

**BACHELORARBEIT**

# **Optimierung und Evaluation der Retrieval-Qualität in lokalen RAG-Systemen**

---

vorgelegt am 05.08.2026  
Cengiz-Han Kerim Brüggen

Erstprüferin: Prof. Dr. Larissa Putzar  
Zweitprüfer: Kilian Hildebrandt

---

**HOCHSCHULE FÜR ANGEWANDTE  
WISSENSCHAFTEN HAMBURG**

Fakultät Informatik und Digitale Gesellschaft  
Finkenau 35  
22081 Hamburg

# Zusammenfassung

Retrieval-Augmented Generation (RAG) ist eine Methode, um Large Language Models mit externen Daten zu erweitern. Die Qualität und der Energieverbrauch einer RAG-Pipeline werden von verschiedenen Komponenten und Parametern beeinflusst. Ziel dieser Arbeit ist es, diese Parameter zu untersuchen und die Qualität der RAG-Pipeline zu optimieren. Dabei konzentriert sich diese Arbeit auf Parameter, die für die Retrieval-Qualität relevant sind.

Es werden unterschiedliche Chunking-Konfigurationen, Embedding-Modelle und Retrieval-Verfahren auf einer lokalen RAG-Pipeline untersucht. Als Datengrundlage wird eine deutsche Dissertation verwendet, zu der ein Golden Set mit Fragen und Referenzantworten erstellt wurde. Zum Bewerten werden Metriken vom Test-Framework RAGAS verwendet. Zusätzlich wird der Energieverbrauch gemessen.

In dem untersuchten Anwendungsfall zeigen die Ergebnisse, dass größere Chunks bessere Gesamtscores als kleinere Chunks erzielen. Außerdem führt eine Überlappung von Chunks zu keiner Verbesserung des Gesamtscores, erhöht jedoch den Energieverbrauch. Semantisches Chunksen wies ebenfalls einen deutlich erhöhten Energieverbrauch auf, erzielte jedoch niedrigere Gesamtscores.

Die Konfiguration mit dem höchsten Gesamtscore ist die mit dem rekursiven Chunker, dem BAAI/bge-m3-Embedder und dem Dense-Retrieval. Der Energieverbrauch von dem BAAI/bge-m3-Embedder ist aber deutlich höher im Vergleich zu den anderen Embedding-Modellen. Für eine lokale RAG-Pipeline bietet die Konfiguration mit der rekursiven Chunking-Methode, dem intfloat/multilingual-e5-base-Embedder und dem Dense-Retriever den besseren Kompromiss zwischen Gesamtscore und Energieverbrauch. Die Ergebnisse gelten nur für den untersuchten Anwendungsfall und sind aufgrund der ausgewählten Datengrundlage und des begrenzten Golden Sets nicht auf andere RAG-Pipelines übertragbar.

## Abstract

Retrieval-Augmented Generation (RAG) is a method that extends Large Language Models with external data. The quality and energy consumption of a RAG pipeline are affected by different components and parameters. The aim of this thesis is to examine these parameters and optimize the quality of the RAG pipeline. This thesis focuses on parameters that are relevant to retrieval quality. Different chunking configurations, embedding models, and retrieval methods are examined in a local RAG pipeline. The data basis comes from a German dissertation, for which a golden set with questions and reference answers was created. Metrics from the RAGAS evaluation framework are used for the evaluation and in addition, energy consumption is measured.

For the examined use case, the results show that larger chunks achieve better overall scores than smaller chunks. Furthermore, chunk overlap does not improve the overall score, but increases energy consumption. Semantic chunking also showed significantly higher energy consumption, but achieved lower overall scores.

The configuration with the highest overall score is the recursive chunker with the BAAI/bge-m3 embedder and dense retrieval. But the energy consumption of the BAAI/bge-m3 embedder is significantly higher compared to the other embedding models. For a local RAG pipeline, the configuration with the recursive chunking method, the intfloat/multilingual-e5-base embedder, and the dense retriever offers a better compromise between overall score and energy consumption.

The results only apply to the examined use case and cannot be transferred to other RAG pipelines due to the selected data basis and the limited golden set.

# Inhaltsverzeichnis

|                                                                         |           |
|-------------------------------------------------------------------------|-----------|
| <b>Abbildungsverzeichnis</b>                                            | <b>IV</b> |
| <b>Tabellenverzeichnis</b>                                              | <b>V</b>  |
| <b>1 Einleitung</b>                                                     | <b>1</b>  |
| <b>2 Grundlagen</b>                                                     | <b>3</b>  |
| 2.1 Künstliche Intelligenz . . . . .                                    | 3         |
| 2.2 Maschinelles Lernen . . . . .                                       | 3         |
| 2.3 Neuronale Netze . . . . .                                           | 4         |
| 2.4 Natural Language Processing . . . . .                               | 5         |
| 2.5 Large Language Models . . . . .                                     | 5         |
| 2.6 LLM-Limitationen . . . . .                                          | 8         |
| 2.7 Embeddings . . . . .                                                | 8         |
| 2.7.1 Word2Vec . . . . .                                                | 10        |
| 2.7.2 Bidirectional Encoder Representations from Transformers . . . . . | 10        |
| 2.8 Vektordatenbanken . . . . .                                         | 11        |
| 2.9 Information Retrieval . . . . .                                     | 12        |
| 2.10 Retrieval-Augmented Generation . . . . .                           | 13        |
| 2.10.1 Architektur . . . . .                                            | 13        |
| 2.10.2 Warum ist präzises Retrieval wichtig? . . . . .                  | 15        |
| <b>3 Verwandte Arbeiten</b>                                             | <b>16</b> |
| 3.1 Einfluss der Dokumentensegmentierung . . . . .                      | 16        |
| 3.1.1 Bedeutung der Segmentierung für das Retrieval . . . . .           | 16        |
| 3.1.2 Chunk-Größe . . . . .                                             | 16        |
| 3.1.3 Überlappung benachbarter Chunks . . . . .                         | 17        |
| 3.1.4 Struktur- und semantische Segmentierung . . . . .                 | 17        |
| 3.2 Embedding . . . . .                                                 | 18        |
| 3.3 Retrieval-Verfahren . . . . .                                       | 18        |
| 3.4 Abgrenzung der eigenen Arbeit . . . . .                             | 19        |

|          |                                                                  |           |
|----------|------------------------------------------------------------------|-----------|
| <b>4</b> | <b>Methodik</b>                                                  | <b>20</b> |
| 4.1      | Aufbau der RAG-Pipeline                                          | 20        |
| 4.1.1    | Haystack                                                         | 20        |
| 4.1.2    | Weaviate                                                         | 21        |
| 4.1.3    | LLM                                                              | 21        |
| 4.1.4    | Datensatz                                                        | 21        |
| 4.1.5    | Versuchsaufbau                                                   | 22        |
| 4.2      | Untersuchte Systemkomponenten                                    | 23        |
| 4.2.1    | Chunking-Strategie                                               | 23        |
| 4.2.2    | Embedding-Modelle                                                | 25        |
| 4.2.3    | Retrieval-Verfahren                                              | 26        |
| 4.3      | Experimenteller Ablauf                                           | 26        |
| 4.3.1    | Begründung des sequentiellen Ablaufs                             | 26        |
| 4.3.2    | Phase 1: Vorauswahl der Chunking-Konfigurationen                 | 27        |
| 4.3.3    | Phase 2: Testen der drei besten Chunking-Konfigurationen         | 27        |
| 4.3.4    | Phase 3: Vergleich der Embedding-Modelle und Retrieval-Verfahren | 27        |
| 4.4      | Evaluations-Frameworks                                           | 28        |
| 4.4.1    | RAGAS                                                            | 28        |
| 4.4.2    | Bewertungsmetriken                                               | 28        |
| 4.4.3    | CodeCarbon                                                       | 30        |
| 4.5      | Hardwarekonfiguration                                            | 30        |
| 4.6      | Softwareversionen                                                | 31        |
| <b>5</b> | <b>Ergebnisse</b>                                                | <b>32</b> |
| 5.1      | Metriken                                                         | 32        |
| 5.1.1    | Phase 1: Vorauswahl der Chunking-Konfigurationen                 | 33        |
| 5.1.2    | Phase 2: Testen der besten Chunking-Konfigurationen              | 38        |
| 5.1.3    | Phase 3: Vergleich der Embedding-Modelle und Retrieval-Verfahren | 40        |
| 5.1.4    | Beste Kombinationen                                              | 42        |
| <b>6</b> | <b>Diskussion</b>                                                | <b>43</b> |
| 6.1      | Einfluss der Chunk-Konfiguration                                 | 43        |
| 6.1.1    | Einfluss der Chunk-Größe                                         | 43        |
| 6.1.2    | Einfluss der Chunk-Überlappung                                   | 43        |
| 6.1.3    | Rekursives und semantisches Chunking                             | 44        |
| 6.2      | Einfluss von Embedding-Modell und Retrieval-Strategie            | 44        |
| 6.3      | Empfohlene Konfiguration für ein lokales RAG-System              | 45        |
| 6.4      | Limitationen                                                     | 45        |
| 6.4.1    | Ungetestete Kombinationen                                        | 45        |
| 6.4.2    | Begrenzter Anwendungsfall                                        | 46        |

|          |                                             |           |
|----------|---------------------------------------------|-----------|
| 6.4.3    | Begrenzte Anzahl an Fragen . . . . .        | 46        |
| 6.4.4    | Probleme der RAGAS-Metriken . . . . .       | 47        |
| 6.4.5    | Nicht untersuchte Parameter . . . . .       | 48        |
| <b>7</b> | <b>Fazit</b>                                | <b>49</b> |
| 7.1      | Ausblick . . . . .                          | 50        |
|          | <b>Literatur</b>                            | <b>51</b> |
|          | <b>Erklärung zur Nutzung generativer KI</b> | <b>56</b> |

# Abbildungsverzeichnis

|      |                                                                    |    |
|------|--------------------------------------------------------------------|----|
| 2.1  | Neuronales Netz . . . . .                                          | 4  |
| 2.2  | Transformer-Architektur . . . . .                                  | 7  |
| 2.3  | Beziehung zwischen Embeddeten Wörtern . . . . .                    | 9  |
| 2.4  | Hierarchical Navigable Small World Verfahren . . . . .             | 12 |
| 2.5  | RAG-Architektur . . . . .                                          | 14 |
| 5.1  | Retrieval-Metrik Ergebnisse Phase eins/wiederholung eins . . . . . | 32 |
| 5.2  | Gesamtergebnisse der ersten Wiederholung . . . . .                 | 35 |
| 5.3  | Stromverbrauch der ersten Wiederholung . . . . .                   | 35 |
| 5.4  | Gesamtergebnisse der zweiten Wiederholung . . . . .                | 36 |
| 5.5  | Stromverbrauch der zweiten Wiederholung . . . . .                  | 36 |
| 5.6  | Gesamtergebnisse der dritten Wiederholung . . . . .                | 37 |
| 5.7  | Stromverbrauch der dritten Wiederholung . . . . .                  | 37 |
| 5.8  | Gesamtergebnisse Phase 2 . . . . .                                 | 38 |
| 5.9  | Stromverbrauch Phase 2 . . . . .                                   | 39 |
| 5.10 | Endergebnis . . . . .                                              | 40 |
| 5.11 | Endergebnis Energieverbrauch . . . . .                             | 41 |

# Tabellenverzeichnis

|     |                                           |    |
|-----|-------------------------------------------|----|
| 4.1 | Untersuchte Chunking-Strategien . . . . . | 24 |
|-----|-------------------------------------------|----|

# 1 Einleitung

Large Language Models (LLMs) können eine große Menge an sprachlichen Informationen verarbeiten und dadurch Text generieren oder Fragen beantworten. Ihr Wissen ist jedoch auf bestimmte Daten begrenzt und sie können zum Beispiel keine Fragen zu proprietären oder privaten Daten beantworten.

Retrieval-Augmented Generation (RAG) ist eine Methode, um LLMs mit externen Daten zu erweitern. So können RAG-Systeme Fragen zu externen Daten beantworten und sie referenzieren. RAG-Systeme sind vor allem für Anwendungsfälle interessant, in denen nicht öffentliche Informationen verarbeitet werden müssen. Durch den Betrieb eines lokalen RAG-Systems können diese Daten lokal verarbeitet werden, ohne dass sie an externe Cloud-Dienste weitergegeben werden müssen. Diese Arbeit untersucht deswegen vor allem lokale RAG-Systeme und erhebt zu den Qualitätsmetriken zusätzlich den Energieverbrauch.

Die Qualität einer RAG-Pipeline hängt stark von verschiedenen Komponenten ab. Der Retrieval-Schritt ist besonders wichtig für die RAG-Pipeline, da hier die Informationen aus den externen Daten abgerufen werden. Es existieren verschiedene Komponenten, die die Retrieval-Qualität einer RAG-Pipeline beeinflussen. Diese Arbeit wird sich vor allem mit der Vorverarbeitung der Dokumente und dem Retriever auseinandersetzen. Dabei werden die Dokumentensegmentierung, das Embedding-Modell und das Retrieval-Verfahren untersucht.

Aktuelle Forschungen untersuchen meist einzelne Komponenten einer RAG-Pipeline isoliert voneinander. Es fehlt eine Untersuchung von mehreren Komponenten unter Berücksichtigung von Antwortqualität und des Ressourcenverbrauchs.

Ziel dieser Arbeit ist es, eine geeignete Konfiguration aus Dokumentensegmentierung, Embedding-Modell und Retrieval-Verfahren für eine lokal betriebene RAG-Pipeline zu finden. Dabei soll untersucht werden, welchen Einfluss die untersuchten Komponenten auf die Qualität der RAG-Pipeline haben. Zusätzlich wird der Energieverbrauch erfasst, um zu untersuchen, ob eine Qualitätssteigerung einen erhöhten Energieverbrauch rechtfertigt.

Aus der Zielsetzung ergeben sich folgende Forschungsfragen:

**Hauptforschungsfrage** Welche Konfiguration aus Dokumentensegmentierung, Embedding-Modell und Retrieval-Verfahren bietet in einer lokal betriebenen RAG-Pipeline das beste Verhältnis zwischen Qualität und Energieverbrauch?

## **Unterfragen**

1. Welchen Einfluss haben Chunk-Größe, Chunk-Überlappung und Chunking-Verfahren auf die Qualität und den Energieverbrauch der RAG-Pipeline?
2. Lohnt sich ein semantisches Chunking-Verfahren in Hinblick auf Qualität und Energieverbrauch?
3. Welchen Einfluss hat die Wahl des Embedding-Modells und des Retrieval-Verfahrens auf die Qualität und den Energieverbrauch der RAG-Pipeline?

## 2 Grundlagen

Dieses Kapitel vermittelt die Grundlagen, die für das Verständnis von RAG-Systemen und den nachfolgenden Versuch erforderlich sind.

### 2.1 Künstliche Intelligenz

Künstliche Intelligenz (KI) ist ein Teilgebiet der Informatik, das sich mit der Erforschung und Entwicklung intelligenter Systeme auseinandersetzt. Diese Systeme können ihre Umgebung über Sensorik wahrnehmen, Informationen verarbeiten und auf dieser Grundlage Handlungen ausführen oder Entscheidungen treffen [1, S. viii]. Solche Systeme werden als Agenten bezeichnet. Nach Russell und Norvig werden Agenten als rationale Agenten betrachtet, deren Ziel es ist, rational zu handeln. Rationales Handeln bedeutet in diesem Zusammenhang, dass ein Agent auf Grundlage der verfügbaren Informationen die Handlung auswählt, die den erwarteten Erfolg maximiert [1, S. 1-5, 37]. Dieses Ziel kann erreicht werden, indem der Agent einen manuell aufgestellten Regelsatz bekommt, an den er sich halten soll. Das Problem an solch einer Strategie ist, dass mit der Komplexität des Problems auch die Komplexität der Regeln steigt und potenziell sogar Ausnahmen und Sonderfälle betrachtet werden müssen. Aus diesem Grund stoßen solche Systeme bei komplexen Problemen oft an ihre Grenzen [2, S. 2].

### 2.2 Maschinelles Lernen

Eine alternative Lösung, um den erwarteten Erfolg von einer Aufgabe zu maximieren, ist das maschinelle Lernen. Beim maschinellen Lernen werden die Entscheidungsregeln aus Daten abgeleitet. Im Rahmen des überwachten Lernens wird hierfür ein Trainingsdatensatz verwendet, der aus Eingabedaten und Zielwerten besteht. Das Modell erstellt eine Vorhersage auf Grundlage der Eingabewerte. Während des Trainings werden die Modellparameter so angepasst, dass die Vorhersagen des Modells möglichst gut mit den zugehörigen Zielwerten übereinstimmen. [2, S.2].

Nach Abschluss des Trainings kann das Modell auch bisher unbekannte Eingabedaten eine Vorhersage treffen. Diese Fähigkeit wird als Generalisierung bezeichnet und ist ein zentrales Ziel des maschinellen Lernens [2, S.2].

## 2.3 Neuronale Netze

Künstliche neuronale Netze sind leistungsfähige Verfahren des maschinellen Lernens. Der Begriff „neuronale Netze“ geht auf frühere Versuche zurück, die Informationsverarbeitung biologischer Nervensysteme mathematisch zu beschreiben. Heutzutage werden künstliche neuronale Netze vor allem als statistische Modelle verstanden, die komplexe Muster in großen Datenmengen erkennen können[2, S.226]. In Abbildung 2.1 ist der typische Aufbau eines neuronalen Netzes dargestellt.

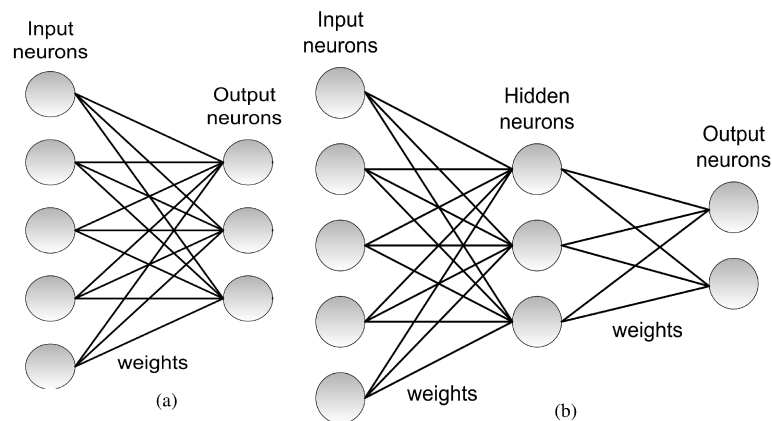


Abbildung 2.1: Neuronales Netz

Quelle: [3]

Neuronale Netze bestehen aus verschiedenen Schichten, der Eingabeschicht, keiner, einer oder mehrerer verborgenen Schichten und der Ausgabeschicht.

Die Tiefe des Netzes wird über die Anzahl der Schichten definiert und die Breite der Schicht wird über die Anzahl der Einheiten in dieser Schicht definiert [4, S.168-169].

In Abbildung 2.1 ist auf der linken Seite ein neuronales Netz ohne verborgene Schichten mit der Tiefe zwei dargestellt. Auf der rechten Seite ist ein neuronales Netz mit einer verborgenen Schicht dargestellt, die Eingabeschicht hat hier die Breite fünf, die verborgene Schicht die Breite drei und die Ausgabeschicht die Breite zwei, insgesamt hat dieses neuronale Netz eine Tiefe von drei.

Die Einheiten werden als Neuronen bezeichnet. Die Schichten bestehen aus Neuronen, die parallel arbeiten. Die Neuronen bekommen mehrere Eingabewerte von anderen Neuronen,

diese werden verarbeitet und daraus wird ein Aktivierungswert berechnet. [4, S.169]. Die Schichten sind über ihre Neuronen mit der jeweiligen nächsten Schicht über Gewichte verbunden, diese steuern, wie sehr die Schicht den Aktivierungswert der Neuronen der nächsten Schicht beeinflusst. Zusätzlich hat jedes Neuron einen Bias, der einen Schwellenwert oder eine Grundaktivierung definiert. Diese Werte kombiniert sind die Parameter eines neuronalen Netzes, die sich beim Training verändern [4, S.174, S.6].

Die Eingabeschicht nimmt Daten auf die beobachtbar sind, beispielsweise Pixel eines Bildes. Die verborgenen Schichten extrahieren aus den Daten der Eingabeschicht Features, die mit zunehmender Tiefe des Netzwerkes immer abstrakter werden. Die Ausgabeschicht erstellt eine Vorhersage basierend auf den Eingabedaten und den aktuellen Parametern des Neuronalen Netzes. [4, S.6, S.169].

## 2.4 Natural Language Processing

Natural Language Processing ist ein Teilgebiet der Künstlichen Intelligenz, der sich mit dem Verstehen, Analysieren und Generieren menschlicher Sprache auseinandersetzt.

Natürliche Sprache besteht grundlegend aus Wörtern, die Regeln der Grammatik und Semantik befolgen. Grammatik diktiert die Struktur und Reihenfolge, in der Wörter angeordnet sein müssen, damit ein Satz zulässig ist. Semantik beschreibt die Bedeutung dieser Worte.

Aus diesen Regeln lässt sich aber keine feste Menge an Sätzen sammeln, die eine Sprache definieren würde, weil die Grammatik variabel sein kann und durchaus auch Unstimmigkeiten existieren können, ob ein Satz grammatikalisch richtig ist. Sie werden deswegen häufig als Wahrscheinlichkeitsverteilung modelliert, Sprachmodelle ordnen Wortabfolgen Wahrscheinlichkeiten zu und bewerten, wie plausibel eine Wortabfolge in einer Sprache ist.

Auch die Semantik von Sätzen kann unstimmig sein, denn ein Satz oder ein Wort kann je nach Kontext unterschiedlich interpretiert werden, weshalb auch hier eine Wahrscheinlichkeit über mehrere Bedeutungen gezogen wird [1].

## 2.5 Large Language Models

Large Language Models (LLMs) basieren auf künstlichen neuronalen Netzen [5, S. 59]. Sie können Wörter bzw. Tokens auf Grundlage des vorangegangenen Kontexts vorhersagen. Dies tun LLMs, indem sie eine Wahrscheinlichkeitsverteilung über mögliche nachfolgende Wörter bzw. Tokens erzeugen [5, S. 147].

LLMs sind auf großen Textdatenmengen trainiert und benutzen die Transformer-Architektur.

Transformer sind eine besondere Art von neuronalen Netzen, die heute die Standardarchitektur für Sprachmodelle sind. Sie wurden von Vaswani et al. im Paper „Attention Is All You Need“ [6] eingeführt und haben die Eigenschaft, dass sie ganze Sequenzen von Eingabedaten verarbeiten können und Elemente während des Trainings parallel verarbeiten können [5, S.173-174].

In der Sprachverarbeitung werden diese Eingabedaten Tokens genannt. Ein Token kann ein ganzes Wort, ein Wortteil, ein einzelner Buchstabe oder ein Satzzeichen sein. Wörter werden normalerweise in Tokens umgewandelt, um sie verarbeitbarer zu machen, das nennt man Tokenisierung [5, S.13].

Die Self-Attention ist ein wichtiger Mechanismus von Transformern, der dafür sorgt, dass Beziehungen zwischen Tokens erkannt werden können. Das Modell gewichtet relevante Tokens in der aktuellen Eingabesequenz, deren Länge durch ein sogenanntes Kontextfenster begrenzt wird, kombiniert deren Informationen und erkennt dadurch semantische Beziehungen auch über größere Entfernungen hinweg[5, S.174].

Auf Abbildung 2.2 sieht man eine typische Transformer-Architektur. Die Eingabetokens werden entlang des Residualstroms verarbeitet. Die Self-Attention und Feedforward-Schichten ergänzen Informationen auf dieser Basis kann das Modell dann eine Wahrscheinlichkeitsverteilung für das nächste Token erzeugen [5, S. 173-174].

LLMs besitzen einen Tokenizer, der Wörter in Tokens zerlegt und jedem dieser Tokens eine Token-ID zuordnet. Die Token-ID verweist auf einen Eintrag in einer gelernten Embedding-Matrix, in der jedem Token eine mehrdimensionale Vektorrepräsentation zugeordnet ist. So können Tokens maschinell verarbeitet werden. Wie diese Vektoren aussehen, wird während des Trainings gelernt [5, S. 16-17].

LLMs können Gespräche mit Menschen führen, Texte vervollständigen oder sogar ganz neue Inhalte basierend auf Prompts generieren. Prompts sind Texteingaben, auf deren Grundlage LLMs Folge-Tokens generieren. Prompts können direkte Fragen sein, die das LLM beantworten soll, Anweisungen sein, die das LLM befolgen oder ausführen soll, oder eine Struktur vorgeben, an der sich das LLM orientieren soll. [5, S. 150-152].

Umso genauer ein Prompt formuliert ist und wenn mögliche Antwortoptionen vorgegeben sind, desto besser ist in der Regel die Leistung des LLMs [7].

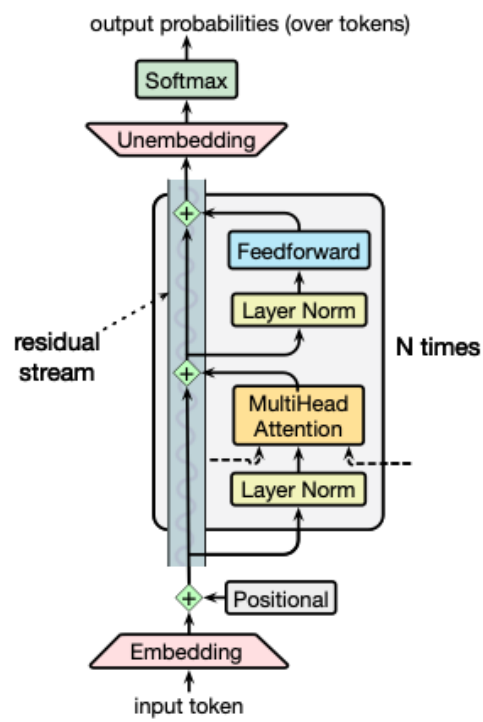


Abbildung 2.2: Transformer-Architektur  
Quelle: [5, S.173]

## 2.6 LLM-Limitationen

Ein großes Problem von LLMs ist, dass sie Halluzinationen erzeugen können. Halluzinationen sind Fakten, die zwar plausibel klingen, aber nicht mit den Fakten der realen Welt übereinstimmen. Dies passiert, weil LLMs zwar darauf trainiert sind, schlüssige und zusammenhängende Texte zu generieren, aber es gibt keine Mechanismen, die dafür sorgen, dass der generierte Text richtig ist oder der Wahrheit entspricht [5, S. 167, S. 236].

Außerdem haben LLMs einen begrenzten Wissenshorizont und können keine proprietären oder privaten Fragen beantworten oder Fragen, die nicht Teil ihres Trainings waren, da sie nur auf vortrainiertes Wissen zurückgreifen können, das Bestandteil von ihren Trainingsdaten ist [5, S. 236].

Ein weiteres Problem ist, dass LLMs statisch sind, sie wurden nur einmal zu einem bestimmten Zeitpunkt trainiert und können keine Aussagen über Informationen treffen, die nach ihrem Training stattgefunden haben [5, S. 236].

Ein Lösungsansatz, um diese Probleme zu beheben, besteht darin, dem LLM mehr Kontext zur Verfügung zu stellen, indem beispielsweise Daten aus privaten Dokumenten übergeben werden. Diese Methode, LLMs zu erweitern, nennt man Retrieval-Augmented Generation (RAG) [5, S. 236].

## 2.7 Embeddings

Unter Embeddings versteht man Vektorrepräsentationen von Wörtern, Sätzen oder Texten, die die semantische Bedeutung dieser darstellen. Die semantische Bedeutung beschreibt den inhaltlichen Sinn eines Begriffes und seine Beziehungen zu anderen Begriffen. [5, S. 96] Ähnliche Wörter stehen hier im Vektorraum nah beieinander [8].

In Abbildung 2.3 sieht man, wie die semantischen Bedeutungen der Wörter als Embeddings im Vektorraum bestehen bleiben. Auf der linken Seite kann man sehen, wie die semantische Beziehung zwischen männlichen und weiblichen Wörtern dargestellt wird und dass die Distanzen zwischen dem männlichen und dem weiblichen Wort immer ähnlich sind. Auf der rechten Seite von Abbildung 2.3 sieht man, dass zusätzlich auch die Beziehung zu den jeweiligen Pluralformen der Vektorrepräsentationen dargestellt werden kann.

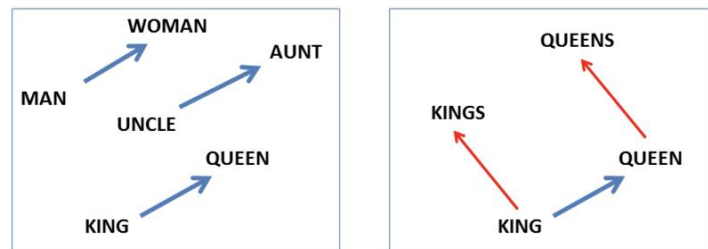


Abbildung 2.3: Beziehung zwischen Embeddeten Wörtern  
Quelle: [9]

Dadurch, dass ähnliche Wörter nah beieinanderliegen, kann man die Ähnlichkeit von zwei Wort-Embeddings messen. Dies kann man mit verschiedenen Methoden machen. Eine häufige Methode ist die Kosinusähnlichkeit, mit der man messen kann, ob zwei Embeddings in eine ähnliche Richtung zeigen. In Formel (2.1) sieht man, wie die Kosinusähnlichkeit berechnet wird. Die Ergebnisse liegen typischerweise zwischen -1 und 1, und je näher das Ergebnis an eins liegt, desto ähnlicher sind sich die zwei Embeddings.

$$\text{cosine}(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{\|\mathbf{v}\| \cdot \|\mathbf{w}\|} \quad (2.1)$$

Es gibt verschiedene Arten von Embeddings: statische Embeddings, die für jedes Wort einen festen Vektor gelernt haben, und kontextbezogene Embeddings, die abhängig vom Kontext, das bedeutet, von den umgebenden Wörtern, eine Vektorrepräsentation erstellen.[5, S. 105-106]

### 2.7.1 Word2Vec

Ein Beispiel für ein statisches Embedding wäre das mit der Arbeit „Efficient Estimation of Word Representations in Vector Space“ vorgestellte Word2Vec[8]. Word2Vec hat für jedes Wort eine feste Vektorrepräsentation gelernt, die unabhängig vom Kontext ist, in dem das Wort im Trainingsdatensatz stand.

Wichtig ist zu erwähnen, dass nur beim Lernprozess die Kontexte der Wörter wichtig sind und beim Embedden von Wörtern der Kontext nicht mehr betrachtet wird. Dadurch entsteht auch der Nachteil von Word2Vec, dass ein Wort unabhängig vom Kontext und auch unabhängig von Mehrdeutigkeit die gleiche Vektorrepräsentation hat. Es gibt nur ein Embedding für das Wort „Bank“, egal, ob es sich um das Geldinstitut oder um eine Sitzbank handelt.

### 2.7.2 Bidirectional Encoder Representations from Transformers

„Bidirectional Encoder Representations from Transformers“, kurz BERT, ist ein kontextbezogenes Embedding. Das bedeutet, dass das gleiche Wort je nach Kontext eine andere Vektorrepräsentation erhält. So kann BERT im Gegensatz zu Word2Vec mehrdeutige Wörter besser unterscheiden. „Bidirectional“ bezieht sich hier darauf, dass sowohl Wörter, die vor dem Wort stehen, als auch Wörter, die nach dem Wort stehen, berücksichtigt werden, um eine Vektorrepräsentation zu erstellen.

BERT verwendet die Transformer-Architektur und damit auch den Self-Attention-Mechanismus und kann dadurch Beziehungen zwischen Wörtern erkennen.

BERT wurde mit großen Textmengen trainiert, in denen einige Wörter verdeckt wurden. Diese versucht das Modell beim Training vorherzusagen. So lernt das Modell sprachliche Zusammenhänge und Bedeutungen von Wörtern in unterschiedlichen Kontexten[10].

Sentence-BERT, kurz SBERT, ist eine Erweiterung von BERT, in der ganze Sätze einen Vektor als Repräsentation zugeteilt bekommen. So kann ein ganzer Satz als nur eine Vektorrepräsentation dargestellt werden. Das ermöglicht, dass zwei Sätze oder Textpassagen direkt miteinander verglichen werden können[11].

## 2.8 Vektordatenbanken

Vektordatenbanken sind spezielle Datenbanken, die dafür entwickelt wurden, Embeddings zu speichern und zu durchsuchen. Es können zusätzlich zu den Embeddings auch noch die zugehörigen Texte und weitere Daten gespeichert werden.

Bei Suchanfragen werden die Anfragen ebenfalls in ein Embedding umgewandelt, so kann die Vektordatenbank die Anfrage mit den gespeicherten Embeddings vergleichen und Textstellen finden, die der Anfrage am ähnlichsten sind.

Eine Anfrage mit allen Embeddings zu vergleichen, die in der Vektordatenbank gespeichert sind, würde sehr lange dauern und mit dem Wachstum der Vektordatenbank immer mehr Zeit in Anspruch nehmen. Das Approximate-Nearest-Neighbor-Verfahren (ANN) soll dieses Problem lösen, indem die Embeddings mithilfe von Indexstrukturen organisiert werden. So muss nicht jedes Embedding mit der Anfrage verglichen werden, sondern nur ein vielversprechender Bereich. Bei dem Verfahren kann es jedoch passieren, dass nicht das ähnlichste Embedding zu der Anfrage gefunden wird. Es wird also Ungenauigkeit in Kauf genommen, um die Geschwindigkeit zu erhöhen.

Ein spezielles ANN-Verfahren ist das Hierarchical-Navigable-Small-World-Verfahren, kurz HNSW. Beim HNSW werden Embeddings in verschiedenen Schichten gespeichert. Die Embeddings sind als Knoten in einem Graphen verbunden. In den hohen Schichten liegen wenige Embeddings und umso tiefer eine Schicht ist, umso mehr Embeddings liegen in dieser Schicht. In Abbildung 2.4 sieht man den Aufbau eines HNSW-Graphen. Die Anfrage startet in einer hohen Schicht an einem Einstiegspunkt. Von dort springt sie schrittweise zu einem benachbarten Embedding mit höherer Ähnlichkeit. Wenn es keine benachbarten Embeddings mit höherer Ähnlichkeit mehr gibt, springt die Anfrage eine Schicht tiefer und wiederholt diesen Prozess, bis auf der tiefsten Ebene ein Embedding mit hoher Ähnlichkeit gefunden wurde[12].

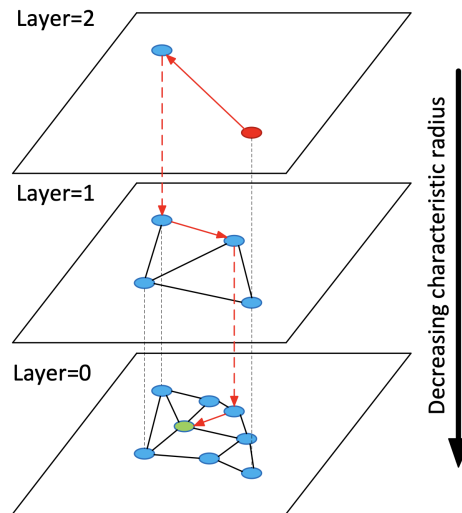


Abbildung 2.4: Hierarchical Navigable Small World Verfahren

Quelle: [12]

## 2.9 Information Retrieval

Information Retrieval beschreibt, wie Daten aus Datensätzen wie Vektordatenbanken gefunden werden. Dabei werden gespeicherte Inhalte mit der Suchanfrage verglichen und nach Relevanz sortiert[13]. Es gibt verschiedene Strategien des Information Retrievals.

BM25 ist beispielsweise ein klassisches Information-Retrieval-Verfahren, bei dem ausschließlich auf die Begriffe in der Suchanfrage geschaut und mit den Begriffen im Datensatz verglichen wird. Kommen viele Begriffe aus der Anfrage in einem Dokument im Datensatz vor, bekommt dieses Dokument einen hohen BM25-Score. Die Seltenheit der Begriffe im kompletten Datensatz spielt für den BM25-Score auch eine wichtige Rolle. Begriffe, die selten im Datensatz vorkommen, werden höher gewichtet. Weil BM25 ein Verfahren ist, das nur lexikalisch vergleicht, kann es schwerer Dokumente finden, die semantisch das Gleiche meinen wie die Suchanfrage, aber andere Begriffe verwenden [14].

Dieses Problem kann teilweise durch Dense Retrieval gelöst werden. Bei Dense Retrieval werden Suchanfragen und Datensatz mithilfe eines Embedding-Modells embeddet und im Vektorraum des Datensatzes nach Dokumenten gesucht, die der Anfrage am ähnlichsten sind. ANN-Verfahren wie HNSW werden hier eingesetzt, um die Suchzeit zu verkürzen.

## 2.10 Retrieval-Augmented Generation

Wie bereits in vorherigen Kapiteln erläutert, handelt es sich bei Retrieval-Augmented Generation (RAG) um eine Methode, bei der LLMs mit externen Daten erweitert werden.

Eine Anfrage wird durch Informationen aus externen Daten während der Laufzeit ergänzt. Es können relevante Dokumente oder Textstellen direkt aus den Daten retrieved werden, um dem LLM mehr Kontext zu einer Frage bereitstellen zu können.

Dadurch können Limitierungen von LLMs wie Halluzinationen, veraltete Daten und der begrenzte Wissenshorizont reduziert werden. RAG-Systeme können auch dazu beitragen, die Transparenz und Rückverfolgbarkeit zu erhöhen. Da bei RAG auf Daten aus externen Quellen zugegriffen wird, können diese auch angegeben werden.

### 2.10.1 Architektur

Grundlegend bestehen RAG-Systeme aus drei Phasen: der Indexierungsphase, der Retrievalphase und der Generierungsphase.

In der Indexierungsphase werden Daten aus Dateiformaten wie PDF, HTML, Word oder Markdown extrahiert, in Text umgewandelt und, wenn nötig, in kleinere, verarbeitbare Teile unterteilt. Das Verfahren, den Text in kleinere Teile zu unterteilen, wird als Chunking bezeichnet. Nachdem die Daten in Text umgewandelt wurden und, falls nötig, in Chunks unterteilt wurden, werden sie zu Vektoren embeddet und in einer Vektordatenbank gespeichert.

Bei einer Anfrage an das RAG-System wird während der Retrievalphase in der Vektordatenbank nach Chunks gesucht, die eine hohe semantische Ähnlichkeit zu der Anfrage haben.

Im letzten Schritt, der Generierungsphase, werden die gefundenen Chunks mit hoher semantischer Ähnlichkeit gesammelt und mit der ursprünglichen Anfrage zu einem Prompt kombiniert. Das LLM nutzt die gesammelten Chunks, die hier als zusätzlicher Kontext betrachtet werden, und kann so potenziell präzisere Antworten generieren [15].

Auf Abbildung 2.5 wird eine typische RAG-Architektur abgebildet. Die Abbildung wird von oben nach unten betrachtet, der User stellt eine Frage an das System, in der Retrieval-Phase werden Dokumente gefunden, die eine semantische Ähnlichkeit zu der Frage haben und potenziell die Frage beantworten könnten. Diese Dokumente werden dann mit der Frage zu einem Prompt kombiniert und an das LLM übergeben, das aus diesem Prompt dann versucht, eine Antwort zu generieren, die die Frage des Users beantwortet.

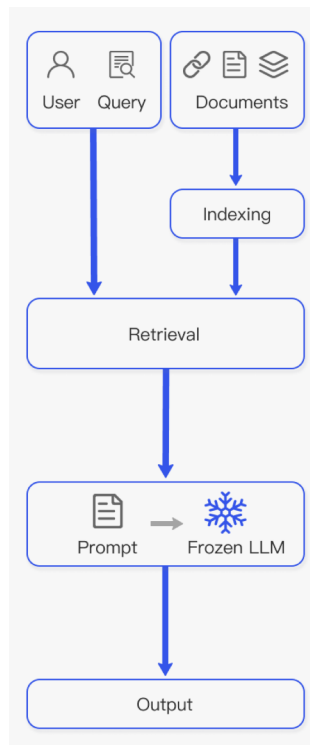


Abbildung 2.5: RAG-Architektur  
Quelle:[15]

### 2.10.2 Warum ist präzises Retrieval wichtig?

Da sich die Antwort des LLMs im Generierungsschritt einer RAG-Pipeline auf den zusätzlichen Kontext bezieht, der in der Retrieval-Phase gefunden wurde, ist die Qualität des gefundenen Kontextes sehr wichtig. Wenn nur unpassende oder nicht relevante Dokumente gefunden wurden, muss das LLM ohne Informationen eine Antwort generieren, wie im Kapitel 2.6 beschrieben, müsste das LLM sich auf seinen begrenzten Wissenshorizont beziehen [16].

Der Retriever kann zu einer Frage auch semantisch ähnliche Dokumente finden, die jedoch für die Beantwortung einer Frage irrelevant sind. Diese Dokumente sind besonders schädlich für die LLM-Generation und können die Qualität der Antwort verringern, indem sie beispielsweise Halluzinationen provozieren[17].

Auch die Position relevanter Dokumente innerhalb des Prompts beziehungsweise des zusammengeführten Kontexts ist wichtig. LLMs neigen dazu, Informationen, die sich in der Mitte eines längeren Textes befinden, weniger stark zu gewichten. Gleichzeitig leidet die Antwortqualität auch, wenn zu viel Kontext beachtet wird [18].

Zusammenfassend lässt sich sagen, dass schlechtes Retrieval die Qualität der Antwort beeinträchtigen kann. Semantisch ähnliche, aber für die Frage irrelevante Dokumente können das LLM ablenken und zu falschen Aussagen führen. Werden relevante Dokumente gar nicht abgerufen, kann die generierte Antwort unvollständig oder halluziniert sein. Zu viel Kontext kann dazu führen, dass relevante Dokumente weniger stark berücksichtigt werden.

Daraus lässt sich schließen, dass beim Retrieval nur relevante Dokumente ausgewählt werden sollten, die für die Beantwortung der Frage tatsächlich erforderlich sind. Dabei sollte nicht wahllos eine große Menge an Dokumenten abgerufen und als Kontext in den Prompt aufgenommen werden. Für eine gute Antwortqualität wird ein möglichst präziser und begrenzter Kontext benötigt.

## 3 Verwandte Arbeiten

Die Retrieval-Qualität ist entscheidend für die Antwortqualität einer RAG-Pipeline. Verschiedene Komponenten der Indexierungs- und Retrieval-Phase beeinflussen diese Qualität. Dieser Abschnitt fasst den aktuellen Forschungsstand zusammen und untersucht, wie verschiedene Komponenten der Indexierungs- und Retrieval-Phase die Retrieval-Qualität beeinflussen. In der Indexierungsphase haben beispielsweise die Auswahl des Embedding-Modells und der Chunking-Strategie, also die Art, wie ein Text unterteilt wird, einen wesentlichen Einfluss auf das Retrieval. Außerdem beeinflusst auch die gewählte Retrieval-Strategie in der Retrieval-Phase die Qualität der abgerufenen Informationen.

### 3.1 Einfluss der Dokumentensegmentierung

#### 3.1.1 Bedeutung der Segmentierung für das Retrieval

Die Chunking-Strategie beeinflusst, in welcher Form die Dokumente in der Vektordatenbank liegen und abgerufen werden können. Dabei spielt es eine große Rolle, wie und vor allem, wo große Texte unterteilt werden. Zu grobe Unterteilung in große Chunks kann dazu führen, dass relevante Textstellen für die Frage in umfangreichen irrelevanten Textstellen eingebettet sind, dadurch kann die Qualität der LLM-Generation sinken wie in Kapitel 2.10.2 beschrieben. Kleine Chunks können dazu führen, dass Informationen präzise abgerufen werden können, aber zu kleine Chunks können notwendigen Kontext verlieren, beispielsweise, wenn sich Informationen über einen längeren Textabschnitt erstrecken [19].

#### 3.1.2 Chunk-Größe

Aktuelle Forschungsarbeiten zur Wahl geeigneter Chunk-Größen zeigen, dass keine universell optimale Chunk-Größe existiert. Bhat et al. untersuchen den Einfluss unterschiedlicher Chunk-Größen auf die Retrieval-Qualität und zeigen, dass deren Eignung unter anderem von dem Datensatz und der Komplexität der Inhalte abhängt. Kleine Chunks mit 64-128 Token eignen sich gut für kurze, faktenbasierte Antworten, und große Chunks mit 512-1024 Token sind vorteilhaft, wenn viel Kontext gebraucht wird, um eine Antwort zu generieren[19].

Zhong et al. greifen die Problematik zu großer beziehungsweise zu kleiner Chunks auf und schlagen als Lösung vor, die Auswahl der Chunk-Größe dynamisch an die jeweilige Anfrage zu koppeln [20].

### 3.1.3 Überlappung benachbarter Chunks

Neben der Chunk-Größe spielt auch die Überlappung der Chunks eine Rolle. Überlappungen verhindern, dass an den Chunkgrenzen Informationen verloren gehen, führen aber dazu, dass identische Textstellen mehrfach in der Vektordatenbank gespeichert werden[21].

Inwiefern die Überlappung von Chunkgrenzen die Retrieval-Qualität tatsächlich positiv beeinflusst, ist in aktueller Forschung umstritten. Bennani und Moslonka untersuchen in ihrer Arbeit verschiedene Chunking-Strategien und unter anderem auch, wie Chunking-Überlappungen sich auf die Qualität der RAG-Pipeline auswirken und kommen zu dem Ergebnis, dass Chunking-Überlappung keinen messbaren Vorteil bietet, jedoch die Kosten der Indexierungsphase erhöhen [22].

Auch Amiri und Bocklitz kommen zu ähnlichen Ergebnissen, stellen jedoch bei einem Overlap von 60% eine leichte Verbesserung der Trefferquote fest. Sie stellen aber auch fest, dass die Konfiguration ohne Chunk-Überlappung in ihrem Versuchsaufbau den besten Kompromiss aus Retrieval-Qualität und Kosten der Indexierungsphase bietet [23].

### 3.1.4 Struktur- und semantische Segmentierung

Um die Dokumentenstruktur zu erhalten und logisch zusammenhängende Textpassagen zusammenzulassen, werden Chunking-Strategien verwendet, in denen der Text hierarchisch, beispielsweise an bestimmten Trennzeichen wie an Absätzen, Zeilenumbrüchen oder Sätzen, unterteilt wird.

Taiwo und Yusoff untersuchen das Verhalten von Chunking auf die Qualität einer RAG-Pipeline. Dabei beziehen sie sich auf Dokumente aus der Öl- und Gasindustrie. Sie kommen zu dem Ergebnis, dass strukturorientiertes Chunking die beste Qualität liefert, gefolgt von rekursivem Chunking. Die Struktur des Textes zu erhalten, hat also einen Qualitätsvorteil geschaffen [24].

Taiwo und Yusoff haben auch untersucht, ob eine Chunking-Strategie, die die semantische Distanz zwischen Textteilen mithilfe eines Embedders erkennt, effektiv ist. In ihrer Untersuchung schnitt diese Chunking-Strategie jedoch relativ schlecht ab[24].

Qu et al. haben in ihren Untersuchungen festgestellt, dass semantische Chunker vor allem bei künstlich zusammengesetzten Texten mit starken Themenwechseln besser abschneiden.

Das ausgewählte Embedding-Modell hat außerdem einen großen Einfluss darauf, wie gut semantisches Chunking tatsächlich funktioniert [25].

## 3.2 Embedding

Zusätzlich zur Segmentierung von Dokumenten stellt das Embedding-Modell eine zentrale Komponente der Indexierungsphase dar. Embedding-Modelle unterscheiden sich in ihrer Architektur, ihren Trainingsdaten und ihren Optimierungszielen. Diese Faktoren bestimmen, in welche Vektorrepräsentation die Chunks überführt werden, welche semantischen Beziehungen abgebildet werden und auch, wie diese abgerufen werden können [26].

Wie gut Embedding-Modelle performen, wurde im Paper *Massive Multilingual Text Embedding Benchmark* verglichen. Der Benchmark enthält mehr als 500 Aufgaben und über 250 Sprachen. Der Vergleich zeigt, dass die Qualität der Embeddings von der Sprache und der Aufgabe abhängt, die ein Embedding-Modell lösen muss. *GritLM-7B*, ein auf *Mistral 7B* aufbauendes Modell, belegt im MTEB-Europe-Benchmark den ersten Platz, gefolgt von *multilingual-e5-large-instruct*, das über deutlich weniger Parameter als *GritLM-7B* verfügt [26][27][28].

## 3.3 Retrieval-Verfahren

Aktuelle Forschungen zeigen, dass BM25 zwar nur lexikalisch vergleicht, aber dadurch vor allem besser Informationen abrufen kann, in denen präzise Begriffe verwendet werden[29].

Karpukhin et al. haben außerdem festgestellt, dass Informationen mit BM25 vor allem dann gut abgerufen werden können, wenn die Frage lexikalisch stark mit dem Inhalt der Dokumenten übereinstimmt. Auch sie kommen zu dem Entschluss, dass, wenn seltene Begriffe in der Anfrage stehen, die in den Dokumenten wiedergefunden werden können, die Qualität durch BM25-Retrieval besser sein kann als durch Dense-Verfahren. Jedoch übertraf der von Karpukhin et al. untersuchte Dense Retriever den BM25-Retriever in den meisten anderen Fällen[30].

Wie schon im Kapitel 2.9 beschrieben, übertrifft Dense Retrieval einfaches BM25-Retrieval vor allem, wenn Anfragen nicht die gleichen Wörter benutzen, die in den Dokumenten verwendet werden, auch wenn sie semantisch das Gleiche meinen.

Hybrides Retrieval kombiniert BM25 und Dense Retrieval und vereint ihre Stärken. Dies kann laut Akarsu et al. vor allem bei heterogenen Dokumenten, die aus Fließtexten und präzisen Begriffen, beispielsweise in Form von Tabellen, bestehen, zu besseren Retrieval-Qualität führen [29]

### **3.4 Abgrenzung der eigenen Arbeit**

Die meisten Arbeiten untersuchen einzelne Komponenten einer RAG-Pipeline. Es werden entweder Chunking-Strategien, Embedding-Modelle oder Retrieval-Verfahren isoliert in einer kontrollierten Umgebung miteinander verglichen. Die meisten Arbeiten zeigen, dass es von den verwendeten Dokumenten abhängt, welche Komponenten die beste Leistung erbringen. Ein Vergleich mehrerer Komponenten innerhalb eines lokal betriebenen RAG-Systems liegt bislang auch nicht vor. Es fehlt eine Untersuchung, welche Kombination der Komponenten unter Berücksichtigung von Antwortqualität und Ressourcenverbrauch die beste Gesamtleistung in einem Anwendungsfall erzielt.

## 4 Methodik

Es gibt verschiedene Komponenten, die die Qualität des Retrievals in einer RAG-Pipeline beeinflussen. Dieser Versuch soll zeigen, wie sich Chunking-Strategien, unterschiedliche Embedding-Modelle und Retrieval-Verfahren auf die Retrieval-Qualität auswirken.

Wie im Kapitel 2.6 beschrieben, sind RAG-Systeme ein guter Ansatz, um einem LLM die Möglichkeit zu geben, proprietäre und private Fragen zu beantworten. Ein großes Problem von Online-Cloud-LLMs ist, dass Daten den proprietären und privaten Raum verlassen und externe Anbieter diese Daten verarbeiten können. Dieser Versuch wird sich aufgrund dieses Problems auf eine lokale RAG-Pipeline begrenzen.

Ziel ist es, die oben genannten Komponenten in diesen Rahmenbedingungen zu evaluieren und eine geeignete Konfiguration für einen beispielhaften Anwendungsfall zu finden. Außerdem soll untersucht werden, welchen Einfluss die Komponenten auf das Retrieval haben. Im Laufe des Versuchs werden Chunking-Strategien, Embedding-Modelle und Retrieval-Verfahren einzeln angepasst und das Verhalten auf die RAG-Pipeline untersucht.

### 4.1 Aufbau der RAG-Pipeline

Die untersuchte RAG-Pipeline besteht aus verschiedenen Komponenten, die verschiedene Aufgaben übernehmen. Haystack wird verwendet, um die Pipeline zu strukturieren und auszuführen, es bietet Schnittstellen für typische RAG-Komponenten. Um Dokumente zu speichern und abzurufen, wird die Vektordatenbank Weaviate verwendet. Das lokale LLM gemma3:4b wird von Ollama bereitgestellt und generiert am Ende der Pipeline die Antwort.

#### 4.1.1 Haystack

Haystack ist ein Open-Source-Orchestrierungs-Framework, entwickelt von deepset, das Komponenten zum Einlesen, Bereinigen, Aufteilen, Einbetten und Abrufen von Dokumenten bereitstellt. Es ermöglicht, fortgeschrittene RAG-Systeme und weitere LLM-basierte Anwendungen zu entwickeln. Haystack stellt zudem auch Schnittstellen für andere RAG-Komponenten bereit.

In diesem Versuch übernimmt Haystack die Steuerung der RAG-Pipeline. Die Dokumente werden mit Haystack eingelesen, aufbereitet und in einer Vektordatenbank gespeichert. Danach nimmt Haystack eine Frage an, kann mit ihr nach ähnlichen Dokumenten suchen, einen kombinierten Prompt aus Frage und gefundenen Dokumenten erstellen und mit diesem Prompt und einem LLM eine Antwort generieren.

Haystack wurde als Framework gewählt, da es Verarbeitungsschritte als Pipeline zusammenführt und einzelne Komponenten in diesen Pipelines einfach ausgetauscht werden können, solange der Output und Input zu den vorherigen und nachfolgenden Komponenten passen. So konnte ein sauberer Versuchsaufbau erreicht werden [31].

#### **4.1.2 Weaviate**

Weaviate ist eine Datenbank, die Vektoren speichern und durchsuchen kann. Haystack bietet mit der WeaviateDocumentStore eine Schnittstelle für Weaviate-Instanzen an. Die Weaviate-Instanz, die in diesem Versuch verwendet wird, läuft auf einem Docker-Container. Für die Instanz wurde die von Haystack als Beispiel angeführte `docker-compose.yml` verwendet, die auf der WeaviateDocumentStore-Seite von Haystack zu finden ist [32] [33].

Weaviate wurde als Vektordatenbank ausgewählt, weil sie lokal betrieben werden kann. Entscheidender Faktor war auch, dass Haystack eine Schnittstelle für Weaviate anbietet. Diese Schnittstelle ermöglicht es außerdem, Weaviate als Dense-, BM25- und hybriden Retriever zu verwenden, was entscheidend für den Versuchsaufbau.

#### **4.1.3 LLM**

Das LLM ist in diesem Versuchsaufbau dafür zuständig, auf Basis des abgerufenen Kontextes Antworten zu Fragen zu generieren. Für jeden Durchgang wird das LLM `gemma3:4b`, bereitgestellt durch `Ollama`, verwendet.

`gemma3:4b` wird verwendet, da es ein vergleichbar kleines Modell ist, das problemlos lokal betrieben werden kann und mit 39 Millionen Downloads weit verbreitet ist. Außerdem ist es ein multilinguales LLM, was wichtig für diesen Versuch ist, da das verwendete Dokument und die Fragen dazu auf Deutsch sind [34].

#### **4.1.4 Datensatz**

In diesem Versuchsaufbau wird ein beispielhafter Anwendungsfall für eine lokale RAG-Pipeline untersucht. Als Datensatz wird das deutschsprachige PDF-Dokument `Funktionales`

Kostensplitting zur Ermittlung von Mieterhöhungen nach energetischen Maßnahmen – Eine Handlungsempfehlung auf Basis theoretischer und empirischer Untersuchungen verwendet [35]. Das Dokument behandelt Mieterhöhungen nach energetischen Modernisierungsmaßnahmen und die Frage, wie deren Kosten in mieterhöhungsrelevante und nicht mieterhöhungsrelevante aufgeteilt werden können.

Das Dokument ist in einem Fließtext geschrieben und bietet sich für diesen Versuch an, da es Fachinformationen enthält und überprüft werden kann, ob diese Fachinformationen korrekt von der RAG-Pipeline gefunden werden.

Im Rahmen der Forschung des Verbundprojektes Intelmod wurde für dieses Dokument ein Golden Set erstellt, das 21 Fragen mit Referenzantwort und dem Kontext aufführt, dass zur Beantwortung der Fragen benötigt wird.

#### **4.1.5 Versuchsaufbau**

Für den Versuch wurde eine lokale RAG-Pipeline auf Basis von Haystack entwickelt, die aus den Phasen Indexierungsphase, Retrievalphase und der Generierungsphase besteht. In der Indexierungsphase wird das untersuchte PDF-Dokument mithilfe von PyPDFToDocument eingelesen und in ein Dokumentenformat übertragen, das Haystack verwenden kann. Danach wird der eingelesene Text mit dem DocumentCleaner von Haystack bereinigt. Der DocumentCleaner macht das Dokumentenformat lesbarer, indem er leere Zeilen und aufeinanderfolgende Leerzeichen entfernt. Danach wird der Text von der eingestellten Chunking-Strategie in kleinere Teile unterteilt und Metainformationen entfernt, weil sie nicht benötigt werden. Am Ende werden die Chunks mit dem SentenceTransformersDocumentEmbedder und dem eingestellten Embedding-Modell in Vektorrepräsentationen überführt und der DocumentWriter schreibt die Daten schließlich in die WeaviateDocumentStore-Vektordatenbank.

In der Retrievalphase wird unterschieden, welche Retrieval-Strategie ausgewählt ist. Wenn der BM25-Retriever ausgewählt wurde, werden direkt ähnliche Dokumente zu der Frage gesucht. Wenn aber der hybride oder der Dense-Retriever ausgewählt wurde, wird die Frage erst mal mit dem SentenceTransformersTextEmbedder und dem gleichen ausgewählten Embedding-Modell in eine Vektordarstellung gebracht, das auch für die Dokumente verwendet wurde, und anschließend werden mit der Vektordarstellung semantisch ähnliche Dokumente in der Vektordatenbank gesucht.

In der Generierungsphase werden die gefundenen Dokumente dann mithilfe von dem ChatPromptBuilder mit der ursprünglichen Frage zu einem Prompt zusammengeführt. Schließlich wird dieser Prompt dem OllamaChatGenerator übergeben, der mit dem Ollama-Modell gemma3:4b eine Antwort generiert.

Der BM25-, Dense- und der hybride Retriever nehmen alle ein  $\text{top}_k$  entgegen, das definiert, wie viele Dokumente der Retriever zurückgibt, in diesem Versuch wird dieser Wert auf  $\text{top}_k = 3$  fixiert. Der hybride Retriever nimmt zusätzlich ein  $\alpha$  entgegen, das bestimmt, ob der Retriever die Ergebnisse des BM25-Retrievers oder des Dense-Retrievers bevorzugt, ein Wert von null entspricht dem BM25-Retriever und ein Wert von eins dem Dense-Retriever. Da im Versuch mit dem hybriden Retriever untersucht werden soll, welchen Einfluss hybrides Abrufen hat, wird dieses auf  $\alpha = 0,5$  gesetzt. Zusätzlich sollen vor allem Komponenten untersucht werden, die Einfluss auf die Retrieval-Qualität haben, deswegen wird für den ganzen Versuch das von Ollama bereitgestellte LLM `gemma3:4b` verwendet.

## 4.2 Untersuchte Systemkomponenten

In diesem Versuch werden Komponenten untersucht, die einen Einfluss auf die Retrieval-Qualität haben können, dabei wird das Chunking-Verfahren, der Embedder, der die Chunks als Vektorrepräsentation in die Vektordatenbank speichert, und der Einfluss von Dense-, BM25- und hybridem Retrieval untersucht.

### 4.2.1 Chunking-Strategie

Beim Chunking wird untersucht, welchen Einfluss die Chunking-Strategie in dem untersuchten Anwendungsfall auf die Retrieval-Qualität hat. Es wird beispielhaft die Größe der Chunks und der Überlappung untersucht. Es werden außerdem Chunk-Grenzen wie Wörter oder Sätze untersucht. Zusätzlich wird untersucht, ob ein rekursives Chunken hilft, die Dokumentenstruktur zu erhalten und die Qualität der RAG-Pipeline zu erhöhen. Außerdem wird ein Chunker untersucht, der nach semantischen Grenzen chunkt.

Es werden die in Abbildung 4.1 aufgelisteten Chunking-Strategien untersucht. Das Versuchsziel sagt aus, was mit der jeweiligen Chunking-Strategie untersucht werden soll. Das Verfahren beschreibt, wie die einzelnen Chunks getrennt werden.

#### DocumentSplitter

Für die wortbasierten und satzbasierten Verfahren wurde der von Haystack bereitgestellte DocumentSplitter verwendet, der das Dokumentenformat von Haystack unterstützt. Bei dem DocumentSplitter kann eingestellt werden, ob die Chunks nach Wörtern, Sätzen, Paragraphen, Seiten oder anderes getrennt werden sollen. Außerdem kann bei dem DocumentSplitter ein Wert für die Chunk-Größe und für die Überschneidung der Chunks eingestellt werden[36].

Tabelle 4.1: Untersuchte Chunking-Strategien

| Versuchsziel                  | Verfahren        | Chunkgröße          | Überschneidung |
|-------------------------------|------------------|---------------------|----------------|
| Chunkgröße                    | wortbasiert      | 30 Wörter           | 0 Wörter       |
| Chunkgröße                    | wortbasiert      | 100 Wörter          | 0 Wörter       |
| Chunkgröße                    | wortbasiert      | 300 Wörter          | 0 Wörter       |
| Überlappung                   | wortbasiert      | 300 Wörter          | 60 Wörter      |
| Überlappung                   | wortbasiert      | 300 Wörter          | 180 Wörter     |
| Satzgrenzen                   | satzbasiert      | 5 Sätze             | 0 Sätze        |
| Satzgrenzen                   | satzbasiert      | 10 Sätze            | 0 Sätze        |
| Dokumentstruktur              | rekursiv         | maximal 300 Wörter  | 0 Wörter       |
| Einfluss semantischer Grenzen | embeddingbasiert | 50 bis 1000 Zeichen | 0 Wörter       |

### RecursiveDocumentSplitter

Für das rekursive Chunking wird der von Haystack bereitgestellte `RecursiveDocumentSplitter` [37] verwendet. Die maximale Chunkgröße wird über den Parameter `split_length` auf 300 Wörter festgelegt. Der `RecursiveDocumentSplitter` erwartet eine Liste von Trennzeichen, die er anschließend nacheinander abarbeitet. In diesem Versuchsaufbau wurde dem `RecursiveDocumentSplitter` folgende Liste zugewiesen:

```
["\n\n", "sentence", "\n", " "]
```

Der Text wird entsprechend der Reihenfolge der Elemente in dieser Liste unterteilt. Als erstes versucht der `RecursiveDocumentSplitter`, den Text an den Stellen zu trennen, an denen ein doppelter Zeilenumbruch (`\n\n`) vorkommt. Diese doppelten Zeilenumbrüche kennzeichnen im Haystack-Dokumentenformat die Grenzen zwischen Absätzen.

Überschreitet ein dadurch entstandener Chunk weiterhin die festgelegte Länge von 300 Wörtern, wird dieser Chunk mithilfe des nächsten Elements in der Liste weiter unterteilt. In diesem Fall ist das nächste Element `sentence`. Der Chunk wird dadurch an Satzgrenzen aufgeteilt.

Sollte ein Chunk danach immer noch zu groß sein, werden als nächste Trennzeichen Zeilenumbrüche (`\n`) und danach Leerzeichen verwendet. Dadurch wird der Text schrittweise in immer kleinere Einheiten unterteilt.

Vom rekursiven Chunking erhofft man sich, die Dokumentenstruktur in den einzelnen Chunks möglichst gut zu erhalten.

## EmbeddingBasedDocumentSplitter

Für das embeddingbasierte Chunking-Verfahren wird der `EmbeddingBasedDocumentSplitter` [38] verwendet, der durch Haystack bereitgestellt wird. Der `EmbeddingBasedDocumentSplitter` unterteilt Chunks nicht in feste Größen, sondern anhand semantischer Unterschiede.

Dabei werden Dokumente in Sätze unterteilt. In diesem Versuch ist `sentences_per_group=3` eingestellt. Das Dokument wird also in Gruppen eingeteilt, die jeweils drei Sätze beinhalten. Danach wird die Kosinusähnlichkeit zwischen zwei benachbarten Gruppen gemessen. Eine hohe Kosinusähnlichkeit deutet darauf hin, dass die zwei Gruppen das gleiche Thema behandeln. Mit dem Parameter `percentile` kann man zusätzlich einstellen, wie empfindlich der `EmbeddingBasedDocumentSplitter` auf semantische Unterschiede reagieren soll. In diesem Versuch ist `percentile=0.95` eingestellt, das bedeutet, dass immer dann unterteilt wird, wenn die Kosinusdistanz zwischen zwei benachbarten Gruppen aus drei Sätzen höher ist als 95 Prozent aller gemessenen Kosinusdistanzen in dem Dokument. Außerdem ist eine Mindestlänge von 50 Zeichen und eine maximale Länge von 1000 Zeichen eingestellt. Vom semantischen Chunken erhofft man sich, dass thematisch zusammenhängende Textpassagen zusammen in einem Chunk bleiben und so zusammen abgerufen werden können.

### 4.2.2 Embedding-Modelle

In diesem Versuch werden verschiedene Embedding-Modelle miteinander verglichen: der `intfloat/multilingual-e5-base-Embedder` [28], der `BAAI/bge-m3-Embedder` [39] und der `sentence-transformers/paraphrase-multilingual-mpnet-base-v2-Embedder` [40].

Alle ausgewählten Embedding-Modelle sind lokal über den `SentenceTransformersDocumentEmbedder` [41] beziehungsweise den `SentenceTransformersTextEmbedder` [42], die von Haystack bereitgestellt werden, ansprechbar und unterstützen die deutsche Sprache.

Die Modelle `intfloat/multilingual-e5-base` und `BAAI/bge-m3` wurden unterschiedlich trainiert und für verschiedene Einsatzbereiche entwickelt. `multilingual-e5-base` ist hauptsächlich für Dense Retrieval gedacht. `BAAI/bge-m3` unterstützt dagegen mehrere Retrieval-Verfahren. Durch die unterschiedlichen Trainingsmethoden stellen beide Embedding-Modelle Texte im Vektorraum unterschiedlich dar. Es wird untersucht, ob sich dieser Unterschied auf die Retrieval-Qualität auswirkt.

Das Modell `sentence-transformers/paraphrase-multilingual-mpnet-base-v2` wird verglichen, weil es vor allem darauf trainiert ist, Sätze mit ähnlicher Bedeutung zu erkennen. So kann untersucht werden, ob dieses Modell beim Retrieval andere Ergebnisse liefert als die speziell für Retrieval trainierten Modelle `multilingual-e5-base` und `BAAI/bge-m3`.

### 4.2.3 Retrieval-Verfahren

In diesem Versuch werden drei Retrieval-Verfahren miteinander verglichen: Dense-, BM25- und Hybrid-Retrieval.

Die Funktionsweisen von BM25 und Dense Retrieval wurden bereits im Grundlagenkapitel 2.9 beschrieben. Beim Hybrid Retrieval werden die Ergebnisse von BM25 und Dense Retrieval miteinander kombiniert. Dadurch werden sowohl direkte Übereinstimmungen von Begriffen als auch semantische Ähnlichkeiten berücksichtigt. Wie bereits erwähnt, bietet Haystack mit Weaviate für alle drei Retrieval-Verfahren eine Schnittstelle an.

Der WeaviateEmbeddingRetriever wird verwendet, um aus der Weaviate-Vektordatenbank mit Dense Retrieval Dokumente zu finden. Dafür muss erst mal die Frage mit dem SentenceTransformerTextEmbedder embeddet werden, damit sie beim Dense Retrieval mit den Dokumenten in der Vektordatenbank verglichen werden kann [43]. Für das BM25-Retrieval wird der WeaviateBM25Retriever verwendet, dieser kann mit der Frage direkt die Vektordatenbank durchsuchen und ähnliche Dokumente nach dem BM25-Verfahren finden [44].

Der WeaviateHybridRetriever vereint beide Retriever-Verfahren. Wie bereits erwähnt, wird ihm zusätzlich ein alpha übergeben, das bestimmt, welchem Verfahren mehr Relevanz übergeben wird. Dieses alpha wird im Versuchsaufbau auf  $\alpha = 0.5$  gesetzt und somit sind BM25- und Dense-Verfahren gleich relevant im hybriden Test [45].

Das top\_k der Weaviate-Retriever ist auf top\_k = 3 gestellt, somit werden immer nur die ähnlichsten drei Dokumente als Suchergebnis zurückgegeben.

Die drei Verfahren werden untersucht, weil sie auf unterschiedliche Weise nach passenden Dokumenten suchen. Durch einen Vergleich soll herausgefunden werden, welches Verfahren für die verwendeten Fragen und das verwendete Dokument am besten funktioniert.

## 4.3 Experimenteller Ablauf

### 4.3.1 Begründung des sequentiellen Ablaufs

Bei diesem Versuch soll eine möglichst gute Kombination aus Chunking-Verfahren, Embedding-Modell und Retrieval-Strategie gefunden werden. Es werden neun Chunking-Verfahren, drei Embedding-Modellen und die drei Retrieval-Strategien untersucht. Daraus entstehen 81 mögliche Kombinationen an möglichen Setups. Eine Konfiguration mit allen Fragen zu testen, dauert circa 2500 Sekunden. Hochgerechnet auf die 81 Kombinationen würde eine komplette Untersuchung ca. 56 h dauern. Beim semantischen Chunks wird von einer längeren Laufzeit ausgegangen, da noch Ähnlichkeitsmaße berechnet werden müssen.

Das Testen jeder dieser 81 Kombinationen mit allen Fragen des Golden-Sets würde also viel Zeit-, Rechen- und Energieaufwand erzeugen. Um diesen Aufwand zu verringern, wird der Versuch in drei Phasen aufgeteilt. Die ersten zwei Phasen sollen eine geeignete Chunking-Konfiguration finden. In der dritten Phase wird diese Chunking-Konfiguration mit allen Embedding-Modellen und Retrieval-Verfahren kombiniert. So können aus ursprünglich 81 Kombinationen neun mögliche Kombinationen gemacht werden. Dabei wird davon ausgegangen, dass die Chunking-Konfiguration, die in den ersten beiden Phasen am besten abschneidet, auch in den nicht getesteten Kombinationen die beste Konfiguration ist.

#### **4.3.2 Phase 1: Vorauswahl der Chunking-Konfigurationen**

In der ersten Phase werden alle Chunking-Strategien gegen fünf fest ausgewählte Fragen aus dem Golden-Set verglichen. Dabei bleibt der Embedder `intfloat/multilingual-e5-base` und das Retrieval-Verfahren `Dense` unverändert, um nur die Auswirkungen der Chunking-Konfiguration zu untersuchen.

Dieser Versuch wird dreimal wiederholt, um Schwankungen zwischen den Testläufen mitzuberechnen. Die drei am besten abschneidenden Chunking-Konfigurationen werden mit in Phase 2 übernommen.

#### **4.3.3 Phase 2: Testen der drei besten Chunking-Konfigurationen**

In der zweiten Phase werden die am besten abschneidenden Chunking-Konfigurationen gegen alle Fragen im Golden-Set verglichen. Die zweite Phase soll zeigen, ob die Qualität der besten Chunking-Konfiguration aus Phase 1 mit dem vollen Golden-Set bestehen bleibt, und außerdem soll die am besten geeignete Chunking-Konfiguration gefunden werden.

Das Embedding-Modell `intfloat/multilingual-e5-base` und `Dense` Retrieval bleiben weiterhin unverändert. Die am besten abschneidende Konfiguration wird mit in die Phase 3 übernommen.

#### **4.3.4 Phase 3: Vergleich der Embedding-Modelle und Retrieval-Verfahren**

In der dritten Phase wird die in den ersten beiden Phasen ermittelte Chunking-Strategie mit den drei Embedding-Modellen und den drei Retrieval-Verfahren kombiniert. Daraus entstehen neun mögliche Kombinationen, die alle anhand des Golden Sets evaluiert werden. Das Ziel dieser Phase ist es, das beste Embedding-Modell in Kombination mit dem besten

Retrieval-Verfahren für die in den ersten beiden Phasen ermittelte Chunking-Strategie zu finden.

## 4.4 Evaluations-Frameworks

### 4.4.1 RAGAS

RAGAS ist ein Framework zur Bewertung von RAG-Systemen. Es stellt verschiedene Metriken bereit, mit denen geprüft werden kann, wie gut ein RAG-System arbeitet. Dabei kann zum Beispiel untersucht werden, ob passende Chunks gefunden wurden, ob wichtige Informationen fehlen und ob die erzeugte Antwort zur Frage passt. Außerdem kann geprüft werden, ob sich die Antwort wirklich auf die gefundenen Inhalte stützt. In diesem Versuch wird RAGAS verwendet, um die Ergebnisse der verschiedenen Retrieval-Konfigurationen miteinander zu vergleichen [46].

### 4.4.2 Bewertungsmetriken

#### Retrival-Metriken

Um die Retrieval-Qualität mit RAGAS zu messen, werden die von RAGAS bereitgestellten Evaluationsmetriken `ContextPrecision`, `ContextRecall`, `NonLLMContextPrecisionWithReference` und `NonLLMContextRecall` verwendet.

`Context Precision` bewertet, wie gut relevante Chunks in den Suchergebnissen angeordnet sind. Es wird geprüft, ob relevante Chunks möglichst weit oben in den Suchergebnissen stehen. Ein hoher Wert bedeutet, dass passende Chunks vor weniger relevanten Chunks gefunden werden [47].

`Context Recall` bewertet, ob die für die Beantwortung benötigten Informationen in den gefundenen Chunks enthalten sind. Dafür wird geprüft, wie viele Aussagen aus der Referenzantwort durch die gefundenen Chunks gestützt werden können. Ein hoher Wert bedeutet, dass viele Aussagen durch den gefundenen Kontext gestützt werden können und nur wenige Informationen fehlen, um eine Antwort zu generieren, die der Referenzantwort nahekommt [48].

`NonLLMContextRecall` prüft, wie viele der vorgegebenen Referenz-Chunks durch das Retrieval gefunden wurden. Dafür werden die gefundenen Chunks direkt mit den Referenz-Chunks verglichen. Diese Metrik verwendet kein LLM. Das Ergebnis liegt zwischen null und eins. Ein

hoher Wert bedeutet, dass viele passende Referenz-Chunks gefunden wurden. Ein niedriger Wert bedeutet, dass viele Referenz-Chunks nicht gefunden wurden [48].

`NonLLMContextPrecisionWithReference` prüft, wie viele der gefundenen Chunks zu den vorgegebenen Referenz-Chunks passen. Dafür wird jeder gefundene Chunk mit allen Referenz-Chunks verglichen. Diese Metrik verwendet ebenfalls kein LLM. Ein hoher Wert bedeutet, dass viele der gefundenen Chunks den Referenz-Chunks ähnlich sind. Ein niedriger Wert bedeutet, dass viele gefundene Chunks nicht zu den Referenz-Chunks passen [47].

## End-to-End-Metriken

Zusätzlich werden End-to-End-Metriken mit aufgezeichnet, um nachzuvollziehen, inwiefern hohe Retrieval-Qualität die Antwortqualität der RAG-Pipeline beeinflusst.

Als End-to-End-Metriken werden `faithfulness`, `contextUtilization`, `noiseSensitivity`, `answerCorrectness` und `answerRelevancy` benutzt.

`faithfulness` bewertet, ob die Antwort durch den gefundenen Kontext gestützt wird. Dafür wird die Antwort in einzelne Aussagen aufgeteilt, danach wird überprüft, welche dieser Aussagen mit dem gefundenen Kontext gestützt sind.

Ein hoher Wert bedeutet, dass alle Aussagen in der Antwort durch den gefundenen Kontext belegt werden können. Ein niedriger Wert bedeutet, dass die Antwort Aussagen enthält, die nicht mit dem gefundenen Kontext gestützt werden können [49].

`contextUtilization` bewertet, ob die gefundenen Chunks für die Antwort nützlich waren. `contextUtilization` überprüft, welche Informationen aus den gefundenen Chunks in der Antwort verwendet wurden [47].

`noiseSensitivity` bewertet, wie stark sich relevante oder nicht relevante gefundene Chunks negativ auf die Antwort auswirken. Es werden falsche Aussagen in der Antwort identifiziert und kontrolliert, ob diese Aussagen durch die gefundenen Chunks verursacht wurden. Im Gegensatz zu den anderen Metriken ist hier ein niedriger Wert besser [50].

`answerCorrectness` bewertet, wie stark die Antwort mit der Referenzantwort übereinstimmt [51].

`answerRelevancy` bewertet, wie gut die Antwort zur gestellten Frage passt. Geprüft wird, ob die Antwort die Frage direkt beantwortet und keine unnötigen oder unpassenden Informationen enthält [52].

Die End-to-End-Metriken sollen zwar hauptsächlich die End-to-End-Qualität der RAG-Pipeline messen, die Metriken `contextUtilization`, `faithfulness` und `noiseSensitivity` wurden jedoch bewusst ausgewählt, um zu überprüfen, wie die RAG-Pipeline mit dem gefundenen Kontext umgeht.

Fast alle verwendeten Metriken nutzen ein LLM, um zu bewerten, ob die jeweiligen Kriterien erfüllt sind. In diesem Versuchsaufbau wird dafür das LLM-Modell `mistral:7b` bereitgestellt durch Ollama verwendet [53].

#### 4.4.3 CodeCarbon

CodeCarbon ist eine Open-Source-Bibliothek, mit der der Energieverbrauch und die geschätzten Emissionen eines Programms aufgezeichnet werden können.

CodeCarbon wurde mit dem bereitgestellten `EmissionsTracker` im Code verwendet. Der Energieverbrauch während der Indexierungsphase und während der Ausführung der RAG-Pipeline wird getrennt aufgezeichnet [54].

Der Energieverbrauch wird gemessen, um neben der Retrieval-Qualität auch beurteilen zu können, wie viel Energie eine Konfigurationen verbraucht. Für den Betrieb einer lokalen RAG-Pipeline ist dies von Bedeutung, weil erhöhter Energieverbrauch die Betriebskosten der RAG-Pipeline erhöhen.

### 4.5 Hardwarekonfiguration

Der Versuchsaufbau wurde auf einem System mit folgender Hardwarekonfiguration durchgeführt:

- **Prozessor:** AMD Ryzen 7 5800X
- **Arbeitsspeicher:** 32 GB DDR4
- **Grafikkarte:** NVIDIA GeForce RTX 2070 SUPER
- **Festplatte:** Samsung SSD 990 PRO 1TB
- **Betriebssystem:** Windows 11 Pro, 64 Bit version: 25H2

## 4.6 Softwareversionen

Folgende Software wurde mit der entsprechenden Version verwendet:

- **Python:** 3.12.10
- **Visual Studio Code:** 1.131.0
- **Haystack:** 2.31.0
- **RAGAS:** 0.4.3
- **CodeCarbon:** 3.2.8
- **Weaviate Client:** 4.22.0
- **Weaviate:** 1.30.17
- **Ollama:** 0.32.5

# 5 Ergebnisse

## 5.1 Metriken

Bei den Ergebnissen hat sich ergeben, dass es deutliche Unterschiede zwischen den LLM-basierten Retrieval-Metriken und den Retrieval-Metriken ohne LLM gab. Die LLM-basierten Retrieval-Metriken Context Precision und Context Recall zeigen, wie in Abbildung 5.1 zu sehen, beinahe für jede Chunking-Konfiguration einen perfekten Score an. Wohingegen die Retrieval-Metriken ohne LLM-Unterstützung für beinahe jede Konfiguration den schlechtesten Score angeben.

| Run / Methode                                   | Context Precision | Context Recall | Gold Context Precision | Gold Context Recall |
|-------------------------------------------------|-------------------|----------------|------------------------|---------------------|
| document_word_300_60_intfloat_multilingual      | 1.0000            | 1.0000         | 0.0000                 | 0.0000              |
| document_word_300_180_intfloat_multilingual     | 1.0000            | 1.0000         | 0.0000                 | 0.0000              |
| recursive_paragraph_first_intfloat_multilingual | 1.0000            | 1.0000         | 0.0000                 | 0.0000              |
| document_word_300_0_intfloat_multilingual       | 1.0000            | 0.8000         | 0.0000                 | 0.0000              |
| document_word_100_0_intfloat_multilingual       | 1.0000            | 1.0000         | 0.0000                 | 0.0000              |
| semantic_p90_intfloat_multilingual-e5-base      | 0.9667            | 1.0000         | 0.0000                 | 0.0000              |
| document_sentence_5_0_intfloat_multilingual     | 1.0000            | 1.0000         | 0.0000                 | 0.0000              |
| document_sentence_10_0_reference_intfloat       | 1.0000            | 0.7200         | 0.2000                 | 0.0667              |
| document_word_30_0_intfloat_multilingual-e      | 1.0000            | 0.9500         | 0.0667                 | 0.0667              |

Abbildung 5.1: Retrieval-Metrik Ergebnisse Phase eins/wiederholung eins

Da die Retrieval-Metriken beinahe ausschließlich Werte an den Extrempunkten ausgeben, kann allein auf ihrer Grundlage kein sinnvoller Vergleich durchgeführt werden. Deswegen werden für den weiteren Versuch außerdem die End-to-End-Metriken berücksichtigt. Die End-to-End-Metriken bewerten die Qualität der erzeugten Antwort, die auf dem gefundenen Kontext basiert. Dadurch kann indirekt untersucht werden, ob eine Konfiguration geeigneten Kontext liefert, mit dem die RAG-Pipeline eine gute Antwort erzeugen kann. Dabei wird angenommen, dass ein besseres Retrieval auch zu einer besseren Antwortqualität führt.

In diesem Versuch wird für jede Konfiguration ein Gesamtscore gebildet, der sich aus allen verwendeten Metriken zusammensetzt. Dafür werden die Werte aller Metriken addiert und anschließend durch die Anzahl der Metriken geteilt. Bei der Metrik Noise Sensitivity ist ein niedriger Wert besser, deswegen wird dieser Wert vorher mit  $(1 - \text{NoiseSensitivity})$  umgekehrt.

Der Gesamtscore liegt zwischen null und eins. Der Gesamtscore wird verwendet, um die untersuchten Konfigurationen miteinander vergleichen zu können.

### **5.1.1 Phase 1: Vorauswahl der Chunking-Konfigurationen**

#### **Wiederholung 1**

In der ersten Phase wurden alle neun Chunking-Konfigurationen gegen fünf feste Fragen aus dem Golden-Set getestet. Dieser Versuch wurde dreimal wiederholt, um Schwankungen zwischen den Testläufen zu berücksichtigen.

Auf der Abbildung 5.2 sieht man, dass in der ersten Wiederholung sich die Chunking-Konfiguration durchgesetzt hat, bei der der Text nach 300 Wörtern unterteilt wird und sich die Chunks um 60 Wörter überschneiden.

Der Unterschied zwischen den besten vier Konfiguration ist vergleichsweise gering. Die Konfigurationen mit der Chunkgröße von 300 Wörtern erreichen die besten Ergebnisse. Das rekursive Chunking-Verfahren erzielt ein ähnlich gutes Ergebnis und liegt nur knapp hinter den besten wortbasierten Konfigurationen.

Deutlich schlechter schneiden die satzbasierten Chunking-Verfahren ab. Am schlechtesten schneidet jedoch die wortbasierte Konfiguration mit einer Größe von 30 Wörtern ab.

Wie in der Abbildung 5.3 abgebildet, haben die Konfigurationen alle einen ähnlichen Energieverbrauch, nur das semantische Chunking-Verfahren verbraucht mehr als das Vierfache von den anderen Konfigurationen. Auch auffällig ist, dass die wortbasierte Konfiguration mit 300 Wörtern und 180 Wörtern Überlappung ebenfalls einen erhöhten Energiebedarf hat.

#### **Wiederholung zwei und drei**

Die Abbildungen 5.4 und 5.6 zeigen die Ergebnisse der zweiten und dritten Wiederholung. Die Abbildungen 5.5 und 5.7 zeigen den Energieverbrauch dieser Wiederholung.

In der zweiten und dritten Wiederholung bleiben die Ergebnisse überwiegend gleich. Auffällig ist, dass die schlechteren Konfigurationen relativ stabile niedrige Scores erreichen und zwischen den Wiederholungen keine großen Schwankungen auftreten. Die Konfigurationen mit den besten Ergebnissen zeigen dagegen stärkere Schwankungen. Ihre Scores liegen zwischen 0,711 und 0,755.

Die besten vier Konfigurationen bleiben die besten Konfigurationen auch in der zweiten und dritten Wiederholung, die Position zwischen ihnen verändert sich jedoch. Während im ersten Durchlauf noch die Konfiguration mit einer Chunkgröße von 300 Wörtern und 60 Wörter

Überlappung den besten Score hatte, performt im zweiten Durchlauf die Konfiguration ohne Überlappung und Chunkgröße von 300 Wörtern am besten. Im dritten Durchgang hat dann die Konfiguration mit einer Chunkgröße von 300 und einer Überlappung von 180 Wörtern den besten Score.

Aufgrund dieser Ergebnisse werden folgende Konfigurationen in Phase zwei übernommen:

- Chunkgröße von 300 Wörtern ohne Überlappung,
- Chunkgröße von 300 Wörtern mit 60 Wörtern Überlappung,
- rekursives Chunking,
- Chunkgröße von 100 Wörtern ohne Überlappung.

Die Konfiguration mit einer Chunkgröße von 300 Wörtern und 180 Wörtern Überlappung wird nicht mit in die zweite Phase genommen, da sie zwar gute Ergebnisse erzielt, der Score aber relativ ähnlich zu den anderen Konfigurationen ist. Außerdem hat diese Konfiguration einen erhöhten Energieverbrauch, ohne einen Vorteil gegenüber den energieeffizienten Konfigurationen zu zeigen.

Stattdessen wird die Konfiguration mit einer Chunkgröße von 100 Wörtern und 0 Wörtern Überlappung mit in Phase 2 übernommen. Sie ist während der ersten Phase sehr konstant geblieben und war bei jeder Wiederholung immer direkt hinter den besten vier Konfigurationen. Sie wird vor allem als Kontrollkonfiguration mit in Phase zwei genommen, um zu untersuchen, ob der schlechte Score der kürzeren Chunkgrößen in Phase eins nur durch die kleine Auswahl von fünf Fragen verursacht wurde. Außerdem ermöglicht sie einen direkten Vergleich mit der Konfiguration mit einer Chunkgröße von 300 Wörtern ohne Überlappung.

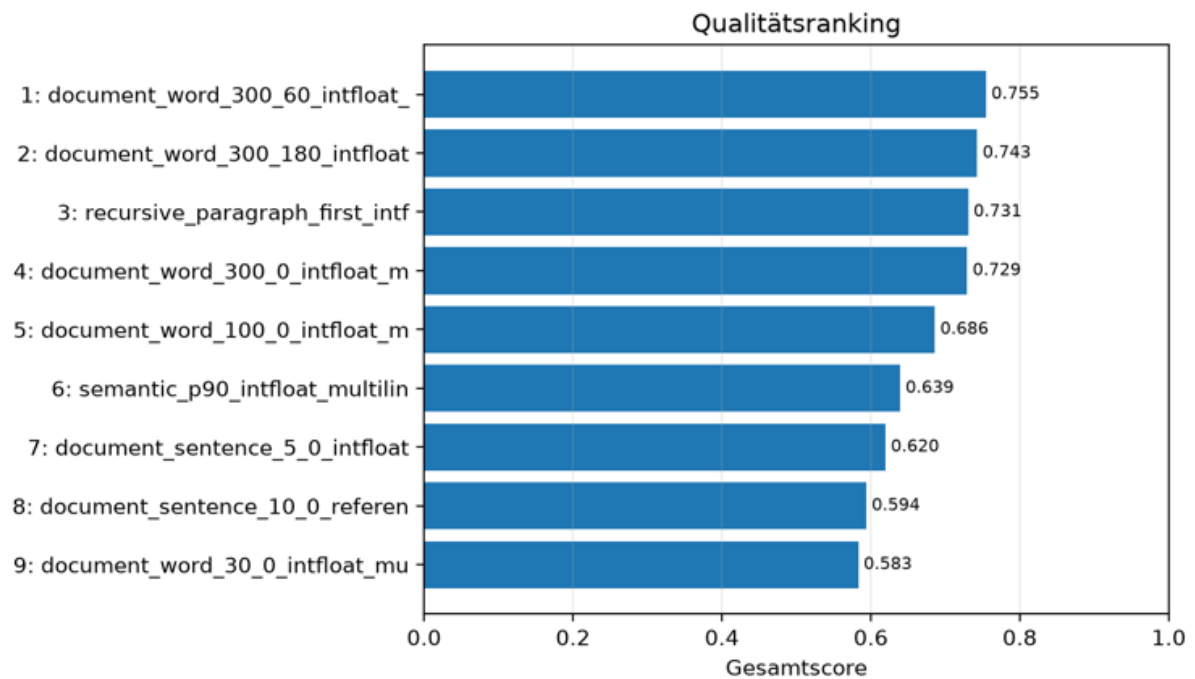


Abbildung 5.2: Gesamtergebnisse der ersten Wiederholung

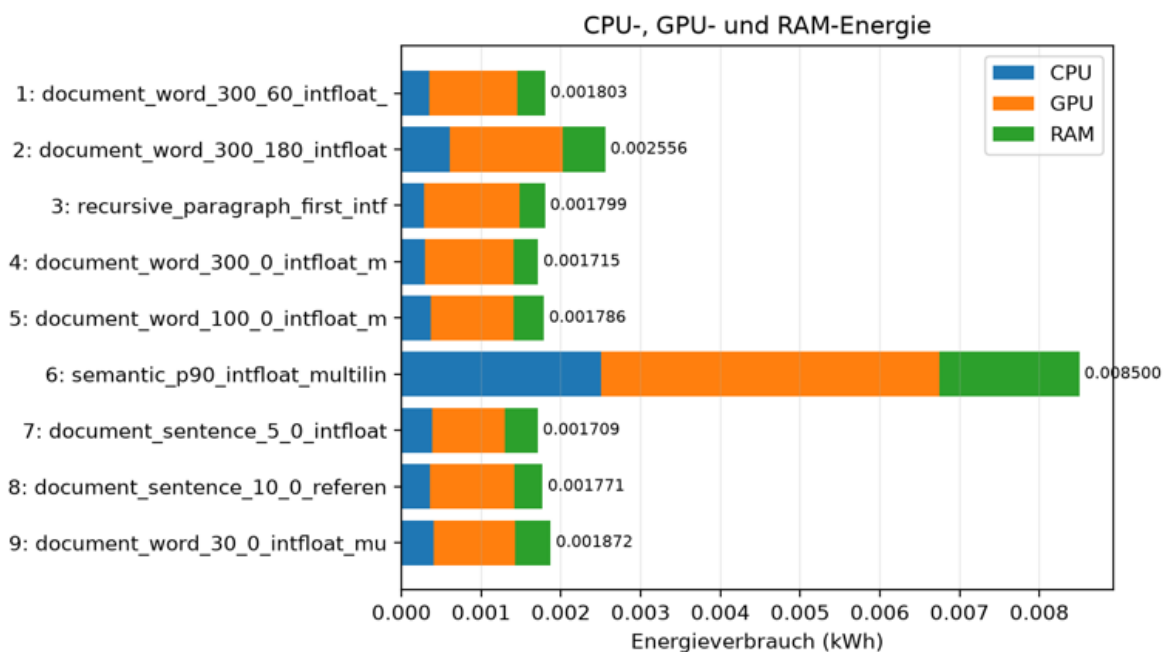


Abbildung 5.3: Stromverbrauch der ersten Wiederholung

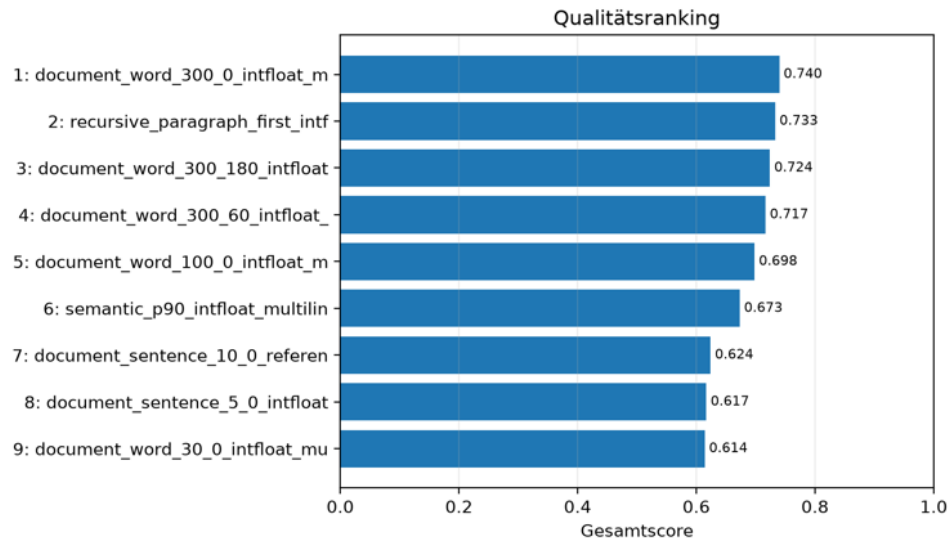


Abbildung 5.4: Gesamtergebnisse der zweiten Wiederholung

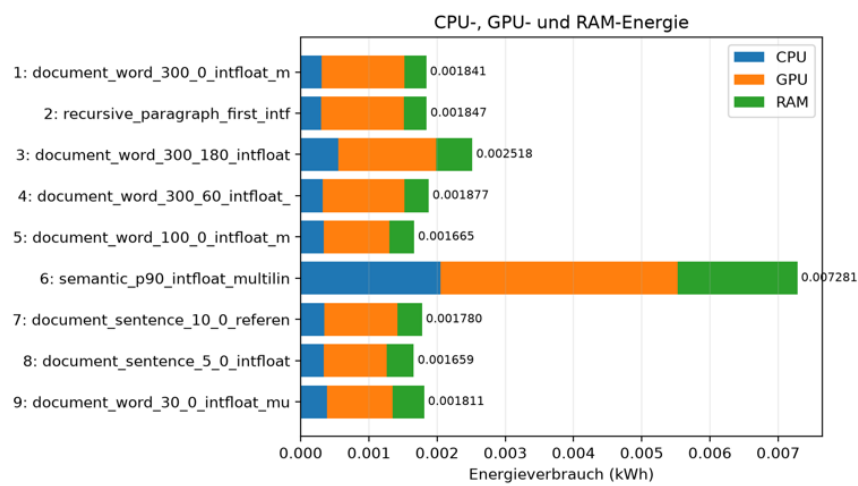


Abbildung 5.5: Stromverbrauch der zweiten Wiederholung

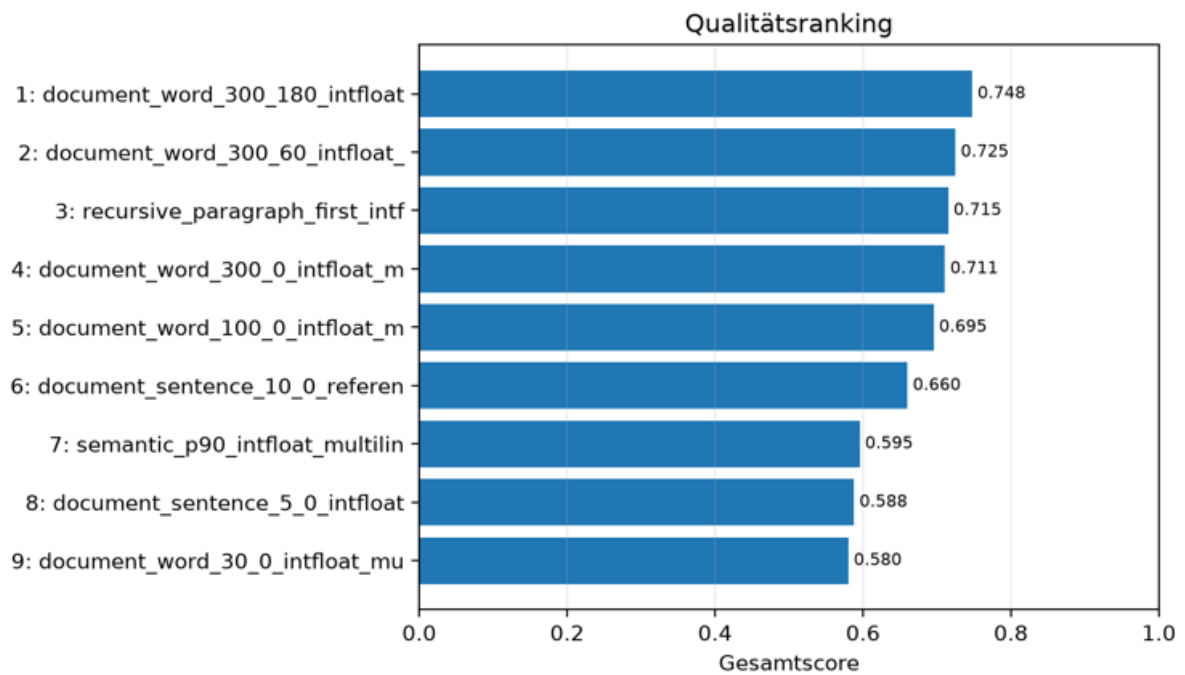


Abbildung 5.6: Gesamtergebnisse der dritten Wiederholung

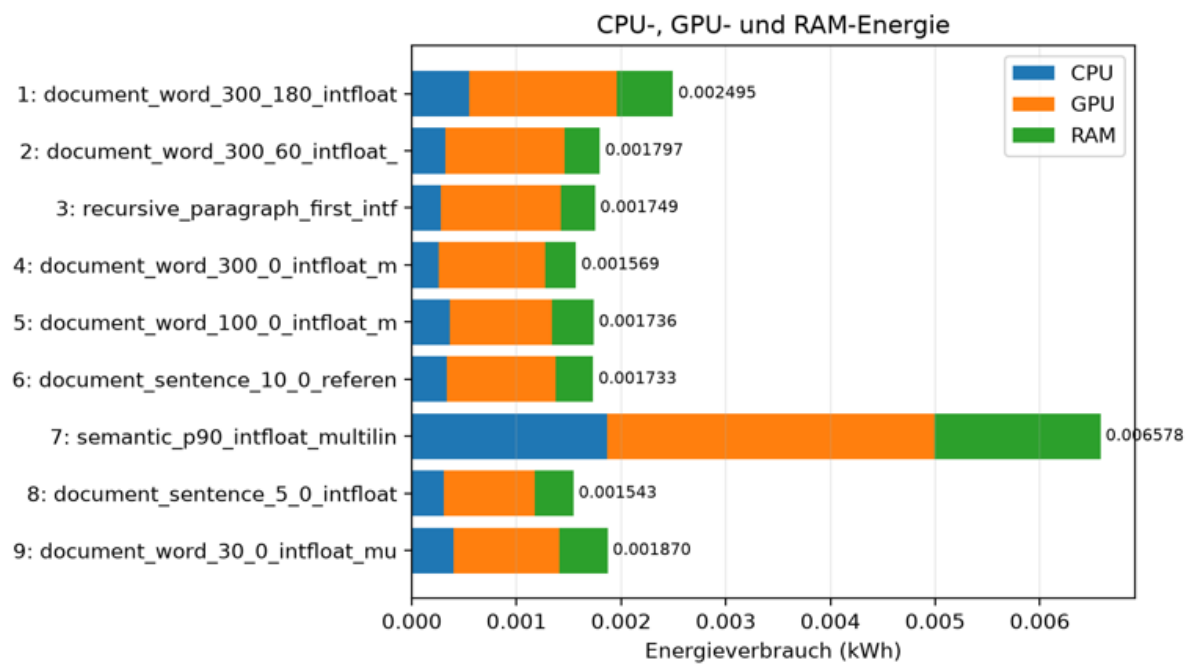


Abbildung 5.7: Stromverbrauch der dritten Wiederholung

### 5.1.2 Phase 2: Testen der besten Chunking-Konfigurationen

In Phase 2 werden die Chunking-Konfigurationen nochmal gegen die volle Fragen-Sammlung getestet. In Phase 2 wurden folgende Konfigurationen untersucht:

- Chunkgröße von 300 Wörtern ohne Überlappung,
- Chunkgröße von 300 Wörtern mit 60 Wörtern Überlappung,
- rekursives Chunking,
- Chunkgröße von 100 Wörtern ohne Überlappung.

In Abbildung 5.8 ist das Gesamtergebnis der Phase 2 abgebildet.

Insgesamt setzt sich das rekursive Chunking-Verfahren durch. Die Scores der anderen Konfigurationen sind leicht schlechter im Vergleich zu dem Test gegen die fünf Fragen in Phase eins. Das Verfahren mit einer Chunkgröße von 100 Wörtern und ohne Überlappung hat weiterhin auch mit allen Fragen einen schlechteren Gesamtscore.

In der Abbildung 5.9 ist zu erkennen, dass die Konfigurationen alle einen ähnlichen Energiebedarf haben. Das Verfahren mit einer Chunkgröße von 100 Wörtern und ohne Überlappung ist die energiesparendste Konfiguration, hat dafür aber einen schlechteren Gesamtscore.

In Phase drei wird das rekursive Verfahren übernommen, da es über die vollen 21 Fragen den besten Gesamtscore hat.

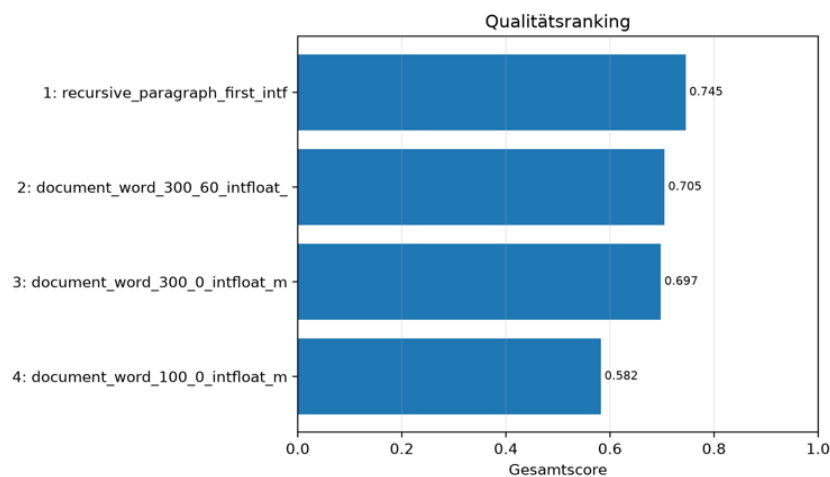


Abbildung 5.8: Gesamtergebnisse Phase 2

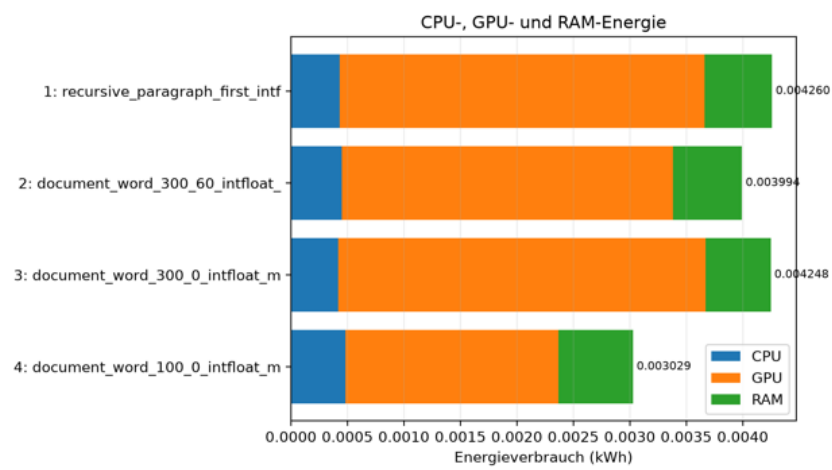


Abbildung 5.9: Stromverbrauch Phase 2

### 5.1.3 Phase 3: Vergleich der Embedding-Modelle und Retrieval-Verfahren

In Phase drei wurde das rekursive Chunking-Verfahren in Kombination mit allen Embedding-Modellen und allen Retriever-Verfahren getestet.

Wie in Abbildung 5.10 abgebildet, setzt sich der bge-m3 mit dem Dense- und mit dem hybriden Retrieval durch.

Am schlechtesten schneiden der intfloat/multilingual-e5-base- und der BAAI/bge-m3-Embedder mit dem Sparse-Retrieval ab.

In Abbildung 5.11 ist der Energieverbrauch der Konfiguration abgebildet. Auffällig ist, dass der BAAI/bge-m3-Embedder deutlich mehr Energie verbraucht als die anderen Embedder. Der intfloat/multilingual-e5-base-Embedder mit dem Dense-Retrieval ist relativ energiesparend und hat einen ähnlich guten Gesamtscore wie der BAAI/bge-m3-Embedder mit dem Dense- und hybriden Retrieval.

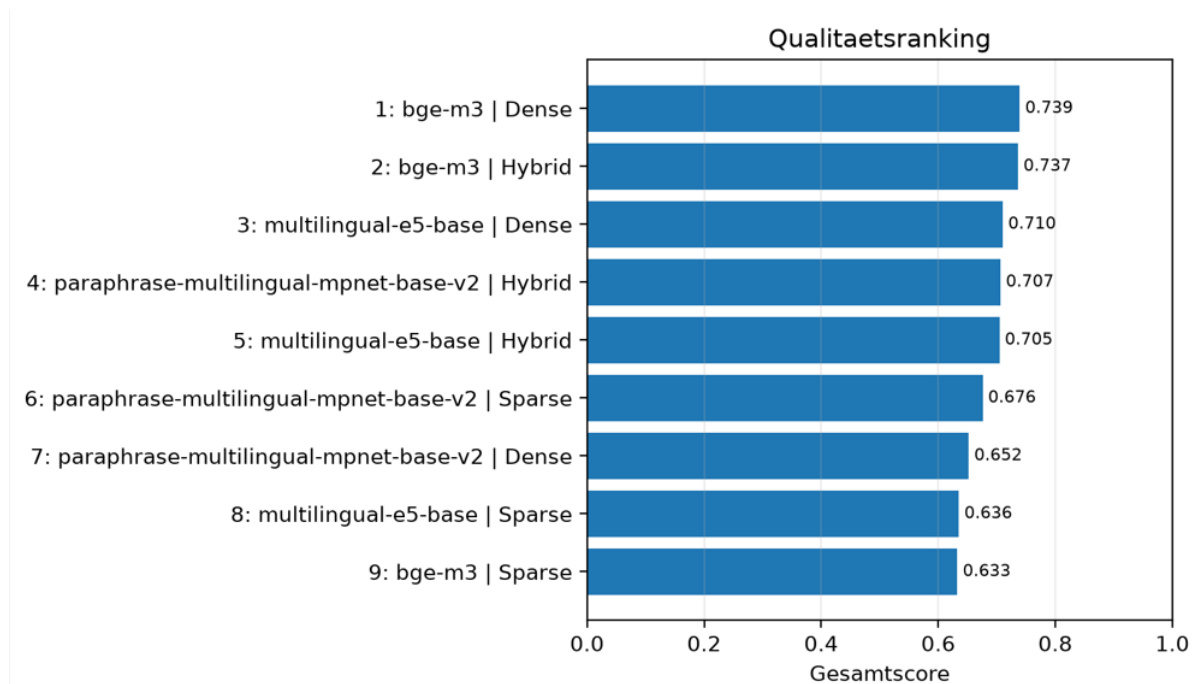


Abbildung 5.10: Endergebnis

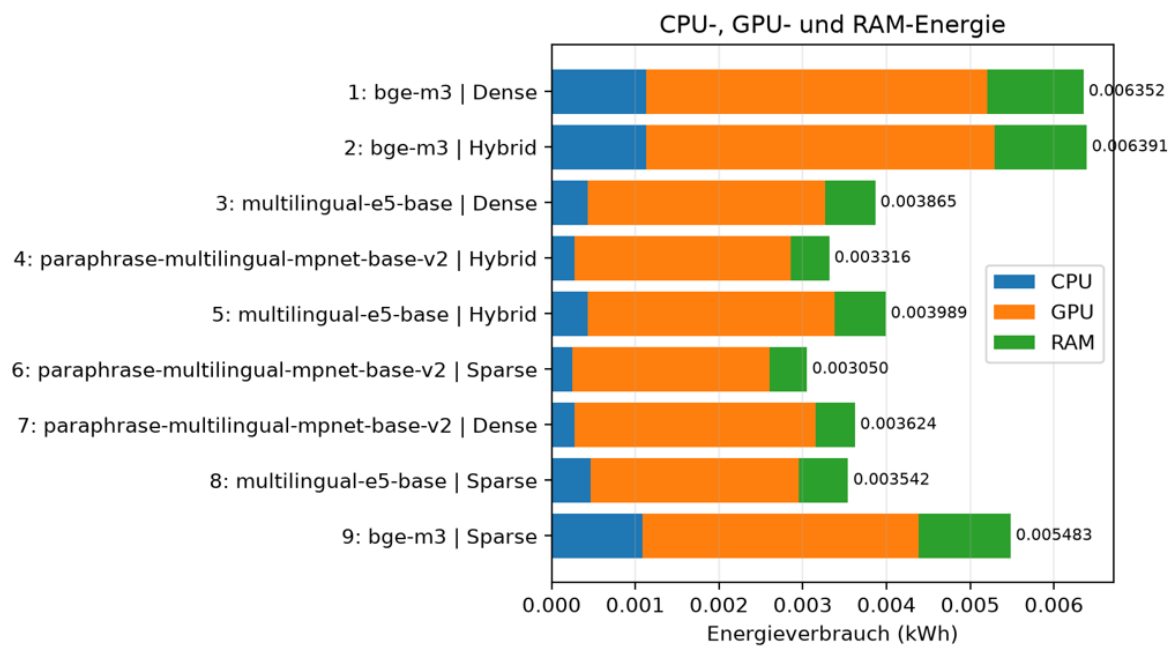


Abbildung 5.11: Endergebnis Energieverbrauch

#### 5.1.4 Beste Kombinationen

In diesem Versuch hat sich das rekursive Chunking-Verfahren als das beste Chunking-Verfahren für diesen Anwendungsfall herausgestellt. In Kombination mit dem rekursiven Chunking-Verfahren hat der BAAI/bge-m3-Embedder mit dem Dense- und dem hybriden Retrieval die höchsten Gesamtscores in Phase drei.

Der BAAI/bge-m3-Embedder hat jedoch einen erhöhten Energiebedarf. Der intfloat/multilingual-e5-base-Embedder mit dem Dense-Retrieval erreicht einen ähnlich hohen Gesamtscore mit einem niedrigeren Energieverbrauch.

Die Auswahl der besten Kombination hängt davon ab, ob man den Gesamtscore maximieren möchte oder der Energieverbrauch im Vordergrund steht.

# 6 Diskussion

## 6.1 Einfluss der Chunk-Konfiguration

### 6.1.1 Einfluss der Chunk-Größe

In dem Versuch hat sich gezeigt, dass Konfigurationen mit größeren Chunks einen höheren Gesamtscore erreichen. Die Konfigurationen mit einer Chunkgröße von 300 Wörtern schneiden besser ab als die Konfigurationen mit 100 oder 30 Wörtern als Chunkgröße.

Große Chunks können zusammenhängenden Kontext beinhalten, der bei kleineren Chunks auf mehrere Chunks aufgeteilt werden muss. Da in diesem Versuch nur die ähnlichsten drei Chunks abgerufen werden, kann wichtiger Kontext verloren gehen.

Die Konfiguration mit einer Chunkgröße von 100 Wörtern wurde mit in die Phase zwei übernommen, um die Auswirkung der kleineren Chunks mit allen Fragen zu evaluieren und die Untersuchungen aus der Phase eins zu bestätigen. Das Ergebnis zeigt, dass diese Konfiguration auch in Phase zwei einen niedrigeren Gesamtscore hat als die Konfiguration mit der Chunkgröße von 300 Wörtern.

In diesem Anwendungsfall hat eine größere Chunkgröße von 300 Wörtern deutlich bessere Gesamtscores als die kleineren Chunkgrößen von 100 und 30 Wörtern und eignet sich daher besser, um Fragen zu beantworten. Daraus kann nicht geschlossen werden, dass eine große Chunkgröße allgemein besser ist. Wie im Kapitel [3.1.2](#) diskutiert, hängt dies sehr stark von dem Datensatz und der Komplexität der Inhalte ab.

### 6.1.2 Einfluss der Chunk-Überlappung

Eine Überlappung von Chunks scheint in diesem Versuch keinen Einfluss auf den Gesamtscore zu haben. Die Konfigurationen mit 300 Wörtern und einer Überlappung von 0, 60 und 180 Wörtern haben alle einen relativ ähnlichen Gesamtscore.

Eine mögliche Erklärung hierfür kann sein, dass die 300 Wörter bereits genügend Kontext beinhalten und die Überlappung kaum zusätzliche Informationen bietet.

Eine hohe Überlappung von 180 Wörtern, also 60 Prozent, hat dafür einen deutlich erhöhten Energieverbrauch. Durch die hohe Überlappung entstehen mehr Chunks und gleiche Textpassagen werden mehrfach embeddet und in der Vektordatenbank gespeichert. Dadurch steigt der Energieverbrauch in der Indexierungsphase.

In diesem Anwendungsfall bietet die Überlappung keinen Vorteil, erhöht jedoch den Energieverbrauch. Darum sollte eine Konfiguration mit keiner oder geringer Überlappung gewählt werden.

### **6.1.3 Rekursives und semantisches Chunking**

Das semantische Chunking hat vergleichsweise schlechte Gesamtscores in diesem Versuch. Gleichzeitig hat das semantische Chunking aber einen erhöhten Energieverbrauch, der vor allem dadurch entsteht, dass das vorhandene Dokument erst mal in Satzgruppen aufgeteilt und dann embeddet wird. Danach müssen zwischen den ganzen embeddeten Satzgruppen noch Ähnlichkeiten gemessen werden. Das erhöht den Energieverbrauch in der Indexierungsphase drastisch.

Zwar wird beim semantischen Chunking-Verfahren an semantischen Grenzen unterteilt, dies scheint aber keinen Vorteil gegenüber den wortbasierten Verfahren mit 300 Wörtern zu haben. Im Gegenteil: Das semantische Chunking-Verfahren hat schlechtere Gesamtscores und einen erhöhten Energieverbrauch. Der zusätzliche Aufwand ist daher nicht gerechtfertigt. Das rekursive Chunking-Verfahren hingegen zählte in der ersten Phase zu den besten Chunking-Verfahren, erreichte in der zweiten Phase den höchsten Gesamtscore und wurde deswegen als beste Konfiguration in die dritte Phase übernommen.

Das rekursive Chunking-Verfahren versucht, Textstrukturen wie Absätze und Sätze zu erhalten. Dadurch werden Inhalte seltener an ungünstigen Stellen getrennt. In dem Anwendungsfall hat dies einen positiven Effekt auf den Gesamtscore und erwies sich als gut geeignete Chunking-Konfiguration.

## **6.2 Einfluss von Embedding-Modell und Retrieval-Strategie**

Die Wahl des Embedding-Modells hatte einen Einfluss auf den Gesamtscore. Den besten Gesamtscore erreichte BAAI/bge-m3 mit dem Dense- und hybriden Retrieval. BAAI/bge-m3 ist speziell für Retrieval-Aufgaben entwickelt worden, was die guten Gesamtscores erklärt. BAAI/bge-m3 hatte aber einen deutlich höheren Energieverbrauch als die anderen Embedding-Modelle.

Das Embedding-Modell `intfloat/multilingual-e5-base` erreichte mit dem Dense-Retrieval einen ähnlichen Gesamtscore wie die Konfiguration mit `BAAI/bge-m3`, hat aber einen deutlich geringeren Energieverbrauch und bietet somit einen guten Kompromiss zwischen Gesamtscore und Energieverbrauch.

`BAAI/bge-m3` und `intfloat/multilingual-e5-base` haben besonders schlechte Gesamtscores, wenn man sie mit einem Sparse-Retriever verwendet. Das liegt vermutlich daran, dass beim Sparse-Retrieval vor allem Wortübereinstimmungen berücksichtigt werden und die semantischen Eigenschaften dieser Modelle nicht genutzt werden.

## 6.3 Empfohlene Konfiguration für ein lokales RAG-System

Um ein Frage-Antwort-System auf Grundlage einer lokalen RAG-Pipeline zu erstellen, würde sich für den hier behandelten Anwendungsfall die Konfiguration aus rekursivem Chunking, dem Embedding-Modell `intfloat/multilingual-e5-base` und Dense Retrieval am besten anbieten. Diese Konfiguration erreicht den besten Kompromiss aus Gesamtscore und Energieverbrauch.

Die Konfigurationen mit dem `BAAI/bge-m3`-Embedder und Dense- beziehungsweise Hybrid-Retriever erreichen zwar höhere Gesamtscores, benötigen aber wesentlich mehr Energie, und der zusätzliche Energieverbrauch führt nur zu einer kleinen Verbesserung des Gesamtscores. Das rekursive Chunking-Verfahren wird verwendet, da es in der Phase zwei insgesamt den besten Gesamtscore hatte und nicht wesentlich mehr Energie verbraucht hat als vergleichbare Chunking-Verfahren.

Auch andere Chunking-Konfigurationen, vor allem die wortbasierten Verfahren mit einer Chunkgröße von 300 Wörtern, erzielten gute Gesamtscores und könnten für den untersuchten Anwendungsfall ebenfalls geeignete Konfigurationen sein.

## 6.4 Limitationen

### 6.4.1 Ungetestete Kombinationen

In diesem Versuch wurden neun Chunking-Konfigurationen, drei Embedding-Modelle und drei Retriever-Strategien verglichen. Daraus ergeben sich 81 mögliche Konfigurationen.

Um den Zeit- und Energieaufwand zu verringern, wurde der Versuch in mehrere Phasen unterteilt. In den ersten zwei Phasen wurden alle neun Chunking-Konfigurationen mit dem

intfloat/multilingual-e5-base-Embedder und Dense Retrieval verglichen. In Phase drei wurde die beste Chunking-Konfiguration, die in den ersten zwei Phasen ermittelt wurde, mit allen Embeddern und Retrieval-Strategien kombiniert.

Es wurde davon ausgegangen, dass die in den ersten zwei Phasen ermittelte Chunking-Konfiguration auch mit den anderen Embeddern und Retrieval-Strategien die beste ist. Es kann aber nicht ausgeschlossen werden, dass eine andere Chunking-Konfiguration in einer anderen Kombination bessere Ergebnisse erreicht hätte.

Da nicht alle 81 Kombinationen durchgetestet wurden, konnten Wechselwirkungen nur teilweise untersucht werden.

### **6.4.2 Begrenzter Anwendungsfall**

Da dieser Versuch nur mit einem einzigen deutschen Dokument durchgeführt wurde, beziehen sich die Ergebnisse nur auf dieses eine Dokument. Die verwandten Arbeiten zeigen, wie die Qualität des Retrievals schwanken kann, je nachdem, wie Dokumente aufgebaut und strukturiert sind und welche Fragen gestellt werden.

Eine Konfiguration, die in diesem Versuch gut funktioniert, funktioniert also nicht zwangsläufig auch in anderen RAG-Systemen mit anderen Datensätzen.

### **6.4.3 Begrenzte Anzahl an Fragen**

In diesem Versuch wird ein Golden Set mit nur 21 Fragen verwendet. Dadurch werden nicht alle Inhalte und alle Fragetypen zu diesem Dokument abgebildet. Es wird nur ein Teil der Anfragen getestet, die in einem realen Anwendungsfall an das RAG-System gestellt werden können.

Außerdem hat jede Frage eine relativ hohe Gewichtung auf das Endergebnis. Wenn eine Frage zufällig besonders gut oder schlecht abschneidet, hat das einen größeren Einfluss auf den Gesamtscore einer Konfiguration. Wenn mehr Fragen im Golden Set vorhanden wären, würden sich einzelne Ausreißer nicht so stark auf den Gesamtscore auswirken.

Um zuverlässiger evaluieren zu können, müsste ein Golden Set viele Fragen und unterschiedliche Fragetypen behandeln.

## 6.4.4 Probleme der RAGAS-Metriken

Wie bereits erwähnt gab es deutliche Unterschiede zwischen den LLM-basierten Retrieval-Metriken und den Retrieval-Metriken ohne LLM.

### LLM-basierte Metriken

Die LLM-basierten Metriken verwenden für diesen Versuch das LLM `mistral:7b`. Um belastbare Aussagen treffen zu können, müssten die Tests mehrfach mit unterschiedlichen LLMs durchgeführt und miteinander verglichen werden, da LLM-basierte Metrikergebnisse Schwankungen unterliegen können.

Context Precision ist wahrscheinlich so hoch, weil die verwendeten Fragen vergleichsweise einfach waren und, da nur maximal drei Chunks abgerufen wurden, relevante Chunks häufig vor weniger relevanten Chunks waren.

Context Recall hingegen sollte nur hohe Werte erreichen, wenn Chunks gefunden wurden, die Aussagen der Referenzantwort belegen. Eine mögliche Erklärung für die hohen Werte wäre, dass das LLM hier zu großzügig bewertet und Chunks als relevant einstuft, auch wenn sie nur allgemeinen Bezug zur Referenzantwort haben. Die Bewertung hängt außerdem von einem einzelnen, relativ kleinen 7B-LLM ab. Andere LLMs könnten dieselben abgerufenen Chunks anders bewerten.

### Nicht LLM-basierte Metriken

Die nicht LLM-basierten Metriken haben die gefundenen Chunks mit Referenzchunks verglichen. Hier sollten hohe Werte erreicht werden, wenn möglichst viele Referenzchunks getroffen werden.

Bei unterschiedlichen Chunking-Verfahren entstehen auch unterschiedliche Chunkgrenzen. Ein gefundener Chunk kann den gleichen Text wie ein Referenzchunk haben, aber an einer anderen Stelle beginnen oder enden. Ein Vergleich kann also scheitern, obwohl die richtige Textstelle abgerufen wurde.

Außerdem kann ein gefundener Chunk nicht nur weniger, sondern auch mehr Text enthalten, als in den Referenzchunks erwartet wird. Das führt dazu, dass ein Vergleich scheitert, obwohl alle Referenzchunk-Textstellen in dem abgerufenen Chunk vorhanden sind. Das Problem ist also, dass die Referenzchunks nicht zu den Chunks passen, die durch die Chunking-Verfahren erzeugt werden. Nicht LLM-basierte Metriken bewerten daher teilweise zu streng.

### 6.4.5 Nicht untersuchte Parameter

In diesem Versuch wurden einige Parameter nicht untersucht, die zusätzlich eine Auswirkung auf die Retrieval-Qualität oder den Energieverbrauch haben könnten. Dazu gehören die Werte `top_k` und die Gewichtung des Hybrid-Retrievals `alpha`.

Der Parameter `top_k` legt fest, wie viele Chunks von dem Retriever abgerufen werden. In diesem Versuch wurde der Wert auf `top_k=3` gestellt. Ein höherer Wert könnte vor allem den Chunking-Konfigurationen mit kleinen Chunk-Größen helfen, zusätzlichen Kontext abzurufen. Durch die zusätzlichen Chunks, die an das LLM übergeben werden, kann aber auch der Energiebedarf steigen.

Der Parameter `alpha` des Hybrid-Retrievers wurde auch nicht verändert. Er wurde auf `alpha=0,5` gestellt und somit waren Dense- und BM25-Retrieval gleich gewichtet. Andere Werte könnten das Ergebnis beeinflussen.

Es wurden außerdem nur die ausgewählten Chunkgrößen, Überlappungen und Chunking-Verfahren untersucht. Andere Konfigurationen könnten bessere Ergebnisse erzielen.

## 7 Fazit

In dieser Arbeit wurde untersucht, welchen Einfluss verschiedene Chunking-Konfigurationen, Embedding-Modelle und Retrieval-Strategien auf die Qualität und den Stromverbrauch einer RAG-Pipeline haben.

Größere Chunks mit 300 Wörtern erreichten in diesem Versuch einen höheren Gesamtscore als kleine Chunks mit 100 oder 30 Wörtern. Eine hohe Chunk-Überlappung führte jedoch nicht zu einer Verbesserung des Gesamtscores. Die Konfigurationen mit 300 Wörtern und den Überlappungen von 0, 60 und 180 Wörtern haben alle ähnliche Gesamtscores. Eine große Überlappung von 60 Prozent (180 Wörtern) führte jedoch zu einem deutlich gestiegenen Energieverbrauch.

Das rekursive Chunking-Verfahren hat in der Phase zwei den besten Gesamtscore gehabt und wurde deswegen als bestes Chunking-Verfahren mit in Phase drei übernommen. Rekursive Chunking-Verfahren versuchen, die Dokumentenstruktur zu erhalten, was in dem untersuchten Anwendungsfall wahrscheinlich zu einer Verbesserung des Gesamtscores geführt hat.

Das semantische Chunking-Verfahren hat einen deutlich erhöhten Energieverbrauch und bietet keine Verbesserung des Gesamtscores. Dass der Text in semantisch zusammenhängende Chunks unterteilt wurde, hat in dem untersuchten Anwendungsfall also keinen positiven Einfluss auf den Gesamtscore.

In der Endphase erreichte das Embedding-Modell BAAI/bge-m3 mit dem Dense- und hybriden Retrieval die höchsten Gesamtscores. Der BAAI/bge-m3-Embedder hat jedoch einen deutlich erhöhten Energieverbrauch. Die Konfiguration mit dem rekursiven Chunking-Verfahren, dem intfloat/multilingual-e5-base-Embedder und dem Dense-Retriever erreicht einen ähnlich hohen Gesamtscore, ist dafür aber wesentlich energiesparender.

Insbesondere für den Betrieb einer lokalen RAG-Pipeline wäre die Konfiguration mit dem rekursiven Chunking-Verfahren, dem intfloat/multilingual-e5-base-Embedder und dem Dense-Retriever die beste Konfiguration im Hinblick auf den Gesamtscore und den Energieverbrauch.

## 7.1 Ausblick

Zukünftige Experimente sollten alle möglichen Kombinationen an Chunking-Konfigurationen miteinander vergleichen, um Wechselwirkungen zu erkennen, die durch den sequentiellen Ablauf in dieser Arbeit nicht untersucht wurden.

Außerdem sollten sie eine größere Vielfalt an Chunking-Konfigurationen untersuchen, die eine breitere Abdeckung von Chunkgrößen bietet. Zusätzlich sollten Parameter untersucht werden, die zusätzlich noch eine Auswirkung auf das Retrieval haben, wie `top_k` und die Gewichtung des Hybrid-Retrievals `alpha`.

So eine vollständige Untersuchung würde jedoch deutlich mehr Zeit und Energie benötigen. Je nach Anzahl der getesteten Kombinationen könnte die Durchführung mehrere Tage oder sogar Wochen dauern.

Um ein zukünftiges Experiment noch aussagekräftiger zu machen, müsste man auch die Anzahl der Referenzfragen und Antworten des Golden Sets erhöhen. Außerdem müsste man die Versuche öfter mit unterschiedlichen LLMs für die Evaluation durchführen, um zu untersuchen, wie stark das bewertende LLM die Ergebnisse beeinflusst.

Mit diesem Versuch hätte man dann aber immer noch nur eine optimale Konfiguration für diesen einen Anwendungsfall gefunden.

Für weitere Arbeiten sollte auch eine geeignete Methode gefunden werden, verschiedene Chunking-Konfigurationen mit unterschiedlich aufgebauten Chunks zu bewerten, ohne dabei ein LLM zu verwenden, das unpassende Chunks als relevant klassifiziert.

# Literatur

- [1] S. J. Russell, P. Norvig und E. Davis, *Artificial intelligence: a modern approach* (Prentice Hall series in artificial intelligence), en, 3rd ed. Prentice Hall, 2010, ISBN: 978-0-13-604259-4.
- [2] C. M. Bishop, *Pattern recognition and machine learning* (Information science and statistics), en. New York: Springer, 2006, ISBN: 978-0-387-31073-2.
- [3] L. A. Camuñas-Mesa, B. Linares-Barranco und T. Serrano-Gotarredona, „Neuromorphic Spiking Neural Networks and Their Memristor-CMOS Hardware Implementations,“ *Materials*, Jg. 12, Nr. 17, 2019, ISSN: 1996-1944. DOI: [10.3390/ma12172745](https://doi.org/10.3390/ma12172745). Adresse: <https://www.mdpi.com/1996-1944/12/17/2745>.
- [4] I. Goodfellow, Y. Bengio und A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [5] D. Jurafsky und J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*, 3rd. 2026, Online manuscript released January 6, 2026. Adresse: <https://web.stanford.edu/~jurafsky/slp3/>.
- [6] A. Vaswani u. a., „Attention Is All You Need,“ *CoRR*, Jg. abs/1706.03762, 2017. arXiv: [1706.03762](https://arxiv.org/abs/1706.03762). Adresse: <http://arxiv.org/abs/1706.03762>.
- [7] J. Robinson, C. M. Rytting und D. Wingate, *Leveraging Large Language Models for Multiple Choice Question Answering*, 2023. arXiv: [2210.12353](https://arxiv.org/abs/2210.12353) [cs.CL]. Adresse: <https://arxiv.org/abs/2210.12353>.
- [8] T. Mikolov, K. Chen, G. Corrado und J. Dean, *Efficient Estimation of Word Representations in Vector Space*, 2013. arXiv: [1301.3781](https://arxiv.org/abs/1301.3781) [cs.CL]. Adresse: <https://arxiv.org/abs/1301.3781>.
- [9] T. Mikolov, W.-t. Yih und G. Zweig, „Linguistic Regularities in Continuous Space Word Representations,“ in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, L. Vanderwende, H. Daumé III und K. Kirchhoff, Hrsg., Atlanta, Georgia: Association for Computational Linguistics, Juni 2013, S. 746–751. Adresse: <https://aclanthology.org/N13-1090/>.

- [10] J. Devlin, M.-W. Chang, K. Lee und K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019. arXiv: 1810.04805 [cs.CL]. Adresse: <https://arxiv.org/abs/1810.04805>.
- [11] N. Reimers und I. Gurevych, *Sentence-BERT: Sentence Embeddings using Siamese BERT- Networks*, 2019. arXiv: 1908.10084 [cs.CL]. Adresse: <https://arxiv.org/abs/1908.10084>.
- [12] Y. A. Malkov und D. A. Yashunin, *Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs*, 2018. arXiv: 1603.09320 [cs.DS]. Adresse: <https://arxiv.org/abs/1603.09320>.
- [13] B. Mitra und N. Craswell, *An Introduction to Neural Information Retrieval*, Jan. 2018. DOI: 10.1561/9781680835335.
- [14] S. Robertson und H. Zaragoza, „The Probabilistic Relevance Framework: BM25 and Beyond,“ *Foundations and Trends in Information Retrieval*, Jg. 3, S. 333–389, Sep. 2009. DOI: 10.1561/1500000019.
- [15] Y. Gao u. a., *Retrieval-Augmented Generation for Large Language Models: A Survey*, 2024. arXiv: 2312.10997 [cs.CL]. Adresse: <https://arxiv.org/abs/2312.10997>.
- [16] P. Lewis u. a., „Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,“ *CoRR*, Jg. abs/2005.11401, 2020. arXiv: 2005.11401. Adresse: <https://arxiv.org/abs/2005.11401>.
- [17] F. Cuconasu u. a., „The Power of Noise: Redefining Retrieval for RAG Systems,“ in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Ser. SIGIR 2024, ACM, 2024, S. 719–729. DOI: 10.1145/3626772.3657834. Adresse: <http://dx.doi.org/10.1145/3626772.3657834>.
- [18] N. F. Liu u. a., *Lost in the Middle: How Language Models Use Long Contexts*, 2023. arXiv: 2307.03172 [cs.CL]. Adresse: <https://arxiv.org/abs/2307.03172>.
- [19] S. R. Bhat, M. Rudat, J. Spiekermann und N. Flores-Herr, *Rethinking Chunk Size For Long-Document Retrieval: A Multi-Dataset Analysis*, 2025. arXiv: 2505.21700 [cs.IR]. Adresse: <https://arxiv.org/abs/2505.21700>.
- [20] Z. Zhong, H. Liu, X. Cui, X. Zhang und Z. Qin, *Mix-of-Granularity: Optimize the Chunking Granularity for Retrieval-Augmented Generation*, 2025. arXiv: 2406.00456 [cs.LG]. Adresse: <https://arxiv.org/abs/2406.00456>.
- [21] D. Berdyugina, A. Cohen und Y. Rioual, *Reducing Redundancy in Retrieval-Augmented Generation through Chunk Filtering*, 2026. arXiv: 2604.24334 [cs.CL]. Adresse: <https://arxiv.org/abs/2604.24334>.
- [22] S. Bennani und C. Moslonka, *A Systematic Analysis of Chunking Strategies for Reliable Question Answering*, 2026. arXiv: 2601.14123 [cs.CL]. Adresse: <https://arxiv.org/abs/2601.14123>.

- [23] M. Amiri und T. Bocklitz, *Chunk Twice, Embed Once: A Systematic Study of Segmentation and Representation Trade-offs in Chemistry-Aware Retrieval-Augmented Generation*, 2025. arXiv: 2506.17277 [cs.IR]. Adresse: <https://arxiv.org/abs/2506.17277>.
- [24] S. Taiwo und M. Amaluddin Yusoff, „EVALUATING CHUNKING STRATEGIES FOR RETRIEVAL-AUGMENTED GENERATION IN OIL AND GAS ENTERPRISE DOCUMENTS,“ in *Network Security amp; Applications*, Ser. CNSA 2026, Academy und Industry Research Collaboration Center (AIRCC), März 2026, S. 49–67. DOI: 10.5121/csit.2026.160507. Adresse: <http://dx.doi.org/10.5121/csit.2026.160507>.
- [25] R. Qu, R. Tu und F. S. Bao, „Is Semantic Chunking Worth the Computational Cost?“ In *Findings of the Association for Computational Linguistics: NAACL 2025*, L. Chiruzzo, A. Ritter und L. Wang, Hrsg., Albuquerque, New Mexico: Association for Computational Linguistics, Apr. 2025, S. 2155–2177, ISBN: 979-8-89176-195-7. DOI: 10.18653/v1/2025.findings-naacl.114. Adresse: <https://aclanthology.org/2025.findings-naacl.114/>.
- [26] K. Enevoldsen u. a., *MMTEB: Massive Multilingual Text Embedding Benchmark*, 2025. arXiv: 2502.13595 [cs.CL]. Adresse: <https://arxiv.org/abs/2502.13595>.
- [27] N. Muennighoff u. a., *Generative Representational Instruction Tuning*, 2024. arXiv: 2402.09906 [cs.CL].
- [28] L. Wang, N. Yang, X. Huang, L. Yang, R. Majumder und F. Wei, „Multilingual E5 Text Embeddings: A Technical Report,“ *arXiv preprint arXiv:2402.05672*, 2024.
- [29] M. Akarsu, R. K. Karaman und C. Mierbach, *From BM25 to Corrective RAG: Benchmarking Retrieval Strategies for Text-and-Table Documents*, 2026. arXiv: 2604.01733 [cs.IR]. Adresse: <https://arxiv.org/abs/2604.01733>.
- [30] V. Karpukhin u. a., „Dense Passage Retrieval for Open-Domain Question Answering,“ in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, B. Webber, T. Cohn, Y. He und Y. Liu, Hrsg., Online: Association for Computational Linguistics, Nov. 2020, S. 6769–6781. DOI: 10.18653/v1/2020.emnlp-main.550. Adresse: <https://aclanthology.org/2020.emnlp-main.550/>.
- [31] deepset GmbH. „Introduction to Haystack.“ Haystack Documentation, Version 3.0, besucht am 4. Aug. 2026. Adresse: <https://docs.haystack.deepset.ai/docs/intro>.
- [32] deepset GmbH. „WeaviateDocumentStore.“ Haystack Documentation, Version 3.0, besucht am 4. Aug. 2026. Adresse: <https://docs.haystack.deepset.ai/docs/weaviatedocumentstore>.
- [33] Weaviate. „Docker Installation.“ Weaviate Documentation, besucht am 4. Aug. 2026. Adresse: <https://docs.weaviate.io/deploy/installation-guides/docker-installation>.
- [34] Ollama. „Gemma 3: 4B.“ Ollama Model Library, besucht am 4. Aug. 2026. Adresse: <https://ollama.com/library/gemma3:4b>.

- [35] D. Kirsten, *Funktionales Kostensplitting zur Ermittlung von Mieterhöhungen nach energetischen Maßnahmen - Eine Handlungsempfehlung auf Basis theoretischer und empirischer Untersuchungen*, doctoralThesis, 2019. Adresse: <https://repos.hcu-hamburg.de/handle/hcu/505>.
- [36] deepset GmbH. „DocumentSplitter.“ Haystack Documentation, Version 3.0, besucht am 4. Aug. 2026. Adresse: <https://docs.haystack.deepset.ai/docs/documentsplitter>.
- [37] deepset GmbH. „RecursiveDocumentSplitter.“ Haystack Documentation, Version 3.0, besucht am 4. Aug. 2026. Adresse: <https://docs.haystack.deepset.ai/docs/recursivesplitter>.
- [38] deepset GmbH. „EmbeddingBasedDocumentSplitter.“ Haystack Documentation, Version 3.0, besucht am 4. Aug. 2026. Adresse: <https://docs.haystack.deepset.ai/docs/embeddingbaseddocumentsplitter>.
- [39] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian und Z. Liu, *BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation*, 2024. arXiv: 2402.03216 [cs.CL].
- [40] N. Reimers und I. Gurevych, „Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,“ in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Nov. 2019. Adresse: <http://arxiv.org/abs/1908.10084>.
- [41] deepset GmbH. „SentenceTransformersDocumentEmbedder.“ Haystack Documentation, Version 3.0, besucht am 4. Aug. 2026. Adresse: <https://docs.haystack.deepset.ai/docs/sentencetransformersdocumentembedder>.
- [42] deepset GmbH. „SentenceTransformersTextEmbedder.“ Haystack Documentation, Version 3.0, besucht am 4. Aug. 2026. Adresse: <https://docs.haystack.deepset.ai/docs/sentencetransformerstextembedder>.
- [43] deepset GmbH. „WeaviateEmbeddingRetriever.“ Haystack Documentation, Version 3.0, besucht am 4. Aug. 2026. Adresse: <https://docs.haystack.deepset.ai/docs/weaviateembeddingretriever>.
- [44] deepset GmbH. „WeaviateBM25Retriever.“ Haystack Documentation, Version 3.0, besucht am 4. Aug. 2026. Adresse: <https://docs.haystack.deepset.ai/docs/weaviatebm25retriever>.
- [45] deepset GmbH. „WeaviateHybridRetriever.“ Haystack Documentation, Version 3.0, besucht am 4. Aug. 2026. Adresse: <https://docs.haystack.deepset.ai/docs/weaviatehybridretriever>.
- [46] Ragas. „Get Started.“ Ragas Documentation, besucht am 4. Aug. 2026. Adresse: <https://docs.ragas.io/en/stable/getstarted/>.
- [47] Ragas. „Context Precision.“ Ragas Documentation, besucht am 4. Aug. 2026. Adresse: [https://docs.ragas.io/en/stable/concepts/metrics/available\\_metrics/context\\_precision/](https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/context_precision/).

- [48] Ragas. „Context Recall.“ Ragas Documentation, besucht am 4. Aug. 2026. Adresse: [https://docs.ragas.io/en/stable/concepts/metrics/available\\_metrics/context\\_recall/](https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/context_recall/).
- [49] Ragas. „Faithfulness.“ Ragas Documentation, besucht am 4. Aug. 2026. Adresse: [https://docs.ragas.io/en/stable/concepts/metrics/available\\_metrics/faithfulness/](https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/faithfulness/).
- [50] Ragas. „Noise Sensitivity.“ Ragas Documentation, besucht am 4. Aug. 2026. Adresse: [https://docs.ragas.io/en/stable/concepts/metrics/available\\_metrics/noise\\_sensitivity/](https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/noise_sensitivity/).
- [51] Ragas. „Answer Correctness.“ Ragas Documentation, besucht am 4. Aug. 2026. Adresse: [https://docs.ragas.io/en/stable/concepts/metrics/available\\_metrics/answer\\_correctness/](https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/answer_correctness/).
- [52] Ragas. „Response Relevancy.“ Ragas Documentation, besucht am 4. Aug. 2026. Adresse: [https://docs.ragas.io/en/stable/concepts/metrics/available\\_metrics/answer\\_relevance/](https://docs.ragas.io/en/stable/concepts/metrics/available_metrics/answer_relevance/).
- [53] Ollama. „Mistral.“ Ollama Model Library, Mistral 7B, Version 0.3, besucht am 4. Aug. 2026. Adresse: <https://ollama.com/library/mistral>.
- [54] CodeCarbon. „CodeCarbon Documentation.“ Documentation, Version 3.2, besucht am 4. Aug. 2026. Adresse: <https://docs.codecarbon.io/3.2/>.

# Erklärung zur Nutzung generativer KI

Bei der Erstellung dieser Arbeit wurde das generative KI Modell GPT-5.6 Thinking ausschließlich verwendet, um Rechtschreibung und Kommasetzung zu Überprüfen und zu Korrigieren. Satzbau Wortwahl und Formulierungen wurden eigenständig erarbeitet.

# **Eigenständigkeitserklärung**

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit mit dem Titel

## **Optimierung und Evaluation der Retrieval-Qualität in lokalen RAG-Systemen**

selbstständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z. B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Hamburg, 04. August 2026