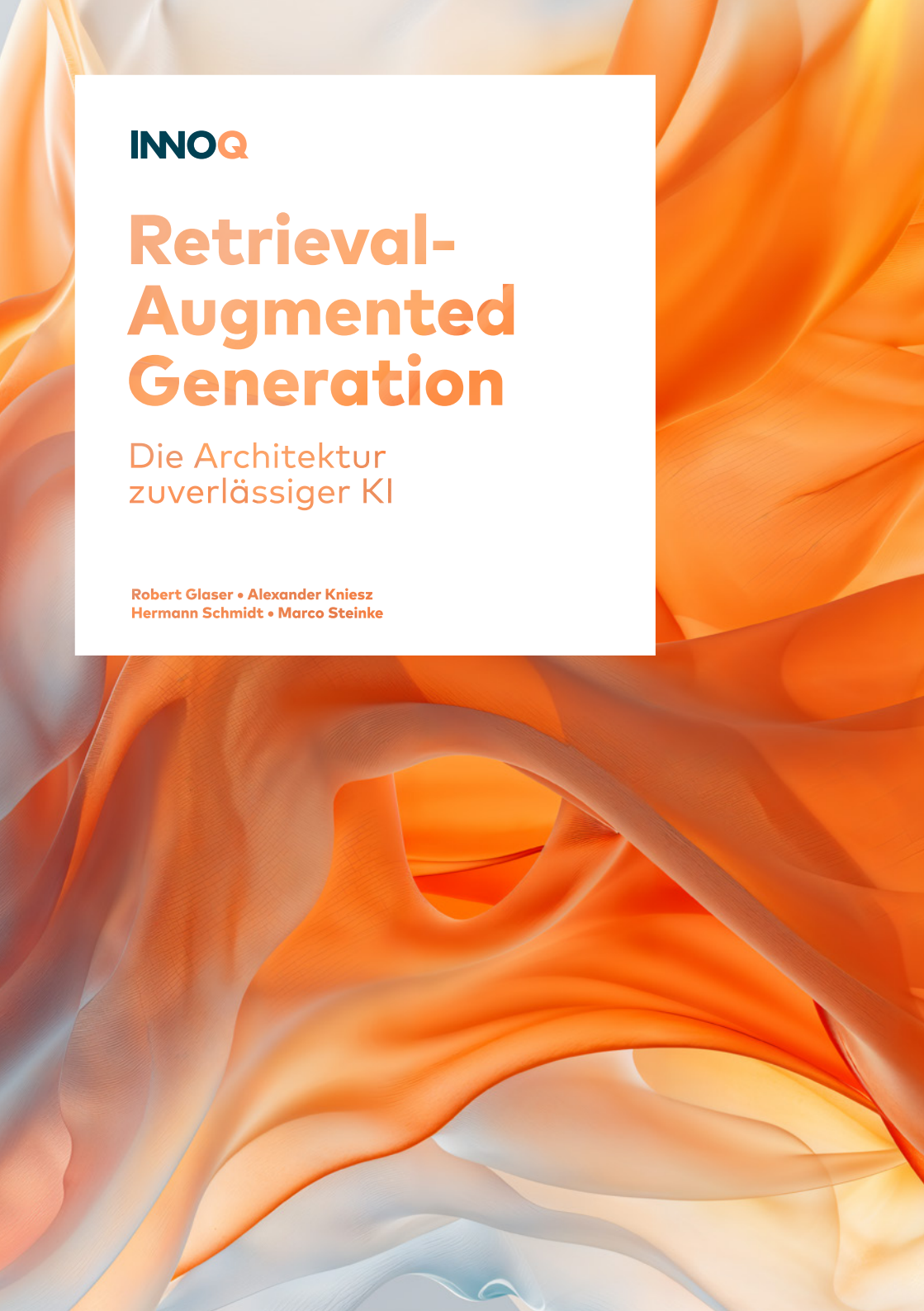

INNOQ
Retrieval-Augmented Generation
Die Architektur zuverlässiger KI
Robert Glaser • Alexander Kniesz
Hermann Schmidt • Marco Steinke

RAG: Retrieval-Augmented Generation

Die Architektur zuverlässiger KI

Robert Glaser
Alexander Kniesz
Hermann Schmidt
Marco Steinke

innoQ Deutschland GmbH
Krischerstraße 100 · 40789 Monheim am Rhein · Germany
Phone +49 2173 33660 · www.INNOQ.com

Layout: Tammo van Lessen with XeLaTeX
Umschlag: Murat Akgöz
Typesetting: André Deuerling

RAG: Retrieval-Augmented Generation – Die Architektur zuverlässiger KI
Published by innoQ Deutschland GmbH
1. Auflage · November 2024

Inhaltsverzeichnis

1 Grundlagen von Large Language Models (LLMs)

Generative KI und Large Language Models (LLMs) im Speziellen haben in den letzten Jahren die Welt der künstlichen Intelligenz revolutioniert. Sie stellen nicht nur einen technologischen Durchbruch in Form einer neuen Allzwecktechnologie dar, sondern verändern grundlegend die Art und Weise, wie wir mit Computern interagieren und Informationen verarbeiten. Für Softwareentwickler:innen und -architekt:innen ist es unerlässlich, die Funktionsweise, Stärken und Grenzen dieser Technologie zu verstehen, um fundierte Entscheidungen über ihren Einsatz treffen zu können.

1.1 Wie funktionieren LLMs?

Im Kern sind LLMs hochkomplexe neuronale Netzwerke, die auf enormen Mengen von Daten trainiert wurden. Sie nutzen die Transformer-Architektur mit ihrem Attention-Mechanismus, um weitreichende Textzusammenhänge zu erfassen und parallel zu verarbeiten. Eine interaktive Erklärung bietet der Transformer Explainer[1].

Der Begriff „Language“ in „Large Language Models“ kann irreführend sein. Obwohl LLMs ursprünglich für Sprach- und Textverarbeitung konzipiert wurden, sind ihre Einsatzmöglichkeiten deutlich vielfältiger. Grundsätzlich eignet sich alles für die Verarbeitung durch LLMs, was man in Tokenform in Transformer-Modelle einspeisen kann. Dazu zählen nicht nur Text, sondern auch:

- Bild
- Video
- Gesprochene Sprache
- Molekulare Strukturen (z. B. Proteine)

Diese Vielseitigkeit macht LLMs zu einem mächtigen Werkzeug für eine Vielzahl von Anwendungen, die weit über die reine Textverarbeitung hinausgehen.

[1] https://poloclub.github.io/transformer-explainer/

Der Trainingsprozess eines LLM lässt sich vereinfacht so beschreiben:

1. **Datensammlung**: Zunächst wird eine große Menge an Daten aus verschiedenen Quellen gesammelt. Dies können Texte, aber auch andere Arten von Daten sein, die in eine tokenisierte Form gebracht werden können.
2. **Vorverarbeitung**: Die Daten werden bereinigt und in ein einheitliches Format gebracht. Sie werden in sogenannte Tokens zerlegt - kleine Einheiten, die je nach Datentyp Wörter, Wortteile, Bildpixel, Audioframes oder andere Elemente repräsentieren können.
3. **Training**: Das Modell wird darauf trainiert, das nächste Token in einer Sequenz vorherzusagen. Dies geschieht durch wiederholtes Durchlaufen der Trainingsdaten und Anpassung der Modellparameter.
4. **RLHF** (**Reinforcement Learning from Human Feedback**): Nach dem initialen Training wird das Modellverhalten durch RLHF weiter verbessert. Dieser Prozess wird auch als „Alignment“ (Ausrichtung) bezeichnet. Dabei wird menschliches Feedback genutzt, um das Modell so anzupassen, dass es Antworten generiert, die besser den menschlichen Präferenzen, Werten und Erwartungen entsprechen. Ziel ist es, das KI-System stärker mit menschlichen Intentionen in Einklang zu bringen.

Der Schlüssel zum Verständnis von LLMs liegt in ihrer Fähigkeit zur Tokenprädiktion. Sie lernen nicht nur einzelne Fakten, sondern erfassen komplexe statistische Muster in den Daten. Dies ermöglicht es ihnen, kontextabhängige und oft erstaunlich kohärente Ausgaben zu generieren, sei es in Form von Text, Bildern oder anderen Datentypen.

1.2 Stärken und Grenzen von LLMs

Die Stärken von LLMs sind beeindruckend:

- **Flexibilität**: Sie können auf eine Vielzahl von Aufgaben angewendet werden, ohne spezifisch dafür programmiert worden zu sein. Dabei lassen sie bisherige Machine Learning (ML) Ansätze oft starr wirken und kannibalisieren zahlreiche ML Use Cases.

- **Kontextverständnis**: LLMs können den Kontext einer Anfrage erstaunlich gut erfassen und relevante Antworten generieren.
- **Kreativität**: Sie können neue Ideen generieren und bei kreativen Aufgaben unterstützen.
- **Commodity-Faktor:** LLMs sind für Entwickler:innen sehr einfach zu integrieren, oft durch einfache API-Aufrufe.

Allerdings haben LLMs auch bedeutende Einschränkungen:

- **Mangelnde Erklärbarkeit**: Im Gegensatz zu regelbasierten Systemen ist es durch den Nicht-Determinismus von LLMs oft schwierig nachzuvollziehen, wie sie zu einer bestimmten Ausgabe kommen.
- **Ressourcenintensität**: Das Training von LLMs erfordert enorme Rechenleistung und Energie. Die Inferenzkosten sind zur Drucklegung hoch, profitieren aber bereits jetzt deutlich von Skaleneffekten.
- **Aktualität der Informationen**: LLMs sind auf den Stand ihres Trainingsdatums beschränkt und können ohne zusätzliche Maßnahmen wie RAG oder *Function Calling* keine aktuellen oder internen Informationen verarbeiten.
- **Halluzinationen**: LLMs können plausibel klingende, aber faktisch falsche Informationen generieren, besonders, wenn der Kontext fehlt.

Da es sich hier um eine Allzwecktechnologie handelt, ist die Aufzählung von Use Cases über Branchen hinweg gleichzeitig sehr müßig und sehr spannend[2]. Im Grundsatz ermöglichen LLMs Features, die zuvor zu teuer oder unmöglich waren.

1.3 Vergleich verschiedener Ansätze

Es ist wichtig, LLMs im Kontext anderer KI-Ansätze zu betrachten:

1. **Regelbasierte Systeme**:

- Vorteile: Vorhersehbarkeit, Erklärbarkeit, geringer Ressourcenbedarf
- Nachteile: Begrenzte Flexibilität, aufwändige manuelle Pflege

[2] https://www.innoq.com/en/articles/2024/10/generative-ai-in-business-software/

2. **Traditionelle Machine Learning-Modelle**:

- Vorteile: Effizient für sehr spezifische Aufgaben, oft besser interpretierbar als LLMs
- Nachteile: Erfordern oft manuelles Feature Engineering und aufwändige Datenvorbereitung, weniger flexibel

3. **Large Language Models**:

- Vorteile: Hohe Flexibilität, Fähigkeit zur Generalisierung, Verarbeitung natürlicher Sprache
- Nachteile: Hoher Ressourcenbedarf, Erklärbarkeit, potenzielle Halluzinationen

Die Wahl des richtigen Ansatzes hängt stark von der spezifischen Aufgabe, den verfügbaren Ressourcen und den Anforderungen an Erklärbarkeit und Kontrolle ab.

1.4 Missverständnisse und Klarstellungen

Ein häufiges Missverständnis ist, dass LLMs „verstehen“, was sie ausgeben. In Wirklichkeit operieren sie auf statistischen Mustern, ohne echtes Verständnis im menschlichen Sinne. Dies kann zu unerwarteten Ergebnissen führen, wenn das Modell mit statistisch unbekannten bzw. ungenauen Kontexten konfrontiert wird.

Ein weiterer kritischer Punkt ist die potenzielle Voreingenommenheit (Bias) in LLMs. Da sie auf existierenden Daten trainiert werden, können sie gesellschaftliche Vorurteile und Ungleichheiten reproduzieren. Führende große Modellanbieter wie OpenAI oder Anthropic setzen zwar aufwändiges RLHF ein, um diese Problematik zu minimieren, jedoch kann Bias nicht vollständig eliminiert werden. Besonders bei kleineren oder spezialisierten Modellen bleibt dies eine Herausforderung.

1.5 Fazit und Ausblick

LLMs stellen einen bedeutenden Durchbruch in der KI dar, bringen aber auch neue Herausforderungen mit sich. Ihre effektive Nutzung erfordert ein tiefes Verständnis ihrer Funktionsweise, Stärken und Grenzen.

Eine zentrale Frage, die sich für viele Organisationen stellt, ist: Wie können wir die Leistungsfähigkeit von LLMs für unsere internen Daten nutzen? Dies führt uns zum Konzept der Retrieval-Augmented Generation (RAG), das wir in den folgenden Kapiteln genauer betrachten werden.

2 Retrieval-Augmented Generation: Wenn Weltwissen auf Spezialwissen trifft

Im ersten Kapitel haben wir die grundlegenden Funktionsweisen und Stärken von LLMs kennengelernt. Während LLMs ein enormes Potenzial besitzen, um auf Basis ihres umfassenden Wissensschatzes durch das Training generische Anfragen zu beantworten, zeigen sich ihre Schwächen, wenn es um aktuelle, spezialisierte oder verifizierte Informationen geht. Genau hier setzt Retrieval-Augmented Generation (RAG) an – ein Ansatz, der die Sprachfähigkeiten der LLMs mit der Fähigkeit verbindet, auf dynamische und spezifische Datenquellen zuzugreifen. So lässt sich Weltwissen mit Spezialwissen verknüpfen, um kontextabhängige und präzise Antworten zu liefern und außerdem zu reproduzieren, welche Daten für die Beantwortung eines Prompts verwendet wurden.

2.1 Die Grenzen von LLMs und RAG als Lösung

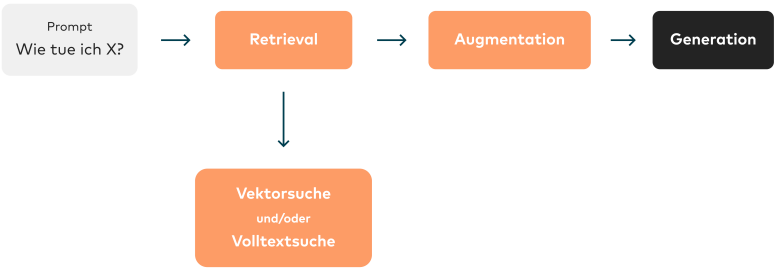
LLMs erhalten ihr Wissen aus einem Trainingsdatensatz und können daher nur Informationen abrufen, die bis zum letzten Trainingszeitpunkt existierten. In dynamischen Umfeldern, wo Aktualität und Fachwissen entscheidend sind, ist dies eine große Einschränkung. RAG bietet eine Lösung, indem es LLMs ermöglicht, Informationen aus externen und aktuellen Datenquellen hinzuzuziehen. Diese Verknüpfung mit verifiziertem Wissen erhöht die Präzision und Aktualität der generierten Inhalte.

2.1.1 Was ist RAG?

Retrieval-Augmented Generation kombiniert die Sprachfähigkeiten eines LLMs mit einem Retrieval-System, das in der Lage ist, auf relevante Daten zuzugreifen und diese dem Modell zur Verfügung zu stellen. Der Ablauf sieht dabei in etwa so aus:

1. **Benutzeranfrage**: Eine Benutzerin stellt eine Anfrage, die möglicherweise aktuelle oder spezifische Informationen erfordert.
2. **Retrieval**: Ein Retrieval-System durchsucht definierte externe Quellen, wie Datenbanken oder Wissensspeicher, und findet passende Dokumente oder Textabschnitte.
3. **Augmentation**: Diese relevanten Informationen werden dem LLM übergeben und dienen als kontextuelle Grundlage für die Generierung der Antwort.
4. **Generierung**: Das LLM verwendet die zusätzlichen Daten und erstellt eine fundierte, kontextualisierte Antwort.

Durch diese Kombination kann RAG Antworten liefern, die nicht nur allgemeines Wissen, sondern auch hochspezifische und aktuelle Informationen enthalten – eine erhebliche Verbesserung gegenüber reinen LLMs. [1]



[1] Es ist eigentlich eine Illusion, dass dem LLM durch *In-Context Learning* zusätzliches Wissen mitgegeben wird. Der bereitgestellte Kontext beeinflusst die Berechnungen im *Attention-Mechanismus* des Transformer-Netzwerks. Dennoch kann ein LLM nichts errechnen, was es nicht kennt. Durch die enorme Größe der LLMs ist der Vorrat an Mustern quasi unerschöpflich und durch Kontext aus typischen Geschäftsdaten kaum an seine Grenzen zu bringen. Was wir als Halluzinationen wahrnehmen, ist das Ergebnis von fehlgeleiteten Berechnungen. RAG sorgt mit zusätzlichem Kontext für eine stabilere, zielgerichtete Berechnung in unserem Themenbereich. Daraus entsteht die Illusion, das LLM hätte unsere Daten „verstanden“.

2.2 Grounding: Die Verankerung in verifizierten Daten

Ein zentraler Vorteil von RAG ist das sogenannte „Grounding“ – die Verankerung der Antworten in überprüfbaren Datenquellen. Dadurch wird das generierte Wissen präziser und zuverlässiger, da die Antworten auf expliziten Datenquellen basieren, die der Benutzer nachvollziehen kann. Diese Verankerung ist entscheidend in Bereichen wie Medizin, Wissenschaft und Recht, wo genaue und verifizierbare Informationen unabdingbar sind. Mit RAG lassen sich generierte Antworten auf eine überprüfbare Datenbasis stützen, was die Sicherheit und Qualität der Informationen erhöht.

2.3 Die Motivation hinter RAG

Die Entwicklung von RAG basiert auf mehreren wichtigen Motivationen:

1. **Aktualität**: RAG ermöglicht es, LLMs mit aktuellen Informationen zu versorgen, indem sie auf externe Quellen zugreifen. Organisationen, die häufig neue Inhalte produzieren – beispielsweise Forschungsberichte oder Marktanalysen – können durch RAG stets aktuelle Daten in ihre Antworten einfließen lassen.

2. **Spezialisiertes Wissen**: Viele Unternehmen verfügen über internes Wissen, das für spezifische Anwendungsfälle unverzichtbar ist. Durch RAG kann dieses spezielle Wissen in die Antworten eingebunden werden, wodurch die Nützlichkeit und Anwendbarkeit der generierten Antworten in professionellen Kontexten gesteigert wird.

3. **Vertrauenswürdigkeit**: Mit RAG lässt sich nachvollziehen, welche Quellen in die Antwort eingeflossen sind. Dies ist besonders in kritischen Szenarien wertvoll, in denen Genauigkeit und Verlässlichkeit der Antworten entscheidend sind.

4. **Effizienz und Skalierbarkeit**: RAG macht es möglich, spezialisiertes Wissen effizient zu nutzen, ohne das zugrunde liegende LLM ständig neu trainieren

zu müssen. Außerdem werden einem LLM ausschließlich relevante Informationen übergeben, was die Kosten und die Antwortzeit reduziert.

2.4 Fazit: Ein LLM mit verlässlichen Daten

RAG ist die Brücke zwischen den generischen Fähigkeiten eines LLMs und den Anforderungen an aktualisiertes, spezialisiertes Wissen. Durch die Kombination von LLM und dynamischen, verifizierten Datenquellen wird das Modell zu einem umfassenden Wissenslieferanten und zugleich einem spezialisierten Berater. So entsteht ein System, das nicht nur informative, sondern auch fundiertere und kontextualisierte Antworten liefern kann – eine entscheidende Verbesserung für viele professionelle und industrielle Anwendungen.

In den kommenden Kapiteln werden wir uns vertieft mit den Komponenten und dem praktischen Einsatz von RAG beschäftigen und aufzeigen, wie man Retrieval-Augmented-Generation implementieren und sinnvoll im Unternehmen einsetzen kann.

3 Document Ingestion

Damit Retrieval-Augmented Generation umgesetzt werden kann, müssen wir zuallererst ausgewählte Daten vorbereiten und in ein Format bringen, und sie dann im richtigen Format dem LLM zur Verfügung stellen. Document Ingestion ist ein zentraler Schritt im Retrieval-Augmented Generation (RAG) Prozess, da er die Basis für den Zugriff auf externe Datenquellen bildet. Die Qualität der Antworten, die ein RAG-System liefern kann, hängt maßgeblich davon ab, wie sorgfältig der Schritt der Document Ingestion umgesetzt wird. In diesem Kapitel gehen wir darauf ein, wie Dokumente für die spätere Verwendung aufbereitet werden, welche Herausforderungen dabei zu bewältigen sind und warum das sogenannte „Chunking" eine entscheidende Rolle spielt.

3.1 Was ist Document Ingestion?

Document Ingestion ist der Prozess des Sammelns, Aufbereitens und Speicherns von Dokumenten, sodass sie für ein Retrieval-System verfügbar gemacht werden können. Diese Dokumente können sehr unterschiedliche Formate und Inhalte haben, darunter PDFs, Webseiten, Datenbankeinträge, technische Dokumentation, Forschungsberichte oder FAQs. Das Ziel der Document Ingestion ist es, diese verschiedenen Informationsquellen in eine strukturierte, durchsuchbare Form zu bringen, die das Retrieval-System effizient und präzise durchsuchen kann.

3.2 Warum ist Document Ingestion so wichtig?

Die Qualität der Antworten, die ein RAG-System generiert, hängt zu einem erheblichen Teil davon ab, wie gut die zugrunde liegenden Dokumente aufbereitet und strukturiert sind. Eine schlecht durchgeführte Vorbereitung der Dokumente führt dazu, dass wichtige Informationen fehlen oder das Retrieval-System schwer relevante Inhalte findet. Daher muss die Document Ingestion sorgfältig geplant und an die Art und Struktur der vorhandenen Dokumente angepasst werden.

Beispiel: Um Fragen möglichst präzise zu beantworten, teilen wir ein Buch je nach Informationsverteilung unterschiedlich auf: Entweder seitenweise, wenn die relevanten Inhalte kompakt sind, oder kapitelweise, wenn sich Informationen über mehrere Seiten erstrecken.

Eine zentrale Herausforderung ist dabei die Heterogenität der Dokumente. Verschiedene Dokumentarten können unterschiedliche Strukturen und Inhalte aufweisen. Ein wissenschaftlicher Artikel ist beispielsweise in Absätze, Überschriften und Zitate gegliedert, während eine technische Dokumentation möglicherweise aus Tabellen, Code-Snippets und ausführlichen Schritt-für-Schritt-Anleitungen besteht. Eine pauschale Verarbeitung aller Dokumente nach dem gleichen Schema wäre daher nicht zielführend. Hier kommt das „Chunking" ins Spiel.

3.3 Chunking: Die richtige Granularität für den Erfolg

Der Begriff „Chunking" bezieht sich auf den Prozess, bei dem Dokumente in kleinere, inhaltlich kohärente Abschnitte (Chunks) zerlegt werden. Die Größe und Struktur dieser Chunks ist von entscheidender Bedeutung, da sie die Einheit bilden, auf die das Retrieval-System später zugreift. Die Herausforderung besteht darin, die optimale Granularität der Chunks zu finden, die dem Retrieval-System genug Kontext liefert, ohne zu große Informationsblöcke zu durchsuchen.

- **Warum ist Chunking so wichtig?** Die Genauigkeit der späteren Antworten hängt stark davon ab, wie relevant und präzise die Chunks sind, die bei einer Anfrage zurückgegeben werden. Zerlegen wir die Dokumente in zu große Chunks, besteht die Gefahr, dass irrelevante Informationen mitgeliefert werden. Das verwässert die generierte Antwort. Zerlegen wir sie in zu kleine Chunks, fehlt der nötige Kontext für eine präzise Antwort.

- **Anpassung an die Dokumentstruktur:** Ein entscheidender Aspekt beim Chunking ist die Anpassung an die Struktur des jeweiligen Dokumententyps. Bei einem wissenschaftlichen Artikel könnten Chunks beispielsweise einzelne Absätze oder thematische Sektionen sein. Bei einer technischen Dokumentation könnten Chunks die einzelnen Schritte einer Anleitung

oder die Beschreibungen spezifischer Funktionen sein. Das bedeutet, dass es keinen universellen „One-size-fits-all"-Ansatz für das Chunking gibt. Stattdessen müssen wir es individuell an den Dokumententyp und die geplanten Anwendungsfälle anpassen.

- **Dynamisches Chunking:** In einigen Fällen kann es sinnvoll sein, ein dynamisches Chunking zu implementieren, bei dem die Granularität der Chunks je nach Kontext und Aufgabe variiert. Hierbei kann eine initiale Klassifikation der Dokumenttypen und -strukturen helfen, um die Document Ingestion automatisch zu steuern. Ein solches flexibles Vorgehen erhöht die Präzision des Retrievals, indem es den Kontext besser berücksichtigt. Beispielsweise könnte die gleiche Ingestion-Pipeline ein PDF in Seiten und HTML-Dateien in Absätze zerschneiden. Die Chunks aus verschiedenen Dokumenten werden also auf spezifische Weisen verarbeitet, damit sie im Produktivbetrieb bereit liegen.

3.4 Die richtige Strukturierung: Mehr als nur Text

Ein weiterer kritischer Aspekt der Document Ingestion ist die richtige Strukturierung der Inhalte. Nicht alle Informationen in einem Dokument sind gleich wichtig oder gleich relevant. Überschriften, Stichpunkte, Tabellen und Hervorhebungen liefern oft wichtige Hinweise auf den inhaltlichen Schwerpunkt eines Chunks. Daher ist es wichtig, während der Document Ingestion Metadaten und strukturelle Informationen zu extrahieren und zu erhalten.

- **Metadaten:** Informationen wie Dokumenttitel, Autor:in, Erstellungsdatum, Seitenangabe aus dem Originaldokument und Schlüsselwörter liefern dem Retrieval-System wertvolle Hinweise, welche Chunks für eine bestimmte Anfrage besonders relevant sind.

- **Inhaltsindexierung:** Neben der Zerlegung in Chunks ist die Indexierung der Inhalte ein essenzieller Schritt. Dabei wird jedem Chunk eine eindeutige Identifikation sowie relevante Schlagwörter und Kontexteigenschaften zugeordnet. Dieser Index ermöglicht dem Retrieval-System eine schnelle und zielgenaue

Suche nach den relevantesten Inhalten. Beispielsweise eine URI, oder eine Seitenzahl.

3.5 Herausforderungen und Best Practices

- **Heterogene Datenquellen:** Eine der größten Herausforderungen ist der Umgang mit einer Vielzahl unterschiedlicher Dokumenttypen und -formate. PDF-Dateien, HTML-Seiten, CSV-Dateien, Datenbankeinträge – all diese Formate erfordern unterschiedliche Verarbeitungsstrategien. Daher sollten im Document-Ingestion-Prozess geeignete Konvertierungstools und Parser zum Einsatz kommen, die die Daten in eine einheitliche, durchsuchbare Form bringen, oder passende Datenbanken verwendet werden, um die verschiedenen Formate bestmöglich durchsuchbar zu gestalten.

- **Qualitätskontrolle:** Da die Qualität der Chunks direkt die Qualität der generierten Antworten beeinflusst, ist eine regelmäßige Überprüfung der Document Ingestion unerlässlich. Es ist empfehlenswert, einen automatisierten Prozess zur Validierung und Qualitätskontrolle einzurichten, der sicherstellt, dass die Document Ingestion konsistent und korrekt arbeitet.

- **Langfristige Wartung:** Document Ingestion ist kein einmaliger Prozess. Da sich Informationen ständig ändern und neue Dokumente hinzukommen, muss sie kontinuierlich überwacht und angepasst werden. Automatisierte Aktualisierungs- und Überwachungsmechanismen stellen sicher, dass die Datenbank immer auf dem neuesten Stand ist und somit das RAG-System stets auf aktuelle, relevante Informationen zugreifen kann.

3.6 Fazit: Die Grundlage für qualitativ hochwertige Antworten

Die Document Ingestion ist ein grundlegender und komplexer Schritt im RAG-Prozess, der einen entscheidenden Einfluss auf die Qualität der Antworten hat.

Durch die sorgfältige Strukturierung und Aufbereitung von Dokumenten, angepasst an deren individuelle Beschaffenheit, wird die Grundlage für ein effizientes und präzises Retrieval gelegt. Insbesondere das Chunking ist ein kritischer Faktor: Es beeinflusst, wie viel Kontext dem Retrieval-System zur Verfügung steht und wie zielgenau die Informationen extrahiert werden können.

Eine gut durchdachte Document Ingestion sorgt dafür, dass das RAG-System aus einer breiten und vielfältigen Informationsbasis schöpfen kann und so zuverlässige, präzise und kontextualisierte Antworten liefert. Die Qualität der Document Ingestion ist daher ein zentraler Baustein für den Erfolg einer RAG-Architektur und sollte mit höchster Sorgfalt und Expertise umgesetzt werden.

4 Retrieval

4.1 Die Rolle des Retrieval

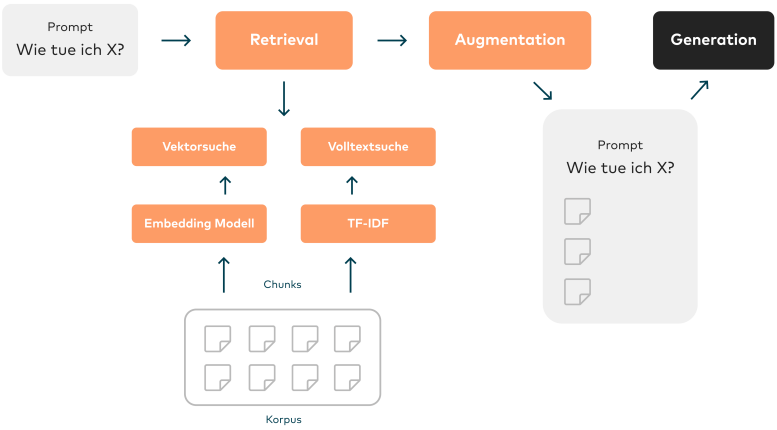
Der Schritt *Retrieval* hat die Aufgabe, zu einem Prompt die relevanten Chunks zu finden. Die Chunks wurden im Ingestion-Schritt erzeugt und sind in einer Datenbank indiziert.

- Eingabe: Prompt-Text
- Ausgabe: Liste von Chunks (Text-Abschnitte)

Das Retrieval muss einen guten Kompromiss zwischen Gründlichkeit (Tiefe), Laufzeit und Kosten ergeben. Es nutzt in einer interaktiven Anwendung nichts, das LLM mit 100 Chunks à 1.000 Tokens zu füttern, wenn man danach eine Minute warten muss. Man kann sich hier nicht beliebig viel Aufwand leisten.

Die Retrieval-Techniken entwickeln sich wie alles im LLM-Raum weiter. Man muss hier unterscheiden zwischen *Vorschlägen* und *realen Lösungen*. Wir behandeln hier Techniken, die in der Praxis funktionieren und leicht und kostengünstig implementierbar sind.

Retrieval ist eigentlich keine neue Aufgabe. Gesucht wurde schon immer. Neu im Werkzeugkasten ist die Vektorsuche, die auf Konzepten der Language Models aufbaut.



4.1.1 Prompt versus Query

Im Kontext vom Retrieval sprechen wir von der Query, nicht vom Prompt. Im einfachsten Fall sind beide identisch, aber das reicht für ein gutes Retrieval nicht aus.

4.2 Vektor-Suche

Fast alle Fachartikel zu RAG behandeln die Vektorsuche. Diese wird auch *Semantische Suche* genannt.

Aus der Query wird ein *Embedding-Vektor* berechnet und einem Vektor-Index als Suchwert übergeben. Der Index berechnet die (semantische) Ähnlichkeit des Suchvektors zu den Embeddings der Chunks und liefert die besten Treffer. Als Ähnlichkeitsmaß kann z.B. das *Dot-Product* oder die *Cosinus-Ähnlichkeit* verwendet werden.

Das Maß der Ähnlichkeit ist nicht entscheidend. Wichtig ist, dass der Index eine nach Ähnlichkeit sortierte Liste der Chunks liefert.

Wir nennen die Ähnlichkeit auch *Score*. Ein hoher Score bedeutet potentiell hohe Relevanz.

4.2.1 Eigenschaften

Eine Vektorsuche findet immer etwas. Sie funktioniert anders als eine Volltextsuche oder eine klassische Datenbankabfrage.

Bei der Cosinus-Ähnlichkeit sind folgende Extremwerte theoretisch interessant:

- -1: Suchvektor und Vergleichsvektor sind exakt das Gegenteil voneinander.
- 0: Suchvektor und Vergleichsvektor sind völlig unabhängig, haben nichts gemeinsam.
- +1: Suchvektor ist exakt identisch zum Vergleichsvektor.

In der Praxis mit Dokumenten-Chunks stellt sich heraus, dass Werte zwischen ungefähr 0,5 und 0,87 erscheinen. Selbst die abstruseste Frage, die scheinbar nichts mit dem Thema zu tun hat, landet etwa in diesem Bereich.

4.2.2 Irrtum: Viel hilft viel

Man könnte jetzt versuchen, bei der Vektorsuche unbedingt hohe Scores zu erzielen um vermeintlich Chunks mit höherer Relevanz zu finden. Eine Methode ist *Query Rewriting*, indem ein LLM aus einem dünnen Prompt eine kleine Geschichte aus mehreren Sätzen erzeugt und dadurch die Query „anfettet".

Das schraubt natürlich die Scores hoch, weil mehr Material in der Query steckt, was zu Inhalten passen kann. Es liefert aber deshalb nicht relevantere Chunks. Das Gegenteil kann passieren. Durch das Anfetten kommen Begriffe und Dinge bei der Query hinzu, die in der ursprünglichen Frage gar nicht vorhanden sind. Sind diese überhaupt relevant? Außerdem denkt sich das LLM jedes Mal eine leicht andere Geschichte aus und das Ergebnis der Suche wird dadurch bei identischer Eingabe sehr instabil.

Wir verwenden deshalb das **unveränderte Prompt als Query**. Ein *Query Rewriting* kann Sinn machen, wenn die eigene Anwendung über relevanten Kontext (bspw. zur aktuellen Nutzerin) verfügt.

Die Höhe der Scores ist unwichtig. Es kommt auf die Reihenfolge der Relevanz der Treffer an. Der beste Chunk muss vorne sein, der zweitbeste danach, etc. Es ist egal, ob 0,87 oder 0,73 der beste Score ist.

4.2.3 Irrtum: Man braucht ein Ähnlichkeitsminimum

Es scheint logisch zu sein, die Anzahl der Ergebnisse anhand einer Mindesschwelle für die Ähnlichkeit zu begrenzen. Das ist falsch. Wie oben erläutert, kommt es nicht auf den absoluten Wert an. Für die eine Abfrage wäre z.B. 0,75 ideal, für eine andere 0,80. Es gibt keine allgemeine Schwelle. **Es ist besser, einfach die Anzahl zu begrenzen**.

4.2.4 Irrtum: Es dreht sich alles um die Vektorsuche

Die Vektorsuche ist als Ergänzung sehr nützlich, besonders bei allgemeinen Fragen, sie reicht aber nicht aus. Sie muss unterstützt werden durch eine Volltextsu-

che. In dieser Deutlichkeit wird das in der Mehrzahl der Fachliteratur nicht gesagt. Das leitet uns direkt über zum nächsten Thema.

4.3 Hybride Suche

Die Vektorsuche muss durch eine Volltextsuche unterstützt werden. Es ist sogar so, dass die Volltextsuche *die führende Suche* ist!

Die beiden Suchen ergänzen sich. Die Volltextsuche hat ihre Stärken besonders bei spezifischen Fragen. Sie gleicht eine eventuelle Unschärfe bei den Embeddings aus.

Volltextsuche ist eine ausgereifte Technik und braucht hier nicht näher erklärt zu werden. Die übliche Technik nennt sich **TF-IDF** in Kombination mit **BM25** und wird z. B. vom Indexer Lucene implementiert. Analog zur Vektorsuche gibt es einen Volltext-Index, der zu einem Suchtext ein Scoring durchführt und die besten Treffer liefert. Im Gegensatz zur Vektorsuche kann eine Textsuche komplett leer ausgehen.

4.3.1 Mehrstufige Text-Suchstrategie

Wir verwenden eine mehrstufige Suchstrategie, die zusätzlich Details und alternative Begriffe aus einem Prompt herauszieht, um daraus mehrere Queries zu formen.

Stufe 1: Prompt = Query

Im ersten Schritt wird das Prompt unverändert dem Suchindex übergeben.

Stufe 2: Keywords, Compounds, Synonyme

- Keyword-Extraktion. Die Keywords werden vom LLM aus dem Prompt extrahiert. Dem LLM wird der fachliche Kontext mitgeteilt, damit es relevante Keywords erkennen kann.

- Compound words. Zusammengesetzte Wörter werden vom LLM in seine Einzelteile zerlegt. Aus „Gerichtsurteil“ wird zum Beispiel „Gericht“ und „Urteil“.
- Synonyme. Über alle soweit gewonnenen Wörter werden vom LLM Synonmye generiert.

Die so eingesammelten Keywords werden an den Suchindex gegeben.

4.3.2 Volltextsuche führend

Wir verzichten auf eine nachfolgende Vektorsuche, wenn die Textsuche zu wenige Ergebnisse fand. Die Chance, durch semantische Suche relevante Chunks zu finden, ist dann praktisch null.

4.3.3 Rank Fusion

Die zweistufige Textsuche und die Vektorsuche ergeben zusammen drei Listen von Chunks. Das Scoring des Textindex passt nicht zur Vektorsuche. Wir können nicht einfach alles zusammenwerfen und sortieren. Das Verfahren der *Reciprocal Rank Fusion* kommt zum Einsatz.

Die genaue Formel kann leicht nachgesehen werden. Das Prinzip ist, dass nur noch die Reihenfolge, der *Rank*, in einer Liste zählt. Der Top-Treffer hat den Rank 1 in seiner Liste. Das Zusammenmischen funktioniert so, dass pro Auftreten eines Chunks in einer Liste der Gesamt-Rank erhöht wird. Chunks, die in mehreren Listen vorkommen, werden sozusagen „nach oben gespült“.

4.3.4 Top Chunks

Nach der Rank Fusion begrenzen wir die Anzahl der Chunks. Die optimale Größe dieser Begrenzung muss in der Praxis ermittelt werden. Unsere Experimente haben gezeigt, dass maximal 30 Chunks sinnvoll sind. Eine größere Menge führte nicht zu besseren Ergebnissen.

4.4 Overlaps bzw. Windowing

In Experimenten stellte sich heraus, dass um einen Treffer herum das LLM gerne mal auf den Chunk vor oder nach dem Treffer zugreift. Nicht selten sogar lieber als auf den Treffer selbst. Der Treffer sticht sozusagen in ein Nest.

Wir packen nach dem Ranking einen Chunk davor und zwei Chunks nach dem Treffer zusätzlich in die Liste. Wir wenden einen Filter an, sodass nur Chunks mit einer Cosinus-Ähnlichkeit zum Treffer von > 0,82 infrage kommen. Dieser Wert muss experimentell ermittelt werden. Außerdem werden nur die besten 10 Chunks dafür verwendet.

4.5 Limitierung

Wir senden maximal ca. 25.000 Tokens an das LLM für interaktive Anwendungen. Das entspricht ungefähr 40 Seiten eines dicht beschriebenen PDF-Dokuments. Bei der OpenAI API mit GPT-4o (Okt. 2024) ergibt das eine Laufzeit von etwa 15 Sekunden bei langen Antworten.

4.6 Das LLM hat das letzte Wort

In der Praxis stellt sich heraus, dass das LLM im Generation-Schritt, also dort wo die Chunks zusammen mit dem Prompt verarbeitet werden, ganz eigene „Vorstellungen“ davon hat, was relevant ist und was nicht.

Nach unserer Erfahrung sind die Top 10 allgemein sehr beliebt, aber danach wird es wild. Beispielsweise werden Chunks an der 20sten Stelle überraschend herausgepickt. Die Verteilung sieht bei jedem Prompt völlig anders aus.

Auch wird mal ein Chunk bevorzugt, der von der Vektorsuche einen hohen Score bekam, ein anderes Mal ist die Textsuche relevanter. **Besonders diese Beobachtung hat uns zu der Strategie geleitet, besser komplementäre Suchstrategien zu mischen anstatt viel Aufwand in die Vektorsuche zu stecken**.

4.7 Allgemeine Grenzen des Retrieval

Das Retrieval kann nur einen Ausschnitt der Chunks liefern. Es kann keine breiten, vollständigen Suchen leisten, so wie man es von Datenbankqueries erwartet.

Fragen wie „finde alle…“ oder „was ist die Zusammenfassung von…“ sind zwangsläufig nur unvollständig zu beantworten. Mit einer höheren Anzahl von Chunks kann man die Treffer bei breiten Fragen ausweiten, was aber bei anderen, spezifischen Fragen wieder unnötig Ballast erzeugen würde.

Wie haben auch versucht, die Fragen zu qualifizieren, ob sie „breit“ oder „schmal“ sind. Das gelang nicht zuverlässig. Außerdem verschiebt man ggf. nur das Limit und bekommt trotzdem keine vollständige Antwort.

Das Retrieval kann keine Aggregation leisten.

4.8 Einfaches RAG ist nur ein Zwischenschritt

Wir sehen bei dem Retrieval deutlich, dass es ineffizient ist. Dem LLM wird per *Force Feeding* Material gesendet, von dem wir hoffen, dass es relevant ist.

Die nächste Entwicklungsstufe ist, das LLM selbst entscheiden zu lassen, ob es ein Retrieval braucht oder nicht. Wir sprechen dann von *Agentic Workflow*. Dafür brauchen wir *Function Calling*, mit dem theoretisch das Retrieval dynamisch und on-demand werden könnte. Ob das wirklich zuverlässig funktioniert, wissen wir noch nicht.

Ein LLM weiß nicht, was es nicht weiß. Die „Zwangsfütterung“ umgeht dieses Problem auf Kosten der Effizienz.

4.9 Contextual Retrieval

Von Anthropic kommt die Technik *Contextual Retrieval*[1]. Hier werden alle Chunks bei der Ingestion einzeln mit zusätzlichem Kontext „angefettet“. Die Sucharbeit

[1] https://www.anthropic.com/news/contextual-retrieval

wird dann verdoppelt, indem der Zusatzkontext genauso verwendet wird wie die Chunks. Das Verfahren arbeitet ansonsten genau wie oben beschrieben.

4.9.1 Ingestion

Ein LLM bekommt das gesamte Dokument und den Chunk als Kontext. Das Prompt weist an, eine Einordnung des Chunk-Inhalts im Kontext des Dokuments zu schreiben. Das Ergebnis geht zusammen mit dem Chunk in das Embedding. Anthropic verwendet die Phrase

```
Please give a short succinct context to situate this chunk
within the overall document for the purposes of improving search
retrieval of the chunk.
```

(Quelle: Anthropic Cookbook[2] zu Contextual Retrieval)

Der Zusatzkontext verweist auf den Chunk und wird ansonsten wie der Chunk abgespeichert und indiziert.

4.9.2 Retrieval

Das Retrieval wird auf die gleiche Weise sowohl in den Chunks, als auch in den Zusatzkontexten durchgeführt. Anthropic verwendet ebenso eine Hybride Suche mit Volltext (BM25) und Vektor. Nach der Rank Fusion und der nachfolgenden Top-Auswahl folgt optional ein Reranking-Model, um die Relevanz der Chunks nochmal zu ordnen.

4.9.3 Was man bedenken muss

- Bei der Ingestion wird das gesamte Dokument und der Chunk in den Kontext gestellt. Das kann je nach Model und Material problematisch werden.
- Embeddingmodelle haben kleine Kontextgrößen. Das Embedding verarbeitet den Chunk + Extrakontext. Wenn man das Chunken von PDFs nach Seiten macht, kann es bereits knapp werden bei sehr vollen Seiten. Der Extrakontext kann die maximale Kontextgröße sprengen.

[2] https://github.com/anthropics/anthropic-cookbook/tree/main/skills/contextual-embeddings

- Der extra Zeitaufwand kann erheblich sein für die Generierung des Zusatzkontext.

5 Augmentation

5.1 Die Rolle der Augmentation

Der Begriff bedeutet *Ergänzung*. Der Kontext des LLM wird um die gefundenen Chunks aus dem Retrieval-Schritt ergänzt. Hier wird die gesamte Message-Liste präpariert, die dem LLM übergeben wird.

Eingabe:

- Liste von Chunks
- System Message
- Prompt
- Ggf. Liste der Messages aus einem vorangehenden Dialog

Ausgabe:

- Liste der Messages an das LLM (der Kontext) bestehend aus System Message, Chunk-Texten, bisheriger Dialog, letztes Prompt

5.2 Chunks einbauen

Es gibt unterschiedliche Empfehlungen, wo und wie die Chunks eingebaut werden sollen. Wir haben bei OpenAI-Modellen gute Erfahrungen damit gemacht, sie in die *System Message* zu stellen. Bei Anthropic wird empfohlen, die erste *User Message* zu verwenden. Anthropic schlägt allgemein präzisere Formate vor als OpenAI. Im Zweifel hält man sich an diese.

Die Chunks bauen wir wie folgt ein:

```
<data>
[document_id:Musterdokument.pdf]
[pagenumber:123]
…weitere Metadaten…
```

```
Hier beginnt der Text des Chunks.
</data>
```

Die XML-Tags funktionieren genauso gut wie triple backticks “‘. Die Metadaten können in Anweisungen in der System Message verwendet werden, um zum Beispiel präzise Referenzen anzuhängen. Diese Aufgabe ist tatsächlich schwieriger als sie klingt.

5.3 Dokument-Referenzen

Für unseren Use Case war es notwendig, die genaue Seitennummer für eine Referenz geben zu können. Deshalb basierte unser Chunking auf den Seiten der PDFs. Die sonst üblichen Chunking-Methoden, die auf die Seite keine Rücksicht nehmen, funktionierten für uns nicht.

Eine zuverlässige Anweisung in der System Message war nicht einfach zu finden. Kleinste Änderungen in der Formulierung hatten große Wirkung. Wir sind bei der folgenden Form gelandet:

```
WICHTIG: Verwende überall Referenzen auf die Quelle
im Format [id:document_id,page:pagenumber].
Schreibe die Referenzen direkt hinter die Aussage,
in der die Quelle verwendet wird.
Jeder Absatz muss mindestens eine Referenz enthalten.
Jede Aussage muss eine Referenz enthalten.
Gib document_id immer mit Dateiextension an.
Verwende den Originaltext der document_id.
Lasse ae, oe, ue unangetastet.
```

Mit GPT-4o funktioniert diese Form absolut zuverlässig. GPT-4o mini vergisst gern bei größeren Chunk-Listen die Aufgabe.

5.4 Ohne Referenzen ist alles Blindflug

Woher soll man wissen, ob die Chunks für das LLM am Ende relevant waren, wenn man keine Referenzen generieren lässt? Dieses Feedback ist essenziell! Damit können wir unser Retrieval fein abstimmen. Ohne Referenzen ist alles ein Blindflug. Es hilft auch nichts, sich die Arbeit zu machen, die Chunks selber zu beurteilen. Im Kapitel zu Retrieval haben wir erwähnt, dass das LLM das letzte Wort hat. **Die Vorstellung, was relevant ist und was nicht, kann nicht intuitiv nachvollzogen werden**.

5.5 Prompt Rewriting

Angenommen, jemand gibt nur ein Stichwort ein, wie z. B. „Barwert“. Was soll das LLM damit anfangen? Wir geben dem LLM zuerst die Aufgabe, zu prüfen, ob das Prompt ein vollständiger Satz ist. Falls nicht, lassen wir das LLM das Prompt umschreiben in einen Fragesatz, der unseren fachlichen Kontext enthält. Dieses Prompt wird den Anwendern nicht gezeigt und verdeckt verwendet.

5.6 Chunks behalten oder nicht?

Soll man die Chunks behalten, welche beim letzten Retrieval aufgesammelt wurden? Wir denken nein. Die Gründe sind folgende:

- Der Dialog enthält die Antwort des LLMs auf die Prompts. Man kann auf vorherige Inhalte Bezug nehmen, ohne die zugrundeliegenden Chunks.
- Das Management der Kontextgröße wird kompliziert, wenn man die Chunks nicht wegwirft. Welche soll man behalten? Was ist, wenn die Anwenderin das Thema wechselt?
- Jeder neue Durchlauf wird langsamer und teurer.
- Das LLM ignoriert die Chunks aus dem Retrieval auch gerne mal völlig, wenn das Prompt sich auf den Dialog bezieht. Zum Beispiel mit dem Prompt „Mache mir daraus eine Gliederung für PowerPoint-Folien“.

Es ist, wie so oft, nicht zu empfehlen, zu erraten, wie das LLM arbeitet. Man liegt zu oft falsch.

6 Generation

Generation ist die dritte Stufe in einer RAG-Architektur und trägt dazu bei, die Grenzen der traditionellen natürlichen Sprachverarbeitung (NLP) zu erweitern. Während der Retrieval-Teil dazu dient, relevante Informationen aus einer oder mehreren Quellen zu extrahieren, übernimmt die Generation die anspruchsvolle Aufgabe, diese Informationen in kohärente, flüssige und kontextuell passende Textausgaben zu transformieren.

Grundprinzipien der Generierung

Die Generierung in einem RAG-System nutzt ein LLM mit Transformer-Architektur. Diese Transformer wurden speziell optimiert, um menschenähnliche Texte zu erstellen. Das bekannteste Modell dieser Art ist der GPT (Generative Pre-trained Transformer), das durch ein umfangreiches Vortraining auf großen Textkorpora gelernt hat, sowohl Syntax als auch Semantik natürlicher Sprache wiederzugeben. Die Stärke eines solchen Modells liegt in seiner Fähigkeit, bestehende Wissenslücken mit plausiblen Informationen zu füllen – ein Prozess, der durch fein abgestimmte Mechanismen der Sprachmodellierung ermöglicht wird.

Prozess der Textgenerierung

Der Generierungsprozess beginnt mit einem Input, der, im RAG-Fall, aus dem Retrieval-Part des RAG-Systems stammt und im Augementation-Part mit einem System Prompt zu einem finalen Prompt zusammengesetzt wurde. Dieser Input wird also im Allgemeinen *Prompt* genannt. Es gibt dabei verschiedene Arten von Prompts, wie System und User Prompts, die vom LLM unterschiedlich berücksichtigt werden. Zur besseren Unterscheidung werden wir sie mit ihrem alternativen Namen, Messages, bezeichnen. Ein Prompt besteht also aus verschiedenen Messages.

Bei RAG setzt sich der Prompt aus zwei Teilen zusammen: Erstens aus Textstellen aus dem Retrieval als Kontext. Zweitens aus der System Message mit Anweisungen an das Modell. Diese legt zum Beispiel fest, dass das LLM nur Informationen aus den gegebenen Textstellen verwenden darf.

Das LLM nutzt dann diesen Prompt, um einen Anfangszustand für die Textproduktion zu definieren. Oft wurden LLMs wie GPT-4o bereits genau auf diese User<->LLM Interaktion trainiert, sodass sie wissen, wie verschiedene Aussagen zu gewichten sind.

Von diesem Anfangszustand aus startet die Textgenerierung. Das System erzeugt Schritt für Schritt Wort für Wort (genauer: Token für Token). Dabei berücksichtigt es immer den bisher erzeugten Text, um einen verständlichen und zusammenhängenden Text zu erstellen.

Herausforderungen und Optimierung

Ein zentrales Anliegen bei der Generierung ist die Sicherstellung von Qualität und Relevanz des Ergebnisses. Dafür sind bestimmte Techniken sehr hilfreich, wie die *Beam Search*. Diese betracht mehrere mögliche Fortsetzungen eines Texts. Das *Top-k Sampling*, wählt nur die wahrscheinlichsten Wörter für die nächste Stelle im Text. Eine bekanntere Technik ist das *Temperature-Adjustment*, mit dem man steuern kann, wie kreativ oder sicher die Wortwahl sein soll. Diese Methoden helfen, die Balance zwischen Kreativität und Relevanz der generierten Texte zu finden, indem sie die Auswahl an möglichen Wörtern klug einschränken und so unpassende oder uninteressante Ergebnisse vermeiden.

Wichtig zu beachten ist: Ein LLM kann zwar so konfiguriert werden, dass es bestimmte unerwünschte Ausgaben vermeidet. Eine absolute Garantie dafür gibt es jedoch nicht - das Modell könnte trotz aller Vorsichtsmaßnahmen solche Inhalte generieren.

Solche Garantien können aber durch sogenannte *Guardrails* um das LLM herum gebaut werden. Dabei gibt es Input- und Output-Guardrails, die das LLM vor der Nutzer und den Nutzer vor dem LLM schützen.

Integration ins Business-Umfeld

Im Business-Kontext ist besonders die Anpassbarkeit der *Generation* an spezifische Geschäftsbedürfnisse von Interesse. RAG stellt hierfür eine gute Option dar. Eine andere Option bietet Fine Tuning. Durch ein gezieltes Fine Tuning lässt sich das Modell anpassen, um z. B. branchenspezifische Terminologie korrekt zu verwenden. Dieser Prozess bietet erhebliche Vorteile in Anwendungsfällen

wie automatisierten Kundenservice-Systemen, Marketing-Content-Generierung oder personalisierten Benutzerinteraktionen. Jedoch ist beim Fine Tuning etwas vorsicht geboten, da das Modell hierbei seine Gewichte an das Gelernte anpasst. Diese Anpassungen sind nicht zückgängig zu machen, außer durch ein erneutes Fine Tuning. Ein häufiges Fine Tuning kann dabei einen beträchtlichen Kostenfaktor darstellen, sowohl einmalig als auch regelmäßig. In einem Umfeld, in dem sich die Datengrundlage häufig ändert, ist daher zu einem RAG-System zu raten. Beide Optionen schließen sich nicht gegenseitig aus, sondern können bei Bedarf auch kombiniert werden.

7 Anwendungsszenarien

Kundensupport

Im Bereich des Kundensupports stellen RAG-Systeme einen transformativen Ansatz dar, indem sie traditionelle Chatbots mit der Fähigkeit verbessern, kontextuell relevante und gut strukturierte Antworten abzurufen und zu generieren. Zum Beispiel kann in der Automobilindustrie ein RAG-System auf umfangreiches Werkstattwissen und technische Handbücher zugreifen. Wenn ein Kunde nach spezifischen Problemlösungen oder Wartungsarbeiten fragt, kann das System relevante Daten aus seinem Repository abrufen. In Kombination mit einem LLM führt das zu maßgeschneiderten Antworten, die effektiv auf die Supportanliegen eingehen. Diese Fähigkeit verbessert erheblich die Wirksamkeit von Kundeninteraktionen, die Zufriedenheit und verkürzt gleichzeitig die Lösungszeiten.

Im Gegensatz zu herkömmlichen regelbasierten Systemen, die oft bei der Behandlung nuancierter Fragen oder komplexer Abfragen versagen, passen sich RAG-gestützte Lösungen an, indem sie große Mengen an Informationen abrufen und in kompakter Weise präsentieren. Diese Anpassungsfähigkeit ist in Bereichen wie E-Commerce oder technischen Support entscheidend, wo Produkte und Dienstleistungen schnell weiterentwickelt werden und die Wissensdatenbank ständig aktualisiert werden muss, um relevant zu bleiben.

Dokumentenverarbeitung in der Logistik

Logistikunternehmen bearbeiten häufig enorme Mengen an Dokumenten wie Rechnungen, Versanddokumente und Compliance-Formulare. LLMs und RAG-Systeme haben das Potenzial, den Workflow der Dokumentenverarbeitung erheblich zu verbessern, indem sie die Extraktion und/oder Interpretation relevanter Daten automatisieren. So können diese Systeme schnell verschiedene Dokumentationsquellen analysieren, um wichtige Versanddetails zu identifizieren, Datenverarbeitungsprozesse zu optimieren und die Genauigkeit logistischer Operationen zu verbessern. Durch die Erhöhung der Verarbeitungsgeschwindigkeit wird der manuelle Aufwand reduziert und das Risiko menschlicher Fehler minimiert, was den Weg für eine effizientere Lieferkettenverwaltung ebnet.

Medizinische Anwendungen

Im Gesundheitswesen sind Informationen der Schlüssel für zeitnahe und präzise Entscheidungen. RAG kann hier eine entscheidende Rolle spielen, indem es Ärzt:innen und Pflegepersonal den Zugang zu aktuellen Informationen wie Wechselwirkungen von Medikamenten oder klinischen Forschungsergebnissen ermöglicht. Wenn medizinisches Fachpersonal z. B. nach potenziellen Medikamenteninteraktionen fragt, kann ein RAG-System sofort präzise und umfassende, wissenschaftlich fundierte Daten aus den neuesten medizinischen Forschungen und Datenbanken liefern. Dies unterstützt zeitnahe und fundierte Entscheidungen, verbessert das Patient:innenmanagement und sorgt für eine bessere Versorgung.

Überlegungen für eine Nicht-Empfehlung

Während RAG in verschiedenen Bereichen zahlreiche Vorteile bietet, ist es keine universelle Lösung. Die Abhängigkeit von abrufbaren Daten macht es ungeeignet für Umgebungen, die unmittelbare Echtzeit-Verarbeitung und Entscheidungsfindung erfordern. Beispielsweise erfordern Szenarien wie die Flugverkehrskontrolle oder der Finanzhandel schnelle Reaktionen, die auf kontinuierlichen, minutengenauen Daten und komplexen Entscheidungsfähigkeiten beruhen, die RAG nicht bereitstellen kann.

Darüber hinaus hängt die Effektivität von RAG von der Qualität und dem Umfang der zugrunde liegenden Datenbanken ab. In Kontexten, die hohe interpretative Fähigkeiten oder kreative Entscheidungsfindung erfordern, wie z. B. im strategischen Management oder bei abstrakten künstlerischen Unternehmungen, könnte RAG möglicherweise nicht ausreichen. In hochsensiblen Umgebungen mit strengen Datenschutzanforderungen ist Vorsicht geboten, da die Datenabrufkomponenten potenzielle Sicherheitsrisiken darstellen können.

Zusammenfassend überzeugen RAG-Systeme in Szenarien, die das Abrufen und Generieren von kontextuell genauen Informationen aus textbasierten Ressourcen erfordern. Ihre Integration in den Kundensupport, die Logistik und das Gesundheitswesen zeigt ihr Potenzial zur Transformation von datenintensiven Prozessen.

Bei der Implementierung ist es wichtig, jeden Anwendungsfall sorgfältig zu prüfen. Dabei sollten vor auch rechtliche Vorgaben zum Datenschutz und die Qualität

der verfügbaren Daten berücksichtigt werden. Ein gut geplanter Evaluierungsprozess in Form von Proof-of-Concepts kann helfen, festzustellen, ob RAG die beste Lösung ist oder ob einfachere Alternativen vorzuziehen sind. Durch gezielte Tests und Analysen kann dann sichergestellt werden, dass RAG-Systemexs dort eingesetzt werden, wo sie den größten Nutzen bringen.

8 Technische Herausforderungen bei der Implementierung

Die Implementierung von RAG bringt verschiedene technische Herausforderungen mit sich. Diese reichen von der effektiven Verwaltung großer Datenmengen über die Optimierung von Retrieval- und Generierungsprozessen bis hin zu Fragen der Skalierbarkeit und Datensicherheit. In diesem Kapitel werfen wir einen detaillierten Blick auf die wesentlichen technischen Hürden, die bei der Umsetzung eines RAG-Systems auftreten können, und geben Hinweise, wie diese Herausforderungen bewältigt werden können.

8.1 Aufbau und Betrieb von Vektordatenbanken

Eine zentrale Komponente im RAG-Ansatz ist das Retrieval-System, das mit Hilfe von Vektordatenbanken und relationalen Datenbanken implementiert werden kann. Vektordatenbanken sind speziell für die Speicherung und schnelle Abfrage von hochdimensionalen Vektoren ausgelegt, wie sie bei Embeddings entstehen. Dabei stellen sich folgende Herausforderungen:

- **Speicherbedarf**: Da Embeddings für jedes Dokument in der Datenbank erstellt werden, kann der Speicherbedarf rasch steigen. Die Auswahl einer Datenbank, die sich gut skalieren lässt und eine effiziente Speicherverwaltung bietet, ist daher essenziell.

- **Abfragegeschwindigkeit**: Kurze Antwortzeiten beim Retrieval sind entscheidend, um relevante Dokumente in Echtzeit zu finden. Techniken wie *Approximate Nearest Neighbor* (*ANN*) helfen, die Suche zu beschleunigen, allerdings auf Kosten der Genauigkeit. Statt exakter Treffer werden hierbei ausschließlich nächstgelegene Einträge aus der Vektordatenbank geladen.

- **Indexierung und Aktualisierung**: Dokumente werden kontinuierlich ergänzt und aktualisiert. Das bedeutet, dass die Dokumente in der Vektordatenbank

immer wieder ergänzt und aktualisiert werden müssen, um stets aktuelle Informationen zu liefern. Zusätzlich müssen veraltete Dokumente aus der Vektordatenbank entfernt werden.

8.2 Herausforderungen der Embedding-Erstellung

RAG setzt Embeddings ein, um den semantischen Inhalt von Dokumenten abzubilden und somit die Grundlage für den Retrieval-Prozess zu schaffen. Hier treten folgende Herausforderungen auf:

- **Qualität der Embeddings**: Unterschiedliche Modelle und API-Anbieter bieten verschiedene Embedding-Qualitäten. Die Wahl des richtigen Embedding-Modells ist entscheidend, um eine möglichst genaue Repräsentation der Dokumente zu erhalten.

- **Konsistenz**: Um relevante Dokumente für ein Prompt zu finden müssen die Dokumente in der Vektordatenbank und das Prompt mit dem gleichen Embedding-Modell verarbeitet werden. Sollte das Embedding-Modell gewechselt werden, müssen alle in der Datenbank befindlichen Einträge mit dem neuen Embedding-Modell erneut indiziert werden.

- **Performance und API-Kosten**: Die Erstellung von Embeddings über die APIs der Modellanbieter verursacht Kosten. Kleinere Embedding-Modelle kosten um ein Vielfaches weniger, sind allerdings auch deutlich schlechter für das Retrieval relevanter Dokumente geeignet. Hier muss evaluiert werden, mit welchem Modell die Kosten in Ordnung sind, ohne auf zu viel Qualität beim Retrieval zu verzichten.

8.3 Kombination von Retrieval und Generierung

Eine der größten Herausforderungen bei der Implementierung von RAG besteht darin, die abgerufenen Informationen effektiv in den Generation-Prozess einzubinden. Dies erfordert eine nahtlose Integration der beiden Module „Retrieval“ und „Generation“ hinsichtlich der folgenden Aspekte:

- **Kontextgröße des LLM**: Alle LLMs haben eine begrenzte Kontextlänge. Wenn die abgerufenen Dokumente oder Textfragmente diese Länge überschreiten, müssen relevante Inhalte ausgewählt oder zusammengefasst werden. Eine intelligente Auswahlstrategie ist hier entscheidend, um das Modell nicht mit irrelevanten Informationen zu überfrachten und die Kosten nicht ungewollt in die Höhe zu treiben.

- **Unwissenheit der Modelle**: Das LLM soll die erhaltenen Informationen verwenden, um eine Antwort zu generieren. Allerdings weiß das LLM nicht, ob die Informationen überhaupt dafür geeignet sind und welche zusätzlichen Informationen fehlen.

- **Kontrolle der Antwortqualität**: Die Qualität der generierten Antwort hängt stark von der Relevanz der abgerufenen Informationen ab. Um konsistente und verlässliche Antworten zu erhalten, sind oft Tests und Anpassungen der Retrieval-Parameter notwendig.

8.4 Skalierbarkeit und Leistungsoptimierung

Die Verarbeitung großer Datenmengen und die Integration von Echtzeitanfragen stellen erhebliche Herausforderungen an die Skalierbarkeit und Leistung eines RAG-Systems:

- **Durchsatz**: Bei stark frequentierten Anwendungen kann die Abfragefrequenz in der Vektordatenbank und die Generierungslast auf das LLM stark anstei-

gen. Lastverteilung und die Bereitstellung ausreichend skalierbarer Ressourcen sind essenziell, um eine gleichbleibende Antwortgeschwindigkeit zu gewährleisten. Die Modelle selbst zu betreiben ist hierbei schwierig, da für LLMs Spezialhardware benötigt wird.

- **Cache-Mechanismen**: Um Ressourcen zu sparen und die Geschwindigkeit zu erhöhen, kann ein Caching-System implementiert werden, das häufig angefragte oder ähnliche Antworten speichert. Dies reduziert die Notwendigkeit für wiederholte Abfragen und Generierungen.

- **Optimierung der API-Anfragen**: Die Nutzung von APIs wie der OpenAI API für Generierung und Embedding-Erstellung bringt eine Kostenstruktur mit sich, die durch optimierte Anfragen reduziert werden kann. Techniken wie Batch-Verarbeitung und Caching können hier ebenfalls zur Kostenreduktion beitragen.

8.5 Datenverwaltung und -sicherheit

RAG-Systeme greifen oft auf sensible oder geschützte Informationen zu, insbesondere bei Anwendungen in Unternehmen, die auf interne Datenquellen zugreifen. Hier treten verschiedene Herausforderungen im Bereich der Datensicherheit auf:

- **Zugriffskontrollen**: Es muss sichergestellt werden, dass nur autorisierte Benutzer:innen Zugriff auf das Retrieval-System und die verarbeiteten Daten haben. Eine fein abgestimmte Zugriffsverwaltung ist unerlässlich, um sensible Daten zu schützen.

- **Compliance und Auditierbarkeit**: Je nach Branche unterliegen Daten spezifischen Compliance-Vorgaben. Ein RAG-System muss sicherstellen, dass alle abgerufenen und generierten Daten den geltenden Vorschriften entsprechen und bei Bedarf auditiert werden können.

8.6 Bias und Qualitätskontrolle

Obwohl RAG eine Möglichkeit bietet, die Antworten von LLMs auf verifizierte Informationen zu stützen, bleibt die Qualitätskontrolle eine Herausforderung:

- **Bias in der Generierung**: Da LLMs auf statistischen Mustern basieren, können sie gesellschaftliche Voreingenommenheiten reproduzieren. RAG kann dies teilweise durch verifizierte externe Daten mindern, jedoch ist eine kontinuierliche Überwachung und Feinabstimmung notwendig.

- **Bewertung der Antwortqualität**: Die Qualität und Genauigkeit der generierten Antworten sind kritisch. Regelmäßige Tests, etwa durch Qualitätsmetriken oder menschliches Feedback, sind notwendig, um das System anzupassen und zu verbessern. Außerdem ist es hilfreich, die Qualität der Antworten für verschiedene Auslegungen des Systemprompts und Umformulierungen der Nutzeranfragen zu vergleichen. Diese Methodik nennt sich „Prompt Evaluation“.

8.7 Kosten- und Ressourcenmanagement

Die Implementierung eines RAG-Systems kann sowohl hohe Rechen- als auch Finanzressourcen beanspruchen. Folgende Aspekte sind wichtig, um die Kosten zu kontrollieren:

- **Effiziente Nutzung der Cloud-Ressourcen**: Viele RAG-Systeme nutzen Cloud-Services für das Management von Vektordatenbanken und die Generierung von Antworten. Eine bedarfsgerechte Bereitstellung und die Nutzung von Serverless-Ansätzen können helfen, die Betriebskosten zu senken.

- **Monitoring und Kostenkontrolle**: Ein umfassendes Monitoring ist entscheidend, um die Performance und die anfallenden Kosten kontinuierlich zu beobachten. Tools zur Überwachung der API-Nutzung und Optimierungsstrategien tragen zur Kostensenkung bei.

8.8 Erfolgsfaktoren und Handlungsempfehlungen

Die technische Implementierung eines RAG-Systems ist mit einigen Herausforderungen verbunden, die jedoch durch sorgfältige Planung und die Auswahl geeigneter Tools und Strategien bewältigt werden können. Die wichtigsten Erfolgsfaktoren sind:

- **Das passende Embedding-Modell** für schnelle und relevante Ergebnisse.
- **Know your Domain** um zu erkennen, welche Dokumente wirklich relevant sind und sie tatsächlich für passende Anfragen auch mit dem Retrieval widerzugeben.
- **Integration eines verlässlichen Sicherheitskonzepts** zum Schutz sensibler Daten.
- **Regelmäßige Qualitätskontrolle und Feinabstimmung**, um Bias zu minimieren und die Antwortqualität zu verbessern.

9 Unser Angebot

9.1 Entwicklung, Beratung und Betrieb

Generative KI ermöglicht Ihrem *Business*, zuvor unmögliche oder teure Features umzusetzen und manuelle Prozesse schneller zu digitalisieren.

Die *IT* ist Enabler fürs Business. Es lohnt sich, wiederkehrende und zeitraubende Tätigkeiten mit generativer KI zu erleichtern, um lieferfähig zu bleiben. Dabei bleibt zu jeder Zeit der Mensch „in the loop".

Wir unterstützen Sie gerne bei Ihren Entwicklungsvorhaben – von der frühen Erkundungsphase, in der Produktentwicklung, bis hin zu allen Betriebsthemen.

Mehr Infos: https://www.innoq.ai

9.2 Training: GenAI mit RAG für Entwickler:innen

In diesem Training beleuchten wir, wie man LLMs mit eigenen Daten versorgt. Dabei setzen wir auf die Architektur der Retrieval-Augmented Generation (RAG). Ziel des Kurses ist es, alle Bausteine einer einfachen RAG-Architektur kennenzulernen. Die Experimente sind darauf ausgelegt, diese Bausteine Schritt für Schritt vorzustellen. Am Ende des Trainings werden wir alle Komponenten zu einem Chatbot zusammenführen, der mit unseren eigenen Dokumenten arbeitet.

Mehr Infos und Termine: https://www.socreatory.com/de/trainings/genAI

Wir beraten ehrlich, denken innovativ und entwickeln leidenschaftlich gern. Das Ergebnis: Erfolgreiche Softwarelösungen, Infrastrukturen und Geschäftsmodelle.

Als Technologieunternehmen fokussieren wir uns auf Strategie- und Technologieberatung, Softwarearchitektur und -entwicklung, Methoden- und Technologietraining sowie Plattform-Infrastrukturen.

Wir unterstützen mit über 160 Mitarbeiter*innen an Standorten in Deutschland und der Schweiz Unternehmen und Organisationen bei Konzeption und Umsetzung komplexer Vorhaben und der Verbesserung bestehender Softwaresysteme.

Wir engagieren uns in Open-Source-Projekten sowie dem iSAQB e.V., und geben Wissen und Erfahrungen auf Konferenzen und Meetups sowie in zahlreichen Büchern und Fachartikeln weiter.

Besuchen Sie uns: **www.innoq.com**

Die Autoren

Robert Glaser

Robert Glaser ist Head of Data & AI bei INNOQ. Mit einem Hintergrund im Software Engineering und einer Leidenschaft für benutzerfreundliche Webanwendungen begleitet er heute Unternehmen durch die KI-Landschaft. Dabei unterstützt er sie bei der Entwicklung von Strategien und Produkten für schwierige technische Herausforderungen. Fasziniert von der Vielseitigkeit generativer KI, moderiert er den Podcast „AI und jetzt“. Ihm ist die Brücke zwischen Technologie und Business ein Herzensanliegen. In seiner Freizeit erkundet er die lokale Food Szene.

Alexander Kniesz

Seit seinem abgeschlossenen Masterstudium in Data Science fokussiert sich Alexander auf Themen wie Datenverarbeitung, Machine Learning und Künstliche Intelligenz. Er ist dabei ebenfalls an verwandten Themen, wie z.B. die Datenhaltung und MLOps, interessiert, da nur mit diesen Machine Learning erfolgreich in der Produktion eingesetzt werden kann.

Hermann Schmidt

Hermann Schmidt arbeitet als Senior Consultant bei INNOQ. Nach über zwei Jahrzehnten als Entwickler und Architekt, die sich hauptsächlich um die Frage des „Wie“ der Softwareentwicklung gedreht haben, rückt bei ihm heute das „Was“ und „Wer“ in den Vordergrund. Als Facilitator interessieren ihn Teamstrukturen, Entwicklungs- und Innovationsprozesse, sowie Kreativtechniken. Probleme, die sich in der Wolke zwischen Fachbereich und Entwicklungsteam verstecken, sind sein Lieblingsgebiet. In letzter Zeit haben die Large Language Models bei ihm einen Funken gezündet, der ihn an die Zeit als 17-Jähriger im Gymnasium erinnert, als er fasziniert mit großen Augen vor dem einzigen Computer saß und seine ersten Programme schrieb.

Marco Steinke

Marco Steinke ist Consultant bei INNOQ. Sein Schwerpunkt liegt in der Software-Architektur. Außerdem beschäftigt er sich mit künstlicher Intelligenz und dabei vor allem mit der Architektur und der Integration von KI-Systemen.

Retrieval-Augmented Generation (RAG) ermöglicht es, die Fähigkeiten von Large Language Models für unternehmensinternes Wissen zu erschließen. Das ermöglicht nicht nur Antwortsysteme wie Chatbots, sondern auch die Umsetzung von zuvor schwierigen oder unmöglichen Features auf der Grundlage von internen, unstrukturierten Daten.

In diesem Primer führen wir systematisch in die Konzepte und Architektur von RAG ein. Wir behandeln sowohl theoretische Grundlagen als auch praktische Implementierungsaspekte wie Chunking, Embedding und Vektordatenbanken. Außerdem teilen wir unsere Praxiserfahrung aus echten Projekten.

Für Softwarearchitekt:innen und -entwickler:innen, die einen kompakten, aber fundierten Einstieg ins Thema suchen und den Einsatz von RAG in der eigenen Organisation bewerten wollen.

innoq.com