

Semantische Normalisierung für RAG-Systeme

Ein Vorgehensmodell zur LLM-gestützten
Vorverarbeitung strukturierter Wiki-Inhalte

vorgelegt von

Friederike Buchner

EDV.Nr.:xxxxxx

dem Fachbereich VI – Informatik und Medien
der Berliner Hochschule für Technik Berlin
vorgelegte Arbeit
zur Absolvierung des Moduls

Wissenschaftliches Seminar

im Studiengang

Medieninformatik (Online)

Tag der Abgabe 6. Januar 2026



Studiere Zukunft

Version 0.5 α
letzte Änderung: 6. Januar 2026

Gutachter

Prof. Dr. S. Edlich Berliner Hochschule für Technik

Inhaltsverzeichnis

Eigenständigkeitserklärung	2
1 Einleitung	3
1.1 KI-Nutzung	4
2 Theoretische Grundlagen	5
2.1 Large Language Models (LLMs) und Foundation Models	5
2.2 Retrieval-Augmented Generation (RAG)	7
2.3 Evaluation von RAG-Modellen	9
3 Related Works	11
3.1 LLM und strukturierte Daten	11
3.2 LLM-gestützte ETL-Pipelines	11
4 Methode	12
4.1 Zielsetzung und Forschungslogik	12
4.2 Überblick über das Vorgehensmodell	13
4.3 Datenselektion und Datenextraktion	14
4.4 Regelbasierte vs. LLM-basierte Transformation	14
5 LLM-basierte Preprocessing-Pipeline	16
5.1 Prompt-Engineering für die Tabellenanreicherung	17
5.2 Architektur	17
5.2.1 Datenextraktion	17
5.2.2 DOM-Vorverarbeitung	17
5.2.3 Strukturelles Parsing	18
5.2.4 Probabilistische Normalisierung	18
5.2.5 RAG-LLM-Integration	19
5.3 Implementierung	19
5.3.1 Technischer Rahmen und Projektstruktur	19
5.3.2 Interne Datenrepräsentation	19
5.3.3 Implementierung der Vorverarbeitungsschritte (Step 1–3)	19
5.3.4 LLM-gestützte semantische Normalisierung (Step 4)	20
5.3.5 Generierung der normalisierten Markdown-Dokumente	20
5.3.6 Limitationen der Implementierung	20
6 Evaluation	21
6.1 Zielsetzung der Evaluation	21
6.2 Methoden	21
6.3 Ergebnisse	21

7 Diskussion	23
8 Fazit und Ausblick	24
Literatur- und Quellenverzeichnis	24

Entwurf

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Masterarbeit mit dem Titel

*Semantische Normalisierung für RAG-Systeme
Ein Vorgehensmodell zur LLM-gestützten Vorverarbeitung strukturierter Wiki-Inhalte*

selbstständig und ohne unerlaubte Hilfe angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Quellen entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit wurde in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Sofern im Rahmen der Erstellung dieser Arbeit Softwarewerkzeuge (z. B. Textverarbeitungs-, Übersetzungs- oder KI-basierte Assistenzsysteme) eingesetzt wurden, erfolgte dies ausschließlich unterstützend; die inhaltliche Verantwortung für Auswahl, Bewertung und Einordnung der Ergebnisse sowie die finale Ausarbeitung liegt vollständig bei mir. Alle verwendeten Quellen und Hilfsmittel sind im Literatur- bzw. Hilfsmittelverzeichnis angegeben.

Ort, Datum: _____

Unterschrift: _____

Kapitel 1

Einleitung

Ein oft unterschätztes Problem in großen Organisationen ist neben dem Wissenserwerb das Wissensmanagement und insbesondere der Wissenstransfer. Über Jahre und Jahrzehnte entwickeln sich Strukturen und das Wissen darüber wird über verschiedene Stellen verteilt dokumentiert. Je nach Bedarf und oft auch Motivation, kann die Qualität der Dokumentation sehr unterschiedlich ausfallen.

Im Falle von großen IT-Organisationen, insbesondere im Behördenumfeld, sind die Zuständigkeiten oft ebenso verteilt. IT-Infrastruktur und Virtualisierung liegt in einem Referat, IT-Anwendungen und Betrieb im anderen, IT-Service-Helpdesk und Arbeitsplatzausstattung im nächsten. Alle arbeiten Hand in Hand, aber unter unterschiedlicher Leitung, mit selbst definierten Prozessen. Die Schnittstellen sind teilweise undefiniert und beruhen nur auf der Zuarbeit der Kolleg*innen untereinander.

Das Herzstück der internen Dokumentationen ist ein Wiki, in dem mit wenig Struktur die IT-Systeme dokumentiert werden. Darin enthalten sind Servertagebücher, Installationsanleitungen, Vorlagen für Anträge (bspw. für die Bereitstellung virtueller Maschinen). Dieses wird ergänzt durch offizielle Softwaredokumentationen der Hersteller, die meistens im PDF-Format vorliegen. Mit einer Vielzahl von Systemen und Prozessen sind nur ein, höchstens zwei Administrator*innen vertraut, Vertretungen sind nur sehr rudimentär möglich und bei Ausfall der Haupt-Administrator*innen wird der IT-Betrieb nur mit viel Glück und Daumendrücken aufrecht erhalten.

Das IT-Wiki ist eine gute Quelle von Administrations-Interna. Zusammen mit den offiziellen Dokumentation hat es das Potenzial, im Falle von Urlaubsvertretung oder Personalwechsel einen guten Überblick über spezifische IT-Systeme zu geben. Das Problem liegt in der weiten Verstreutheit der Informationen und die schlechte Qualität der Schlagwort-Suche. Auch Veraltung und Irrelevanz der Informationen kann zum Problem werden. Um das Wissensmanagement effizienter zu gestalten, entstand die Idee, die Informationen aus dem Wiki mithilfe eines LLM-Chatbots aufzubereiten.

Mithilfe des RAG-Ansatzes (*Retrieval Augmented Generation*, s. Abschnitt 2.2) können die zugegrundeliegenden Informationen an ein vortrainiertes LLM (*Large Langage Modell*, s. Abschnitt 2.1) übergeben werden. Dies kann diese Dokumente nach Relevanz bezogen auf den ebenfalls mitgelieferten Prompt überprüfen und quellenbasiert Stellung zu der Anfrage beziehen.

Ein Problem dabei gibt es allerdings bei strukturierten und semi-strukturierten Daten wie Tabellen und Übersichten (s. Abschnitt 3.1), da diese dem LLM zu wenig Kontext liefern, um adequate verarbeitet zu werden. Es existiert kaum Forschung zu RAG-Systemen im Kontext von domänenspezifischen, semi-strukturierten IT-Administrationsdaten, wie Wiki-Tabellen oder Kon-

figurationsübersichten. Während RAG gut mit Fließtext funktioniert [Lewis u. a. 2021], ist die automatische Verarbeitung von Tabellen und strukturierten Daten weiterhin ein aktives Forschungsproblem [Sui u. a. 2023]. Dies zeigt sich besonders schwer im IT-Betrieb, da insbesondere IT-Dokumentationen sehr stark von strukturierten Daten geprägt sind, wie beispielsweise Übersichtsdokumentationen zu IP-Adressräumen, Portfreigaben oder Systemumfeldern.

Daraus ergibt sich die folgende Forschungsfrage: Wie kann eine LLM-basierte Preprocessing-Pipeline strukturierte Wiki-Daten so transformieren, dass sie zur Verbesserung eines RAG-Systems für IT-administratives Wissensmanagement beitragen? Die zugrunde liegende Hypothese lautet: Eine automatische LLM-basierte Semantisierung strukturierter Wiki-Daten führt zu qualitativ hochwertigeren Wissensdokumenten und damit zu besseren Antworten eines RAG-Systems.

Um diese Frage zu beantworten und die Hypothese zu beweisen, ist eine Pipeline¹ geplant, die die Daten aus dem Wiki vorverarbeitet, bevor sie an das RAG-LLM übergeben wird. Diese Vorverarbeitung enthält zunächst ein Parsing, da die Daten nach der Extraktion aus dem Wiki zunächst im HTML-Format vorliegen werden. Im zweiten Schritt werden die Dokumente LLM-basiert umgewandelt, sodass aus Tabellen und Übersichten kontextreiche Fließtexte entstehen, die vom RAG-LLM verarbeitet werden können. In Kapitel 5 wird diese Pipeline genauer erläutert. Nach der Vorverarbeitung können die angereicherten Dokumente in eine Wissensbasis transformiert werden, die die Grundlage für den Chatbot bildet. In Kapitel 6 wird das resultierende RAG-Modell evaluiert, was schlussendlich ebenso eine Evaluation der Qualität der Vorverarbeitung darbietet. In Kapitel 7 werden die Ergebnisse und Erkenntnisse diskutiert und in Kapitel 8 ein Fazit gegeben.

Diese Arbeit adressiert damit die Forschungslücke zwischen RAG-Modellen und domänenspezifischem, strukturiertem Wissen aus administrativen Wikis und entwickelt eine prototypische Pipeline zur datengetriebenen Vorbereitung solcher Wissensbestände.

1.1 KI-Nutzung

Als Hilfsmittel für die Erstellung der Arbeit wurde KI benutzt, namentlich OpenAI ChatGPT 5.0. Zunächst wurde für die Themenfindung viel mit der KI gechattet, wobei eine grobe Idee in Richtung „RAG-LLM für Wissensmanagement in IT-Organisation“ von der Verfasserin dieser Arbeit vorgeschlagen wurde. Diese hat sich mit der Zeit immer weiter verfeinert, während von der KI wertvolle Hinweise gegeben wurden. Die KI hat auch bei einem theoretischen Einstieg in das Thema geholfen, indem sie Literaturvorschläge gemacht hat, und das Forschungsfeld erläuterte. Auch beim Lesen und Verstehen der Texte hat die KI geholfen, Sachverhalte zu erläutern, sowie beim Übersetzen von englischen Fachbegriffen ins Deutsche. Später hat die KI dazu beigetragen, die Arbeit zu strukturieren und zu gliedern und auch bei der Herangehensweise konnte sie wertvolle Tipps geben. Die KI hat die ganze Arbeit begleitet, indem sie bei technischen und auch psychologischen Hürden half, einen neuen Ansatz zu finden.

Inhaltlich hat die KI die ersten Dummy-Daten und auch erste Code-Snippets für die Python-Skripte erzeugt. Diese wurden von der Verfasserin der Arbeit kuratiert, bewertet, überarbeitet, verbessert und weiterentwickelt. Die Nutzung der KI diente immer dem Einstieg, dem Befähigen zur Weiterentwicklung. Die daraus entstandenen Gedanken, Texte und Skripte sind die der Verfasserin. An keiner Stelle wurden generierte Texte übernommen und in die Arbeit eingefügt.

¹<https://github.com/free-da/struct2prose>

Kapitel 2

Theoretische Grundlagen

In dieser Arbeit werden Grundlagen zu verschiedenen Phasen des AI-Lifecycle unabdingbar sein. Zunächst soll der Begriff der Large Language Models allgemein und der *Foundation Models* im Speziellen geklärt werden. Danach wird der RAG-Ansatz stärker beleuchtet. Zum Schluss wird sich mit der Evaluation von RAG-Modellen beschäftigt.

2.1 Large Language Models (LLMs) und Foundation Models

Foundation Models nach [Bommasani u. a. 2022] sind Modelle, die üblicherweise selbstüberwacht (*self-supervised*) auf umfassenden Daten trainiert sind und auf eine große Anzahl an nachgelagerten Aufgaben (*downstream tasks*) angepasst werden können. Aktuelle Beispiele beinhalten BERT, GPT-3 und CLIP. Von einem technologischen Standpunkt her sind *Foundation Models* nicht neu, da sie auf tiefen neuronalen Netzwerken und selbstüberwachtem Lernen basieren, was beides bereits seit Jahrzehnten existiert. Beachtenswert sind *Foundation Models* heutzutage deshalb, weil sich ihre schiere Größe in den letzten Jahren vervielfacht hat und sie somit alle Vorstellungen dessen, was man vor wenigen Jahren für möglich hielt, sprengen. GPT-3 beispielsweise hat 175 Milliarden Parameter und kann durch natürlichsprachige Prompts so angepasst werden, dass es eine passable Leistung in vielfältigen Aufgaben zeigt, obwohl es nicht explizit für diese Aufgaben trainiert wurde [Bommasani u. a. 2022, S. 3].

Nach technischen Gesichtspunkten funktionieren *Foundation Models* durch Transferlernen (*transfer learning*) und Skalierung (*scale*). Transferlernen bedeutet, das „Wissen“, was in einer Anwendung (bspw. Bilderkennung) erlernt wurde, auf eine andere Aufgabe (bspw. das Erkennen von Aktivitäten in Videos) zu übertragen. Innerhalb des *Deep Learning* ist das Vortraining (*pretraining*) der vorherrschende Ansatz für Transferlernen: ein Model trainiert eine Ersatzaufgabe und wird dann via *fine-tuning* für die eigentlich relevante nachgelagerte Aufgabe angepasst. Zusammen mit der Skalierung von *Foundation Models* entsteht nun eine sehr mächtige Kombination. Hierfür werden drei entscheidende Punkte wichtig: die Verbesserungen in Computer Hardware, die Entwicklung der Transformer Architektur und die Verfügbarkeit von viel mehr Trainingsdaten [Bommasani u. a. 2022, S. 4].

Letzteres kann nicht deutlich genug hervorgehoben werden: die Wichtigkeit des Vorhandenseins von Daten und die Fähigkeit, sich diese zunutze zu machen. Transferlernen durch annotierte Datensätze war jahrelang die gängige Praxis. Die hohen Kosten von Annotationen, insbesondere von hochqualitativen, händisch erzeugten Annotationen, haben jedoch eine natürliche Grenze in der Skalierung von Trainingsdaten dargestellt. Im selbstüberwachten Lernen ergibt sich die Ersatzaufgabe für das Vortraining automatisch aus unannotierten Daten. Selbstüberwachte Aufgaben sind nicht nur besser skalierbar und ausschließlich abhängig von unannotierten Daten,

sondern sie zwingen das Modell dazu, Teile der Eingabe vorherzusehen, was sie reichhaltiger und potentiell nützlicher machen als Modelle, die in einem begrenzteren Sprachraum trainiert sind [Bommasani u. a. 2022, S. 4].

Selbstüberwachtes Lernen war zunächst eine Unterdisziplin von NLP, die sich parallel zu anderen Entwicklungen ergab. Ab der Einführung des BERT-Modells [Devlin u. a. o. D.] 2019 wurde selbstüberwachtes Lernen zur Norm in NLP. Die Akzeptanz, dass ein einzelnes Modell derart nützlich über eine weite Bandbreite von Aufgaben sein könnte, markiert den Beginn der Ära von *Foundation Models* [Bommasani u. a. 2022, S. 5].

Homogenisierung ist ein Ergebnis der Konsolidierung von Systemen für Maschinelles Lernen über eine weite Palette an Anwendungen. Es ermöglicht das Erledigen vieler Aufgaben aber bildet auch *single points of failure* [Bommasani u. a. 2022, S. 3]. *Foundation Models* haben ein nie zuvor gesehenes Maß an Homogenisierung herbeigeführt, da fast alle *state-of-the-art* NLP-Modelle aus einem von wenigen Modellen wie BERT, GPT o.ä. hervorgehen. Dadurch können alle NLP-Anwendungen direkt von Verbesserungen in *Foundation Models* profitieren. Es birgt aber auch die Gefahr, dass alle KI-Systeme dieselben problematischen Verzerrungen (*biases*) einiger weniger *Foundation Models* erben.

Ein zweites Charakteristikum von *Foundation Models* ist die Emergenz. Das bedeutet, dass das Verhalten eines Systems implizit induziert ist, anstatt explizit konstruiert. Das zeigt sich, indem ein System (zur Überraschung seiner Schaffer*innen) Verhaltensweisen oder Fähigkeiten aufweist, die nicht von außen definiert wurden, sondern die sich eher als Nebenprodukt zum hauptsächlichen Einsatzzweck ergeben. Es ist gleichzeitig die Quelle wissenschaftlicher Erregung sowie Besorgnis über unerwartete Konsequenzen [Bommasani u. a. 2022, S. 3]. Emergenz wird umso bedeutender, je größer das Modell skaliert ist. Während GPT-2 mit 1,5 Milliarden Parametern trainiert wurde, wurde GPT-3 mit 175 Milliarden Parametern trainiert, was kontextsensitives Lernen ermöglichte, in welchem das Sprachmodell einfach auf eine nachgelagerte Aufgabe angepasst wird, indem es mit einer natürlichsprachlichen Beschreibung einer Aufgabe (*prompt*) versorgt wird. Dies war eine emergente Fähigkeit, die weder speziell trainiert noch überhaupt antizipiert wurde [Bommasani u. a. 2022, S. 5].

Homogenisierung und Emergenz interagieren miteinander auf potenziell beunruhigende Art und Weise. Homogenisierung kann potenziell enorme Vorteile für viele Domänen bringen, in denen aufgabenspezifische Daten knapp sind. Auf der anderen Seite kann jeder Fehler im Modell blind von allen angepassten Modellen geerbt werden. Da die Macht von *Foundation Models* viel mehr in ihren emergenten Qualitäten als in ihrer expliziten Konstruktion steckt, sind die existierenden Modelle schwer zu verstehen und haben unvorhergesehene Fehlverhalten. Da Emergenz substantielle Unsicherheiten über die Fähigkeiten und Fehler von *Foundation Models* erzeugt, ist mit der umfassenden Homogenisierung über diese Modelle hinweg ein erhebliches Risiko verbunden. Dieses Risiko zu mitigieren ist eine der zentralen Herausforderungen in der weiteren Entwicklung von *Foundation Models* aus einer ethischen sowie aus einer KI-Sicherheitsperspektive [Bommasani u. a. 2022, S. 6].

Zur Bezeichnung von *Foundation Models* und zur Abgrenzung von Sprachmodellen allgemein kann man sagen, dass der Geltungsbereich von *Foundation Models* schlüsselweg weit über die Grenzen von Sprache hinaus reicht. Es wurden auch andere Bezeichnungen wie beispielsweise *General-Purpose Model* oder *Multi-Purpose Model* in Betracht gezogen, da sie den Aspekt, dass diese Modelle vielfältige nachgelagerte Aufgaben bewältigen können, besser einfangen. Sie täuschen aber darüber hinweg, dass *Foundation Models* unfertig sind und weiter angepasst werden müssen. Weitere Namensvorschläge wie *Task-agnostic Model* würden zwar die Art des Trainings wiederspiegeln, aber nicht die Relevanz für weitere nachgelagerte Aufgaben. Es wurde sich also für den Begriff *Foundation* (engl. für Basis, Grundlage) entschieden, da ein *Foundation Model*

an sich unfertig ist, aber als allgemeine Grundlage dient, aus der vielfältige aufgabenspezifische Modelle durch Anpassung entstehen können. Gleichzeitig weist der Begriff *Foundation* auch auf die Wichtigkeit von architektonischer Stabilität, funktionaler Sicherheit (engl. *safety*) sowie dem Schutz vor Angriffen (engl. *security*) hin. So wie schlecht konstruierte Fundamente fast schon eine Garantie für strukturelles Versagen sind, sind gut ausgeführte Fundamente verlässliche Grundlagen für zukünftige Anwendungen. Zu betonen ist weiterhin, dass momentan die Natur oder Qualität dieser Art von Fundamenten, die *Foundation Models* bieten, nicht in Gänze verstanden wird und dass nicht einwandfrei beurteilt werden kann, ob diese Fundamente verlässlich sind, oder nicht.

2.2 Retrieval-Augmented Generation (RAG)

LLMs haben neben ihrem bemerkenswerten Erfolg auch signifikante Grenzen, speziell in domänen spezifischen oder wissensintensiven Aufgaben. Eins der größten Probleme ist das „Haluzinieren“ [Zhang u. a. 2025] beim Verarbeiten von Anfragen, die Informationen betreffen, die nicht in den Trainingsdaten enthalten waren. Um diese Herausforderungen zu bewältigen, werden LLMs per *Retrieval-Augmented-Generation* (RAG) erweitert, indem relevante Inhalte mithilfe semantischer Ähnlichkeitsberechnungen aus externen Wissensbasen abgerufen werden. Indem externes Wissen referenziert wird, reduziert RAG effektiv das Problem, faktisch inkorrekte Inhalte zu generieren. Die Integration in LLMs ist mittlerweile weit verbreitet, was RAG zu einer Schlüsseltechnologie im Voranbringen von Chatbots und der Eignung von LLMs für Anwendungen in der realen Welt gemacht hat [Gao u. a. 2024].

Die Erforschung von RAG traf mit der Entwicklung der Transformer Architektur zeitlich aufeinander. Zu Beginn lag der Fokus darauf, Sprachmodelle durch zusätzliche Wissensquellen zu verbessern, insbesondere durch die Integration externer Informationen in vortrainierte Modelle (*Pretrained Models*, PTMs). Mit dem Aufkommen von ChatGPT gab es einen Wendepunkt: Große Sprachmodelle (LLMs) zeigten nun ihre Fähigkeit zum *In-Context Learning* (ICL), also dazu, zur Laufzeit neues Wissen aus Eingabekontexten aufzunehmen und zu verwenden. Das führte die RAG-Forschung dahin, bessere Informationen für LLMs bereitzustellen, um komplexere und wissensintensive Aufgaben in der Inferenz-Phase (also während der Antwortgenerierung) beantworten zu können. Mit voranschreitender Forschung war RAG dann nicht mehr auf die Inferenz-Phase beschränkt, sondern fügte sich immer mehr in LLM *fine-tuning*-Techniken, also das gezielte Nachtrainieren der Modelle mit domänen spezifischen oder aufgabenspezifischen Daten, ein [Gao u. a. 2024].

In Abbildung 2.1 ist die grundsätzliche Funktionsweise von *Naive RAG*, angewendet auf die Aufgabe der Fragenbeantwortung, dargestellt. Die in der Abbildung dargestellte Frage bezieht sich auf aktuelle Ereignisse, das heißt, dass die zur Beantwortung benötigten Informationen nicht in den Trainingsdaten enthalten gewesen sein können. RAG überbrückt diese Lücke, indem das benötigte Wissen aus externen Datenbanken abgerufen wird und zusammen mit der initialen Frage einen umfassenden Prompt ergibt, der das LLM befähigt, eine wohlinformierte Antwort zu generieren. Im Wesentlichen besteht *Naive RAG* aus drei Schritten:

1. *Indexing*: Dokumente werden in Abschnitte (engl. *chunks*) unterteilt, in Vektoren kodiert und in einer Vektordatenbank gespeichert
2. *Retrieval*: die relevantesten Top k Abschnitte werden abgerufen, basierend auf semantischer Ähnlichkeit
3. *Generation*: die ursprüngliche Frage wird gemeinsam mit den abgerufenen Abschnitten an ein LLM übergeben, um eine Antwort zu generieren

Obwohl dieses sogenannte *Retrieve-Read*-Framework des *Naive RAG* kosteneffektiv ist und die

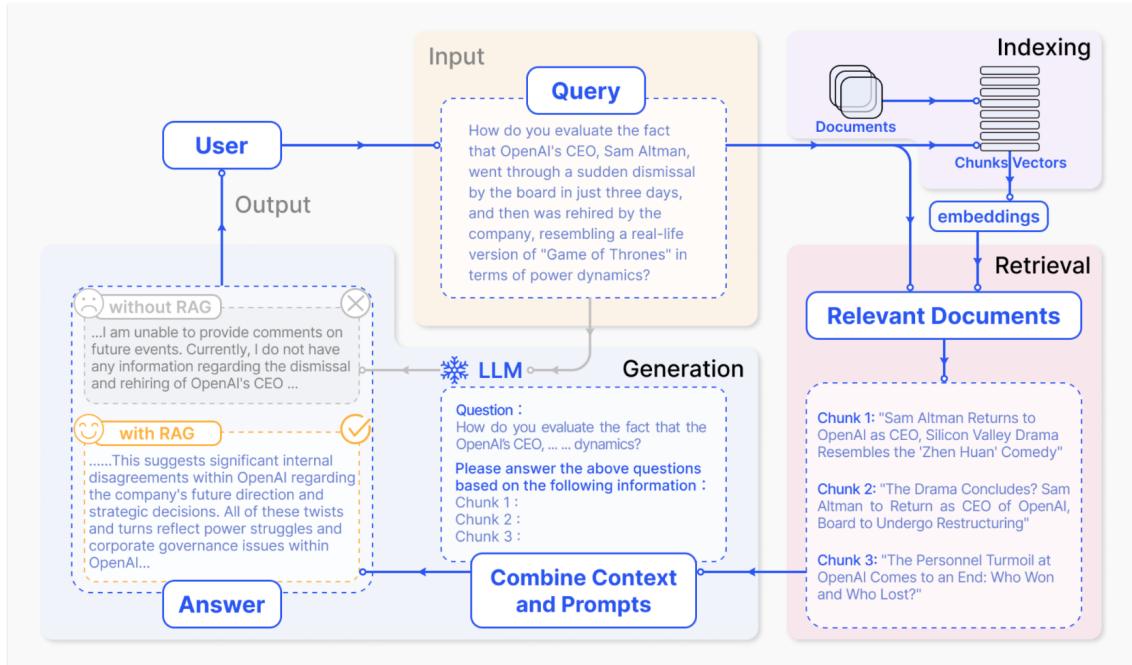


Abbildung 2.1: Überblick über die Funktionsweise von RAG nach [Gao u. a. 2024, S. 3]

Performanz des nativen LLM weit übertrifft, hat es dennoch mehrere Schwächen die durch die Entwicklung von *Advanced RAG* sowie *Modular RAG* adressiert wurden [Gao u. a. 2024].

Beim *Advanced RAG* wird ein Fokus auf die Steigerung der Qualität der Abrufe (engl. *retrievals*) gesetzt. Dies passiert, indem Strategien für die Vor- und Nachbearbeitung von Abrufen angewendet werden. Um die Indizierung zu verbessern, werden feinere Segmentierungsgrade sowie Metadaten zusätzlich zu weiteren Optimierungsmethoden angewendet, um die originale Frage klarer zu machen und für die Abfrage aufzubereiten. In der Nachbearbeitung der Abfrage werden die abgefragten Informationen aufbereitet, um die relevantesten Informationsblöcke hervorzuheben. Die direkte Übergabe aller relevanten Dokumente an das LLM könnte zu einer Informationsüberlastung führen. Um dies zu vermeiden, konzentriert sich die Nachbereitung der Abfrage auf die Auswahl essentieller Informationen und die Kürzung des zu verarbeitenden Kontextes. Der Gesamtprozess ähnelt jedoch weiterhin dem des *Naive RAG* und folgt ebenso einer linearen Struktur [Gao u. a. 2024].

Bei der modularen RAG-Architektur wird diese lineare Struktur hinter sich gelassen und eine größere Anpassbarkeit und Vielseitigkeit geboten. Das modulare RAG-Framework führt spezialisierte Komponenten ein, um die Abfrage- und Verarbeitungskapazitäten zu erhöhen. Ein Such-Modul ermöglicht direkte Suchen über diverse Datenquellen hinweg. Ein Speicher-Modul nutzt den Speicher des LLM, um Abfragen zu steuern. Ein Vorhersage-Modul zielt darauf ab, Redundanz und Rauschen zu verringern, indem es Kontext direkt durch das LLM generiert, und damit Relevanz und Akkurateit sicherstellt. Diese und weitere Module steigern die Qualität und Relevanz der abgefragten Informationen und ermöglichen so das Ausführen einer Vielfalt von Aufgaben mit erhöhter Präzision und Flexibilität [Gao u. a. 2024].

Modulares RAG geht über die bisherige lineare Struktur von *Naive* sowie *Advanced RAG* hinaus und erlaubt durch seinen modularen Charakter eine bemerkenswerte Anpassbarkeit, indem Module ausgetauscht und rekonfiguriert werden können. Der herkömmliche *Read-Retrieve-Ansatz* wird durch Erfindungen wie *Rewrite-Retrieve-Read*, *Generate-Read*, *Recite-Read* oder anderen erweitert und bietet viele Möglichkeiten, die Fähigkeit des jeweiligen Models, spezifische Aufgaben zu behandeln, zu verbessern. Die flexible Orchestrierung von modularem RAG zeigt mit weite-

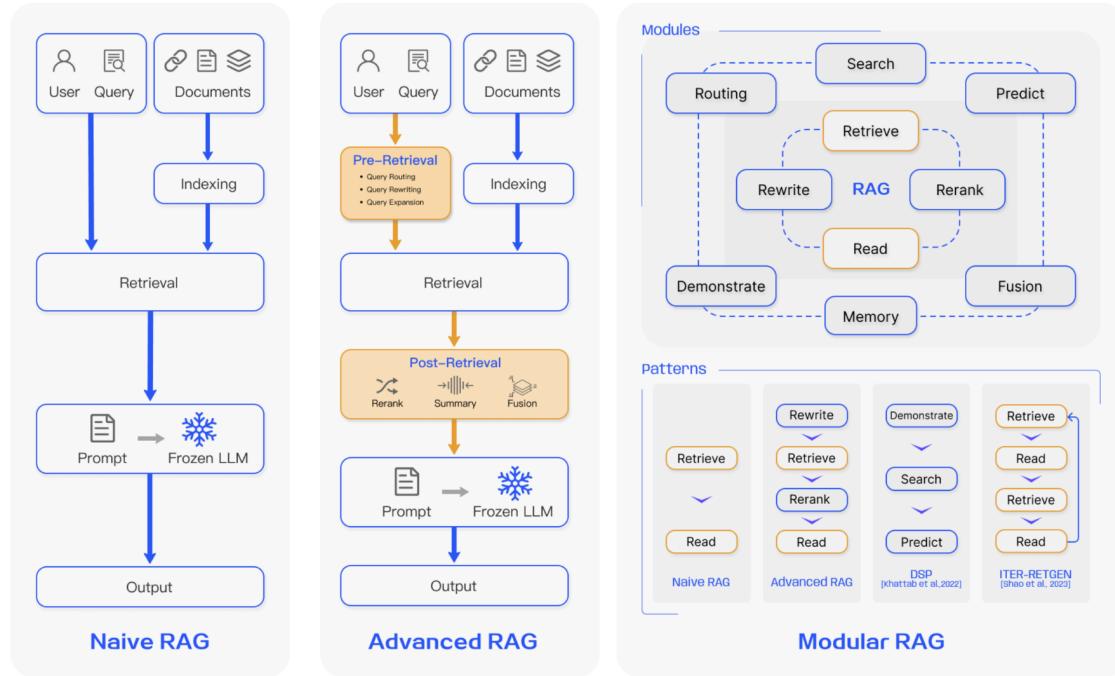


Abbildung 2.2: Überblick über *Naive RAG*, *Advanced RAG* und *Modular RAG* [Gao u. a. 2024, S. 3]

ren Abfragetechniken wie „FLARE“ [Jiang u. a. 2023] und „Self-RAG“ [Asai u. a. 2023], dass dieser Ansatz den starren RAG-Abfrage-Prozess übertrifft, indem die Notwendigkeit einer Abfrage je nach verschiedenem Scenario bewertet wird. Ein weiterer Vorteil der flexiblen Architektur ist, dass das RAG-System leichter mit anderen Technologien (wie etwa *fine-tuning* oder *reinforcement learning*) kombiniert werden kann [Gao u. a. 2024].

2.3 Evaluation von RAG-Modellen

Eine gelungene, durchdachte und gewissenhafte Evaluation ist von großer Bedeutung für den Erfolg von LLMs. Sie kann ein gutes Leitbild für die Verbesserung von Mensch-LLM-Interaktion bieten, was sich positiv auf zukünftige Entwicklungen von Interaktionsdesign und -implementierung auswirkt. Außerdem unterstreicht die weitreichende Anwendbarkeit von LLMs die Wichtigkeit von Sicherheit und Zuverlässigkeit, vor allem in sicherheitssensiblen Sektoren wie Finanzinstitutionen oder staatlichen Behörden. Evaluationsmethoden müssen konstant überprüft und angepasst werden, da LLMs mit steigender Größe und mehr emergenten Fähigkeiten potenzielle neue Risiken entwickeln, die existierende Evaluationsprotokolle nicht abdecken [Chang u. a. 2023].

Im Folgenden soll auf die Evaluation von RAG-Modellen eingegangen werden. Hierfür liefern [Knollmeyer u. a. 2024] einen guten Überblick über die Möglichkeiten vom Benchmarking. Die Resultate ihrer Recherche sollen hier kurz zusammengefasst werden. Sie teilen sich auf in folgende Bereiche: Die Vorhersagen-Evaluation (engl. *predictive evaluation*) und die Datensatz-Evaluation (engl. *dataset evaluation*). Wir wollen uns hier zunächst auf die Evaluation der Vorhersagen konzentrieren.

In der vorgeschlagenen Evaluations-Pipeline werden verschiedene Dimensionen evaluiert und dabei ein Fokus auf die *Retrieval*- und Generierungs-Stufen eines typischen RAG-Systems gelegt. Wie in der Abbildung 2.3 zu sehen, startet der Evaluationsprozess mit dem *Retrieval*-Schritt, in dem die Kontextrelevanz als kritische Evaluationsdimension zu betrachten ist. Danach beginnt die Generierungs-Evaluation, in der die Evaluationsdimensionen Antwortrelevanz, Korrektheit,

Faktentreue (engl. *faithfulness*) und Qualität der Quellenangaben (engl. *citation quality*) überprüft werden [Knollmeyer u. a. 2024, S. 142].

Für die Auswahl von Metriken und die Kalkulation der jeweiligen Evaluationsdimension ist der ausgewählte Evaluatör zuständig. Hier werden durch [Knollmeyer u. a. 2024, s.142] drei verschiedene aufgeführt: Lexikalische Übereinstimmung (engl. *lexical matching*), Semantische Ähnlichkeit (engl. *semantic similarity*), und LLM als Richter (engl. *LLM as a judge*) [Knollmeyer u. a. 2024, S. 142]. Der Unterschied liegt darin, ob der Evaluationsfokus auf exakter wörtlicher Übereinstimmung, konzeptueller Ähnlichkeit oder einer nuancierten Abwägung des Kontexts durch ein LLM liegt.

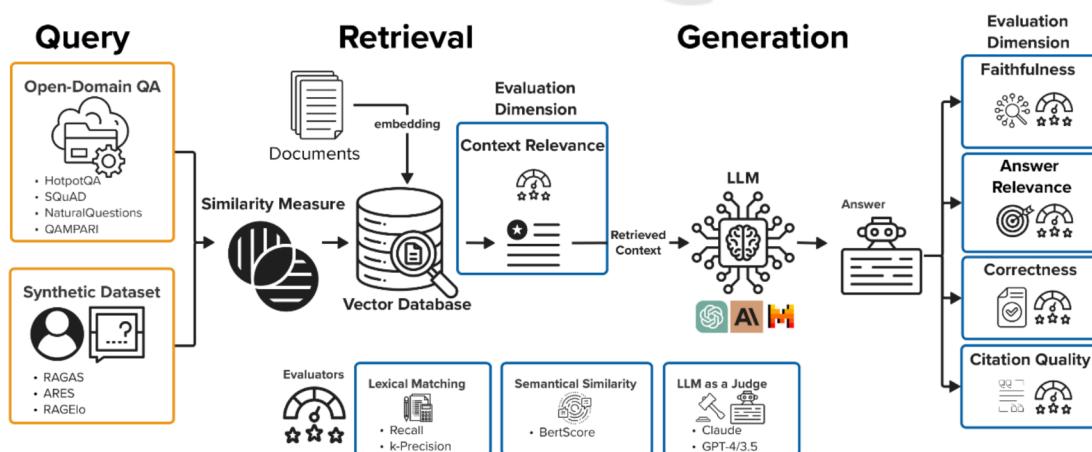


Abbildung 2.3: Evaluationsprozess mit Evaluationsdimensionen nach [Knollmeyer u. a. 2024, S. 142].

Durch [Knollmeyer u. a. 2024, S. 142ff.] werden die Evaluationsdimensionen genauer dargelegt . Hier soll ein kurzer Überblick erfolgen, um eine Grundlage für das spätere Evaluationsdesign zu schaffen. Details zur Implementierung werden dann in Kapitel xxx dargelegt.

Kontextrelevanz Kontextrelevanz ist die essentielle Evaluationsdimension aus der *Retrieval*-Phase im RAG-Modell. Es bewertet, inwiefern der abgerufene Kontext nur wichtigen Informationen enthält, die nötig sind, um die Abfrage zu beantworten. Durch die Minimierung irrelevanten Kontexts werden Rechnerressourcen gespart und die Effizienz gesteigert. Zudem ist die Konzentration auf das Wesentliche auch deshalb wichtig, weil LLMs derzeit noch viel schlechter dazu in der Lage sind, aus größeren Sinnzusammenhängen relevante Informationen abzurufen als aus kleineren [Knollmeyer u. a. 2024, S.142].

Faktentreue Die Faktentreue evaluiert (wie die folgenden Evaluationsdimensionen) die Generierungsphase eines LLM. Hier ist die Frage, wie sehr die Antwort des LLMs auf eine Eingabe in dem abgerufenen Kontext eingebettet ist. Das Ziel wäre, dass in einem RAG-Modell alle generierten Fakten direkt aus dem abgerufenen Kontext ableitbar ist. Dies ist ein wichtiger Evaluationsschritt, um Halluzinationen zu identifizieren, indem eine Diskrepanz aus generierter Antwort und zugrundeliegenden Fakten festgestellt wird [Knollmeyer u. a. 2024, S. 143].

Antwortrelevanz Ebenfalls in der Generierungsphase verortet ist die Antwortrelevanz, die bewertet, ob das LLM die Eingabe direkt beantwortet. Hier werden unvollständige oder redundante Antworten bestraft, auch wenn sie inhaltlich korrekt wären.

Korrektheit Bei der Korrektheit wird innerhalb der Generierungsphase evaluiert, wie *tbc...*

Kapitel 3

Related Works

Hier steht ein kluger Einleitungssatz, damit ich nicht direkt mit den Unterkapiteln beginne.

3.1 LLM und strukturierte Daten

Verschiedene Arbeiten beschäftigen sich mit der Frage, wie LLM strukturierte und semi-strukturierte Daten erfassen und „verstehen“ können. In [Sui u. a. 2023] wird hierfür der Ansatz *self-augmented prompting* verwendet. In Abbildung 3.1 ist die Funktionsweise dieses Ansatzes dargestellt. Eine Anfrage, die auf strukturierten Daten, wie beispielsweise einer Tabelle, basiert, durchläuft das LLM in zwei Phasen. Bei der ersten werden wichtige Informationen und Wertebereiche der Tabelle durch das LLM identifiziert und beim zweiten Durchlauf werden natürlichsprachige Informationen für die relevanten Bereiche augmentiert.

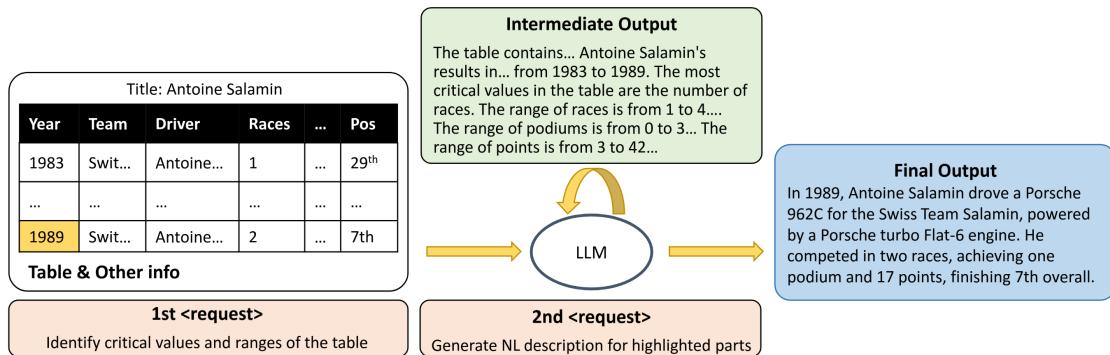


Abbildung 3.1: Illustration des *self-augmented prompting* nach [Sui u. a. 2023]

Im Unterschied zu dieser Arbeit ist in [Sui u. a. 2023] die Serialisierung in dem Abfrageschritt ans LLM enthalten. Hier soll die Datenvorverarbeitung als abgeschlossener Schritt passieren, bevor die Daten in die Wissensdatenbank des RAG-Modells einfließen. Der Vorteil liegt darin, dass die Datenvorverarbeitung so model-agnostisch funktioniert und auf beliebige Anwendungen anwendbar ist. Selbst wenn die Daten durch ein geschlossenes System weiterverarbeitet werden können, in dem kein Zugriff auf den LLM-Lifecycle besteht, können die Daten derart vorverarbeitet werden und die Anwendungen von der Pipeline profitieren.

TODO: [Wolff und Hulsebos 2025]

3.2 LLM-gestützte ETL-Pipelines

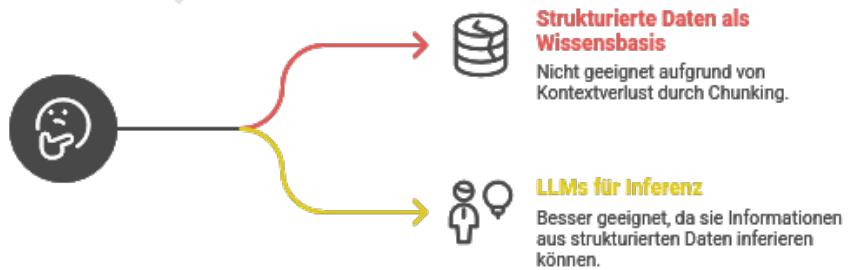
Kapitel 4

Methoden

Das wissenschaftliche Vorgehensmodell soll in diesem folgenden Kapitel erläutert werden. Zunächst wird im Unterkapitel 4.1 die Zielsetzung und Forschungslogik dargelegt. Hier wird die Forschungsfrage konkretisiert und das Forschungsdesign erläutert, außerdem kurz umrissen, was diese Arbeit nicht enthalten wird. Im Unterkapitel 4.2 wird das Vorgehen zur Beantwortung der Forschungsfrage erläutert. Im Unterkapitel 4.3 werden Entscheidungen zur Auswahl der Versuchsdaten dargelegt und im Unterkapitel 4.4 die angewandten Transformationsschritte erläutert.

4.1 Zielsetzung und Forschungslogik

Die zentrale Forschungsfrage dieser Arbeit lautet: „Inwiefern trägt eine LLM-gestützte semantische Normalisierung strukturierter Wiki-Inhalte zur verbesserten Nutzbarkeit dieser Inhalte in RAG-Systemen bei?“. Sie zielt darauf ab, die bekannte Schwäche von RAG-Modellen in der Verarbeitung strukturierter und semistrukturierter Daten in der Wissensbasis auszugleichen und somit Informationen, die in vielfältigen Formaten vorliegen, für ein RAG-System nutzbar zu machen.



Made with Napkin

Abbildung 4.1: Unterschied in der Fähigkeit zur Verarbeitung strukturierter Daten

Strukturierte Daten finden wir in der Nutzung interner IT-Dokumentationen zuhauf: Dokumentationen über Benutzerpflege, Inventarisierungen und Konfigurationen von IT-Systemen im Allgemeinen. Da keine formelle Redaktion vorhanden ist, schreiben alle Administrator*innen ih-

re Dokumentationen selbst. Dies führt zu einer strukturellen Heterogenität der Daten in der Wissensrepräsentation. Zusammen mit der Schwäche von RAG-Systemen bei der Verarbeitung strukturierter Daten ergibt sich ein Problem, das nicht trivial zu lösen ist.

An dieser Stelle ist eine Klärung wichtig: obwohl RAG-Systeme wie dargestellt nicht ausreichend in der Lage sind, mit strukturierten Daten als Wissenbasis umzugehen, können LLMs an sich sehr wohl strukturierte Daten verarbeiten und daraus Informationen inferieren. Dieser Unterschied wird in Abbildung 4.1 dargestellt und soll in der folgenden Arbeit zunutze gemacht werden.

Das Ziel dieser Arbeit ist, Erkenntnis darüber zu schaffen, inwiefern die Vorverarbeitung der Daten eine erhebliche Verbesserung in ihrer Nutzbarkeit durch ein RAG-Modell erreichen kann. Der Weg zu diesem Ziel ist eine design-orientierte Untersuchung, in der explorativ und artefaktbasiert eine solche Vorverarbeitung durchgeführt, und das Ergebnis mit einem vergleichbaren System ohne diese Vorverarbeitung unter Beibehaltung identischer Parameter verglichen wird.

Das zentrale Artefakt ist eine mehrstufige LLM-gestützte Vorverarbeitungs- und Normalisierungspipeline, die durch mehrere Transformationsschritte reproduzierbar und überprüfbar das Format der zugrundeliegenden Daten an die Verarbeitungsweise eines RAG-Systems anpasst. Die Überbrückung der Lücke zwischen heterogenen strukturierten Rohdaten und der Unfähigkeit zur Verarbeitung strukturierter Daten eines RAG-Systems durch die semantische Normalisierung bietet die Grundlage der Nutzbarmachung von Wiki-Daten für ein RAG-LLM mit domänenspezifischen, internen IT-Daten.

Hier werden gewisse Vereinfachungen bewusst gemacht. Für eine komplette Pipeline wären Schnittstellendefinitionen für den Ein- und Ausgang der Daten nötig, die hier händisch gemacht wurden. Diese Entscheidung liegt darin begründet, dass diese Arbeit eine Prüfung der Transformation sein soll. Erst im Falle einer positiven Bewertung dieses Ansatzes ist es sinnvoll, sich auf konkrete APIs für Ein- und Ausgang der Pipeline oder andere technische Werkzeuge festzulegen.

Des Weiteren finden kein Benchmarking in der Evaluation und auch keine Nutzerstudien statt. Es werden auch keine LLMs oder RAG-Systeme miteinander verglichen, da der Fokus auf der Vorverarbeitung der Daten liegt und nicht auf der Bewertung eines LLMs. Das fertige RAG-LLM oder gar ein Chatbot als Anwendungsbereich für das Wissensmanagement könnten das Ergebnis weiterer Arbeiten sein, die auf dieser Arbeit aufbauen. Diese Arbeit ist aber explizit eine Untersuchung über die Tauglichkeit dieses konkreten Vorverarbeitungs-Ansatzes.

4.2 Überblick über das Vorgehensmodell

Die vorgeschlagene Lösung für das Problem der Verarbeitung von strukturierten Daten bei RAG-Systemen ist ein Vorgehensmodell, was strukturierte Wiki-Inhalte in eine RAG-geeignete Wissensrepräsentation überführt. Operationalisiert wird dieses Vorgehensmodell mit einer Datenvorverarbeitungs- und Normalisierungspipeline, die in drei Schritten die Wiki-Inhalte transformiert. Der dritte Schritt ist das Kernstück dieser Pipeline: die LLM-gestützte semantische Normalisierung strukturierter Inhalte in Fließtext.

Die drei Transformationsschritte sind die folgenden:

1. **DOM-Vorverarbeitung** - Hier findet die strukturelle Bereinigung des Dokuments statt, in der überflüssige Informationen über Präsentation und Navigation verworfen werden.
2. **Strukturelles Parsing** - Dies ist der Schritt, in dem die bereinigten Informationen in eine interne Objektstruktur überführt werden.

3. **Semantische Normalisierung** - An dieser Stelle wird die bereinigte Objektrepräsentation an ein LLM übergeben mit dem Ziel, diese in einen kontextreichen Fließtext umzuformulieren.

Die normalisierten Daten dienen als Grundlage zur Evaluation.

4.3 Datenselektion und Datenextraktion

Die Rohdaten, die transformiert werden sollen, entstammen einem Wiki zur internen Dokumentation einer größeren IT-Organisation. Sie sind textlastig, aber durch den vermehrten Einsatz von Tabellen, Listen und Übersichten sehr heterogen, mit semi-strukturierten bis strukturierten Fragmenten.

Es wurde sich auf die Auswahl von fünf geeigneten Wiki-Seiten beschränkt. Diese wurden ausgewählt, da sie von der Autorin dieser Arbeit selbst verfasst wurden, und somit Fragen der Urheberschaft zu eindeutig geklärt sind. Außerdem sind sie dadurch ähnlicher in der inhaltlichen Tiefe, als wenn sie von mehreren Autor*innen verfasst worden wären. Darüber hinaus finden sich in den Seiten verschiedene semistrukturierte Repräsentationen von Informationen, wie beispielsweise Konfigurationstabellen oder Datenbankübersichten. Dies ist interessant, da hier leicht zu beobachten ist, wie die geplante Transformation die Tauglichkeit der Daten für die Verwendung im RAG-LLM verändert. Die Auswahl der Eingangsdaten wurde bewusst auf diese fünf Seiten eingegrenzt, da der Fokus dieser Arbeit auf der grundsätzlichen Erprobung des Vorgehensmodells liegt, und nicht auf der Anwendbarkeit oder Skalierung.

Die Extraktion der Seiten manuell durchgeführt, sodass reine Textdateien lokal gespeichert wurden. Es wurde auf die Sonderbehandlung von Bildern und Anhängen verzichtet, beziehungsweise wurde bei der Auswahl der Beispelseiten gezielt darauf geachtet, dass keine wichtigen Bilder enthalten sind. Die bewusste Entscheidung der manuellen Extraktion liegt darin begründet, dass die Datentransformation im Fokus dieser Arbeit liegt. Die manuelle Extraktion ist einfach und zweckdienlich und bietet eine Entkopplung dieses Schritts von der Transformation.

Nach der Extraktion fand eine Anonymisierung statt, bei der Zeichenketten, die auf den Unternehmenskontext hinweisen könnten, durch arbiträre Zeichenketten ersetzt wurden. Dazu gehörten die Web-Domäne, der Unternehmensname, der Nutzernamen der Bearbeiterin und weitere organisationsinterne Namen und Bezeichnungen. Dieser Schritt war nötig, um keine Bedenken bezüglich der Verarbeitung personenbezogener oder anderer sensibler Daten auf privater Hardware oder im Internet haben zu müssen.

Die Auswahl der Daten bedeutet natürlich auch eine Einschränkung für die Ergebnisse dieser Arbeit. Die hier erarbeitete Pipeline dient zur generellen Erprobung des vorgeschlagenen Vorgehensmodells. Im Falle einer positiven Evaluation ist weitere Arbeit zur Anwendbarkeit auf weitere Domänen, Portierung verschiedener Datenquellen, Automatisierung des Datenein- und Ausgangs und insbesondere zur Skalierung notwendig.

4.4 Regelbasierte vs. LLM-basierte Transformation

An dieser Stelle soll einmal gezielt darauf eingegangen werden, warum in zwei der Transformationschritte ein regelbasierter Ansatz und nur beim dritten Transformationsschritt ein probabilistischer Ansatz gewählt wurde. Zunächst werden die beiden Ansätze konzeptionell eingeordnet.

Eine regelbasierte Transformation bezeichnet die Veränderung von Daten aufgrund von festgelegten Regeln. Dieser Ansatz ist deterministisch, und damit reproduzierbar und transparent. Wenn dieselbe Transformation vielfach ausgeführt wird, soll stets dasselbe Ergebnis erzeugt wer-

den. Die Ergebnisse lassen sich stets anhand der Regeln ableiten und theoretisch sollten sogar die Ursprungsdaten aus den transformierten Daten wiederhergestellt werden.

Eine LLM-basierte Transformation unterscheidet sich von der regelbasierten Transformation, indem sie probabilistisch ist, also auf Wahrscheinlichkeiten basierend. Die Ergebnisse einer solchen Transformation können bei wiederholten Durchläufen leicht variieren, insofern als dass sie unterschiedlich formuliert sein können oder Informationen unterschiedlich gewichtet werden. Somit ist dieser Ansatz nicht vollständig deterministisch und man kann vom Ergebnis auch nicht eindeutig auf die Eingabe rückschließen.

Für die DOM-Verarbeitung und das strukturelle Parsing, also Schritt 1 und 2 in der Vorverarbeitungs-Pipeline, wurde ein regelbasiertes Ansatz gewählt, da es sich hier um klare Strukturen und formale Kriterien handelt, anhand derer das gewünschte Ergebnis erreicht werden kann. Für diese Vorverarbeitungsschritte ist kein semantisches Verständnis der Informationen nötig, somit wäre ein LLM-basierter Ansatz nicht sinnvoll. Ein LLM sollte aus Gründen der Datensparsamkeit, Ressourcenschonung und Nachvollziehbarkeit nur eingesetzt werden, wenn eine regelbasierte nicht ausreicht.

Beim dritten Verarbeitungsschritt, der semantischen Normalisierung, ist jedoch genau dies der Fall. Hier reichen Regeln nicht aus, da aufgrund der Heterogenität der Datenlage ein semantisches Verständnis unerlässlich ist. Strukturierte und semistrukturierte Daten müssen aus dem Kontext heraus erfasst und verstanden werden. Hier ist der Einsatz eines LLM angemessen, da es in der Lage ist, die Bedeutung der Daten zu erschließen, sie in einen Kontext einzubetten und entsprechend der Anweisung umzuvormulieren.

Eine Trennung dieser beiden Ansätze in Form von drei Verarbeitungsschritten, ist sinnvoll, da die regelbasierte Vorverarbeitung von Schritt 1 und 2 die Grundlage für die Verwendung des LLM in Schritt 3 bieten. Durch den probabilistischen Charakter der semantischen Normalisierung ist die Nachvollziehbarkeit nicht in demselben Maße gegeben, wie bei der regelbasierten Transformation. Durch die Kontrolle über den Eingang der Daten in den probabilistischen Transformationsschritt wird Fehlern vorgebeugt und die Qualität der Ergebnisse erhöht. Dies ermöglicht einen sparsamen Einsatz eines LLM zugunsten besserer Kontrollierbarkeit, einer klaren Evaluation und einer besseren Reproduzierbarkeit.

Die Forschungsfrage zielt auf die Erprobung der semantischen Normalisierung zur Nutzbarmachung strukturierter Daten in einem RAG-System ab, aber für eine gezielte und reproduzierbare Untersuchung des Ansatzes ist der Einsatz von sowohl regelbasiertem als auch LLM-gestütztem Vorverarbeitungsschritt notwendig. Die Trennung zwischen den drei Vorverarbeitungsschritten ist sinnvoll, da dadurch die Möglichkeit besteht, den Transformationsprozess transparent zu überwachen und bei Fehlern gezielt nachzusteuern.

Kapitel 5

LLM-basierte Preprocessing-Pipeline

Dieses Kapitel beschreibt die konkrete Ausgestaltung der im vorherigen Kapitel eingeführten Vorverarbeitungspipeline zur Transformation strukturierter und semistrukturierter Wiki-Inhalte in eine für RAG-Systeme geeignete Wissensbasis. Die Pipeline ist dabei als vorgelagertes Preprocessing-System konzipiert, das zwischen dem Quellsystem (XWiki) und dem eigentlichen RAG-Modell angesiedelt ist. Sie ist nicht Bestandteil des RAG-Systems selbst, sondern stellt eine vorbereitende Infrastruktur dar, deren Aufgabe in der strukturierten Aufbereitung und semantischen Normalisierung der zugrundeliegenden Inhalte liegt.

Aus systemischer Sicht übernimmt die Pipeline die Verantwortung für die Transformation der Darstellungs- und Organisationsform von Wissen, nicht jedoch für dessen inhaltliche Erweiterung oder Bewertung. Ausgangspunkt sind HTML-basierte Wiki-Seiten, die neben Fließtext in hohem Maße strukturierte Elemente wie Tabellen, Listen und Konfigurationsfragmente enthalten. Diese Strukturen sind für menschliche Leser gut erfassbar, erweisen sich jedoch als problematisch für die direkte Nutzung in retrievalbasierten Sprachmodellen. Die Pipeline adressiert diese Diskrepanz, indem sie solche Inhalte in prosebasierte, kontextualisierte Wissenseinheiten überführt, ohne dabei externes Wissen einzubringen oder bestehende Aussagen semantisch zu verändern.

Die Pipeline operiert dabei offline und batch-orientiert und ist zeitlich vor der Indexierung der Inhalte in einem Vektorspeicher angesiedelt. Sie greift weder in den Retrieval- noch in den Generationsprozess des RAG-Systems ein und trifft keine Entscheidungen zur Laufzeit von Nutzeranfragen. Ihr Ergebnis ist ein persistenter Korpus normalisierter Texte, das als Eingabe für nachgelagerte RAG-Komponenten dient und deren Nutzbarkeit maßgeblich beeinflusst.

Kern des Vorgehens ist eine LLM-gestützte semantische Normalisierung, bei der strukturierte oder semistrukturierte Inhalte gezielt in Fließtext überführt werden. Diese Normalisierung stellt den einzigen probabilistischen Verarbeitungsschritt innerhalb der Pipeline dar und ist bewusst von deterministischen Vorverarbeitungsschritten wie Dokumentbereinigung, strukturellem Parsing und Chunking getrennt. Auf diese Weise wird eine klare Trennung zwischen regelbasierter Strukturverarbeitung und probabilistischer Sprachmodellnutzung realisiert.

Im Folgenden wird zunächst die konkrete Ausgestaltung dieser Pipeline beschrieben. Dazu werden das eingesetzte Prompt-Design für die Normalisierung strukturierter Inhalte, die zugrunde liegende Architektur sowie die technische Implementierung der einzelnen Verarbeitungsschritte dargestellt.

5.1 Prompt-Engineering für die Tabellenanreicherung

Für die Anreicherung der Rohdaten mit Kontext wird ein LLM angewiesen, eine strukturierte Datei einzulesen, zu interpretieren und die Informationen mit Kontext versehen in eine neue, unstrukturierte Datei zu schreiben. Die ersten Versuche wurden mit einer 100-zeiligen CSV-Datei gemacht. Hierfür wurde folgender Systemprompt verwendet:

Systemprompt

Du erhältst eine Tabelle aus einer (fiktiven) Dokumentation.

Tabelleninhalt (CSV): {table_csv}

Aufgabe: 1. Beschreibe kurz, was diese Tabelle insgesamt darstellt.

2. Erkläre die Bedeutung jeder Spalte.

3. Formuliere jede einzelne Zeile in einen beschreibenden Satz um, in dem das Attribut benannt und mit dem Wert versehen wird.

Formatiere die Antwort als gut lesbaren Fließtext.

Dieser Prompt wird noch weiter angepasst, um ihn auf verschiedene Formate (semi-)strukturierter Daten anzupassen.

5.2 Architektur

Das Vorgehen folgt dem Prinzip einer mehrstufigen ETL-artigen Datenpipeline, in der Rohdaten aus einem Wiki extrahiert, schrittweise transformiert und schließlich in eine für RAG-Systeme geeignete Wissensbasis überführt werden. Ziel der Architektur ist eine modulare, reproduzierbare Datenpipeline, in der regelbasierte und LLM-basierte Transformationsschritte klar voneinander getrennt sind. Die dafür nötigen Schritte sind in Abbildung 5.1 zu sehen und werden im Folgenden näher beschrieben.

5.2.1 Datenextraktion

Der Extraktionsschritt stellt den Einstiegspunkt der Pipeline dar und umfasst die Bereitstellung der Rohdokumente in einem einheitlichen Eingangsformat. Da wie in 4.3 beschrieben, eine händische Extraktion stattfand, wird dieser Punkt in dieser Arbeit nicht weiter beachtet.

5.2.2 DOM-Vorverarbeitung

An dieser Stelle beginnt in der praktischen Umsetzung der erste regelbasierte Transformationsschritt. Hier findet eine DOM-Vorverarbeitung statt, in der das inhaltsstärkste DOM-Element basierend auf bekannter Wiki-Syntax gefunden und der Rest verworfen wird. Aus diesem Block können dann noch UI-Nodes wie Navigations- und Bearbeitungslinks entfernt werden. Dieser Schritt dient dazu, das Rauschen in den Daten zu reduzieren, und damit die Ergebnisqualität zu erhöhen. Durch die Reduktion von unnötigen Datenflüssen in der Pipeline werden außerdem Speicherressourcen gespart. Das Eingangsformat ist aus anwendungsspezifischen Gründen .htm, und das Ausgangsformat aus diesem Verarbeitungsschritt wird .html sein, da es einfacher bekannter und intuitiver ist. In Abgrenzung zu den anderen Transformationsschritten ist dieser Schritt regelbasiert und deterministisch und sollte deshalb klar vor dem semantischen Parsing liegen.

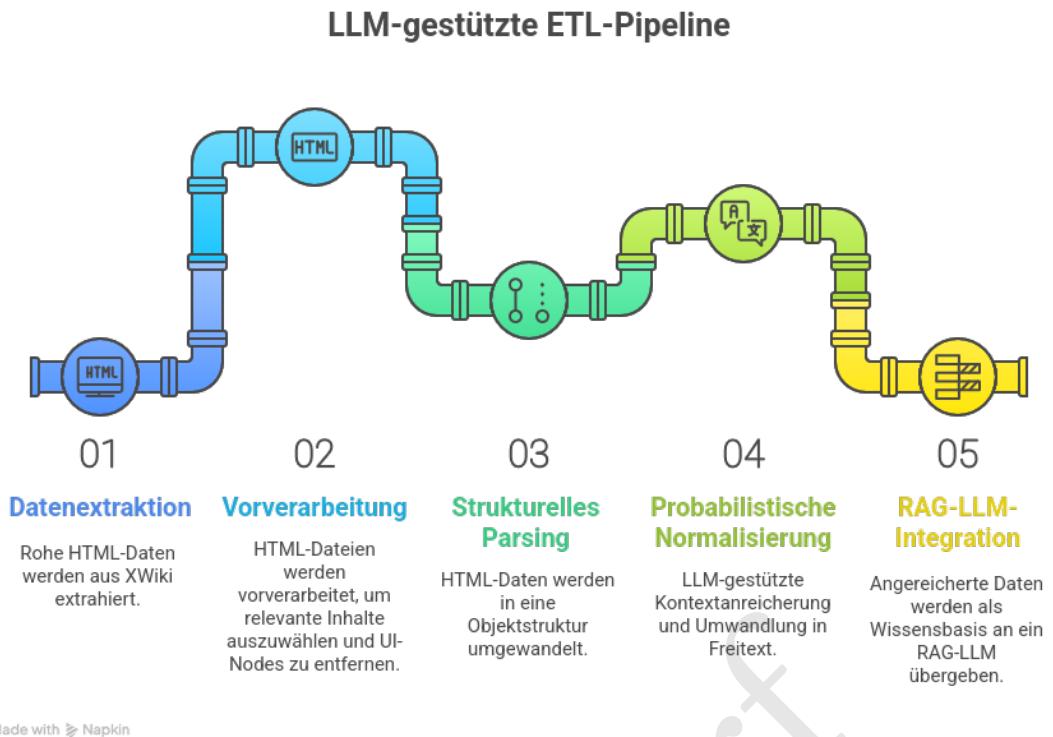


Abbildung 5.1: Illustration der Datenvorverarbeitung

5.2.3 Strukturelles Parsing

Im zweiten Transformationsschritt findet ein strukturelles Parsing statt. Das Ziel dieses Schritts ist, aus dem bereinigten HTML eine Objektstruktur zu machen. Überschriften werden in Sections umgewandelt, Absätze in Paragraph-Blöcke, Tabellen in Tabellen-Blöcke und so weiter. Das Eingabeformat ist das Ergebnis der DOM-Vorverarbeitung, also .html-Blöcke. Das Ausgangsformat sind .json-Dateien, die eine Objektrepräsentation mit hoher Informationsdichte sein sollen, die von Steuer- und Anzeigeeinrichtungen bereinigt sind und somit als Eingangsformat in den nächsten Transformationsschritt dienen. Dieser Schritt ist ein struktureller Adapter zwischen bereinigten .html-Dateien und interner Objektrepräsentation und begründet somit seine KapSELung.

5.2.4 Probabilistische Normalisierung

Die probabilistische Normalisierung ist der dritte Transformationsschritt, in dem die LLM-basierte semantische Normalisierung stattfindet. Hier wird das erste Mal ein LLM in die Pipeline integriert. Dieses bekommt die geparsten Objekte aus dem Semantischen Parsing zugeliefert und übergibt sie an ein LLM mit der Anweisung, diese Informationen in einen Fließtext umzuwandeln. Das LLM soll die Datei erfassen, einen Kontext schaffen, indem es die Datei zusammenfasst, selbst Strukturen wie „Parameter“/„Wert“-Paare erkennen, und die in den Kontext einbetten. Der wichtigste Teil dieses Schrittes liegt darin, dass semi- und strukturierte Daten durch das LLM erläutert werden, selbst wenn das bedeutet, dass auf repetitive Weise dieselbe Satz immer wieder wiederholt wird. Ein Beispiel dafür, wie das aussehen könnte, ist in ?? und ?? zu sehen. Das Ziel dieses Schritts ist, die Fähigkeit eines LLMs zur Inferenz zu nutzen, um strukturierte Daten zu verstehen und umzuformen und damit bekannte Schwächen von RAG-Systemen im Umgang mit strukturierten Wissensrepräsentationen zu adressieren. Das Schwierige an diesem Schritt ist, den Systemprompt so generisch wie möglich zu halten, damit verschiedenen Eingangsdaten

erfasst und weiterverarbeitet werden, aber trotzdem präzise genug, um brauchbare Ergebnisse zu erhalten. Das Eingangsformat ist das Ergebnis des letzten Verarbeitungsschritts und das Ausgangsformat ist .md.

5.2.5 RAG-LLM-Integration

Hier ist der *Load*-Schritt der klassischen ETL-Pipeline, nur dass der in diesem Projekt die Bereitstellung der transformierten Daten an die Wissensbasis des RAG-Modells umfasst. Das Ergebnis dieses Schritts ist die Integration in die RAG-Wissensbasis. Hier wird dies, genauso wie der Extraktionsschritt händisch durchgeführt, indem die Ergebnisdateien der Probabilistischen Normalisierung in eine Wissensbasis in der Open WebUI-Anwendung überführt werden. Zukünftig sollte auch dieser Schritt automatisiert stattfinden, indem die Übergabe an die RAG-Wissensbasis ebenfalls per API-Schnittstelle stattfindet, um Fehlern vorzubeugen.

NEU

5.3 Implementierung

Die Implementierung der LLM-gestützten Vorverarbeitungspipeline wurde in Python realisiert und als eigenständiges Paket mit der Bezeichnung *struct2prose* umgesetzt. Ziel der Implementierung ist es, die in Kapitel 4 beschriebene Vorgehensweise in eine reproduzierbare, modular ausführbare Pipeline zu überführen, ohne dabei unnötige technische Komplexität einzuführen. Der Fokus liegt auf der Nachvollziehbarkeit der einzelnen Verarbeitungsschritte sowie auf der klaren Trennung deterministischer und LLM-basierter Transformationslogik.

5.3.1 Technischer Rahmen und Projektstruktur

Die Pipeline wird über eine Kommandozeilenschnittstelle gesteuert, die den Einstiegspunkt des Systems bildet. Die Ausführung erfolgt über den Befehl `python -m struct2prose`, wobei sowohl einzelne Verarbeitungsschritte als auch ein vollständiger End-to-End-Durchlauf angestoßen werden können. Die einzelnen Schritte sind als Subcommands implementiert und folgen einer linearen Abfolge von der Rohdatenverarbeitung bis zur semantisch normalisierten Ausgabe. Die zugehörige Verzeichnisstruktur orientiert sich an den Pipeline-Stufen und umfasst getrennte Ordner für Rohdaten (`raw_data`), bereinigte HTML-Dokumente (`clean_data`), strukturierte Zwischenrepräsentationen (`processed_data`) sowie normalisierte Markdown-Dokumente (`normalized_data`).

5.3.2 Interne Datenrepräsentation

Als zentrales internes Datenformat dient eine explizite Objektstruktur, bestehend aus den Klassen `WikiDocument`, `Section` und `ContentBlock`. Diese Repräsentation abstrahiert von der ursprünglichen HTML-Struktur und bildet stattdessen die logische Gliederung der Wiki-Seiten ab. Ein Dokument besteht aus einem Titel, einer geordneten Liste von Abschnitten sowie Metadaten zur Herkunftsdatei. Jeder Abschnitt enthält wiederum eine Folge typisierter Inhaltsblöcke, etwa Absätze, Tabellen, Listen oder Codeblöcke. Diese Struktur stellt sicher, dass strukturierte und unstrukturierte Inhalte bereits vor der LLM-gestützten Verarbeitung explizit unterschieden vorliegen.

5.3.3 Implementierung der Vorverarbeitungsschritte (Step 1–3)

Die Vorverarbeitungsschritte eins bis drei sind vollständig deterministisch implementiert. Im ersten Schritt wird aus den HTML-Exporten der zentrale Inhaltsbereich der Wiki-Seite extrahiert. Hierzu wird heuristisch ein definierter DOM-Knoten als Content-Root ausgewählt, um

Navigations- und Metadatenbereiche frühzeitig auszuschließen. Der zweite Schritt entfernt verbliebene UI-Elemente wie Inhaltsverzeichnisse oder Editier-Controls anhand einfacher, regelbasierter Filter. Dieser Schritt ist bewusst minimal gehalten und verfolgt nicht das Ziel eines vollständigen DOM-Cleanings. Im dritten Schritt erfolgt das eigentliche Parsing des bereinigten HTML in ein strukturiertes JSON-Zwischenformat. Überschriften werden dabei als Abschnittsgrenzen interpretiert, während Absätze, Tabellen, Listen und Codeblöcke jeweils als eigenständige Inhaltsblöcke modelliert werden.

5.3.4 LLM-gestützte semantische Normalisierung (Step 4)

Die semantische Normalisierung strukturierter Inhalte erfolgt im vierten Schritt der Pipeline und stellt den zentralen Einsatzpunkt eines Large Language Models dar. Hierzu wird das Groq-Framework verwendet, wobei der Zugriff über einen API-Key erfolgt, der als Environment-Variable konfiguriert ist. Als Modell kommtt `llama-3.3-70b-versatile` mit einer niedrigen Temperatur zum Einsatz, um möglichst konsistente und reproduzierbare Ausgaben zu erzielen. Die Normalisierung wird selektiv angewendet: Tabellen und optional Listen werden mittels promptbasierter LLM-Anfragen in beschreibenden Fließtext überführt, während bereits gut lesbare Absätze unverändert übernommen werden. Codeblöcke werden ebenfalls nicht inhaltlich verändert, sondern lediglich als zusammenhängende Codeabschnitte in die Ausgabe integriert.

5.3.5 Generierung der normalisierten Markdown-Dokumente

Als Ergebnis der Normalisierung werden die Dokumente im Markdown-Format ausgegeben. Die Struktur der Ausgabedateien folgt dabei der ursprünglichen Abschnittsgliederung der Wiki-Seiten, ersetzt jedoch tabellarische Darstellungen durch prosebasierte Beschreibungen. Dieses Format eignet sich besonders für die spätere Integration in Retrieval-Augmented-Generation-Systeme, da es semantisch kohärente Textsegmente erzeugt und gängige Chunking-Strategien unterstützt.

5.3.6 Limitationen der Implementierung

Die Implementierung ist bewusst als Minimalprototyp ausgelegt. Sie verzichtet auf komplexe Heuristiken, tiefgreifende HTML-Normalisierung oder umfangreiche Nachbearbeitung der LLM-Ausgaben. Diese Einschränkungen sind bewusst gewählt, um den methodischen Kern der Arbeit – die LLM-gestützte semantische Normalisierung strukturierter Inhalte – klar sichtbar zu machen und eine saubere Grundlage für die nachfolgende Evaluation zu schaffen.

Kapitel 6

Evaluation

Um das hier vorgestellte System auf Tauglichkeit zu prüfen, wird im Folgenden zunächst die Zielsetzung der Evaluation definiert, dann die Methoden erläutert und zum Schluss die Ergebnisse vorgestellt.

6.1 Zielsetzung der Evaluation

Zur Evaluation der Pipeline und des daraus resultierenden LLM werden keine Benchmarks verwendet, sondern eher ein Test auf Funktion durchgeführt. Folgende Fragen sollen in der Evaluation beantwortet werden:

- Ist der Ansatz grundsätzlich sinnvoll?
- Ist er in dem vorhandenen Umfeld anwendbar?
- Stellt das System eine Verbesserung zu der Arbeit mit den Rohdaten dar?

6.2 Methoden

Hier wird das angereicherte RAG-LLM (im Folgenden: Enhanced-LLM), was Ergebnis dieser Arbeit war, gegen ein RAG-LLM gegenübergestellt, was mit Rohdaten ohne Anreicherung auskommen muss (im Folgenden: Baseline-LLM). Diesen beiden LLMs werden 5-10 Fragen gestellt, deren Antworten in ihren Wissensbasen enthalten sein sollten. Zudem werden ihnen Fragen gestellt, deren Antworten nicht in den Wissensbasen enthalten sein sollten. Zuletzt werden Fragen gestellt, deren Antworten aus verschiedenen Quellen zu ziehen sind und die Antwortqualität der beiden Systeme verglichen. Die Qualität wird vor allem auf die Punkte Faktentreue, Nützlichkeit und Verständlichkeit überprüft.

RAG-Evaluation Beispiel: https://github.com/groq/groq-api-cookbook/blob/main/tutorials/04-rag/benchmarking-rag-langchain/benchmarking_ra

6.3 Ergebnisse

- Qualitative Beispiele
- Kleine Tabelle

Welchen Port verwendet LO_TEST05 in der Datei `instanzen.csv`?

openai/gpt-oss-120b

- Keine Quellen gefunden

Für weniger als eine Sekunde nachgedacht ▾

Um dir die Information zum Port von **LO_TEST05** geben zu können, brauche ich den Inhalt der entsprechenden Zeile aus deiner Datei `instanzen.csv`. Bitte kopiere die Zeile (oder den relevanten Abschnitt) aus der CSV-Datei in deine Nachricht, dann kann ich den gewünschten Port für dich heraussuchen.

Welchen Port verwendet LO_TEST05 in der Datei `enriched_Instanzen.csv`?

openai/gpt-oss-120b

- 1 Quelle abgerufen

Für weniger als eine Sekunde nachgedacht ▾

Der Port für die Instanz **LO_TEST05** ist **8084** `enriched_Instanzen.txt`.

1 Quelle

Was weißt du über DMS_TEST05?

openai/gpt-oss-120b

- Keine Quellen gefunden

Nachgedacht für 1 Sekunden ▾

Ich habe leider keine Informationen zu „DMS_TEST05“. Wenn du mir mehr Kontext oder Details dazu gibst, kann ich dir gerne weiterhelfen.

Was weißt du über DMS_TEST05?

openai/gpt-oss-120b

- 1 Quelle abgerufen

Nachgedacht für 1 Sekunden ▾

Die Server-Instanz **DMS_TEST05** hat folgende Eigenschaften:

- IP-Adresse:** 10.0.10.84
- fachlicher FQDN:** dms-test.app.machlab.internal
- technischer FQDN:** dms-test05.srv.machlab.internal
- SID:** DMST05
- Benutzername:** svc_dms_test
- Port:** 8084

Diese Angaben stammen aus dem bereitgestellten Kontext `enriched_Instanzen.txt`.

1 Quelle

Abbildung 6.2: Test 2

Kapitel 7

Diskussion

- Nutzen des Ansatzes
- Grenzen (Kosten, LLM-Fehler, Halluzinationen, Zeitaufwand)
- Abhängigkeit von Qualität der Wiki-Daten
- Vergleich mit Related Work

Entwurf

Kapitel 8

Fazit und Ausblick

-Skalierung -Anwendbarkeit auf andere Daten/Domänen -Automatisierung von Extract und Load

- Zusammenfassung
- Beitrag der Arbeit
- Perspektiven:
 - Graph-RAG
 - Mixed-modal embeddings
 - Automatische Aktualisierung

Entwurf

Literatur

- Asai, Akari u. a. [17. Okt. 2023]. *Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection*. doi: 10 . 48550 / arXiv . 2310 . 11511. arXiv: 2310 . 11511[cs]. URL: http : //arxiv.org/abs/2310.11511 [besucht am 03. 11. 2025].
- Bommasani, Rishi u. a. [12. Juli 2022]. *On the Opportunities and Risks of Foundation Models*. doi: 10 . 48550/arXiv.2108.07258. arXiv: 2108 . 07258[cs]. URL: http://arxiv.org/abs/2108 . 07258 [besucht am 09. 10. 2025].
- Chang, Yupeng u. a. [29. Dez. 2023]. *A Survey on Evaluation of Large Language Models*. doi: 10 . 48550/arXiv . 2307 . 03109. arXiv: 2307 . 03109[cs]. URL: http://arxiv.org/abs/2307 . 03109 [besucht am 09. 10. 2025].
- Devlin, Jacob u. a. [o. D.] “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: []
- Gao, Yunfan u. a. [27. März 2024]. *Retrieval-Augmented Generation for Large Language Models: A Survey*. doi: 10 . 48550/arXiv . 2312 . 10997. arXiv: 2312 . 10997[cs]. URL: http://arxiv . org/abs/2312 . 10997 [besucht am 03. 11. 2025].
- Jiang, Zhengbao u. a. [22. Okt. 2023]. *Active Retrieval Augmented Generation*. doi: 10 . 48550 / arXiv . 2305 . 06983. arXiv: 2305 . 06983[cs]. URL: http : //arxiv . org / abs / 2305 . 06983 [besucht am 03. 11. 2025].
- Knollmeyer, Simon u. a. [2024]. “Benchmarking of Retrieval Augmented Generation: A Comprehensive Systematic Literature Review on Evaluation Dimensions, Evaluation Metrics and Datasets.” In: *Proceedings of the 16th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*. 16th International Conference on Knowledge Management and Information Systems. Porto, Portugal: SCITEPRESS - Science und Technology Publications, S. 137–148. ISBN: 978-989-758-716-0. doi: 10 . 5220 / 0013065700003838. URL: https : / / www . scitepress . org / DigitalLibrary / Link . aspx ? doi = 10 . 5220 / 0013065700003838 [besucht am 09. 10. 2025].
- Lewis, Patrick u. a. [12. Apr. 2021]. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. doi: 10 . 48550/arXiv . 2005 . 11401. arXiv: 2005 . 11401[cs]. URL: http://arxiv.org/abs/2005 . 11401 [besucht am 09. 10. 2025].
- Sui, Yuan u. a. [22. Mai 2023]. *Table Meets LLM: Can Large Language Models Understand Structured Table Data? A Benchmark and Empirical Study*. arXiv.org. URL: https : //arxiv . org / abs / 2305 . 13062v5 [besucht am 25. 11. 2025].
- Wolff, Cornelius und Madelon Hulsebos [4. Nov. 2025]. *How well do LLMs reason over tabular data, really?* doi: 10 . 48550 / arXiv . 2505 . 07453. arXiv: 2505 . 07453[cs]. URL: http : //arxiv.org/abs/2505 . 07453 [besucht am 01. 12. 2025].
- Zhang, Yue u. a. [14. Sep. 2025]. *Siren’s Song in the AI Ocean: A Survey on Hallucination in Large Language Models*. doi: 10 . 48550 / arXiv . 2309 . 01219. arXiv: 2309 . 01219[cs]. URL: http : //arxiv.org/abs/2309 . 01219 [besucht am 03. 11. 2025].