Retrieval Augmented Generation

Jiayi Wang

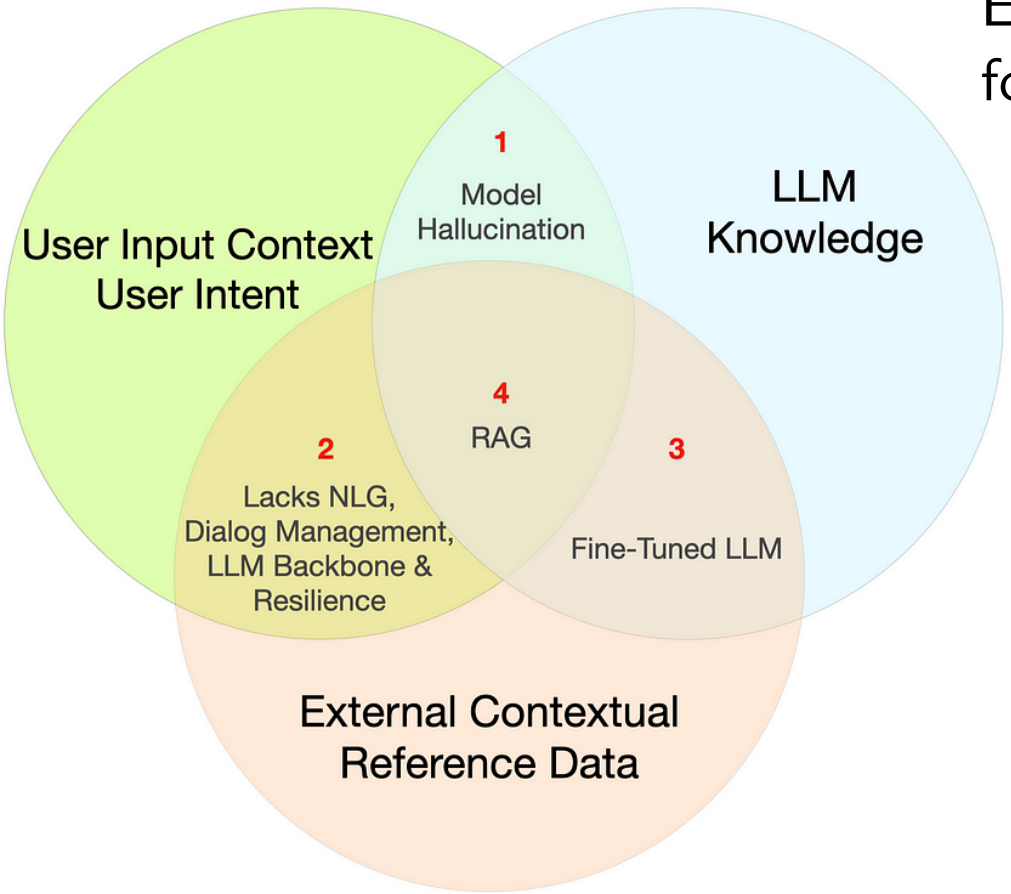
Masterseminar: Conversational AI

28. November 2024

Outline

Outline

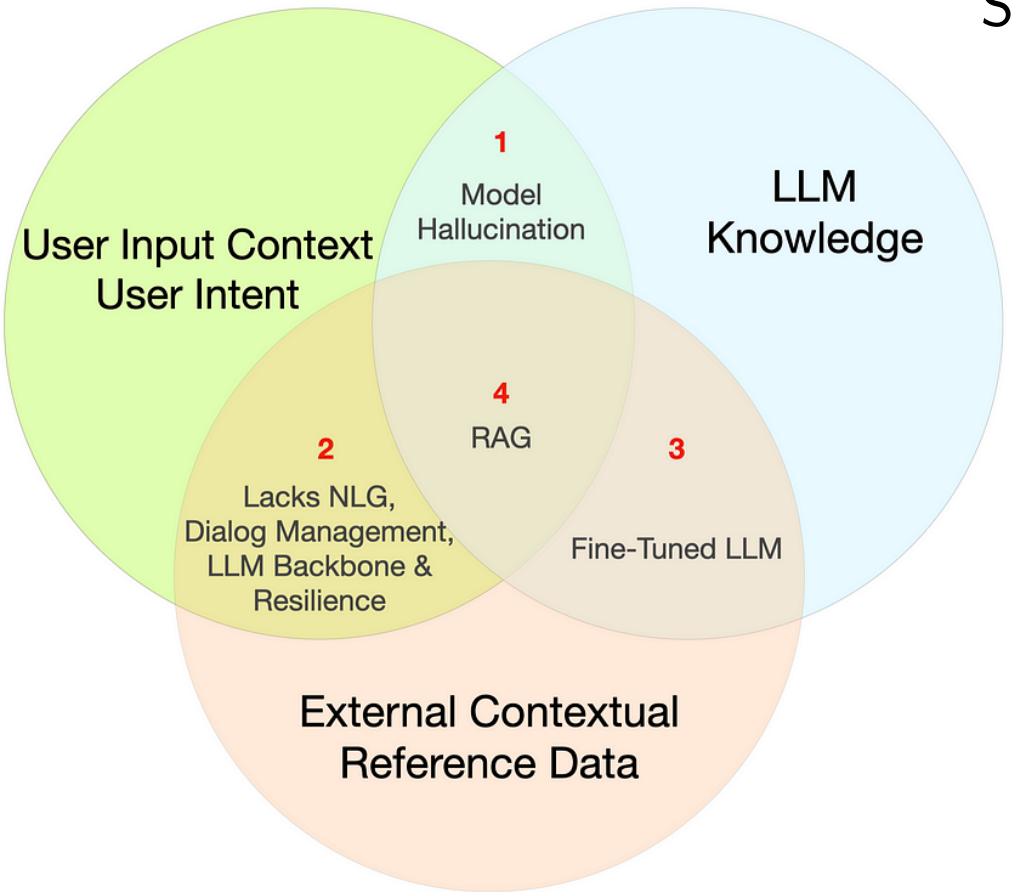
Einleitung



Ein LLM erwirbt Kenntnisse aus den folgenden Bereichen:

- Benutzereingaben, Intentionen: **Wo** *liegt die Stadt* **München***?*
- LLM Trainingsdaten: **München**, *die Hauptstadt des Bundeslandes* **Bayern**, *hat den Titel Grüne Hauptstadt erhalten.* (Nachrichtentext)
- Externe Referenzdaten: Wikipedia (Datenbanken), Google (Web-Suche)

Einleitung



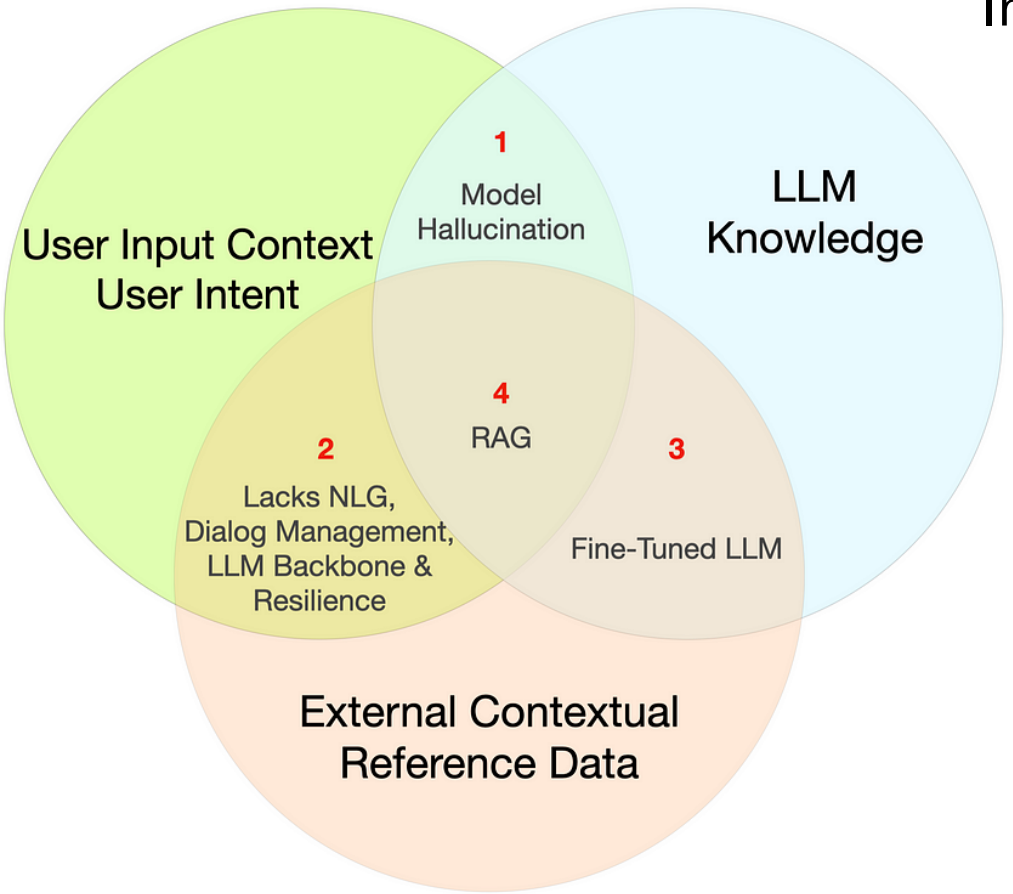
Schnitt 1: Halluzination

- Die Benutzereingabe wird direkt über Prompt-Engineering an den LLM übermittelt.
- Frage: **Wo** *liegt die Stadt* **Ulm***?*

  Trainingsdaten: *Die Stadt Neu-***Ulm** *in* **Bayern** *hat ihr 200-jähriges Bestehen gefeiert.*

  Antwort: *In Bayern liegt Ulm.*

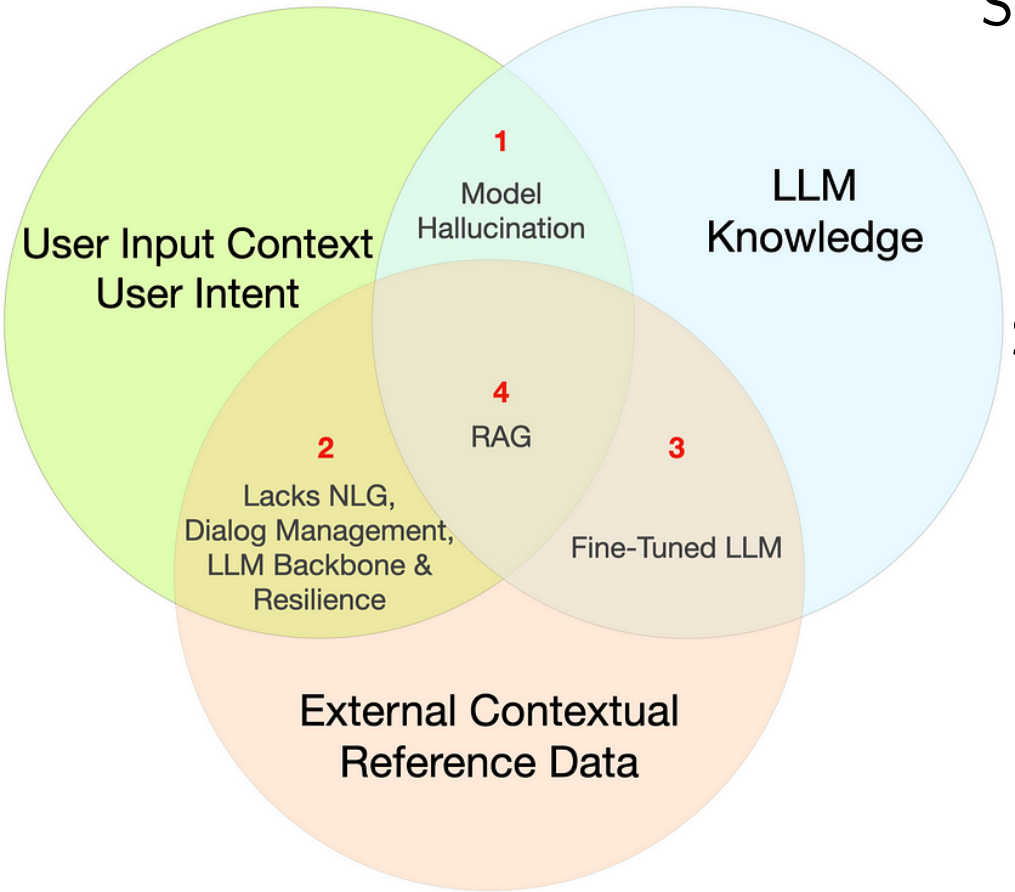
Einleitung



In Bayern liegt Ulm. $\perp$

- LLMs können fehlerhafte oder sinnlose Informationen erzeugen.
- Eine Halluzination liegt vor, wenn ein LLM sehr plausible und glaubwürdige Antworten liefert, die jedoch tatsächlich falsch sind.
- Gründe: Unvollständige, veraltete Trainingsdaten

Einleitung



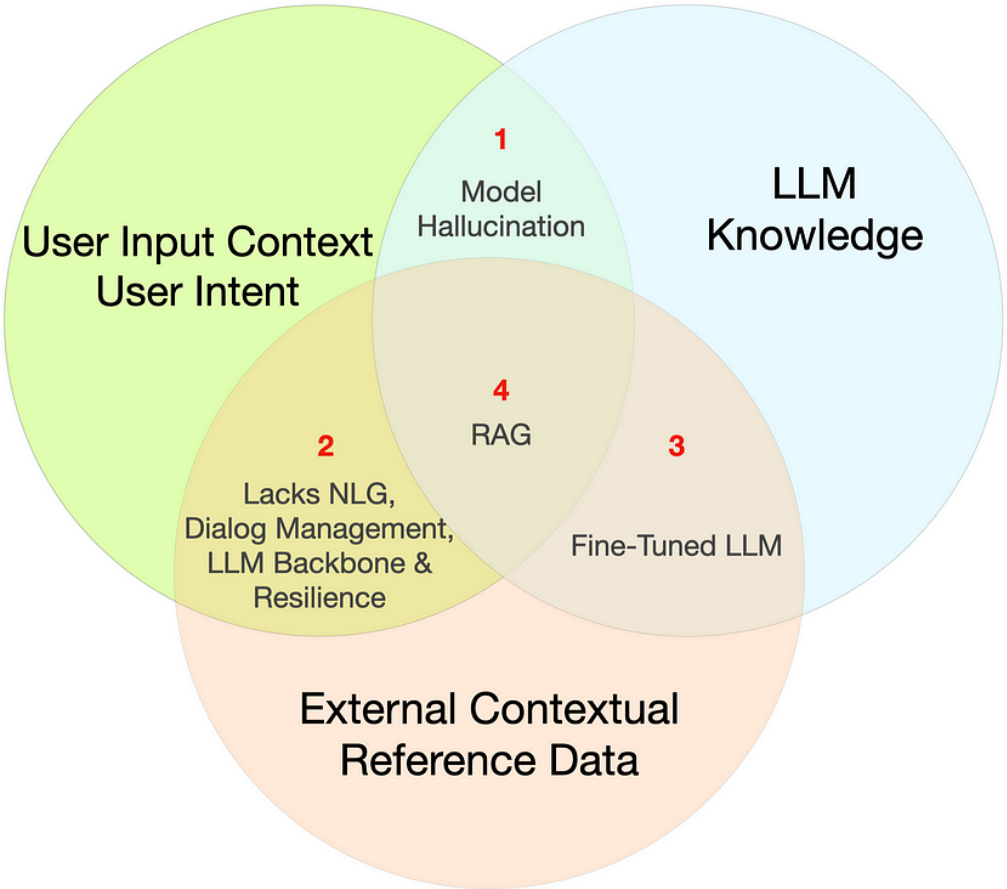
Schnitt 2: NLG Chatbots

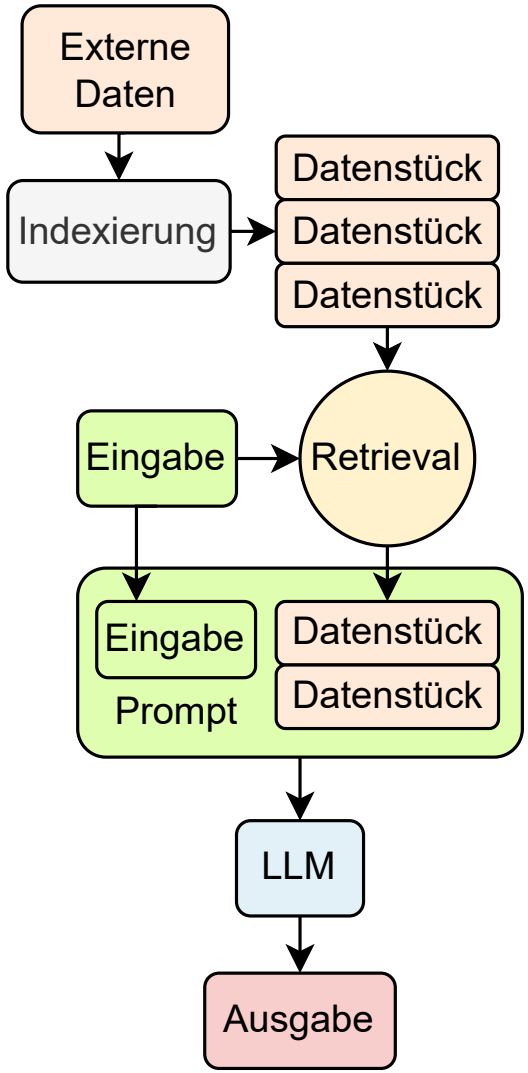
- Natürliche Sprachgenerierung (NLG) basierende Chatbots, aber keine LLMs.

Schnitt 3: Fine-Tuned LLM

- Pros:
  funktioniert gut in bestimmten Industrie-Bereichen wie z.B. Stadtgeographie.
- Cons:
  ohne einen kontextuellen Zusammenhang für jede spezifische Benutzereingabe.

Schnitt 4: RAG





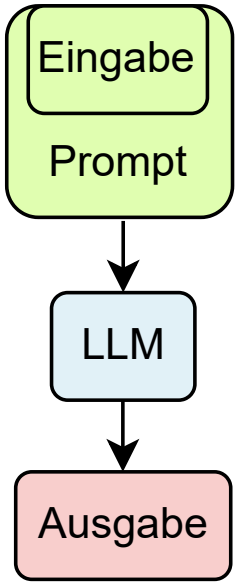
RAG kombiniert Information Retrieval und generative Modelle.

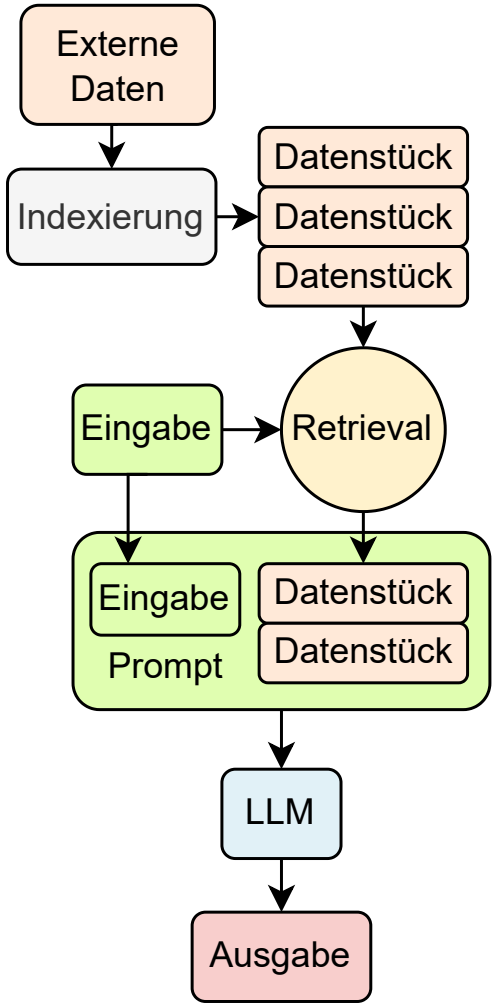
Vorteile von RAG:

- Aktualität: Mit RAG können LLMs auf aktuelle Daten zugreifen.
- Flexibilität: Mit RAG können LLMs auf externe Quellen, eben auch private oder geschützte Daten, zugreifen.
- Verbesserte Genauigkeit: Die Antwortgenauigkeit der LLMs wird erhöht, indem Halluzinationen vermieden werden.

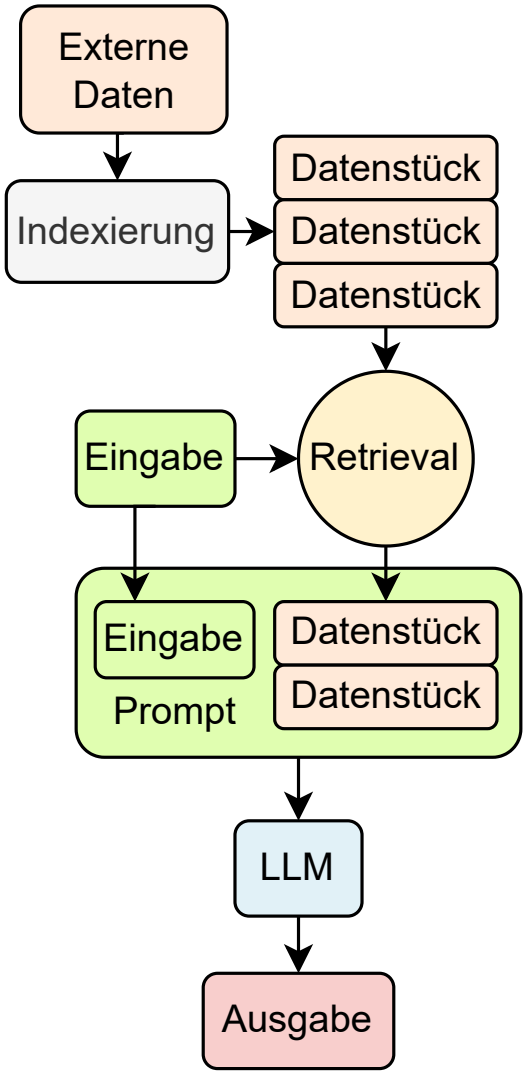
Outline

Naive RAG


Eingabe
Prompt
LLM
Ausgabe


Externe Daten
Indexierung
Datenstück
Datenstück
Datenstück
Eingabe
Retrieval
Eingabe
Prompt
Datenstück
Datenstück
LLM
Ausgabe

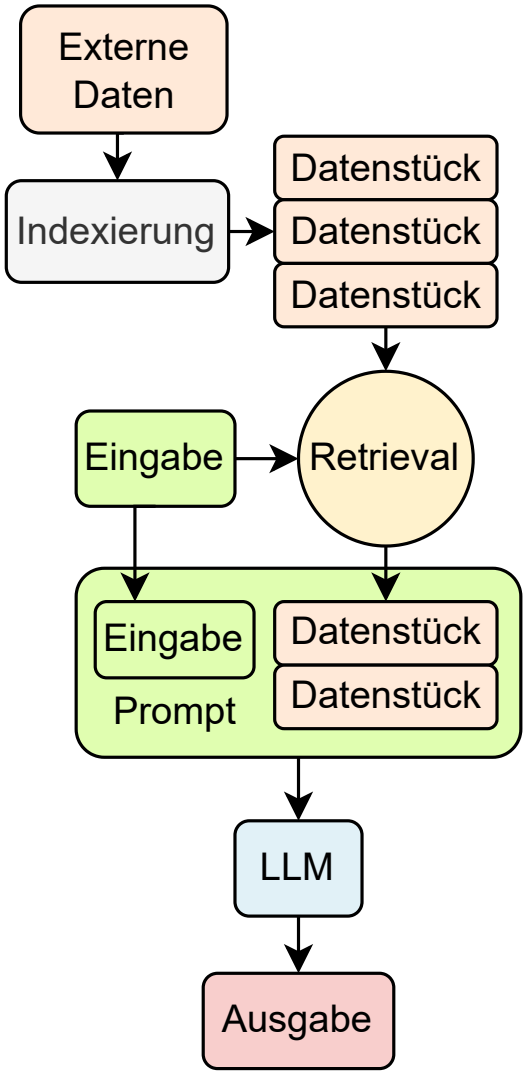
Naive RAG: Indexierung

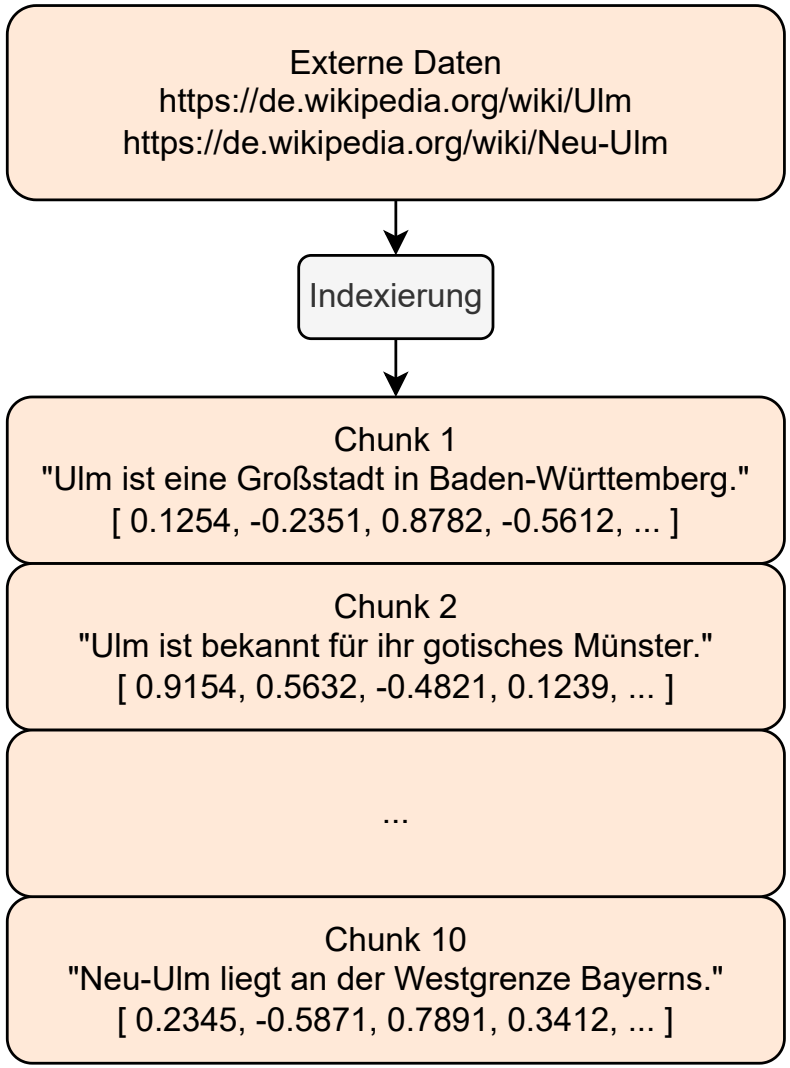


Indexierung:

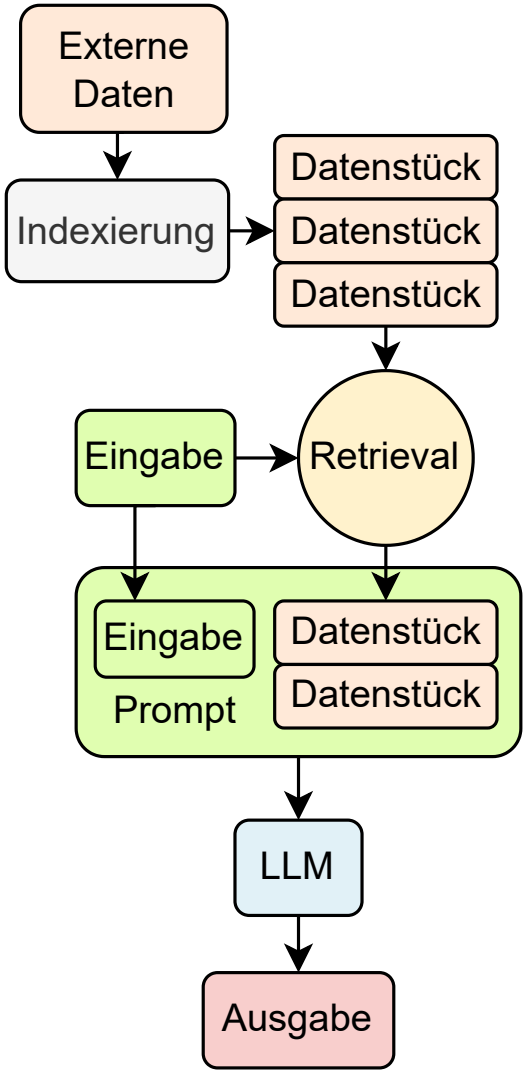
- Bei der Indexierung werden die Daten aus verschiedenen Quellen wie PDF und HTML in ein Klartextformat umgewandelt.
- Die Texte werden in kleinen Datenstücke (Chunks) unterteilt.
- Die Datenstücke werden dann mit einem Embedding-modell in Vektordarstellungen kodiert und gespeichert.

Indexierung: Beispiel



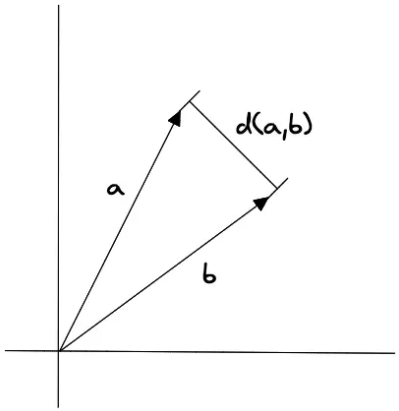


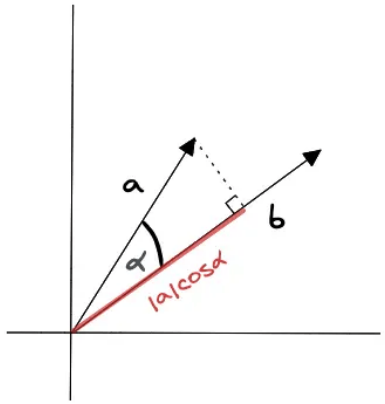
Naive RAG: Retrieval

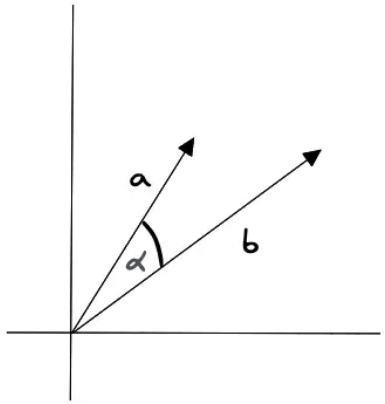


Retrieval:

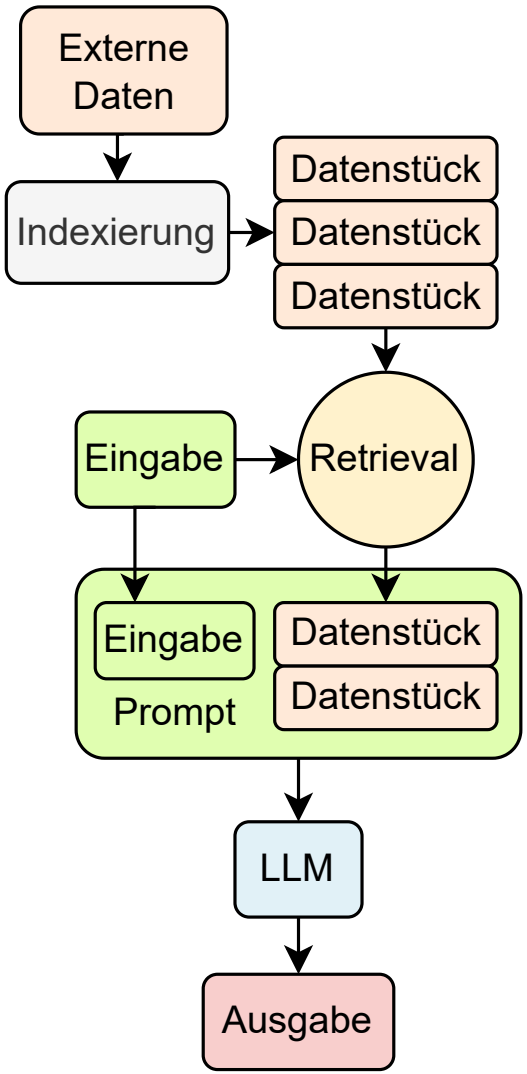
- Die Benutzereingabe wird auch in eine Vektordarstellung umgewandelt.
- Dann wird die ähnlichkeit zwischen dem Eingabe-Vektor und jedem Chunk-Vektor berechnet.
- Das System ruft die Top-K Datenstücke ab, die am ähnlichsten zur Anfrage sind.

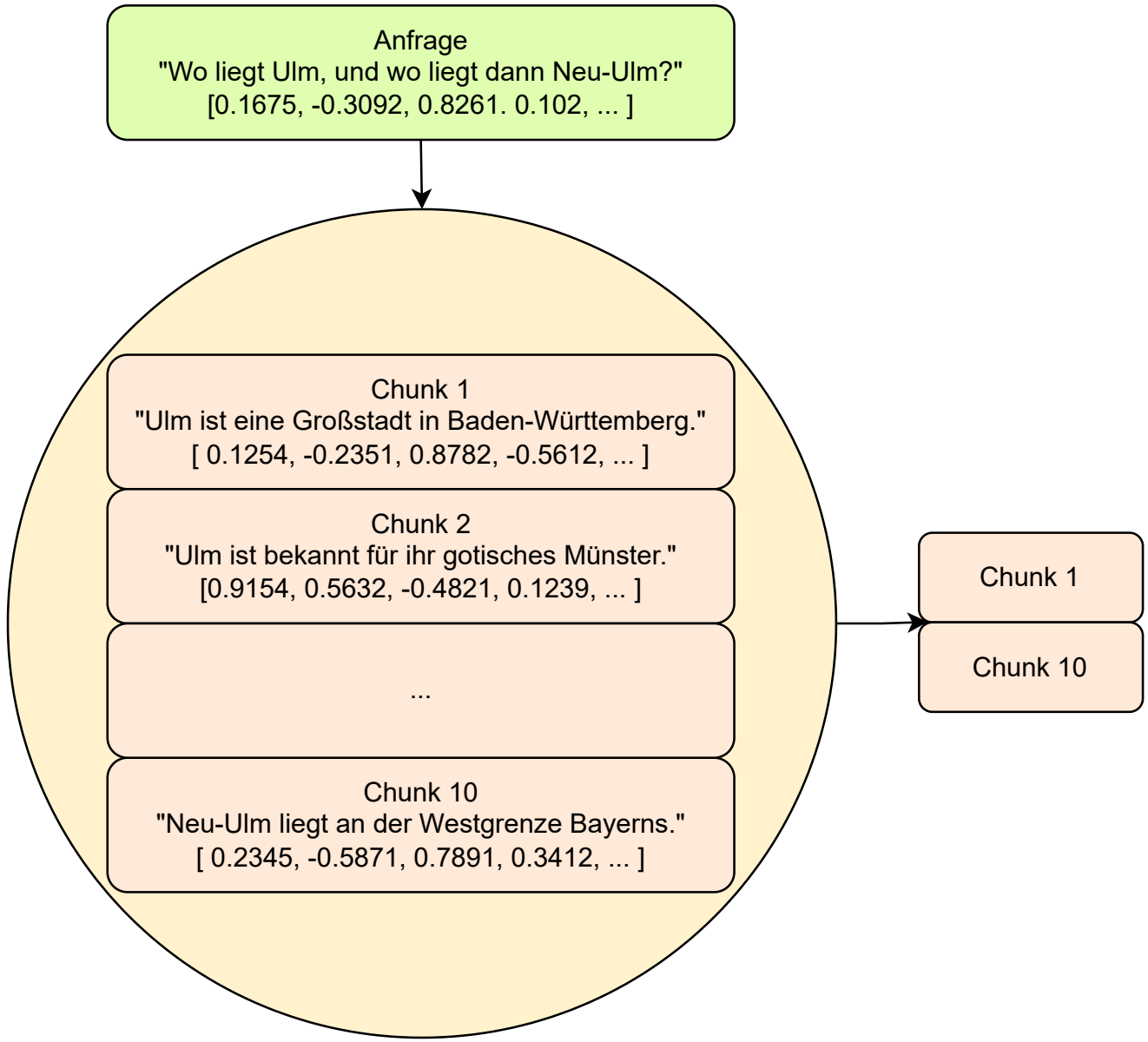




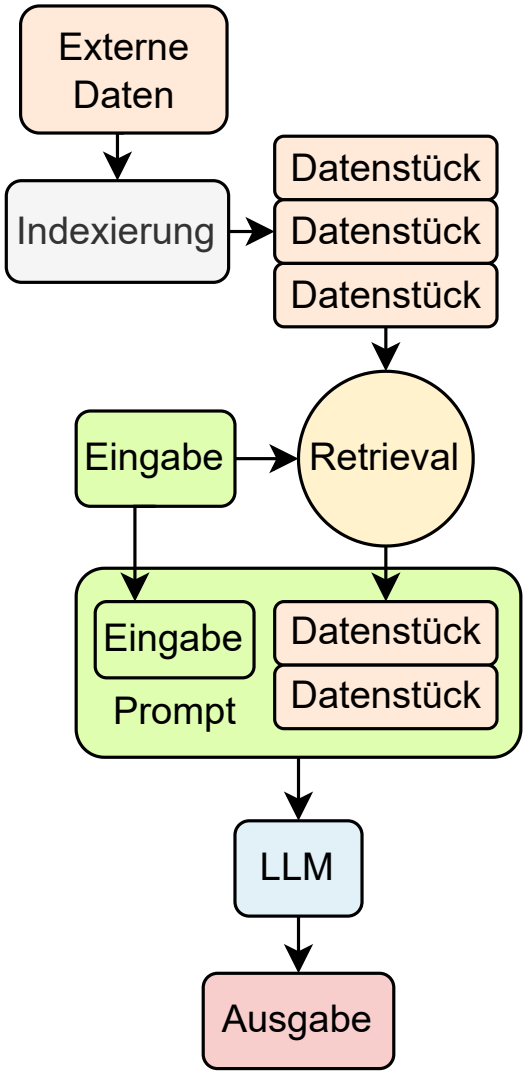


Retrieval: Beispiel





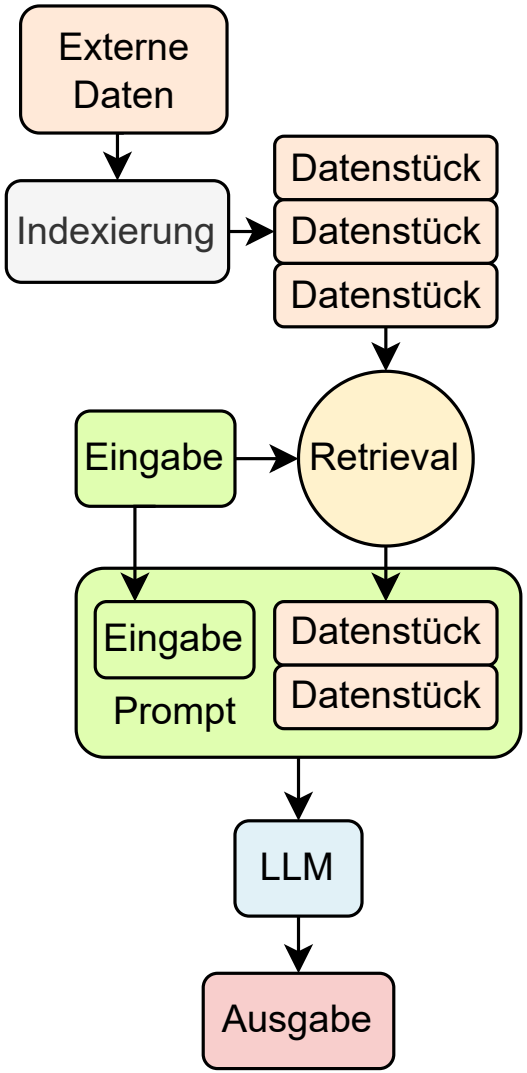
Naive RAG: Generierung

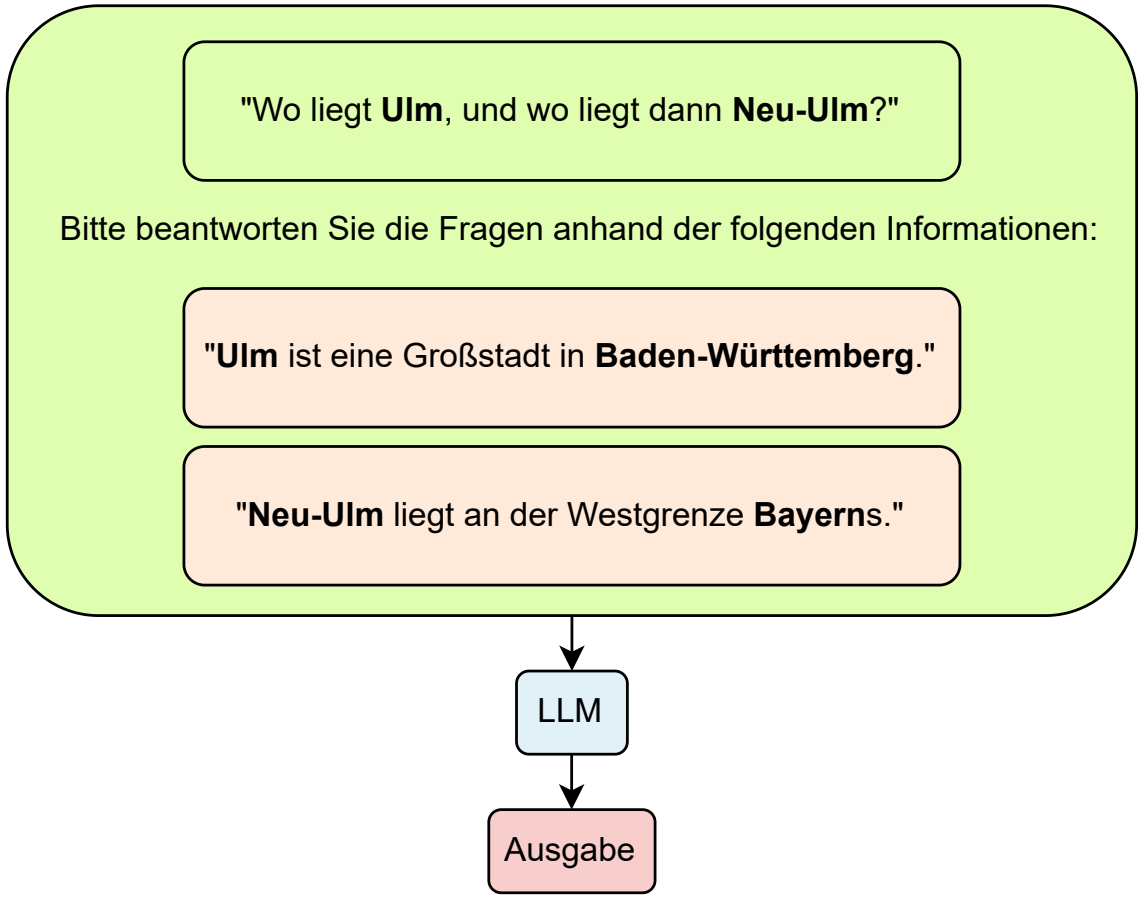


Generierung:

- Die gestellte Anfrage und die ausgewählten Datenstücke werden zu der neuen Eingabe (Prompt) für den LLM zusammengesetzt.
- Das Modell kann entweder auf eigene Kenntnisse zurückgreifen oder die in den Datenstücken enthaltenen Informationen nutzen, um zu antworten.

Generierung: Beispiel





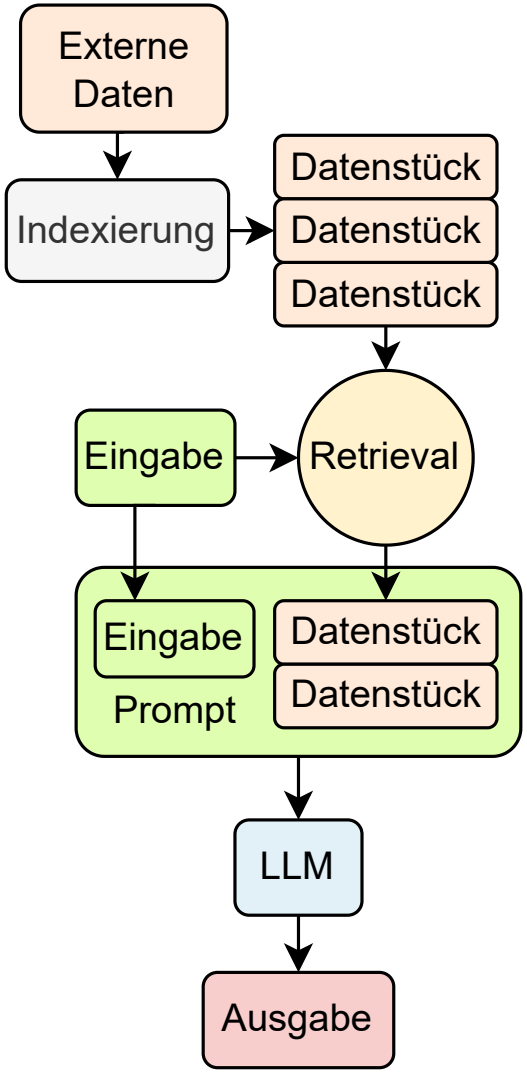
Naive RAG: Nachteile und Probleme

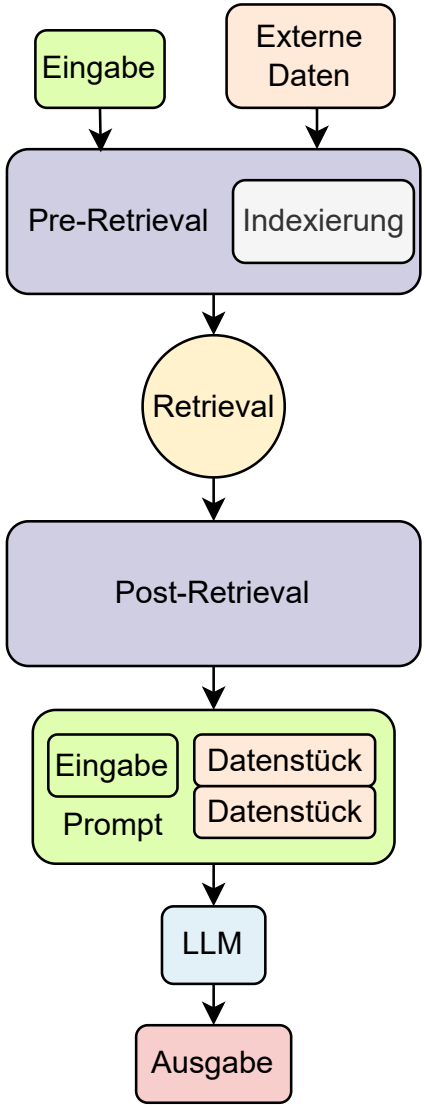
Gao et al. (2023)

- Problemen bei der **Retrieval-Genauigkeit**:
  Falsch ausgerichtete oder irrelevante Chunks werden ausgewählt und wichtige Informationen gehen verloren.
- **Halluzinationen**:
  Halluzinationen in Naive-RAG-Modellen können immer noch auftreten, wenn das Modell Texte anhand unvollständigen oder irrelevanten Informationen generiert.
- **Prompt-Integration**:
  Die Integration der abgerufenen Informationen in die verschiedenen Aufgaben führt manchmal zu disjunkten oder inkohärenten Ergebnissen.
  Wenn ähnliche Informationen aus mehreren Quellen abgerufen werden, führt dies zu repetitiven Antworten.
- stilistische und sprachliche **Konsistenz**; übermäßige **Verlassen** auf augmentierte Informationen; ...

Outline

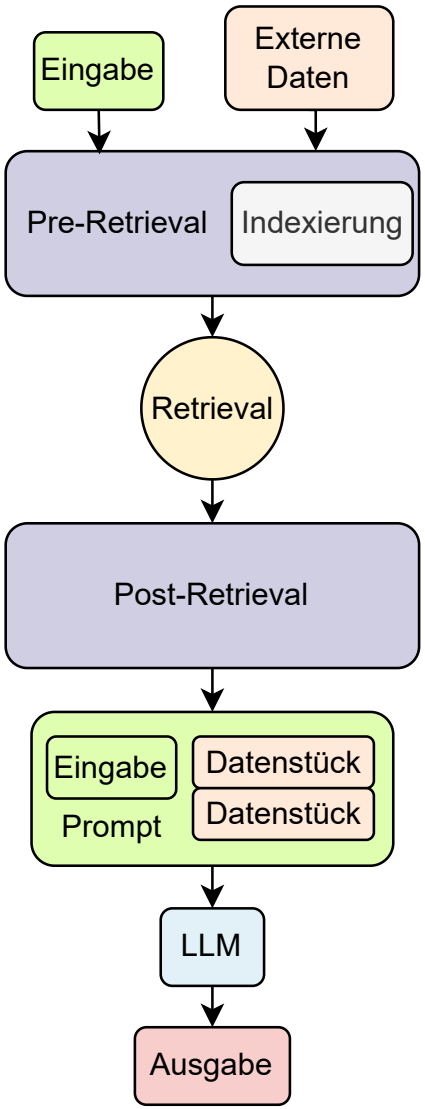
Advanced RAG





Advanced RAG konzentriert sich auf die Verbesserung der Retrieval-Qualität und verwendet **Pre-Retrieval** und **Post-Retrieval** Strategien.

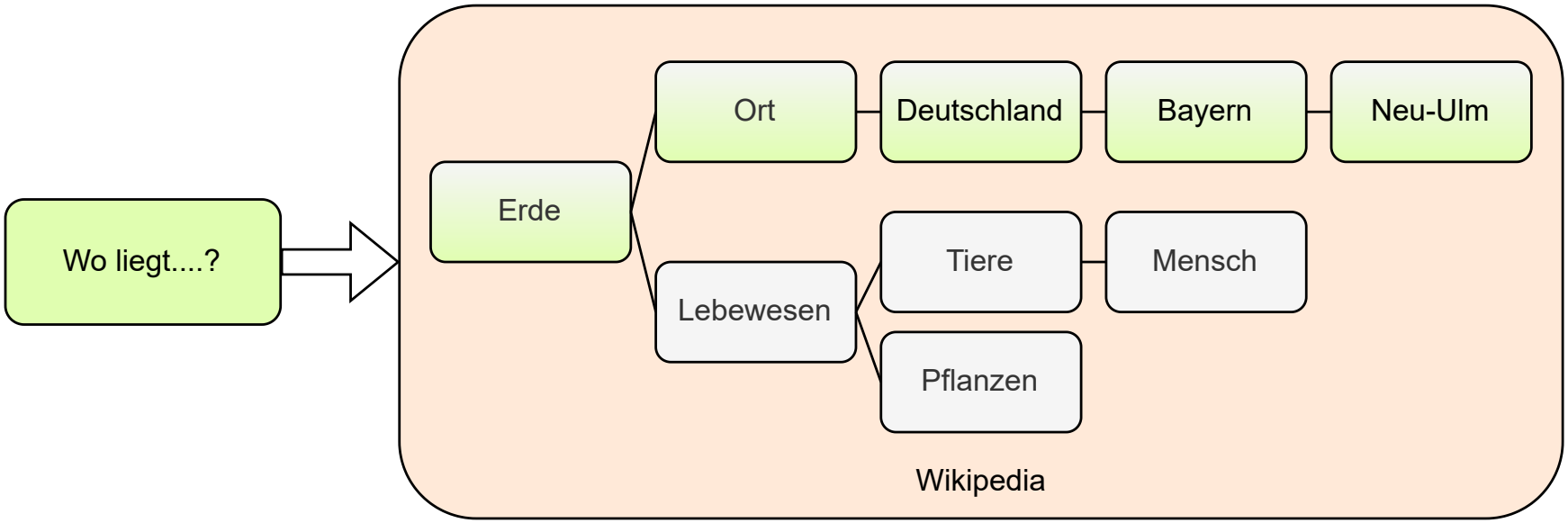
Advanced RAG: Pre-Retrieval



Im Pre-Retrieval liegt das Hauptziel in der Optimierung der Indexstruktur und der Benutzereingabe.
Dazu gehören folgende Strategien:

- Optimierung der **Indexstruktur** durch hierarchische Relation
- Verbesserung der Datengranularität durch **Kontextanreicherung**
- **Umschreiben** & **Umformen** von Anfragen

Hierarchischer Index



Verbesserung der Datengranularität

Ulm ist eine Großstadt mit 129.942 Einwohnern in Baden-Württemberg. Die Universitätsstadt liegt an der Donau am südöstlichen Rand der Schwäbischen Alb an der Grenze zu Bayern. Ulm ist nach dem Landesentwicklungsplan Baden-Württemberg eines von insgesamt 14 Oberzentren des Landes. Diese Beziehung wird auch als Doppelstadt oder Zweilandstadt bezeichnet, weil der eine Teil in Baden-Württemberg und der andere in Bayern liegt. Ulm ist die größte Stadt im Regierungsbezirk Tübingen und in der Region Donau-Iller, zu der auch Gebiete des bayerischen Regierungsbezirks Schwaben gehören.
Ulm, erstmals am 22. Juli 854 urkundlich genannt, war Königspfalz und Reichsstadt, ab 1802 bayerisch, seit 1810 württembergisch, nach 1945 württemberg-badisch und seit 1952 baden-württembergisch. Seit 1810 ist Ulm getrennt von seinem ehemaligen Gebiet rechts der Donau, das bei Bayern blieb und auf dem sich die Stadt Neu-Ulm entwickelte. Die Stadt ist bekannt für ihr gotisches Münster, dessen Kirchturm mit 161,53 Metern der höchste der Welt ist.

Verbesserung der Datengranularität

1. Frage: Wo liegt Ulm?

LLM: In BW liegt die Stadt Ulm.

2. Frage: Ist Ulm immer eine baden-württembergische Stadt?

LLM: Ja. $\perp \rightarrow$ Halluzination

Verbesserung der Datengranularität

Ulm ist eine Großstadt mit 129.942 Einwohnern in **Baden-Württemberg**.

...

Ulm, erstmals am 22. Juli 854 urkundlich genannt, war Königspfalz und Reichsstadt, ab 1802 **bayerisch**, seit 1810 württembergisch, nach 1945 württemberg-badisch und seit 1952 **baden-württembergisch**.

Kontextanreicherung:

- Kontexte, die neben dem ersten Zieltext liegen, werden ebenfalls weitergegeben, um die Datengranularität zu erhöhen.
- Satzfenster. In jedem Chunk: Satz $\rightarrow$ Absatz

Prompt: Ist Ulm immer eine baden-württembergische Stadt, ggb. 1. Satz und 2. Satz?

LLM: Nein.

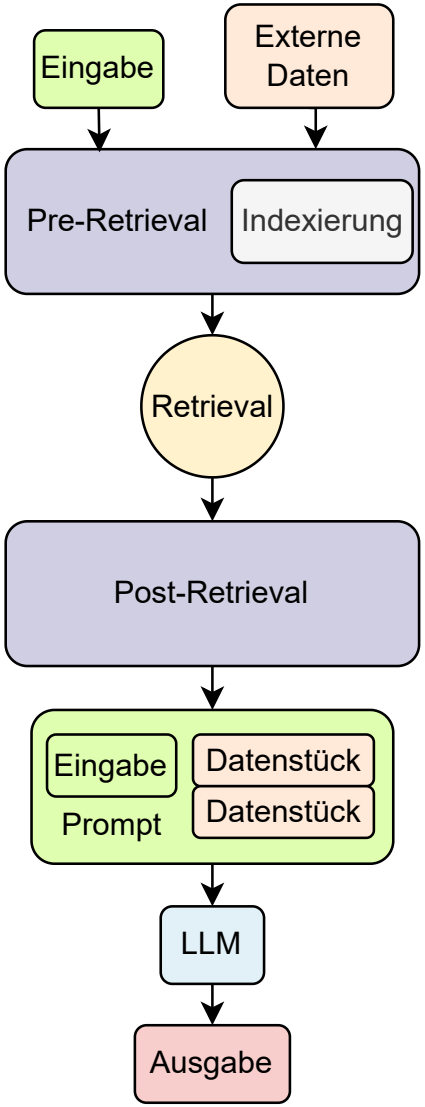
Umschreiben & Umformen von Anfragen

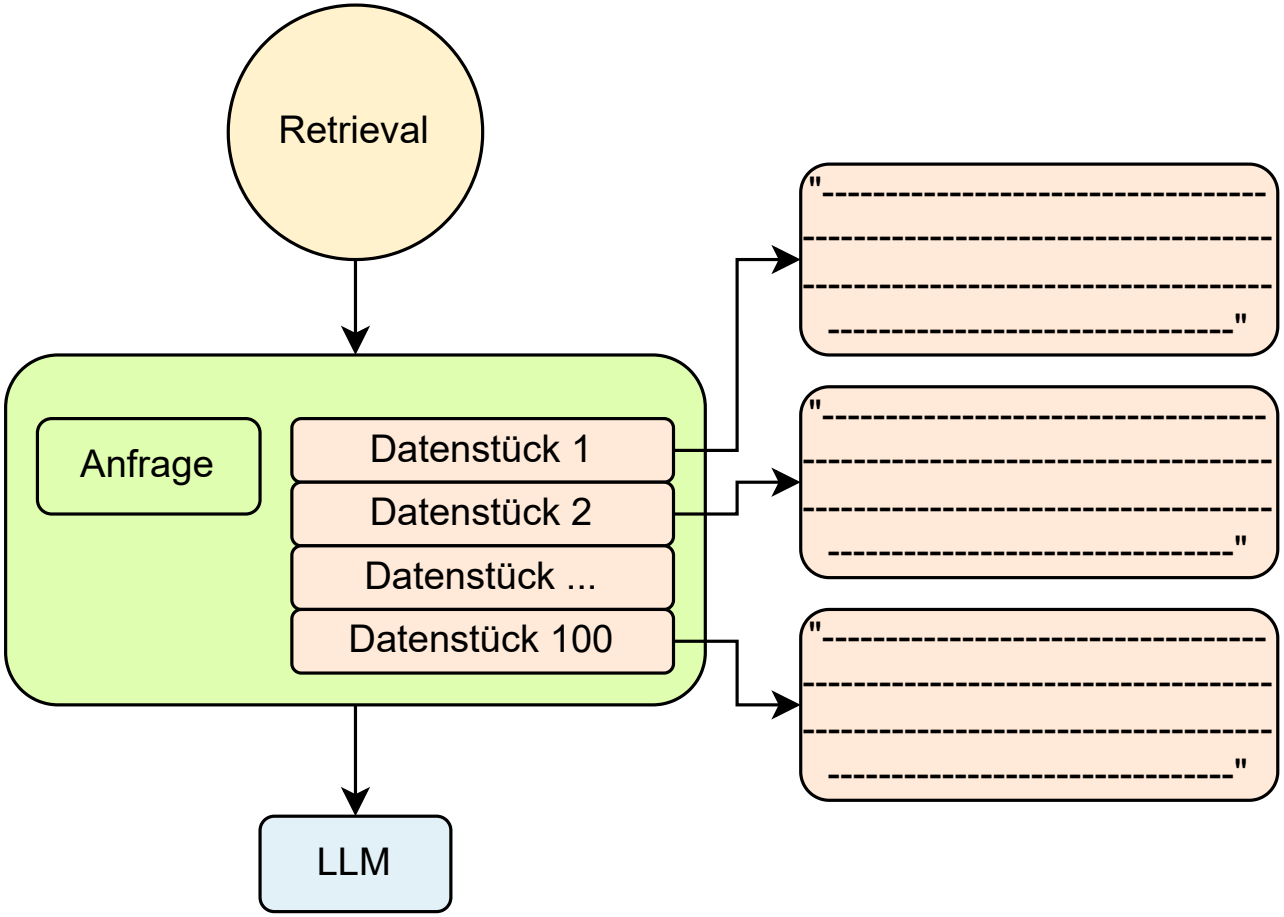
Frage: Wo liegt Neu-Ulm und wo liegt Alt-Ulm? $\rightarrow$ unklar

Umformuliert:

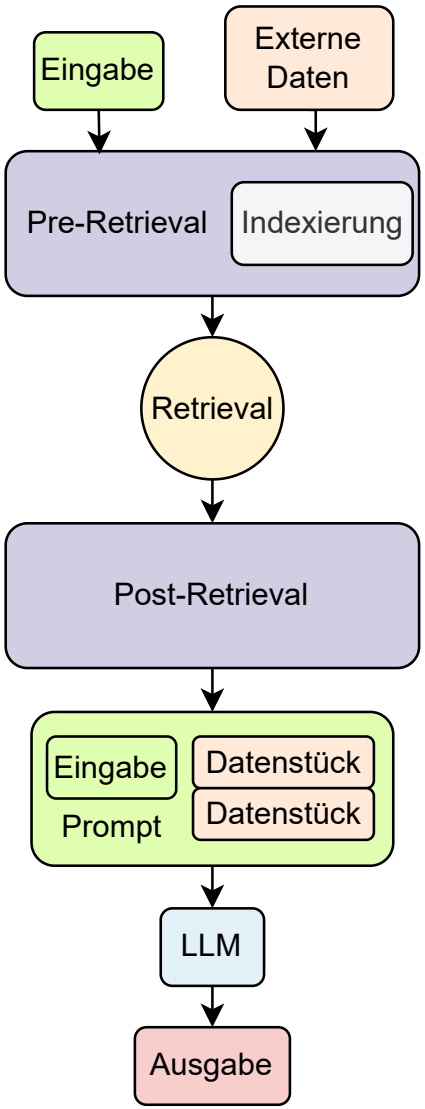
- 1. Unterfrage: Wo liegt die Stadt Neu-Ulm?
- 2. Unterfrage: Wie verhält sich dies zur historischen Stadt Ulm (Alt-Ulm)?
- 3. Unterfrage: Wo liegt die Stadt Ulm in der Vergangenheit?
- 4. Unterfrage: Existiert die Stadt Ulm noch?
- 5. Unterfrage: Wo liegt die Stadt Ulm jetzt?

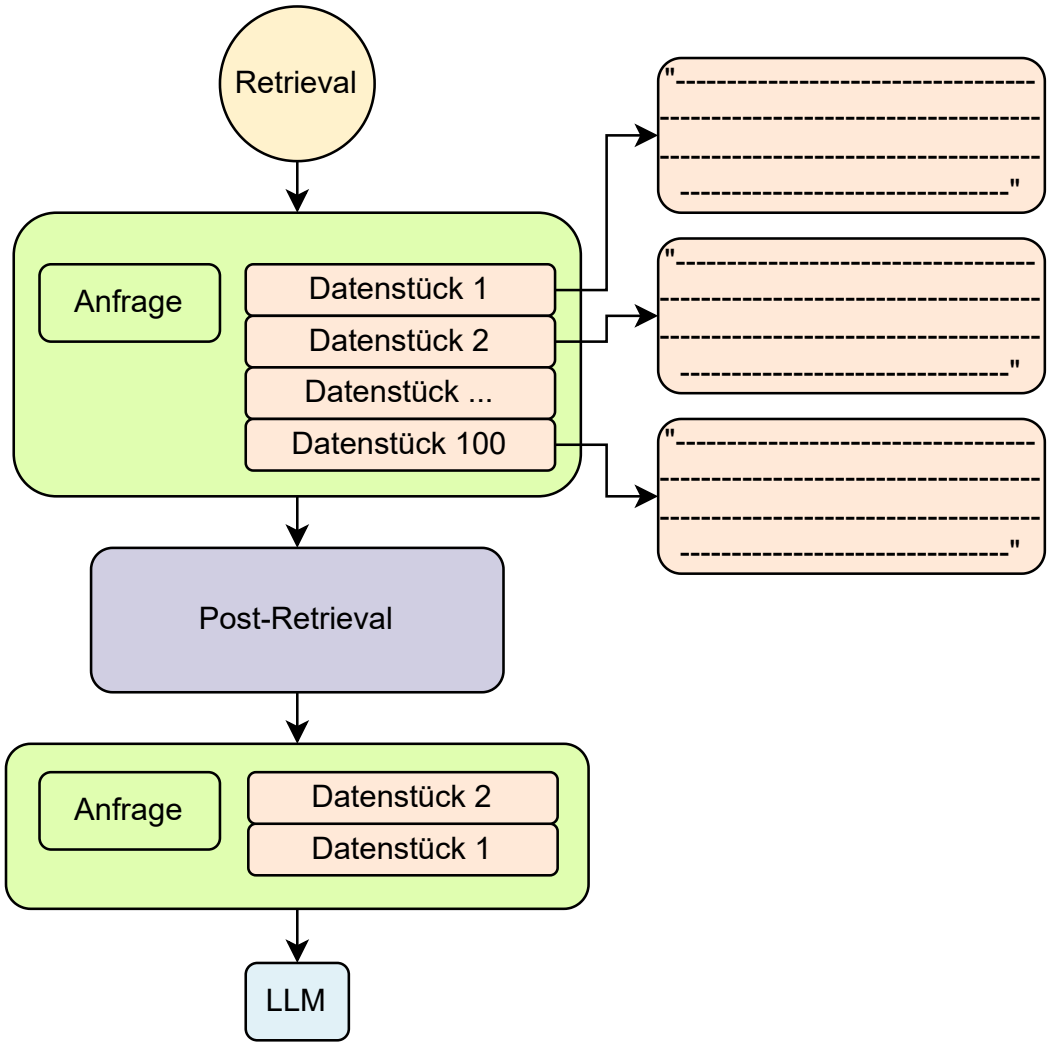
Advanced RAG: Post-Retrieval



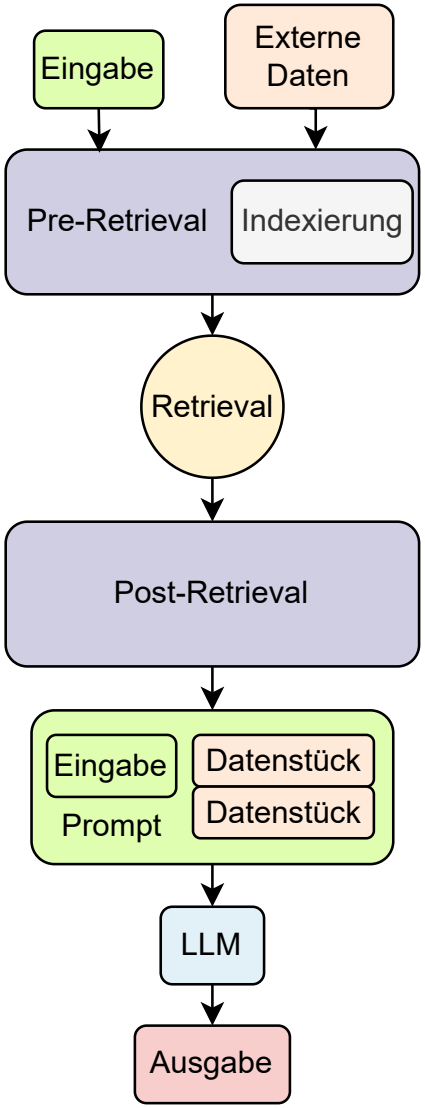


Advanced RAG: Post-Retrieval



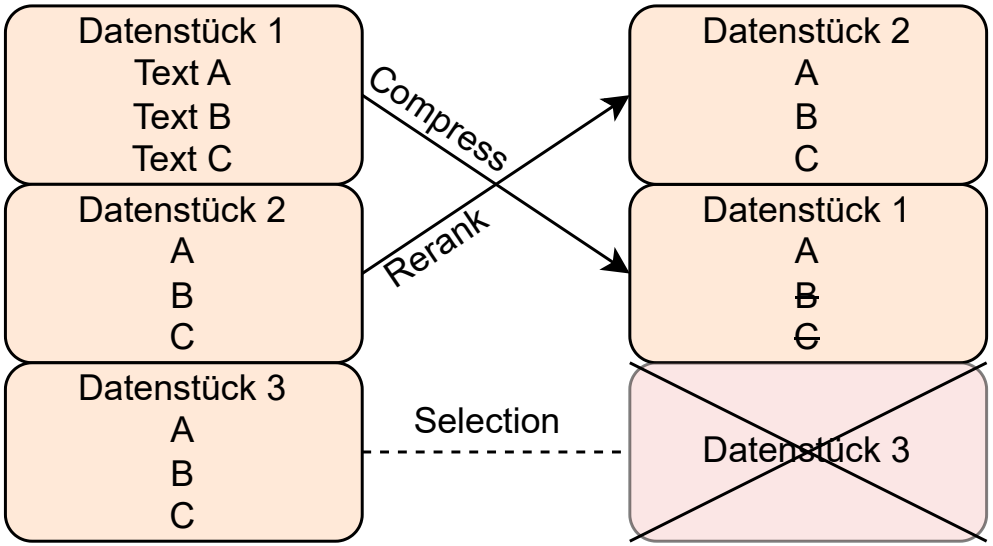


Advanced RAG: Post-Retrieval



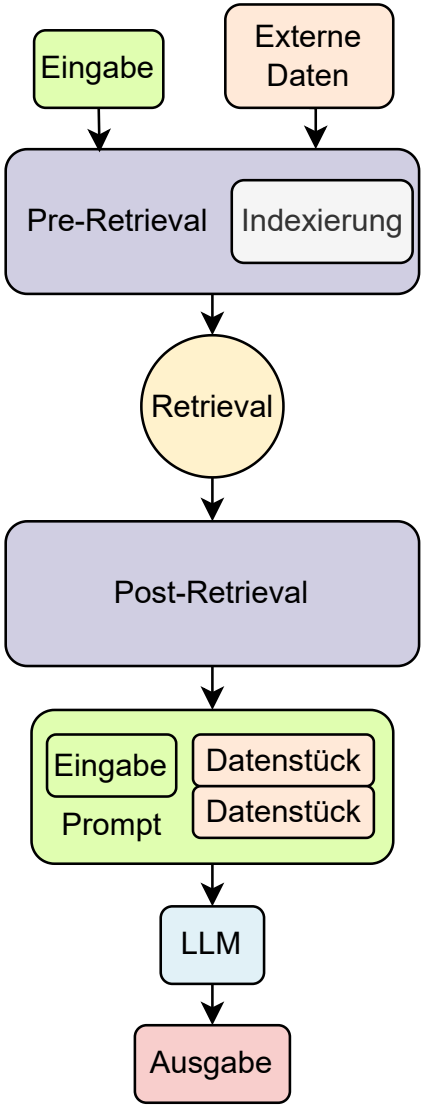
Die direkte Eingabe aller relevanten Dokumente in LLMs kann zu einer Informationsüberlastung führen. Daher sind die folgenden Post-Retrieval-Methoden hilfreich:

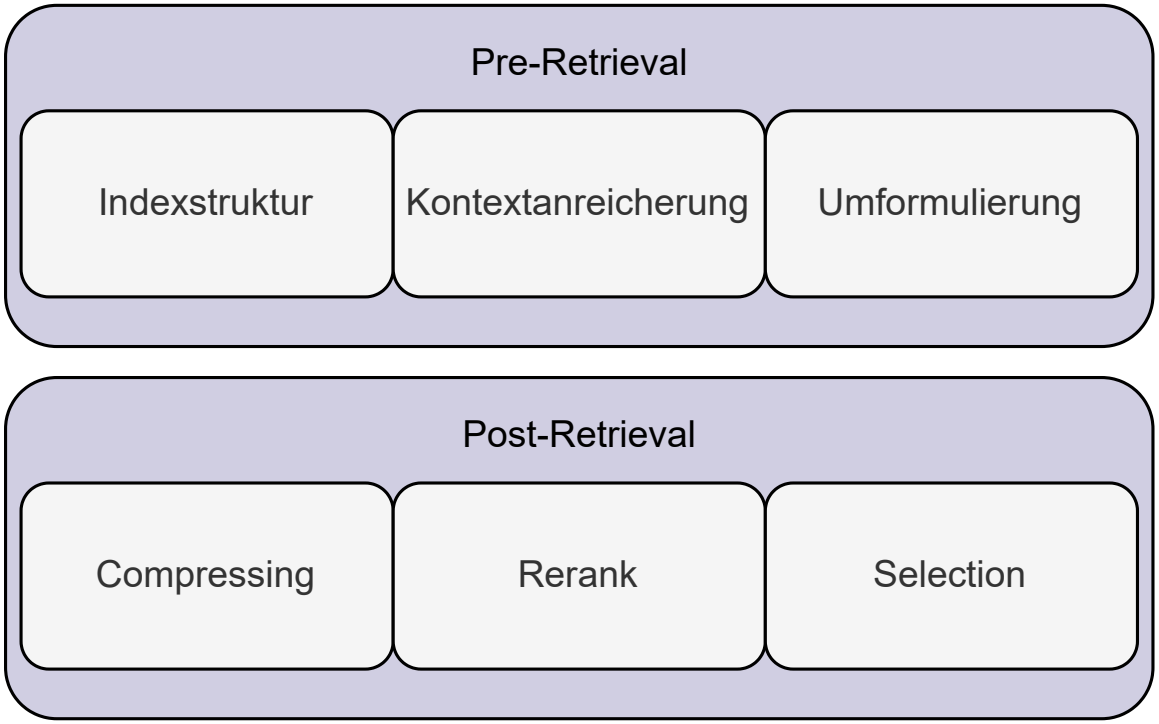
- Chunk **Compressing**
- Chunk **Rerank**
- Chunk **Selection**



Outline

Module in Advanced RAG



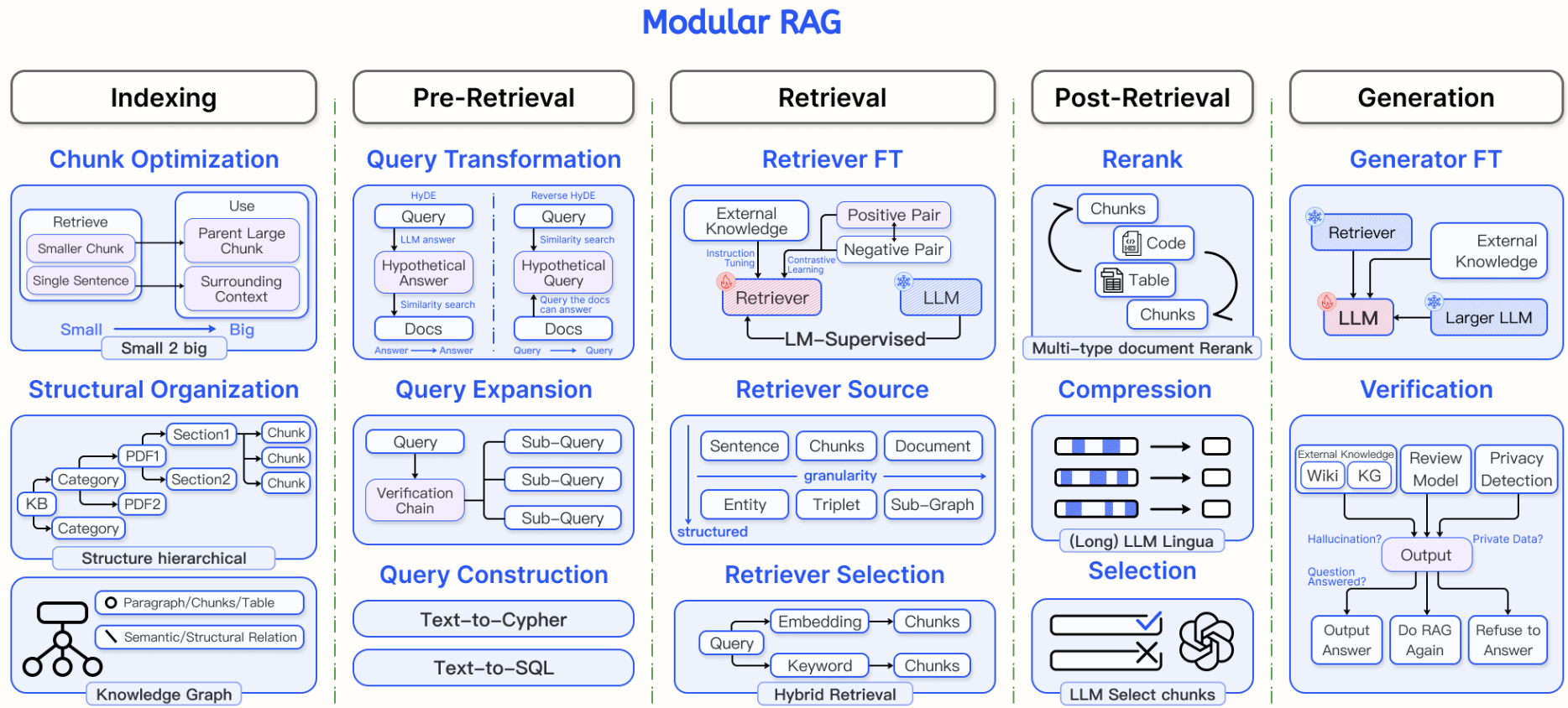


Modular RAG: Transforming RAG Systems into LEGO-like Reconfigurable Frameworks

Yunfan Gao, Yun Xiong, Meng Wang, Haofen Wang

Gao et al. (2024)

Modular RAG


Modular RAG
Indexing
Chunk Optimization
Retrieve
Smaller Chunk
Single Sentence
Use
Parent Large Chunk
Surrounding Context
Small
Big
Small 2 big
Structural Organization
KB
Category
PDF1
PDF2
Category
Section1
Section2
Chunk
Chunk
Chunk
Structure hierarchical
Paragraph/Chunks/Table
Semantic/Structural Relation
Knowledge Graph
Pre-Retrieval
Query Transformation
HyDE
Query
LLM answer
Hypothetical Answer
Similarity search
Docs
Answer → Answer
Reverse HyDE
Query
Similarity search
Hypothetical Query
Query the docs can answer
Docs
Query → Query
Query Expansion
Query
Verification Chain
Sub-Query
Sub-Query
Sub-Query
Query Construction
Text-to-Cypher
Text-to-SQL
Retrieval
Retriever FT
External Knowledge
Instruction Tuning
Positive Pair
Contrastive Learning
Negative Pair
Retriever
LLM
LM-Supervised
Retriever Source
Sentence
Chunks
Document
granularity
Entity
Triplet
Sub-Graph
structured
Retriever Selection
Query
Embedding
Chunks
Keyword
Chunks
Hybrid Retrieval
Post-Retrieval
Rerank
Chunks
Code
Table
Chunks
Multi-type document Rerank
Compression
(Long) LLM Lingua
Selection
LLM Select chunks
Generation
Generator FT
Retriever
External Knowledge
LLM
Larger LLM
Verification
External Knowledge
Wiki
KG
Review Model
Privacy Detection
Hallucination?
Output
Private Data?
Question Answered?
Output Answer
Do RAG Again
Refuse to Answer

Outline

RAG with LangChain: Aufteilung und Indexierung

```python
from langchain_text_splitters import
                                    RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,  # chunk size (characters)
    chunk_overlap=200,  # chunk overlap (characters)
    add_start_index=True,  # track index in original
                                        document
)

all_splits = text_splitter.split_documents(docs)
print(f"Split blog post into {len(all_splits)} sub-documents
                                    .")

document_ids = vector_store.add_documents(documents=
                                    all_splits)
print(document_ids[:3])
```

Split blog post into 66 sub-documents.
'07c18af6-ad58-479a-bfb1-d508033f9c64', ...

RAG mit LangChain: Prompting

```python
from langchain import hub

prompt = hub.pull("rlm/rag-prompt")

example_messages = prompt.invoke(
    {"context": "(context goes here)", "question": "(
                                    question goes here)"}
).to_messages()

assert len(example_messages) == 1
print(example_messages[0].content)
```

You are an assistant for question-answering tasks. Use the following pieces of retrieved context to answer the question. If you don't know the answer, just say that you don't know. Use three sentences maximum and keep the answer concise.
Question: (question goes here)
Context: (context goes here)
Answer:

RAG mit LangChain: State and Nodes

```python
from langchain_core.documents import Document
from typing_extensions import List, TypedDict

class State(TypedDict):
    question: str
    context: List[Document]
    answer: str
def retrieve(state: State):
    retrieved_docs = vector_store.similarity_search(state["
                                        question"])
    return {"context": retrieved_docs}
def generate(state: State):
    docs_content = "\n\n".join(doc.page_content for doc in
                                        state["context"])
    messages = prompt.invoke({"question": state["question"],
                                         "context": docs_content})
    response = llm.invoke(messages)
    return {"answer": response.content}
```

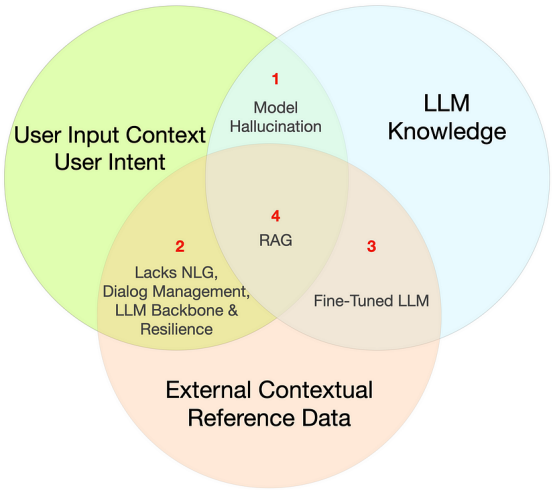
RAG mit LangChain: Kontrollfluss

```
from langgraph.graph import START, StateGraph

graph_builder = StateGraph(State).add_sequence([retrieve,
                                                generate])
graph_builder.add_edge(START, "retrieve")
graph = graph_builder.compile()

result = graph.invoke({"question": "What is Task
                                   Decomposition?"})

print(f'Context: {result["context"]}\n\n')
print(f'Answer: {result["answer"]}')
```

Context: [Document(id='a42dc78b-8f76-472a-9e25-180508af74f3', metadata='source': 'https://...']
Answer: Task decomposition is a technique used to break down complex tasks into...

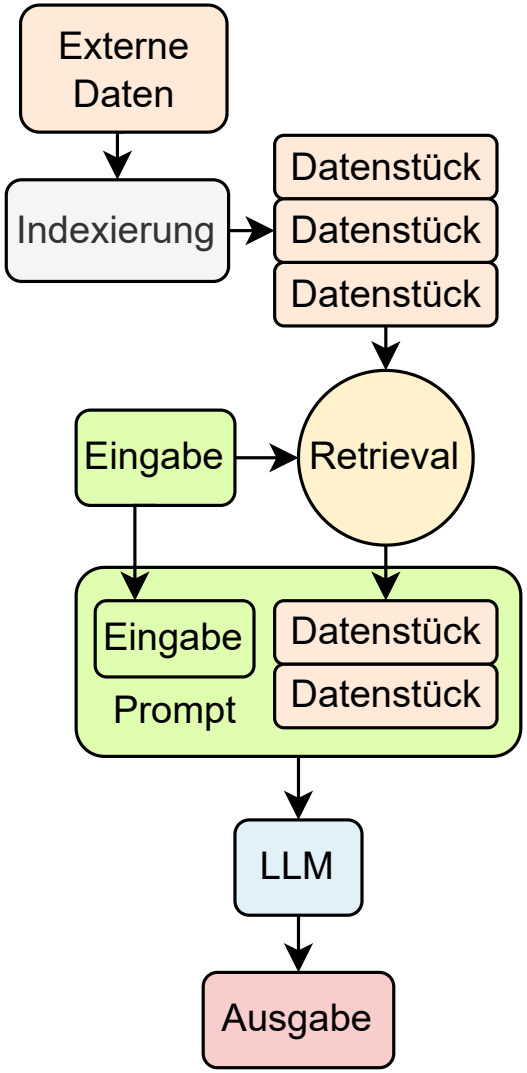
Outline

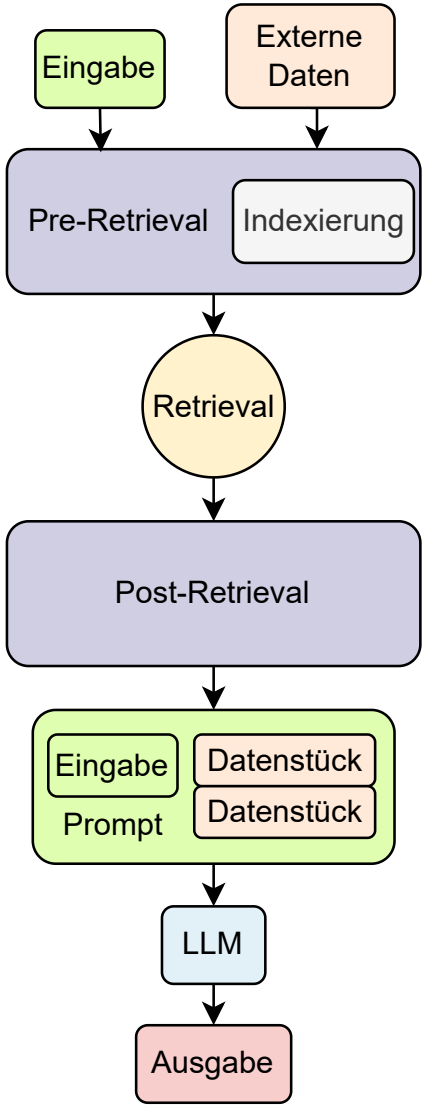
Zusammenfassung



$$\text{LLM Trainingsdaten} \xrightarrow[\text{Benutzereingaben}]{\text{Externe Referenzdaten}} \text{RAG}$$

Zusammenfassung





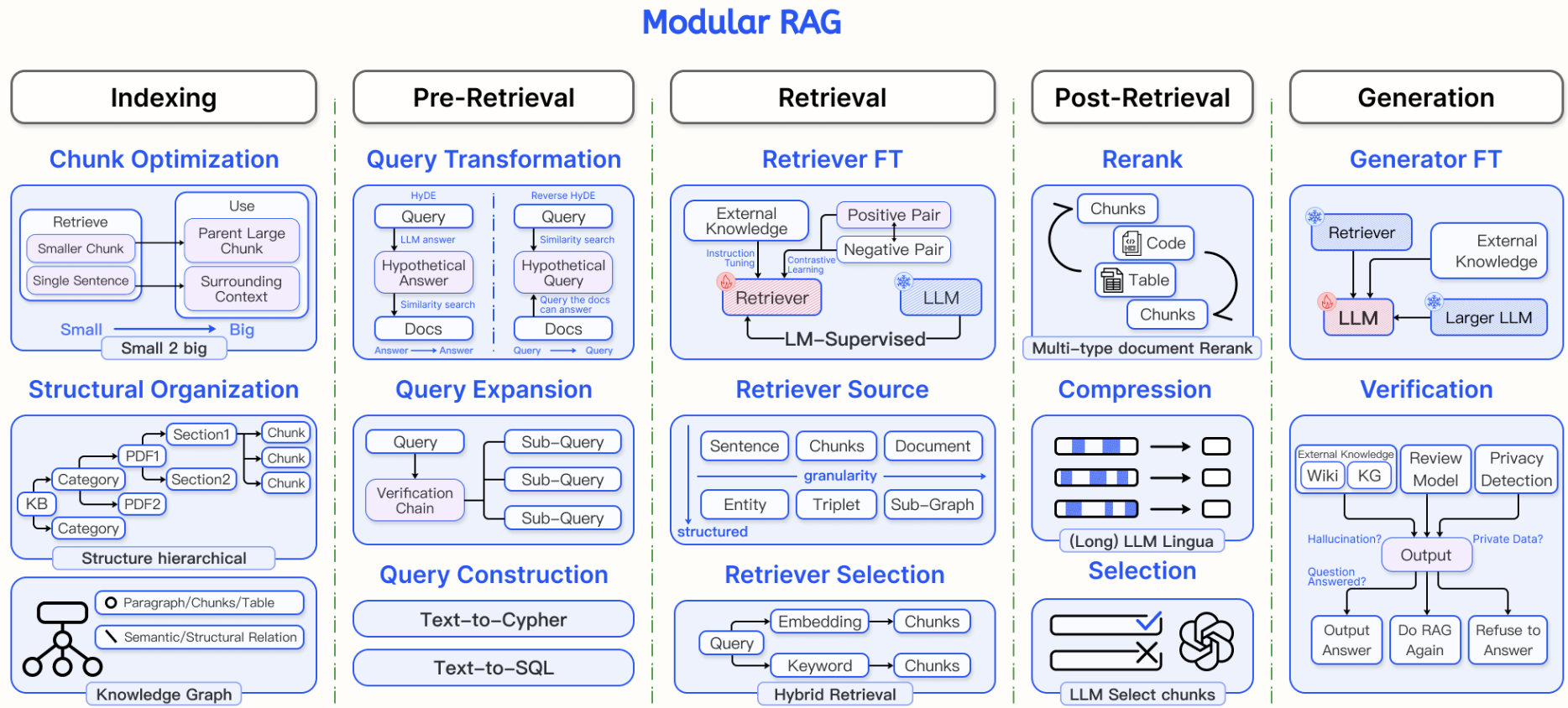
Naive RAG

Pre-Retrieval

Post-Retrieval

Advanced RAG

Zusammenfassung



So, What Are Some Techniques to Enhance RAG?

Metadata Filtering

Main Improvement Areas:

Retrieval Augmentation

What & How:

- Automatically recognizing scope of retrieval and filtering the documents using their metadata.
- Attaching metadata like dates, purpose, chapter summaries, etc., to chunks.

Query Engineering & Fusion

Main Improvement Areas:

Pre-retrieval Query Processing
Retrieval Augmentation
Generation

What & How:

- Editing, rewriting, or spliting user queries to remove errors and biases or to retrieve specific information.
- Expanding user queries into multiple diverse perspectives and running parallel vector searches (multi-query approach).
- Ranking/merging multiple responses and aligning the final response with both explicit information and implicit user intentions.

Task Adapter

Main Improvement Areas:

Retrieval

What & How:

- Automating retrieval of prompts for zero-shot task inputs from a pre-constructed data pool, enhancing universality across tasks and models.
- Utilizing LLM as a few-shot query generator and creates task-specific retrievers based on generated data.
- Leveraging LLM's generalization capability to develop task-specific retrievers with minimal examples.

Memory

Main Improvement Areas:

Retrieval Generation

What & How:

- Identifying LLM memories most similar to the current input.
- Utilizing a retrieval-enhanced generator for iterative memory creation and self-improvement.
- Aligning the output with the target

Information or Context Compression

Main Improvement Areas:

Augmentation Generation

What & How:

- Condensing and compressing a vast amount of relevant information extracted from extensive knowledge bases.
- Filtering the contents and keeping

Extra Generation & Predicting

Main Improvement Areas:

Augmentation Generation

What & How:

- Utilizing LLM to generate necessary context instead of direct retrieval.
- Addressing redundancy and noise in retrieved content.

Zusammenfassung

intentions.

Memory

Main Improvement Areas:

Retrieval Generation

What & How:

- Identifying LLM memories most similar to the current input.
- Utilizing a retrieval-enhanced generator for iterative memory creation and self-improvement.
- Aligning the output with the target data distribution during this reasoning process.

Information or Context Compression

Main Improvement Areas:

Augmentation Generation

What & How:

- Condensing and compressing a vast amount of relevant information extracted from extensive knowledge bases.
- Filtering the contents and keeping only the most relevant points before passing to LLM.

Extra Generation & Predicting

Main Improvement Areas:

Augmentation Generation

What & How:

- Utilizing LLM to generate necessary context instead of direct retrieval.
- Addressing redundancy and noise in retrieved content.

Routing | Multiple Indexing Structures | Hypothetical Document

Zusammenfassung

Routing

Main Improvement Areas:

Pre-retrieval Query Processing

Retrieval

What & How:

- Flexibly alternating between sources diverse in domain, language, and format based on situation.
- Determining subsequent action to user queries, including summarization, specific database searches (vector, graph, or relational databases, or a hierarchy of indices), or merging different paths.

Multiple Indexing Structures

Main Improvement Areas:

Pre-retrieval Data Indexing

Retrieval

What & How:

- Introducing graph structures to enhance retrieval by leveraging nodes and their relationships.
- Creating multi-index paths to increase efficiency.

Hypothetical Document Embeddings (HyDE)

Main Improvement Areas:

Retrieval Augmentation

What & How:

- Creating a hypothetical document (answer) to a query and retrieve contexts similar to the hypothetical document (answer) based on its embeddings (HyDE).
- Emphasizing the similarities between embeddings of potential documents (answers) and those of real answers.

Search Module & Hybrid Search

Main Improvement Areas:

Retrieval

What & How:

- Implementing direct searches on additional data sources, such as search engines, SQL/No-SQL/graph databases, or user-specified texts or tables.
- Integrating results with those based on semantic search from vector databases.

Adaptive Retrieval & Self-RAG

Main Improvement Areas:

Retrieval Augmentation Generation

What & How:

- Enabling LLMs to determine when to search for necessary information, similar to how an agent uses tools.
- Evaluating relevance and level of support of retrieved contexts.
- Critiquing and assessing quality of final output.

Retrieval-augmented Reasoning: RAT & RAGAR

Main Improvement Areas:

Retrieval Augmentation Generation

What & How:

- Merging the concepts of RAG with Chain of Thought, enabling the system to logically reason in a certain direction and retrieve relevant contexts (RAT, Retrieval-augmented Thought [4]).
- Incorporating both Chain of RAG and Tree of RAG alongside Chain of Verification steps against the most current external web resources for multimodal fact-checking (RAGAR, RAG-augmented reasoning [5]).

Thanks for your attention!

References I

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Yunfan Gao, Yun Xiong, Meng Wang, and Haofen Wang. 2024. Modular rag: Transforming rag systems into lego-like reconfigurable frameworks. *arXiv preprint arXiv:2407.21059*.

appliedAI Initiative GmbH. 2024. Retrieval-augmented Generation Realized: Strategic & Technical Insights for Industrial Applications

Cobus Greyling @ Kore.ai, www.cobusgreyling.com

© LangChain, Inc.

© LEGO Group