DeepCite

# Inhaltsverzeichnis

# 1 Einleitung

Wissenschaftliche Literatur wächst rasant, Recherchen dauern lange und Antworten aus LLMs sind oft schwer nachprüfbar. Retrieval-Augmented Generation (RAG) [1] lindert dieses Problem, leidet jedoch in der Praxis unter zwei Hürden:

- (a) Nutzer:innen können die genaue Quelle des Kontexts selten steuern.
- (b) Antworten liefern keine präzisen, seiten-genauen Belege.

Für forschungsnahe Szenarien (Seminare, Literaturübersichten, Methodenvergleiche) sind Kontrollierbarkeit und Zitierbarkeit zentral: Wer die relevanten Workspaces/Dokumente/Seiten bestimmen kann, reduziert Halluzinationen und erhöht Vertrauen.

Aktuelle Systeme wählen Kontexte meist vollautomatisch (Top-k Retrieval über ganze Korpora). Dadurch:

- **Geringe Steuerbarkeit**: Nutzer:innen können selten gezielt nur bestimmte Workspaces, Dokumente oder Seitenbereiche (z. B. "Methoden S. 3–6") festlegen.
- **Qualitätsrisiken**: Unpassende Kontexte führen zu Halluzinationen oder zu vagen Aussagen ohne belastbare Quelle.
- **Fehlende Nachprüfbarkeit**: Quellenangaben verweisen oft nur auf ein Dokument, nicht auf Seiten, Absätze oder Figuren.
- **Technische Lücke**: PDF-Parsing produziert selten stabile Absatz-IDs und Seiten-Offsets, die für seiten-genaue Evidenz zwingend nötig sind.
- **Onboarding-Hürde**: Relevante Artikel sind über arXiv/Preprints zwar auffindbar, fehlen aber als schnell importierbare Workspaces.

Ziel ist daher ein RAG-System, das den Kontext steuerbar macht (bis zur Seite/zum Abschnitt) und jede Aussage mit seiten-genauer Evidenz belegt.

# 2 Grundlagen und Stand der Forschung

## 2.1 Large Language Models

LLMs sind großskalige, vortrainierte, neuronale Sprachmodelle, meist Transformer-basiert (Abb. 1), mit Milliarden Parametern, die auf Web-Skalentexten gelernt werden und dadurch generalistische Sprachverstehens- und -generationsfähigkeiten sowie emergente Fähigkeiten wie In-Context-Learning, Instruktionsbefolgung und mehrstufiges Reasoning zeigen [2, S. 1-2].

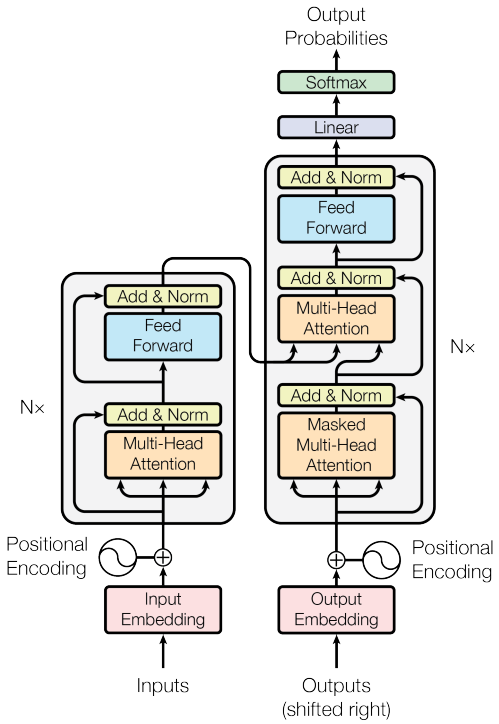


Abbildung 1: Transformer Architektur. [3]

Die Entwicklung lässt sich heuristisch in vier Wellen gliedern [2, S. 1-2]: (i) statistische n-Gramm-Modelle [4], die Wortwahrscheinlichkeiten unter Markov-Annahmen schätzen, Glättungsverfahren benötigen und stark unter Datensparsamkeit leiden; (ii) frühe neuronale Sprachmodelle auf Basis von RNN/LSTM/GRU [5], die Wörter in kontinuierliche Vektorräume abbilden, Datenknappheit mildern und breite Anwendungen in maschineller Übersetzung und Textklassifikation fanden; (iii) vortrainierte Sprachmodelle (PLMs), die ein Pretraining → Finetuning-Paradigma auf Web-Skalen mit RNNs oder (vor allem) Transformern etablierten (z.B. BERT-Familie [6], Encoder-Decoder-Modelle); und (iv) LLMs im engeren Sinn, die PLMs um Größenordnungen skalieren und dabei emergente Fähigkeiten ausgeprägt zeigen (z. B. GPT-[7], LLaMA-[8], PaLM-Familien[9])

### 2.1.1 Grenzen

Große Sprachmodelle lassen sich primär als Next-Token-Prädiktoren charakterisieren, aus diesem Grundprinzip ergeben sich zentrale Grenzen, darunter das Fehlen eines genuinen "Wahrheitsverständnisses" [2, S. 18].

- **Fehlender interner Zustand (Memory)**: Ohne explizite Speicherung verfügen LLMs über kein persistentes Gedächtnis und "vergessen" selbst die vorherige Eingabe. Zahlreiche Anwendungsfälle erfordern jedoch zustandsbehaftete Architekturen bzw. externe Speichermechanismen [2, S. 21].
- **Stochastizität/Nicht-Determinismus**: Identische Prompts können unterschiedliche Antworten erzeugen. Steuerparameter wie die Temperatur können die Varianz begrenzen, eliminieren sie jedoch nicht [2, S. 21].
- **Veraltetes Wissen und fehlender Echtzeitzugriff**: Ein LLM kennt weder die aktuelle Zeit noch Informationen außerhalb seines Trainingskorpus [2, S. 21, 24].
- **Halluzinationen (mangelnde Faktentreue)**: Modelle können inhaltlich plausible, aber falsche Ausgaben erzeugen [10][11][2, S. 21, 22]. Man unterscheidet intrinsische Halluzinationen (im Widerspruch zu vorhandenen Quellen) von extrinsischen (nicht verifizierbar).

### 2.1.2 Prompting

Prompting-Methoden haben sich in den letzten Jahren von heuristischen Formulierungsregeln zu systematisch untersuchten Verfahren entwickelt, die die Reasoning-Leistung, Faktentreue und Effizienz großer Sprachmodelle (LLMs) messbar beeinflussen [12].

- **Reasoning-Genauigkeit**: Als Baseline etabliert sich Chain-of-Thought (CoT) [13], das mehrschrittige Schlussfolgerungen explizit macht und die Leistung auf gängigen Benchmarks signifikant erhöht (z. B. PaLM-540B  90,2 % auf Reasoning-Suiten). Über die lineare Sequenz hinausgehend formalisieren Tree-of-Thoughts (ToT) [14] und Graph-of-Thoughts (GoT) [15] den Suchraum von Überlegungen. ToT zeigt beispielsweise im "Game of 24" eine Erfolgsrate von 74 % gegenüber 4 % mit CoT und 60 % gegenüber 16 % bei Wortpuzzles. GoT modelliert nicht-lineare Abhängigkeiten und erzielt auf GSM8K Zugewinne von +3,41 pp/+5,08 pp (T5-base/-large) sowie auf ScienceQA +6,63 pp/+1,09 pp gegenüber Multimodal-CoT. [12, S. 2, 4]
- **Kontextmanagement und Repräsentation langer Eingaben**: Lange und "chaotische" Kontexte verschlechtern die Fokussierung von LLMs. Thread-of-Thought (ThoT) [16] adressiert dieses Problem durch Segmentierung und Re-Organisation des Materials und führt zu beträchtlichen Leistungsanstiegen (z. B. +47,2 % in QA-Settings und +17,8 % in Konversationen). [12, S. 4]
- **Reduktion von Halluzinationen**: Faktentreue profitiert von externer Evidenz und expliziten Verifikationsschritten. Retrieval-Augmented Generation (RAG) [1] hebt die Exact-Match-Werte u. a. auf 56,8 % (TriviaQA) bzw. 44,5 % (Natural Questions) und adressiert damit Aktualität und Quellenbindung. [12, S. 7]

## 2.2 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) [1] ist ein Verfahren, bei dem ein Sprachmodell vor oder während der Antwort gezielt Informationen aus externen Datenquellen abruft und diese in die Erzeugung integriert, um die Ausgabe genauer und robuster zu machen [17, S. 1][18, S. 1]. RAG hilft insbesondere dabei, Halluzinationen zu mindern, aktuelles Wissen einzubinden und Antworten in wissensintensiven Domänen zu verbessern [19, S. 1].

### 2.2.1 Von TF-IDF zu Embeddings

TF-IDF und Word Embeddings (z.B. Word2Vec, BERT) sind beides Techniken zur numerischen Darstellung von Wörtern, sie unterscheiden sich jedoch grundlegend in ihrer Funktionsweise und Anwendung.

TF-IDF steht für Term Frequency-Inverse Document Frequency. Es ist eine Methode aus der Information Retrieval und Textmining, die dazu verwendet wird, die Wichtigkeit eines Wortes in einem Dokument in Bezug auf eine Sammlung von Dokumenten (Korpus) zu bestimmen [20]. TF-IDF kombiniert zwei Metriken:

$$\text{TF-IDF} = TF(t,d) \times IDF(t,D)$$

**Term Frequency (TF)**: Die Häufigkeit eines Begriffs in einem Dokument. TF misst, wie oft ein Begriff in einem Dokument vorkommt $\rightarrow TF(t,d) = \frac{f(t,d)}{\sum_{t \in d} f(t',d)}$

- $t$ ist der Begriff.

- $d$ ist das Dokument.
- $f(t, d)$ ist die Häufigkeit des Begriffs $t$ im Dokument $d$.
- $\sum_{t \in d} f(t', d)$ ist die Gesamtanzahl der Begriffe im Dokument $d$

**Inverse Document Frequency (IDF)**: Die Umkehrung der Dokumenthäufigkeit, die misst, wie wichtig ein Begriff ist. Je seltener ein Begriff in allen Dokumenten vorkommt, desto höher ist der IDF-Wert $\rightarrow IDF(t, D) = log(\frac{N}{|\{d \in D : t \in d\}|})$

- $D$ ist das Korpus (eine Sammlung von Dokumenten).
- $N$ ist die Gesamtzahl der Dokumente im Korpus.
- $|\{d \in D : t \in d\}|$ ist die Anzahl der Dokumente, die den Begriff $(t)$ enthalten.

**Word Embeddings** sind eine fortschrittlichere Methode zur numerischen Darstellung von Wörtern, die auf der Idee basiert, dass Wörter in einem Vektorraum dargestellt werden, wobei ähnliche Wörter durch Vektoren repräsentiert werden, die sich nahe beieinander befinden. Diese Technik ermöglicht es, semantische Ähnlichkeiten und Beziehungen zwischen Wörtern besser zu erfassen als TF-IDF [21].

Im Gegensatz zu TF-IDF, das auf der Häufigkeit von Begriffen basiert, lernen Word Embeddings kontinuierliche Vektoren, die die Bedeutung von Wörtern durch deren Kontext in großen Textmengen erfassen. Es gibt verschiedene Ansätze zur Erstellung von Word Embeddings, darunter Word2Vec, BERT.

Die **Kosinusähnlichkeit** misst, wie ähnlich zwei Vektoren in einem hochdimensionalen Raum sind, basierend auf dem Winkel zwischen ihnen.

Um Kosinusähnlichkeit zwischen **Abfragevektor** $q = \begin{bmatrix} q_1 \\ \dots \\ q_n \end{bmatrix} \in \mathbb{R}^{n \times 1}$ und Dokumentenvektoren $D = \begin{bmatrix} d_{1,1} & \dots & d_{1,n} \\ \dots & \dots & \dots \\ d_{m,1} & \dots & d_{m,n} \end{bmatrix} \in \mathbb{R}^{m \times n}$ zu berechen, kann folgende Formel benutzt werden:

$$sim(q, d_i) = \frac{q \cdot d_i}{||q|| \cdot ||d_i||}$$

- Skalarprodukt zwischen der Abfrage und jedem Dokument $q \cdot d_i = \sum_{j=1}^{N} q_j \times d_{i,j}$.
- Normen der Vektoren $||v|| = \sqrt{\sum_{j=1}^{N} v_j^2}$

### 2.2.2 RAG Workflow

Ein typischer RAG-Workflow umfasst vier Schritte (Abb. 2) [18, S. 3]:

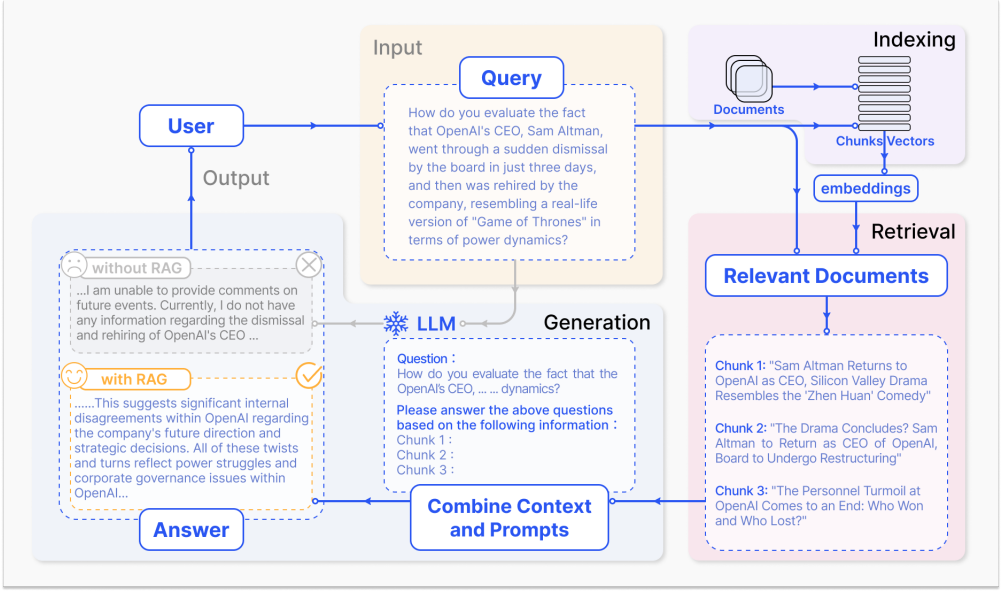


Abbildung 2: zeigt den technischen Ablauf eines Retrieval-Augmented Generation (RAG) Systems dar, das für die Beantwortung von Benutzeranfragen verwendet wird und die Hauptphasen umfasst: Eingabe (Input), Indexierung (Indexing), Abruf (Retrieval-Augmentation), Generierung (Generation) und Ausgabe (Output). [18]

1. **Indexing (Indexierung)**: Dokumente werden in Chunks (Textblöcke) aufgeteilt, die durch Vektorisierung in Embeddings (Vektoren) umgewandelt werden. Die Embeddings werden gespeichert, um die schnelle Suche nach relevanten Inhalten zu ermöglichen..
2. **Input (Eingabe)**: Ein Benutzer stellt eine Anfrage (Query), die an das System übergeben und als ein Vektor im semantischen Raum dargestellt wird.
3. **Retrieval (Abruf)**: Der Prozess des Abrufens von relevanten Dokumenten oder Informationen aus einer Datenbank oder Sammlung basierend auf einer gegebenen Anfrage oder Query. Das Embedding-Modell transformiert die Query in einen Vektorraum. Danach wird eine Menge von Dokumenten oder Wissensdaten aus einer Datenbank abgerufen. Der Retriever durchsucht eine vordefinierte Sammlung von Texten (interne Datenbanken), berechnet die Ähnlichkeit zwischen Vektorräumen der Query und Dokumenten und liefert die relevantesten Dokumente für die gegebene Query zurück
4. **Augmented Generation (Erweiterte Generierung)**: Generierung der Antwort basierend auf die Query und Kontext (relevante Dokumente). Der generative Teil, oft ein großes Sprachmodell, wird dann mit den abgerufenen Dokumenten und der Query gefüttert. Das Sprachmodell verwendet diese zusätzlichen Informationen, um eine präzisere und kontextbezogenere Antwort zu generieren.

### 2.2.3 Beste Praktiken

Die Leistungsfähigkeit eines Retrieval-Augmented Generation (RAG) Systems hängt maßgeblich von einer Reihe bewährter Methoden in der Daten- und Anfrageverarbeitung ab. Eine optimierte Vorverarbeitung der Daten sowie eine effiziente Handhabung der Suchanfragen sind entscheidend für die Präzision und Effizienz der Informationsextraktion.

**Chunking** bezieht sich auf das Aufteilen von Dokumenten in kleinere, verarbeitbare Einheiten oder ”Chunks”. Dies kann auf verschiedenen Ebenen geschehen, z. B. auf Token-, Satz- oder semantischen Ebenen. Ziel ist es, die Verarbeitung zu optimieren und relevante Informationen präziser abzurufen. [19, S, 5-6][18, S, 8]. Die Wahl der Chunk-Größe beeinflusst, wie viel Kontext ein einzelner Chunk behält. Kleinere Chunks erhöhen die Retrieval-Präzision, da sie spezifische Informationen isolieren,

aber sie können den Kontext verlieren. Größere Chunks bieten mehr Kontext, was die Kohärenz verbessert, aber sie können auch irrelevante Informationen enthalten. (Tab. 1)

| Chunk Size | lyft 2021 | | Uber 10K SEC Filings 2021 | |
|---|---|---|---|---|
| | **Average Faithfulness** | **Average Relevancy** | **Average Faithfulness** | **Average Relevancy** |
| 2048 | 80.37 | 91.11 | 90 | 89 |
| 1024 | 94.26 | 95.56 | **93** | **90** |
| 512 | **97.59** | 97.41 | 85 | 85 |
| 256 | 97.22 | **97.78** | 90 | 78 |
| 128 | 95.74 | 97.22 | 85 | 78 |

Tabelle 1: Chunk Größen Vergleich [19][22]

Fortgeschrittene Chunking-Techniken wie "small-to-big" oder "sliding window" verbessern die Retrieval-Qualität (Tab. 2), indem sie den Kontext beibehalten und sicherstellen, dass relevante Informationen abgerufen werden.

| Chunk Skill | lyft 2021 | |
|---|---|---|
| | **Average Faithfulness** | **Average Relevancy** |
| Original | 95.74 | 95.37 |
| small2big | 96.67 | 95.37 |
| sliding window | **97.41** | **96.85** |

Tabelle 2: Chunk Skills Vergleich [19]

Maximale Genauigkeit kann mit **Query-Klassifikation**, **Hybrid + HyDE** Retrieval und **Reranking** erreicht werden.

**HyDE** [23] (Pseudo-Antwort → dichte Einbettung) erhöht mAP/nDCG gegenüber reinem Dense und schlägt BM25 in Zero-Shot-Szenarien deutlich. Hybrid (BM25 + Dense) liefert zusätzliche Gewinne. In TREC DL19/20 und BEIR zeigt HyDE starke Zugewinne (z. B. DL19: mAP 50.87 vs. 23.99 für Contriever; Recall@1k 88.76 vs. 74.59) [23, S, 5]. Kombiniert mit Hybrid erzielt man die höchsten Retrieval-Scores [19, S, 7-8]. (Tab. 3)

| **Method** | **TREC DL19** | | | | | **TREC DL20** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | mAP | nDCG@10 | R@50 | R@1k | Latency | mAP | nDCG@10 | R@50 | R@1k | Latency |
| *unsupervised* | | | | | | | | | | |
| BM25 | 30.13 | 50.58 | 38.32 | 75.01 | 0.07 | 28.56 | 47.96 | 46.18 | 78.63 | 0.29 |
| Contriever | 23.99 | 44.54 | 37.54 | 74.59 | 3.06 | 23.98 | 42.13 | 43.81 | 75.39 | 0.98 |
| *supervised* | | | | | | | | | | |
| LLM-Embedder | 44.66 | 70.20 | 49.06 | 84.48 | 2.61 | 45.60 | 68.76 | 61.36 | 84.41 | 0.71 |
| + Query Rewriting | 44.56 | 67.89 | 51.45 | 85.35 | 7.80 | 45.16 | 65.62 | 59.63 | 83.45 | 2.06 |
| + Query Decomposition | 41.93 | 66.10 | 48.66 | 82.62 | 14.98 | 43.30 | 64.95 | 57.74 | 84.18 | 2.01 |
| + HyDE | 50.87 | 75.44 | 54.93 | 88.76 | 7.21 | 50.94 | 73.94 | 63.80 | 88.03 | 2.14 |
| + Hybrid Search | 47.14 | 72.50 | 51.13 | 89.08 | 3.20 | 47.72 | 69.80 | 64.32 | 88.04 | 0.77 |
| + HyDE + Hybrid Search | 52.13 | 73.34 | 55.38 | 90.42 | 11.16 | 53.13 | 72.72 | 66.14 | 90.67 | 2.95 |

Tabelle 3: Ergebnisse von unterschiedlichen Retrieval Methoden TREC DL19 / TREC DL20 [19]

**Reranking**: Ohne Reranker fällt die Leistung signifikant, monoT5 war am stärksten. Reverse-Repacking (relevanteres Material näher an die Query) verbesserte die Ausgabequalität. Recomp half, Kontextlängen zu beherrschen, bei vertretbarer Latenz [19, S, 2, 11-12]. (Tab. 4)

| Method | MS MARCO Passage ranking | | | | | | |
|---|---|---|---|---|---|---|---|
| | Base Model | # Params | MRR@1 | MRR@10 | MRR@1k | Hit Rate@10 | Latency |
| *w/o Reranking* | | | | | | | |
| Random Ordering | - | - | 0.011 | 0.027 | 0.068 | 0.092 | - |
| BM25 | - | - | 6.52 | 11.65 | 12.59 | 24.63 | - |
| *DLM Reranking* | | | | | | | |
| monoT5 | T5-base | 220M | 21.62 | 31.78 | 32.40 | 54.07 | 4.5 |
| monoBERT | BERT-large | 340M | 21.65 | 31.69 | 32.35 | 53.38 | 15.8 |
| RankLLaMA | Llama-2-7b | 7B | 22.08 | 32.35 | 32.97 | 54.53 | 82.4 |
| *TILDE Reranking* | | | | | | | |
| TILDEv2 | BERT-base | 110M | 18.57 | 27.83 | 28.60 | 49.07 | 0.02 |

Tabelle 4: Resultate verschiedener Re-Ranking-Algorithmen auf dem Dev-Set des MS MARCO Passage-Ranking. Pro Anfrage werden die Top-1000 Kandidaten, die BM25 zurückliefert, erneut gereiht; gemessen wird die Latenz in Sekunden pro Anfrage. [19]

**Metadaten** sind zusätzliche Informationen, die mit Chunks verknüpft werden, wie Titel, Schlagwörter, Autorinformationen oder Zeitstempel. Sie bieten zusätzlichen Kontext und verbessern die Effizienz der Informationssuche. [19, S, 6][18, S, 8]. Metadaten verbessern die Relevanz der abgerufenen Chunks, da sie als zusätzliche Filterkriterien dienen können. Sie ermöglichen zeitbewusste Suchen, bei denen Informationen auf Basis von Zeitstempeln gewichtet werden, um aktuelle Informationen zu bevorzugen. Auch dazu erleichtern Metadaten es, Dokumente strukturiert zu indizieren, was die Effizienz und Genauigkeit von Retrieval-Prozessen erhöht. Dies ist besonders relevant in Szenarien, in denen verschiedene Arten von Abfragen unterschiedliche Arten von Informationen erfordern.

**Vektordatenbanken** speichern die Embeddings und deren Metadaten und ermöglichen effiziente Abfragen basierend auf semantischen Ähnlichkeiten. [19, S, 2, 6][18, S, 8]. Vektordatenbanken wie Milvus[24], Faiss[25], Qdrant[26], Weaviate[27] oder Chroma[28] ermöglichen schnelle und skalierbare Abfragen in großen Datensätzen, was die Effizienz von RAG-Systemen signifikant verbessert. Einige Vektordatenbanken unterstützen hybride Suchen, die sowohl Vektor-basierte als auch traditionelle Keyword-basierte Abfragen kombinieren, um die Genauigkeit zu erhöhen. Die Wahl der Vektordatenbank beeinflusst die Skalierbarkeit eines RAG-Systems erheblich, da sie bestimmt, wie gut das System mit wachsenden Datenmengen und komplexeren Abfragen umgehen kann.

### 2.2.4 Paradigmen

Die traditionellen RAG-Paradigmen bilden ein Spektrum, das von minimalistischer Einfachheit bis zu hochgradiger Zusammensetzbarkeit reicht (Abb. 3).

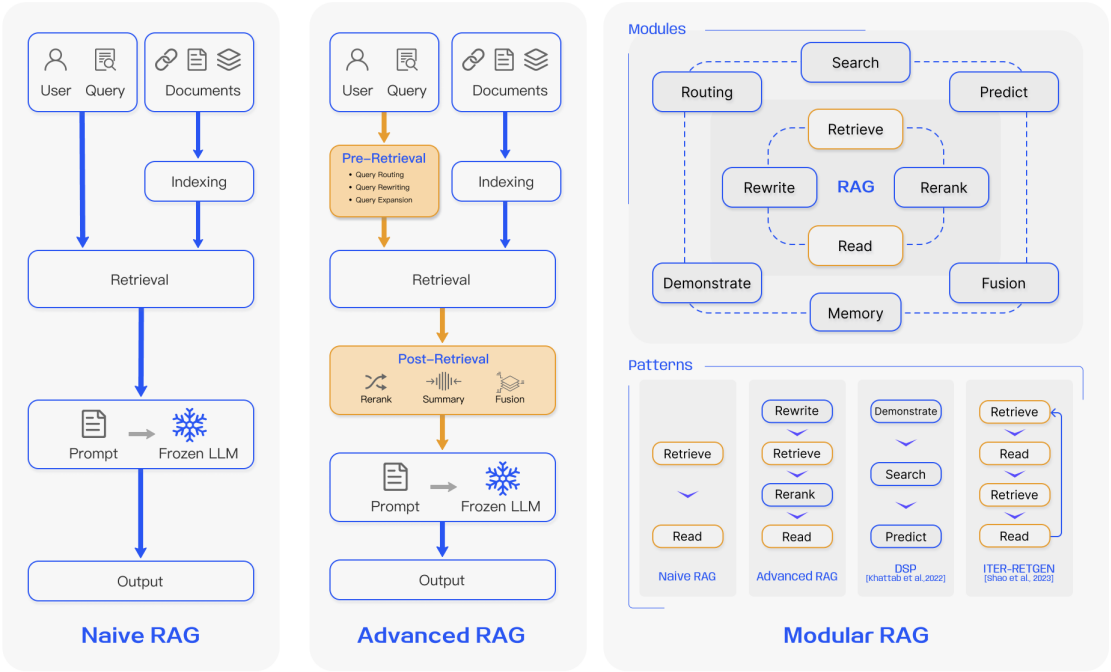


Abbildung 3: Die traditionellen RAG-Paradigmen (Naive, Advanced, Modular). [18]

- **Naive RAG** steht für die klassische Pipeline aus Indexierung, Abruf und Generierung. Dokumente werden normalisiert, in Chunks segmentiert, in einen Vektorspeicher oder eine Sparse-Struktur überführt und zur Laufzeit per Ähnlichkeitssuche eingestreut. Naive Systeme neigen zudem zu repetitiven Antworten, wenn redundante Chunks fast identische Evidenz liefern, und sie stoßen bei Multi-Hop-Fragen ohne zusätzliche Orchestrierung an Grenzen.
- **Advanced RAG** adressiert diese Defizite durch Optimierungen vor und nach dem Abruf. Vor dem Abruf stehen Query-Reformulierung, semantische Ausweitung und De-Ambiguisierung, oft mithilfe der LLM-gestützten Paraphrase oder Subquery-Generierung. Auf Index-Seite verbessern feinere Segmentierung, Sliding-Windows, reichhaltige Metadaten und gemischte Sparse-/Dense-Indizes die Trefferqualität. Nach dem Abruf erhöhen Re-Ranking und Kontextkompression die Präzision an der Prompt-Peripherie, indem zentrale Evidenz nach vorn priorisiert und irrelevanter Ballast entfernt wird.
- **Modular RAG** verschiebt den Fokus von Optimierung einzelner Stufen zur Komponierbarkeit des gesamten Systems. Neue Module (Such-Wrapper über Web-Engines, domänenspezifische Datenbank-Connectors, Query-Router, Langzeit-Speicher, Vorhersage-/”Predict”-Bausteine oder Task-Adapter) lassen sich bedarfsabhängig einfügen oder ersetzen. Ebenso können Interaktionsmuster variiert werden: Rewrite–Retrieve–Read, Demonstrate–Search–Predict [29], iterative Retrieve–Read-Schleifen [30] oder HyDE-Varianten [23].

**Agentic RAG** markiert einen qualitativen Sprung von rein sequentiellen ”Retrieve–Read”-Pipelines hin zu adaptiven, selbststeuernden Systemen [31, S, 6-11]. Im Zentrum steht nicht länger ein starrer Datenfluss, sondern ein Entscheidungsprozess: Ein Agent (typischerweise ein LLM mit expliziter Rolle und Ziel) plant, reflektiert und wählt Werkzeuge aus, um Retrieval und Generierung iterativ zu verzahnen (Abb. 4).

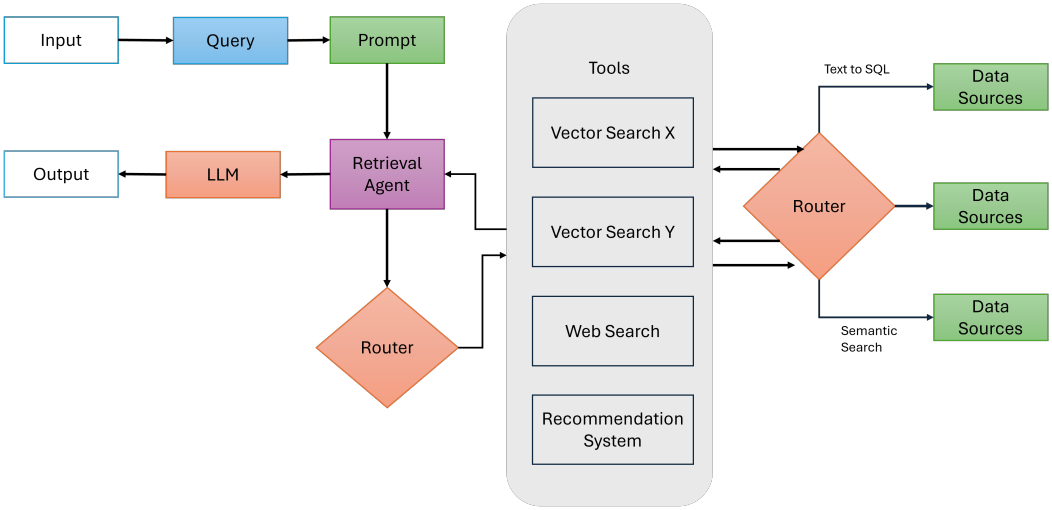


Abbildung 4: Agentic RAG. [31]

Vier Bausteine prägen diese Architektur. Erstens fungiert das LLM als Steuer- und Begründungsinstanz, es zerlegt Aufgaben, prüft Zwischenergebnisse und ändert Strategien, wenn Evidenz oder Unsicherheit dies nahelegen. Zweitens etabliert das System Gedächtnis in zwei Zeitskalen: kurzlebige Konversationskontexte und persistente, domänenspezifische Langzeitgedächtnisse. Drittens erweitert gezielter Werkzeugeinsatz (von Vektorsuche und Websuche über strukturierte Abfragen bis hin zu Kalkulations- und Simulations-APIs) den Handlungsraum über bloße Textgenerierung hinaus. Viertens ermöglichen Planungs[32]- und Reflexionsschleifen [33][34][35], ggf. in Multi-Agent-Konfigurationen, eine fortlaufende Qualitätssicherung durch Selbstkritik, Re-Reranking, Evidenzprüfung und Budgetsteuerung.

Im Ergebnis sind die "klassischen" RAG-Ansätze nicht obsolet, sondern bilden das Fundament, auf dem Adaptivität, Komponierbarkeit und Relationsexplizitheit aufsetzen. Wer ein System entwirft, beginnt sinnvollerweise mit einer schlanken Advanced-RAG-Baseline, misst gründlich und erweitert gezielt: Module dort, wo Domänenanforderungen es verlangen. Graph-Strukturen, wo Relationen die Hauptrolle spielen. Agentische Orchestrierung, wo Unsicherheit hoch, Informationslage volatil und die Begründungspflicht streng ist. So wird RAG vom statischen Kontextpflaster zur integrierten, erklärbaren Wissensmaschine.

## 2.3 PDF-Parsing wissenschaftlicher Artikel: Layout, Metadaten, Referenzen

Das Parsen wissenschaftlicher PDF-Dokumente stellt eine komplexe Aufgabe dar, da PDFs primär für visuelle Darstellung optimiert sind und die semantische Strukturinformation verloren geht. Besonders herausfordernd sind mathematische Ausdrücke, komplexe Tabellen, verschachtelte Layouts und die Erhaltung hierarchischer Dokumentstrukturen [36, S, 1-2].

### 2.3.1 Parsing Tools

Im Kern unterscheiden sich drei Klassen von Werkzeugen für die strukturtreue Parsing wissenschaftlicher PDFs, einschließlich Inhaltsverzeichnis, Metadaten und Literaturverzeichnis, durch ihre Trainingsziele und Systemschnittstellen: klassische, auf domänenspezifische Strukturen optimierte Parser, allgemeine VLMs [37] mit Dokumentfähigkeiten und spezialisierte OCR/VLM-Modelle. Ein zentrales Vergleichskriterium über alle Klassen hinweg ist die Page-Grounding-Fähigkeit, also

die reproduzierbare Verankerung extrahierter Einheiten in Seitenkoordinaten, neben Genauigkeit, Durchsatz/Kosten und Integrationsaufwand.

Für born-digitale Artikel ist **GROBID** [38] der robuste Referenzpunkt, weil es den gesamten bibliographischen Lebenszyklus (Header, Affiliations, Referenzen mit typisiertem Feldschema) in TEI/XML abbildet und durch CRF-basierte Modelle über Jahre stabilisiert wurde. In der Praxis ist GROBID sehr schnell und kosteneffizient. Die offizielle Dokumentation berichtet von ∼10,6 PDFs/s auf einem 16-CPU-Server über Millionen Dokumente hinweg, bereitgestellt über einen Docker-basierten REST-Dienst. Für reine Image-Scans führt GROBID selbst kein OCR durch, hier wird eine vorgelagerte OCR-Stufe benötigt.

Allgemeine VLMs sind heute in der Lage, lange PDFs nativ zu verarbeiten, inklusive Diagrammen und Tabellen, und strukturierte Outputs zu generieren. Solche Modellen wie Gemini 2.5 Pro

# 3 Anforderungen und Use-Cases (optional)

## 3.1 Stakeholder, Rollen und typische Szenarien

## 3.2 Funktionale Anforderungen: Workspaces, Import, Parsing, flexible Scope-Auswahl (WS/Dokument/Seite), seitenexakte Zitate, Web-/arXiv-Suche

## 3.3 Nicht-funktionale Anforderungen: Latenz, Kosten, Robustheit, Reproduzierbarkeit

## 3.4 Annahmen, Einschränkungen und Datenquellen (OA, Preprints, Domains)

## 3.5 Risiken, Ethik und Compliance (Urheberrecht, Datenschutz, medizinische Hinweise)

# 4 Systemarchitektur

## 4.1 Architekturüberblick und Kontextdiagramm

## 4.2 Komponenten: Workspace-Service, Ingest/Parsing-Pipeline, RAG-Service, Embedding-Service, Query-Orchestrator, Re-Ranker, LLM-Service, Zitationsprüfer, UI

## 4.3 Datenmodell und Schemata (Chunk, Metadaten inkl. page/section, Workspace/Dokument-IDs)

## 4.4 Schnittstellen und APIs (Import, Suche, QueryContext, Antwortformat)

## 4.5 Skalierung, Observability und Kostensteuerung

# 5 Evaluation (optional)

# 6 Diskussion (optional)

# 7 Fazit und Ausblick

# References

[1] Patrick Lewis u. a. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2020. arXiv: 2005.11401v4. URL: https://arxiv.org/abs/2005.11401v4.

[2] Shervin Minaee u. a. *Large Language Models: A Survey*. 2024. arXiv: 2402.06196v3. URL: https://arxiv.org/abs/2402.06196v3.

[3] Ashish Vaswani u. a. *Attention Is All You Need*. 2017. arXiv: 1706.03762v7. URL: https://arxiv.org/abs/1706.03762v7.

[4] Brown u. a. *Class-based n-gram models of natural language*. 1994. URL: https://dl.acm.org/doi/10.5555/176313.176316.

[5] Farhad Mortezapour Shiri u. a. *A Comprehensive Overview and Comparative Analysis on Deep Learning Models: CNN, RNN, LSTM, GRU*. 2023. arXiv: 2305.17473v4. URL: https://arxiv.org/abs/2305.17473v4.

[6] Jacob Devlin u. a. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. arXiv: 1810.04805v2. URL: https://arxiv.org/abs/1810.04805v2.

[7] Tom B. Brown u. a. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165v4. URL: https://arxiv.org/abs/2005.14165v4.

[8] Hugo Touvron u. a. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971v1. URL: https://arxiv.org/abs/2302.13971v1.

[9] Aakanksha Chowdhery u. a. *PaLM: Scaling Language Modeling with Pathways*. 2022. arXiv: 2204.02311v5. URL: https://arxiv.org/abs/2204.02311v5.

[10] Ziwei Xu, Sanjay Jain und Mohan Kankanhalli. *Hallucination is Inevitable: An Innate Limitation of Large Language Models*. 2023. arXiv: 2401.11817v2. URL: https://arxiv.org/abs/2401.11817v2.

[11] Lei Huang u. a. *A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions*. 2024. arXiv: 2311.05232v2. URL: https://arxiv.org/abs/2311.05232v2.

[12] Pranab Sahoo u. a. *A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications*. 2024. arXiv: 2402.07927v2. URL: https://arxiv.org/abs/2402.07927v2.

[13] Jason Wei u. a. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2022. arXiv: 2201.11903v6. URL: https://arxiv.org/abs/2201.11903v6.

[14] Shunyu Yao u. a. *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*. 2023. arXiv: 2305.10601v2. URL: https://arxiv.org/abs/2305.10601v2.

[15] Maciej Besta u. a. *Graph of Thoughts: Solving Elaborate Problems with Large Language Models*. 2023. arXiv: 2308.09687v4. URL: https://arxiv.org/abs/2308.09687v4.

[16] Yucheng Zhou u. a. *Thread of Thought Unraveling Chaotic Contexts*. 2023. arXiv: 2311.08734v1. URL: https://arxiv.org/abs/2311.08734v1.

[17] Penghao Zhao u. a. *Retrieval-Augmented Generation for AI-Generated Content: A Survey*. 2024. arXiv: 2402.19473. URL: https://arxiv.org/abs/2402.19473.

[18] Yunfan Gao u. a. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2023. arXiv: 2312.10997. URL: https://arxiv.org/abs/2312.10997.

[19] Xiaohua Wang u. a. *Searching for Best Practices in Retrieval-Augmented Generation*. 2024. arXiv: 2407.01219. URL: https://arxiv.org/abs/2407.01219.

[20] Bijoyan Das und Sarit Chakraborty. *An Improved Text Sentiment Classification Model Using TF-IDF and Next Word Negation*. 2018. arXiv: 1806.06407 [cs.CL]. URL: https://arxiv.org/abs/1806.06407.

[21] *Word Embeddings — Text Representation for Neural Networks*. URL: https://medium.com/codex/word-embeddings-text-representation-for-neural-networks-65fd934d1fa2.

[22] *Evaluating the Ideal Chunk Size for a RAG System using LlamaIndex*. URL: https://www.llamaindex.ai/blog/evaluating-the-ideal-chunk-size-for-a-rag-system-using-llamaindex-6207e5d3fec5.

[23] Luyu Gao u. a. *Precise Zero-Shot Dense Retrieval without Relevance Labels*. 2022. arXiv: 2212.10496. URL: https://arxiv.org/abs/2212.10496.

[24] *Milvus is a high-performance vector database built for scale*. URL: https://github.com/milvus-io/milvus.

[25] Matthijs Douze u. a. "The Faiss library". In: (2024). arXiv: 2401.08281 [cs.LG].

[26] *Qdrant (read: quadrant) is a vector similarity search engine and vector database*. URL: https://github.com/qdrant/qdrant.

[27] *Weaviate is an open-source, cloud-native vector database that stores both objects and vectors, enabling semantic search at scale*. URL: https://github.com/weaviate/weaviate.

[28] *Chroma - the open-source embedding database*. URL: https://github.com/chroma-core/chroma.

[29] Omar Khattab u. a. *Demonstrate-Search-Predict: Composing retrieval and language models for knowledge-intensive NLP*. 2022. arXiv: 2212.14024v2. URL: https://arxiv.org/abs/2212.14024v2.

[30] Zhihong Shao u. a. *Enhancing Retrieval-Augmented Large Language Models with Iterative Retrieval-Generation Synergy*. 2023. arXiv: 2305.15294v2. URL: https://arxiv.org/abs/2305.15294v2.

[31] Aditi Singh u. a. *Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG*. 2025. arXiv: 2501.09136. URL: https://arxiv.org/abs/2501.09136.

[32] Xu Huang u. a. *Understanding the planning of LLM agents: A survey*. 2024. arXiv: 2402.02716v1. URL: https://arxiv.org/abs/2402.02716v1.

[33] Aman Madaan u. a. *Self-Refine: Iterative Refinement with Self-Feedback*. 2023. arXiv: 2303.17651v2. URL: https://arxiv.org/abs/2303.17651v2.

[34] Noah Shinn u. a. *Reflexion: Language Agents with Verbal Reinforcement Learning*. 2023. arXiv: 2303.11366v4. URL: https://arxiv.org/abs/2303.11366v4.

[35] Zhibin Gou u. a. *CRITIC: Large Language Models Can Self-Correct with Tool-Interactive Critiquing*. 2023. arXiv: 2305.11738v4. URL: https://arxiv.org/abs/2305.11738v4.

[36] Narayan S. Adhikari und Shradha Agarwal. *A Comparative Study of PDF Parsing Tools Across Diverse Document Categories*. 2025. arXiv: 2410.09871v2. URL: https://arxiv.org/abs/2410.09871v2.

[37] Jingyi Zhang u. a. *Vision-Language Models for Vision Tasks: A Survey*. 2023. arXiv: 2304.00685v2. URL: https://arxiv.org/abs/2304.00685v2.

[38] *GROBID is a machine learning library for extracting, parsing and re-structuring raw documents such as PDF into structured XML/TEI encoded documents with a particular focus on technical and scientific publications*. URL: https://github.com/kermitt2/grobid.