit machen muss. Diese Funktionalität wird durch die Integration der Chat-Historie in den Prompt-Kontext realisiert.

Die Methode *answer()* nimmt optional eine Liste vorheriger Konversationsnachrichten entgegen. Die letzten zehn Austausche werden in das Message-Array aufgenommen, das dem Sprachmodell übermittelt wird. Benutzeranfragen werden als HumanMessage-Objekte, Systemantworten als AIMessage-Objekte repräsentiert. Diese Unterscheidung ermöglicht dem Modell die korrekte Interpretation des Gesprächsverlaufs und die Auflösung anaphorischer Referenzen wie *dieses Modell* oder *der vorherige Prozessor*.

Die Begrenzung auf zehn historische Austausche stellt einen Kompromiss zwischen Kontexterhaltung und Token-Effizienz dar. Umfangreichere Historien würden das verfügbare Context-Window des Sprachmodells belasten und die Kosten pro Anfrage erhöhen, ohne typischerweise proportional zur Antwortqualität beizutragen.

4.3.4 Sprachmodell-Konfiguration

Als generatives Sprachmodell kommt gpt-4o-mini von OpenAI zum Einsatz, das über die LangChain-Integration der ChatOpenAI-Klasse angesprochen wird. Dieses Modell bietet eine günstige Balance zwischen Generierungsqualität und Kosten. Reasoning-intensive Modelle wie GPT-4 / GPT-5 oder deren Omni-Varianten wurden bewusst nicht eingesetzt, da deren erhöhter Token-Konsum für die typischen Spezifikationsabfragen nicht erforderlich ist und die Betriebskosten unverhältnismäßig steigern würde

Die Temperatur-Einstellung von 0,3 reduziert die Stochastizität der Generierung und fördert deterministische, faktisch orientierte Antworten. Die maximale Token-Anzahl von 2000 ermöglicht ausführliche Antworten bei komplexen Spezifikationsabfragen, ohne das Gesamt-Context-Window zu erschöpfen. Die API-Authentifizierung erfolgt über einen in Umgebungsvariablen hinterlegten API-Schlüssel, wobei das Credential niemals in den Quellcode eingebettet wird.

4.4 Systemintegration

Die Systemintegration verbindet die individuellen Komponenten zu einer kohärenten, deploymentfähigen Anwendung und stellt die notwendige Infrastruktur für Betrieb, Sicherheit und Monitoring bereit.

4.4.1 RESTful API-Architektur

Das Backend exponiert seine Funktionalität über eine RESTful API, die dem OpenAPI 3.0-Standard entspricht. FastAPI generiert automatisch eine interaktive API-Dokumentation, die unter dem Pfad /docs zugänglich ist und Entwicklern das Testen der Endpunkte ohne separates Tooling ermöglicht.

Die API ist in thematisch fokussierte Router gegliedert, die als eigenständige FastAPI-Router-Module implementiert sind. Der Documents-Router unter /api/documents ermöglicht den Upload von Dokumenten - die daraufhin automatisch die Ingestion-Pipeline durchlaufen - sowie die Abfrage und Löschung bestehender Dokumente. Der Query-Router unter /api/query nimmt RAG-Anfragen entgegen und gibt strukturierte Antworten mit Quellenverweisen zurück. Der Benchmark-Router unter /api/benchmark ermöglicht die Initiierung von RAGAS-Evaluierungsläufen über die API.

Die CORS-Middleware-Konfiguration erlaubt Cross-Origin-Anfragen von konfigurierten Ursprüngen, primär dem Streamlit-Entwicklungsserver. Diese Konfiguration ist notwendig, da Frontend und Backend in der Entwicklungsumgebung unter unterschiedlichen Ports operieren und Browser standardmäßig Cross-Origin-Anfragen blockieren.

4.4.2 Datenbankintegration

Die ORM-Integration[1] mit SQLAlchemy abstrahiert die Datenbankoperationen hinter einer Python-Schnittstelle. Die vier Kernentitäten - User, Document, Chunk und QueryHistory - werden als Python-Klassen definiert, die von einer gemeinsamen deklarativen Basisklasse erben. SQLAlchemy übernimmt die Übersetzung von Objektoperationen in SQL-Statements sowie das Management von Datenbankverbindungen und -transaktionen.

Die Beziehungen zwischen Entitäten werden über Fremdschlüssel und SQLAlchemy-Relationships modelliert. Die Eins-zu-Viele-Beziehung zwischen User und Document ermöglicht die benutzerspezifische Dokumentenverwaltung, während die Eins-zu-Viele-Beziehung zwischen Document und Chunk die Chunking-Hierarchie abbildet. Kaskadierendes Löschen ist konfiguriert, sodass das Entfernen eines Dokuments automatisch alle zugehörigen Chunks eliminiert.

Die Datenbankinitialisierung erfolgt beim Anwendungsstart automatisch. SQLAlchemy inspiziert die definierten Modelle und generiert das entsprechende Datenbankschema, sofern dieses noch nicht existiert. Für Produktionsumgebungen wäre eine Migration auf robustere Datenbanksysteme wie PostgreSQL mit minimalen Konfigurationsänderungen möglich.

[1] Object-Relational Mapping

4.4.3 Logging und Monitoring

Das Logging-Subsystem kombiniert zwei komplementäre Bibliotheken: Loguru für die eigentliche Log-Generierung und Structlog für die strukturierte Kontextanreicherung. Diese Kombination ermöglicht sowohl entwicklerfreundliche Konsolenausgaben während der Entwicklung als auch maschinenlesbare JSON-Logs für Produktionsumgebungen.

Drei logische Log-Streams werden konfiguriert: Der Console-Handler gibt farbig formatierte Logs mit Zeitstempel, Log-Level und Quellenangabe aus. Der File-Handler schreibt JSON-serialisierte Logs in rotierende Dateien mit 10 MB Größenlimit. Ein spezieller Interaction-Handler erfasst ausschließlich RAG-Anfragen und -Antworten in einer separaten Logdatei zur nachträglichen Qualitätsanalyse und zum Debugging.

Die Integration mit dem Standard-Logging-Modul von Python erfolgt über einen Custom-Handler, der Logs aus Drittbibliotheken abfängt und an Loguru weiterleitet. Diese Vereinheitlichung stellt sicher, dass alle Anwendungslogs - unabhängig von ihrer Ursprungsquelle - konsistent formatiert und an einheitlicher Stelle gesammelt werden.

4.5 Trade-Offs im Chunking: Recall, Kosten und Bias

Im Rahmen der Vorverarbeitung wurden alle Dokumente in semantisch sinnvolle Textsegmente (Chunks) zerlegt und anschließend vollständig in den Vektor-Index eingebracht. Eine im Code vorgesehene Begrenzung der Chunk-Anzahl (max_chunks) diente in frühen Entwicklungsphasen der Reduktion von Rechenzeit, wurde für die finalen Experimente jedoch deaktiviert, um eine vollständige Korpusabdeckung sicherstellen zu können. Diese Entscheidung zielt darauf ab, den inhaltlichen Recall des Retrieval-Moduls zu maximieren und damit die Antwortqualität des RAG-Systems zu verbessern. Zugleich führt das Indexieren aller Chunks zu längeren Laufzeiten, höherem Speicherbedarf und Embedding-Kosten sowie einem entsprechend umfangreichen Vektor-Speicher.

Das Beschneiden auf die ersten N-Chunks kann einen systematischen Reihenfolgebias erzeugen, weil spätere Dokumentpassagen unterrepräsentiert bleiben. Zur Bias-Reduktion wären eine Zufallsreihenfolge der Chunks vor dem Beschneiden oder eine pro-Dokument-Begrenzung geeignete Maßnahmen. Da im finalen Setup auf eine Limitierung verzichtet wurde, entfällt dieses Risiko, allerdings zum Preis höherer Ressourcennutzung. Insgesamt stellt die Deaktivierung der Begrenzung eine Qualität-gegen-Kosten-Abwägung dar, bei der zugunsten der inhaltlichen Abdeckung und Retrieval-Leistung entschieden wurde. Eine direkte Konsequenz war ebenfalls der Einsatz von HuggingFace-Embeddings anstelle der OpenAI-Embeddings.

5 Evaluation

Die Evaluation des entwickelten RAG-Systems zielt darauf ab, die Qualität der Retrieval- und Generierungskomponenten systematisch zu bewerten und verschiedene Konfigurationen zu vergleichen. Dieses Kapitel präsentiert die Evaluationsmethodik, das Testsetup sowie die Ergebnisse der durchgeführten Experimente.

5.1 Evaluationskriterien

Die Bewertung des RAG-Systems erfolgt anhand mehrdimensionaler Metriken, die sowohl die Retrieval-Qualität als auch die Generierungsqualität erfassen. Als Evaluationsframework kommt RAGAS zum Einsatz, das automatisierte Metriken auf Basis von LLM-as-a-Judge-Ansätzen bereitstellt. [6]

5.1.1 Retrieval-Metriken

Die Qualität des Retrieval-Prozesses wird durch folgende Metrik bewertet:

- **Context Recall**: Misst den Anteil der relevanten Informationen aus dem Gold Standard, der tatsächlich in den abgerufenen Dokumentenabschnitten enthalten ist. Ein hoher Context Recall gewährleistet, dass alle notwendigen Informationen für die Antwortgenerierung verfügbar sind.

5.1.2 Generierungs-Metriken

Die Qualität der generierten Antworten wird durch drei zentrale Metriken erfasst:

- **Faithfulness**: Bewertet, inwieweit die generierte Antwort durch die abgerufenen Dokumentenabschnitte gestützt wird. Eine hohe Faithfulness bedeutet, dass die Antwort faktisch korrekt ist und keine Halluzinationen enthält. Die Metrik wird durch ein LLM bewertet, das prüft, ob alle Aussagen in der Antwort durch den bereitgestellten Kontext belegt werden können.
- **Answer Relevancy**: Misst, wie gut die generierte Antwort die gestellte Frage beantwortet, unabhängig von der faktischen Korrektheit. Diese Metrik bewertet die semantische Übereinstimmung zwischen Frage und Antwort.
- **Answer Correctness**: Eine kombinierte Metrik, die sowohl Faithfulness als auch Answer Relevancy berücksichtigt und zusätzlich einen Vergleich mit der Referenzantwort (Ground Truth) durchführt. Diese Metrik erfordert Ground Truth-Daten und ermöglicht eine objektive Bewertung der Antwortqualität.

Alle Metriken werden auf einer Skala von 0 bis 1 gemessen, wobei 1 die beste mögliche Leistung repräsentiert.

5.2 Testsetup

5.2.1 Gold Standard Dataset

Für die systematische Evaluation wurde ein Gold Standard Dataset erstellt, das 90 Fragen zu technischen Spezifikationen verschiedener Laptop-Modelle umfasst. Die Fragen decken verschiedene Kategorien ab:

- Prozessorspezifikationen (z.B. unterstützte Prozessormodelle)
- Arbeitsspeicher-Konfigurationen (Kapazität, Typ, Frequenz)
- Speicherkapazitäten und -typen
- Display-Eigenschaften (Größe, Auflösung, Helligkeit)
- Grafikkarten-Konfigurationen
- Weitere technische Spezifikationen (Gewicht, Betriebssysteme, Schnittstellen)

Jede Frage ist mit einer Referenzantwort (Ground Truth) versehen, die auf manueller Extraktion aus den Produktdatenblättern basiert. Das Dataset umfasst Fragen zu verschiedenen Produktmodellen, darunter Lenovo ThinkPad E14/E16 (verschiedene Generationen), Dell Pro 16 sowie weitere Modelle. Die Fragen sind sowohl auf Deutsch als auch auf Englisch formuliert, um die Mehrsprachigkeitsfähigkeiten des Systems zu testen.

5.2.2 Evaluationskonfiguration

Die Evaluation wurde mit folgenden Systemkonfigurationen durchgeführt:

- **Chunk-Größe**: 1200 Zeichen mit einem Overlap von 200 Zeichen
- **Retrieval-Parameter**: Top-K = 5 Dokumentenabschnitte für die finale Antwortgenerierung
- **Sprachmodell**: GPT-4o-mini mit einer Temperatur von 0,3
- **Maximale Token-Anzahl**: 2000 Tokens pro Antwort

Für die Evaluierung verschiedener Embedding-Modelle wurde jeweils eine vollständige Neuindexierung des Dokumentenkorpus durchgeführt, um faire Vergleichsbedingungen zu gewährleisten.

5.2.3 Evaluationsprozess

Der Evaluationsprozess folgt einem standardisierten Ablauf:

1. Für jede Frage im Gold Standard Dataset wird die RAG-Pipeline ausgeführt.
2. Die abgerufenen Dokumentenabschnitte werden zusammen mit der generierten Antwort an RAGAS übergeben.
3. RAGAS berechnet die Metriken für jede Frage einzeln.
4. Die Ergebnisse werden aggregiert, um Durchschnittswerte und Verteilungen zu erhalten.

Um Rate-Limiting der LLM-APIs zu vermeiden, wurde zwischen den einzelnen Anfragen eine Verzögerung von 3 Sekunden eingefügt. Die Evaluation wurde mit einem einzelnen Worker durchgeführt, um Konsistenz zu gewährleisten.

5.3 Evaluierung verschiedener Embedding-Modelle

Um die Auswirkung der Embedding-Modellwahl auf die Systemleistung zu untersuchen, wurden vier verschiedene Embedding-Modelle evaluiert. Die Auswahl umfasst sowohl monolinguale als auch multilinguale Modelle sowie Modelle unterschiedlicher Größe und Architektur.

5.3.1 all-MiniLM-L6-v2

Das Modell `sentence-transformers/all-MiniLM-L6-v2` ist ein kompaktes Modell mit 384-dimensionalen Embeddings, das auf der DistilBERT-Architektur basiert. [30] Das Modell bietet eine gute Balance zwischen Qualität und Performance und unterstützt grundlegende multilinguale Fähigkeiten.

Die Evaluationsergebnisse zeigen folgende Durchschnittswerte über alle 90 Fragen:

- **Faithfulness**: 0,749 (74,9%)
- **Answer Relevancy**: 0,752 (75,2%)
- **Context Recall**: 0,876 (87,6%)
- **Answer Correctness**: 0,790 (79,0%)

Das Modell erzielt insbesondere einen hohen Context Recall, was darauf hindeutet, dass die meisten relevanten Informationen erfolgreich abgerufen werden. Die Faithfulness-Werte zeigen, dass die generierten Antworten größtenteils durch die abgerufenen Dokumente gestützt werden, jedoch Raum für Verbesserungen besteht.

Tabelle 5.1: Vergleich der einzelnen Fragen für das all-MiniLM-L6-v2

Question	Faithfulness	Answer Relevancy	Context Recall	Answer Correctness
Q-001	1.0	0.94	1.0	0.95
Q-002	0.0	0.0	0.0	0.02
Q-003	0.0	0.0	1.0	0.01
Q-004	0.0	0.0	0.0	0.02
Q-005	1.0	0.98	0.0	0.04
Q-006	1.0	1.0	1.0	1.0
Q-007	0.0	0.0	0.0	0.03
Q-008	1.0	1.0	1.0	0.92
Q-009	0.5	0.43	1.0	0.15
Q-010	1.0	1.0	1.0	1.0
:	:	:	:	:
Q-090	1.0	1.0	1.0	1.0

5.3.2 all-mpnet-base-v2

Das Modell `sentence-transformers/all-mpnet-base-v2` verwendet die MPNet-Architektur, die sowohl Masked Language Modeling als auch Permutation Language Modeling kombiniert. [24] Mit 768-dimensionalen Embeddings ist es deutlich größer als das Baseline-Modell und sollte theoretisch bessere semantische Repräsentationen liefern.

Die Evaluationsergebnisse:

- **Faithfulness**: 0,766 (76,6%)
- **Answer Relevancy**: 0,788 (78,8%)
- **Context Recall**: 0,894 (89,4%)
- **Answer Correctness**: 0,786 (78,6%)

Im Vergleich zum Baseline-Modell zeigt `all-mpnet-base-v2` eine Verbesserung bei Faithfulness (+1,7 Prozentpunkte) und Answer Relevancy (+3,6 Prozentpunkte). Der Context Recall steigt auf 89,4%, was darauf hindeutet, dass das größere Modell relevante Informationen besser identifiziert. Interessanterweise liegt die Answer Correctness leicht unter der des Baseline-Modells, was auf eine unterschiedliche Gewichtung der Metriken hindeuten könnte.

Tabelle 5.2: Vergleich der einzelnen Fragen für das all-mpnet-base-v2

Question	Faithfulness	Answer Relevancy	Context Recall	Answer Correctness
Q-001	1.0	0.99	1.0	0.77
Q-002	0.0	0.0	0.0	0.02
Q-003	0.0	0.0	1.0	0.01
Q-004	0.0	0.0	1.0	0.02
Q-005	0.0	0.0	0.0	0.03
Q-006	1.0	1.0	1.0	1.0
Q-007	0.0	0.0	0.0	0.03
Q-008	1.0	1.0	1.0	1.0
Q-009	0.5	0.43	1.0	0.15
Q-010	1.0	1.0	1.0	1.0
:	:	:	:	:
Q-090	1.0	1.0	1.0	1.0

5.3.3 paraphrase-multilingual-MiniLM-L12-v2

Das Modell `paraphrase-multilingual-MiniLM-L12-v2` wurde speziell für multilinguale Anwendungen trainiert und unterstützt über 50 Sprachen. [23] Mit 384-dimensionalen Embeddings ist es ähnlich groß wie das Baseline-Modell, jedoch auf sprachübergreifende semantische Ähnlichkeit optimiert.

Die Evaluationsergebnisse:

- **Faithfulness**: 0,787 (78,7%)
- **Answer Relevancy**: 0,792 (79,2%)
- **Context Recall**: 0,906 (90,6%)
- **Answer Correctness**: 0,817 (81,7%)

Dieses Modell erzielt die besten Ergebnisse aller evaluierten Modelle. Die Faithfulness liegt bei 78,7%, was eine deutliche Verbesserung gegenüber dem Baseline-Modell darstellt (+3,8 Prozentpunkte). Der Context Recall von 90,6% ist der höchste aller Modelle und zeigt, dass das multilinguale Training die Fähigkeit zur Identifikation relevanter Informationen verbessert. Die Answer Correctness von 81,7% ist ebenfalls die höchste, was darauf hindeutet, dass dieses Modell aus der Auswahl an Embedding-Modellen am besten für den vorliegenden Anwendungsfall geeignet ist.

Tabelle 5.3: Vergleich der einzelnen Fragen für das paraphrase-multilingual-MiniLM-L12-v2

Question	Faithfulness	Answer Relevancy	Context Recall	Answer Correctness
Q-001	1.0	1.0	1.0	1.0
Q-002	0.0	0.0	0.0	0.02
Q-003	1.0	1.0	1.0	1.0
Q-004	0.0	0.0	0.0	0.01
Q-005	1.0	0.91	0.0	0.04
Q-006	1.0	1.0	1.0	1.0
Q-007	0.0	0.0	0.0	0.03
Q-008	1.0	1.0	1.0	1.0
Q-009	0.5	0.43	1.0	0.15
Q-010	1.0	1.0	1.0	1.0
:	:	:	:	:
Q-090	1.0	1.0	1.0	1.0

5.3.4 intfloat/multilingual-e5-large

Das Modell `intfloat/multilingual-e5-large` ist das größte evaluierte Modell und basiert auf der E5-Architektur, die für Information Retrieval optimiert wurde. [28] Mit 1024-dimensionalen Embeddings bietet es die höchste Dimensionalität, was jedoch auch höhere Speicher- und Rechenanforderungen mit sich bringt.

Die Evaluationsergebnisse:

- **Faithfulness**: 0,706 (70,6%)
- **Answer Relevancy**: 0,739 (73,9%)
- **Context Recall**: 0,806 (80,6%)
- **Answer Correctness**: 0,754 (75,4%)

Trotz der größeren Modellgröße erzielt `multilingual-e5-large` die niedrigsten Werte aller evaluierten Modelle. Die Faithfulness liegt bei nur 70,6%, was deutlich unter den anderen Modellen liegt. Auch der Context Recall von 80,6% ist der niedrigste, was darauf hindeutet, dass das Modell für diesen spezifischen Anwendungsfall weniger geeignet ist. Mögliche Gründe könnten in der unterschiedlichen Trainingsmethodik oder in einer schlechteren Anpassung an technische Dokumente liegen.

Tabelle 5.4: Vergleich der einzelnen Fragen für das multilingual-e5-large

Question	Faithfulness	Answer Relevancy	Context Recall	Answer Correctness
Q-001	1.0	1.0	1.0	1.0
Q-002	0.5	0.63	0.0	0.15
Q-003	0.0	0.0	1.0	0.01
Q-004	0.0	0.0	0.0	0.02
Q-005	1.0	1.0	0.0	0.04
Q-006	1.0	1.0	1.0	1.0
Q-007	0.0	0.0	0.0	0.03
Q-008	1.0	1.0	1.0	0.93
Q-009	0.0	0.40	0.0	0.17
Q-010	1.0	1.0	1.0	1.0
:	:	:	:	:
Q-090	1.0	1.0	1.0	1.0

5.4 Vergleichende Analyse der Ergebnisse

5.4.1 Überblick über die Modellleistung

Tabelle 5.5 zeigt eine zusammenfassende Übersicht der Evaluationsergebnisse aller vier Modelle.

Tabelle 5.5: Vergleich der Embedding-Modelle

Modell	Faithfulness	Answer Relevancy	Context Recall	Answer Correctness
all-MiniLM-L6-v2	0,749	0,752	0,876	0,790
all-mpnet-base-v2	0,766	0,788	0,894	0,786
paraphrase-multilingual-MiniLM-L12-v2	**0,787**	**0,792**	**0,906**	**0,817**
multilingual-e5-large	0,706	0,739	0,806	0,754

Die Ergebnisse zeigen, dass `paraphrase-multilingual-MiniLM-L12-v2` in allen Metriken die beste Leistung erzielt. Dies ist bemerkenswert, da es nicht das größte Modell ist, sondern durch sein spezialisiertes Training auf multilinguale Paraphrase-Erkennung optimiert wurde.

5.4.2 Interpretation der Ergebnisse

Die Ergebnisse des `paraphrase-multilingual-MiniLM-L12-v2`-Modells lassen sich durch mehrere Faktoren erklären:

- **Multilinguales Training**: Das Modell wurde explizit darauf trainiert, semantische Ähnlichkeiten zwischen Texten verschiedener Sprachen zu erfassen. Dies ist für den vorliegenden Anwendungsfall von Vorteil, da sowohl deutsch- als auch englischsprachige Dokumente verarbeitet werden müssen.

- **Paraphrase-Optimierung**: Die Fokussierung auf Paraphrase-Erkennung hilft dem Modell, verschiedene Formulierungen derselben technischen Konzepte zu identifizieren. Dies ist besonders wichtig bei technischen Dokumenten, wo dieselben Spezifikationen in unterschiedlicher Terminologie formuliert sein können.
- **Balance zwischen Größe und Qualität**: Mit 384 Dimensionen bietet das Modell eine gute Balance zwischen Repräsentationsfähigkeit und Effizienz.

Die schlechte Leistung von `multilingual-e5-large` ist überraschend, da größere Modelle typischerweise bessere Ergebnisse erzielen. Mögliche Erklärungen sind:

- Das Modell wurde möglicherweise auf anderen Domänen trainiert und ist weniger gut an technische Dokumente angepasst.
- Die höhere Dimensionalität könnte zu Overfitting führen oder die Ähnlichkeitssuche in hochdimensionalen Räumen erschweren.
- Unterschiedliche Normalisierungs- oder Trainingsverfahren könnten die Performance beeinflussen.

5.4.3 Trade-offs zwischen Modellen

Die Evaluationsergebnisse zeigen verschiedene Trade-offs zwischen den Modellen:

- **Performance vs. Effizienz**: Während `paraphrase-multilingual-MiniLM-L12-v2` die beste Qualität bietet, ist `all-MiniLM-L6-v2` das effizienteste Modell. Für Anwendungen mit strikten Latenzanforderungen könnte das Baseline-Modell trotz geringerer Qualität bevorzugt werden.
- **Speicherbedarf**: Die größeren Modelle (`all-mpnet-base-v2` und `multilingual-e5-large`) erfordern mehr Speicherplatz für die Vektorspeicherung. Bei sehr großen Dokumentenkorpora kann dies ein entscheidender Faktor sein.
- **Multilingualität**: Nur `paraphrase-multilingual-MiniLM-L12-v2` und `multilingual-e5-large` sind explizit für multilinguale Anwendungen optimiert. Für Systeme, die ausschließlich englischsprachige Dokumente verarbeiten, könnte `all-mpnet-base-v2` eine Alternative darstellen.

5.5 Fehleranalyse

5.5.1 Häufige Fehlertypen

Eine qualitative Analyse der Evaluationsergebnisse zeigt mehrere wiederkehrende Fehlertypen:

- **Fehlende Informationen**: In einigen Fällen antwortet das System mit "Nicht spezifiziert", obwohl die Informationen tatsächlich in den Dokumenten vorhanden sind. Dies deutet auf Retrieval-Probleme hin, bei denen relevante Dokumentenabschnitte nicht identifiziert werden.
- **Unvollständige Antworten**: Bei Fragen nach Listen von Spezifikationen (z.B. alle unterstützten Prozessormodelle) werden manchmal nicht alle Optionen genannt. Dies kann auf Chunk-Grenzen zurückzuführen sein, die zusammengehörige Informationen trennen.
- **Modellverwechslungen**: In seltenen Fällen werden Spezifikationen eines anderen Modells genannt, insbesondere bei ähnlich benannten Produkten (z.B. E14 Gen 6 vs. E14 Gen 7). Dies deutet auf Verbesserungspotenzial bei der Produktfilterung hin.

5.5.2 Beispielanalysen

Ein konkretes Beispiel illustriert die Herausforderungen: Bei der Frage nach der maximalen RAM-Kapazität des Dell Pro 16 PC16250 antwortet das System häufig mit "Nicht spezifiziert", obwohl die Information (64 GB) in den Dokumenten vorhanden ist. Die Analyse zeigt, dass die relevanten Dokumentenabschnitte zwar abgerufen werden, aber möglicherweise nicht mit ausreichend hoher Relevanz bewertet werden, um in den finalen Kontext aufgenommen zu werden.

Ein weiteres Beispiel zeigt das Problem der Modellverwechslung: Bei Fragen zum "Lenovo Thinkpad E14 Gen 6 (AMD)" werden gelegentlich Spezifikationen des Intel-Modells genannt. Dies deutet darauf hin, dass die Produktfilterung im Retriever verbessert werden könnte, um präziser zwischen verschiedenen Modellvarianten zu unterscheiden.

5.6 Diskussion

5.6.1 Zusammenfassung der Ergebnisse

Die Evaluation zeigt, dass das entwickelte RAG-System grundsätzlich in der Lage ist, technische Fragen zu Produktdatenblättern zu beantworten. Die besten Ergebnisse werden mit dem `paraphrase-multilingual-MiniLM-L12-v2`-Modell erzielt, das eine Answer Correctness von 81,7% erreicht. Dies bedeutet, dass etwa vier von fünf Fragen korrekt beantwortet werden, was für einen produktiven Einsatz akzeptabel ist.

Die hohen Context Recall-Werte (über 87% bei allen Modellen außer `multilingual-e5-large`) zeigen, dass das Retrieval-System effektiv relevante Informationen identifiziert. Die Faithfulness-Werte zwischen 70% und 79% deuten darauf hin, dass die generierten Antworten größtenteils durch die Dokumente gestützt werden, jedoch weiteres Verbesserungspotenzial besteht.

5.6.2 Limitationen der Evaluation

Die durchgeführte Evaluation weist mehrere Limitationen auf:

- **Datensatzgröße**: Mit 90 Fragen ist das Gold Standard Dataset relativ klein. Eine größere Anzahl von Fragen würde robustere statistische Aussagen ermöglichen, musste jedoch aufgrund von Token-Limits reduziert werden.
- **Domänenspezifität**: Der Datensatz fokussiert ausschließlich auf Laptop-Spezifikationen. Die Generalisierbarkeit der Ergebnisse auf andere technische Domänen ist ungewiss.
- **LLM-as-a-Judge**: Die Verwendung eines LLMs zur Bewertung der Metriken kann Inkonsistenzen und Bias einführen. Eine manuelle Evaluation einer Stichprobe würde zusätzliche Validierung ermöglichen.
- **Fehlende Metriken**: Context Precision wurde nicht berechnet, da dies die Evaluationszeit erheblich verlängert hätte. Diese Metrik würde zusätzliche Einblicke in die Retrieval-Qualität bieten.

5.6.3 Implikationen für die Praxis

Die Evaluationsergebnisse haben mehrere Implikationen für den produktiven Einsatz des Systems:

- **Modellauswahl**: `paraphrase-multilingual-MiniLM-L12-v2` sollte auf Basis der Auswahl an Modellen als Standard-Embedding-Modell verwendet werden, da es die beste Balance zwischen Qualität und Effizienz für den vorliegenden Use Case bietet.
- **Verbesserungspotenzial**: Die Fehleranalyse zeigt, dass insbesondere die Produktfilterung und die Behandlung von Chunk-Grenzen verbessert werden könnten. Zukünftige Arbeiten könnten spezialisierte Chunking-Strategien für technische Dokumente untersuchen.
- **Benutzerkommunikation**: Da nicht alle Fragen korrekt beantwortet werden können, sollte das System Benutzer explizit darauf hinweisen, wenn Informationen nicht verfügbar sind, anstatt falsche Antworten zu generieren. Die bereits implementierte Strategie, *Nicht spezifiziert* zu antworten, ist hier ein Schritt in die richtige Richtung.

Insgesamt zeigt die Evaluation, dass das entwickelte RAG-System eine solide Grundlage für die Beantwortung technischer Fragen bietet, jedoch weiteres Verbesserungspotenzial besteht.

6 Fazit & Ausblick

6.1 Fazit

Retrieval-Augmented Generation gilt als robuste Methode zur Informationsverarbeitung in wissensintensiven Anwendungsbereichen. Insbesondere die Fähigkeit von RAG, faktenbasierte Antworten zu generieren und dabei das Risiko von Halluzinationen im Vergleich zu rein generativen Ansätzen zu reduzieren, unterstreicht seine Relevanz für den Einsatz im wissenschaftlichen und praktischen Kontext. Die vorliegende Arbeit demonstriert die erfolgreiche Anwendung eines RAG-Systems zur automatisierten Beantwortung technischer Fragen aus Produktdatenblättern im Technical Consulting Center (TCC) von Bechtle.

Die systematische Evaluation des entwickelten Systems mit einem Gold Standard Dataset von 90 Fragen zu Laptop-Spezifikationen zeigt, dass RAG-Architekturen bei entsprechender Modellierung ein vielversprechendes Fundament für den Aufbau zuverlässiger und kontextsensitiver NLP-Systeme bilden können. Die Evaluierung von vier verschiedenen Embedding-Modellen ergab, dass das System mit dem optimalen Modell (`paraphrase-multilingual-MiniLM-L12-v2`) eine Answer Correctness von 81,7% erreicht, was bedeutet, dass etwa vier von fünf Fragen korrekt beantwortet werden. Die hohen Context Recall-Werte von über 90% zeigen, dass das Retrieval-System effektiv relevante Informationen identifiziert, während Faithfulness-Werte von 78,7% darauf hindeuten, dass die generierten Antworten größtenteils durch die Dokumente gestützt werden.

Gleichzeitig zeigen die Experimente, dass kein universell optimales RAG-System identifiziert werden konnte. Die Effektivität hängt maßgeblich von Faktoren wie der Dokumentenstruktur, dem zugrunde liegenden Retrieval-Verfahren sowie den eingesetzten Modellen ab. Vor allem die Struktur und Qualität der zugespielten Daten beeinflussen die Leistung erheblich, was eine adaptive Systemkonfiguration erforderlich macht. Dies folgt durch etwa die gezielte Auswahl von Chunking-Verfahren, Größen und Retrieval-Strategie oder den Einsatz leistungsstärkerer Sprachmodelle, um auf unterschiedliche Anforderungen und Datenbedingungen reagieren zu können.

Die Evaluation lieferte mehrere wichtige Erkenntnisse, die für die praktische Anwendung von RAG-Systemen von Bedeutung sind. Erstens zeigt sich, dass größere Modelle nicht automatisch zu besseren Ergebnissen führen. Das `paraphrase-multilingual-MiniLM-L12-v2`-Modell erzielte trotz ähnlicher Größe wie das Baseline-Modell deutlich bessere Ergebnisse, was auf die Bedeutung spezialisierter Trainingsmethoden hinweist. Zweitens erwies sich das multilinguale Training als entscheidend für den vorliegenden Anwendungsfall, da sowohl deutsch- als auch englischsprachige Dokumente verarbeitet werden müssen.

Die Fehleranalyse identifizierte drei Hauptprobleme, die charakteristisch für RAG-Systeme sind: fehlende Informationen trotz vorhandener Dokumente, unvollständige Antworten aufgrund von

Chunk-Grenzen und Modellverwechslungen bei ähnlich benannten Produkten. Diese Erkenntnisse unterstreichen die Notwendigkeit einer sorgfältigen Systemkonfiguration und zeigen gleichzeitig, dass die Evaluation der unterschiedlichen Ansätze noch weiteres Optimierungspotenzial erkennen lässt.

Das entwickelte System adressiert ein konkretes Problem im Arbeitsalltag des Technical Consulting Centers von Bechtle und demonstriert erfolgreich, wie moderne KI-Technologien praktische Probleme im Unternehmenskontext lösen können. Durch die Automatisierung der Informationsbeschaffung aus Produktdatenblättern kann das System die Bearbeitungszeit von Ausschreibungen reduzieren, die Konsistenz der Informationsbeschaffung verbessern und Fehler durch manuelle Extraktion minimieren. Die Evaluation zeigt, dass das System bereits in der Lage ist, einen Großteil der technischen Fragen korrekt zu beantworten, was einen wertvollen Beitrag zur Digitalisierung und Wissensautomatisierung im technischen Beratungsfeld darstellt.

6.2 Ausblick

Für zukünftige Arbeiten ergeben sich aus den Evaluationsergebnissen und den identifizierten Limitationen mehrere konkrete Optimierungspotenziale, die das System weiter verbessern und seinen Anwendungsbereich erweitern können.

Erstens sollte der Dense Retriever weiter feinjustiert werden, um eine noch präzisere Kontextselektion zu ermöglichen. Die Evaluationsergebnisse weisen darauf hin, dass trotz der bereits guten Context Recall-Werte von über 90% weiteres Verbesserungspotenzial bei der Retrieval-Präzision besteht. Insbesondere die Faithfulness-Werte von 78,7% deuten darauf hin, dass die Antwortgenerierung weiter optimiert werden kann, was durch eine präzisere Kontextselektion unterstützt werden könnte. Mögliche Ansätze umfassen die Integration von Hybrid Retrieval-Strategien, die Dense Retrieval mit Sparse Retrieval-Methoden kombinieren. Zusätzlich könnte die Entwicklung spezialisierter Chunking-Strategien, die besser an technische Dokumente angepasst sind, die Probleme mit Chunk-Grenzen adressieren und die Vollständigkeit der Antworten verbessern.

Zweitens bietet sich eine erweiterte Integration der RAGAS-Metriken an, um die Bewertung der Antwortqualität noch differenzierter und reproduzierbarer zu gestalten. Hier zeigte sich das Framework als vertrauenswürdiges Evaluations-Tool, das wertvolle Einblicke in die Systemleistung liefert. Die Implementierung zusätzlicher Metriken wie Context Precision würde ein vollständigeres Bild der Retrieval-Qualität ermöglichen und könnte in einem kontinuierlichen Monitoring-System für den produktiven Einsatz integriert werden. Ein solches System könnte die Systemleistung im produktiven Einsatz überwachen und frühzeitig Probleme identifizieren, während gleichzeitig Feedback-Mechanismen wertvolle Daten für die kontinuierliche Verbesserung liefern könnten.

Als dritter und ausschlaggebendster Punkt wird die Implementierung eines Agentic RAG-Ansatzes empfohlen, da dieser durch die Einbindung agentenbasierter Entscheidungslogik eine höhere

Flexibilität und Steuerbarkeit bei der Kontextauswahl und Antwortgenerierung verspricht. Ein solcher Ansatz könnte die identifizierten Probleme wie fehlende Informationen trotz vorhandener Dokumente oder Modellverwechselungen bei ähnlich benannten Produkten durch intelligente Entscheidungslogik adressieren. Die agentenbasierte Architektur würde es dem System ermöglichen, dynamisch auf unterschiedliche Anforderungen zu reagieren und die Retrieval-Strategie an die spezifischen Charakteristika der gestellten Frage anzupassen.

Für den produktiven Einsatz in größerem Maßstab sind verschiedene Aspekte der Skalierung zu berücksichtigen. Das System sollte in der Lage sein, mit größeren Dokumentenkorpora umzugehen und gleichzeitig eine akzeptable Antwortzeit zu gewährleisten. Dies erfordert Optimierungen sowohl auf der Ebene der Datenverarbeitung als auch der Infrastruktur, einschließlich der Implementierung von Batch-Processing für Embeddings und der Optimierung der Vektorsuche für größere Indizes.

Die entwickelte Lösung wirft auch mehrere interessante Forschungsfragen auf, die in zukünftigen Arbeiten untersucht werden könnten. Eine wichtige Frage ist die Generalisierbarkeit des Systems auf andere Domänen und Produktkategorien. Die Evaluation wurde primär mit Laptop-Spezifikationen durchgeführt, und es bleibt zu untersuchen, wie gut das System auf andere Bereiche wie Vorgangsdokumentationen aus beispielsweise dem HR-Bereich übertragbar ist. Weiterhin wäre ein Vergleich mit alternativen Ansätzen wie strukturierter Datenextraktion oder regelbasierten Systemen von Interesse, um die Vor- und Nachteile verschiedener Methoden besser zu verstehen und zu klären, wann RAG-Systeme gegenüber anderen Ansätzen vorteilhaft sind.

Die identifizierten Verbesserungsmöglichkeiten bieten klare Richtungen für zukünftige Entwicklungen, die das System weiter optimieren und erweitern können. Durch die kontinuierliche Weiterentwicklung und Anpassung an die spezifischen Anforderungen des Einsatzbereichs kann das System zu einem unverzichtbaren Tool für die technische Beratung werden und einen nachhaltigen Beitrag zur Digitalisierung und Wissensautomatisierung im Technical Consulting Center von Bechtle leisten.

Hinweis zur Nutzung von Künstlicher Intelligenz

Zur sprachlichen Unterstützung und zur Verbesserung der Lesbarkeit wurden Claude und GPT-5 als KI-gestützte Tools verwendet. Alle generierten Inhalte wurden kritisch überprüft, überarbeitet und durch eigene wissenschaftliche Reflexion ergänzt.

A | Anhang

A.1 Vollständige Evaluationsergebnisse

Dieser Anhang enthält die vollständigen Evaluationsergebnisse für alle 90 Fragen des Gold Standard Datasets für jedes evaluierte Embedding-Modell.

Legende der Spalten:

- **Q**: Question (Frage)
- **F**: Faithfulness
- **AR**: Answer Relevancy
- **CR**: Context Recall
- **AC**: Answer Correctness

A.1.1 all-MiniLM-L6-v2

Tabelle A.1: Vollständige Evaluationsergebnisse für all-MiniLM-L6-v2

Q	F	AR	CR	AC
Q-001	1.00	0.94	1.00	0.95
Q-002	0.00	0.00	0.00	0.02
Q-003	0.00	0.00	1.00	0.01
Q-004	0.00	0.00	0.00	0.02
Q-005	1.00	0.98	0.00	0.04
Q-006	1.00	1.00	1.00	1.00
Q-007	0.00	0.00	0.00	0.03
Q-008	1.00	1.00	1.00	0.92
Q-009	0.50	0.43	1.00	0.15
Q-010	1.00	1.00	1.00	1.00
Q-011	1.00	1.00	1.00	1.00
Q-012	1.00	1.00	1.00	1.00
Q-013	1.00	1.00	1.00	1.00

Fortsetzung auf nächster Seite

Tabelle A.1 – *Fortsetzung von vorheriger Seite*

Q	**F**	**AR**	**CR**	**AC**
Q-014	1.00	1.00	1.00	1.00
Q-015	0.67	0.31	1.00	0.94
Q-016	0.00	0.39	1.00	0.19
Q-017	1.00	0.98	0.00	0.70
Q-018	0.00	0.28	0.00	0.17
Q-019	0.00	0.40	0.67	0.84
Q-020	0.00	0.36	1.00	0.57
Q-021	1.00	0.98	1.00	0.53
Q-022	0.50	0.30	1.00	0.74
Q-023	0.50	0.43	1.00	0.99
Q-024	1.00	0.96	1.00	0.73
Q-025	0.00	0.00	1.00	0.24
Q-026	1.00	1.00	1.00	1.00
Q-027	1.00	0.96	1.00	0.71
Q-028	1.00	1.00	1.00	1.00
Q-029	0.50	0.41	1.00	0.68
Q-030	1.00	1.00	1.00	1.00
Q-031	0.50	0.03	1.00	0.74
Q-032	1.00	0.94	1.00	0.68
Q-033	1.00	0.92	1.00	0.65
Q-034	1.00	1.00	1.00	1.00
Q-035	1.00	1.00	1.00	1.00
Q-036	1.00	1.00	1.00	1.00
Q-037	1.00	1.00	1.00	1.00
Q-038	1.00	1.00	1.00	1.00
Q-039	1.00	1.00	1.00	1.00
Q-040	1.00	0.93	0.00	0.71
Q-041	0.50	0.26	0.50	0.78
Q-042	1.00	1.00	1.00	1.00
Q-043	0.00	0.40	0.67	0.93
Q-044	0.33	0.23	1.00	0.79
Q-045	0.00	0.41	0.00	0.91

Fortsetzung auf nächster Seite

Tabelle A.1 – *Fortsetzung von vorheriger Seite*

Q	**F**	**AR**	**CR**	**AC**
Q-046	0.50	0.30	1.00	0.73
Q-047	0.00	0.09	1.00	0.24
Q-048	0.40	0.46	1.00	0.92
Q-049	1.00	0.93	1.00	0.84
Q-050	1.00	1.00	1.00	1.00
Q-051	1.00	0.91	1.00	0.66
Q-052	1.00	1.00	1.00	1.00
Q-053	1.00	0.96	0.00	0.12
Q-054	1.00	1.00	1.00	1.00
Q-055	0.00	0.45	1.00	0.91
Q-056	1.00	1.00	1.00	1.00
Q-057	1.00	0.91	1.00	0.66
Q-058	1.00	1.00	1.00	1.00
Q-059	1.00	1.00	1.00	1.00
Q-060	1.00	0.91	1.00	1.00
Q-061	0.00	0.39	1.00	0.99
Q-062	1.00	0.95	1.00	0.99
Q-063	1.00	1.00	1.00	1.00
Q-064	1.00	1.00	1.00	1.00
Q-065	1.00	1.00	1.00	1.00
Q-066	0.00	0.05	1.00	0.24
Q-067	1.00	1.00	1.00	1.00
Q-068	1.00	0.93	1.00	0.99
Q-069	1.00	1.00	1.00	1.00
Q-070	1.00	1.00	1.00	1.00
Q-071	1.00	1.00	1.00	1.00
Q-072	1.00	1.00	1.00	1.00
Q-073	1.00	1.00	1.00	1.00
Q-074	1.00	0.99	1.00	0.99
Q-075	1.00	1.00	1.00	1.00
Q-076	1.00	1.00	1.00	1.00
Q-077	1.00	1.00	1.00	1.00

Fortsetzung auf nächster Seite

Tabelle A.1 – *Fortsetzung von vorheriger Seite*

Q	F	AR	CR	AC
Q-078	0.00	0.04	0.00	0.24
Q-079	0.50	0.40	1.00	0.94
Q-080	1.00	1.00	1.00	1.00
Q-081	1.00	0.96	1.00	0.66
Q-082	1.00	1.00	1.00	1.00
Q-083	0.00	0.00	1.00	0.03
Q-084	1.00	1.00	1.00	1.00
Q-085	1.00	1.00	1.00	1.00
Q-086	1.00	1.00	1.00	1.00
Q-087	1.00	0.90	1.00	0.92
Q-088	1.00	0.93	1.00	0.73
Q-089	1.00	1.00	1.00	1.00
Q-090	1.00	1.00	1.00	1.00

A.1.2 all-mpnet-base-v2

Tabelle A.2: Vollständige Evaluationsergebnisse für all-mpnet-base-v2

Q	F	AR	CR	AC
Q-001	1.00	0.99	1.00	0.77
Q-002	0.00	0.00	0.00	0.02
Q-003	0.00	0.00	1.00	0.01
Q-004	0.00	0.00	1.00	0.02
Q-005	0.00	0.00	0.00	0.03
Q-006	1.00	1.00	1.00	1.00
Q-007	0.00	0.00	0.00	0.03
Q-008	1.00	1.00	1.00	1.00
Q-009	0.50	0.43	1.00	0.15
Q-010	1.00	1.00	1.00	1.00
Q-011	1.00	1.00	1.00	1.00

Fortsetzung auf nächster Seite

Tabelle A.2 – *Fortsetzung von vorheriger Seite*

Q	**F**	**AR**	**CR**	**AC**
Q-012	1.00	1.00	1.00	1.00
Q-013	0.00	0.32	0.00	0.20
Q-014	1.00	1.00	1.00	1.00
Q-015	1.00	0.92	1.00	0.94
Q-016	1.00	1.00	1.00	0.62
Q-017	1.00	1.00	1.00	1.00
Q-018	0.00	0.23	0.00	0.16
Q-019	0.00	0.37	1.00	0.73
Q-020	1.00	1.00	1.00	1.00
Q-021	0.00	0.00	1.00	0.02
Q-022	1.00	1.00	1.00	1.00
Q-023	1.00	1.00	1.00	1.00
Q-024	0.00	0.49	1.00	0.60
Q-025	1.00	0.99	1.00	0.24
Q-026	0.17	0.49	1.00	0.98
Q-027	1.00	0.94	1.00	0.71
Q-028	1.00	1.00	1.00	1.00
Q-029	0.00	0.00	1.00	0.02
Q-030	1.00	1.00	1.00	1.00
Q-031	0.50	0.03	1.00	0.61
Q-032	0.50	0.30	1.00	0.69
Q-033	1.00	0.98	0.00	0.67
Q-034	1.00	1.00	1.00	1.00
Q-035	1.00	1.00	1.00	1.00
Q-036	1.00	1.00	1.00	1.00
Q-037	1.00	1.00	1.00	1.00
Q-038	1.00	1.00	1.00	1.00
Q-039	0.00	0.37	0.00	0.21
Q-040	1.00	0.91	0.00	0.23
Q-041	1.00	0.96	0.50	0.55
Q-042	1.00	1.00	1.00	1.00
Q-043	0.00	0.43	1.00	0.89

Fortsetzung auf nächster Seite

Tabelle A.2 – *Fortsetzung von vorheriger Seite*

Q	**F**	**AR**	**CR**	**AC**
Q-044	1.00	1.00	1.00	1.00
Q-045	0.00	0.41	1.00	0.91
Q-046	1.00	1.00	1.00	1.00
Q-047	1.00	0.97	1.00	0.24
Q-048	1.00	1.00	1.00	1.00
Q-049	1.00	1.00	1.00	1.00
Q-050	1.00	1.00	1.00	1.00
Q-051	1.00	0.98	1.00	0.66
Q-052	1.00	0.92	1.00	0.99
Q-053	1.00	0.98	1.00	0.23
Q-054	1.00	1.00	1.00	1.00
Q-055	0.00	0.45	1.00	0.85
Q-056	1.00	1.00	1.00	1.00
Q-057	1.00	0.93	1.00	0.66
Q-058	1.00	1.00	1.00	1.00
Q-059	1.00	1.00	1.00	1.00
Q-060	1.00	1.00	1.00	1.00
Q-061	0.25	0.39	1.00	0.99
Q-062	1.00	0.94	1.00	0.99
Q-063	1.00	1.00	1.00	1.00
Q-064	1.00	0.97	1.00	0.99
Q-065	1.00	1.00	1.00	1.00
Q-066	1.00	1.00	1.00	1.00
Q-067	1.00	0.91	1.00	0.99
Q-068	1.00	0.99	1.00	0.99
Q-069	1.00	1.00	1.00	1.00
Q-070	1.00	1.00	1.00	1.00
Q-071	1.00	1.00	1.00	1.00
Q-072	1.00	1.00	1.00	1.00
Q-073	1.00	1.00	1.00	1.00
Q-074	1.00	0.91	1.00	0.99
Q-075	1.00	1.00	1.00	1.00

Fortsetzung auf nächster Seite

Tabelle A.2 – *Fortsetzung von vorheriger Seite*

Q	F	AR	CR	AC
Q-076	0.50	0.27	1.00	0.69
Q-077	1.00	1.00	1.00	1.00
Q-078	0.00	0.05	0.00	0.24
Q-079	0.50	0.40	1.00	0.94
Q-080	1.00	1.00	1.00	1.00
Q-081	1.00	0.92	1.00	0.91
Q-082	1.00	1.00	1.00	1.00
Q-083	1.00	1.00	1.00	1.00
Q-084	1.00	1.00	1.00	1.00
Q-085	1.00	1.00	1.00	1.00
Q-086	1.00	1.00	1.00	1.00
Q-087	0.00	0.41	1.00	0.67
Q-088	1.00	0.94	1.00	0.73
Q-089	1.00	1.00	1.00	1.00
Q-090	1.00	1.00	1.00	1.00

A.1.3 paraphrase-multilingual-MiniLM-L12-v2

Tabelle A.3: Vollständige Evaluationsergebnisse für paraphrase-multilingual-MiniLM-L12-v2

Q	F	AR	CR	AC
Q-001	1.00	1.00	1.00	1.00
Q-002	0.00	0.00	0.00	0.02
Q-003	1.00	1.00	1.00	1.00
Q-004	0.00	0.00	0.00	0.01
Q-005	1.00	0.91	0.00	0.04
Q-006	1.00	1.00	1.00	1.00
Q-007	0.00	0.00	0.00	0.03
Q-008	1.00	1.00	1.00	1.00
Q-009	0.50	0.43	1.00	0.15

Fortsetzung auf nächster Seite

Tabelle A.3 – *Fortsetzung von vorheriger Seite*

Q	**F**	**AR**	**CR**	**AC**
Q-010	1.00	1.00	1.00	1.00
Q-011	1.00	1.00	1.00	1.00
Q-012	1.00	1.00	1.00	1.00
Q-013	1.00	1.00	1.00	1.00
Q-014	1.00	1.00	1.00	1.00
Q-015	1.00	1.00	1.00	1.00
Q-016	1.00	0.92	1.00	0.73
Q-017	1.00	1.00	1.00	1.00
Q-018	0.50	0.22	0.00	0.17
Q-019	0.00	0.42	0.50	0.74
Q-020	0.00	0.33	1.00	0.57
Q-021	0.67	0.42	1.00	0.53
Q-022	0.50	0.33	1.00	0.19
Q-023	1.00	1.00	1.00	1.00
Q-024	1.00	0.95	1.00	0.60
Q-025	1.00	0.93	1.00	0.24
Q-026	1.00	1.00	1.00	1.00
Q-027	0.50	0.39	1.00	0.69
Q-028	1.00	1.00	1.00	1.00
Q-029	1.00	1.00	1.00	0.66
Q-030	1.00	1.00	1.00	0.85
Q-031	0.00	0.05	1.00	0.23
Q-032	1.00	0.91	1.00	0.69
Q-033	1.00	0.92	1.00	0.67
Q-034	1.00	1.00	1.00	1.00
Q-035	1.00	1.00	1.00	1.00
Q-036	1.00	1.00	1.00	1.00
Q-037	1.00	1.00	1.00	1.00
Q-038	1.00	1.00	1.00	1.00
Q-039	1.00	1.00	1.00	1.00
Q-040	0.50	0.32	0.00	0.73
Q-041	0.50	0.16	1.00	0.94

Fortsetzung auf nächster Seite

Tabelle A.3 – *Fortsetzung von vorheriger Seite*

Q	**F**	**AR**	**CR**	**AC**
Q-042	1.00	1.00	1.00	1.00
Q-043	0.00	0.40	1.00	1.00
Q-044	1.00	1.00	1.00	1.00
Q-045	0.00	0.42	1.00	0.66
Q-046	1.00	1.00	1.00	1.00
Q-047	0.00	0.09	0.00	0.24
Q-048	0.17	0.46	1.00	0.98
Q-049	1.00	0.91	1.00	0.84
Q-050	1.00	1.00	1.00	1.00
Q-051	1.00	0.98	1.00	0.66
Q-052	1.00	1.00	1.00	1.00
Q-053	1.00	0.98	1.00	0.23
Q-054	1.00	1.00	1.00	1.00
Q-055	0.00	0.46	1.00	0.85
Q-056	1.00	1.00	1.00	1.00
Q-057	1.00	0.97	1.00	0.66
Q-058	1.00	1.00	1.00	1.00
Q-059	1.00	1.00	1.00	1.00
Q-060	1.00	1.00	1.00	1.00
Q-061	0.00	0.39	1.00	0.99
Q-062	1.00	0.94	1.00	0.99
Q-063	1.00	1.00	1.00	1.00
Q-064	0.50	0.20	0.00	0.74
Q-065	1.00	1.00	1.00	1.00
Q-066	1.00	1.00	1.00	1.00
Q-067	1.00	1.00	1.00	0.99
Q-068	1.00	0.96	1.00	0.99
Q-069	1.00	1.00	1.00	1.00
Q-070	1.00	1.00	1.00	1.00
Q-071	1.00	1.00	1.00	1.00
Q-072	1.00	1.00	1.00	1.00
Q-073	1.00	1.00	1.00	1.00

Fortsetzung auf nächster Seite

Tabelle A.3 – *Fortsetzung von vorheriger Seite*

Q	F	AR	CR	AC
Q-074	1.00	1.00	1.00	1.00
Q-075	1.00	1.00	1.00	1.00
Q-076	0.50	0.25	1.00	0.74
Q-077	1.00	1.00	1.00	1.00
Q-078	0.00	0.06	1.00	0.24
Q-079	0.50	0.41	1.00	0.94
Q-080	1.00	1.00	1.00	1.00
Q-081	1.00	1.00	1.00	0.91
Q-082	1.00	1.00	1.00	1.00
Q-083	1.00	1.00	1.00	1.00
Q-084	1.00	1.00	1.00	1.00
Q-085	1.00	1.00	1.00	1.00
Q-086	1.00	1.00	1.00	1.00
Q-087	0.00	0.41	1.00	0.67
Q-088	0.50	0.34	1.00	0.73
Q-089	1.00	1.00	1.00	1.00
Q-090	1.00	1.00	1.00	1.00

A.1.4 intfloat/multilingual-e5-large

Tabelle A.4: Vollständige Evaluationsergebnisse für intfloat/multilingual-e5-large

Q	F	AR	CR	AC
Q-001	1.00	1.00	1.00	1.00
Q-002	0.50	0.63	0.00	0.15
Q-003	0.00	0.00	1.00	0.01
Q-004	0.00	0.00	0.00	0.02
Q-005	1.00	1.00	0.00	0.04
Q-006	1.00	1.00	1.00	1.00
Q-007	0.00	0.00	0.00	0.03

Fortsetzung auf nächster Seite

Tabelle A.4 – *Fortsetzung von vorheriger Seite*

Q	**F**	**AR**	**CR**	**AC**
Q-008	1.00	1.00	1.00	0.93
Q-009	0.00	0.40	0.00	0.17
Q-010	1.00	1.00	1.00	1.00
Q-011	0.00	0.00	0.00	0.02
Q-012	1.00	1.00	1.00	1.00
Q-013	0.00	0.32	0.00	0.21
Q-014	1.00	1.00	1.00	1.00
Q-015	0.00	0.31	1.00	0.94
Q-016	1.00	1.00	1.00	0.85
Q-017	1.00	1.00	1.00	1.00
Q-018	0.50	0.22	0.00	0.17
Q-019	0.00	0.42	0.50	0.75
Q-020	1.00	0.99	1.00	0.59
Q-021	0.00	0.42	1.00	0.53
Q-022	1.00	1.00	1.00	1.00
Q-023	1.00	0.97	1.00	0.99
Q-024	0.00	0.00	0.00	0.02
Q-025	1.00	0.93	1.00	0.24
Q-026	1.00	1.00	1.00	1.00
Q-027	1.00	0.91	1.00	0.70
Q-028	1.00	1.00	1.00	1.00
Q-029	0.50	0.37	1.00	0.15
Q-030	0.00	0.00	0.00	0.08
Q-031	0.50	0.05	1.00	0.61
Q-032	1.00	1.00	1.00	0.69
Q-033	1.00	0.94	0.00	0.65
Q-034	1.00	1.00	1.00	1.00
Q-035	1.00	1.00	1.00	1.00
Q-036	1.00	1.00	1.00	1.00
Q-037	1.00	1.00	1.00	1.00
Q-038	1.00	1.00	1.00	1.00
Q-039	0.00	0.37	0.00	0.21

Fortsetzung auf nächster Seite

Tabelle A.4 – *Fortsetzung von vorheriger Seite*

Q	**F**	**AR**	**CR**	**AC**
Q-040	1.00	0.92	0.00	0.71
Q-041	0.50	0.16	1.00	0.94
Q-042	1.00	1.00	1.00	1.00
Q-043	1.00	1.00	1.00	1.00
Q-044	1.00	1.00	1.00	1.00
Q-045	0.00	0.00	1.00	0.02
Q-046	0.00	0.33	1.00	0.74
Q-047	1.00	1.00	1.00	1.00
Q-048	1.00	1.00	1.00	1.00
Q-049	1.00	1.00	1.00	1.00
Q-050	1.00	1.00	1.00	1.00
Q-051	0.00	0.00	1.00	0.02
Q-052	1.00	1.00	1.00	1.00
Q-053	0.00	0.00	0.00	0.02
Q-054	1.00	1.00	1.00	1.00
Q-055	0.00	0.43	1.00	0.92
Q-056	0.50	0.38	0.00	0.47
Q-057	0.00	0.42	1.00	0.91
Q-058	1.00	1.00	1.00	1.00
Q-059	1.00	0.98	1.00	0.24
Q-060	1.00	0.90	1.00	1.00
Q-061	0.00	0.38	1.00	0.99
Q-062	1.00	1.00	1.00	1.00
Q-063	1.00	1.00	1.00	1.00
Q-064	1.00	0.97	0.00	0.99
Q-065	1.00	1.00	1.00	1.00
Q-066	1.00	1.00	1.00	1.00
Q-067	1.00	0.92	1.00	0.99
Q-068	1.00	0.95	1.00	0.99
Q-069	1.00	1.00	1.00	1.00
Q-070	1.00	1.00	1.00	1.00
Q-071	1.00	1.00	1.00	1.00

Fortsetzung auf nächster Seite

Tabelle A.4 – *Fortsetzung von vorheriger Seite*

Q	**F**	**AR**	**CR**	**AC**
Q-072	1.00	1.00	1.00	1.00
Q-073	1.00	1.00	1.00	1.00
Q-074	1.00	1.00	1.00	1.00
Q-075	1.00	1.00	1.00	1.00
Q-076	1.00	1.00	1.00	1.00
Q-077	1.00	1.00	1.00	1.00
Q-078	0.00	0.04	0.00	0.24
Q-079	0.50	0.41	1.00	0.94
Q-080	1.00	1.00	1.00	1.00
Q-081	1.00	0.97	1.00	0.66
Q-082	1.00	1.00	1.00	1.00
Q-083	1.00	1.00	1.00	1.00
Q-084	1.00	1.00	1.00	1.00
Q-085	1.00	1.00	1.00	1.00
Q-086	0.00	0.35	1.00	0.99
Q-087	0.00	0.40	1.00	0.67
Q-088	0.00	0.32	1.00	0.73
Q-089	1.00	1.00	1.00	1.00
Q-090	1.00	1.00	1.00	1.00

Literatur

[1] Mahmoud Amiri und Thomas Bocklitz. *Chunk Twice, Embed Once: A Systematic Study of Segmentation and Representation Trade-offs in Chemistry-Aware Retrieval-Augmented Generation*. 06/2025. DOI: 10.48550/arXiv.2506.17277. arXiv: 2506.17277 [cs]. (Besucht am 29. 10. 2025) (siehe S. 6, 18).

[2] Payal Bajaj u. a. *MS MARCO: A Human Generated MAchine Reading COmprehension Dataset*. 10/2018. DOI: 10.48550/arXiv.1611.09268. arXiv: 1611.09268 [cs]. (Besucht am 03. 02. 2026) (siehe S. 32).

[3] Jaime Carbonell und Jade Goldstein. „The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries“. In: *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '98. New York, NY, USA: Association for Computing Machinery, 08/1998, S. 335–336. DOI: 10.1145/290941.291025. (Besucht am 29. 11. 2025) (siehe S. 13).

[4] Davidjonietz. *Building and evaluating multilingual RAG systems*. 11/2024. (Besucht am 01. 11. 2025) (siehe S. 5, 8, 12).

[5] Matthijs Douze u. a. *The Faiss Library*. 10/2025. DOI: 10.48550/arXiv.2401.08281. arXiv: 2401.08281 [cs]. (Besucht am 06. 12. 2025) (siehe S. 11).

[6] Shahul Es u. a. *Ragas: Automated Evaluation of Retrieval Augmented Generation*. 04/2025. DOI: 10.48550/arXiv.2309.15217. arXiv: 2309.15217 [cs]. (Besucht am 05. 11. 2025) (siehe S. 16 ff., 37).

[7] Yunfan Gao u. a. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 03/2024. DOI: 10.48550/arXiv.2312.10997. arXiv: 2312.10997 [cs]. (Besucht am 13. 09. 2025) (siehe S. 10).

[8] Shailja Gupta, Rajesh Ranjan und Surya Narayan Singh. *A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions*. 10/2024. DOI: 10.48550/arXiv.2410.12837. arXiv: 2410.12837 [cs]. (Besucht am 15. 10. 2025) (siehe S. 12).

[9] Kailash A. Hambarde und Hugo Proenca. „Information Retrieval: Recent Advances and Beyond“. In: *IEEE Access* 11 (2023), S. 76581–76604. DOI: 10.1109/ACCESS.2023.3295776. arXiv: 2301.08801 [cs]. (Besucht am 25. 10. 2025) (siehe S. 5).

[10] Ayman Asad Khan u. a. *Developing Retrieval Augmented Generation (RAG) Based LLM Systems from PDFs: An Experience Report*. 10/2024. DOI: 10.48550/arXiv.2410.15944. arXiv: 2410.15944 [cs]. (Besucht am 02. 11. 2025) (siehe S. 10).

[11] Aadit Kshirsagar. „Enhancing RAG Performance Through Chunking and Text Splitting Techniques“. In: *International Journal of Scientific Research in Computer Science, Enginee-*

ring and Information Technology 10 (09/2024), S. 151–158. DOI: 10.32628/CSEIT2410593 (siehe S. 6).

[12] Songjiang Lai u. a. *Enhancing Technical Documents Retrieval for RAG*. 09/2025. DOI: 10.48550/arXiv.2509.04139. arXiv: 2509.04139 [cs]. (Besucht am 13. 09. 2025) (siehe S. 18).

[13] David Lau u. a. „Multimodal RAG Analysis of Product Datasheet“. In: *Open International Journal of Informatics* 12.2 (12/2024), S. 1–12. DOI: 10.11113/oiji2024.12n2.309. (Besucht am 13. 09. 2025) (siehe S. 18).

[14] Patrick Lewis u. a. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 04/2021. DOI: 10.48550/arXiv.2005.11401. arXiv: 2005.11401 [cs]. (Besucht am 13. 09. 2025) (siehe S. 8 f., 14 f.).

[15] *List of Available Metrics - Ragas*. <https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/> (besucht am 29. 01. 2026) (siehe S. 16 f.).

[16] Nelson F. Liu u. a. *Lost in the Middle: How Language Models Use Long Contexts*. 11/2023. DOI: 10.48550/arXiv.2307.03172. arXiv: 2307.03172 [cs]. (Besucht am 29. 11. 2025) (siehe S. 14).

[17] Yu A. Malkov und D. A. Yashunin. *Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs*. 08/2018. DOI: 10.48550/arXiv.1603.09320. arXiv: 1603.09320 [cs]. (Besucht am 06. 12. 2025) (siehe S. 11).

[18] Christopher Manning, Prabhakar Raghavan und Hinrich Schuetze. „Introduction to Information Retrieval“. In: (2009) (siehe S. 4).

[19] Lang Mei u. a. *A Survey of Multimodal Retrieval-Augmented Generation*. 03/2025. DOI: 10.48550/arXiv.2504.08748. arXiv: 2504.08748 [cs]. (Besucht am 13. 09. 2025) (siehe S. 18).

[20] „Okapi BM25“. In: *Wikipedia* (12/2025). (Besucht am 31. 01. 2026) (siehe S. 13).

[21] Kobkaew Opasjumruskit, Sirko Schindler und Diana Peters. „Automatic Data Sheet Information Extraction for Supporting Model-Based Systems Engineering“. In: *Cooperative Design, Visualization, and Engineering*. Hrsg. von Yuhua Luo. Bd. 12983. Cham: Springer International Publishing, 2021, S. 97–102. DOI: 10.1007/978-3-030-88207-5_10. (Besucht am 01. 11. 2025) (siehe S. 4 f., 15).

[22] *RAG*. https://huggingface.co/docs/transformers/en/model_doc/rag. (Besucht am 02. 11. 2025) (siehe S. 9).

[23] Nils Reimers und Iryna Gurevych. *Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks*. 08/2019. DOI: 10.48550/arXiv.1908.10084. arXiv: 1908.10084 [cs]. (Besucht am 29. 11. 2025) (siehe S. 12, 30, 41).

[24] Kaitao Song u. a. *MPNet: Masked and Permuted Pre-training for Language Understanding*. 11/2020. DOI: 10.48550/arXiv.2004.09297. arXiv: 2004.09297 [cs]. (Besucht am 22. 01. 2026) (siehe S. 40).

[25] Ryota Tanaka u. a. *VDocRAG: Retrieval-Augmented Generation over Visually-Rich Documents*. 04/2025. DOI: 10.48550/arXiv.2504.09795. arXiv: 2504.09795 [cs]. (Besucht am 13. 09. 2025) (siehe S. 18).

[26] *Text Splitters | LangChain Reference*. https://reference.langchain.com/python/langchain_text_splitters/. (Besucht am 03. 02. 2026) (siehe S. 24, 29).

[27] Ashish Vaswani u. a. *Attention Is All You Need*. 08/2023. DOI: 10.48550/arXiv.1706.03762. arXiv: 1706.03762 [cs]. (Besucht am 02. 11. 2025) (siehe S. 6 ff.).

[28] Liang Wang u. a. *Multilingual E5 Text Embeddings: A Technical Report*. 02/2024. DOI: 10.48550/arXiv.2402.05672. arXiv: 2402.05672 [cs]. (Besucht am 22. 01. 2026) (siehe S. 42).

[29] Qiuchen Wang u. a. *ViDoRAG: Visual Document Retrieval-Augmented Generation via Dynamic Iterative Reasoning Agents*. 06/2025. DOI: 10.48550/arXiv.2502.18017. arXiv: 2502.18017 [cs]. (Besucht am 13. 09. 2025) (siehe S. 18).

[30] Wenhui Wang u. a. *MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers*. 04/2020. DOI: 10.48550/arXiv.2002.10957. arXiv: 2002.10957 [cs]. (Besucht am 22. 01. 2026) (siehe S. 30, 39).

[31] Hengran Zhang u. a. *LLM-Specific Utility: A New Perspective for Retrieval-Augmented Generation*. 10/2025. DOI: 10.48550/arXiv.2510.11358. arXiv: 2510.11358 [cs]. (Besucht am 15. 10. 2025) (siehe S. 18).

[32] Peitian Zhang u. a. *Hybrid Inverted Index Is a Robust Accelerator for Dense Retrieval*. 10/2023. DOI: 10.48550/arXiv.2210.05521. arXiv: 2210.05521 [cs]. (Besucht am 06. 12. 2025) (siehe S. 11).

[33] Dongfang Zhao. *FRAG: Toward Federated Vector Database Management for Collaborative and Secure Retrieval-Augmented Generation*. 2024. DOI: 10.48550/ARXIV.2410.13272. (Besucht am 13. 09. 2025) (siehe S. 12, 18).

[34] Zijie Zhong u. a. *Mix-of-Granularity: Optimize the Chunking Granularity for Retrieval-Augmented Generation*. 01/2025. DOI: 10.48550/arXiv.2406.00456. arXiv: 2406.00456 [cs]. (Besucht am 29. 10. 2025) (siehe S. 18).

[35] Yutao Zhu u. a. „Large Language Models for Information Retrieval: A Survey“. In: *ACM Transactions on Information Systems* (09/2025), S. 3748304. DOI: 10.1145/3748304. arXiv: 2308.07107 [cs]. (Besucht am 01. 11. 2025) (siehe S. 4, 6, 8, 11, 13).