

# **Entwicklung und Evaluation eines LLMs zur automatisierten Bewertung von Requirements**

## **Masterprojekt**

In Kooperation mit Spicy SE

Vorgelegt von

Benedikt Horn, 300889  
Sascha Tauchmann, 306432

Betreuer

Prof. Dr.-Ing. Christopher Knievel

eingereicht am

27.02.2025

# Eigenständigkeitserklärung

Wir versichern hiermit, dass wir dieses Masterprojekt mit dem Thema:

**Entwicklung und Evaluation eines LLMs  
zur automatisierten Bewertung von Requirements**

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Konstanz, 27.02.2025

B. Horn

---

Benedikt Horn

S. Tauchmann

---

Sascha Tauchmann

# Abstract

Die Qualität von Requirements stellt einen entscheidenden Faktor für den Erfolg von Software- und Systementwicklungsprojekten dar. Eine manuelle Bewertung dieser Anforderungen ist jedoch zeitaufwendig und arbeitsintensiv. In diesem Projekt wird untersucht, inwiefern Large Language Models (LLMs) eine Automatisierung dieser Evaluierung ermöglichen können.

Dazu wurden zwei unterschiedliche Bewertungsansätze entwickelt: ein sukzessiver Ansatz, bei dem alle formalen Qualitätskriterien in einem Schritt bewertet werden, und ein iterativer Ansatz, der jede Metrik einzeln beurteilt. Beide Methoden wurden mit verschiedenen LLMs getestet, darunter kleinere, lokal ausführbare Modelle sowie leistungsfähigere Modelle, die der Benchmark zur Zeit der Bearbeitung des Projektes entsprachen. Die Evaluierung erfolgte dabei anhand zuvor definierter Metriken wie *Correctness*, *Unambiguity* und *Consistency*, welche mit einer Ratings-Skala versehen wurden.

Zur Bewertung der Performance der jeweiligen Ansätze wurden sowohl automatisierte Methoden (*LLM as a Judge*) als auch manuelle Expertenbewertungen (*Human as a Judge*) herangezogen. Hieraus ergab sich das Bild, dass der sukzessive Ansatz strengere und über Modellgrenzen hinweg konsistentere Bewertungen liefert, während der iterative Ansatz eine differenziertere Analyse ermöglicht. Zudem erwies sich der Einsatz leistungsfähigerer LLMs in beiden Ansätzen als vorteilhaft.

Das Projekt liefert wertvolle Erkenntnisse zur Nutzung von LLMs im Requirements Engineering und legt gleichermaßen die Grundlage für zukünftige Weiterentwicklungen, darunter dynamische Prompt-Optimierungen mithilfe eines Templates und die Verwendung mehrerer Modelle, die miteinander interagieren und so ein ganzheitliches Bild liefern.

# Inhaltsverzeichnis

Abbildungsverzeichnis.....	II
Quellcodeverzeichnis.....	III
Abkürzungsverzeichnis.....	IV
1 Einführung.....	1
2 Theoretische Grundlagen.....	2
3 Entwicklungsprozess .....	6
3.1 Sprachmodelle.....	6
3.2 Datensätze .....	7
3.3 Prompt Engineering.....	8
3.3.1 Ansätze.....	8
3.3.2 Rating-Skalierung .....	9
3.3.3 Bestandteile.....	10
3.3.4 Sukzessiv.....	11
3.3.5 Iterativ .....	13
3.4 Performance-Evaluierung.....	14
3.4.1 LLM as a Judge.....	14
3.4.2 Human as a Judge.....	15
4 Ergebnisse.....	16
4.1 Quality Separation Index.....	16
4.2 Cross Model Consistency .....	18
4.3 Overall Rating Diversity.....	19
4.4 LLM as a Judge .....	20
4.5 Human as a Judge.....	22
4.6 Improvement Convergence .....	24
4.7 Zusammenfassung .....	28
5 Zukünftige Entwicklung .....	29
5.1 Dynamisches Prompt Template.....	29
5.1.1 Sektionen und Variablen .....	29
5.1.2 Few Shot Sektion und RAG.....	30
5.2 Erweiterte Anwendung in Konversationsketten.....	31
5.2.1 EvaluationChain.....	31
5.2.2 Evaluation-Wrapper .....	32
5.3 Weiterführende Ideen .....	33
6 Fazit .....	36
Literaturverzeichnis .....	V

## Abbildungsverzeichnis

Abbildung 1: Anzahl der Veröffentlichungen pro Metrik .....	2
Abbildung 2: Häufigkeit von Fehlern im Zusammenhang mit Requirements .....	3
Abbildung 3: Verfügbare Groq-Modelle .....	6
Abbildung 4: Berechnung des gewichteten Gesamtratings .....	10
Abbildung 5: Aufbau des sukzessiven Ansatzes.....	11
Abbildung 6: Aufbau des iterativen Ansatzes.....	13
Abbildung 7: Llama8b – Verteilung der Gesamtratings bei guten und schlechten Requirements .....	16
Abbildung 8: Claude Haiku – Verteilung der Gesamtratings bei guten und schlechten Requirements .....	18
Abbildung 9: Metrik spezifische Ratingdifferenzen zwischen verschiedenen LLMs .....	19
Abbildung 10: Verteilung der Gesamtratings bei breitem Qualitätsspektrum .....	19
Abbildung 11: Beziehung zwischen Requirement und Bewertungsqualität.....	21
Abbildung 12: Verteilung der Gesamtratings im Vergleich mit menschlichem Experten .....	22
Abbildung 13: Verteilung der Gesamtratingdifferenzen zum menschlichen Experten .....	23
Abbildung 14: Metrik-spezifische Ratingdifferenzen zwischen LLM und Experte .....	24
Abbildung 15: Rating-Verteilung der verbesserten Requirements laut "Llama 70b as a Judge" .....	24
Abbildung 16: HyDE-Ablauf für verbesserte Evaluationen .....	34

## Quellcodeverzeichnis

Quellcode 1: Metrik Beschreibung Correctness .....	5
Quellcode 2: Rating Definition der Correctness Metrik .....	9
Quellcode 3: Strukturierter Output des sukzessiven Ansatzes.....	12
Quellcode 4: Strukturierter Output des iterativen Ansatzes pro Metrik .....	13
Quellcode 5: Strukturierter Output des iterativen Ansatzes für den finalen Prompt .....	14
Quellcode 6: Sukzessiv/Llama 8b – inkorrekte Bewertung der Atomizität.....	17
Quellcode 7: Iterativ/Llama 8b - passende Bewertung der Atomizität.....	17
Quellcode 8: Iterativ/Llama 8b - unstimmgige Begründung des Metrik Ratings .....	17
Quellcode 9: kritische Beurteilung des LLM as a Judge .....	20
Quellcode 10: Positive Beurteilung des LLM as a Judge .....	21
Quellcode 11: Einsatz der Metrik Sektion im Prompt Template .....	29
Quellcode 12: Few Shot Sektion des Prompt Templates .....	30
Quellcode 13: Format Dummy für die Antwort des sukzessiven Ansatzes.....	32

## Abkürzungsverzeichnis

HyDE *Hypothetical Document Embeddings*

LLM *Large Language Model*

NLP *Natural Language Processing*

RAG *Retrieval Augmented Generation*

RE *Requirements Engineering*

# 1 Einführung

Eine erfolgreiche Produktentwicklung ist untrennbar mit den eingangs festgelegten Anforderungen verbunden, die das finale System einmal erfüllen soll. Nur wenn diese Requirements mit den Wünschen und Vorstellungen des Kunden übereinstimmen, kann das Endergebnis aus dessen Sicht als zufriedenstellend gewertet werden. Wie relevant dieses Feld ist, zeigt der Umstand, dass sich mit dem Requirements Engineering, kurz RE, ein ganzes Teilgebiet des Systems Engineering allein mit ebenjenen Requirements befasst. Die Qualität eines solchen Requirements beeinflusst maßgeblich den Entwicklungsprozess von der Planung über die Implementierung bis hin zur Fertigstellung. Eine händische Überprüfung der Güte von Requirements ist dabei oftmals aufgrund der schier unendlichen Anzahl an Anforderungen, die ein Produkt erfüllen soll, ein langwieriger Prozess. Bereits sprachliche Eigenheiten des Verfassers können zu Fehlinterpretationen führen. Diese lassen nicht immer auf die zugrundeliegende Intention schließen. Derartige missverständliche Formulierungen sollten möglichst früh ausgeräumt werden, um nicht hinderlich zu sein. Darüber hinaus definiert sich die Qualität noch über weitere Kriterien, die ebenfalls eng mit der sprachlichen Ausgestaltung eines jeden Requirements verknüpft sind. Solche Schwachstellen sollten möglichst frühzeitig aufgezeigt werden, um die späteren Phasen des Entwicklungsprozesses nicht zu beeinträchtigen.

Diesen Vorgang per Natural Language Processing, kurz NLP, zu automatisieren, birgt großes Potential. In Anbetracht der Entwicklungen, die Large Language Models, kurz LLMs, in ebendiesem Bereich in den letzten Jahren erfahren haben, liegt es nahe, sie auch für diesen Zweck einzusetzen. Nicht nur sind sie in der Lage, große Datenmengen schnell und effizient zu verarbeiten, durch ihre Fähigkeit, natürliche Sprache zu verarbeiten, erscheinen sie geradezu prädestiniert dafür, die Eignung für diesen Anwendungszweck auf die Probe zu stellen.

Das Projekt, welches in dieser Arbeit dargelegt wird, widmet sich der Frage, inwiefern LLMs eine automatisierte Evaluierung von Requirements leisten können. Dabei wurde sich auf die individuelle Bewertung einzelner, atomarer Requirements beschränkt. Die zum Zeitpunkt der Erstellung dieser Arbeit weitverbreiteten LLMs, allen voran ChatGPT, werfen Fragen im Hinblick auf den Datenschutz auf. Dies ist insbesondere problematisch, da Requirements oft sensitive Daten enthalten, die nicht außerhalb des Unternehmens geteilt werden dürfen. Eine Alternative bieten jene LLMs, welche aufgrund reduzierter Parameterzahl auf lokalen Rechnern betreibbar sind. Daher wurden diese herangezogen und untersucht. Die Ergebnisse dieser Untersuchung sollen im Folgenden aufgezeigt werden. Hierzu sollen zunächst die grundlegenden Überlegungen im Hinblick auf die formalen Qualitätskriterien dargelegt werden. Anschließend soll der Entwicklungsprozess erläutert werden, um die Resultate nachvollziehen und einordnen zu können. Daraufhin werden die gesammelten Ergebnisse aufbereitet. Hierauf aufbauend soll auf die Möglichkeiten für zukünftige Weiterentwicklungen eingegangen werden. Abschließend werden die Erkenntnisse in einem Fazit zusammengefasst.



## 2 Theoretische Grundlagen

Die Beurteilung, was ein gutes Requirement ausmacht, gestaltet sich vor dem Hintergrund der eingangs gezeigten Herausforderungen als komplex. Um die qualitative Vergleichbarkeit unabhängig von der Art des Requirements zu gewährleisten, ist es naheliegend, die Qualität anhand verschiedener Kriterien einzuschätzen. Solche formale Qualitätskriterien, im Folgenden Metriken genannt, ermöglichen es, einzelne Requirements aus verschiedenen Blickwinkeln zu betrachten und somit eine fundierte Einschätzung zu erhalten.

In der Literatur gibt es keinen universellen Ansatz für ebenjene Metriken. Ein fester Satz an Kriterien ist nicht definiert, doch in einer Vielzahl von Veröffentlichungen lassen sich wiederkehrende Elemente ausmachen. Montgomery et al. [1] untersuchen in ihrer Arbeit, welche Attribute am weitesten verbreitet in der Literatur sind. Hieraus lässt sich darauf schließen, welche Kriterien als etabliert angesehen werden können. Die dieser Veröffentlichung entnommene Abbildung 1 zeigt dabei auf, wie oft diese Metriken jeweils über 243 Studien verteilt auftraten.

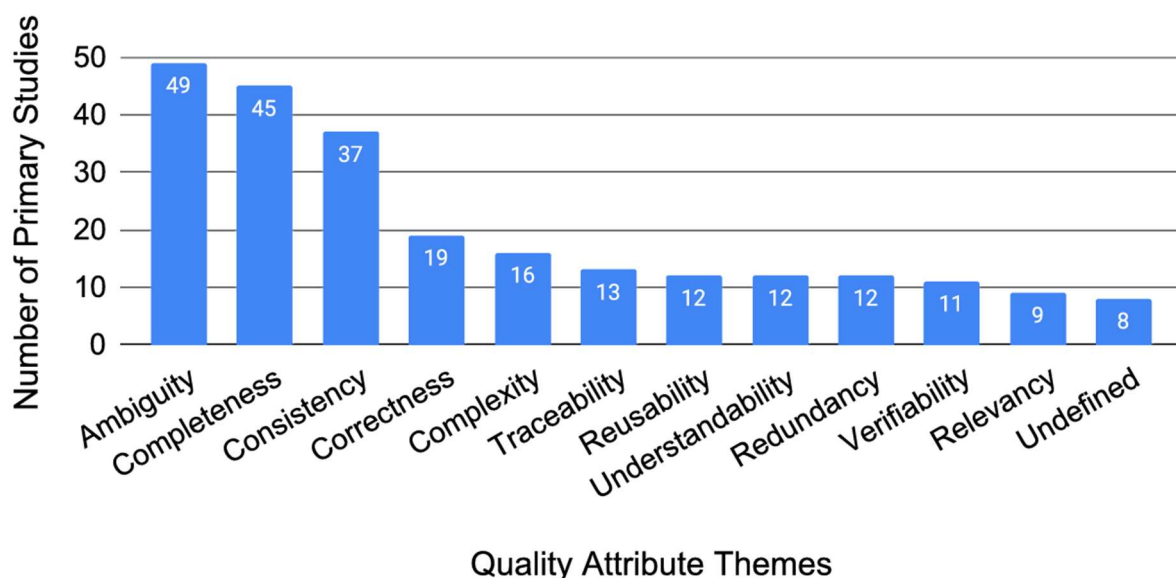


Abbildung 1: Anzahl der Veröffentlichungen pro Metrik

Anhand des Histogramms wird ersichtlich, dass vor allem Aspekte wie die Eindeutigkeit, Vollständigkeit, Konsistenz und Korrektheit mehrfach vertreten sind. Gallego et al. [2] gehen hier sogar noch einen Schritt weiter und bezeichnen in ihrer Fallstudie *Correctness*, *Completeness* und *Consistency* als die wichtigsten Qualitätsattribute. Auch andere Metriken wie die Verifizierbarkeit finden sich hier wieder, während Zowghi und Gervasi [3] ihre Arbeit ausschließlich den oben genannten „Three Cs of Requirements“ widmen. Génova et al. [4] hingegen wählen einen ganzheitlichen Ansatz und verweisen neben den bereits genannten auf eine Reihe weiterer Metriken wie *Precision* und *Atomicity*. Auch hier sind wieder die von Montgomery et al. [1] aufgezeigten häufigsten Vertreter gegenwärtig. Die Wichtigkeit der bis dato dargestellten Kriterien ist also zumindest aus der Literatursicht unstrittig.

Dass ebenjenen Metriken allerdings darüber hinaus auch in realen Anwendungen eine hohe Bedeutung zukommt, deckt sich mit der Arbeit von Fanmuy et al. [5], die in ihrer Veröffentlichung untersuchen, wie die Qualität von Requirements in Industrieanwendungen vorangebracht werden kann. Die

Fallstudie zeigt dabei unter anderem auf, in welchen Bereichen Fehlerquellen im RE-Prozess zu verorten sind. Abbildung 2 lässt sich entnehmen, dass viele der auftretenden Defekte direkt mit Metriken zur Qualitätsbestimmung in Verbindung gesetzt werden können.

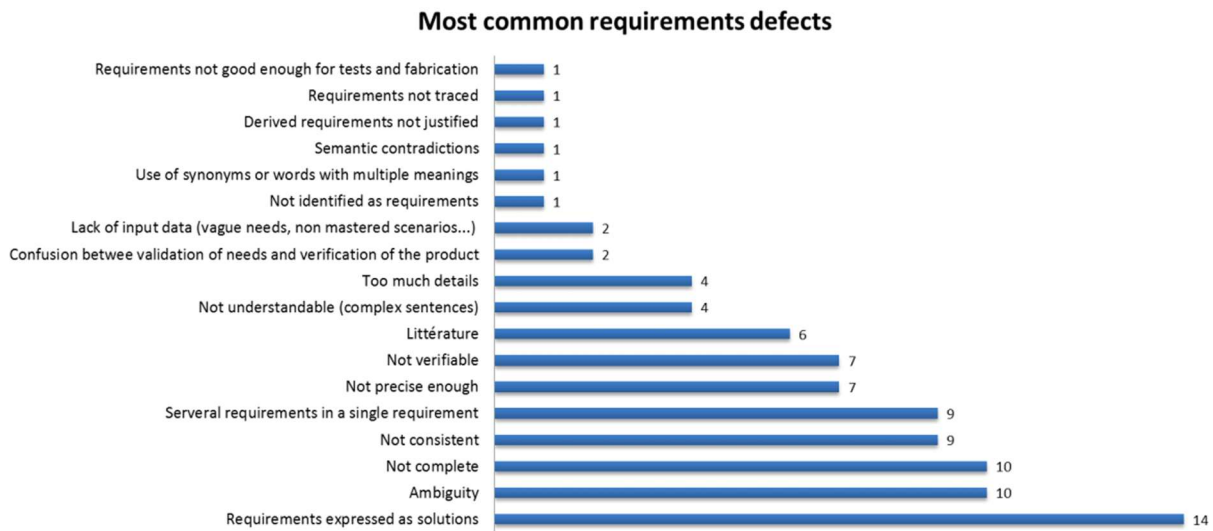


Abbildung 2: Häufigkeit von Fehlern im Zusammenhang mit Requirements

Hier lässt sich erkennen, dass alle der bereits genannten Kriterien tatsächlich merkliche Auswirkungen haben. Aus dieser Erkenntnis lässt sich eine Schnittmenge an Metriken ableiten, die ein qualitatives Requirement erfüllen muss. Es darf keine Fehler aufweisen, sollte eindeutig verständlich und in sich geschlossen kohärent sein. Die beschriebene Funktionalität muss vollständig abgedeckt sein und sollte dennoch präzise dargelegt werden. Darüber hinaus sollte ebenjene Funktionalität verifiziert und getestet werden können. Ein weiterer wichtiger Aspekt ist, dass jedes Requirement nur eine einzelne Funktion beschreibt, welche atomar ist.

Somit wurden sieben Schlüssel-Metriken identifiziert, welche im Rahmen der Evaluierung durch ein Sprachmodell als geeignet erachtet wurden:

- Correctness
- Unambiguity
- Completeness
- Consistency
- Precision
- Verifiability
- Atomicity

Diese Metriken decken die in der Literatur relevantesten Kriterien ab [1]. Darüber hinaus erweitern sie diese um in der Praxis auftretende Problemfelder, die es zu identifizieren gilt [5]. Somit bieten sie eine fundierte Grundlage, auf deren Basis das Sprachmodell eine umfassende Einschätzung des Requirements liefern sollte.

Hieraus ergibt sich allerdings eine Herausforderung: Die Bezeichnung der Metriken als einzelnes, für sich stehendes Wort geht wieder mit mehreren Interpretationsmöglichkeiten einher. Begriffe wie *Consistency* können mehrdeutig ausgelegt werden, was eine Vergleichbarkeit wiederum beeinträchtigt. Daher bedarf es weiterem Kontext, der dem Sprachmodell in Form einer Definition zur Verfügung

gestellt werden soll. Allgemeingültige Definitionen finden sich in der Literatur nicht, weshalb sich an der Herangehensweise von Honig [6] orientiert wurde. In dieser Veröffentlichung wird eine Liste an Metriken für einzelne, atomare Requirements als auch für ein ganzes Dokument definiert. Die Beschreibungen der jeweiligen Kriterien sind dabei als Fragen formuliert. Dies ist insbesondere im Kontext der Bewertung durch LLMs zweckdienlich, da das Modell diese Fragen direkt in der Evaluation anwenden kann. Die meisten der in diesem Projekt verwendeten Metriken sind in der Liste ebenfalls vertreten, allerdings wurde sich aus den oben genannten Gründen nicht ausschließlich darauf beschränkt. Darüber hinaus unterscheiden sich die Beschreibungen in ihrem Detailgrad, dem eine einheitliche Ausgestaltung jedoch vorzuziehen ist. Somit wurden die Definitionen zwar als Ausgangspunkt verwendet, allerdings erweitert und modifiziert. Dabei ist eine Reihe an Einschränkungen zu beachten.

Was für eine klare Definition hinderlich ist, ist die von Zowghi und Gervasi [3] beschriebene Beziehung zwischen verschiedenen Metriken. Auch wenn Metriken sich unter Umständen gegenseitig beeinflussen, sollte ihre Definition sich dennoch möglichst nicht überschneiden. Der Grund hierfür soll an folgendem fiktivem Beispiel aufgezeigt werden:

*„Das System soll bei der Detektion aller Objekte in einem Umkreis von 50 Metern hinter dem Fahrzeug Alarm auslösen. Dabei soll der Alarm nur ausgelöst werden, wenn sich das Objekt dem Fahrzeug nähert.“*

Wie unschwer zu erkennen ist, birgt dieses Requirement eine Reihe an Problemen. Aus der Perspektive vieler der oben genannten Metriken offenbaren sich Schwachstellen. Vor allem ist die technische Machbarkeit der „Detektion aller Objekte“ nicht umsetzbar, somit ist *Correctness* nicht gegeben. Auch ist es sprachlich unklar und unpräzise, sodass sowohl *Precision* als auch *Unambiguity* Raum für Verbesserungen bieten. Darüber hinaus weist es keine *Consistency* auf, da sich der zweite Satz unmittelbar mit dem ersten widerspricht. Es kann also vorkommen, dass Metriken sich unmittelbar bedingen.

Dennoch ist es ein Trugschluss, nun allgemeingültig davon auszugehen, dass Metriken sich in sämtlichen Fällen überschneiden. Vielmehr ist eine klare Abgrenzung vonnöten, um die Metriken individuell unabhängig voneinander bewerten zu können. Nur so können spezifische Schwachstellen identifiziert und behoben werden. Das lässt sich am obigen Beispiel demonstrieren. Eine Eingrenzung auf die Art der Objekte würde sowohl *Correctness*, *Unambiguity* und *Precision* fördern, ändert jedoch nichts an der *Consistency*, da die Warnung ausschließlich bei sich nähernden Objekten nach wie vor im Widerspruch zur eigentlichen Funktionalität steht. Läge nun keine klare Trennung der Definitionen vor, könnten vermeintlich allgemeingültige Verbesserungen vorgeschlagen werden, die jedoch nicht alle Metriken abdecken. Dementsprechend sollten die einzelnen Definitionen keine Überschneidungen aufweisen.

Ferner sollten die Definitionen die relevantesten Aspekte abdecken, um eine klare Veranschaulichung des jeweiligen Begriffes zu liefern. So kann das Sprachmodell angeleitet werden zu erkennen, wo der Fokus der jeweiligen Metrik liegt und die Erfüllung dieser Punkte besser überprüfen. Dennoch ist es wünschenswert, die Formulierungen möglichst kompakt und sprachlich präzise zu halten, um den späteren Prompt nicht zu überladen. Von diesen Überlegungen ausgehend ergab sich eine erste Sammlung verschiedener Metrik-Definitionen. Aus der Diskussion über die Ergebnisse, welche unter Verwendung dieser Definitionen hervorgegangen waren, wurden die Beschreibungen iterativ verbessert. Hierbei wurde ChatGPT zurate gezogen, insbesondere um die prägnante Ausdrucksweise ohne

Inhaltsverlust zu ermöglichen. Schlussendlich wurden die Definitionen noch einmal händisch überarbeitet, um für das Projekt relevante Punkte mit einfließen zu lassen. Quellcode 1 zeigt beispielhaft die finale Beschreibung für die Metrik *Correctness*.

```
{  
  "Correctness":  
    [  
      "Does the requirement define a genuine system function or need?",  
      "Is the requirement technically feasible within the system context?",  
      "Is the requirement free from errors and consistent with established  
        standards or domain knowledge?"  
    ],  
}
```

Quellcode 1: Metrik Beschreibung Correctness

Hier wird deutlich, dass die beschreibenden Fragen knapp gefasst sind und sich auf die wesentlichen Aspekte beschränken. Darüber hinaus weisen sie keinerlei Überschneidungen zu anderen Metriken auf, sondern bieten eine klare Abgrenzung. In dieser Art wurde auch bei den verbleibenden Definitionen verfahren, wie an der entsprechenden Stelle im zugrundeliegenden Repository eingesehen werden kann.

Mit der Zusammenstellung aus Metriken und den damit verbundenen Definitionen basierend auf den gesammelten Erkenntnissen aus der Literatur ist eine fundierte Grundlage geschaffen. Diese soll dazu beitragen, LLMs zu ermöglichen, eine qualifizierte Evaluation von Requirements zu liefern. Die weitere Vorgehensweise soll im Folgenden aufgezeigt werden.

### 3 Entwicklungsprozess

In diesem Kapitel sollen Überlegungen und Ansätze im Hinblick auf die wichtigsten Elemente des Projektes aufgezeigt werden.

#### 3.1 Sprachmodelle

Wie bereits eingangs erwähnt, sollte der Fokus in diesem Projekt auf lokal ausführbaren LLMs liegen. Hierbei war insbesondere von Interesse, ob sich auch mit kleineren Modellen ein zufriedenstellendes Ergebnis erreichen lässt. Die Idee hieran war, durch Prompt Engineering eine Performance zu erzielen, die im Idealfall mit derer von Benchmark-LLMs vergleichbar ist. Hier kommt zunächst eine Reihe an Modellen infrage, die jedoch aufgrund der Gegebenheiten des Projekts schnell eingegrenzt wurden.

Für eine einfache Testbarkeit während der Entwicklung war die Verwendung in einer Chatinterface hilfreich, für ein uniformes Format sollte das Modell die Ausgabe in einer vorgegebenen Struktur unterstützen. Ein weiterer zentraler Punkt war die Geschwindigkeit des Modells, da es in frühen Phasen des Projekts absehbar war, dass im Sinne einer fundierten Ergebnisbildung große Datenmengen produziert werden sollten. Außerdem erforderten die in den folgenden Kapiteln beschriebenen Funktionalitäten einen möglichst dynamischen Wechsel zwischen Modellen, um etwa ein schnelles Ausprobieren verschiedener Ansätze zu ermöglichen. Während der Download für ein einzelnes kleines Modell noch vertretbar ist, ist diese Herangehensweise für viele Modelle nicht mehr praktikabel.

Aus diesen Gründen wurde sich darauf festgelegt, auf die *Groq* API zurückzugreifen. Diese lässt sich nicht nur einfach handhaben, auch lassen sich sehr hohe Geschwindigkeiten bei der Erzeugung der Antworten erzielen. Diese API stellt zum Zeitpunkt der Erstellung dieser Arbeit die in Abbildung 3 gezeigten Modelle zur Verfügung [7].

MODEL ID	DEVELOPER	CONTEXT WINDOW (TOKENS)	MAX COMPLETION TOKENS	MAX FILE SIZE	MODEL CARD LINK
distil-whisper-large-v3-en	HuggingFace	-	-	25 MB	<a href="#">Card</a>
gemma2-9b-it	Google	8,192	-	-	<a href="#">Card</a>
llama-3.3-70b-versatile	Meta	128K	32,768	-	<a href="#">Card</a>
llama-3.1-8b-instant	Meta	128K	8,192	-	<a href="#">Card</a>
llama-guard-3-8b	Meta	8,192	-	-	<a href="#">Card</a>
llama3-70b-8192	Meta	8,192	-	-	<a href="#">Card</a>
llama3-8b-8192	Meta	8,192	-	-	<a href="#">Card</a>
mixtral-8x7b-32768	Mistral	32,768	-	-	<a href="#">Card</a>
whisper-large-v3	OpenAI	-	-	25 MB	<a href="#">Card</a>
whisper-large-v3-turbo	OpenAI	-	-	25 MB	<a href="#">Card</a>

Abbildung 3: Verfügbare Groq-Modelle [7]

Die in der Grafik markierten Modelle wurden nach den oben beschriebenen Anforderungen und dem Stellenwert der *Llama*-Modelle ausgewählt. Die jeweiligen Modelle kamen dabei für verschiedene

Anwendungszwecke zum Einsatz. Nach ersten Tests mit *llama3-8b-8192* wurde für die eigentlichen Evaluationen auf *llama-3.1-8b-instant* zurückgegriffen da dieses Modell neuer ist. Das größere 70b-Modell wurde für eine andere Anwendung verwendet, welche in Kapitel 3.4.1 beschrieben wird.

Es sei jedoch erwähnt, dass die *Groq*-API mitunter eine starke Fluktuation in der Verfügbarkeit einzelner Modelle aufweist. Einige LLMs, die in frühen Phasen des Projektes zum Einsatz kamen, wurden über den Verlauf der Bearbeitung eingestellt. Dies ist wohl den rasanten Entwicklungen in diesem Bereich geschuldet.

Darüber hinaus wurde die Möglichkeit implementiert, kostenpflichtige *Anthropic*-Modelle zu nutzen. Diese wurden verwendet, um Benchmark-Evaluationen zu erhalten. Um die Vergleichbarkeit der Modelle untersuchen zu können, bedarf es einer Vielzahl an verschiedenen Requirements.

## 3.2 Datensätze

Um ein möglichst breit gestreutes Bild zu erhalten, war der ursprüngliche Wunsch, je einen Datensatz für Positivbeispiele und einen für qualitativ minderwertige Requirements zu erstellen. Als erste Anlaufstelle wurde dabei *Kaggle* herangezogen, wo ein Datensatz für Software Requirements zur Verfügung steht [8]. Dieser enthält nahezu 1000 Requirements. Nach einer Durchsicht konnte festgestellt werden, dass der Datensatz bereits eine gewisse Streuung des Qualitätsniveaus aufweist. Dennoch sind die Requirements qualitativ überwiegend im mittelmäßigen Bereich angeordnet und bieten kaum Ausreißer. Um dem Modell durch die in den folgenden Kapiteln beschriebenen Methodiken eine bessere Orientierung liefern zu können, waren solche stark ausgeprägten Positiv- und Negativbeispiele jedoch wünschenswert. Da diese dem Modell außerdem eine gewisse Erwartungshaltung vermitteln, können sie dazu beitragen, verschieden geartete Aspekte des Requirements besser einzuordnen. Da so eine gewisse Vergleichbarkeit geschaffen wird, sollte es dabei helfen, die Plausibilität einer Evaluation besser einschätzen zu können.

Da zum Zeitpunkt der Erstellung dieser Arbeit online kein Datensatz für Negativbeispiele zur Verfügung stand, wurde zu diesem Zweck ein LLM herangezogen. In einem eigens entwickelten Programm wird dabei das *Kaggle* Software Requirements Dataset durchiteriert und das Sprachmodell wird aufgefordert, eine qualitativ minderwertige Version der einzelnen Requirements zu erstellen. Eine solche Vorgehensweise ist für rein positive Beispiele wenig sinnvoll, da dies bereits ein perfektes Verständnis der Thematik seitens des Sprachmodells voraussetzt. Auch der *Kaggle*-Datensatz ist im Hinblick auf die Varianz auch eher ungeeignet, um ausschließlich gute Requirements zu repräsentieren. Aus diesem Grund wurde für die Positivbeispiele ein firmeninterner Datensatz an Requirements der *Spicy SE* herangezogen. Dieser umfasst über 200 Requirements und wurde aufgrund der realen Anwendung in der Praxis als hinreichend gut angenommen.

Somit standen Datensätze zur Verfügung, welche insgesamt eine stark ausgeprägte Streuung aufweisen, aber in sich geschlossen gleichbleibend sind. Dementsprechend wurden die drei resultierenden Datensets mit *bad*, *average* und *good* gelabelt. Neben der Verwendung als Orientierungshilfe liegt es nahe, anhand dieser Listen auch Evaluierungen der Datensätze durchzuführen, um zu beurteilen, ob diese die vorherrschende Streuung akkurat abbilden. Um die Performance genauer zu überprüfen und anhand eines Vergleichs mit einem (menschlichen) Experten zu messen, wurden zusätzlich zehn *Benchmark* Requirements aus dem firmeninternen Satz gewählt und händisch evaluiert. Dies reduziert den Aufwand der manuellen Bewertung seitens des Experten und erlaubt es gleichermaßen, eine fundierte Einschätzung der Leistung des LLMs zu erhalten. Um ebenjene Sprachmodelle zu befähigen,

erwartungsgemäße Evaluationen durchzuführen, ist ein hochqualitativer Prompt das mächtigste Werkzeug.

### 3.3 Prompt Engineering

Über den Zeitraum der Entwicklung wurde eine Reihe verschiedener Prompts erprobt, bevor sich schlussendlich zwei Versionen herauskristallisierten.

#### 3.3.1 Ansätze

Die generelle Herangehensweise basierte auf Best Practices, welche zum Zeitpunkt der Projektbearbeitung etablierte Vorgehensweisen waren. Aus diesen ließ sich ableiten, dass es von Vorteil ist, die Anweisungen unmittelbar zu Beginn des Prompts zu platzieren [9]. Eine grundlegende Überlegung war außerdem, dem LLM eine fiktive Rolle zuzuweisen. Laut *Meta* [10] trägt dies dazu bei, relevante und domänenpezifische Ergebnisse zu erhalten, da das Sprachmodell die Perspektive eines Experten auf diesem Gebiet einnimmt. Dementsprechend wurden dem Modell in der initialen Instruktion Bezeichnungen wie *Requirements Expert* oder *Software Engineer* zugeordnet. Dieser Ansatz lässt sich mit der Verwendung von *System* und *User Prompts* noch weiter vereinfachen. Die Idee hierbei war, die Rollenzuweisung zusammen mit generellen Anweisungen in der *System Message* zu platzieren. Diese könnten dann auch bei schrittweisen Ansätzen wiederverwendet werden, um Token zu sparen, da das Modell sich die System Message „merkt“ [11]. Die *Human Message* wäre dann für jeden Prompt unterschiedlich und könnte beispielsweise das zu bewertende Requirement enthalten. Da dieser Ansatz keine merkliche Verbesserung gezeigt hat, wurde er wieder verworfen. Bei größeren Modellen wäre die Verwendung eines solchen „Gedächtnisses“ mutmaßlich effektiver.

Neben den einfachen Instruktionen wurde eine Reihe weiterer Techniken erprobt. Ein erster Ansatz, um akkuratere Ergebnisse zu erhalten, war der Einsatz von *Few-Shot-Prompting*. Eine grundlegende Überlegung war dabei, dem Prompt Beispiele von guten und schlechten Requirements hinzuzufügen, um dem Sprachmodell weitere Orientierungshilfen bereitzustellen. Dies wurde insofern weiterentwickelt, dass die Beispiele um Benchmark-Evaluationen erweitert wurden. Hierdurch ergibt sich der große Vorteil, dass das Modell konkrete Anhaltspunkte hat, warum ein Requirement im Hinblick auf die Metriken ein bestimmtes Qualitätsniveau aufweist. Aufgrund der genannten Vorteile findet sich *Few-Shot-Prompting* in einem der finalen Ansätze wieder. Gemäß Brown et al. [12] ist diese Technik bei kleineren Modellen deutlich weniger effektiv. Für Modelle um die acht Milliarden Parameter, wie sie in diesem Projekt verwendet wurden, ist seiner Arbeit jedoch keine eindeutige Aussage zu entnehmen. Daher wurde der Ansatz dennoch auf die Probe gestellt.

Der oben beschriebene Ansatz lässt sich mit statischen *Few-Shot*-Beispielen durchführen, eine naheliegende Methode wäre außerdem die Verwendung von *Retrieval Augmented Generation*, kurz RAG. Dies hätte den Vorteil, dass dem jeweiligen Requirement optimal zugeschnittene Beispiele zugewiesen werden könnten. Gerade bei domänenspezifischen Requirements hätte dies großes Potential, da dem Modell so Zusammenhänge und Eigenheiten des jeweiligen Bereiches nähergebracht werden könnten. Bei der Erprobung zeigte sich jedoch, dass sich durch die Verwendung von RAG keine Vorteile ergaben. Das lässt sich darauf zurückführen, dass keine perfekten Bewertungsbeispiele in der benötigten Menge existieren, um dem Modell tatsächlich eine nennenswert bessere Unterstützung im Vergleich zu statischen Beispielen zu liefern.

Eine weitere Herangehensweise war die Verwendung von *Chain-of-Thought-Prompting*, um eine stufenweise Durchführung der einzelnen Evaluationsschritte zu ermöglichen. Da diese Technik einfach umzusetzen ist und laut Wei et al. [13] zu deutlichen Verbesserungen beitragen kann, wurde sie bei einem der finalen Ansätze implementiert. Hierbei waren keine größeren Anpassungen nötig, da eine logische Aufteilung auf einzelne Schritte bereits in frühen Tests angewandt wurde.

Damit die Antworten des Sprachmodells sich besser vergleichen lassen, sollten die Antworten einer festen Struktur folgen. Hierbei wurde das JSON-Format als sinnvoll erachtet, da die verwendeten Modelle dieses unterstützen und es einem klar verständlichen Schema folgt. Dieser strukturierte Output wurde ebenfalls im Prompt festgelegt und unterschied sich nur minimal in den zwei finalen Ansätzen. Die wichtigsten Elemente finden sich in beiden Herangehensweisen wieder, allen voran die Metriken und die Definitionen. Dabei war es vonnöten, dass sich die Bewertung der einzelnen Metriken durch das LLM nachvollziehbar einschätzen lässt.

### 3.3.2 Rating-Skalierung

Um die Qualität des Requirements basierend auf den definierten Metriken in Zahlen auszudrücken, wurden verschiedene Skalen getestet. Dabei musste ein Kompromiss zwischen Granularität und Einfachheit gefunden werden. Zu Beginn wurde ein Rating von 1 bis 10 festgelegt, jedoch fällt es damit schwerer, jeder „Note“ eine genaue Bedeutung zu zuordnen. Dies führte zu der Idee, neben der Reduzierung des Wertebereichs auch präzise Definitionen für jede Note festzulegen, um das LLM bei der Entscheidungsfindung zu unterstützen. Zunächst wurde dies mit einem 1-3 Rating probiert, in der Hoffnung, eine möglichst simple, aber klare Bewertung zu erhalten. Allerdings erwies sich dies nicht als sinnvoll, da trotz offensichtlicher Qualitätsunterschiede zu häufig dieselben Noten vergeben wurden. Daher wurde sich final für eine 5er Skalierung entschieden. Anstatt einer allgemeingültigen Rating-Definition, wurden diese Metrik-spezifisch zugeschnitten. Beispielhaft ist nachfolgend die Definition passend zur Correctness-Metrik (Quellcode 1) zu sehen:

"Correctness": [

- "1 - **Very Poor**: The requirement is invalid, describing a function or need that is infeasible, erroneous, or inconsistent with established domain knowledge.",
- "2 - **Poor**: The requirement contains significant inaccuracies, errors, or misalignments with technical feasibility or domain standards.",
- "3 - **Average**: The requirement is mostly valid but has minor technical errors, feasibility concerns, or unclear assumptions.",
- "4 - **Good**: The requirement is valid, technically feasible, and aligns well with established standards, with only minor areas for improvement.",
- "5 - **Excellent**: The requirement is completely valid, technically sound, and fully aligned with domain standards or commonly accepted principles."

]

Quellcode 2: Rating Definition der Correctness Metrik

Bei der Analyse eines Requirements ist neben der Bewertung der einzelnen Metriken auch ein finales Gesamtrating wünschenswert. Ein trivialer Ansatz zur Berechnung wäre der Mittelwert aus allen Einzelwertungen. Allerdings lassen sich nach genauerer Betrachtung der Metriken unterschiedliche Erwartungshaltungen und Prioritäten ableiten. So ist beispielsweise die Vollständigkeit oder das Vorliegen einer atomaren Funktionalität weniger kritisch als die Korrektheit eines Requirements. Ebenso wird Konsistenz bzw. das Vermeiden von Widersprüchen in sich selbst viel eher erwartet als



absolute Präzision. Testbarkeit und Eindeutigkeit sind zwar wichtig, lassen aber je nach Kontext Freiräume zu. Aus diesen Überlegungen heraus lässt sich durch entsprechende Gewichtung eine Rechenvorschrift für eine differenziertere Gesamtbewertung definieren, welche in Abbildung 4 dargestellt ist.

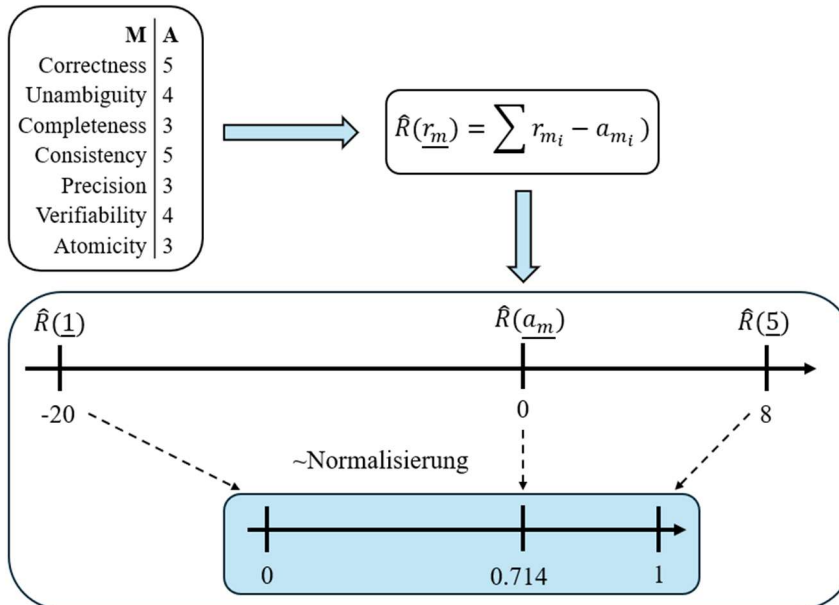


Abbildung 4: Berechnung des gewichteten Gesamtratings

Die Gewichtung basiert auf die in der Tabelle festgelegten Akzeptanzschwellen **A** für jede Metrik **M**, wobei diese jeweils vom eigentlichen 5er Rating abgezogen werden. Dadurch werden Metriken mit höherer Akzeptanzschwelle stärker bestraft und damit indirekt priorisiert. Für das Gesamtrating **R** werden die gewichteten Einzelwertungen aufaddiert. Damit ergibt sich ein Wertebereich von  $[-20, 8]$  welcher auf  $[0, 1]$  normalisiert wird. Hierbei wurde zusätzlich die normalisierte Gesamtakzeptanzschwelle ( $0 \rightarrow 0.714$ ) bestimmt, um das Ergebnis einordnen zu können.

Gemeinsam mit den Metriken fungieren die Ratings sowie deren Gewichtungen als elementare Grundbausteine für die Erstellung der Prompts, auf deren Basis die Evaluationen durchgeführt werden sollten.

### 3.3.3 Bestandteile

Auch wenn zwei unterschiedliche Ansätze finalisiert wurden, so weisen diese nichtsdestotrotz viele gemeinsame Elemente auf. Grundlegend bestehen die Prompts aus den initialen *Anweisungen*, welche die eigentliche Aufgabe des Sprachmodells definieren und somit Instruktionen stellen. Ein weiterer zentraler Bestandteil sind die *Metriken*, welche mit den jeweiligen Definitionen und Ratings übergeben werden. Optional werden *Few-Shot-Beispiele* dynamisch eingefügt. Das eigentliche *zu bewertende Requirement* wird dem Prompt ebenfalls dynamisch hinzugefügt. Darüber hinaus wird die gewünschte Struktur der Antwort mit der entsprechenden Formatierung vorgegeben. Diese Elemente lassen sich in beiden Prompts wiederfinden, sie werden jedoch auf unterschiedliche Weisen eingesetzt.

### 3.3.4 Sukzessiv

Der sukzessive Ansatz führt die gesamte Evaluierung in einem einzigen Schritt durch. Wie in Abbildung 5 dargestellt, folgen auf die Anweisungen zunächst das zu bewertende Requirement und anschließend die Metriken, bevor schlussendlich die Formatierung festgelegt wird.

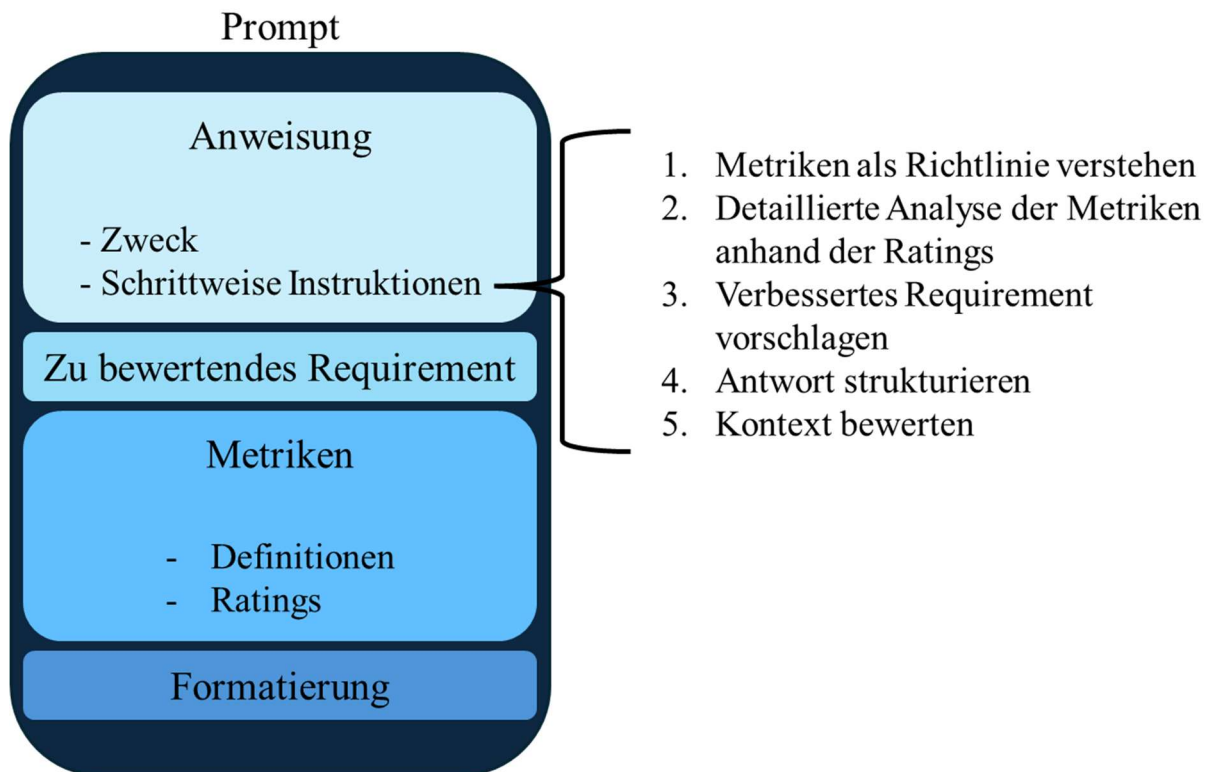


Abbildung 5: Aufbau des sukzessiven Ansatzes

In der Anweisung werden nach der Rollenzuweisung die eigentlichen Instruktionen gestellt. Diese sind entsprechend dem *Chain-of-Thought*-Ansatz schrittweise formuliert. Nachdem das Modell aufgefordert wird, eine detaillierte Analyse durchzuführen, soll es eine verbesserte Version des ursprünglichen Requirements erstellen. Diese soll anhand der Schwachstellen der jeweiligen Metriken erfolgen. Darüber hinaus soll das Sprachmodell eine Abschätzung darüber treffen, ob zusätzlicher Kontext für eine Akzeptanz des Requirements nötig ist. Ist dies nicht der Fall, so wird das Requirement als in sich geschlossen verständlich angenommen. Zusätzlicher Kontext wird beispielsweise nötig, wenn Begriffe enthalten sind, die in anderen Teilen des Dokuments definiert sind.

Das zu bewertende Requirement folgt unmittelbar nach den Instruktionen und soll mithilfe der Metrik-Definitionen anhand der Ratingskala bewertet werden. Dabei sind in dem Prompt alle Metriken gemeinsam enthalten. Dies gilt sowohl für die Definitionen als auch die Ratings. Aufgrund der in Kapitel 3.3.1 erwähnten Einschränkungen wurde im finalen Ansatz auf *Few-Shot*-Beispiele verzichtet, da bereits ohne diese zufriedenstellende Ergebnisse erzielt werden konnten. Der Hintergrund hierfür war die Herangehensweise, den sukzessiven Prompt so kurz wie möglich zu halten, da dieser durch die eingefügte Liste der Metrik Definitionen und der entsprechenden Ratings bereits eine hohe Länge erreicht.

Das letzte Teilstück des Prompts ist die Vorgabe des strukturierten Outputs. In den schrittweisen Anweisungen wird bereits darauf verwiesen und dem Modell mitgeteilt, dass es sich bei seiner Antwort

an diese vorgegebene Struktur halten soll. Wie in Quellcode 3 dargestellt, wird aus Gründen der Übersichtlichkeit das zu bewertende Requirement vor die eigentlichen Evaluierung gesetzt.

```
{
  "requirement": "<Requirement to Evaluate>",
  "evaluation": {
    "<Metric 1>": {
      "rating": "(int:<1-5>)",
      "comment": "<justification>"
    },
    ...
    "<Metric 7>": {
      "rating": "(int:<1-5>)",
      "comment": "<justification>"
    }
  },
  "proposed_requirement": {
    "text": "<Improved Requirement or statement for missing proposal>",
    "justification":
      "<Explanation of changes or why improvement is unnecessary>"
  },
  "context": {
    "verdict": "<accepted/not accepted>",
    "justification":
      "<Explanation why additional context would be needed or not for acceptance>"
  }
}
```

Quellcode 3: Strukturierter Output des sukzessiven Ansatzes

Jede Metrik wird neben einem Rating auf der entsprechenden Skala auch mit einer Begründung des LLM versehen, wie die Festlegung auf einen Wert erfolgt ist. Der Evaluation folgt das verbesserte Requirement, wobei das Sprachmodell auch hier seine Vorgehensweise erläutern soll. Dabei treten auch Sonderfälle auf, wenn das ursprüngliche Requirement kein Verbesserungspotential mehr aufweist oder so vage oder unzureichend formuliert ist, dass es nicht als vollwertiges Requirement anzusehen ist und somit nicht verbessert werden kann. Zu guter Letzt folgt die bereits erwähnte Beurteilung des Kontexts, welche ebenfalls begründet werden muss.

### 3.3.5 Iterativ

Im Gegensatz zur sukzessiven Methode wird beim iterativen Ansatz jede Metrik mit einem für sich stehenden Prompt einzeln bewertet. Abbildung 6 zeigt auf, dass die initialen Anweisungen dabei über alle Metriken gleich bleiben.

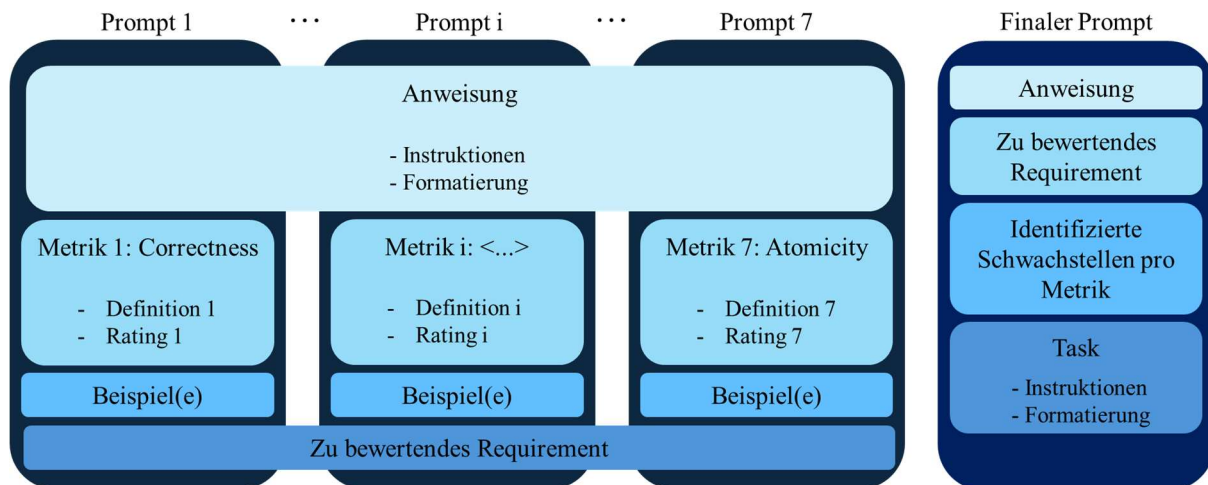


Abbildung 6: Aufbau des iterativen Ansatzes

Nach den Instruktionen, die ebenfalls schrittweise formuliert sind, wird in der Anweisung auch die in Quellcode 4 dargestellte Struktur vorgegeben. Diese kann als verkürzte Version der Struktur des sukzessiven Ansatzes gesehen werden, da sie neben dem originalen Requirement auch das Rating und eine Begründung erhält.

```
{
  "requirement": "<original requirement>",
  "rating": "(int:<1-5>)",
  "justification": "<justification for the rating>",
  "proposed_requirement": "<suggested improvement, if applicable else null>"
}
```

Quellcode 4: Strukturierter Output des iterativen Ansatzes pro Metrik

Darüber hinaus wird bereits für jede Metrik einzeln ein verbessertes Requirement erstellt, welches nur fehlerhafte Aspekte dieses einen Qualitätskriteriums korrigiert. Somit werden hier die in Kapitel 2 beschriebenen spezifischen Verbesserungen mit klaren Abgrenzungen durchgeführt. In dieser Art und Weise wird jede Metrik durchiteriert und jeweils dazu passende Beispiele per *Few-Shot*-Prompting eingefügt. Diese sind statisch, für jede Metrik gibt es festgelegte Beispiele, die den jeweiligen Datensätzen entnommen sind und somit die verschiedenen Qualitätsstufen repräsentieren.

Nachdem die Evaluationen der einzelnen Metriken abgeschlossen sind, werden in einem finalen Prompt die gesammelten Ergebnisse aufbereitet. Dazu werden aus den vorherigen sieben Evaluationen jeweils die Schwachstellen pro Metrik entnommen und gesammelt. Das Sprachmodell nutzt diese, um eine Optimierung durchzuführen. Hierbei gibt es eine Akzeptanzschwelle, wird diese nicht unterschritten, so werden diese nicht berücksichtigt. Der Vorteil daran ist, dass Metriken, für die kein Verbesserungsbedarf besteht, gar nicht erst im finalen Prompt vertreten sind, wodurch sich wieder die

gewünschten klaren Abgrenzungen ergeben. Auch dieser finale Prompt gibt eine Struktur vor, welche in Quellcode 5 dargestellt ist.

```
{  
  "requirement": "<original requirement>",  
  "proposed_requirement": "<suggested improvement if applicable, else null>",  
  "justification": "<Explanation how each metric is addressed if applicable, else null>"  
}
```

Quellcode 5: Strukturierter Output des iterativen Ansatzes für den finalen Prompt

Hierbei wird die Begründung für eine Verbesserung pro Metrik ausgeführt. Der finale Prompt wird konsequenterweise von den anderen losgelöst ausgeführt, beruft sich dabei aber zugleich auf den Erkenntnissen der vorherigen. Führt man diese Idee fort, kann dies als Grundlage für die Implementierung eines Multi-Agenten-Systems angesehen werden. Bei diesem Ansatz könnten LLMs miteinander agieren und so unterschiedliche Aufgaben im Kontext des Projektes ausführen. Da ein solches viel Potential bietet, wird diese Idee in Kapitel 5 näher ausgeführt. Ein naheliegender Einsatzzweck ist die Beurteilung der erstellten Evaluationen der beiden Ansätze.

## 3.4 Performance-Evaluierung

Die Performance der gewählten Ansätze bewerten zu können, gestaltet sich im Hinblick auf mehrere Aspekte als schwierig. Nicht immer ist zweifelsfrei festzustellen, was im Kontext eines Requirements richtig und was falsch ist. Selbst Expertenmeinungen können je nach fachlichem Hintergrund auseinandergehen, da jeder relevanten Punkten und Ausschlusskriterien eine andere Bedeutung beimisst. Um diesem Umstand der individuellen Bewertungsgrundlage beizukommen, wurden wie schon in anderen Bereichen des Projektes verschiedene Ansätze erprobt.

### 3.4.1 LLM as a Judge

Gemäß dem oben formulierten Ansatz eines Multi-Agenten-Systems liegt es nahe, die Performance eines Sprachmodells durch ein anderes Modell zu bewerten. Die erste Idee war hierbei die Verwendung eines größeren LLMs, um eine fundierte Beurteilung zu erhalten. Dabei fiel die Wahl auf das bereits erwähnte *llama-3.3-70b-versatile*, welches laut eigener Aussage von *Meta* [14] nahezu gleichauf mit der Performance des deutlich größeren *llama-405b*-Modell liegt. Auch unabhängig von der Modellgröße ist die Verwendung eines anderen Sprachmodells hier von Vorteil. Wataoka et al. [15] zeigen in ihrer Studie einen nicht zu vernachlässigen *Self-Preference Bias* bei der Verwendung von LLMs zur Performance-Evaluierung auf. Daraus lässt sich schließen, dass Sprachmodelle dazu neigen, eigene Antworten besonders gut zu bewerten. Dementsprechend wäre die Vertrauenswürdigkeit einer Selbsteinschätzung durch ein LLM fragwürdig, was die Verwendung eines zweiten Modells rechtfertigt.

Das Vorgehen entsprach dabei in großen Teilen den bisher aufgezeigten Schritten bei der Erstellung des Prompts. Dem Modell wurde die Rolle als RE-Experte zugewiesen und aufgefordert, die Bewertung eines Requirements anhand der definierten Metriken zu überprüfen. Dabei soll für jede Metrik die Korrektheit des Ratings sowie die Qualität und Kohärenz der dazu abgegebenen Erklärung auf einer eigenen Skala von eins bis fünf beurteilt werden. Gleichmaßen soll auch bei dem vorgeschlagenen verbesserten Requirement verfahren werden, sodass dieses ebenfalls mit einer numerischen Bewertung versehen wird. Auch bei der Implementierung des Judge wurde sowohl ein sukzessiver als auch ein

iterativer Ansatz verfolgt. Der Aufbau der Prompts sowie die vorgegebene Struktur der Antwort orientiert sich dabei stark an den bereits beschriebenen Aspekten aus Kapitel 3.3, weshalb hier auf genauere Ausführungen verzichtet wird.

Bei der durch den Judge generierten Bewertung sind eine Reihe an Einschränkungen zu beachten. Während es für die Evaluierung selbst von Vorteil ist, das Requirement anhand von den definierten Metriken einzuschätzen, könnte das Urteil des Judge beruhend auf diesen Metriken eine Einschränkung darstellen. Wenn die Evaluation beispielsweise Schwachstellen abseits der Metriken aufweist, wäre der Judge unter Umständen nicht dazu in der Lage, diese aufzudecken, wenn er sich bloß auf diese fokussiert. Dazu kommt noch, dass ein LLM als Judge mitnichten eine unfehlbare Referenz darstellt. Unabhängig von der Modellgröße ist es fraglich, ob das Modell ausreichend Domänenwissen im Falle von sehr spezifischen Requirements besitzt. Aufgrund dieser Einschränkungen ist die Beurteilung der erstellten Evaluationen allein anhand des LLM-Judge nur bedingt aussagekräftig, was eine zweite Instanz wünschenswert macht.

### **3.4.2 Human as a Judge**

Um eine valide Einschätzung der Performance zu erhalten, ist es hilfreich, die generierten Judgements mit einer händischen Beurteilung abzugleichen. Wie bereits in Kapitel 3.2 erläutert, wurden aus diesem Grund zehn repräsentative Requirements aus dem firmeninternen Datensatz entnommen. Jedes dieser Requirements wurde individuell jeweils von mehreren LLMs und von einem (menschlichen) RE-Experten mithilfe der definierten Metriken bewertet. Daran anschließend erfolgte eine Betrachtung der Abweichungen für die jeweiligen Ratings der LLMs zum Experten. Auch hier lässt sich wieder das Argument anführen, dass eine Einzelmeinung für eine repräsentative Bewertung nur eine begrenzte Aussagekraft bietet. Dennoch ließ sich in Verbindung mit den gesammelten Beurteilungen ein grundlegendes Bild über die Performance der verschiedenen Ansätze skizzieren.

## 4 Ergebnisse

Für die Performance-Evaluierung wurden zuvor zwei verschiedene Ansätze eingeführt. Anhand der generierten Daten der Requirement-Bewertungen lassen sich durch geeignete Visualisierung jedoch noch weitere Indikatoren für deren Vertrauenswürdigkeit prüfen, auf die im Folgenden eingegangen werden soll.

### 4.1 Quality Separation Index

Eine naheliegende Erwartungshaltung an die Evaluierung durch ein LLM wären unterscheidbare Tendenzen bei der Bewertung der verschiedenen Datensätze. So wäre es wünschenswert, dass schlechte Requirements im Schnitt schlecht bewertet würden und gute Requirements entsprechend besser. Die Einstufungen „gut“ und „schlecht“ sind hier natürlich nicht eindeutig, jedoch kann für die entsprechenden Datensätze zumindest ein relativer Qualitätsunterschied angenommen werden, was dieser Analyse genügt.

Die in Abbildung 7 dargestellten Grafiken untersuchen dieses Kriterium für die beiden Ansätze, wobei jeweils in blau und orange die Dichteverteilung der Gesamtbewertungen für die Datensätze *bad\_requirements* und *good\_requirements* aufgetragen sind. Dabei wurden ungefähr 30 Stichproben pro Datensatz untersucht.

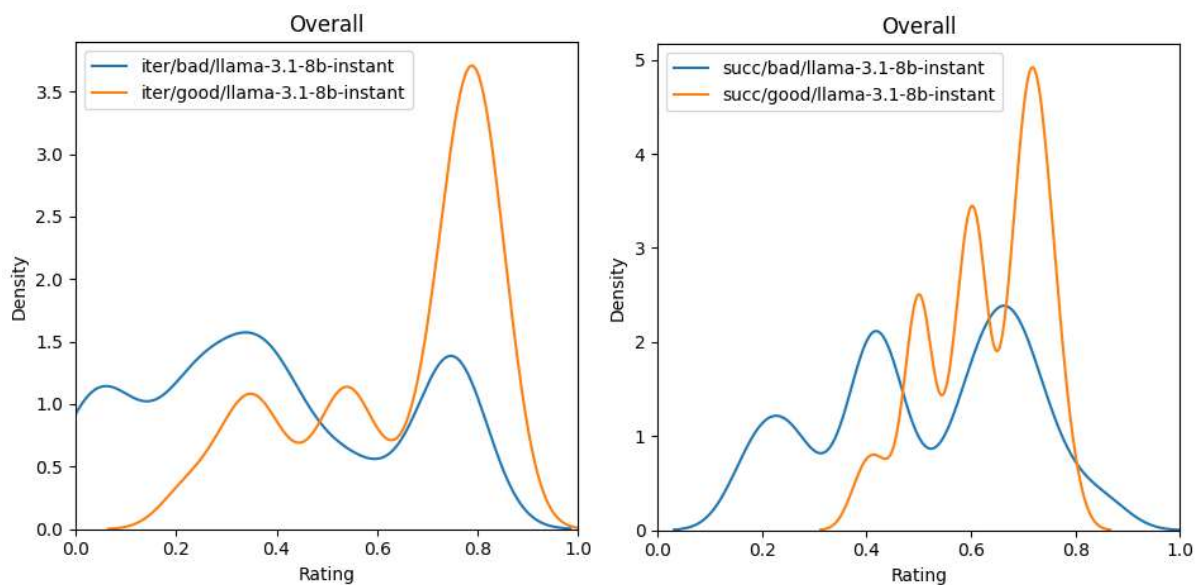


Abbildung 7: Llama8b – Verteilung der Gesamtratings bei guten und schlechten Requirements

Man erkennt, dass es bei der sukzessiven Herangehensweise keine klare Trennung zwischen guten und schlechten Requirements gibt. Die Verteilung des iterativen Ansatzes wirkt mit den beiden Maxima bei ca. 0.3 (bad) und 0.8 zwar etwas plausibler, jedoch zeigt der zweite Peak bei ca. 0.75 für die schlechten Requirements hier ebenfalls eine vermeintliche Ungenauigkeit auf. Dieses Verhalten lässt sich teilweise durch Stichproben erklären.

```

"requirement":
    "The Search functionality should be able to search for disputes by any or all of the
    following: dispute number, merchant account number, cardmember account number, issuer
    number. Also, the search should allow the user to filter by date range, dispute type,
    case status, and dispute reason code, with the option to sort the results by relevance
    or date.",
"Atomicity": {
"rating": 4,
"comment":
    "The requirement focuses on a single, distinct need, functionality, or constraint.
    However, it includes minor compound phrasing in the filtering options."
}

```

Quellcode 6: Sukzessiv/Llama 8b – inkorrekte Bewertung der Atomizität

Das in Quellcode 6 enthaltene Requirement ist hochgradig nicht-atomar, da es mehrere Funktionalitäten beschreibt (Suchkriterien, Filter und Sortierung), dennoch wird es mit dem sukzessiven Ansatz durch das *Llama 8b* Modell mit einem Rating von 4 sehr positiv bewertet. Auch in der Begründung werden die Mängel nicht hinreichend erkannt. Auffällig ist außerdem, dass häufig einfach der exakte Wortlaut der Rating- oder Metrik-Definition kopiert wird, was auf eine mangelnde Eigenleistung in der Analyse hindeutet. Im Gegensatz dazu findet das Modell mit dem iterativen Ansatz eine etwas treffendere Analyse:

```

"Atomicity": {
    "rating": 2,
    "comment":
        "The requirement includes compound statements and multiple objectives, making it
        difficult to address or test independently. It describes multiple search
        functionalities (searching by various fields, filtering, and sorting) within a
        single statement."
}

```

Quellcode 7: Iterativ/Llama 8b - passende Bewertung der Atomizität

Das folgende Beispiel stimmt mit dem zuvor analysierten Verhalten überein und bestärkt den Eindruck der erwähnten Schwäche des iterativen Ansatzes.

```

"requirement":
    "The Disputes System must send a confirmation email to the user when a chargeback
    request is made, with some details.",
"Unambiguity": {
    "rating": 4,
    "comment":
        "This requirement is mostly clear and allows for only one reasonable
        interpretation. However, the phrase 'with some details' is somewhat vague and
        could be open to varying interpretations. Stakeholders might disagree on what
        constitutes 'some details'."
},

```

Quellcode 8: Iterativ/Llama 8b - unstimmige Begründung des Metrik Ratings



Hier wird *Unambiguity* sehr positiv mit 4 bewertet, obwohl das Model korrekt den sehr mehrdeutigen Ausdruck „some details“ kritisiert. Dennoch erklärt es das Requirement als mehr oder weniger klar, was aufgrund des mehrdeutigen Ausdrucks jedoch faktisch nicht gegeben ist. Da ein ähnliches Ergebnis hier auch beim sukzessiven Ansatz festgestellt werden konnte, lässt sich darauf schließen, dass eine akkurate Bewertung nicht immer vorliegt. Dies deckt sich auch mit den zuvor gezeigten Graphen, wo sich je nach Prompt zwar Tendenzen erkennen ließen, aber dennoch eine gewisse Streuung auftrat. Zieht man nun anstelle des *Llama 8b* das größere *Claude Haiku* Modell von *Anthropic* heran, zeigen sich die in Abbildung 8 dargestellten starken Veränderungen.

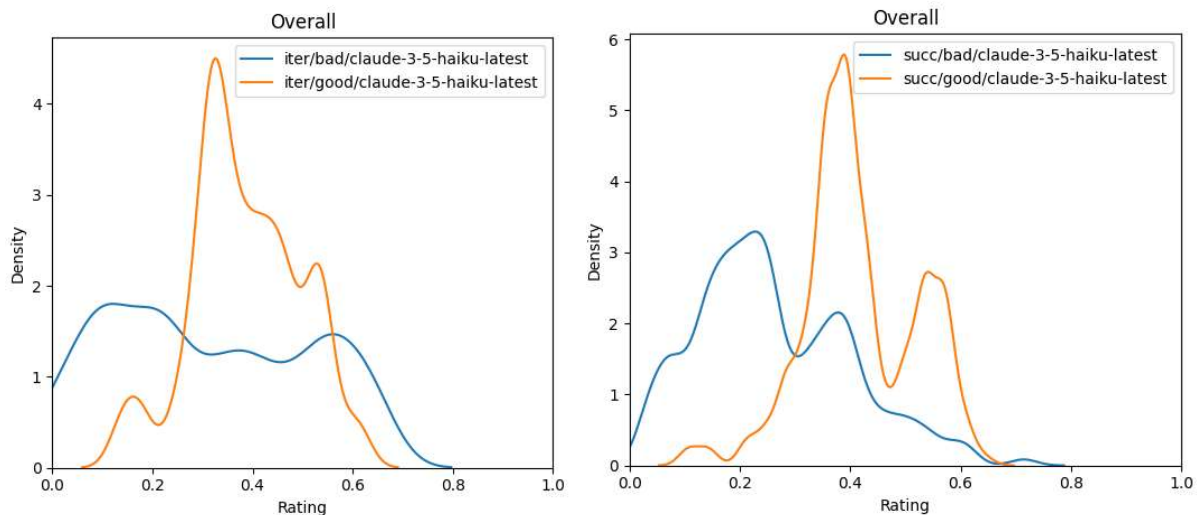


Abbildung 8: Claude Haiku – Verteilung der Gesamtratings bei guten und schlechten Requirements

Dabei lässt sich erkennen, dass die Bewertung allgemein deutlich strenger als zuvor ist. Interessant ist darüber hinaus, dass hier der sukzessive Ansatz gemäß der rechten Grafik deutlich besser zu funktionieren scheint, da im Gegensatz zum iterativen (links) eine klare Trennung zwischen guten und schlechten Requirements erkennbar ist. Aus dieser Betrachtungsweise heraus scheint das Ergebnis stark von der Wahl des LLMs abzuhängen, was nicht unbedingt für eine vertrauenswürdige Evaluierung spricht. Daraus ergibt sich ein weiterer Indikator.

## 4.2 Cross Model Consistency

Wie bereits dargelegt, steigt die Plausibilität einer Meinung, wenn sie von mehreren Experten geteilt wird. Gleichmaßen ist ein LLM vertrauenswürdiger, wenn seine Bewertungen denen anderer Modelle ähneln. Die Ähnlichkeit zwischen den hier geführten Bewertungen lässt sich jedoch nicht allein anhand des Gesamtratings beurteilen, denn hierbei werden eventuelle Differenzen unter den einzelnen Metriken, welche sich im Mittel ausgleichen, nicht berücksichtigt.

Beim Versuch, die Ratingverteilungen aller Metriken separat zu analysieren, verliert man schnell den Überblick. Stattdessen wurden die Metrik-spezifischen Ratingdifferenzen zu einer eindimensionalen Verteilung zusammengeführt und in folgendem Histogramm für verschiedene Modelle aufgetragen. Hierbei wurden die beiden Prompt-Ansätze getrennt ausgewertet und als Referenz jeweils das größte zur Verfügung stehende Modell, *Claude Sonnet* von *Anthropic*, gewählt.

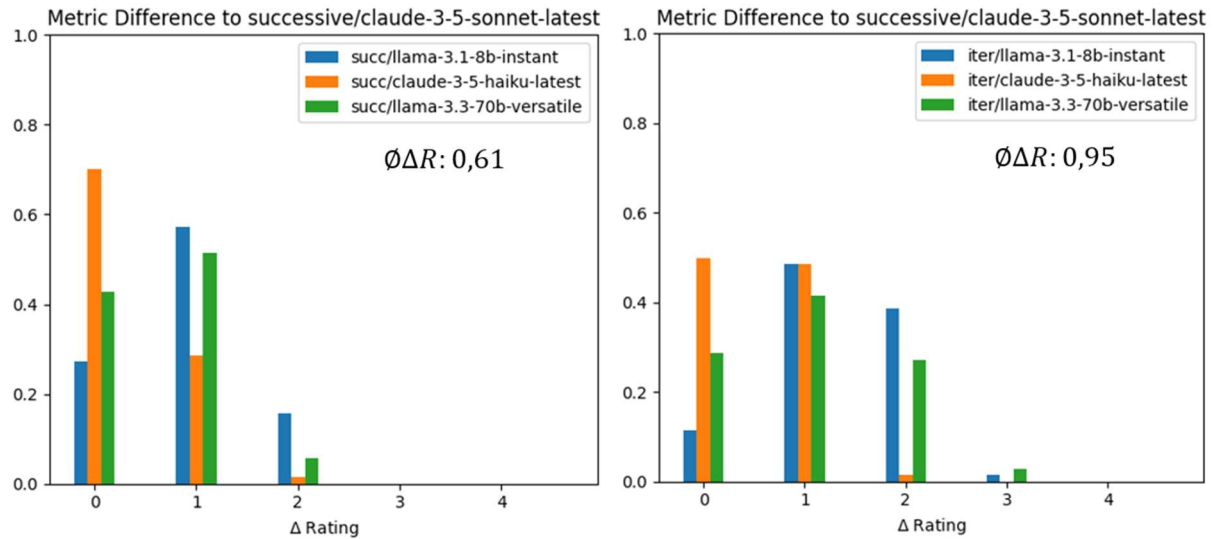


Abbildung 9: Metrik spezifische Ratingdifferenzen zwischen verschiedenen LLMs

Wie in Abbildung 9 zu erkennen ist, schneidet der sukzessive Ansatz hier deutlich besser ab. Die meisten Metrik-Wertungen stimmen überein oder unterscheiden sich nur um einen Zähler. Auf der anderen Seite verzeichnet der iterative Ansatz auch einige größere Abweichungen. Dies führt zu einer höheren durchschnittlichen Rating-Differenz von 0,95. Mit einem Schnitt von 0,61 wirkt der sukzessive Ansatz damit robuster gegenüber der Wahl des LLMs.

### 4.3 Overall Rating Diversity

Der nächste Indikator untersucht die Diversität der Gesamtratings. Dazu wird eine Testreihe von Requirements aus allen drei Datensets zusammengesetzt, um ein möglichst breites Qualitätsspektrum abzudecken. Dabei ergibt sich eine Erwartungshaltung, dass die Gesamtratings ebenfalls über einen möglichst breiten Wertebereich verteilt sein sollten.

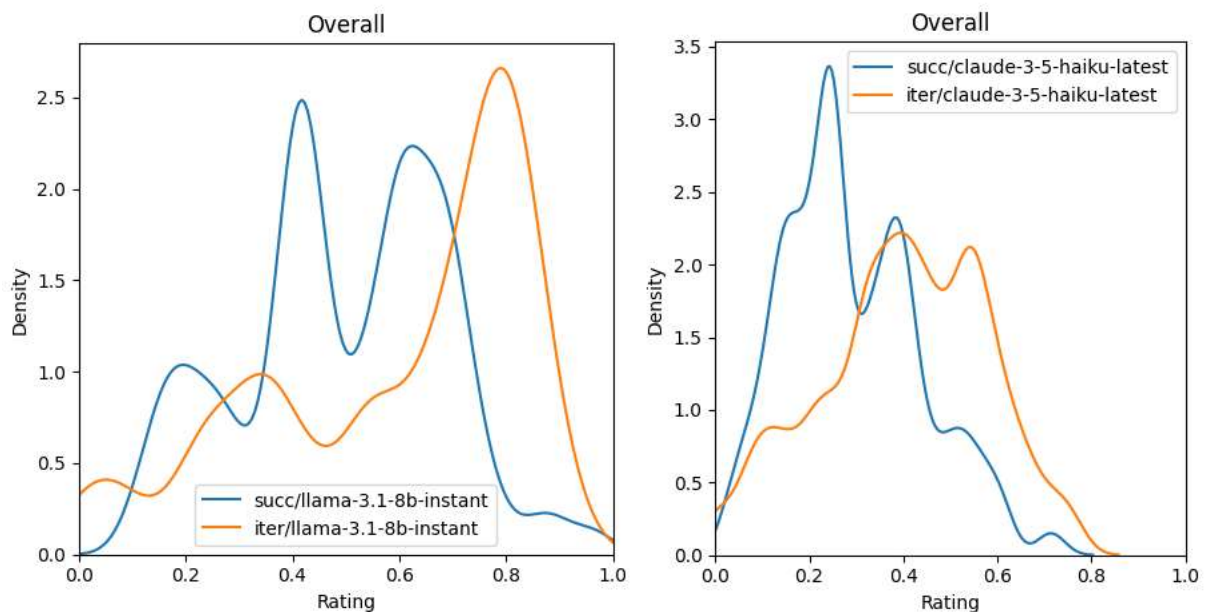


Abbildung 10: Verteilung der Gesamtratings bei breitem Qualitätsspektrum

Wie in Abbildung 10 zu erkennen, weist der iterative Ansatz eine etwas größere Varianz im Gesamtrating auf, was die Vermutung nahelegt, dass dieser eine differenziertere Analyse ermöglicht. Im Gegensatz dazu ergab der sukzessive Ansatz mit dem *Llama 8b* Modell wenig bis keine Bewertungen außerhalb des „mittelmäßigen“ Bereichs [0.2, 0.8]. Auffällig ist auch, dass sowohl dieser Ansatz als auch die Verwendung des *Haiku*-Modells insgesamt für eine strengere Bewertung sorgen. Auf der anderen Seite wirkt die Kombination des iterativen Ansatzes mit *Llama 8b* ein wenig zu optimistisch.

## 4.4 LLM as a Judge

Verwendet man ein LLM, um die Performance eines anderen LLMs zu bewerten, sollte zuvor auch die Zurechnungsfähigkeit des Judge Models überprüft werden. Da sich die LLM-basierte Bewertung bereits als durchaus fehleranfällig erwiesen hat, sind dementsprechend kritische Beurteilungen zu erwarten. Daher wird stichprobenartig die negativste Beurteilung eines Satzes von Requirement-Bewertungen betrachtet.

"original\_requirement":

"The Disputes System shall provide view access capability for users who have a username starting with 'A' or 'B' or 'C' ... or 'Z' or 'a' or 'b' or 'c' ... or 'z' to the application.",

"Correctness": {

"accuracy\_of\_rating": 1,

"comment\_on\_accuracy":

"The SLM's rating of 5 (Excellent) does not align with the metric definition and scale because the requirement, as stated, does not define a genuine system function or need in a clear and feasible manner. The requirement seems overly broad and does not specify what 'view access capability' entails or how it aligns with established standards or domain knowledge. Thus, the rating is significantly flawed and incorrect."

}

Quellcode 9: kritische Beurteilung des LLM as a Judge

Wie in Quellcode 9 zu sehen ist, fällt diese in der Tat und zurecht sehr negativ aus. Der Judge erkennt das unangemessene Rating von 5 für die *Correctness*-Metrik bei einem Requirement, welches schlichtweg keine sinnvolle Funktionalität definiert. Wie bereits in Quellcode 1 aufgezeigt, ist die Systemfunktionalität allerdings eine zwingende Voraussetzung für ein solches Rating im Kontext dieser Metrik. Um dieses Ergebnis jedoch richtig einordnen zu können, sollte im Gegenzug auch für die Fähigkeit, exzellente Performances anzuerkennen, ein Nachweis erbracht werden. Hierzu wird äquivalent die positivste Beurteilung untersucht, welche in Quellcode 10 dargestellt ist.

```

"original_requirement":
  "The system shall refresh the display when it feels like it.",
"Verifiability": {
  "accuracy_of_rating": 5,
  "comment_on_accuracy":
    "The SLM's rating of 1 (Very Poor) for verifiability is accurate because the
    requirement 'The system shall refresh the display when it feels like it.' cannot
    be objectively tested or measured. The phrase 'when it feels like it' is highly
    subjective and does not provide any clear, measurable success criteria, aligning
    perfectly with the definition of a 'Very Poor' rating."
}

```

Quellcode 10: Positive Beurteilung des LLM as a Judge

Auch hier ist das Urteil des Judge gerechtfertigt. Das bewertete Requirement ist offensichtlich schlecht testbar. Somit ist die Metrik-Wertung von 1 (Very Poor) passend und wird ebenso treffend als „Exzellent“ abgesegnet. Das *LLM as a Judge* ist also in der Lage, sowohl schwache als auch starke Performances korrekt zu erkennen und bildet damit eine zuverlässige Referenz. Zur Performance-Evaluierung wurden insgesamt 90 Requirement-Bewertungen aus allen drei Datensätzen vom Judge-Modell beurteilt. Abbildung 11 zeigt ein Streudiagramm über die Gesamtratings der Requirements (Evaluation) und deren Bewertungen (Judgement). Damit lässt sich die Qualität der Bewertungen für verschiedene Ansätze vergleichen. Darüber hinaus visualisiert es aber auch, inwiefern diese möglicherweise von der Güte der Requirements abhängen.

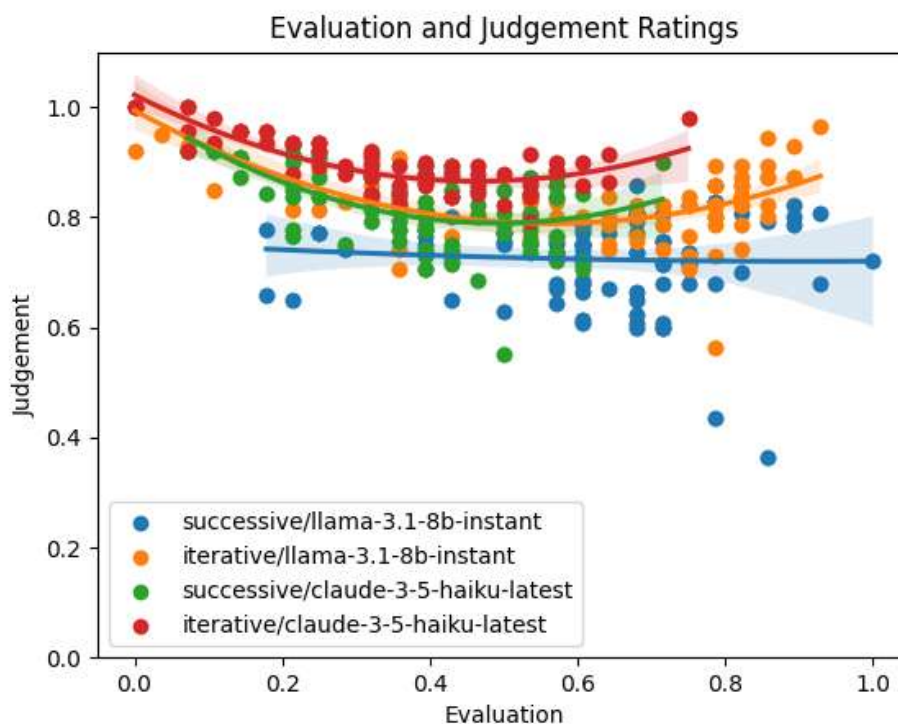


Abbildung 11: Beziehung zwischen Requirement und Bewertungsqualität

Wie sich zeigt, hebt der iterative Ansatz die Performance des *8b*-Modells ungefähr auf das Niveau des *Haiku*-Modells mit dem sukzessiven Ansatz an. Bemerkenswert sind einige Ergebnisse mit exzellentem Gesamtrating (Judgement: 1.0). Tatsächlich wird hier jedoch eine gewisse Abhängigkeit deutlich,

sodass sich bei sehr schlechten und sehr guten Requirements eine bessere Performance bei der Bewertung ergibt als bei durchschnittlichen Requirements. Dies ist nicht unbedingt wünschenswert, da ein solches Tool in der Realität gerade für mittelklassige Requirements interessant sein wird, bei denen Verbesserungen hilfreich sind, während das Unterschreiten einer gewissen Grundqualität im professionellen Bereich auch eher unwahrscheinlich ist. Allerdings liegt das Minimum der Trendlinie immer noch auf einem akzeptablen Niveau von ca. 0.8.

Im Gegensatz dazu liefert der sukzessive Ansatz mit dem *8b*-Model zwar eine etwas konstantere Performance. Dennoch ist zu erkennen, dass keine einzige Bewertung unter dem Gesamtrating von 0.2 liegt, obwohl das gegebene Datenset dies definitiv erwarten lässt, was sich auch in den Ergebnissen der anderen Testreihen widerspiegelt. Zudem liegen die Gesamtratings im Schnitt deutlich unter denen der anderen Testobjekte und auch die Ausreißer deuten auf grobe Fehleinschätzungen in der Requirement-Bewertung hin.

Auch wenn diese Form der Evaluierung zuvor als zurechnungsfähig eingestuft wurde, basierte dies lediglich auf Stichproben und ein zur Beurteilung verwendetes LLM kann ebenso Fehler machen wie bei der eigentlichen Aufgabe, Requirements zu bewerten. Daher sind die hier gezeigten Ergebnisse mit Vorsicht zu betrachten, weshalb im Folgenden zusätzlich menschliche Expertise in die Performance-Evaluierung miteinbezogen wird.

## 4.5 Human as a Judge

Die untenstehende Grafik zeigt die Dichteverteilung der Gesamtratings für die 10 Requirements des Benchmark-Datensatzes. Neben den üblichen LLM-basierten Bewertern wurde der Experte gebeten, ein eigenes Rating als Referenz abzugeben, welches hier in lila (human) aufgetragen ist.

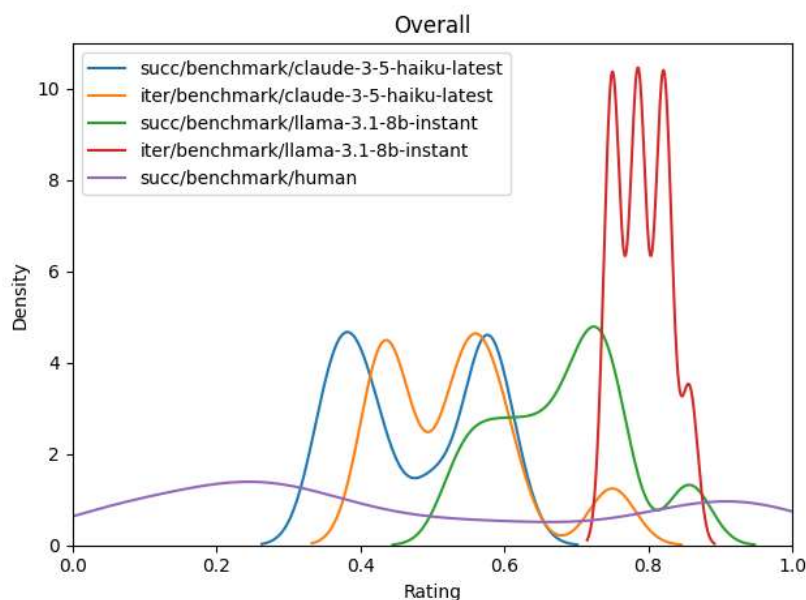


Abbildung 12: Verteilung der Gesamtratings im Vergleich mit menschlichem Experten

Vergleicht man die einzelnen Kurven, kann man sofort eine deutliche Meinungsverschiedenheit zwischen Experte und den LLMs ausmachen. Während sich die Sprachmodelle eher auf den mittelmäßigen bis guten Bereich begrenzen, fällt die Wertung des Experten deutlich variabler und insgesamt auch pessimistischer aus. Auffällig ist aber auch die Varianz zwischen den einzelnen LLM-

basierten Ratings. Dabei ist die Abhängigkeit vom Ansatz vor allem bei dem kleineren *Llama 8b* Modell zu verzeichnen, während sich bei *Claude Haiku* nur geringe Unterschiede ergeben. Nach wie vor bestätigt sich aber die Tendenz, dass letzteres Modell in Kombination mit dem sukzessiven Ansatz in der strengsten Bewertung resultiert. Damit zeigt diese die vermeintlich größte Übereinstimmung mit dem Experten.

Jedoch zeigt diese Darstellung nicht die ganze Wahrheit, denn sie berücksichtigt nicht die tatsächliche Übereinstimmung pro Requirement, sondern nur die allgemeine Häufigkeit bestimmter Ratings. Einen besseren Einblick bietet in diesem Fall die in Abbildung 13 gezeigte Dichteverteilung der Ratingdifferenzen zwischen LLM und dem Experten.

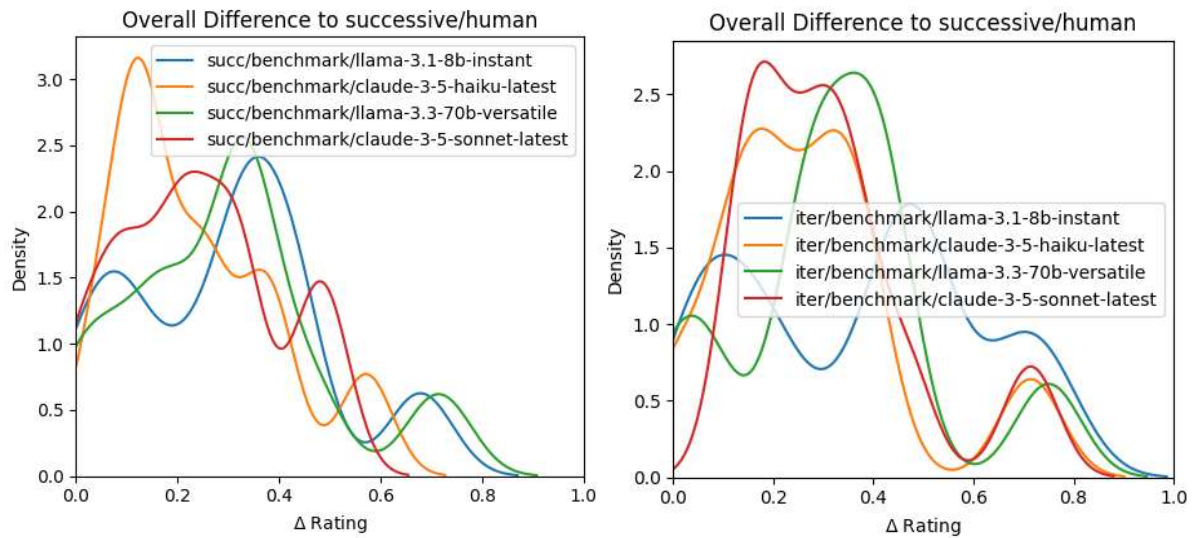


Abbildung 13: Verteilung der Gesamtratingdifferenzen zum menschlichen Experten

Neben den beiden bisher ausgewerteten Modellen wurden hier mit *Llama 70b* und *Claude Sonnet* noch weitere Testobjekte herangezogen. Trotzdem sticht hier der sukzessive Ansatz mit dem *Haiku* Modell noch deutlicher heraus und die Übereinstimmung zum Experten wirkt noch größer als zuvor. Dessen Peak ist deutlich über den anderen bei einer passablen Abweichung unter 0.2. Auf der Seite der iterativen Ansätze sind zwar ähnlich positionierte Peaks zu verzeichnen, jedoch fallen diese deutlich später in Richtung größerer Abweichungen ab.

Wie in Kapitel 4.2 bereits erläutert, lassen sich Ähnlichkeiten in den Bewertungen noch differenzierter anhand der Metrik-spezifischen Ratingdifferenzen untersuchen, welche in den nachfolgenden Histogrammen aufgetragen sind.



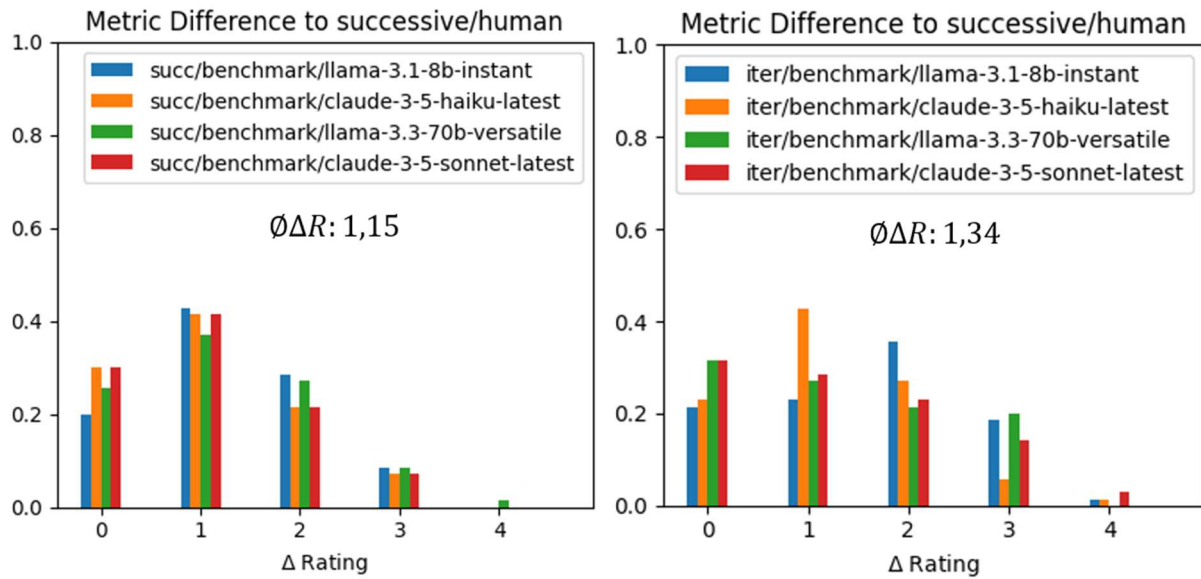


Abbildung 14: Metrik-spezifische Ratingdifferenzen zwischen LLM und Experte

Hier wird noch einmal deutlich, dass der sukzessive Ansatz durch die strengere Tendenz im Allgemeinen näher an der Meinung des Experten liegt. Es bestätigt sich auch eine verbesserte Übereinstimmung unter der Verwendung der *Anthropic*-Modelle. Diese äußert sich allerdings nicht so deutlich, wie es anhand der vorherigen Betrachtung erschien.

## 4.6 Improvement Convergence

Neben der Bewertung der Requirements soll das LLM auch ein Verbesserungsvorschlag liefern. Die Qualität dieser Vorschläge fließt nicht in das Gesamtrating des *LLM as a Judge*. Dessen Output beinhaltet zwar eine separate Bewertung in Form eines einzelnen 5er Ratings, jedoch erwies sich dies nicht als sinnvoll, was folgende Verteilung dieser Ratings über die drei Datensätze zeigt.

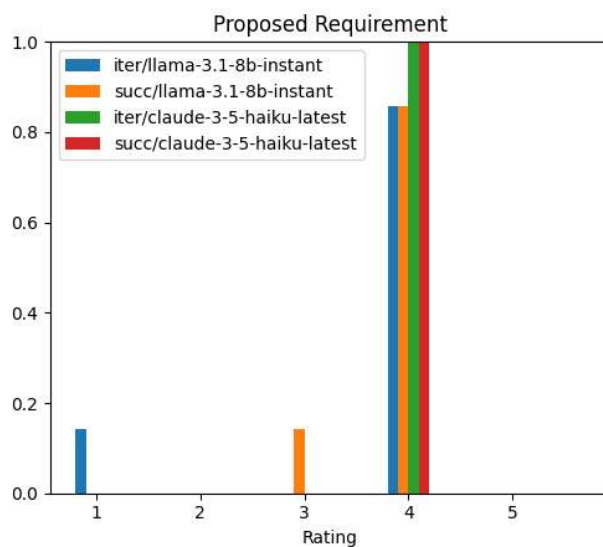


Abbildung 15: Rating-Verteilung der verbesserten Requirements laut "Llama 70b as a Judge"

Die Verbesserungsvorschläge werden fast ausschließlich mit 4 bewertet, was nicht sehr plausibel erscheint. Trotzdem sollte diese Qualität in irgendeiner Form bei der Performance-Evaluierung berücksichtigt werden.

Wünschenswert wäre es, wenn der Verbesserungsvorschlag zumindest nach einer Selbsteinschätzung eine perfekte Bewertung erhält. Dies ist allerdings nicht realistisch, da es schwierig sein mag, immer alle sieben Metriken perfekt einzuhalten. Eine alternative Zielvorgabe bieten hierfür die in Kapitel 3.3.2 definierten Akzeptanzschwellen. In einer Testreihe wurde also ein zufälliges Requirement (das möglichst noch Raum zur Verbesserung bietet) gewählt, das so lange iterativ verbessert wird (durch Bewertung des vorherigen Vorschlags), bis jedes Rating mindestens auf der jeweiligen Akzeptanzschwelle oder darüber liegt. Als Indikator für die Vertrauenswürdigkeit des Verbesserungsvorschlags wird somit untersucht, ob und wie schnell dieser Prozess gegen eine selbsteingeschätzte akzeptable Qualität konvergiert.

Nachfolgend wird das ursprüngliche Requirement sowie die Ergebnisse der Iterationsschritte für die einzelnen Ansätze und Modelle präsentiert. Anschließend werden diese in einer Gegenüberstellung in Tabelle 1 noch einmal zusammengefasst.

**Requirement:**

*"The product will be available most of the time. If the user has a computer, it should work, probably."*

**Sukzessiv (Claude Haiku):****Erste Verbesserung:**

*"The product shall be available 99.9% of the time and compatible with Windows 10 and 11 computers meeting minimum system requirements of 8GB RAM, 2.0 GHz processor, and 20GB free disk space."*

**Gesamt-Rating:** 0.714 | unterschrittene Akzeptanzschwelle(n): Correctness (4)

**Zweite Verbesserung:**

*"The software shall be compatible with Windows 10 and 11 operating systems on computers with minimum specifications of 8GB RAM, 2.0 GHz processor, and 20GB free disk space."*

**Gesamt-Rating:** 0.786 | unterschrittene Akzeptanzschwelle(n): Correctness (4)

**Dritte Verbesserung:**

*"The software shall be compatible with 64-bit Windows 10 and 11 operating systems, requiring a minimum of 8GB RAM, Intel or AMD x86-64 processor with 2.0 GHz clock speed, and 20GB of free SSD or HDD storage."*

**Gesamt-Rating:** 0.893 | unterschrittene Akzeptanzschwelle(n): -

**Iterativ (Claude Haiku)****Erste Verbesserung:**

*"The product is compatible with Windows 10/11, macOS (latest two versions), and Linux (Ubuntu LTS), running on computers with the following minimum specifications:*

- *Processor: Intel Core i5 or equivalent AMD processor*
- *RAM: 8GB*
- *Storage: 256GB SSD*



- *Internet Connection: Minimum 10 Mbps stable broadband connection*

*The system will provide clear error messages and support mechanisms for any operational issues, ensuring reliable and consistent performance across supported platforms. "*

**Gesamt-Rating:** 0.714 | unterschrittene Akzeptanzschwelle(n): Consistency (4), Atomicity (2)

#### **Zweite Verbesserung:**

*"The product shall support the following operating systems:*

- *Windows 10 and Windows 11*
- *macOS (current and previous two versions)*
- *Linux Ubuntu Long-Term Support (LTS) releases"*

**Gesamt-Rating:** 0.75 | unterschrittene Akzeptanzschwelle(n): Consistency (4)

#### **Dritte Verbesserung:**

*"The product shall support the following operating systems:*

- *Windows 10 (version 21H2 and later) and Windows 11 (version 22H2 and later)*
- *macOS 13 (Ventura), macOS 12 (Monterey), and macOS 11 (Big Sur)*
- *Linux Ubuntu Long-Term Support (LTS) releases: 20.04 LTS, 22.04 LTS"*

**Gesamt-Rating:** 0.893 | unterschrittene Akzeptanzschwelle(n): -

#### **Sukzessiv / Llama 8b**

##### **Finale Verbesserung (3):**

*"The product will be available at least 99% of the time on any computer with a Windows or macOS operating system, as measured by a standard uptime metric (e.g., MTBF) and verified through regular monitoring and testing."*

**Gesamt-Rating:** 0.607 → 0.714 → 0.857

#### **Iterativ / Llama 8b**

##### **Zweite Verbesserung:**

*"The product will be available at least 99.9% of the time on devices with a compatible operating system, defined as any device running Windows, macOS, or Linux."*

**Gesamt-Rating:** 0.571 → 0.786 | unterschrittene Akzeptanzschwelle(n): Correctness (4)

##### **Dritte Verbesserung:**

*"The product will be available at least 99.9% of the time on devices with a compatible operating system, defined as any device running Windows, macOS, or Linux, and the technical feasibility of achieving this availability will be ensured through regular monitoring and maintenance."*

**Gesamt-Rating:** 0.679 | u. A.: Correctness (4), Consistency (4), Atomicity (2)

## Gesamtergebnis

Tabelle 1: Gegenüberstellung der Ansätze und Modelle bei der iterativen Requirement Verbesserung

	Konvergenz (Iterationen)	Direkte Verbesserung	Maximale Verbesserung	Maximale Verbesserung (Claude Sonnet)
<b>Sukzessiv/Haiku</b>	Ja (3)	0.714	0.893	0.75
<b>Sukzessiv/Llama 8b</b>	Ja (3)	0.607	0.857	0.607
<b>Iterativ/Haiku</b>	Ja (3)	0.714	0.893	0.714
<b>Iterativ/Llama 8b</b>	nein	0.571	0.786	0.714

Mit Ausnahme des iterativen Ansatzes mit *Llama 8b* konnten alle Bewerter mit monoton steigendem Gesamt-Rating eine Konvergenz nach drei iterativen Verbesserungsschritten erzielen. Das *Haiku* Modell konnte ansatzunabhängig sogar gleich mit der ursprünglichen Bewertung einen Verbesserungsvorschlag liefern, der zumindest der mittleren Akzeptanzschwelle (nach Abbildung 4) entspricht, auch wenn einzelne Metrik-Ratings noch darunter lagen. Ebenfalls bei der maximalen Verbesserung liegt *Haiku* mit beiden Ansätzen knapp vorne.

Diese Selbsteinschätzungen sind ein Maß für die Stabilität und Zuversichtlichkeit, eignen sich jedoch nicht gänzlich zur Beurteilung der tatsächlichen Qualität. Der von Wataoka et al. [15] aufgezeigte *Self-Preference Bias* könnte dazu führen, dass die Sprachmodelle die eigene verbesserte Version eines Requirements besonders gut bewerten. Daher wurden die verbesserten Requirements mit dem maximalen Rating noch einmal unabhängig von dem größten verfügbaren Modell (*Claude Sonnet*) bewertet. Tatsächlich sind die resultierenden Werte (rechte Spalte der Tabelle) im Allgemeinen nun deutlich unter den jeweiligen Selbsteinschätzungen. Die beste Bewertung erhält dabei das *Claude Haiku* Modell mit dem sukzessiven Ansatz. Auch wenn das Modell mit dem iterativen Ansatz eine ähnliche Performance zeigt, enttäuscht dieser mit dem *Llama 8b* Modell. In dieser Kombination neigen die Verbesserungsvorschläge dazu, immer länger, aber nicht besser zu werden. Auch nach einer Vielzahl von Versuchen werden die Akzeptanzschwellen weiterhin unterschritten. Dieser Ansatz ist also zumindest für kleinere LLMs nicht geeignet oder ausgereift genug, um Requirements qualitativ zu verbessern.

## 4.7 Zusammenfassung

Anhand der gesammelten Aspekte erhält man einen Eindruck der Performance innerhalb der jeweiligen Indikatoren, welcher in Tabelle 2 dargestellt ist.

Tabelle 2: Vergleich der Ansätze anhand mehrerer Indikatoren

Indikator \ Ansatz	Sukzessiv	Iterativ
<b>Quality Separation Index</b>	Llama 8b: Schwach Haiku: Gut	Llama 8b: Mittel Haiku: Mittel
<b>Cross Model Consistency</b>	Besser ( $\Delta R: 0.61$ )	Schwächer ( $\Delta R: 0.95$ )
<b>Overall Rating Diversity</b>	Niedrig (streng)	Hoch (optimistisch)
<b>LLM as a Judge</b>	Llama 8b: Mittel Haiku: Gut	Llama 8b: Gut Haiku: Exzellent
<b>Human as a Judge</b>	besser ( $\Delta R: 1.15$ )	Schwächer ( $\Delta R: 1.34$ )
<b>Improvement Convergence</b>	gut	Ungeeignet

Dabei wird klar, dass die Ansätze je nach Indikator unterschiedlich gut abschneiden. Grundsätzlich lässt sich festhalten, dass die iterative Herangehensweise eine größere Streuung und differenziertere Analyse erlaubt. Auch die Performance-Analyse durch einen LLM-Judge schreibt dem iterativen Ansatz eine höhere Genauigkeit zu. Dahingegen stimmt der sukzessive Prompt aufgrund seiner strengeren Bewertung enger mit der Evaluierung durch einen menschlichen Experten überein. Außerdem liefert es über mehrere Modelle hinweg konsistentere Ergebnisse. Zwischen den Sprachmodellen selbst lassen sich ebenfalls Unterschiede aufzeigen, wobei dem größeren *Haiku*-Modell grundsätzlich eine bessere Performance über die beiden Ansätze hinweg zu attestieren ist.

Ein eindeutiges Fazit, welcher Ansatz überlegen ist, lässt sich folgerichtig anhand der Indikatoren nicht schließen. Vielmehr kommt es auf den jeweiligen Anwendungszweck an und welche Parameter für ebendiesen relevanter sind. Daraus lässt sich schließen, dass eine Kombination der beiden Ansätze, welche Elemente aus den jeweiligen Prompts enthält, potenziell zu einer weiteren Verbesserung führen könnte.

## 5 Zukünftige Entwicklung

Auf den bisherigen Ergebnissen aufbauend, bietet sich eine Reihe an Möglichkeiten, um das Projekt weiterzuentwickeln. Um dies effizient voranzubringen, wurden in der Implementierung bereits diverse Vorkehrungen getroffen, welche im Folgenden vorgestellt werden.

### 5.1 Dynamisches Prompt Template

Bei der Entwicklung der Prompts war es wichtig, verschiedene Versionen anzulegen, um kleine Veränderungen auf Performanceunterschiede zu vergleichen. Dabei wurden einzelne Elemente, wie Metrik- und Rating-Definition aber auch *Few-Shot*-Beispiele häufig wiederverwendet. Daher wurden diese in separaten Dateien festgehalten. Um die einzelnen Elemente flexibel in einen neuen Prompt einbauen zu können, wurde ein Tool implementiert, das die dynamische Verarbeitung bzw. Vervollständigung eines Prompt-Templates durch den Einsatz entsprechender Sektionen und Variablen ermöglicht.

#### 5.1.1 Sektionen und Variablen

Zur Veranschaulichung der Syntax und Funktionsweise dient ein nachfolgend gezeigter Ausschnitt aus dem Basistemplate, welches beispielhaft den Einsatz Metrik-bezogener Elemente demonstriert.

```
## Metric Definitions
[section/metric]:# (this section can be extended for each given metric)
[var/m_id]:# (int: e.g. 1,2,3,... | metric ID or evaluation step)
[var/m_name]:# (str: e.g. "Correctness" | metric name)
[var/m_definition]:# (str: e.g. "- is the requirement ...?" | metric definition as per
`metric_definitions.json`, while the list items are formatted as bullet points)
[var/m_rating]:# (equivalent to m_definition as per `rating_definitions.json`)
### {m_id}. {m_name}
#### Definition
{m_definition}
#### Rating Scale
{m_rating}
[section/metric end]:# (end of section)
```

Quellcode 11: Einsatz der Metrik Sektion im Prompt Template

Inspiziert durch die Fähigkeit der Sprachmodelle, ihre Antworten in *Markdown* zu formatieren, erschien es sinnvoll, Prompts ebenfalls auf diese Weise zu gestalten, um eine klare Struktur zu gewährleisten. Das Basistemplate dient selbst dazu, die grundsätzliche Funktionsweise sowie die der einzelnen Sektionen und Variablen zu erklären. Hierzu wurden Kommentare in der *Markdown* Syntax [...] :# (...) verwendet, welche in der Zielform nicht sichtbar sind und in der Vorverarbeitung durch das Tool entfernt werden. Darüber hinaus dienen die [section/...] Marker der separaten Verarbeitung eines bestimmten Bereichs. Dies ermöglicht eine flexible Erweiterung des Templates durch neue Sektionen mit einer getrennten Prozessorfunktion und lokalen Variablen. Im Falle der Metrik-Sektion kann beispielsweise der markierte Bereich für jede als Parameter übergebene Metrik multipliziert werden. Die in den Kommentaren aufgelisteten Variablen [var/<name>] können innerhalb der Sektion

beliebig und optional in der Form {<name>} eingesetzt werden. Durch eine Prozessorfunktion des Tools werden die Platzhalter dann für jede Metrik entsprechend den Beschreibungen dynamisch ausgefüllt.

## 5.1.2 Few Shot Sektion und RAG

Das Tool erlaubt darüber hinaus flexible Gestaltungsmöglichkeiten bei der Einbindung von *Few Shot* Beispielen mithilfe der entsprechenden Sektion und deren Variablen, wie nachfolgend gezeigt:

```
[section/few_shots]:# (removed, if `n_shots` = 0, can be extended for each given example)
```

Example few shots section

```
## Examples
```

```
[section/one_shot]:# (can be extended for each given example)
```

```
[var/os_id]:# (int: e.g. 1,2,3,...)
```

```
[var/os_rating]:# (str: e.g. "Poor" | maps rating to expression: [1,x] -> [1,3] -> {1->"Poor", 2->"Average", 3->"Excellent"})
```

```
[var/os_req]:# (str | requirement of the evaluation example)
```

```
[var/os_eval]:# (dict | evaluation example)
```

Example one shot section

```
### Example {os_id}: {os_rating} Requirement
```

```
**Requirement:**
```

```
> {os_req}
```

```
**Evaluation:**
```

```
```json
```

```
{os_eval}
```

```
```
```

```
[section/one_shot end]:# (end of section)
```

```
[section/few_shots end]:# (end of section)
```

Quellcode 12: Few Shot Sektion des Prompt Templates

Abhängig von einem Parameter `n_shots` wird die `few_shot` Sektion für den Wert `0` komplett ausgeblendet oder aber die darin eingebettete `one_shot` Sektion für `n_shots` Beispiele multipliziert. Für jedes Beispiel kann z.B. wie in Quellcode 12 das Requirement und die zugehörige Bewertung getrennt voneinander eingesetzt werden. Das Tool unterstützt hierbei zwei Möglichkeiten zur Einbindung von *Few Shots*. Diese können entweder im vornherein aus einer festen Liste innerhalb eines JSON-Files eingebaut werden oder während eines RAG basierten LLM-Aufrufs. In diesem Fall wird die markierte Sektion dem RAG-Modul übergeben, bevor sie im Prompt vor der Eingabe in das LLM zunächst durch einen Platzhalter {context} ersetzt wird. Beim Aufruf des LLMs generiert das RAG-Modul zum Input-Requirement passende Bewertungen aus einer Datenbank. Anschließend werden diese der `one_shot` Prozessorfunktion übergeben, um den Kontext entsprechend der zwischengespeicherten Template Sektion zu formatieren und in den dafür vorgesehenen Platzhalter einzusetzen.

## 5.2 Erweiterte Anwendung in Konversationsketten

Der zuvor behandelte iterative Ansatz entwickelte sich aus der Idee heraus, in einer Konversation mit dem LLM schrittweise zu einer optimalen Lösung zu gelangen. LLMs haben grundsätzlich die Fähigkeit, sich den vergangenen Verlauf zu merken, jedoch erwies sich diese gerade bei kleineren Modellen, die untersucht wurden, als sehr begrenzt und damit unpraktikabel. Daher war es notwendig, im letzten Prompt alle vorherigen Antworten bzw. die Teilbewertungen wieder einzubauen, damit das Modell aus diesen Informationen ein finales verbessertes Requirement herleiten kann. Auch wenn dieser Ansatz noch nicht perfekt funktioniert, zeigt er doch das Potential von Konversationsketten auf. Hierbei wären sehr unterschiedliche Architekturen denkbar, weshalb es für zukünftige Entwicklungen sinnvoll erschien, auch hierzu ein flexibles Framework zu implementieren.

### 5.2.1 EvaluationChain

Das als *EvaluationChain* getaufte System basiert auf der Verknüpfung von *ChainLink* Objekten, welche das zuvor beschriebene Template-Tool verwenden, um den Prompt in jedem Schritt dynamisch zu gestalten. Dabei kann jedes Kettenglied mit einem eigenen Template, Metriken und weiteren Parametern versehen werden. Während des sequenziellen Aufrufs durch die Kette, wird das Input-Requirement an jedes Element weitergereicht (wobei die tatsächliche Verwendung vom Einsatz der optionalen Variable im Template abhängt). Zusätzlich werden die Antworten in einer Liste gespeichert, um diese den nachfolgenden Gliedern zur Verfügung zu stellen. Dabei bestimmt jedes Glied selbst, welche vorherigen Antworten es benötigt. Dies kann entweder durch einzelne Indizes, mittels *Slicing* (Python-Feature) oder durch Übergabe einer Funktion der bisherigen Antwortliste definiert werden. Letzteres ermöglicht eine dynamischere Auswahl, welche z.B. beim iterativen Ansatz vollzogen wird. Hierbei hängt die Auswahl nämlich, wie bereits in Kapitel 3.3.5 erwähnt, von den Akzeptanzschwellen der jeweiligen Metriken ab. Um den tatsächlichen Einsatz und eine flexible Gestaltung der vorherigen Antworten in den Prompt eines bestimmten Gliedes zu ermöglichen, wurde im Template eine zusätzliche Sektion und entsprechende Prozessorfunktion eröffnet. Dadurch können wieder für jede Antwort bzw. Teilbewertung einzelne Bestandteile separat durch die jeweilige Variable in den Abschnitt eingebaut werden. Hierzu ist es allerdings notwendig, dass die Antworten eine bestimmte Struktur einhalten, denn die Prozessorfunktion erwartet bestimmte Einträge. Um dies zu gewährleisten und trotzdem eine gewisse Flexibilität zu bewahren, wurde ein *Evaluation* Interface implementiert, welches als LLM-Output-Wrapper dient.

### 5.2.2 Evaluation-Wrapper

Der *Evaluation-Wrapper* ermöglicht u.a. die Vorgabe einer Antwortstruktur, sowie eine automatisierte Kontrolle deren exakter Einhaltung. Die Vorgabe erfolgt durch Definition einer Dummy-Antwort, welche alle gewünschten Einträge und deren Datentypen enthält. Für den sukzessiven Ansatz sieht dies beispielsweise wie folgt aus:

```
class GeneralEval(EvalWrapper):
    """justified requirement evaluation based on different metrics"""
    def __init__(self, metrics):
        format_dummy = {
            "requirement": str,
            "evaluation": {
                m: {
                    "rating": int,
                    "comment": str
                } for m in metrics
            },
            "proposed_requirement": {
                "text": Optional[str],
                "justification": Optional[Union[str, dict]]
            },
        }
        super().__init__(format_dummy)
```

Quellcode 13: Format Dummy für die Antwort des sukzessiven Ansatzes

Die spezifische Struktur wird hierbei in einer Unterklasse `GeneralEval` des Wrapper-Interfaces definiert. Dieses fordert außerdem die Implementierung vorgegebener Methoden zur Verarbeitung bzw. Extrahierung möglicher Inhalte wie eines Gesamtratings oder eines Verbesserungsvorschlags (hier `proposed_requirement`). Der Aufruf des Wrappers erzeugt ein `Evaluation` Objekt, welchem die überschriebenen Methoden zusammen mit dem Format-Dummy und der eigentlichen Antwort übergeben werden. Dieses kann man sich als Python-Dictionary mit zusätzlichen Features vorstellen. Bei der Initialisierung wird zunächst die strukturierte Antwort in Form eines `dict` Objekts einer Methode übergeben, welche rekursiv die Einträge aller Unterebenen durchiteriert und deren Datentypen mit dem gegebenen Format-Dummy abgleicht. Somit kann jede beliebige Struktur auf Einhaltung geprüft werden. Basierend auf diesem und weiteren Checks, wie beispielsweise einem Abgleich des Input-Requirements und der möglichen Wiedergabe des LLMs in der Antwort, wird ein entsprechender Fehlercode bestimmt. Dieser wird dazu verwendet, die überschriebenen Methoden mit einer Validierung zu versehen, um u. a. Laufzeitfehler beim Zugriff auf möglicherweise fehlende oder falsche Einträge zu vermeiden.

Die Verwendung des *Evaluation-Wrappers* erlaubt somit eine variable Antwortstruktur und gleichzeitig einen einheitlichen und fehlersicheren Zugriff auf wiederkehrende Elemente mithilfe überschreibbarer, eingebetteter Methoden. Um auf den zuvor behandelten Anwendungsfall zurückzukommen, ist damit

auch das Einbinden von vorherigen Antworten in Prompts einer Konversationskette mithilfe der entsprechenden Prozessorfunktion gelöst.

Eine weitere Funktionalität des Wrappers bildet die dynamische Erzeugung eines *Schemas* basierend auf dem Format-Dummy. Ein solches wird bei einigen Sprachmodellen (z.B. von *Anthropic*) bei der Initialisierung benötigt, um einen bestimmten strukturierten Output sicherzustellen. Der Vorteil ist, dass bei dieser Methode keine detaillierten Formatanweisungen im Prompt enthalten sein müssen. Um in einer Konversationskette variable Antwortstrukturen zu ermöglichen, kann den *ChainLink* Objekten ein aus dem entsprechenden Wrapper erzeugtes *Schema* übergeben werden, mit welchem das LLM dann vor Aufruf des Kettenglieds neu initialisiert wird.

Die hier vorgestellten Module bieten in Kombination ein effizientes Tool, um neue Prompt-Ansätze für die Requirement-Bewertung zu entwickeln und zu verfeinern. Über bereits bestehende Implementierungen hinaus hält dieses Projekt jedoch noch andere Potenziale bereit, die im gegebenen Rahmen nicht untersucht werden konnten.

### 5.3 Weiterführende Ideen

Basierend auf dem durchgeführten Projekt bieten sich eine Reihe an Weiterentwicklungen, um den Funktionsumfang zu erweitern. Neben den bereits vorgestellten Techniken wie *Few-Shot-Prompting* kommt als weitere Methode das *Fine-Tuning* des Sprachmodells in Frage. Dieses birgt gleich aus einer Reihe an Gründen Potenzial. Chaubey et al. [16] konnten in ihrer Arbeit mithilfe von *Fine-Tuning* die höchste Genauigkeit gegenüber RAG und klassischen Prompt-Engineering-Ansätzen erreichen. Zwar erfordert es ungleich mehr Aufwand, kann aber im Kontext dieses Projekts vielfältig eingesetzt werden. Neben der bereits aufgezeigten Rollenzuweisung innerhalb des Prompts könnte dem LLM so tatsächlich tiefgehendes Wissen über den RE-Prozess nähergebracht werden. Hierdurch könnte das Modell relevante Aspekte besser identifizieren und Requirements basierend auf Best Practices aus dem Bereich evaluieren, da es dann als tatsächlicher RE-Experte fungiert. Gerade bei der Verwendung kleinerer LLMs könnte das Erlernen von Kenntnissen auf diesem Gebiet zu nennenswerten Verbesserungen beitragen. Auch könnte *Fine-Tuning* dafür verwendet werden, dem Modell domänenspezifisches Wissen des jeweiligen Anwendungszwecks anzueignen. Dies trägt dazu bei, dass LLMs das Vokabular und bestimmte Eigenschaften dieser Bereiche versteht [17]. Im Kontext dieses Projektes wären dies beispielsweise der Automotive-Bereich und die Softwareentwicklung. Gerade im Hinblick auf Metriken wie *Correctness* oder *Verifiability* könnten domänenspezifische Kenntnisse beispielsweise dazu beitragen, die technische Machbarkeit eines Requirements besser einzuschätzen. Somit könnte eine Erprobung im Rahmen dieses Projektes zu verbesserten Ergebnissen führen. Insgesamt stellt *Fine-Tuning* ein hilfreiches Werkzeug dar, um das LLM an den gewünschten Anwendungszweck anzupassen.

Neben der Untersuchung weiterer Methodiken lassen sich auch die Fähigkeiten des Projekts erweitern. Bisher wurden Requirements nur alleinstehend bewertet. Dies ist im Hinblick auf Metriken wie *Completeness* und *Precision* jedoch unter Umständen nicht ausreichend. So kann ein Requirement alleinstehend unvollständig oder unklar wirken, gleichzeitig kann es aber Teil eines Dokumentes sein, welches die vermeintlich fehlenden Details bereits beinhaltet. Wie in Kapitel 3.3.4 erwähnt, wurde dies im sukzessiven Ansatz bereits berücksichtigt, indem auf den fehlenden Kontext hingewiesen wird. Dies stellt allerdings keine zufriedenstellende Lösung dar, da hier nur eine wenig aussagekräftige Einschätzung getroffen wird, ob zusätzlicher Kontext nötig ist oder nicht. Anstatt wie bisher nur einzelne, atomare Requirements zu bewerten, könnte das Sprachmodell stattdessen auch aufgefordert werden, die Beziehungen eines ganzen Datensatzes untereinander zu evaluieren. Hierzu könnten die



Metriken von Honig [6] für Dokumente herangezogen werden und vergleichbar zur bisherigen Vorgehensweise dokumentenumfassende Metriken und Definitionen erstellt werden. Dadurch hätte man beispielsweise die Möglichkeit, die Konsistenz über einen gesamten Anforderungskatalog hinweg zu bestimmen. Auch lassen sich dadurch etwaige Widersprüche zwischen einzelnen Requirements aufdecken. Da die Anzahl der Token für die aufeinanderfolgende Bewertung über alle Requirements einer Liste hinweg vermutlich sehr schnell an das interne *Groq*-Limit stößt, müsste hier mit einem Gedächtnis gearbeitet werden. Dabei ist es jedoch fraglich, ob ein kleines Modell relevante Aspekte über eine derart große Zahl an Requirements beibehält.

Einen möglichen Lösungsansatz bietet in diesem Zusammenhang die Verwendung von *Hypothetical Document Embeddings*, kurz HyDE [18]. Diese könnten wie in Abbildung 16 dargestellt eingesetzt werden.

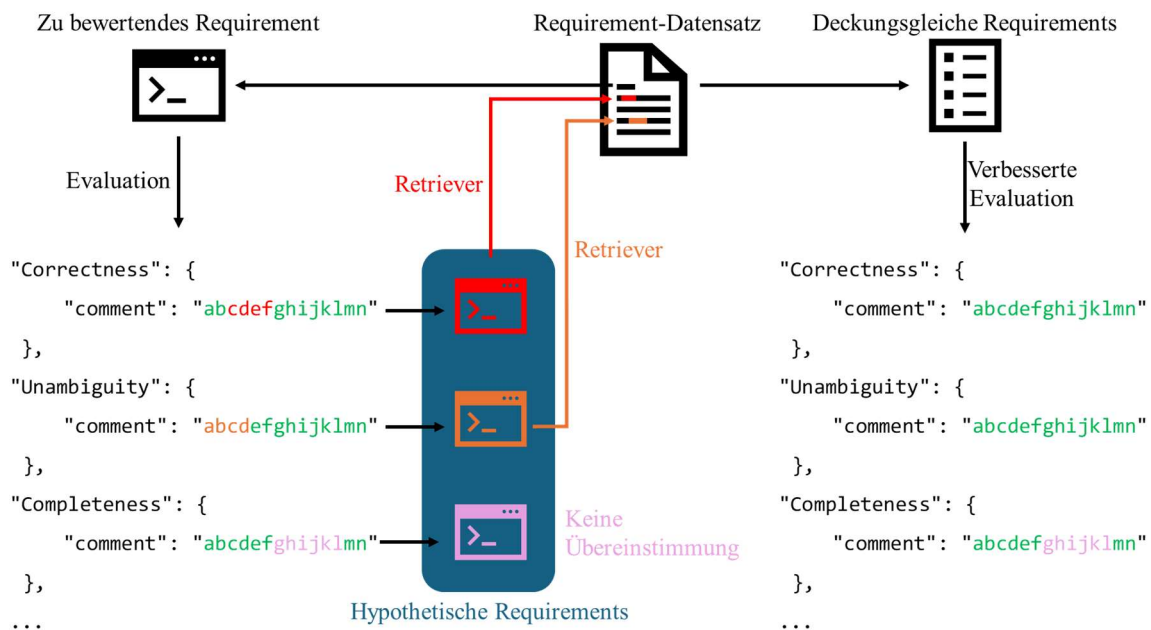


Abbildung 16: HyDE-Ablauf für verbesserte Evaluationen

Wenn das Sprachmodell auf fehlenden Kontext hinweist, wird dieses Requirement als unvollständig eingestuft. Daraufhin könnte das LLM aufgefordert werden, hypothetische Requirements zu generieren, welche diese Lücken abdecken und zusammen mit dem Bewertungsobjekt ein vollständiges Set ergeben würden. Anhand der hypothetischen Beispiele kann ein *Retriever* dann aus dem verfügbaren Datensatz tatsächliche Requirements ermitteln, die im Idealfall deckungsgleich zu den Erstellten sind. Ist dies der Fall, könnte das LLM anschließend erneut eine Evaluation durchführen, in dem Wissen, dass zuvor als fehlend angenommene Aspekte vorhanden sein könnten. Speziell für Metriken, in denen Kontext maßgeblich das Rating bestimmt, könnte dies für eine akkuratere Analyse sorgen. Dies geht einher mit dem Umstand, dass sich etwaiger tatsächlich fehlender Kontext herauskristallisiert. Das liegt daran, dass die Evaluierungen durch HyDE den Datensatz umfassend abbilden könnten.

Eine elementare Schwachstelle wird hierdurch jedoch nach wie vor nicht ausreichend adressiert. Wie bereits mehrfach erwähnt, reicht eine einzelne Meinung für eine faire Beurteilung der Performance nicht aus. Die hieraus entwickelte Idee, ein Multi-Agenten System aufzubauen, könnte signifikante Weiterentwicklungen mit sich bringen. In Kapitel 4 wurde aufgezeigt, dass Modelle verschiedene

Sichtweisen unterschiedlich gut abdecken. Anstatt dies als Schwäche zu verstehen, kann dieser Umstand genutzt werden, mehreren LLMs unterschiedliche Rollen zuzuweisen. Diese können individuell ihre Stärken ausspielen, indem die im Rahmen dieses Projektes gesammelten Best Practices angewandt werden. Zunächst könnten Evaluationen durch mehrere LLMs erstellt werden. Diese nehmen dabei verschiedene Perspektiven ein, etwa als RE-Experte oder als Fachkraft der jeweiligen Domäne. Die Evaluationen werden sowohl iterativ und sukzessiv erstellt, um zunächst ein breites Bild an Bewertungen zu erhalten. Ein anderes Sprachmodell könnte kontextbezogene Schwachstellen per HyDE identifizieren und erneute Evaluationen anfordern. Daraufhin könnten alle bisherigen Modelle sowie ein Benchmark-LLM herangezogen werden, um die resultierenden Evaluationen als Judge zu bewerten. Die sich hieraus ergebenden Tendenzen lassen sich fusionieren, um eine valide Beurteilung des ursprünglichen Requirements zu erhalten. Hierbei könnte man das Prinzip des *Fine-Tunings* aufgreifen, um die Rollenverteilung durch das domänenspezifische Wissen wie beschrieben noch zu bestärken. Optimalerweise werden dabei mehrere Modelle jeweils auf verschiedene spezifische Aspekte angepasst, um eine Betrachtung diverser Aspekte aus unterschiedlichen Blickwinkeln zu forcieren. Eine solche Interaktion verschiedener LLMs kommt einem realen Entwicklungsprozess sehr nahe und könnte dazu beitragen, schlussendlich zu einer ganzheitlicheren Bewertung des Requirements zu gelangen.

## 6 Fazit

Die vorliegende Arbeit befasst sich mit verschiedenen Aspekten des „LLM4RE“-Masterprojektes. Das Ziel ebenjenes Projektes war die Entwicklung eines LLMs zur automatisierten Evaluation von Requirements. Die Bewertungen sollten anhand formaler Qualitätskriterien erfolgen und die Performance mit entsprechenden Mitteln beurteilt werden. Die im Rahmen des Projektes gesammelten Erkenntnisse wurden in dieser Arbeit aufbereitet. Dazu wurde zunächst die Herangehensweise an die Festlegung der Qualitätskriterien dargelegt. Die dabei entstandenen Metriken entstammen der Schnittmenge in der Literatur und stellen ein hilfreiches Werkzeug dar, um die eingangs erwähnten Fehlinterpretationen aufdecken zu können. Mithilfe dieser Metriken wurden verschiedene Prompts entworfen, woraus sich schlussendlich ein sukzessiver und iterativer Ansatz entwickelten. Das jeweilige LLM sollte Kriterien anhand einer zahlenbasierten Skala bewerten. Dabei kamen unterschiedliche Sprachmodelle zum Einsatz, das Größte von ihnen fand dabei als Judge zur Beurteilung der erstellten Evaluationen Anwendung.

Anhand dreier Datensätze, die insgesamt mehr als zweitausend Requirements umfassten, wurden die jeweiligen Ansätze genauestens untersucht. Diese Analyse erfolgte anhand mehrerer Indikatoren, da Stichproben zwar wenig plausible Antworten in vereinzelt Fällen ergaben, jedoch kein aussagekräftiges Gesamtbild ergeben. Daher wurde eine Gegenüberstellung der auftretenden Ratings für den jeweiligen Ansatz erstellt. Darüber hinaus wurden die Evaluationen sowohl mithilfe des LLM-Judge eingeschätzt als auch mit einer Bewertung durch einen menschlichen Experten verglichen. Da beide Herangehensweisen in verschiedenen Aspekten eine bessere Performance aufweisen, lässt sich eine Überlegenheit eines Ansatzes allgemeingültig nicht festhalten. Abhängig vom Indikator zeigen beide Herangehensweisen unterschiedliche qualitative Ausprägungen, was die Kombination beider Ansätze nahelegt.

Die erhaltenen Ergebnisse implizieren eine Reihe möglicher Weiterentwicklungen. Neben der Weiterentwicklung des sukzessiven und des iterativen Ansatzes lassen sich dank des Prompt-Templates auch vollständig neue Ansätze erproben. Dieses ermöglicht eine dynamische Anpassung mithilfe verschiedener Sektionen und wiederkehrender Elemente wie den Metriken. Darüber hinaus lässt sich auch der Funktionsumfang des Projektes flexibel erweitern. Allen voran der Aufbau eines Multi-Agenten-Systems könnte die Aussagekraft der Evaluationen entscheidend voranbringen und verschiedene Ansichten umfassend abbilden.

Im Rahmen des Projektes wurde das große Potenzial LLM-basierter Anwendungen untersucht. Um ebenjenes Potenzial voll auszuschöpfen, ist sehr viel manueller Aufwand nötig, da die erhaltenen Ergebnisse andernfalls lediglich oberflächlich sind. Dazu sei gesagt, dass die rapiden Entwicklungen im Bereich NLP den aktuellen Stand schnell überholen können. Nichtsdestotrotz bietet das Projekt ein außerordentlich wertvolles Framework, in der sich Weiterentwicklungen direkt und einfach implementieren lassen, um zukünftig noch bessere Ergebnisse zu liefern. Die gesammelten Erkenntnisse zeigen jedoch mannigfaltig auf, dass die automatisierte Evaluierung durch ein LLM möglich ist und den Entwicklungsprozess somit entscheidend vereinfachen kann. LLM4RE ist somit ein Werkzeug, das maßgeblich zur Gestaltung des zukünftigen RE-Prozesses beizutragen vermag.

## Literaturverzeichnis

- [1] L. Montgomery, D. Fucci, A. Bouraffa, L. Scholz und W. Maalej, „Empirical research on requirements quality: a systematic mapping study,“ *Requirements Engineering*, Volume 27, p. 183–209, 15 Februar 2022.
- [2] E. Gallego et al., „Requirements Quality Analysis: A Successful Case Study in the Industry,“ in *International Conference on Complex Systems Design & Management*, Paris, 2016.
- [3] D. Zowghi und V. Gervasi, „The Three Cs of Requirements: Consistency, Completeness, and Correctness,“ in *Proceedings of 8th International Workshop on Requirements Engineering: Foundation for Software Quality*, Lund, 2002.
- [4] G. Génova, J. M. Fuentes, J. Llorens, O. Hurtado und V. Moreno, „A framework to measure and improve the quality of textual requirements,“ *Requirements Engineering*, Volume 18, p. 25–41, 2013.
- [5] G. Fanmuy, J. Llorens und A. Fraga, „Requirements Verification in the Industry,“ in *Complex Systems Design & Management - Proceedings of the Second International Conference on Complex Systems Design & Management*, Paris, 2011.
- [6] W. L. Honig, „Requirements Metrics - Definitions of a Working List of Possible Metrics for Requirements Quality,“ Loyola University Chicago, Chicago, 2016.
- [7] Groq, Inc., „GroqCloud Documentation: Supported Models,“ 2025. [Online]. Available: <https://console.groq.com/docs/models>. [Zugriff am 21 Februar 2025].
- [8] Souvik, „Software Requirements Dataset,“ Kaggle, [Online]. Available: <https://www.kaggle.com/datasets/iamsouvik/software-requirements-dataset>. [Zugriff am 22 Februar 2025].
- [9] OpenAI Team, „Best practices for prompt engineering with the OpenAI API,“ OpenAI, 2024. [Online]. Available: <https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api>. [Zugriff am 24 Februar 2025].
- [10] Meta, „How-to guides: Prompting,“ [Online]. Available: <https://www.llama.com/docs/how-to-guides/prompting/>. [Zugriff am 24 Februar 2025].
- [11] LangChain, Inc., „Prompt Templates,“ 2025. [Online]. Available: [https://python.langchain.com/docs/concepts/prompt\\_templates/](https://python.langchain.com/docs/concepts/prompt_templates/). [Zugriff am 25 Februar 2025].
- [12] T. B. Brown et al., „Language Models are Few-Shot Learners,“ in *Advances in Neural Information Processing Systems* 33, 2020.
- [13] J. Wei et al., „Chain-of-thought prompting elicits reasoning in large language models,“ in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, New Orleans, 2022.

- [14] Hugging Face, „Llama 3.3 Model Information,“ Meta, 06 Dezember 2024. [Online]. Available: <https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct>. [Zugriff am 25 Februar 2025].
- [15] K. Wataoka, T. Takahashi und R. Ri, „Self-Preference Bias in LLM-as-a-Judge,“ in *arXiv*, 2024.
- [16] H. K. Chaubey, G. Tripathi, R. Ranjan und S. k. Gopalaiyengar, „Comparative Analysis of RAG, Fine-Tuning, and Prompt Engineering in Chatbot Development,“ in *2024 International Conference on Future Technologies for Smart Society (ICFTSS)*, Kuala Lumpur, 2024.
- [17] V. B. Parthasarathy, A. Zafar, A. Khan und A. Shahid, „The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities,“ CeADAR Connect Group, Ireland, 2024.
- [18] L. Gao, X. Ma, J. Lin und J. Callan, „Precise Zero-Shot Dense Retrieval without Relevance Labels,“ in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Toronto, 2023.