HAW HAMBURG

BACHELORARBEIT

Methoden zur Evaluierung von RAG-Systemen und Analyse der Grenzen, Faktoren und Implikationen für den praktischen Einsatz

vorgelegt am 23. Mai 2025

Jacek Bannach

Erstprüferin: Prof. Dr. Larissa Putzer
Zweitprüfer: Thorben Ortmann

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Department Medientechnik
Finkenau 35
22081 Hamburg

Zusammenfassung

Moderne Sprachmodelle haben in den vergangenen Jahren gezeigt, dass sie als textbasierte Assistentensysteme sehr gut funktionieren. Jedoch stellt für die Modelle weiterhin die Herausforderung dar, auf der Basis verschiedener Fragen vernünftige, konsistente und kontextbezogene Antworten zu generieren. Eine mögliche Verbesserung, um diesem Problem entgegenzuwirken, ist der Aufbau sogenannter RAG-Systeme (Retrieval-Augmented Generation), die einem Sprachmodell spezifischen, relevanten und thematisch passenden Kontext zu einer Frage bereitstellen. Untersuchungen haben zunächst gezeigt, dass es viele verschiedene Metriken gibt, um die Präzision und Leistungsfähigkeit dieser Systeme zu bewerten. Mit den verschiedenen Metriken und einem Datensatz zur Auswertung von Sprachmodellen sowie RAG-Systemen wurde ein umfassender Test entwickelt, um diese Systeme systematisch und objektiv zu evaluieren. Verschiedene Sprachmodelle sowie RAG-Systeme wurden mit diesem Test ausführlich getestet. Anhand der Untersuchungen konnte gezeigt werden, dass die Präzision der Sprachmodelle steigt, sobald sie einen spezifischen Kontext zu einer Frage erhalten. Dies bedeutet, dass RAG-Systeme im Vergleich zu reinen Sprachmodellen in der Regel generell präziser sind.

Abstract

Modern language models have shown in recent years that they work very well as text-based assistant systems. However, these models still face the challenge of generating reasonable, consistent, and context-related answers based on various questions. A possible improvement to counteract this problem is the development of so-called RAG systems (Retrieval-Augmented Generation), which provide a language model with specific, relevant, and thematically appropriate context for a question. Research has initially shown that there are many different metrics to evaluate the precision and performance of these systems. Using various metrics and a dataset to evaluate language models and RAG systems, a comprehensive test was developed to systematically and objectively assess these systems. Various language models and RAG systems were extensively tested with this test. The tests showed that the precision of the language models increases as soon as they are given specific context for a question. This means that RAG systems are generally more precise than pure language models.

Inhaltsverzeichnis

Abbildungsverzeichnis III

Tabellenverzeichnis IV

1 Einleitung 1
- 1.1 Motivation . . . 1
- 1.2 Zielsetzung und Fragestellung . . . 2
- 1.3 Aufbau der Arbeit . . . 2

2 Grundlagen 3
- 2.1 Modelle . . . 3
  - 2.1.1 Neuronale Netzwerke . . . 4
  - 2.1.2 Large Language Models . . . 5
  - 2.1.3 Embeddings . . . 6
  - 2.1.4 BERT . . . 7
  - 2.1.5 Sentence-BERT . . . 7
  - 2.1.6 RAG . . . 8
- 2.2 Metriken . . . 8
  - 2.2.1 Kosinus-Ähnlichkeit . . . 9
  - 2.2.2 Precision und Recall . . . 10
  - 2.2.3 F1-Score . . . 10
  - 2.2.4 Konfusionsmatrix . . . 11
  - 2.2.5 F1-Score auf Wortebene . . . 11
  - 2.2.6 ROUGE-L . . . 12
  - 2.2.7 BLEU . . . 13
  - 2.2.8 BERT-Score . . . 16
  - 2.2.9 SBERT-Score . . . 17
  - 2.2.10 Exact-Match . . . 18

3 Methodik 19
- 3.1 Forschungsdesign . . . 19
- 3.2 Gegenstände der Untersuchung . . . 19
- 3.3 Datengrundlage . . . 19

3.4 Datenerhebung 20
3.5 Auswertung 20

4 Thematische Ausarbeitung 22
4.1 Vorbereitung 22
4.1.1 Evaluationsmethodik 22
4.1.2 Applikation Implementierung 24
4.2 Planung und Durchführung 26
4.2.1 Planung 26
4.2.2 Durchführung 28
4.3 Ergebnisse und Diskussion 28
4.3.1 Einzelfallbeispiel: Testergebnisdatensatz 28
4.3.2 Deskriptive Statistik 30
4.3.3 Auswertung: Sprachmodelle 33
4.3.4 Auswertung: Retriever 37
4.3.5 Auswertung: RAG-Systeme 38
4.3.6 Diskussion 40

5 Fazit 43

Literatur 45

Abbildungsverzeichnis

2.1 „Single Layer Perceptron“ und „Multi Layer Network“ 5
2.2 Transformer - Modell Architekture 6
2.3 RAG - Modell Architekture 9

4.1 „Deskriptive Statistik: Generator Histogramme“ 33
4.2 „Deskriptive Statistik: RAG-System Histogramme“ 34

Tabellenverzeichnis

2.1 Konfusionsmatrix 11
2.2 Beispiel F1-Score auf Wortebene 12
2.3 Beispiel Konfusionsmatrix 12
2.4 Beispiel BLEU Unigrame 15
2.5 Beispiel BLEU Bigramme 15

4.1 Deskriptive Statistik: RAG-Systeme Mittelwerte 31
4.2 Deskriptive Statistik: Sprachmodelle Mittelwerte 31
4.3 Deskriptive Statistik: RAG-Systeme Standardabweichungen 32
4.4 Deskriptive Statistik: Generatoren Standardabweichungen 32

1 Einleitung

1.1 Motivation

Der Durchbruch generativer Sprachmodelle in den Computerwissenschaften hat in den vergangenen Jahren das Interesse vieler Personen im privaten Bereich sowie im Finanzsektor geweckt. Besonders im Fokus steht die Anwendung von Sprachmodellen als privater Assistent in verschiedenen Wirtschafts- und Bildungsbereichen.

Eine der ersten Firmen, die mit Sprachmodellen kommerziellen Erfolg erzielte, war OpenAI mit ihrem Modell GPT-3, das im Jahr 2020 veröffentlicht wurde [Brown et al., 2020] [Simonite, 2022]. Das Sprachmodell wurde speziell für den Einsatz in einem sprachbasierten Assistenzsystem entwickelt. GPT-3 basiert auf einem künstlichen neuronalen Netz. Künstliche neuronale Netzwerke orientieren sich an der Struktur und Funktionsweise biologischer Nervenzellen sowie deren Verknüpfungen im Gehirn. Die technische Architektur von GPT-3 gilt als wegweisend für nachfolgende Sprachmodelle, da sie sich insbesondere im Einsatz als Assistenzsystem bewährt hat.

Sprachmodelle stellen verschiedene Herausforderungen dar. Die größte Herausforderung besteht darin, auf eine Frage oder einen Text eine vernünftige und präzise Antwort zu generieren. Der Begriff der Halluzination beschreibt dabei, dass Sprachmodelle anfällig dafür sind, auch falsche Antworten zu liefern. Es wird vermutet, dass jedes Sprachmodell von Halluzinationen betroffen ist. [Banerjee et al., 2024] [Xu et al., 2024]

Mit dem kommerziellen Erfolg der ersten Sprachmodelle folgten neue Konzepte, um diese Systeme zu erweitern und gleichzeitig den Begriff der Halluzination entgegenzuwirken. Eine mögliche Erweiterung ist eine Wissensbasis, die Sprachmodellen Informationen zur Verfügung stellt, um Fragen effizienter beantworten zu können. Beispiele für solche Informationen könnten verfasste Artikel oder private Dokumente sein. Das Konzept heißt Retrieval Augmented Generation (RAG) und unterstützt Sprachmodelle beim Beantworten von Fragen mit einem passenden Kontext [Lewis et al., 2020]. Auf Basis des vorgestellten Konzepts eröffnen sich neue Möglichkeiten, Personen oder Unternehmen bei ihrer Arbeit gezielt zu unterstützen. Insbesondere für Personen oder Organisationen, denen eine große Menge an privatem Wissen oder wichtigen privaten Informationen zur Verfügung steht.

Eine detailliertere Einführung in diese Thematik erfolgt im Kapitel „Grundlagen“.

1.2 Zielsetzung und Fragestellung

Die Arbeit beschäftigt sich mit der Frage: „Wie lässt sich eine automatisierte Bewertung von Sprachmodellen und RAG-Systemen durchführen?“.

Die Zielsetzung unterteilt sich in folgende Punkte: Zunächst wird der Begriff „Retrieval-Augmented Generation“ (RAG) definiert und kontextualisiert. Anschließend werden geeignete Evaluationsmetriken vorgestellt, mit denen die Leistungsfähigkeit von RAG-Systemen beurteilt werden kann. Im letzten Schritt werden exemplarisch verschiedene RAG-Architekturen aufgebaut und im Hinblick auf ihre Antwortgenauigkeit verglichen und Implikationen auf die Präzision der Systeme gestellt.

1.3 Aufbau der Arbeit

Um die Zielsetzung zu erreichen und um die Arbeit systematisch darzustellen, gliedert sich diese in vier verschiedene Kapitel.

Im Folgenden wird der Aufbau der Arbeit erläutert:

- Kapitel 2 – **Grundlagen** erläutert die theoretischen Konzepte, die zum Verständnis der Zielsetzung erforderlich sind.
- Kapitel 3 – **Methodik** beschreibt das methodische Vorgehen zur Beantwortung der zentralen Forschungsfrage.
- Kapitel 4 – **Thematische Ausarbeitung** behandelt die durchgeführte Forschung, deren Analyse und die Diskussion der Ergebnisse.
- Kapitel 5 – **Fazit** schließt mit einer Zusammenfassung der wichtigsten Ergebnisse, der Beantwortung der Fragestellung sowie Ansätzen für weiterführende Forschung.

Diese Struktur soll sicherstellen, dass der Leser die Argumentation und den Aufbau der Arbeit nachvollziehen kann.

2 Grundlagen

Das Kapitel „Grundlagen“ unterteilt sich in zwei Abschnitte: Modelle und Metriken.

Im Abschnitt zu den Modellen steht das Hauptziel darin, die Frage „Wie funktionieren eigentlich Modelle?“ zu beantworten und um genau festzulegen, was Modelle im Detail sind. Das Ziel ist es, ein systematisches Verständnis von RAG-Systemen (Retrieval-Augmented Generation) zu entwickeln und dieses am Ende detailliert zu präsentieren.

Im Abschnitt über Metriken stehen Evaluationsmethoden im Vordergrund. Ziel ist es zunächst, verschiedene Metriken zu präsentieren und mehr Verständnis für die einzelnen Metriken zu schaffen. Um ihre Anwendung nachvollziehbarer zu machen, werden spezifische Beispiele herangezogen.

2.1 Modelle

Modelle sind auf neuronalen Netzwerken basierende Vorhersagemodelle. Sie werden für spezifische Vorhersagetätigkeiten trainiert. Wenn man einen Text wie z.B. „Es wird“ in das Modell einfügt, könnte eine mögliche Vorhersage „warm“ als nächstes Wort in der Reihenfolge sein. Beliebte Modelle sind beispielsweise „Natural Language Processing“-Modelle [Jurafsky und Martin, 2009], die sich unter anderem in Spracherkennung, Chatbots und Übersetzung unterteilen. Der Fokus der Arbeit liegt vor allem auf Natural Language Processing (NLP)-Modellen, also Sprachmodellen, da RAG-Systeme auf diesen aufbauen.

In den folgenden Abschnitten wird zunächst erklärt, was es mit dem Begriff der künstlichen neuronalen Netzwerke auf sich hat. Basierend darauf werden wichtige Konzepte für moderne RAG-Systeme vorgestellt: Large Language Models (LLMs) und Embeddingmodelle. Zusätzlich werden zwei Embeddingmodelle vorgestellt, die zur Beschreibung von zwei Metriken im nächsten Abschnitt wichtig sind. Das Ziel dieser Herangehensweise besteht darin, die Funktionsweise von RAG-Systemen mithilfe dieses Vorgehens detailliert zu beschreiben.

2.1.1 Neuronale Netzwerke

Neuronale Netzwerke wurden erstmals in dem Paper „A Logical Calculus of the Ideas Immanent in Nervous Activity“ von McCulloch und Pitts eingeführt [McCulloch und Pitts, 1943]. Sie stellen ein vereinfachtes Modell zum Beschreiben von Neuronen im Gehirn dar. Das Modell implementiert Neuronen mit einem binären Operationssystem und basiert auf dem „all-or-none“-Prinzip, bei dem Neuronen entweder aktiv oder inaktiv sind. Das vereinfachte Modell zeigte erstmals, dass sich mathematische Operationen auf neuronalen Netzen durchführen lassen.

Das Modell wurde später von Rosenblatt in seinem Paper „The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain“ für den praktischen Einsatz erweitert [Rosenblatt, 1958]. Er führte dabei den Begriff „Single-Layer Perceptron“ ein. Ein „Single Layer Perceptron“ unterteilt ein neuronales Netzwerk in zwei sogenannte „Layer“: den Input Layer und den Output Layer. Der Input Layer ist für den Eingang von Daten zuständig. Mögliche Werte für den Input Layer sind beispielsweise die Zahlenrepräsentation von Pixel-Daten oder Ähnliches. Der Output Layer ist für die Klassifikation verantwortlich; Outputs wie 1 oder 0 können beispielsweise entscheiden, ob auf einem Bild eine Katze oder ein Hund zu sehen ist. Bei einem „Single Layer Perceptron“ sind beide Layer direkt miteinander durch sogenannte „Weights“, also Gewichtungen, verbunden. Die Gewichtungen bestimmen, ob Neuronen für einen speziellen Input aktiv oder inaktiv sind.

Das wohl bedeutendste Paper zu dem Thema neuronale Netzwerke entstand im Jahr 1986 mit dem Paper „Learning representations by back-propagating errors“ von Rumelhart et al. [Rumelhart et al., 1986]. Das Paper führte speziell die sogenannte „Backpropagation“ für „Multi-layer Networks“ ein. Ein „Multi-Layer Network“ erweitert das „Single Layer Perceptron“ um sogenannte Hidden Layers. Der Hidden Layer wird in einem „Multi-layer Network“ als Brücke zwischen Input- und Output-Layer implementiert. Es können mehrere Hidden Layers hintereinander implementiert werden. Die Verbindungen zwischen den „Layers“ werden weiterhin mit Gewichtungen realisiert. Zusätzlich zu den Gewichtungen wurden hierbei auch sogenannte „Biases“ hervorgehoben. „Biases“ sind wie „Weights“ Parameter in einem neuronalen Netzwerk und dienen dazu, die Ausgabe von Neuronen anzupassen.

Abbildung 2.1 zeigt das „Single-Layer Perceptron“ und das „Multi-Layer Network“ als visualisierte Darstellungen zur Veranschaulichung, wobei die Input-Neuronen sich im Input-Layer befinden, die Hidden-Neuronen im Hidden-Layer und die Output-Neuronen im Output-Layer. Die Schichten sind typischerweise vollständig miteinander verbunden.

Die „Backpropagation“ ist ein sogenannter Lernalgorithmus. Sie verfeinert die Gewichtungen in einem neuronalen Netzwerk zwischen den Layern. Auf Basis eines oder mehrerer Inputs wird jedem Input ein Label zugewiesen. Das Label entspricht dem gewünschten Ausgabewert für einen bestimmten Input. Der Lernalgorithmus stellt die Gewichtungen so ein, dass die

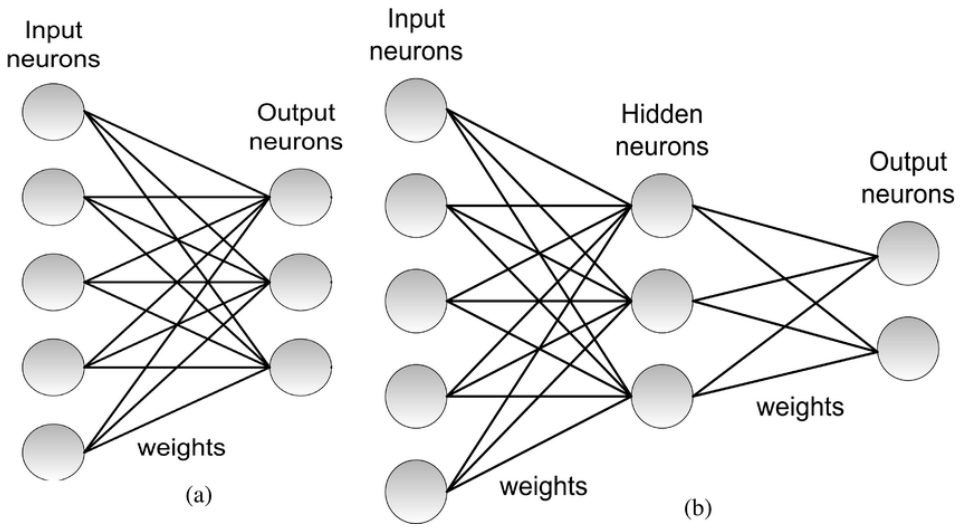

Abbildung 2.1: (a) „Single Layer Perceptron“ und (b) „Multi Layer Network“. (Quelle: Camuñas-Mesa et al., 2019, Abbildunge 2)

Outputs des neuronalen Netzwerks für einen oder mehrere spezifische Inputs möglichst nah am Label sind. Die „Backpropagation“ wird auch oft als Training bezeichnet.

2.1.2 Large Language Models

„Large Language Models“ (LLMs) implementieren neuronale Netzwerke und werden als textbasierte Generatoren verwendet [Naveed et al., 2023]. Sie zählen als Sprachmodelle. Mit ihrer Hilfe kann beispielsweise die Erstellung neuer Texte, das Verfassen von Textzusammenfassungen oder der Betrieb von Chatbots erfolgen. Die Modelle werden als „large“, also groß, durch die Anzahl ihrer Parameter bezeichnet. Parameter bestehen hauptsächlich aus „Weights“ und „Biases“. Beispielsweise besteht das LLM GPT-3 aus circa 175 Milliarden Parametern [Zong und Krishnamachari, 2022]. Die Modelle werden oft mit Datensätzen aus einem weitgefächerten Sortiment an Texten trainiert. Das Sortiment umfasst unter anderem Bücher, Websites, Artikel und Programmcode.

Die Funktionsweise eines Large Language Models (LLM) unterteilt sich in die folgenden Schritte: Zunächst wird die Eingabe tokenisiert. Die Eingabe kann ein beliebiger Text sein. Tokenisieren bedeutet, dass man den Text zunächst in seine einzelnen Wortteile zerlegt. Jedes Wortteil erhält dann eine fest definierte Zahlenrepräsentation. Diese Zahlenrepräsentation wird als Vektor dem Input-Layer übergeben. Nachdem der Input übergeben worden ist, wird ein weiteres Token generiert bzw. als Output aufgeschrieben. Rekursiv wird dann der tokenisierte Text mit dem neuen Token als Input dem LLM übergeben und ein neues

Token wird erstellt. Dies passiert so lange, bis ein Schluss-Token generiert wird, welches die Rekursion beendet.

Als Architektur für LLMs wird oft die Transformer-Architektur verwendet. Die Transformer-Architektur wurde von Vaswani et al. in ihrem Paper „Attention Is All You Need“ eingeführt [Vaswani et al., 2017]. In der Abbildung 2.2 sieht man den Aufbau der Transformers-Architekture

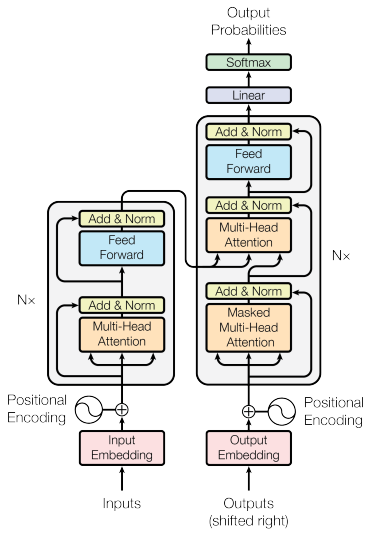

Abbildung 2.2: Transformer - Modell Architekture.
(Quelle: Vaswani et al., 2017, Abbildunge 1)

Hierbei bestehen die Input-Embeddings aus den Tokens des Originaltextes. Das Output-Embedding ist der neu generierter Token. Die Mitte der Darstellung zeigt die Architektur der Hidden Layers. Die Hidden Layers bestehen aus verschiedenen Self-Attention-Mechanismen und Feedforward-Schritten. Output-Probabilities sind die Wahrscheinlichkeiten, welches Token als Nächstes zur Frage steht. Das Token, das am höchsten zu den Wahrscheinlichkeiten passt, wird als nächstes Output-Token genommen.

2.1.3 Embeddings

Embeddings generieren Sätze oder Wörter in eine zahlenbasierte Repräsentation. Beliebt ist hierbei die vektorbasierte Repräsentation. Mit der Zahlenrepräsentation der Wörter ist es möglich, Sätze auf ihre Ähnlichkeit zu prüfen. Zum Beispiel lassen sich für die Wörter „Weihnachten“ und „Weihnachtsmann“ vektorbasierte, zahlenbasierte Repräsentationen erzeugen, deren Ähnlichkeit anschließend mithilfe der Kosinus-Ähnlichkeit geprüft werden kann. Bekannte Modelle für Embeddings sind Word2Vec [CHURCH, 2017], GloVe [Pennington et al., 2014] oder BERT [Devlin et al., 2019]. Bei Embeddings unterscheidet man zwischen Wort- und

Satz-Embeddings. Satz-Embeddings vektorisieren ganze Sätze anstelle einzelner Wörter oder Wortbestandteile. Ein Beispiel für ein Satz-Embeddingmodell ist Sentence-BERT [Reimers und Gurevych, 2019b]. Beliebte Anwendungsbeispiele für Embeddings sind Textklassifikation, maschinelle Übersetzung und semantische Suche. Eine weitere Methode besteht darin, Sätze in Tokens zu unterteilen. Tokens sind spezielle Wortteile, die durch bestimmte Methoden definiert werden. Diese Tokens können dann mithilfe verschiedener Verfahren feste numerische Werte erhalten – also werden Embeddings erstellt. Die Embeddings können anschließend einem neuronalen Input-Layer übergeben werden, um weitere Berechnungen durchzuführen.

2.1.4 BERT

„Bidirectional Encoder Representations from Transformers“ (BERT), eingeführt von Devlin et al. in ihrem Paper „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding“, ist ein vorab trainiertes Embeddingmodell [Devlin et al., 2019]. BERT implementiert hier die Transformer-Architektur, um die kontextuelle Repräsentation von Wörtern auszurechnen.

Die Funktionsweise von BERT lässt sich ungefähr wie folgt beschreiben: Sätze werden zunächst in Tokens umgewandelt. Diese Tokens werden in drei Arten von Embeddings umgewandelt: Token-Embeddings (Wortrepräsentation), Segment-Embeddings (Satzunterscheidung) und Positional-Embeddings (Wortordnung). Schließlich werden die konvertierten Embeddings in das Modell eingespeist, um eine kontextuelle Wortdarstellung als Vektor zu berechnen.

Die Wortdarstellung als Vektor ermöglicht verschiedene Einsatzzwecke. Beliebt ist hierbei der Vergleich auf den Kontext einzelner Wörter. Mithilfe der Kosinus-Ähnlichkeit können Wörter mit ihrer BERT-Vektor-Repräsentation auf ihre Ähnlichkeit geprüft werden.

2.1.5 Sentence-BERT

Sentence-BERT, wurde von Reimers und Gurevych in ihrem Paper „Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks“ [Reimers und Gurevych, 2019b] eingeführt und ist eine modifizierte Version von BERT. Anstatt einzelne Wörter oder Tokens zu vektorisieren, vektorisiert SBERT ganze Sätze. Genau wie bei BERT werden Sätze in Tokens unterteilt und anschließend in drei Arten von Embeddings umgewandelt.

SBERT unterscheidet sich von BERT vor allem durch seinen Output: Anstatt einzelne Wortrepräsentationen zu berechnen, erzeugt SBERT eine kontextuelle Satzdarstellung. Ein relativ beliebter Anwendungsfall, der BERT ähnelt, ist das Prüfen von Sätzen anhand ihrer numerischen Darstellung im jeweiligen Kontext.

2.1.6 RAG

„Retrieval-Augmented Generation“, kurz RAG, eingeführt von Lewis et al. in ihrem Paper „Retrieval-augmented generation for knowledge-intensive nlp tasks“, ist eine Technik zur Kombination von textbasierten Generatoren mit Embeddings [Lewis et al., 2020]. Im Allgemeinen unterteilt sich ein RAG-System in eine Wissensbasis, einen Retrieval-Mechanismus und einen Generator.

Die Funktionsweise der RAG-Systeme unterteilt sich in folgdende schritte: Zunächst wird bei einem RAG eine Wissensbasis erstellt. Die Wissensbasis besteht oft aus einer sogenannten Vektordatenbank [Han et al., 2023]. Mit einem Embeddingmodell werden Texte vektorisiert und dann zusammen mit ihren Vektorrepräsentationen in der Wissensbasis gespeichert. Dabei kann es sich bei den Texten beispielsweise um Artikel, Anleitungen oder wissenschaftliche Publikationen handeln. Nun kann ein Nutzer eine Frage stellen. Auch diese Frage wird mit demselben Embeddingmodell vektorisiert. Anschließend wird die Frage mit den Einträgen der Vektordatenbank verglichen. Eine Vergleichsmethodik könnte z.B. die Kosinus-Ähnlichkeit sein. Die am nächsten liegenden Einträge werden aus der Datenbank extrahiert. Diese Einträge werden als Kontext bezeichnet. Die Anzahl der Einträge kann je nach Implementierung variieren. Anschließend wird die Fragestellung zusammen mit den Kontexten an einen Generator übergeben. Der Generator generiert dann eine Antwort auf die Frage.

Abbildung 2.3 zeigt den Aufbau einer Möglichen RAG Applikation. In der Abbildung sieht man die Architektur unterteilt in drei Teile:

1. Retrieve zeigt den Prozess, um aus der Wissensbasis auf Basis der Eingabe Informationen herauszuschreiben.
2. Augment zeigt den „Prompt“, also die Verbindung von Kontext und Eingabe als einen einzigen Text.
3. Generate zeigt den Generator, bei dem der „Prompt“ als Eingabe in ein LLM eingeht. Das LLM generiert dann eine Ausgabe basierend auf dem Prompt.

Anwendungsbeispiele für Retrieval-Augmented Generation wären z.B. Dokumentzusammenfassungssysteme, Abrufen von Code-Dokumentationen oder die Verbesserung von Frage-Antwort-Systemen.

2.2 Metriken

In folgden Abschnitten werden zunächst grundlegende Metriken wie Kosinus-Ähnlichkeit, Precision, Recall und F1-Score vorgestellt, die als Basis für weiterführende Metriken dienen.

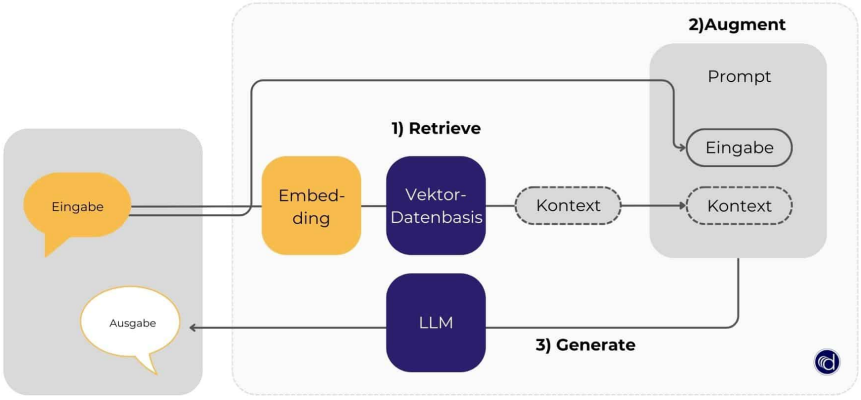

Abbildung 2.3: RAG - Modell Architekture.
(Quelle: Wuttke, 2023)

Dann werden die im Kapitel „Thematische Ausarbeitung“ angewandten Metriken vorgestellt, darunter der F1-Score auf Wortebene, der ROUGE-L-Score, der BLEU-Score sowie der BERT-Score, der SBERT-Score und Exact-Match.

Die Abschnitte zu den angewandten Metriken gliedern sich in eine allgemeine Beschreibung, eine formale Definition sowie ein Anwendungsbeispiel der jeweiligen Metrik.

2.2.1 Kosinus-Ähnlichkeit

Die Kosinus-Ähnlichkeit misst den Winkel zwischen zwei Vektoren [Schütze et al., 2008, Section 6.3]. Die zur besseren Lesbarkeit angepasste Formel lautet wie folgt:

Definition 1 (Kosinus-Ähnlichkeit) *Seien $\vec{a}$ und $\vec{b}$ zwei Vektoren gleicher Dimensionalität. Dann ist ihre Kosinus-Ähnlichkeit csim wie folgt definiert:*

$$csim(\vec{a},\vec{b}) = \cos(\theta) = \frac{\vec{a}\cdot\vec{b}}{|\vec{a}||\vec{b}|} \tag{2.1}$$

cos *ist die Kosinusfunktion;*
θ *ist ein Wert für die Größe des Winkels zwischen beiden Vektoren;*
$\vec{a}\cdot\vec{b}$ *ist das Skalarprodukt der Vektoren;*
$|\vec{a}||\vec{b}|$ *bezeichnet das Produkt der Längen der Vektoren.*

2.2.2 Precision und Recall

„Precision“ und „Recall“, eingeführt von Kent et al. im Paper „Machine literature searching VIII. Operational criteria for designing information retrieval systems“, sind Metriken zur Bewertung von Klassifikationsmodellen, insbesondere in binären Klassifizierungsaufgaben [Kent et al., 1955].

Definition 2 (Precision und Recall) *Im Folgenden ist die Formel dargestellt [Schütze et al., 2008, Section 8.2], welche zur Verbesserung der Lesbarkeit angepasst wurde:*

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

True Positives (TP) steht für die Anzahl der tatsächlichen Positiven, die korrekt als positiv vorhergesagt wurden.
True Negatives (TN) steht für die Anzahl der tatsächlichen Negativen, die korrekt als negativ vorhergesagt wurden.
False Positives (FP) steht für die Anzahl der tatsächlichen Negativen, die fälschlicherweise als positiv vorhergesagt wurden.
False Negatives (FN) steht für die Anzahl der tatsächlichen Positiven, die fälschlicherweise als negativ vorhergesagt wurde.

Hierbei misst „Precision“ die Genauigkeit der Positiven Vorhersagen und „Recall“ die Fähigkeit vom Model zur Identifikation aller relevanten positiven Instanzen.

2.2.3 F1-Score

Der F1-Score, eingeführt von Van Rijsbergen in seinem Buch *Information Retrieval, 2nd edn. Newton, MA* [Van Rijsbergen, 1979], combiniert „Precision“ und „Recall“ in einen Wert.

Definition 3 (F1-Score) *Die Formel, angepasst zur besseren Lesbarkeit, lautet wie folgt [Schütze et al., 2008, Section 8.2]:*

$$F1 = \frac{(1 + \beta^2) \cdot Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall}$$

β steht für die Gewichtung zwischen „Recall“ un „Precision“

Für weitere Beispiele wird angenommen, dass der Wert β gleich 1 ist. Somit ergibt sich für den F1-Score folgende Formel:

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Hierbei misst der Score die Balance zwischen „Precision“ und „Recall“.

2.2.4 Konfusionsmatrix

Die Konfusionsmatrix, stammend aus der statistischen und maschinellen Lernliteratur, ist ein Werkzeug zur Bewertung eines Klassifikationsmodells. Sie zeigt an, wie viele Vorhersagen eines Modells korrekt oder inkorrekt waren. Die Konfusionsmatrix wird in Abbildung 2.1 leicht angepasst dargestellt [Schütze et al., 2008, Section 8.2, Table 8.3].

Tabelle 2.1: Konfusionsmatrix

	Positive Vorhersage	Negative Vorhersage
tatsächlich Positiv	TP	FN
tatsächlich Negativ	FP	TN

2.2.5 F1-Score auf Wortebene

Der F1-Score auf Wortebene implementiert den F1-Score als Methodik zum Vergleich zweier Sätze. Er misst die Überlappung der Wörter zwischen einem Kandidaten und einer Referenz, um den Vergleich darzustellen. Mit folgendem Beispiel wird der F1-Score erklärt.

Nehmen wir z.B. folgenden Satz als Kandidat: „Das Wetter ist oft wunderschön“ und folgenden Satz als Referenz: „Das Hamburger Wetter ist oft nicht so gut.“ Dann schaut man, welche Wörter im Kandidaten und in der Referenz vorkommen, und notiert sie. Also: „Das“, „Wetter“, „ist“, „oft“, „wunderschön“, „Hamburger“, „nicht“, „so“, „gut“. Nun betrachtet man, welche Wörter in der Referenz, welche im Kandidaten und welche sowohl im Kandidaten als auch in der Referenz vorkommen

Wenn ein Wort in beiden Sätzen auftaucht, wird es als „True Positive“ gezählt.
Wenn ein Wort im Kandidaten auftritt, aber nicht in der Referenz, zählt es als „False Negative“.
Falls das Wort nicht im Kandidaten, sondern nur in der Referenz vorkommt, zählt es als „False Positive“.

Tabelle 2.2: Beispiel F1-Score auf Wortebene

Wörter	Kandidat	Referenz	Wert
Das	Wahr	Wahr	TP
Wetter	Wahr	Wahr	TP
ist	Wahr	Wahr	TP
wunderschön	Wahr	Falsch	FN
Hamburg	Falsch	Wahr	FP
nicht	Falsch	Wahr	FP
so	Falsch	Wahr	FP
gut	Falsch	Wahr	FP

Zur Darstellung der Überlappungen dieser Wörter folgt die Tabelle 2.2.

Jetzt können wir die Werte summieren und in die Konfusionsmatrix eintragen. Tabelle 2.3 zeigt die Konfusionsmatrix.

Tabelle 2.3: Beispiel Konfusionsmatrix

	Positive Vorhersage	Negative Vorhersage
tatsächlich Positiv	TP = 3	FN = 1
tatsächlich Negativ	FP = 4	TN = 0

Eingetragen in die Formeln ergbibt sich dan für $Precision \approx 0.42$ und $Recall = 0.75$. Somit ist dan $F1 \approx 0.53$

2.2.6 ROUGE-L

ROUGE-L, entwickelt von Lin und das erste Mal erwähnt im Paper „ROUGE: A Package for Automatic Evaluation of Summaries“, ist eine Vergleichsmetrik, bei der die Reihenfolge der Wörter von Bedeutung ist [Lin, 2004]. Sie implementiert den F1-Score als Metrik zum Vergleich zweier Sätze hinsichtlich ihrer längsten gemeinsamen Teilfolge. Dabei steht „Precision“ für das Verhältnis der längsten gemeinsamen Teilfolge, die im Kandidatensatz vorkommt, und „Recall“ für das Verhältnis in der Referenz. Somit ergeben sich – zur besseren Lesbarkeit angepasst – die folgenden Formeln für „Precision“ und „Recall“ [Lin, 2004]:

Definition 4 (Rouge-L Precision und Recall) *Seien K und R zwei beliebige Sätze. Dann sind Precision und Recall wie folg definiert:*

$$Precision = \frac{LGT(K,R)}{LEN(K)} \qquad Recall = \frac{LGT(K,R)}{LEN(R)}$$

$LGT(K, R)$ ist die Längste gemeinsamen Teilfolge zwischen dem Kandidaten und der Referenz.
$LEN(K)$ ist die Wortanzahl im Kandidaten.
$LEN(R)$ ist die Wortanzahl in der Referenz.

Die Formel für den F1-Score bleibt unverändert. Zur Erläuterung folgt nun ein Beispiel: Wir nehmen diesen Satz als Kandidaten: „Der Sommer wird warm“ und als Referenz: „Der Sommer ist warm“. Die Länge des Kandidaten- und des Referenzsatzes beträgt jeweils vier Wörter. Der Schnitt beider Sätze, also die längste gemeinsame Teilfolge zwischen Kandidat (K) und Referenz (R), lautet: „Der Sommer warm“. Das bedeutet, dass die längste gemeinsame Teilfolge eine Länge von drei Wörtern hat. Somit ergibt sich:

$$LEN(K) = 4 \qquad LEN(R) = 4 \qquad LGT(K, R) = 3$$

Eingesetzt in die Formeln „Precision“, „Recall“ und F1 ergibt sich dann:

$$Precision = \frac{3}{4} = 0.75 \qquad Recall = \frac{3}{4} = 0.75 \qquad F1 = \frac{2 \cdot 0.75 \cdot 0.75}{0.75 + 0.75} = 0.75$$

Somit ist der Wert für Rouge-L der F1-Score also 0.75

2.2.7 BLEU

„Bilingual Evaluation Understudy“ oder kurz BLEU, eingeführt von Papineni et al. in ihrem Paper „Bleu: a Method for Automatic Evaluation of Machine Translation“, ist eine Metrik zur Bewertung der Qualität maschinengenerierter Texte [Papineni et al., 2002]. Sie misst, wie ähnlich ein Kandidat im Vergleich zu einem oder mehreren Referenzen ist. Dabei wird eine Präzisions-basierte Methode mittels n-grams (Reihenfolge von Wörtern) verwendet.

Definition 5 (BLEU) *Zur Lesbarkeitsverbesserung angepasst, ergeben sich die folgenden Formeln für BLEU [Papineni et al., 2002]:*

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log(p_n)\right) \qquad BP = \begin{cases} 1 & c > r \\ e^{(1-r/c)} & c \leq r \end{cases} \qquad \text{es gilt: } \sum_{n=1}^{N} w_n = 1$$

n ist die Anzahl der n-grame, sprich Unigrame (1-grame), Bigrame (2-grame) etc.
N ist das größte n-gram.

p_n sind die Präzisionen für die n-grame.
w_n sind die Gewichtungen für die n-grame die aufsummiert 1 ergeben.
BP ist die Brevity Penalty. Sie wird benutz, um größere Referenzen als der Kandidat zu Strafen.
c ist die Länge vom Kandidaten.
r ist die Länge von der Referenz.

Für p_n nehmen wir folgende Formel:

$$p_n = \frac{\sum_{K \in \{Kandidat\}} \sum_{n\text{-}gram \in \{C\}} Count_{clip}(n\text{-}gram)}{\sum_{K' \in \{Kandidat\}} \sum_{n\text{-}gram' \in \{C'\}} Count_{clip}(n\text{-}gram')}$$

$\sum_{K \in \{Kandidat\}} \sum_{n\text{-}gram \in \{C\}} Count_{clip}(n\text{-}gram)$ steht für die Summe der n-gramme, die im Kandidaten auftreten und auch in der Referenz vorkommen

$\sum_{K \in \{Kandidat\}} \sum_{n\text{-}gram \in \{C\}} Count_{clip}(n\text{-}gram)$ steht für die Summe der n-gramme, die im Kandidaten auftauchen

Um den Score zu erläutern, folgt ein Beispiel. Wir verwenden den folgenden Satz als Kandidaten: „Es schneit im Winter“, während wir diesen als Referenz betrachten: „Es schneit viel mehr im Winter als zuvor“. Wir setzen $N = 2$, was bedeutet, dass wir Unigramme und Bigramme für die Metrik verwenden.

Für Unigramme verwenden wir eine Gewichtung von 0.25 ($w_1 = 0.25$). Für Bigramme verwenden wir eine Gewichtung von 0.75 ($w_2 = 0.75$).

Die Summe der Gewichtungen ergibt $w_1 + w_2 = 0.25 + 0.75 = 1$ was die Bedingung für die Gewichtungen erfüllt.

Wenn wir die Wörter im Kandidaten zählen, erkennen wir, dass der Satz aus vier Wörtern besteht ($c = 4$). Bei der Zählung der Wörter im Referenzsatz stellen wir fest, dass dieser acht Wörter umfasst ($r = 8$).

Da r größer als c ist, gilt für BP:

$$BP = e^{(1-r/c)} = e^{(1-8/4)} \approx 0.367$$

Um die Präzision für die Unigramme zu bestimmen, stellen wir folgende Tabelle auf:

Tabelle 2.4: Beispiel BLEU Unigrame

Unigrame	Kandidat	Referenz	Referenz und Kandidat
Es	1	1	1
schneit	1	1	1
viel	0	1	0
mehr	0	1	0
im	1	1	1
Winter	1	1	1
als	0	1	0
zuvor	0	1	0

Wenn wir die Unigramme zählen, die im Kandidaten auftreten und auch in der Referenz vorkommen, erhalten wir 4. Wenn wir die Unigramme summieren, die im Kandidaten auftauchen, erhalten wir 4. Die Präzision p_1 ist somit das Verhältnis zwischen der Summe der Unigramme, die im Kandidaten auftreten und auch in der Referenz vorkommen, und der Gesamtanzahl der Unigramme, die im Kandidaten auftauchen. Somit ist $p_1 = 4/4 = 1$

Um die Präzision für die Bigramme zu bestimmen, stellen wir folgende Tabelle auf:

Tabelle 2.5: Beispiel BLEU Bigramme

Unigrame	Kandidat	Referenz	Referenz und Kandidat
Es schneit	1	1	1
viel mehr	0	1	0
im Winter	1	1	1
als zuvor	0	1	0

Wenn wir die Bigramme zählen, die im Kandidaten auftreten und auch in der Referenz vorkommen, erhalten wir 2. Wenn wir die Bigramme summieren, die im Kandidaten auftauchen, erhalten wir 2. Die Präzision p_2 ist somit das Verhältnis zwischen der Summe der Bigramme, die im Kandidaten auftreten und auch in der Referenz vorkommen, und der Gesamtanzahl der Bigramme, die im Kandidaten auftauchen. Somit ist $p_2 = 2/2 = 1$

Jetzt können wir alle werte in die BLEU Formel Einsätzen und erhalten:

$$
\begin{aligned}
BP \cdot \exp(\sum_{n=1}^{N} w_n \log(p_n)) &= BLEU \\
0.367 \cdot \exp(0.25 \cdot \log(1) + 0.75 \cdot \log(1)) &= BLEU \\
0.367 \cdot \exp(0) &= BLEU \\
0.367 &= BLEU
\end{aligned}
$$

Somit ist der Bleu-Score ungefähr 0.367.

2.2.8 BERT-Score

Der BERT-Score, eingeführt von Zhang et al. in ihrem Paper „Bertscore: Evaluating text generation with bert“, ist eine Methodik zur Prüfung der Ähnlichkeit von Referenztexten mit einem Kandidaten [Zhang et al., 2019]. Die Metrik zerlegt den Kandidaten und die Referenzen in ihre einzelnen Teile und vektorisiert die Wörter mithilfe eines BERT-Embeddingmodells. Durch die Kosinus-Ähnlichkeit werden die Ähnlichkeiten zwischen den Worten der Referenz und des Kandidaten überprüft. Für jedes Wort wird die größte Ähnlichkeit ermittelt. Anschließend wird der F1-Score wie folgt berechnet:

Für die „Precision“ nimmt man die Summe aller größten Ähnlichkeiten und teilt sie durch die Anzahl der Teile aus dem Kandidaten. Für den „Recall“ nimmt man ebenfalls die Summe aller größten Ähnlichkeiten, teilt diese jedoch durch die Anzahl der Teile aus der Referenz. Dann werden die Werte in die Formel des F1-Scores eingesetzt, um den BERT-Score zu erhalten.

Somit ergibt sich die folgende Formel für den BERT-Score, angepasst zur besseren Lesbarkeit [Zhang et al., 2019]:

Definition 6 (BERT-Score) *Seien K ein Kandidatentext, R ein Referenztext,*
x die Tokens aus dem Referenziertentext,
$\hat{x}$ die Tokens aus dem Kandidatentext,
$\phi(x_i)$ die BERT embedding für Token x_i und
$\phi(x_i)$ die BERT embedding für Token(x_i).

Dann ist die Kosinus-Ähnlichkeit zwischen den Tokens:

$$s_{ij} = cos(\phi(x_i, \hat{x}_j))$$

dann ist „Recall“ wie folgt definiert:

$$R = \frac{1}{m} \sum_{i=1}^{m} \max_j(s_{ij})$$

Dann ist „Precision“ wie folgt definiert:

$$R = \frac{1}{n} \sum_{i=1}^{n} \max_{i}(s_{ij})$$

Schließlich ist der F1-Score wie folgt definiert:

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

Da der BERT-Score auf einem vortrainierten Sprachmodell basiert und mehrereh Rechenschritte beinhaltet, wird an dieser Stelle auf das praktische Beispiel verzichtet, um die Darstellung auf die wesentlichen Inhalte zu konzentrieren und eine zu ausführliche Detailbetrachtung zu vermeiden.

2.2.9 SBERT-Score

Der SBERT-Score ist eine Metrik zum Vergleich zweier Sätze bezüglich ihrer Bedeutung [Reimers und Gurevych, 2019b]. Hierbei wird ähnlich wie beim BERT Score ein Kandidatensatz und einer oder mehrere Referenzsätze mit einem SBERT-Modell vektorisiert. Mit den vektorisierten Sätzen wird dann zwischen dem Kandidaten und einer Referenz die Kosinus-Ähnlichkeit berechnet. Diese ist der endgültige SBERT-Score. Im Gegensatz zu anderen Metriken können Werte zwischen -1 und 1 entstehen. Ein Wert von 1 bedeutet, dass beide Sätze identisch in ihrer Bedeutung sind; ein Wert von 0 bedeutet, dass die beiden Sätze völlig verschieden sind, und ein Wert von -1 bedeutet, dass die beiden Sätze gegensätzliche Bedeutungen haben – was allerdings recht selten auftritt.

Als Beispiel nehmen wir einen Referenzsatz und einen Kandidatensatz. Zunächst erstellen wir Embeddings mit einem SBERT-Modell. Im Anschluss nehmen wir folgende Beispielwerte an:

Das Referenz-Embedding ist:
$\vec{r} = [4, 2, 10]$

Das Kandidaten-Embedding ist:
$\vec{k} = [4, 11, 5]$

Wir wenden nun die Kosinus-Ähnlichkeit wie folgt an:

$$csim(\vec{r}, \vec{k}) = \cos(\theta) = \frac{\vec{r} \cdot \vec{k}}{|\vec{r}||\vec{k}|}$$

Die Berechnung ergibt dann:

$$csim(\vec{r}, \vec{k}) \approx 0.63$$

Somit wäre der SBERT-Score ungefähr 0,63.

2.2.10 Exact-Match

„Exact Match“ ist eine triviale binäre Metrik. Sie kann wie folgt definiert werden:

Nehmen wir an, wir haben einen Referenzsatz und einen Kandidatensatz. Sind beide Sätze exakt identisch, ist der Exact-Match-Wert 1. Andernfalls beträgt der Exact-Match-Wert 0.

Diese Metrik wird unter anderem im Paper „Squad: 100,000+ questions for machine comprehension of text“ von Rajpurkar et al. [Rajpurkar et al., 2016] verwendet.

3 Methodik

3.1 Forschungsdesign

Im Rahmen dieser Untersuchung wird ein quantitativer Forschungsansatz gewählt [Creswell und Creswell, 2017, Chapter 1]. Der quantitative Ansatz verfolgt das Ziel, die Präzision verschiedener Sprachmodelle sowie von RAG-Systemen zu untersuchen. Im Allgemeinen geht es darum, wie präzise die RAG-Systeme spezifischen Kontext aus der Wissensbasis herauslesen können und wie detailliert diese Systeme auf Basis des Kontexts antworten. Um einen Vergleich zu ermöglichen, wie detailliert diese RAG-Systeme ohne Kontext antworten, werden auch deren einzelne Sprachmodelle untersucht.

3.2 Gegenstände der Untersuchung

Im Fokus der Untersuchung stehen unterschiedlich aufgebaute RAG-Systeme, insbesondere deren implementierte Sprachmodelle und Embeddingmodelle. Die Auswahl der Modelle beschränkt sich dabei auf solche mit verschiedener Parameteranzahl sowie unterschiedlicher Hersteller, um Hinweise auf mögliche Unterschiede in deren Präzision zu gewinnen.

3.3 Datengrundlage

Zur Evaluation der RAG-Systeme und Sprachmodelle wird als Datengrundlage das „Retrieval-Augmented Generation (RAG) Dataset 12000“ von Neural Bridge genutzt [AI, 2023]. Der Datensatz ist speziell zur Optimierung von RAG-Systemen gedacht. Mit einer Größe von 12.000 Einträgen bietet der Datensatz die Möglichkeit, eine aussagekräftige Überprüfung durchzuführen.

Der Datensatz unterteilt sich in einen „Test“- und einen „Train“-Abschnitt. Der „Test“-Abschnitt besteht aus 2.400 Zeilen, während der „Train“-Abschnitt 9.600 Zeilen umfasst.

Normalerweise wird der „Train“-Abschnitt dazu verwendet, ein Modell zu trainieren, während der „Test“-Abschnitt für die anschließende Evaluierung vorgesehen ist. Für die folgenden

Tests werden jedoch beide Abschnitte zu einem einheitlichen Datensatz zusammengeführt, um eine möglichst umfangreiche und vielfältige Testmenge zu erhalten.

Die Reihen der Abschnitte unterteilen sich in:

- „Context“ (Kontext): Informationen, mit denen man eine Frage beantworten könnte.
- „Question“ (Frage): Fragen, die an ein RAG-System gestellt werden können.
- „Answer“ (Antwort): die wünschenswerten Antworten auf eine Frage mit einem spezifischen Kontext.

Der Kontext beruht hauptsächlich auf dem Falcon RefinedWeb-Datensatz [Penedo et al., 2023], der Texte aus unterschiedlichen Bereichen wie Sport, Bildung und Allgemeinwissen enthält. Auf Basis dieses Kontexts wurden die Frage-Antwort-Paare unter Verwendung des GPT-4-Sprachmodells [OpenAI, 2023] generiert.

3.4 Datenerhebung

Automatisiert werden die Fragen an die verschiedenen RAG-Systeme übergeben. Dabei werden sowohl die vom Retriever ausgegebenen Kontexte als auch die korrekten Referenzkontexte aus der Datengrundlage gespeichert. Ebenso werden bei den RAG-Systemen sowie den reinen Sprachmodellen die generierten Antworten gemeinsam mit den jeweils korrekten Referenzantworten parallel erfasst und gesichert. Anschließend werden die in den Grundlagen beschriebenen Metriken angewendet, um die entsprechenden Kennwerte auf Basis der erhobenen Daten zu bestimmen und zu dokumentieren.

3.5 Auswertung

Die Auswertung unterteilt sich zunächst in die Erstellung der deskriptiven Statistik [Illowsky und Dean, 2023a]. Dabei werden die erreichten Mittelwerte und Standardabweichungen der verschiedenen RAG-Systeme, reinen Sprachmodelle sowie Retriever anhand der erhobenen Kennwerte bestimmt. Zusätzlich werden Histogramme für das effizienteste RAG-System und das effizienteste reine Sprachmodell erstellt, basierend auf den erhobenen Kennwerten, um Unterschiede zwischen dem RAG-System und dem reinen Sprachmodell zu verdeutlichen. Anschließend werden die Kennwerte mithilfe der deskriptiven Statistik analysiert.

Die Analyse gliedert sich in drei Teile:

- Die Analyse der reinen Sprachmodelle

- Die Analyse der Retriever der RAG-Systeme gemint
- Die Analyse der RAG-Systeme

Bei der Analyse der Retriever in RAG-Systemen steht insbesondere die Präzision der Embeddingmodelle im Fokus. Anschließend weden anhand der Analyse Implikationen über die Systeme gestellt.

4 Thematische Ausarbeitung

Die thematische Ausarbeitung unterteilt sich in drei Teile. Zunächst wird die grundlegende Vorbereitung vorgestellt. Diese umfasst die Ausarbeitung des Aufbaus der Sprachmodelle und RAG-Systeme sowie die Vorbereitung der Auswertung der Systeme mit den vorgestellten Metriken. Dann wird im Abschnitt „Planung und Durchführung“ der allgemeine Verlauf der Durchführung der Tests vorgestellt. Abschließend werden im Abschnitt „Ergebnisse und Diskussion“ die resultierenden Ergebnisse der Tests gezeigt und diskutiert.

4.1 Vorbereitung

Die Vorbereitung unterteilt sich in zwei Abschnitte. Zunächst wird im Abschnitt „Evaluationsmethodik“ die Implementierung der Metriken diskutiert und anschließend die Implementierung der Evaluationsmethodik beschrieben. Abschließend wird im Teil „Implementierung der Applikation“ beschrieben, wie sich die RAG-Systeme in der vorliegenden Arbeit aufbauen, welche Modelle verwendet werden und wie der allgemeine Ablauf der Tests gestaltet ist.

4.1.1 Evaluationsmethodik

Die Evaluationsmethode zur Bewertung der Embedding-Systeme basiert auf dem Exact-Match-Verfahren. Es wird geprüft, wie Präzise die Systeme relevanten Kontext aus einer Wissensbasis extrahieren können.

Zur Bewertung der RAG-Systeme wird eine textbasierte Evaluationsmethode verwendet. Hierbei wird mithilfe einer Frage aus dem Datensatz ein Kandidatensatz mit einem RAG-System und Sprachmodell generiert. Der Kandidatensatz wird dann auf eine Referenzantwort geprüft. Die Referenzantwort ist die korrekte Antwort zur Frage, die im Datensatz enthalten ist. Zur Evaluation werden fünf verschiedene Metriken verwendet, um ein breites Spektrum an Informationen zwischen den Referenz- und Kandidatentexten zu bieten.

Um die Überschneidung der Wörter zwischen den Texten darzustellen, wird der F1-Score auf Wortebene verwendet. Da der F1-Score auf Wortebene die Reihenfolge der Wörter nicht berücksichtigt, wird ergänzend der ROUGE-L-Score eingesetzt.

Der ROUGE-L-Score bewertet die Übereinstimmung der Wortreihenfolge zwischen den Sätzen anhand der längsten gemeinsamen Teilfolge [Lin, 2004]. Da der Rouge-L-Score die allgemeine Überlappung der Wörter nicht berücksichtigt, wird der BLEU-Score verwendet [Papineni et al., 2002].

Mithilfe des n-Gramm-Ansatzes ermöglicht der BLEU-Score die Überprüfung der Wortüberlappung und Reihenfolge zwischen den Texten. Um gesonderte Informationen über Reihenfolge und Wortüberlappung zu erhalten, werden der F1-Score auf Wortebene und der ROUGE-L-Score weiterhin verwendet, da der BLEU-Score dies nicht berücksichtigt.

Da diese drei Metriken jedoch nicht auf die Bedeutung der Wörter und Texte eingehen, werden zwei weitere Metriken eingesetzt: Der BERT-Score bietet eine Metrik an [Zhang et al., 2019], um den Vergleich auf Wortebene zwischen den Texten darzustellen, während der SBERT-Score die Überprüfung der Texte auf ihre Bedeutung ermöglicht [Reimers und Gurevych, 2019b].

Der F1-Score auf Wortebene wird mithilfe des folgenden Python-Codes implementiert, welcher in Codeblock 4.1 gezeigt wird. Dabei gilt der Parameter $\beta = 1$.

Codeblock 4.1: F1 Score auf Wortebene

```python
from collections import Counter

def f1_score(prediction, ground_truth):

    pred_tokens = prediction.lower().split()
    truth_tokens = ground_truth.lower().split()

    common_tokens = Counter(pred_tokens) & Counter(truth_tokens)

    num_same = sum(common_tokens.values())

    if num_same == 0:
        return 0.0

    precision = num_same / len(pred_tokens)

    recall = num_same / len(truth_tokens)
    f1 = (2 * precision * recall) / (precision + recall)

    return f1
```

„Prediction“ ist der Kandidatentext, während die „ground truth“ der Referenztext ist. Der Kandidat wird zunächst in seine Wörter geteilt und anschließend werden Großbuchstaben in Kleinbuchstaben konvertiert und in eine Liste geschrieben. Die Referenz wird ebenfalls zunächst in ihre Wörter geteilt, anschließend werden Großbuchstaben in Kleinbuchstaben konvertiert und in eine Liste geschrieben.

Danach werden die gemeinsamen Wörter in eine Liste geschrieben. Dann wird die Summe der gemeinsamen Wörter berechnet. Wenn der Fall eintritt, dass es keine gemeinsamen Wörter gibt, ist der F1-Score 0. Anschließend werden Präzision, Recall und der F1-Score berechnet; dabei ist der F1-Score der Rückgabewert.

Die ROUGE-L-Metrik wird mithilfe der „ROUGE-Score“ Python-Bibliothek implementiert [Lin, 2004] [Google AI, 2024]. In der Implementierung der ROUGE-L-Metrik ist standardmäßig $\beta = 1.2$.

Die BLEU-Metrik wird mithilfe der „sacrebleu“ Python-Bibliothek implementiert [Post, 2018] [Papineni et al., 2018]. Bei der Implementierung der BLEU-Metrik werden standardmäßig bis zu Tetragramme (4-gramme) geprüft. Die standardmäßigen Gewichtungen für die vier n-Gramme sind $w_1 = 0.25$, $w_2 = 0.25$, $w_3 = 0.25$ und $w_4 = 0.25$.

Die BERT-Score-Metrik wird mithilfe der „BERT-Score“ Python-Bibliothek umgesetzt [Zhang et al., 2019] [Zhang et al., n. d.]. Als Embedding-Modell wird hierbei das auf BERT basierende „deberta-xlarge-mnli“ von Microsoft benutzt [He et al., 2020] [Microsoft, 2021]. DeBERTa „deberta-xlarge-mnli“ hat hier einen Genauigkeitswert von 97,5 im Stanford Sentiment Treebank (SST-2) Benchmark [Socher et al., 2013]. β ist hierbei für den F1-Score standardmäßig gleich 1.

Der SBERT-Score wird mithilfe der „SentenceTransformer“ Python-Bibliothek implementiert [Reimers und Gurevych, 2019b] [Reimers und Gurevych, 2019a]. Als Embedding-Modell wird hierbei das auf SBERT basierende populäre „all-MiniLM-L6-v2“ Modell benutzt [Wang et al., 2020]. Auf der Seite sbert.net hat das Modell einen Performance-Wert von 68,06 auf 14 verschiedenen Datensätzen [Reimers und Gurevych, 2025].

4.1.2 Applikation Implementierung

Das RAG-System wird mithilfe der ChromaDB [Chroma, 2025] und Transformers [Wolf et al., 2020] Python-Bibliotheken implementiert.

Die Retriever werden über die Vektor-Datenbank ChromaDB implementiert. Für die Implementierung der Retriver werde folgende Embeddingmodelle verwendet:

1. all-MiniLM-L6-v2 [Wang et al., 2020]

2. all-mpnet-base-v2 [Song et al., 2020]
3. multi-qa-distilbert-cos-v1 [Reimers und Gurevych, 2021]

Die Embeddingmodelle werden mit einem Exact-Match-Verfahren auf ihre Effizienz geprüft. Dabei wird überprüft, wie präzise die Retriever die Informationen aus der Wissensbasis abrufen können.

Die Sprachmodelle werden mithilfe der Python-Bibliothek Transformers implementiert [Wolf et al., 2020]. Allen Sprachmodellen wird zudem ein „Heat"-Wert (Temperature) von 0,3 sowie eine maximale Token-Anzahl von 100 zugewiesen. Der „Heat"-Wert oder auch „Tempeture"-Wert beschreibt, wie zufällig oder kreativ die generierten Antworten sein sollen: Ein Wert von 0,0 bedeutet, dass die Generierung deterministisch erfolgt, während ein Wert von 1,0 eine hohe Kreativität bzw. Zufälligkeit zur Folge hat [Peeperkorn et al., 2024]. Ein „Heat"-Wert von 0,3 fokussiert sich dabei auf eine eher deterministische Antwort mit etwas Zufälligkeit und Kreativität. Die maximale Token-Anzahl gibt an, wie viele Tokens von einem Sprachmodell höchstens generiert werden können.

Die Transformers-Bibliothek ermöglicht es, vortrainierte Modelle relativ effizient zu nutzen.

Zur Implementierung werden folgende Sprachmodelle als Generatoren verwendet:

1. Phi-4 [Bai et al., 2023]
2. zephyr-7b-beta [Tunstall et al., 2023]
3. Qwen2.5 mit 14 Billionen Parametern [Yang et al., 2024]
4. Qwen2.5 mit 7 Billionen Parametern [Yang et al., 2024]
5. Qwen2.5 mit 0,5 Billionen Parametern [Yang et al., 2024]

Zusätzlich werden alle Embedding-Modelle mit allen Sprachmodellen für den Versuchsaufbau kombiniert. Für den Versuchsaufbau werden somit insgesamt drei Embeddingmodelle und fünf Sprachmodellen genutzt. Somit werden insgesamt fünfzehn verschiedene RAG-Systeme implementiert.

Die Verbindung zwischen Eingabe und Kontext, also der Prompt, wird wie folgt implementiert: Zunächst wird aus der Wissensbasis mithilfe einer Eingabe – in diesem Fall die Frage aus der Datengrundlage – der Kontext über die Vektor-Datenbank abgerufen.

Der Text für den Prompt der RAG-Systeme wird wie folgt definiert:

„scenario: answer one question with one sentence. use more than 2 words
knowledge: <kontext>
question: <frage>
answer:"

„Scenario“ (Szenario) beschreibt, wie der Generator eine Frage beantworten soll. In diesem Fall soll der Generator eine Frage mit einer Antwort beantworten. Zusätzlich soll der Generator mehr als zwei Wörter verwenden. „Knowledge“ (Wissen) beschreibt das Wissen, das der Generator zur Beantwortung der Frage nutzen soll. „<Kontext>“ bezeichnet den spezifischen Kontext zur Frage, der mithilfe eines Embedding-Modells aus einer Wissensbasis extrahiert wurde. „Question“ (Frage) beschreibt die Frage, die der Generator beantworten soll. „<Frage>“ ist eine Frage aus dem „Retrieval-Augmented Generation (RAG) Dataset 12000“ Datensatz. „Answer“ (Antwort) leitet die Generierung der Antwort durch den Generator ein.

Die Tests unterteilen sich in jedes einzelne RAG-System und Sprachmodell. Für jedes RAG-System wird eine neue Wissensbasis mithilfe des zugehörigen Embeddingmodells erstellt. Für die ersten 10.000 Zeile des „Retrieval-Augmented Generation (RAG) Dataset 12000“ werden die Metriken angewandt.

Der allgemeine Ablauf einer Zeile ist wie folgt: Zunächst wird die Frage, der richtige Kontext und die richtige Antwort aus dem Datensatz extrahiert. Anschließend wird mithilfe eines Embedding-Modells ein vorhergesagter Kontext aus der Wissensbasis generiert. Der vorhergesagte Kontext wird mit der Frage als Prompt kombiniert.

Der Prompt wird dann an den Generator geschickt, welcher eine vorhergesagte Antwort generiert. Die fünf Metriken werden anschließend auf die vorhergesagte Antwort (Kandidat) und die richtige Antwort (Referenz) aus dem Datensatz angewendet und dokumentiert. Jetzt wird die gleiche Frage an das Sprachmodell gestellt, diesmal jedoch ohne den Kontext („knowledge: <kontext>“), und eine neue vorhergesagte Antwort wird generiert.

Die neu generierte Antwort (Kandidat) wird dann mit den fünf Metriken auf die richtige Antwort (Referenz) geprüft. Die Ergebnisse werden dokumentiert. Anschließend wird der richtige Kontext, der vorhergesagte Kontext, die erste generierte Antwort, die zweite generierte Antwort, die richtige Antwort und die Ergebnisse der Metriken in einen neuen Datensatz im JSON-Format überführt.

Das JSON-Format ermöglicht es, Texte in einem menschenlesbaren Format abzuspeichern [Bray, 2017]. Zusätzlich können viele Programmiersprachen wie JavaScript oder Python JSON-Daten in ein objektorientiertes Format umwandeln, um sie später auswerten zu können.

4.2 Planung und Durchführung

4.2.1 Planung

Zunächst wurde ein kurzer Test für ein RAG-System mit circa 1.000 Zeilen aus dem Datensatz durchgeführt, um die benötigte Zeit für einen vollständigen Test mit etwa 10.000 Zeilen

abzuschätzen. Der Test wurde auf dem Server der HAW Machine Learning AG durchgeführt und mithilfe der Programmiersprache Python geschrieben [Van Rossum und Drake, 2009]. Der Server hatte folgende Spezifikationen: Zwei Prozessoren mit 32 Kernen, vier Nvidia Tesla A100 Grafikkarten [NVIDIA Corporation, 2020] und insgesamt 1.024 GB RAM.

Dabei stellte sich heraus, dass der Test für etwa 1.000 Zeilen rund 2 Stunden und 30 Minuten benötigte. Das Zehnfache dieser Zeit ergibt ungefähr 25 Stunden. Somit sollte ein vollständiger Test für ein RAG-System etwa 25 Stunden dauern. Mit insgesamt 15 Tests beläuft sich die Gesamtdauer ungefähr auf 375 Stunden. Man kann jedoch davon ausgehen, dass sich die Dauer je nach Anzahl und Konfiguration der Parameter sowie der Implementierung der Generatoren – also der LLM-Modelle – unterscheiden kann. Hinzu kommt, dass die Leistung von CPU und GPU nicht zu 100 Prozent deterministisch ist, was ebenfalls zu Schwankungen führen kann.

Nach dem kurzen Test wurde die Verteilung der Workloads auf die vier Nvidia-Grafikkarten geplant. Dies kann zur Folge haben, dass sich die gesamte Testlaufzeit verkürzt. Mit vier parallel laufenden Prozessen kann sich die Laufzeit um das Vierfache verkürzen. Somit kann die allgemeine Testlaufzeit ungefähr 94 Stunden betragen. Dabei wurde festgelegt, welches RAG-System auf welcher Grafikkarte laufen sollte. RAG-Systeme mit dem gleichen Generator, aber unterschiedlichen Embedding-Modellen liefen auf derselben Grafikkarte.

Im Folgenden ist die genaue Planung der Auslastung dargestellt:

- Auf Grafikkarte 1 sollten die RAG-Systeme mit dem Generator „Microsoft-Phi“ mit 14 Billionen Parametern laufen.
- Auf Grafikkarte 2 sollten die RAG-Systeme mit dem Generator „Qwen2.5“ mit 14 Billionen Parametern laufen.
- Auf Grafikkarte 3 sollten die RAG-Systeme mit dem Generator „zephyr-7b-beta“ mit 7 Billionen Parametern laufen.
- Auf Grafikkarte 4 sollten die RAG-Systeme mit den Generatoren „Qwen2.5“ mit 7 und 0,5 Billionen Parametern laufen.

Die Umsetzung der Workloads erfolgte über die Software Docker [Docker, Inc., 2024]. Docker ermöglicht mithilfe von Container-Virtualisierung und der direkten Implementierung von CUDA [NVIDIA Corporation, 2024] eine relativ einfache parallele Ausführung von Applikationen oder Skripten auf verschiedenen Grafikkarten. Zusätzlich bietet Docker die Möglichkeit, eigene Images zu erstellen. Images sind quasi Blaupausen für Container.

Dabei wurde ein Image erstellt, das Python sowie die für die Tests benötigten Python-Bibliotheken bereits vorinstalliert hatte. Das bedeutet, dass die Umgebung für alle vier

Grafikkarten – und damit auch die Tests – mit demselben Image durchgeführt wurden, wodurch sie sich stark ähneln.

4.2.2 Durchführung

Im Allgemeinen wurden die Skripte in den ersten zwei Tagen angepasst, damit sie auch auf den Umgebungen fehlerfrei laufen. Beim Zugriff auf einige Spalten im Datensatz traten Fehler auf, die dazu führten, dass die Tests abgebrochen wurden.

Zur Fehlerbehebung wurden die Spalten übersprungen. Somit beläuft sich die Spaltenanzahl für die Tests auf circa 9.997 Datensätze. Die tatsächliche Laufzeit der Tests betrug circa vier Tage, also insgesamt 96 Stunden, was relativ ähnlich zur geplanten Zeit war.

Sobald die Tests abgeschlossen waren, wurden sie gesichert und ausgewertet. Das Ergebnis der Durchführung ist der Testergebnisdatensatz.

4.3 Ergebnisse und Diskussion

In diesem Abschnitt werden die Ergebnisse der Tests analysiert und diskutiert.

Zunächst wird ein Einzelfallbeispiel aus dem Testergebnisdatensatz vorgestellt. Das Einzelfallbeispiel soll ein besseres Verständnis des Aufbaus des resultierenden Testdatensatzes vermitteln.

Dann wird die deskriptive Statistik vorgestellt, um eine klare Struktur für die anschließenden Auswertungen zu schaffen. Ziel dieser statistischen Darstellung ist es, die gesammelten Daten zu ordnen und anschaulich darzustellen.

Darauf aufbauend werden in den Abschnitten, „Auswertung: Generatoren“, „Auswertung: Retriever“ und „Auswertung: RAG-Systeme“ die jeweiligen Systeme mithilfe der ermittelten Kennzahlen und Abbildungen ausgewertet. Im abschließenden Abschnitt, „Implikation“ werden die daraus ergebenden Konsequenzen und Ergebnisse präsentiert.

4.3.1 Einzelfallbeispiel: Testergebnisdatensatz

Zur Veranschaulichung der zuvor dargestellten Ergebnisse wird im Folgenden ein Einzelfallbeispiel vorgestellt. Zunächst wird ein zufälliger Punkt aus dem Testergebnisdatensatz genommen.

Konkret nehmen wir einen Datensatz aus den Ergebnissen für das RAG-System mit dem Generator „HuggingFaceH4-zephyr-7b-beta“ und dem all-mpnet-base-v2 als Embeddingmodell.

Danach wird der zufällige Punkt wie folgt ausgewertet: Zunächst wird die gestellte Frage an das RAG-System betrachtet. Die Frage lautet,

„What is the main goal of the CHI 2011 workshop on large interactive displays in public urban contexts?“.

Danach wird der gewählte Kontext betrachtet. Ein Abschnitt des gewählten Kontexts aus der Datengrundlage lautet,

„CHI 2010 Workshop May 7 or 8, 2011 (final date to be announced) for Participation ...“

Ein Abschnitt des gefundenen Kontexts aus dem Retriever lautet,

„CHI 2010 Workshop May 7 or 8, 2011 (final date to be announced) for Participation ...“

Aus den beiden Kontexten kann man erkennen, dass der Retriever den passenden Kontext zur Frage gefunden hat. Das heißt, der Exact-Match-Wert für diesen Fall wäre 1.

Abschließend werden die Antworten betrachtet. Die gewählte Antwort aus der Datengrundlage lautet,

„The main goal of this one-day CHI 2011 workshop is to cross-fertilize insights from different disciplines, to establish a more general understanding of large interactive displays in public urban contexts, and to develop an agenda for future research directions in this area.“

Die generierte Antwort mit dem passenden Kontext lautet,

„The main goal of the CHI 2011 workshop on large interactive displays in public urban contexts is to cross-fertilize insights from different disciplines, establish a more general understanding of large interactive displays in public urban settings, and develop an agenda for future research directions in this area. The workshop aims to trigger active and dynamic group discussions around topics such as designing installations that promote engaging experiences beyond playful interaction, exploring the impact of different interaction methods on people’s experience,“

Um die Präzision der generierten Antwort zu bestimmen, wird die generierte Antwort als Kandidatensatz und die gewählte Antwort als Referenzantwort genommen. Mit dem Kandidatensatz und der Referenzantwort lassen sich die verschiedenen Metriken berechnen. Die Berechnungen werden mit der in Kapitel 4.1.2 (Evaluationsmethoden) beschriebenen Methodik durchgeführt.

Die berechneten Ergebnisse werden dann aufgeschrieben. Sie lauten wie folgt:

- Durchschnitt: 0,66

- BERT-score: 0,80
- BLEU-score: 0,38
- F1-score: 0,62
- ROUGE-l-score: 0,60
- SBERT-score: 0,91

Bis auf den Durchschnitt und den SBERT-Score sind die Werte in Prozent angegeben. Der SBERT-Score, wie im Kapitel 2.2.9 (SBERT-Score) beschrieben, ist ein Wert zwischen -1 und 1; der Durchschnitt hingegen ist der Mittelwert zwischen den Metriken.

4.3.2 Deskriptive Statistik

Zur Auswertung der Testergebnisse wird zunächst die deskriptive Statistik berechnet [Illowsky und Dean, 2023a, Chapter 2]. Diese umfasst Mittelwerte [Illowsky und Dean, 2023b, Section 2.5] und Standardabweichungen [Illowsky und Dean, 2023c, Section 2.7]. Als grafische Darstellung der Daten werden Histogramme für das RAG-System und Sprachmodell mit den besten Mittelwerten erstellt [Illowsky und Dean, 2023d, Section 2.2].

Das Ziel der Auswertung besteht darin, zunächst einen Überblick über die allgemeinen Eigenschaften der untersuchten Sprachmodelle, RAG-Systeme und Embeddings zu erhalten.

Zur Berechnung der deskriptiven Statistik wird das Modul statistics der Python-Programmiersprache verwendet [Python Software Foundation, 2023].

Für die RAG-Systeme und jeweiligen Embeddingmodelle werden folgende Kennwerte analysiert:

- F1-Scores auf Wortebene (in Prozent).
- ROUGE-L-Scores (in Prozent).
- BLEU-Scores (in Prozent).
- BERT-Scores (in Prozent).
- SBERT-Scores (Wertebereich: -1 bis 1).
- Durchschnittswerte aller oben genannten Scores
- Exact-Matches (binär: 0 = falsch, 1 = korrekt) auf Basis der Embeddingmodelle.

Für die Sprachmodelle werden folgende Kennwerte analysiert:

- F1-Scores auf Wortebene (in Prozent).

- ROUGE-L-Scores (in Prozent).
- BLEU-Scores (in Prozent).
- BERT-Scores (in Prozent).
- SBERT-Scores (Wertebereich: -1 bis 1).
- Durchschnittswerte aller oben genannten Scores

Zunächst werden die Mittelwerte über den Ergebnisdatensatz für alle Systeme berechnet. Der Mittelwert dient hierbei als Indikator dafür, wie zuverlässig die RAG-Systeme im Durchschnitt relevante oder zielgerichtete Antworten erzeugen. Die berechneten Mittelwerte für die Systeme sind in den Tablen 4.1 und 4.2 dargestellt.

Tabelle 4.1: Deskriptive Statistik: RAG-Systeme Mittelwerte

Generator	Embedding	Durchschnitt	Bert	Bleu	F1	Rouge-L	SBert	Exactmatch
zephyr-7b-beta	all-MiniLM-L6-v2	0.52	0.72	0.24	0.45	0.43	0.77	0.64
zephyr-7b-beta	all-mpnet-base-v2	0.52	0.72	0.24	0.45	0.43	0.77	0.65
zephyr-7b-beta	multi-qa-distilbert-cos-v1	0.54	0.74	0.26	0.47	0.45	0.79	0.72
phi-4	all-MiniLM-L6-v2	0.61	0.79	0.34	0.54	0.54	0.81	0.69
phi-4	all-mpnet-base-v2	0.61	0.79	0.34	0.54	0.55	0.81	0.69
phi-4	multi-qa-distilbert-cos-v1	0.62	0.8	0.35	0.55	0.56	0.82	0.72
Qwen2.5-0.5B	all-MiniLM-L6-v2	0.45	0.66	0.18	0.36	0.35	0.67	0.64
Qwen2.5-0.5B	all-mpnet-base-v2	0.45	0.66	0.19	0.36	0.36	0.67	0.65
Qwen2.5-0.5B	multi-qa-distilbert-cos-v1	0.45	0.66	0.19	0.37	0.36	0.68	0.68
Qwen2.5-14B	all-MiniLM-L6-v2	0.58	0.77	0.33	0.52	0.51	0.8	0.69
Qwen2.5-14B	all-mpnet-base-v2	0.58	0.76	0.32	0.52	0.51	0.8	0.69
Qwen2.5-14B	multi-qa-distilbert-cos-v1	0.59	0.77	0.33	0.53	0.52	0.8	0.72
Qwen2.5-7B	all-MiniLM-L6-v2	0.56	0.75	0.3	0.49	0.48	0.78	0.69
Qwen2.5-7B	all-mpnet-base-v2	0.56	0.75	0.3	0.49	0.48	0.78	0.69
Qwen2.5-7B	multi-qa-distilbert-cos-v1	0.57	0.75	0.31	0.5	0.49	0.79	0.72

Tabelle 4.2: Deskriptive Statistik: Sprachmodelle Mittelwerte

Generator	Durchschnitt	Bert	Bleu	F1	Rouge-L	SBert
zephyr-7b-beta	0.43	0.71	0.12	0.33	0.31	0.7
phi-4	0.49	0.74	0.19	0.41	0.4	0.74
Qwen2.5-0.5B	0.41	0.65	0.13	0.32	0.3	0.66
Qwen2.5-14B	0.47	0.71	0.18	0.39	0.37	0.72
Qwen2.5-7B	0.46	0.68	0.17	0.37	0.35	0.72

Daraufhin werden die Standardabweichungen über den Ergebnisdatensatz für alle Systeme berechnet. Die Standardabweichungen dienen hierbei als Indikator dafür, wie stark die Werte um den Mittelwert variieren und geben einen genaueren Ausblick auf die Verteilung der erzielten Werte. Auf die Verwendung der Exact-Match-Metrik wird verzichtet, da es sich um

eine binäre Metrik handelt, die in der Regel keine weiterführenden Informationen über die Verteilung der Ergebnisse liefert. Die berechneten Standardabweichungen für die Systeme sind in den Tabellen 4.3 und 4.4 dargestellt.

Tabelle 4.3: Deskriptive Statistik: RAG-Systeme Standardabweichungen

Generator	Embedding	Durchschnitt	Bert	Bleu	F1	Rouge-L	SBert
zephyr-7b-beta	all-MiniLM-L6-v2	0.19	0.13	0.23	0.23	0.24	0.21
zephyr-7b-beta	all-mpnet-base-v2	0.19	0.13	0.23	0.23	0.24	0.2
zephyr-7b-beta	multi-qa-distilbert-cos-v1	0.19	0.13	0.24	0.23	0.24	0.2
phi-4	all-MiniLM-L6-v2	0.22	0.14	0.31	0.26	0.27	0.19
phi-4	all-mpnet-base-v2	0.22	0.14	0.3	0.26	0.27	0.2
phi-4	multi-qa-distilbert-cos-v1	0.22	0.14	0.31	0.26	0.27	0.19
Qwen2.5-0.5B	all-MiniLM-L6-v2	0.2	0.14	0.22	0.24	0.24	0.23
Qwen2.5-0.5B	all-mpnet-base-v2	0.2	0.14	0.23	0.24	0.24	0.24
Qwen2.5-0.5B	multi-qa-distilbert-cos-v1	0.2	0.14	0.23	0.24	0.24	0.23
Qwen2.5-14B	all-MiniLM-L6-v2	0.21	0.14	0.29	0.25	0.27	0.19
Qwen2.5-14B	all-mpnet-base-v2	0.21	0.14	0.29	0.25	0.27	0.19
Qwen2.5-14B	multi-qa-distilbert-cos-v1	0.21	0.14	0.29	0.25	0.27	0.19
Qwen2.5-7B	all-MiniLM-L6-v2	0.21	0.14	0.28	0.25	0.26	0.2
Qwen2.5-7B	all-mpnet-base-v2	0.21	0.14	0.28	0.25	0.26	0.2
Qwen2.5-7B	multi-qa-distilbert-cos-v1	0.21	0.14	0.28	0.25	0.26	0.19

Tabelle 4.4: Deskriptive Statistik: Generatoren Standardabweichungen

Generator	Durchschnitt	Bert	Bleu	F1	Rouge-L	SBert
zephyr-7b-beta	0.13	0.09	0.14	0.16	0.17	0.17
phi-4	0.16	0.11	0.22	0.21	0.22	0.17
Qwen2.5-0.5B	0.16	0.12	0.17	0.2	0.19	0.21
Qwen2.5-14B	0.17	0.13	0.21	0.21	0.21	0.18
Qwen2.5-7B	0.16	0.13	0.2	0.2	0.2	0.17

Abbildung 4.1 zeigt für das Sprachmodell Phi-4 die Verteilung der Häufigkeiten verschiedener erzielter Punktzahlen über verschiedene Metriken in mehreren Histogrammen. Die Histogramme sind in 20 unterschiedliche Klassen unterteilt, wobei jede Klasse durch einen Balken dargestellt wird. Die x-Achse beschreibt die erzielten Werte in verschiedenen Bereichen, während die y-Achse die Anzahl dieser Werte darstellt.

Abbildung 4.2 zeigt für das RAG-System Phi-4 mit dem Retriever multi-qa-distilbert-cos-v1 die Verteilung der Häufigkeiten verschiedener erzielter Punktzahlen über verschiedene Metriken in mehreren Histogrammen. Die Histogramme sind in 20 unterschiedliche Klassen [Illowsky und Dean, 2023d, Section 2.2] unterteilt, wobei jede Klasse durch einen Balken dargestellt wird. Die x-Achse beschreibt die erzielten Werte in verschiedenen Bereichen, während die y-Achse die Anzahl dieser Werte darstellt.

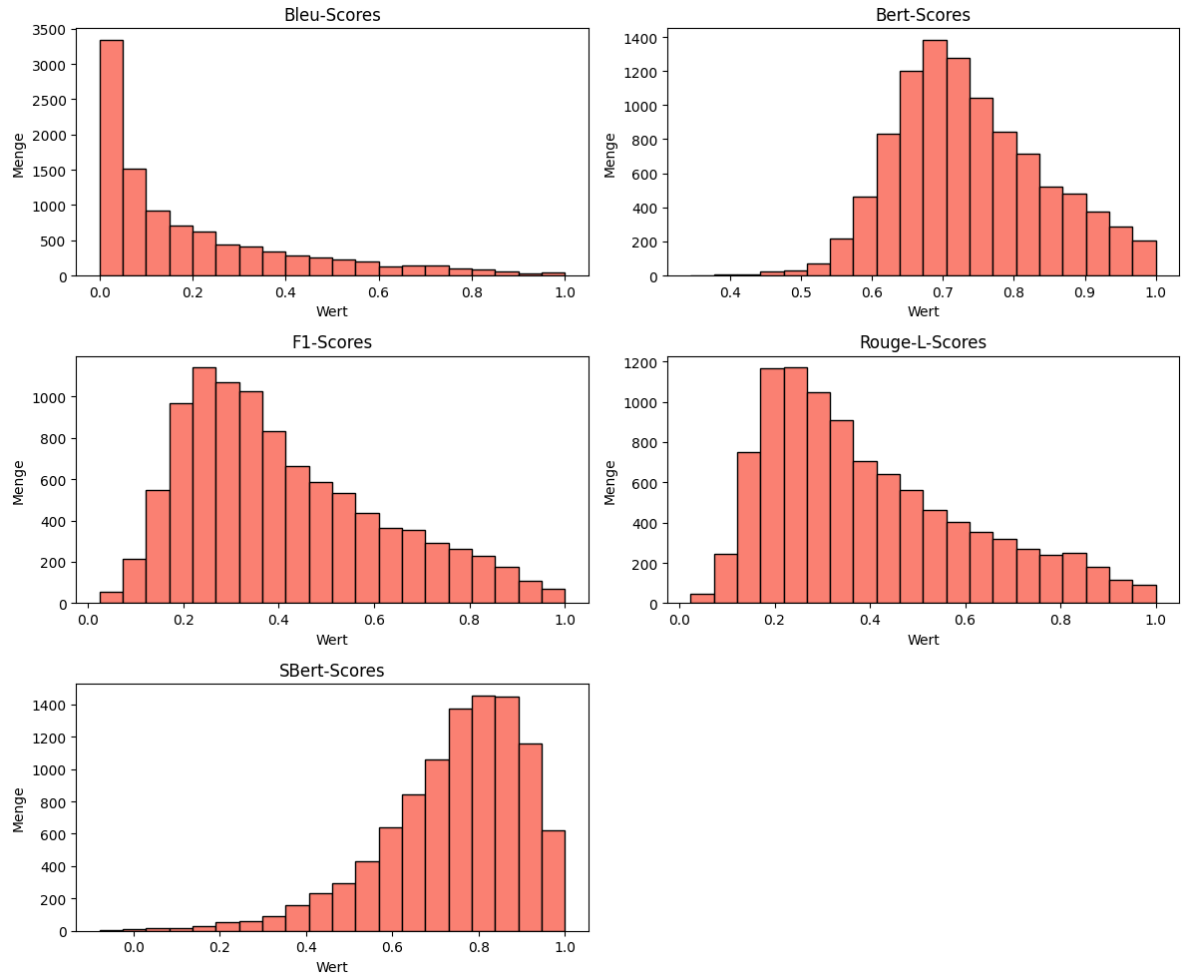

Abbildung 4.1: die Punkteverteilung der Metriken des Generators Phi-4 dargestellt durch mehrere Histogramme.

4.3.3 Auswertung: Sprachmodelle

Die Auswertung der Sprachmodelle unterteilt sich in zwei Teile. Zunächst werden die erzielten Mittelwerte betrachtet; das gibt einen Einblick, wie präzise die Systeme funktionieren. Dann werden die Standardabweichungen untersucht, um die Varianz der erzielten Mittelwerte darzustellen. Schließlich wird die Abbildung 4.1 ausgewertet. Die Interpretation der Werte erfolgt auf heuristischer Grundlage und erhebt keinen Anspruch auf eine Allgemeingültigkeit.

Zur Auswertung der Sprachmodelle werden zunächst die Mittelwerte (Tabelle 4.2) betrachtet.

Die durchschnittlichen Werte zwischen den Sprachmodellen liegen im Bereich von 0,41 bis 0,49. Basierend auf diesen Werten ergibt sich folgende Rangfolge:

1. Phi-4: 0,49

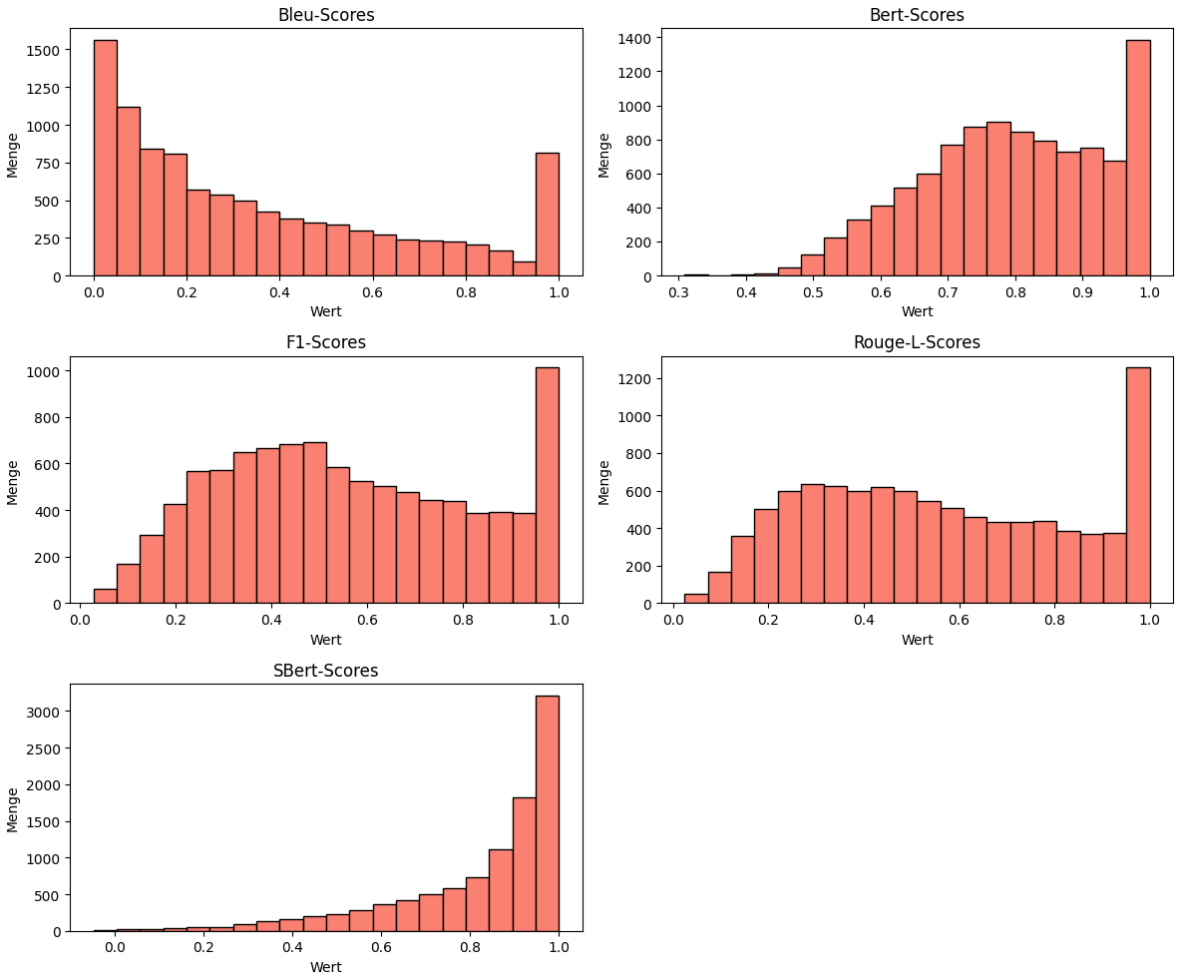

Abbildung 4.2: die Punkteverteilung der Metriken des RAG-System Phi-4 mit dem Retriever multi-qa-distilbert-cos-v1 dargestellt durch mehrere Histogramme.

2. Qwen2.5 mit 14 Billionen Parametern: 0,47
3. Qwen2.5 mit 7 Billionen Parametern: 0,46
4. zephyr-7b-beta: 0,43
5. Qwen2.5 mit 0,5 Billionen Parametern: 0,41

Die durchschnittliche Rangfolge gibt einen ersten Eindruck von der Verteilung und dem Verhältnis der Sprachmodelle zueinander.

Die F1-Score-Werte auf Wortebene der Sprachmodelle liegen bereich von 0,32 bis 0,41. Eine heuristische Analyse lässt darauf schließen, dass der durchschnittliche Anteil der gemeinsamen Wörter zwischen generierten Antworten und Referenzantworten als eher mittelmäßig bis gering zu betrachten ist.

Die BLEU-Score-Werte liegen im Bereich von 0,12 bis 0,19. Unter heuristischen Gesichtspunkten deutet dies darauf hin, dass die Übereinstimmung der n-Gramme zwischen generierten und Referenzantworten eher gering ist.

Die ROUGE-L-Werte zwischen den Sprachmodellen bewegen sich im Bereich von 0,30 bis 0,40. Eine heuristische Interpretation lässt darauf schließen, dass die Übereinstimmung der Satzstrukturen zwischen generierten und Referenzantworten insgesamt als mäßig einzustufen ist.

Die BERT-Score-Werte liegen im Bereich von 0,65 bis 0,74. Eine heuristische Analyse deutet darauf hin, dass die durchschnittliche semantische Ähnlichkeit zwischen den generierten Antworten und den Referenzantworten insgesamt als mäßig zu bewerten ist.

Die SBERT-Score-Werte liegen im Bereich von 0,66 bis 0,74. Es lässt sich heuristisch schlussfolgern, dass die durchschnittliche semantische Ähnlichkeit zwischen den generierten Antworten und den Referenzantworten mäßig ist.

Es zeigt sich, dass das Modell Phi-4 (mit 14 Billionen Parametern) über alle betrachteten Metriken hinweg die besten Mittelwerte erzielt hat

Konkret erzielt Phi-4 folgende Werte:

- Durchschnitt: 0,49
- Bert-Score: 0,74
- BLEU-Score: 0,19
- F1-Score: 0,41
- ROUGE-L: 0,40
- SBert-Score: 0,74

Die niedrigsten Mittelwerte wurden von Qwen2.5 mit 0,5 Billionen Parametern und Zephyr-7B-Beta erzielt.

Qwen2.5 mit 0,5 Billionen Parametern erzielte dabei folgende Werte:

- Durchschnitt: 0,49
- Bert-Score: 0,65
- F1-Score: 0,32
- ROUGE-L: 0,30
- SBert-Score: 0,74

Zephyr-7B-Beta erzielte einen BLEU-Score von 0,12.

Insgesamt zeigt sich, dass die durchschnittliche Rangfolge der meisten Sprachmodelle über die Metriken hinweg relativ konstant bleibt. Eine Ausnahme bilden jedoch Zephyr-7B-Beta und Qwen2.5 mit 7 Billionen Parametern, deren Positionen variieren. Beispielsweise teilen sich Zephyr-7B-Beta und Qwen2.5 mit 14 Billionen Parametern in Bezug auf den BERT-Score denselben Rang. Qwen2.5 erzielte zudem sowohl mit 7 als auch mit 14 Billionen Parametern im SBERT-Score denselben Rang.

Abschließend lässt sich festhalten, dass Phi-4 die besten Ergebnisse erzielte, während Qwen2.5 mit 0,5 Billionen Parametern die schwächste Leistung zeigte.

Nun wird die Tabelle 4.4 mit den Standardabweichungen betrachtet.

Die durchschnittlichen Werte zwischen den Sprachmodellen liegen im Bereich von 0,13 bis 0,17. Heuristisch wird die Variabilität um den Mittelwert als hoch eingeschätzt.

Die BERT-Werte zwischen den Sprachmodellen liegen im Bereich von 0,09 bis 0,13. Heuristisch ist die Variabilität bei den BERT-Werten als moderat bis hoch einzustufen.

Die BLEU-Werte zwischen den Sprachmodellen liegen im Bereich von 0,14 bis 0,22. Eine heuristische Bewertung stuft die Schwankungen bei den BLEU-Werten als hoch bis sehr hoch ein.

Die F1-Werte zwischen den Sprachmodellen liegen im Bereich von 0,16 bis 0,21. Eine heuristische Einschätzung stuft die Schwankungen der F1-Ergebnisse als hoch bis sehr hoch ein.

Die ROUGE-L-Werte zwischen den Sprachmodellen liegen im Bereich von 0,17 bis 0,22. Die heuristische Betrachtung sieht die Schwankungen um den Mittelwert der ROUGE-L-Messwerte als hoch bis sehr hoch an

Die SBERT-Werte zwischen den Sprachmodellen liegen im Bereich von 0,17 bis 0,21. Eine heuristische Einschätzung zeigt eine hohe bis sehr hohe Schwankung um den Mittelwert an.

Insgesamt zeigt sich eine Hohe Variabilität über die Metriken.

Abschließend wird die Abbildung 4.1 betrachtet, die die Werteverteilung für das Sprachmodell Phi-4 als Histogramm zeigt.

Der BLEU-Score weist die niedrigsten Werte auf. Im Vergleich zum Mittelwert von 0,19 sieht man hier, dass die Werte im höheren Bereich den Mittelwert nach oben ziehen. Tatsächlich befinden sich die meisten Werte zwischen 0 und etwa 0,05; dieser Bereich stellt auch den Modus des Histogramms dar, der ungefähr 3.300 Werte umfasst [Illowsky und Dean (2023d, Abschnitt 2.2)].

Die Verteilung der Werte für F1- und ROUGE-L-Score ist am höchsten im Bereich von 0,20 bis 0,39. Der Modus für den F1-Score sowie für den ROUGE-L-Score liegt etwa zwischen 0,21 und 0,25.

Für den BERT-Score liegt der Wert ganz klar am höchsten im Bereich von 0,60 bis 0,79. Der Modus für den BERT-Score liegt etwa zwischen 0,65 und 0,7.

Für den SBERT-Score sind die Werte am häufigsten verteilt im Bereich von 0,60 bis 1,0. Der Modus für den SBERT-Score liegt etwa zwischen 0,79 und 0,84.

4.3.4 Auswertung: Retriever

Der Retriever entscheidet grundlegend darüber, ob ein RAG-System den passenden Kontext zur Eingabe findet. Dies kann einen enormen Einfluss auf die Qualität der generierten Antworten haben.

Die Exact-Match-Werte für die Effizienz der Retriever in der Mittelwert-Tabelle 4.1 zeigen, dass die Ergebnisse der Retriever variieren.

Die Analyse zeigt, dass die höchsten Mittelwerte in der Exact-Match-Metrik bei den einzelnen Retrievern wie folgt ausfielen:

- all-MiniLM-L6-v2: 0,69
- all-mpnet-base-v2: 0,69
- multi-qa-distilbert-cos-v1: 0,72

Die niedrigsten Mittelwerte der Exact-Match-Metrik der verschiedenen Retriever waren:

- all-MiniLM-L6-v2: 0,64
- all-mpnet-base-v2: 0,65
- multi-qa-distilbert-cos-v1: 0,68

Die Ergebnisse zeigen, dass Retriever nicht immer konsistent den gleichen Kontext zur gleichen Eingabe finden.

Abschließend lässt sich beobachten, dass das Modell „multi-qa-distilbert-cos-v1“ relativ konstant die höchsten Werte lieferte, wohingegen „all-MiniLM-L6-v2“ am schlechtesten abschnitt.

4.3.5 Auswertung: RAG-Systeme

Die Auswertung der RAG-Systeme unterteilt sich in drei Teile. Zunächst werden die erzielten Mittelwerte betrachtet; das gibt einen Einblick, wie präzise die Systeme funktionieren. Zusätzlich werden die Mittelwerte mit den Ergebnissen der Sprachmodelle verglichen. Dan werden die Standardabweichungen berechnet, um die Varianz der erzielten Werte darzustellen. Zum Schluss wird die Abbildung 4.2 ausgewertet.

Es werden ausschließlich Systeme mit dem Embedding-Modell multi-qa-distilbert-cos-v1 betrachtet, da dieses die höchsten Mittelwerte in der Exact-Match-Metrik erzielte. Die Interpretation der Werte erfolgt auf heuristischer Grundlage und erhebt keinen Anspruch auf eine Allgemeingültigkeit.

Zur Auswertung der RAG-Systeme wird zunächst die Mittelwert-Tabelle 4.1 betrachtet.

Die durchschnittlichen Werte zwischen den RAG-Systemen liegen im Bereich von 0,45 bis 0,62. Basierend auf diesen Werten ergibt sich folgende Rangfolge:

1. Phi-4: 0,62
2. Qwen2.5 mit 14 Billionen Parametern: 0,59
3. Qwen2.5 mit 7 Billionen Parametern: 0,57
4. zephyr-7b-beta: 0,54
5. Qwen2.5 mit 0,5 Billionen Parametern: 0,45

Die durchschnittliche Rangfolge ermöglicht einen ersten Überblick über die Verteilung und das Verhältnis der einzelnen Metriken zueinander. Darüber hinaus zeigt der Vergleich mit den Sprachmodellen, dass sich die durchschnittlichen Werte bei allen Systemen verbessert haben. Beispielsweise hat sich im Vergleich Phi-4 in einem RAG-System um einen Wert von 0,13 im Durchschnitt verbessert. Die Rangfolge im Vergleich zu den Sprachmodellen bleibt gleich.

Die F1-Score-Werte auf Wortebene der RAG-Systeme liegen bereich von 0,37 bis 0,55. Dies weist heuristisch darauf hin, dass der durchschnittliche Anteil der überlappenden Wörter zwischen den generierten Antworten und Referenzantworten insgesamt eher mäßig bis gering einzustufen ist.

Die BLEU-Score-Werte liegen im Bereich von 0,19 bis 0,35. Das deutet heuristisch auf eine geringe bis zufriedenstellende Übereinstimmung der n-Gramme zwischen den generierten Antworten und den Referenzantworten hin.

Die Rouge-L-Werte zwischen den RAG-Systeme bewegen sich im Bereich von 0,36 bis 0,56. Dies weist heuristisch auf ein insgesamt moderat ausgeprägtes bis zufriedenstellendes Maß

an Übereinstimmung der Satzstrukturen zwischen den generierten Antworten und den Referenzantworten hin.

Die BERT-Score-Werte liegen im Bereich von 0,66 bis 0,8. Heuristisch lässt sich daraus eine insgesamt mäßige bis solide durchschnittliche semantische Ähnlichkeit zwischen den generierten und den Referenzantworten ableiten.

Die SBERT-Score-Werte bewegen sich zwischen 0,68 und 0,82. Heuristisch betrachtet, weist dies auf eine im Durchschnitt eher stabile semantische Übereinstimmung der generierten Antworten mit den Referenzantworten hin.

Es zeigt sich, dass die RAG-Systeme im Allgemeinen bessere Werte über die Metriken hinweg erzielt haben als die reinen Sprachmodelle.

Auch zeigt sich, dass das RAG-System mit Phi-4 (14 Billionen Parameter) über alle betrachteten Metriken hinweg die besten Mittelwerte erzielt hat.

Konkret erzielt RAG-System mit Phi-4 folgende Werte:

- Durchschnitt: 0,62
- Bert-Score: 0,8
- BLEU-Score: 0,35
- F1-Score: 0,55
- ROUGE-L: 0,56
- SBert-Score: 0,82

Abschließend lässt sich Beobachten, dass die durchschnittliche Rangfolge mit dem Embeddingmodell „multi-qa-distilbert-cos-v1“ über alle RAG-Systeme gleich geblieben ist.

Nun wird die Tabelle 4.3 mit den Standardabweichungen betrachtet.

Die durchschnittlichen Werte der RAG-Systeme bewegen sich zwischen 0,19 und 0,22. Dies lässt heuristisch auf eine sehr starke Schwankung rund um den Mittelwert schließen.

Für die BERT-Werte liegt der Bereich bei 0,13 bis 0,14. Dies weist heuristisch auf eine deutliche Variabilität um den Mittelwert hin.

Die BLEU-Werte verteilen sich zwischen 0,22 und 0,27. Das deutet heuristisch auf eine von hoch bis sehr hoch reichende Streuung um den Mittelwert hin.

Die F1-Werte der RAG-Systeme liegen zwischen 0,23 und 0,26. Dies spricht heuristisch für eine hohe bis sehr ausgeprägte Variabilität rund um den Mittelwert.

Die ROUGE-L-Werte bewegen sich im Bereich von 0,24 bis 0,27. Das legt heuristisch eine hohe bis sehr hohe Streuung um den Mittelwert nahe.

Die SBERT-Werte liegen zwischen 0,19 und 0,21. Dies deutet heuristisch auf eine deutliche bis sehr deutliche Varianz um den Mittelwert hin.

Im Allgemeinen wird im Vergleich zu den reinen Sprachmodellen eine allgemeine Erhöhung der Standardabweichungen beobachtet.

Abschließend wird Abbildung 4.2 betrachtet. Die Abbildung zeigt mehrere Histogramme für verschiedene Metriken des RAG-Systems mit dem Generator Phi-4 und dem Embeddingmodell „multi-qa-distilbert-cos-v1“.

Zunächst fällt eine sehr interessante Beobachtung auf: Bei jedem Histogramm gibt es einen deutlichen Anstieg in der letzten Klasse im Vergleich zu Abbildung 4.1. Das Könnte auch die erhöhte Standardabweichungen über die Metriken hinweg erklären. Da der Wertebereich der letzten Klasse weiter von den meisten Mittelwerten entfernt liegt, beeinflusst er die Berechnung der Standardabweichung entsprechend stärker.

Der Bleu-Score weist die niedrigsten Werte auf. Im Vergleich zum Mittelwert von 0,19 sieht man hier, dass die Werte im höheren Bereich den Mittelwert nach oben ziehen. Tatsächlich befinden sich die meisten Werte zwischen 0 und etwa 0,05; dieser Bereich stellt auch den Modus des Histogramms dar, der ungefähr 1.500 Werte umfasst [Illowsky und Dean, 2023d, Section 2.2]. Im Vergleich zur Abbildung 4.1 zeigt sich hier eine deutliche Reduktion im Modus sowie eine stärkere Verteilung über die Klassen hinweg. Es zeigt sich auch ein stark erhöhter Wert in der letzten Klasse im Bereich von 0,95 bis 1,0.

Der Modus für alle folgenden Metriken befindet sich im Bereich von 0,95 bis 1,0. Bis auf die SBERT-Scores befinden sich im Modus der Histogramme etwa zehn Prozent der Werte. Bei den SBERT-Scores sind es sogar dreißig Prozent. Abschließend lässt sich allgemein feststellen, dass die Verteilung der erzielten Werte für diese Metriken größer ist als in Abbildung 4.1. Dies stellt ein weiteres Argument für die erhöhte Standardabweichung bei den meisten Metriken dar.

4.3.6 Diskussion

Das Kapitel widmet sich den Implikationen, die aus den resultierenden Forschungsergebnissen hervorgehen, und diskutiert deren Bedeutung im weiteren Kontext.

Mit den Ergebnissen lässt sich zeigen, dass Sprachmodelle in der praktischen Anwendung in RAG-Systemen um einiges besser abschneiden. Das impliziert, dass die Erweiterung von Generatoren um einen Retriever die Leistung der Generatoren Steigern kann. Es kann auch vermutet werden, dass RAG-Systeme aufgrund der höheren Präzision seltener falsche

oder irreführende Antworten generieren. Implikativ lässt sich auch sagen, dass Modelle mit einer höheren Parameteranzahl am meisten von der Erweiterung profitieren. Beispielsweise zeigte Phi-4 mit 14 Billionen Parametern ein erhöhtes Potenzial bei der Beantwortung von kontextspezifischen Fragen, während Qwen2.5 mit 0,5 Billionen Parametern nur ein geringes erhöhtes Potenzial zeigte.

Einer der interessantesten Beobachtungen war der Modus [Illowsky und Dean, 2023d, Section 2.2] der Histogramme für die meisten Metriken in Abbildung 4.2. Es wurde festgestellt, dass sich in den meisten dieser Histogramme der Modus in der letzten Klasse befindet. Der Modus macht hierbei etwa zehn Prozent der einzelnen Wertemengen aus. Dies deutet sogar stark darauf hin, dass mindestens zehn Prozent der Fragen vom RAG-System Phi-4 mit dem Embeddingmodell „multi-qa-distilbert-cos-v1“ korrekt beantwortet wurden.

Überraschend war, dass die Generatoren ohne Retriever in einigen Metriken mittelmäßige Ergebnisse erzielten. Die Erwartung war eher, dass die Ergebnisse der Generatoren deutlich schlechter abschneiden würden als die der RAG-Systeme. Ein Grund für die mäßigen Werte der Generatoren ohne Retriever könnte sein, dass die erwarteten Antworten aus der Datengrundlage mit dem GPT-4 LLM [OpenAI, 2023] Modell generiert worden sind. Es könnte angenommen werden, dass die Sprachmodelle grundsätzlich ähnliche Antworten wie GPT-4 generieren. Ein möglicher Grund dafür ist, dass sich die Trainingsdaten der verschiedenen Modelle überschneiden. Eine weitere Möglichkeit wäre, dass Teile der Datengrundlage der verschiedenen Modelle für das Training über das Modell GPT-4 generiert sind. In diesem Kontext wäre es denkbar, Frage-Antwort-Paare mithilfe eines Modells zu erzeugen und diese anschließend für das Training anderer Modelle zu nutzen. Der damit verbundene Aufwand wäre nicht groß, da man den Prozess der Generierung automatisieren könnte. Dadurch könnten die Modelle ähnliche Strukturen bei der Generierung von Antworten aufweisen.

Zudem ist die nicht ganz deterministische Art der Embedding-Modelle hervorgetreten. Bei der Auswertung der Embeddings ist aufgefallen, dass Embedding-Modelle im Durchschnitt oft verschiedene Werte liefern. Dies lässt Metriken wie den SBERT-Score und den BERT-Score weniger zuverlässig erscheinen, da sie auf diesen nicht ganz deterministischen Embeddingmodellen basieren. Bei der Betrachtung der Rangfolge der Systeme in Bezug auf embeddingbasierten Metriken wird deutlich, dass die Rangfolgen recht ähnlich zu denjenigen sind, die keine Embedding-Modelle implementieren. Daraus kann geschlossen werden, dass Metriken mit Embedding-Modellen trotzdem brauchbare Ergebnisse liefern.

Es wurde auch festgestellt, dass die Metriken bei Vergleichen oft ein ähnliches Potenzial zwischen den Systemen zeigten, insbesondere in den Tabellen mit den Mittelwerten. Dies legt nahe, dass häufig nur wenige Metriken nötig sind, um RAG-Systeme und ihre Sprachmodelle voneinander zu unterscheiden. Für diesen Zweck bietet sich eine Kombination aus F1-Score und ROUGE-L an, da die erstere allgemeine Wortüberlappungen misst, während die letztere

die längste gemeinsame Teilfolge misst und somit auch die Wortordnung analysiert. Auch hat es den Vorteil, dass die Werte deterministisch sind und somit weniger zufällig.

Abschließend lässt sich sagen, dass RAG-Systeme Potenzial für solide Frage-Antwort-Systeme bieten. Dies zeigte die Kombination aus dem Generator Phi-4 in Kombination mit dem Retriever „multi-qa-distilbert-cos-v1“, die über alle Metriken hinweg solide Ergebnisse erzielten.

5 Fazit

In der vorliegenden Arbeit wurde untersucht, inwieweit ein RAG-System ein auf einem neuronalen Netzwerk basierendes Sprachmodell erweitern kann. Die Analyse der theoretischen Grundlagen sowie der analytischen Fallstudie hat gezeigt, dass die Präzision von Sprachmodellen durch RAG-Systeme gesteigert werden kann, insbesondere bei bereits soliden Sprachmodellen und Embeddingmodellen.

Zudem wurde untersucht, welche Methoden es gibt, um Sprachmodelle sowie RAG-Systeme auf ihre Präzision zu prüfen. Die theoretischen Grundlagen zeigen, dass es viele Metriken gibt, mit denen sich generierte Texte aus Sprachmodellen und RAG-Systemen mit einer gewählten Antwortreferenz vergleichen lassen. Anhand der Übereinstimmung lässt sich erkennen, wie präzise diese Systeme arbeiten.

Die analytische Fallstudie zeigte auch, dass sich die Metriken autonom gestalten lassen. Das hat den Vorteil, dass man mit einer vernünftig starken Rechenmaschine ohne großen Aufwand Tests mit den erarbeiteten Metriken durchführen kann. Zusätzlich ist man nicht komplett auf eine „händische Auswertung“ angewiesen.

Die Leitfrage kann somit bejaht werden. Mit verschiedenen Metriken lassen sich RAG-Systeme und Sprachmodelle automatisch bewerten. Zudem wurde die Zielsetzung erfüllt, RAG-Systeme und Sprachmodelle aufzubauen und auszuwerten. Durch diese Erkenntnisse lässt sich außerdem schließen, dass die Präzision von Sprachmodellen steigt, wenn sie in ein RAG-System integriert werden.

Die Forschung hat gezeigt, dass der Hauptschwerpunkt der Arbeit sehr umfangreich und komplex ist. Einerseits basieren moderne RAG-Systeme auf Sprachmodellen und Embeddingmodellen, die bereits viele Implementierungen unterstützen. Andererseits beruht das Thema auf jahrzehntelanger Forschung, an der zahlreiche Personen mitgewirkt haben. Dies macht den detaillierten Einstieg in das Thema nicht gerade einfach. Darüber hinaus machte es schwierig, die Funktionsweise der Metriken nachzuvollziehen, da diese nur mit guten mathematischen Vorkenntnissen nachvollzogen werden können.

Da die Forschung sich primär mit der Präzision der Systeme befasst hatte, lässt sich nicht darauf schließen, ob die Systeme die Fragen aus dem Datensatz auch korrekt beantwortet haben. Mit einer Präzisionsangabe kann nur impliziert werden, inwiefern die Systeme die

Fragen korrekt beantwortet haben. Aufbauend auf der Forschung könnte eine nachfolgende Frage sein: „Inwieweit beantworten RAG-Systeme Fragen mit passenden Kontexten richtig?“. Spannend wäre hierbei die Frage, ob sich das Thema nur durch ein manuelles Vorgehen beantworten lässt oder ob man die Auswertung beispielsweise auf Basis verschiedener Modelle automatisieren kann.

Abschließend lässt sich festhalten, dass die Ergebnisse dieser Arbeit implizieren, dass die Anwendung der RAG-Systeme sowohl für Privatpersonen als auch für Unternehmen sinnvoll sein kann. Auch lässt sich sagen, dass die Ergebnisse dieser Arbeit einen Beitrag zum besseren Verständnis von RAG-Systemen ermöglichen und Ansatzpunkte für weiterführende Forschung bieten.

Literatur

AI, N. B. (2023). Retrieval-Augmented Generation (RAG) Dataset 12000 [Accessed: 2025-04-01]. https://huggingface.co/datasets/neural-bridge/rag-dataset-12000

Bai, Y., et al. (2023). Phi-4 Technical Report. *arXiv preprint arXiv:2412.08905*. https://doi.org/10.48550/arXiv.2412.08905

Banerjee, S., Agarwal, A., & Singla, S. (2024). Llms will always hallucinate, and we need to live with this. *arXiv preprint arXiv:2409.05746*. https://doi.org/10.48550/arXiv.2409.05746

Bray, T. (2017, Dezember). *The JavaScript Object Notation (JSON) Data Interchange Format* (Techn. Ber.) (Accessed: 2025-03-22). https://doi.org/10.17487/RFC8259

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*. https://doi.org/10.48550/arXiv.2005.14165

Camuñas-Mesa, L., Linares-Barranco, B., & Serrano-Gotarredona, T. (2019). Neuromorphic Spiking Neural Networks and Their Memristor-CMOS Hardware Implementations. *Materials*, *12*, 2745. https://doi.org/10.3390/ma12172745

Chroma. (2025). *Chroma: AI-native open-source embedding database* [Accessed: 2025-03-22]. https://www.trychroma.com/

CHURCH, K. W. (2017). Word2Vec. *Natural Language Engineering*, *23*(1), 155–162. https://doi.org/10.1017/S1351324916000334

Creswell, J. W., & Creswell, J. D. (2017). *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019, Juni). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In J. Burstein, C. Doran & T. Solorio (Hrsg.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (S. 4171–4186). Association for Computational Linguistics. https://doi.org/10.18653/v1/N19-1423

Docker, Inc. (2024). *Docker Documentation* [Accessed: 2025-04-07]. Docker. https://docs.docker.com

Google AI. (2024). rouge-score: A Python implementation of ROUGE [Accessed: 2025-04-03]. https://pypi.org/project/rouge-score/

Han, Y., Liu, C., & Wang, P. (2023). A comprehensive survey on vector database: Storage and retrieval technique, challenge. *arXiv preprint arXiv:2310.11703*. https://doi.org/10.48550/arXiv.2310.11703

He, P., Liu, X., Gao, J., & Chen, W. (2020). Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*. https://doi.org/10.48550/arXiv.2006.03654

Illowsky, B., & Dean, S. (2023a). Introductory Statistics 2e [Chapter 2: Descriptive Statistics]. OpenStax. https://openstax.org/details/books/introductory-statistics-2e

Illowsky, B., & Dean, S. (2023b). Introductory Statistics 2e [Section 2.5: Measures of the Center of the Data (Median)]. OpenStax. https://openstax.org/details/books/introductory-statistics-2e

Illowsky, B., & Dean, S. (2023c). Introductory Statistics 2e [Section 2.7: Measures of the Spread of the Data (Standard Deviation)]. OpenStax. https://openstax.org/details/books/introductory-statistics-2e

Illowsky, B., & Dean, S. (2023d). Introductory Statistics 2e [Section 2.2: Histograms, Frequency Polygons, and Time Series Graphs]. OpenStax. https://openstax.org/details/books/introductory-statistics-2e

Jurafsky, D., & Martin, J. H. (2009). *Speech and Language Processing*. Prentice Hall. https://web.stanford.edu/~jurafsky/slp3/

Kent, A., Berry, M. M., Luehrs, F. U., & Perry, J. W. (1955). Machine literature searching VIII. Operational criteria for designing information retrieval systems. *American Documentation*, *6*(2), 93–101. https://doi.org/10.1002/asi.5090060209

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, *33*, 9459–9474. https://doi.org/10.48550/arXiv.2005.11401

Lin, C.-Y. (2004). ROUGE: A Package for Automatic Evaluation of Summaries. *Text Summarization Branches Out*, 74–81. https://aclanthology.org/W04-1013/

McCulloch, W. S., & Pitts, W. (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity (5. Aufl.). *Bulletin of Mathematical Biophysics*, 115–133. https://doi.org/10.1007/BF02478259

Microsoft. (2021). microsoft/deberta-xlarge-mnli [Accessed: 2025-04-04]. https://huggingface.co/microsoft/deberta-xlarge-mnli

Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., & Mian, A. (2023). A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*. https://doi.org/10.48550/arXiv.2307.06435

NVIDIA Corporation. (2020). NVIDIA A100 Tensor Core GPU Architecture [Accessed: 2025-04-07]. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf

NVIDIA Corporation. (2024). *CUDA Toolkit Documentation* [Accessed: 2025-04-07]. NVIDIA. https://docs.nvidia.com/cuda/

OpenAI. (2023). GPT-4 [Accessed: 2025-04-01]. https://openai.com/research/gpt-4

Papineni, K., Roukos, S., Ward, T., & Zhu, W.-J. (2002, Juli). Bleu: a Method for Automatic Evaluation of Machine Translation. In P. Isabelle, E. Charniak & D. Lin (Hrsg.), *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics* (S. 311–318). Association for Computational Linguistics. https://doi.org/10.3115/1073083.1073135

Papineni, K., Zhu, W.-J., Roukos, S., Ward, T., & Cherry, C. (2018). SacreBLEU: A Standardized BLEU Score Implementation [Version 1.0.0; Accessed: 2025-04-03]. https://pypi.org/project/sacrebleu/1.0.0/

Peeperkorn, M., Kouwenhoven, T., Brown, D., & Jordanous, A. (2024). Is temperature the creativity parameter of large language models? *arXiv preprint arXiv:2405.00492*. https://doi.org/10.48550/arXiv.2405.00492

Penedo, G., Malartic, Q., Hesslow, D., Cojocaru, R., Cappelli, A., Alobeidli, H., Pannier, B., Almazrouei, E., & Launay, J. (2023). The RefinedWeb dataset for Falcon LLM: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*. https://doi.org/10.48550/arXiv.2306.01116

Pennington, J., Socher, R., & Manning, C. (2014, Oktober). GloVe: Global Vectors for Word Representation. In A. Moschitti, B. Pang & W. Daelemans (Hrsg.), *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (S. 1532–1543). Association for Computational Linguistics. https://doi.org/10.3115/v1/D14-1162

Post, M. (2018). A call for clarity in reporting BLEU scores. *arXiv preprint arXiv:1804.08771*. https://doi.org/10.48550/arXiv.1804.08771

Python Software Foundation. (2023). *Python statistics module (Standard Library)* [Accessed: 2025-03-22]. https://docs.python.org/3/library/statistics.html

Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*. https://doi.org/10.48550/arXiv.1606.05250

Reimers, N., & Gurevych, I. (2019a). sentence-transformers Python Library [Version 2.5.1, Accessed: 2025-04-04]. https://pypi.org/project/sentence-transformers/

Reimers, N., & Gurevych, I. (2019b). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks (K. Inui, J. Jiang, V. Ng & X. Wan, Hrsg.), 3982–3992. https://doi.org/10.18653/v1/D19-1410

Reimers, N., & Gurevych, I. (2021). multi-qa-distilbert-cos-v1 [Accessed: 2025-04-05]. https://huggingface.co/sentence-transformers/multi-qa-distilbert-cos-v1

Reimers, N., & Gurevych, I. (2025). Pretrained Models [Accessed: 2025-04-04]. https://www.sbert.net/docs/sentence_transformer/pretrained_models.html

Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, *65*(6), 386–408. https://doi.org/10.1037/h0042519

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*. https://doi.org/10.1038/323533a0

Schütze, H., Manning, C. D., & Raghavan, P. (2008). *Introduction to information retrieval* (Bd. 39) [Scahue Kapitel 8, Sektion 8.2 für F1 score]. Cambridge University Press Cambridge.

Simonite, T. (2022). The Future of the Web Is Marketing Copy Generated by Algorithms [Accessed: 2025-05-06]. *WIRED*. https://www.wired.com/story/ai-generated-marketing-content/

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., & Potts, C. (2013, Oktober). Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In D. Yarowsky, T. Baldwin, A. Korhonen, K. Livescu & S. Bethard (Hrsg.), *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (S. 1631–1642). Association for Computational Linguistics. https://aclanthology.org/D13-1170/

Song, K., Tan, X., Qin, T., Lu, J., & Liu, T.-Y. (2020). Mpnet: Masked and permuted pre-training for language understanding. https://doi.org/https://doi.org/10.48550/arXiv.2004.09297

Tunstall, L., Beeching, E., Lambert, N., Rajani, N., Rasul, K., Belkada, Y., Huang, S., Von Werra, L., Fourrier, C., Habib, N., et al. (2023). Zephyr: Direct distillation of lm alignment. *arXiv preprint arXiv:2310.16944*. https://doi.org/10.48550/arXiv.2310.16944

Van Rijsbergen, C. J. (1979). *Information Retrieval, 2nd edn. Newton, MA* [Schaue Kapitel 7]. USA: Butterworth-Heinemann.

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace. https://doi.org/10.5555/1593511

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention Is All You Need. *arXiv preprint arXiv:1706.03762*. https://doi.org/10.48550/arXiv.1706.03762

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., & Zhou, M. (2020). Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *arXiv preprint arXiv.2002.10957*. https://doi.org/10.48550/arXiv.2002.10957

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., … Rush, A. M. (2020). Transformers: State-of-the-Art Natural Language Processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6

Wuttke, L. (2023, 22. Dezember). *Was ist Retrieval Augmented Generation (RAG)?* Datasolut GmbH. Verfügbar 20. März 2025 unter https://datasolut.com/was-ist-retrieval-augemented-generation/

Xu, Z., Jain, S., & Kankanhalli, M. (2024). Hallucination is inevitable: An innate limitation of large language models. *arXiv preprint arXiv:2401.11817*. https://doi.org/10.48550/arXiv.2401.11817

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al. (2024). Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*. https://doi.org/10.48550/arXiv.2412.15115

Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., & Artzi, Y. (n. d.). BERTScore [Accessed: 2025-04-04]. https://pypi.org/project/bert-score/

Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., & Artzi, Y. (2019). Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*. https://doi.org/10.48550/arXiv.1904.09675

Zong, M., & Krishnamachari, B. (2022). A survey on GPT-3. *arXiv preprint arXiv:2212.00857*. https://doi.org/10.48550/arXiv.2212.00857

Eigenständigkeitserklärung

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

Methoden zur Evaluierung von RAG-Systemen und Analyse der Grenzen, Faktoren und Implikationen für den praktischen Einsatz

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, 22. Mai 2025