Applications of LLM – Part I: Retrieval Augmented Generation (RAG)

Week 9 - LGT

Dr Lin Gui
Lin.1.gui@kcl.ac.uk

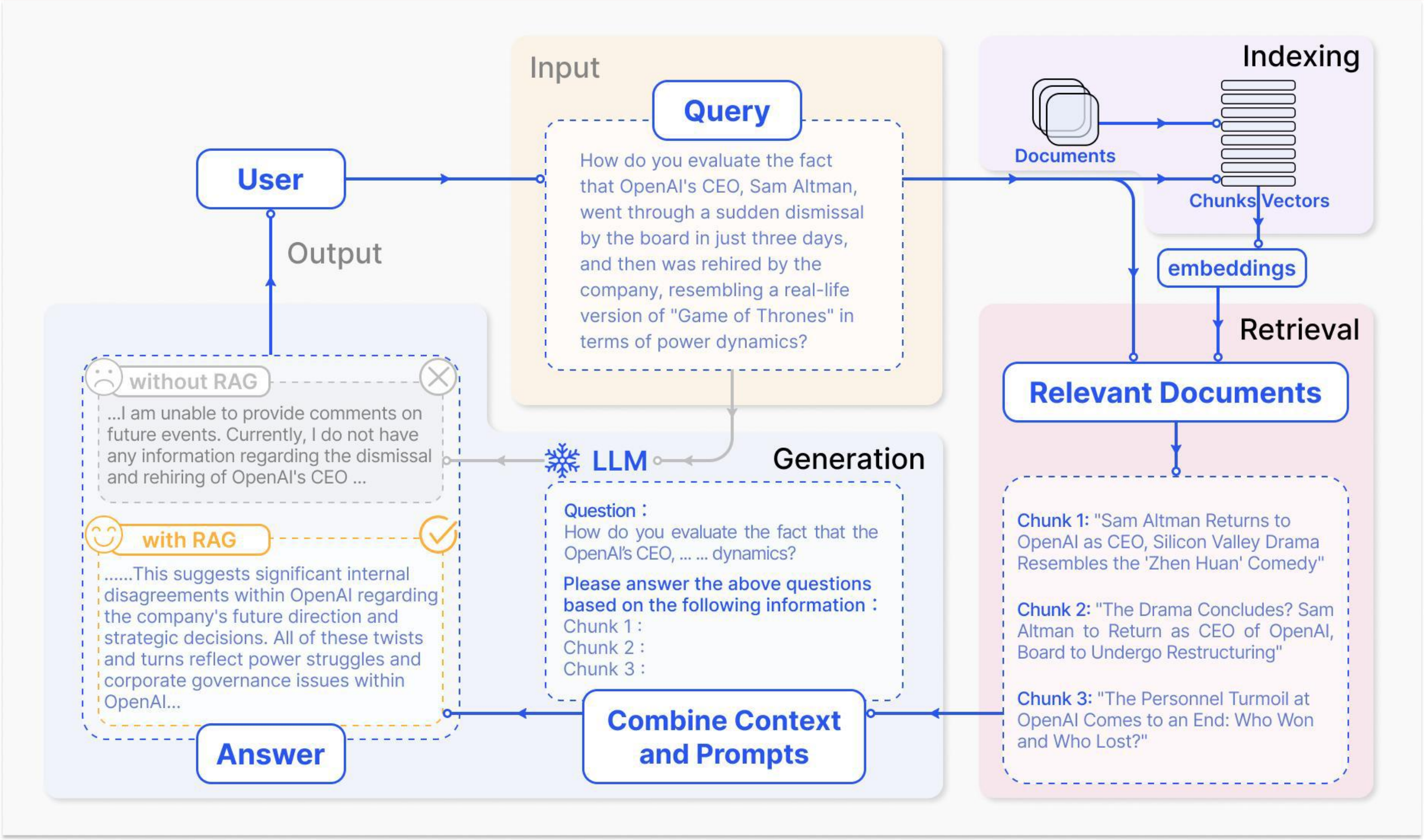
Learning outcomes

- By the end of this topic, you will be able to:
  - Understand the core concepts of Retrieval-Augmented Generation and how it differs from standard LLM approaches.
  - Build and configure a basic RAG pipeline using embeddings, retrievers, and generators.
  - Evaluate and optimize RAG performance through effective data preparation, chunking, and retrieval strategies.

Contents

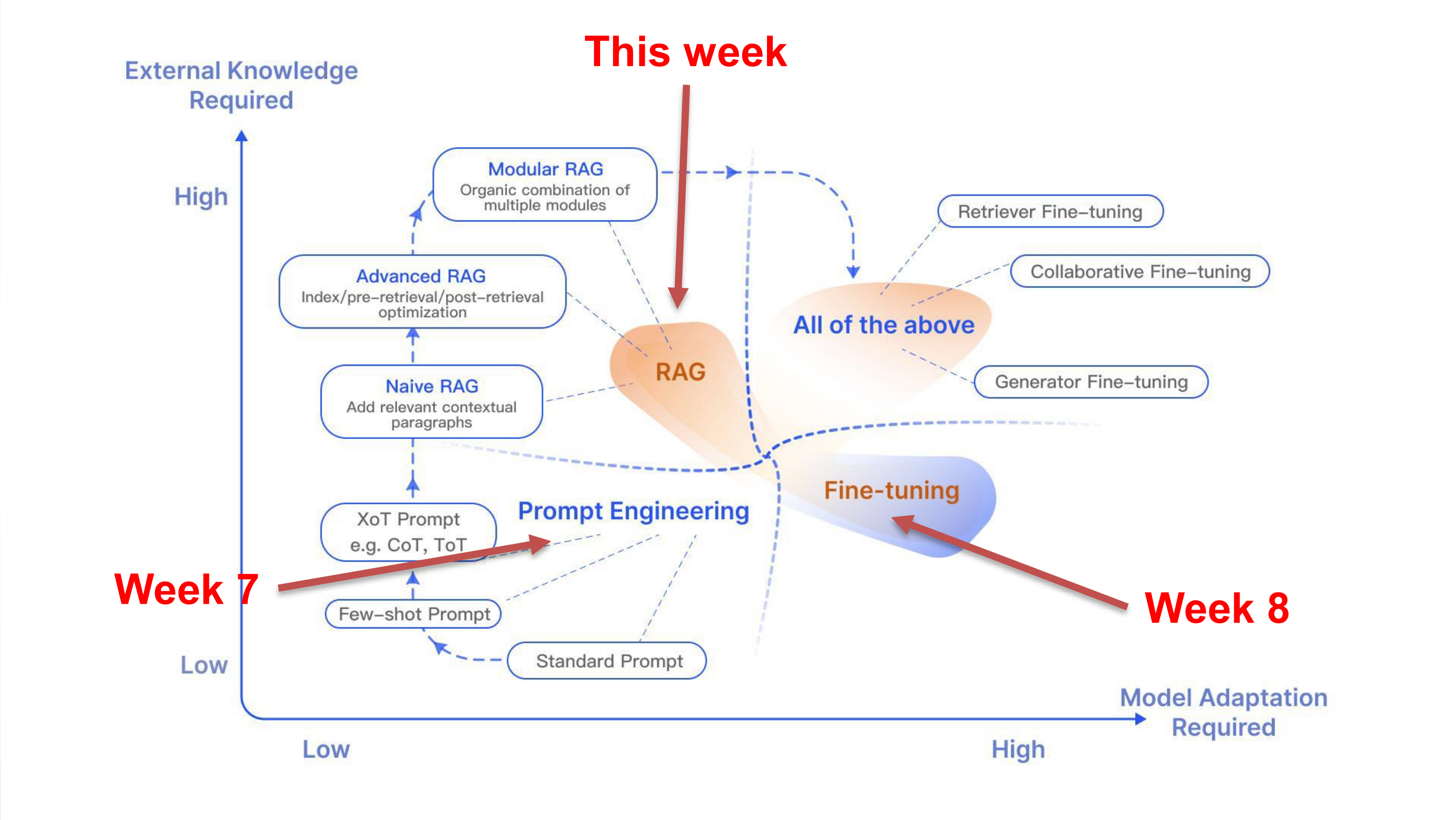
RAG overview

- When answering questions or generating text, it first retrieves relevant information from a large number of documents, and then LLMs generates answers based on this information.
- By attaching an external knowledge base, there is no need to retrain the entire large model for each specific task.
- The RAG model is especially suitable for knowledge-intensive tasks.



Symbolic Knowledge or Parametric Knowledge

- Ways to optimize LLMs.
  - Prompt Engineering
  - Instruct / Fine-tuning
  - Retrieval-Augmented Generation



RAG vs Fine-tuning

Feature Comparison	RAG	Fine-Tuning
Knowledge Updates	Directly updating the retrieval knowledge base ensures that the information remains current without the need for frequent retraining, making it well-suited for dynamic data environments.	Stores static data, requiring retraining for knowledge and data updates.
External Knowledge	Proficient in leveraging external resources, particularly suitable for accessing documents or other structured/unstructured databases.	Can be utilized to align the externally acquired knowledge from pretraining with large language models, but may be less practical for frequently changing data sources.
Data Processing	Involves minimal data processing and handling.	Depends on the creation of high-quality datasets, and limited datasets may not result in significant performance improvements.
Model Customization	Focuses on information retrieval and integrating external knowledge but may not fully customize model behavior or writing style.	Allows adjustments of LLM behavior, writing style, or specific domain knowledge based on specific tones or terms.
Interpretability	Responses can be traced back to specific data sources, providing higher interpretability and traceability.	Similar to a black box, it is not always clear why the model reacts a certain way, resulting in relatively lower interpretability.
Computational Resources	Depends on computational resources to support retrieval strategies and technologies related to databases. Additionally, it requires the maintenance of external data source integration and updates.	The preparation and curation of high-quality training datasets, defining fine-tuning objectives, and providing corresponding computational resources are necessary.
Latency Requirements	Involves data retrieval, which may lead to higher latency.	LLM after fine-tuning can respond without retrieval, resulting in lower latency.
Reducing Hallucinations	Inherently less prone to hallucinations as each answer is grounded in retrieved evidence.	Can help reduce hallucinations by training the model based on specific domain data but may still exhibit hallucinations when faced with unfamiliar input.
Ethical and Privacy Issues	Ethical and privacy concerns arise from the storage and retrieval of text from external databases.	Ethical and privacy concerns may arise due to sensitive content in the training data.

RAG Application

- Scenarios where RAG is applicable:
  - Long-tail distribution of data
  - Frequent knowledge updates
  - Answers requiring verification and traceability
  - Specialized domain knowledge
  - Data privacy preservation

Question Answering | Fact checking | Dialog systems | Summarisation

Machine translation | Code generation | Sentiment Analysis | Commonsense reasoning

Contents

Foundation of information Retrieval

- What is information Retrieval?
  - The system searches collections for items relevant to the user's query. It then returns those items to the user, typically in list form sorted per computed relevance[#]
- Three main questions in information retrieval:
  - How to map the text into features (**Embedding method**)
  - How to measure the similarity between features (**IR Modelling**)
  - How to do it efficiently (**Indexing**)

[#] Qiaozhu Mei and Dragomir Radev, "Information Retrieval," *The Oxford Handbook of Computational Linguistics*, 2nd edition, Oxford University Press, 2016.

Embedding Method

- How to map the text into features (vectors)?
  - Discrete representation
    - Convert the input query/document into vectors based on the lexicon
  - Continuous representation
    - Using representation learning to convert the input text into vectors

Discrete representation

- In discrete representation, for both query and document, we assign each word a specific dimension. If a word appears query/document, then value of the corresponding dimension is:
  - In Binary representation: 1
  - In TF (term frequency) based representation: t (how many times this word appears within the query/documents)
  - In TF-IDF (inverse document frequency) based representation: tlog(n/x)
    - Here, t is term frequency, n is number of documents, x is the number of documents which contains this term.

Discrete representation (example)

- We have the following documents:
  - D1 = “Shipment of gold damaged in a fire”.
  - D2 = “Delivery of silver arrived in a silver truck”.
  - D3 = “Shipment of gold arrived in a truck”

- After pre-processing:
  - D1 = “shipment”, “gold”, “damage”, “fire”.
  - D2 = “delivery”, “silver”, “arrive”, “truck”.
  - D3 = “shipment”, “gold”, “arrive”, “truck”.

Discrete representation (example)

- Building vocabulary:
  - V = “shipment”, “gold”, “damage”, “fire”, “delivery”, “silver”, “arrive”, “truck”.

- Detect the feature for each document. If the feature occurs, the corresponding value is ‘1’, otherwise ‘0’ (binary feature):

	shipment	gold	damage	fire	delivery	silver	arrive	truck
D1	1	1	1	1	0	0	0	0
D2	0	0	0	0	1	1	1	1
D3	1	1	0	0	0	0	1	1

Discrete representation (example)

- **Definition – term frequency (TF):**
  - t - how many times the term appears in the document
- **Example:**
  - D1 = “Shipment of gold damaged in a fire”.
  - D2 = “Delivery of silver arrived in a silver truck”.
  - D3 = “Shipment of gold arrived in a truck”

	shipment	gold	damage	fire	delivery	silver	arrive	truck
D1	1	1	1	1	0	0	0	0
D2	0	0	0	0	1	**2**	1	1
D3	1	1	0	0	0	0	1	1

Discrete representation (example)

- **Definition – inverse document frequency (IDF):**
  - $log(n/x)$ – n is number of documents, x is the number of documents which contains this term
- **Example:**
  - D1 = “Shipment of gold damaged in a fire”.
  - D2 = “Delivery of silver arrived in a silver truck”.
  - D3 = “Shipment of gold arrived in a truck”

shipment	gold	damage	fire	delivery	silver	arrive	truck
0.176	0.176	0.477	0.477	0.477	0.477	0.176	0.176

Inverse document frequency vector

Discrete representation (example)

	shipment	gold	damage	fire	delivery	silver	arrive	truck
D1	1	1	1	1	0	0	0	0
D2	0	0	0	0	1	2	1	1
D3	1	1	0	0	0	0	1	1

Term frequency matrix

shipment	gold	damage	fire	delivery	silver	arrive	truck
0.176	0.176	0.477	0.477	0.477	0.477	0.176	0.176

Inverse document frequency vector

	shipment	gold	damage	fire	delivery	silver	arrive	truck
D1	0.176	0.176	0.477	0.477	0	0	0	0
D2	0	0	0	0	0.477	0.954	0.176	0.176
D3	0.176	0.176	0	0	0	0	0.176	0.176

TF-IDF Matrix

Embedding Method

- How to map the text into features (vectors)?
  - Discrete representation
    - Convert the input query/document into vectors based on the lexicon
  - **Continuous representation**
    - **Using representation learning to convert the input text into vectors**

Continuous representation

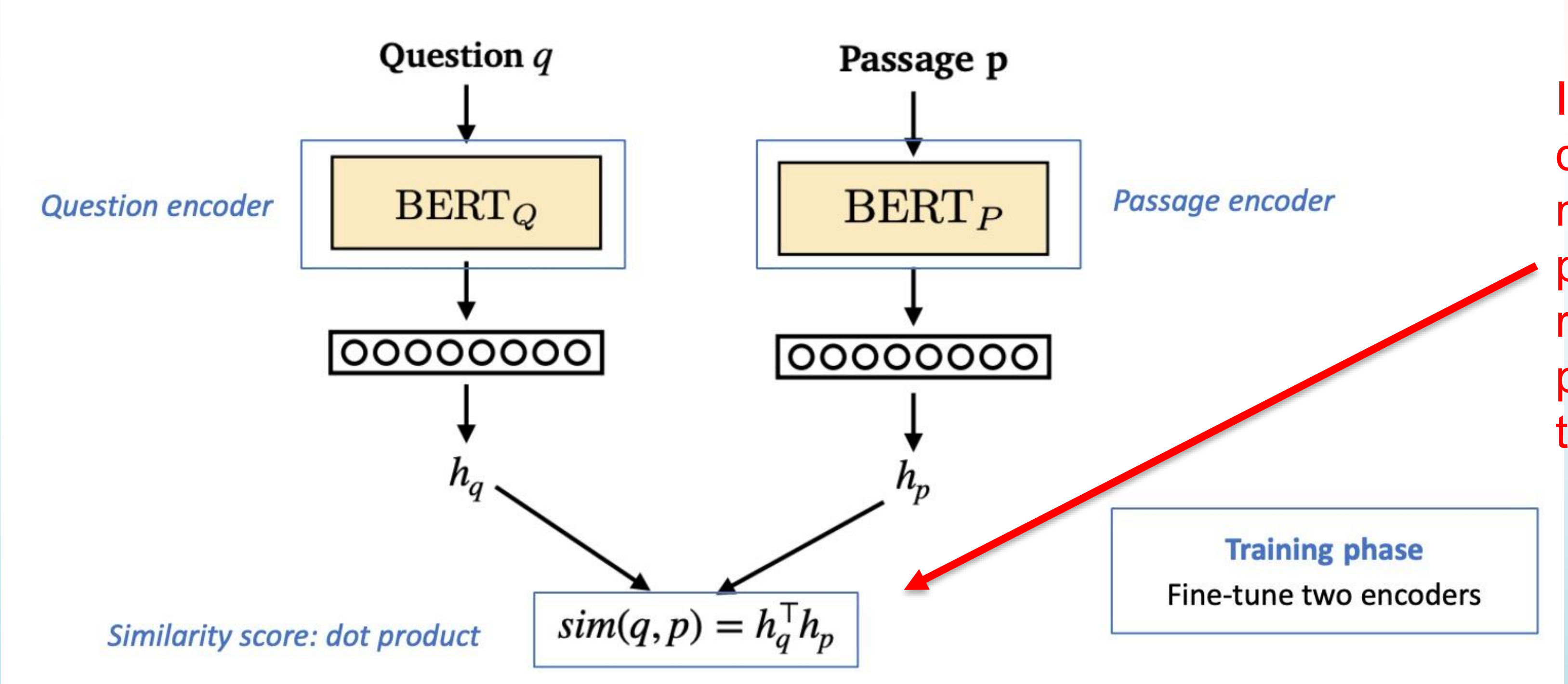
- Continuous representation - using representation learning to convert the input text into vectors
    - Define the relation first, then using optimiser to update the embedding to approximate the relation.

Continuous representation

- Continuous representation - using representation learning to convert the input text into vectors
  - Define the relation first, then using optimiser to update the embedding to approximate the relation.

Dense Passage Retrieval

- Encode questions and text passages into continuous vectors (embeddings) and retrieve passages using vector similarity instead of keyword overlapping.
- Train directly on question–passage pairs, using in-batch negatives to improve efficiency.



In each batch, there are multiple question–answer pairs, both matched and unmatched. Matched pairs should have similar representations, while unmatched pairs should have representations that are far apart.

https://aclanthology.org/2020.emnlp-main.550.pdf

ReContriever

- What if we don't have annotated data (Matched and unmatched QA-pair).
- Using pseudo-examples: For each passage/document p, create an augmented version p′. Then treat (p, p′) as a positive pair:
  - Masking words (random word masking)
  - Span deletion
  - Back-translation Sentence
  - Reordering Adding noise
  - Perturbations Cropping (taking a subset of sentences)

https://aclanthology.org/2023.findings-acl.695.pdf

Using API

- There are many APIs could do this job, for example, Mistral AI:
- Example: link
- Some other options:
  - Sentence Bert
  - SimCSE
  - ……

MISTRAL EMBED API

OPEN IN COLAB

How to Generate Embeddings

To generate text embeddings using Mistral AI's embeddings API, we can make a request to the API endpoint and specify the embedding model `mistral-embed`, along with providing a list of input texts. The API will then return the corresponding embeddings as numerical vectors, which can be used for further analysis or processing in NLP applications.

PYTHON TYPESCRIPT CURL OUTPUT

```python
import os
from mistralai import Mistral

api_key = os.environ["MISTRAL_API_KEY"]
model = "mistral-embed"

client = Mistral(api_key=api_key)

embeddings_batch_response = client.embeddings.create(
    model=model,
    inputs=["Embed this sentence.", "As well as this one."],
)
```

The output is an embedding object with the embeddings and the token usage information.

Let's take a look at the length of the first embedding:

PYTHON TYPESCRIPT CURL

```python
len(embeddings_batch_response.data[0].embedding)
```


https://docs.mistral.ai/capabilities/embeddings

IR Modelling

- What is information Retrieval?
  - The system searches collections for items relevant to the user's query. It then returns those items to the user, typically in list form sorted per computed relevance
- Three main questions in information retrieval:
  - How to map the text into features (**Embedding method**)
  - **How to measure the similarity between features (IR Modelling)**
  - How to do it efficiently (**Indexing**)

IR Modelling

- In IR modelling, we use different metric to measure the similarity/distance between the query and given documents. The target is to find the top-k relevant documents based on the given query.
  - Cosine similarity (for both Discrete & Continuous representation)
  - Jaccard distance (for Discrete representation only)
  - BM25 (for Discrete representation only)

Cosine similarity

- Cosine similarity

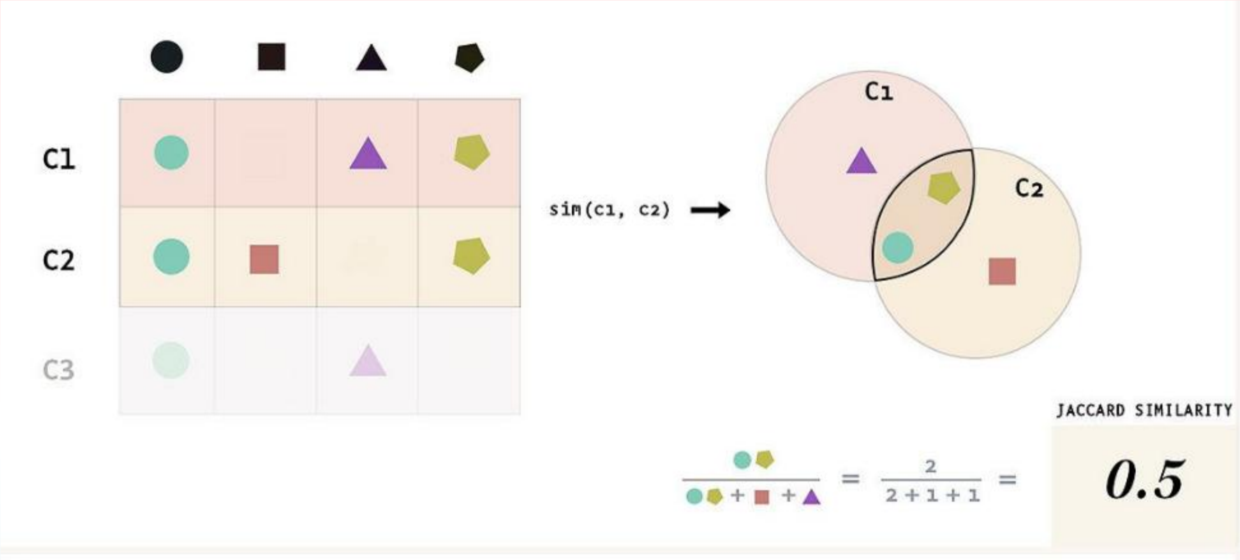
$$Cos(x,y) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n}(x_i)^2}\sqrt{\sum_{i=1}^{n}(y_i)^2}}$$

- Considering
  - D1 = [1,1,1,1,0,0,0,0]
  - D3 = [1,1,0,0,0,0,1,1]

$$Cos(D1,D3)=1/2$$

Jaccard similarity

- Only considering if there is over lapping or not. We don't care about the value.



- For example:
  - $R_x = [2,0,3,3]$
  - $R_y = [1,1,0,5]$

- Jaccard similarity: $sim(x, y) = \frac{|\boldsymbol{R_x} \cap \boldsymbol{R_y}|}{|\boldsymbol{R_x} \cup \boldsymbol{R_y}|}$

BM25

- BM25 is a lexicon based retrieval method that ranks a set of documents based on the query terms appearing in each document, regardless of their proximity within the document.

- Given a query Q, containing keywords $q_1, q_2, \dots, q_n$, the BM25 score of a document D is:

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \frac{f(q_i,D)(k_1+1)}{f(q_i,D) + k_1\left(1 - b + b\frac{|D|}{avgdl}\right)}$$

Length of D

hyperparameters

Average length of all docs

BM25

- BM25 is a lexicon based retrieval method that ranks a set of documents based on the query terms appearing in each document, regardless of their proximity within the document.

- Given a query Q, containing keywords $q_1, q_2, \ldots, q_n$, the BM25 score of a document D is:

$$score(D, Q) = \sum_{i=1}^{n} IDF(q_i) \frac{f(q_i, D)(k_1 + 1)}{f(q_i, D) + k_1(1 - b + b \frac{|D|}{avgdl})}$$

Length of D

hyperparameters

Average length of all docs

Maybe…a bit confusing
Can you speak in English?

BM25

- BM25 is a lexicon based retrieval method that ranks a set of documents based on the query terms appearing in each document, regardless of their proximity within the document.

- Given a query Q, containing keywords $q_1, q_2, \ldots, q_n$, the BM25 score of a document D is:

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \frac{f(q_i,D)(k_1+1)}{f(q_i,D) + k_1\left(1 - b + b\frac{|D|}{avgdl}\right)}$$

Length of D

hyperparameters

Average length of all docs

Relax…it is pretty simple actually

BM25

- BM25 is a lexicon based retrieval method that ranks a set of documents based on the query terms appearing in each document, regardless of their proximity within the document.

- Given a query Q, containing keywords $q_1, q_2, \ldots, q_n$, the BM25 score of a document D is:

IDF term in Q (is it an important word?)

$$score(D, Q) = \sum_{i=1}^{n} IDF(q_i) \frac{f(q_i, D)(k_1 + 1)}{f(q_i, D) + k_1(1 - b + b\frac{|D|}{avgdl})}$$

The 'percentage' of querying words in D

Indexing

- Next question, how to do it efficiently (**Indexing**)

- Suppose we have 1k queries, and there are 1 billion documents in knowledge based, how many times of comparison we need?
- 1k x 1b

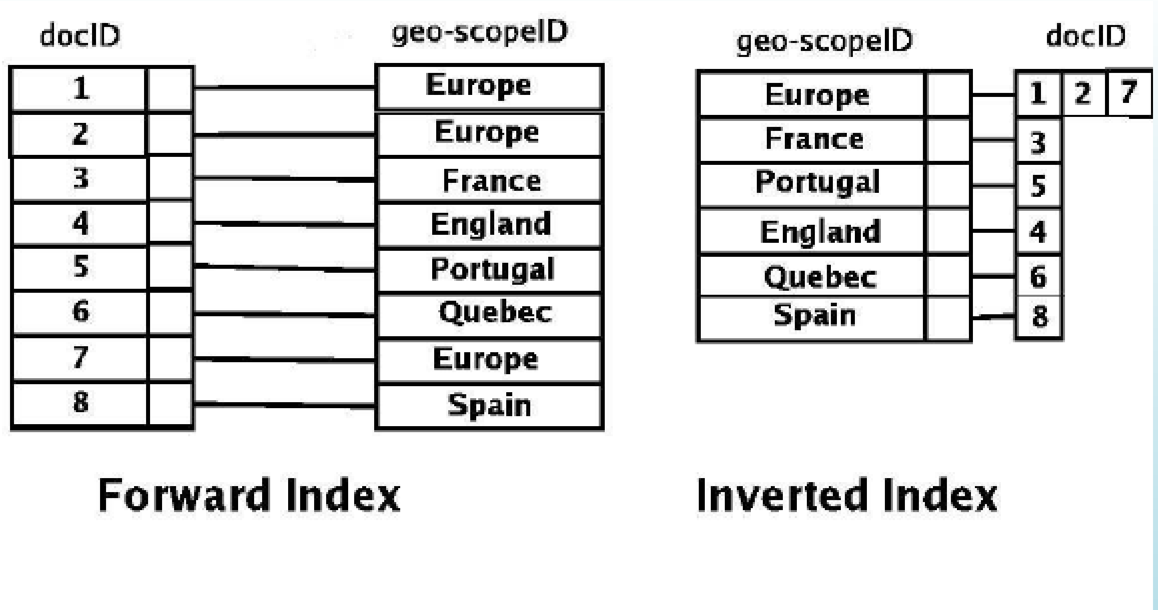
It is a really huge number.

In real world scenario, it could be even larger

If there is only one important task in information retrieval, it must be “indexing”

Indexing - Discrete representation

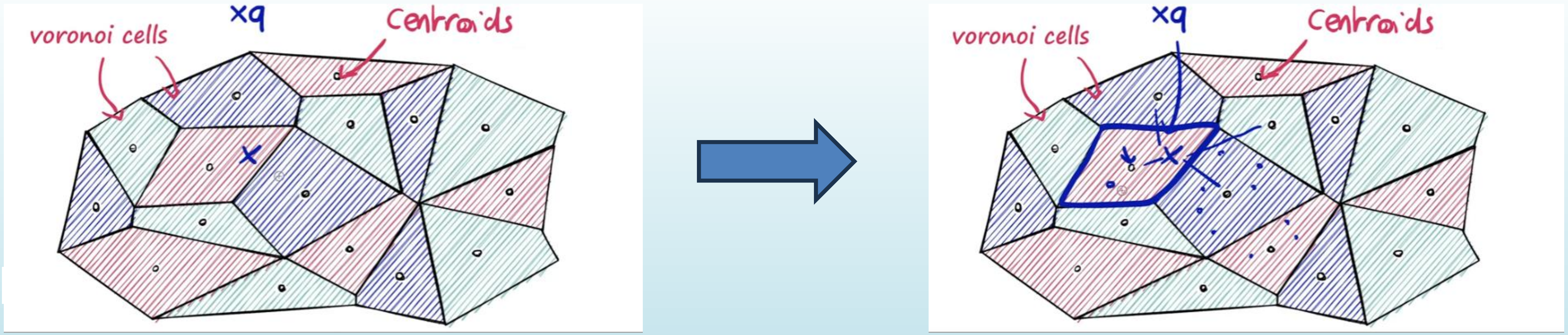
- Inverted index
- Since the discrete representation is sparse (most dims are zero), we can build inverted index. For each word, we build a link list to store all the documents contain this word.
- For the given query, the complexity is now only related to the #unique words in the query. (**In most queries, the size is just few words**)



Indexing - Continuous representation

- In continuous representation, it might be a bit complex. There is no sparse representation anymore.
- We can use the following method to speed up the searching.
  - Vector compression – reduce the size of vectors
  - Hierarchical clustering – in each layer only search the nearest cluster

Clustering the documents first, and then,
Only consider the nearest centroid during the searching



https://www.pinecone.io/learn/series/faiss/faiss-tutorial/

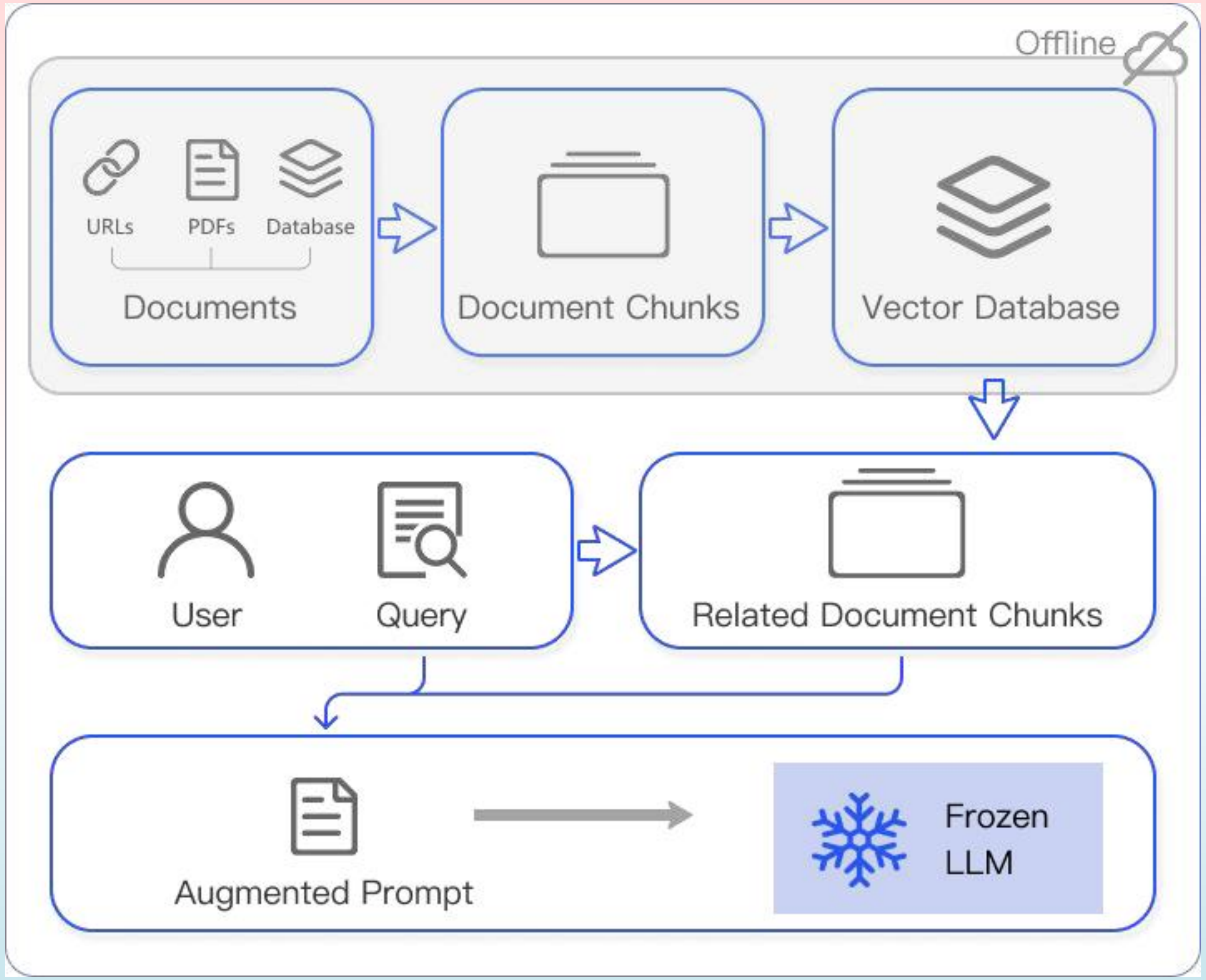
Contents

Naive RAG

- **Step 1 – indexing**
  - Divide the document into even chunks, each chunk being a piece of the original text.
  - Using the encoding model to generate an embedding for each chunk.
  - Store the Embedding of each block in the vector database.
- **Step 2 – Retrieval**
  - Retrieve the k most relevant documents using vector similarity search.
- **Step 3 – Generation**
  - The original query and the retrieved text are combined and input into a LLM to get the final answer

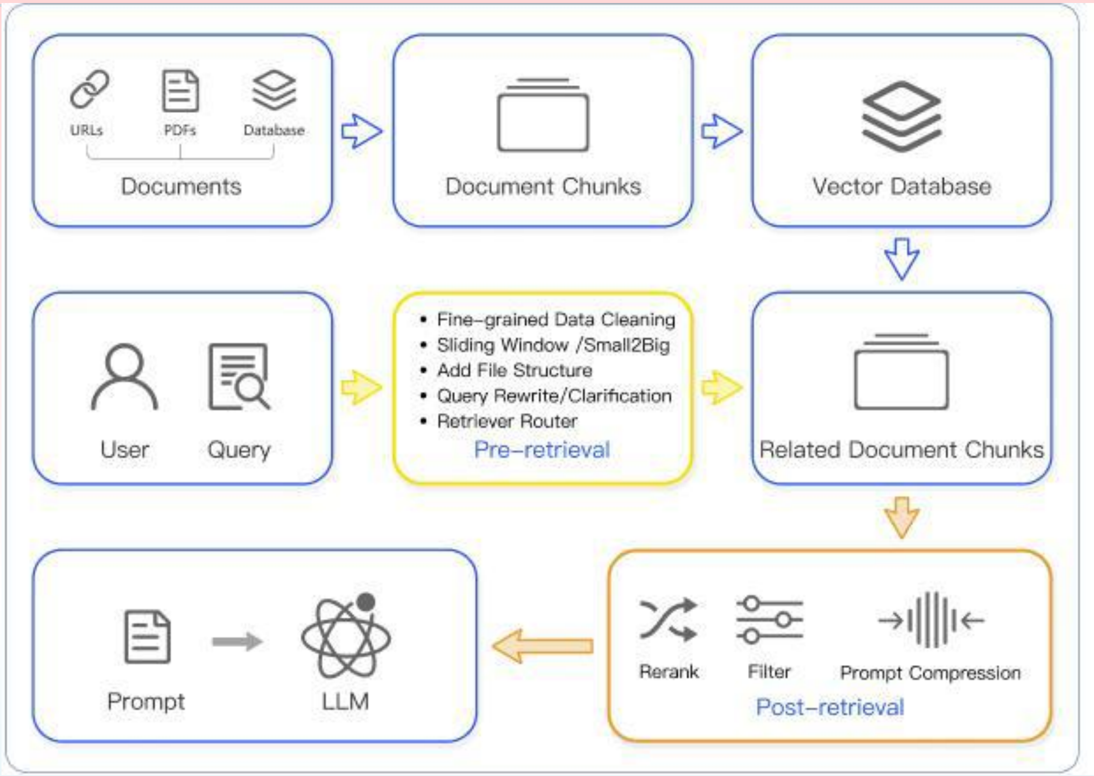
Naive RAG

- **Step 1 – indexing**
- **Step 2 – Retrieval**
- **Step 3 – Generation**



Advanced RAG

- Step 1 – indexing
  - + index optimization
  - + pre-retrieval process
- Step 2 – Retrieval
  - +post-retrieval process
- Step 3 – Generation



Advanced RAG

- **Step 1 – indexing**
  - **+ index optimization**
  - **+ pre-retrieval process**
- **Step 2 – Retrieval**
  - **+post-retrieval process**
- **Step 3 – Generation**

Sliding windows

Fine-grained segmentation

Adding metadata

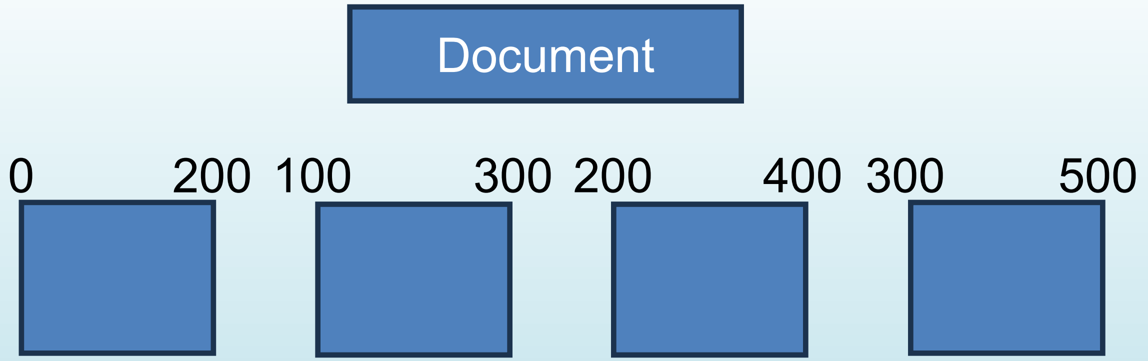
Advanced RAG

- **Step 1 – indexing**
  - **+ index optimization**
  - **+ pre-retrieval process**
- **Step 2 – Retrieval**
  - **+post-retrieval process**
- **Step 3 – Generation**

Sliding windows

Fine-grained segmentation

Adding metadata



Split the doc into chunks, and ensure there is over lapping between chunks (WHY?)

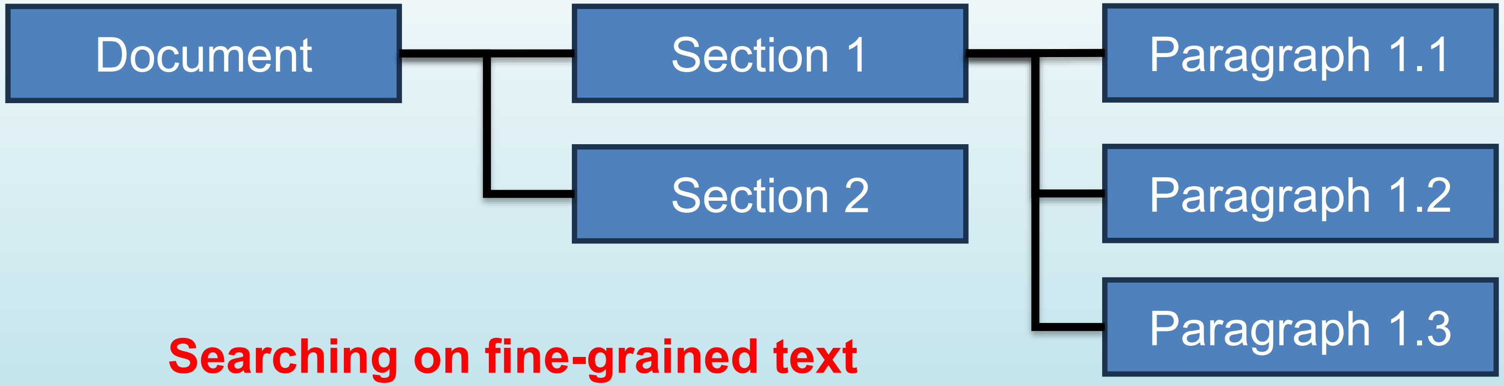
Advanced RAG

- **Step 1 – indexing**
  - **+ index optimization**
  - **+ pre-retrieval process**
- **Step 2 – Retrieval**
  - **+post-retrieval process**
- **Step 3 – Generation**

Sliding windows

Fine-grained segmentation

Adding metadata



Searching on fine-grained text

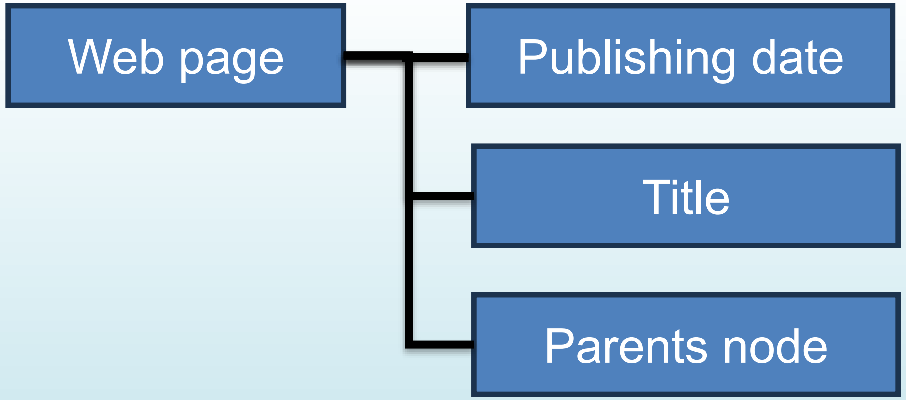
Advanced RAG

- **Step 1 – indexing**
  - **+ index optimization**
  - **+ pre-retrieval process**
- **Step 2 – Retrieval**
  - **+post-retrieval process**
- **Step 3 – Generation**

Sliding windows

Fine-grained segmentation

Adding metadata



The metadata is the aspects of each chunk. It will help both retriever and generator to improve the performance.

Advanced RAG

- Step 1 – indexing
  - + index optimization
  - + pre-retrieval process
- Step 2 – Retrieval
  - +post-retrieval process
- Step 3 – Generation

Retrieve routes

Summarization

Rewriting

Confidence judgment

Advanced RAG

- **Step 1 – indexing**
  - **+ index optimization**
  - **+ pre-retrieval process**
- **Step 2 – Retrieval**
  - **+post-retrieval process**
- **Step 3 – Generation**

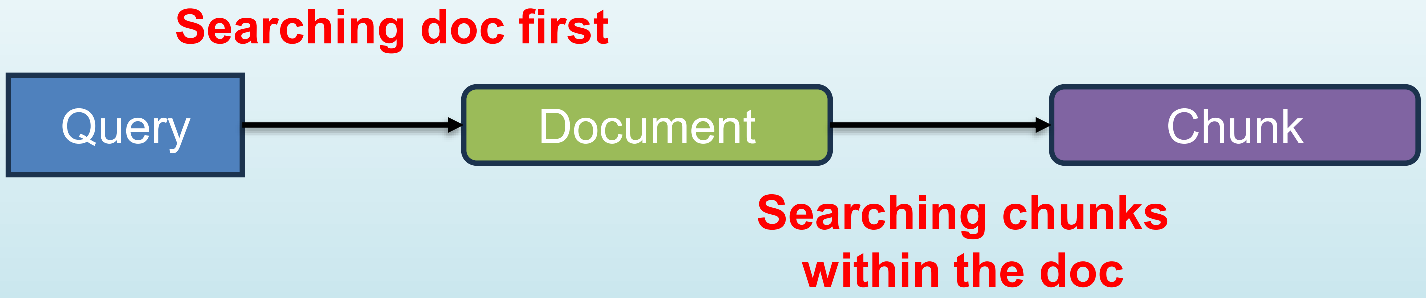
Retrieve routes

Summarization

Rewriting

Confidence judgment

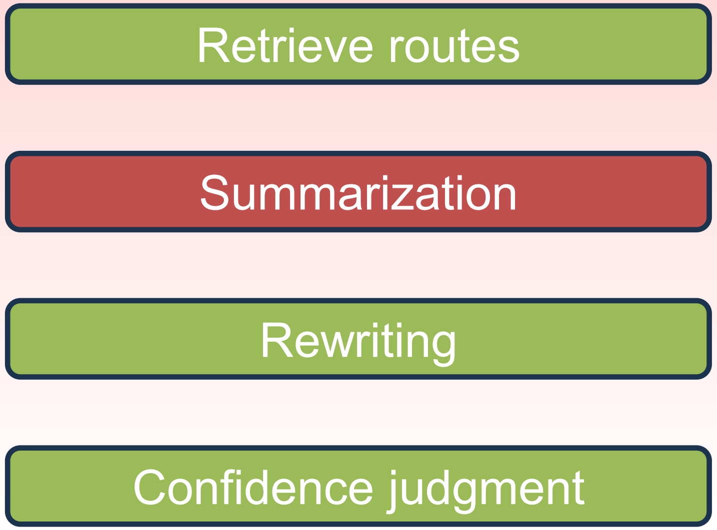
Instead of one flat "retrieve chunks by embeddings" step, you can:

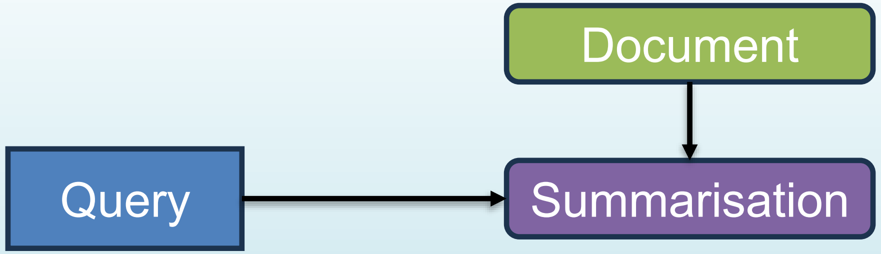


Retrieve routes = multiple retrieval paths that a RAG system can choose from, depending on query intent, data type, or document structure.

Advanced RAG

- **Step 1 – indexing**
  - **+ index optimization**
  - **+ pre-retrieval process**
- **Step 2 – Retrieval**
  - **+post-retrieval process**
- **Step 3 – Generation**





Summarise first

Searching in smmarisation instead of full documents

Advanced RAG

- **Step 1 – indexing**
  - **+ index optimization**
  - **+ pre-retrieval process**
- **Step 2 – Retrieval**
  - **+post-retrieval process**
- **Step 3 – Generation**

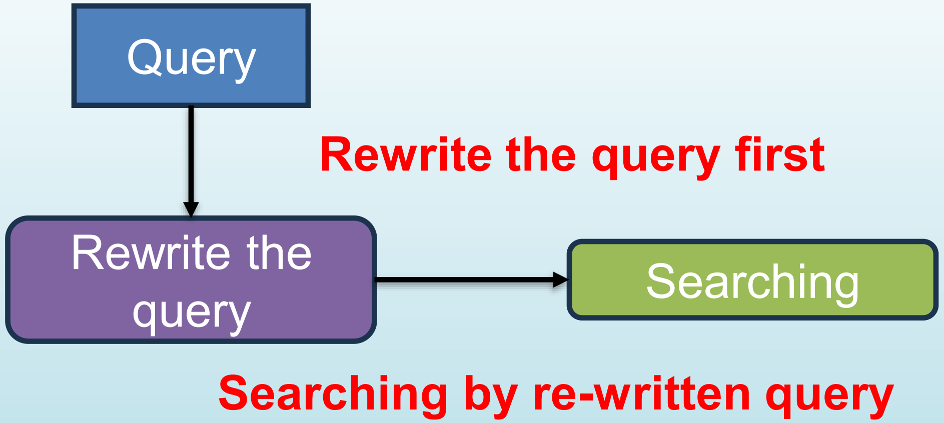
Retrieve routes

Summarization

Rewriting

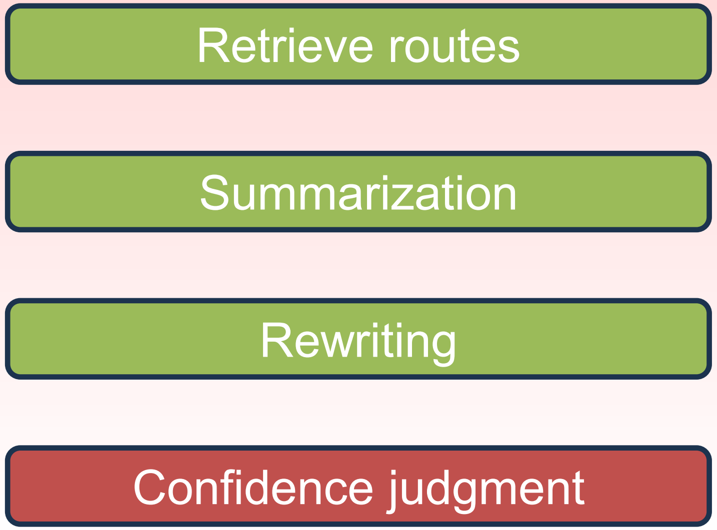
Confidence judgment

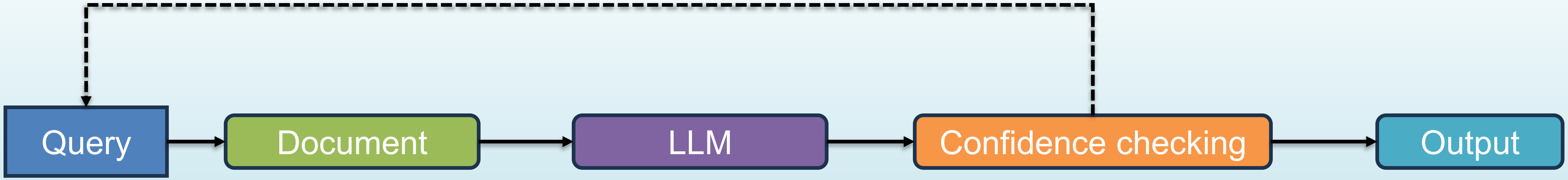
Benefits:
a more explicit query
a more keyword-rich query
a more structured query
multiple diverse sub-queries



Advanced RAG

- **Step 1 – indexing**
  - **+ index optimization**
  - **+ pre-retrieval process**
- **Step 2 – Retrieval**
  - **+post-retrieval process**
- **Step 3 – Generation**





Confirm the Confidence before output
By similarity scores
By LLM Confidence scores

Advanced RAG

- **Step 1 – indexing**
  - **+ index optimization**
  - **+ pre-retrieval process**
- **Step 2 – Retrieval**
  - **+post-retrieval process**
- **Step 3 – Generation**

Re-order

Filter content retrieval

Advanced RAG

- **Step 1 – indexing**
  - **+ index optimization**
  - **+ pre-retrieval process**
- **Step 2 – Retrieval**
  - **+post-retrieval process**
- **Step 3 – Generation**

Re-order

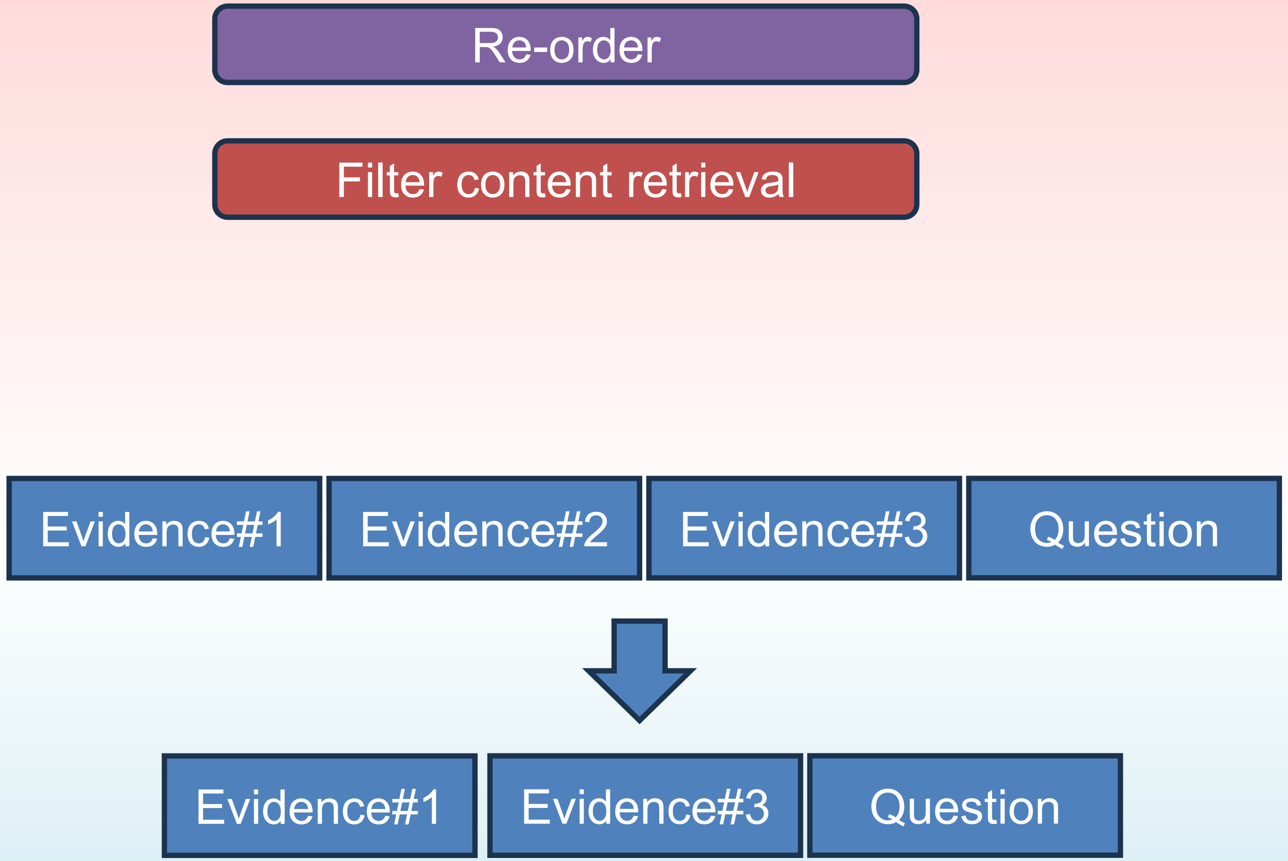
Filter content retrieval

Evidence#1	Evidence#2	Evidence#3	Question

LLMs is sensitive with the input order
The early input chunks has higher weights
How to organize the searched evidence for final output is important

Advanced RAG

- **Step 1 – indexing**
  - **+ index optimization**
  - **+ pre-retrieval process**
- **Step 2 – Retrieval**
  - **+post-retrieval process**
- **Step 3 – Generation**



To avoid possible hallucination, filtering the irrelevant evidences.

Modular RAG

- **Naïve RAG**

Read → Retrieve → Generate

- **DSP**

Demonstrate → Search → Predict → Generate

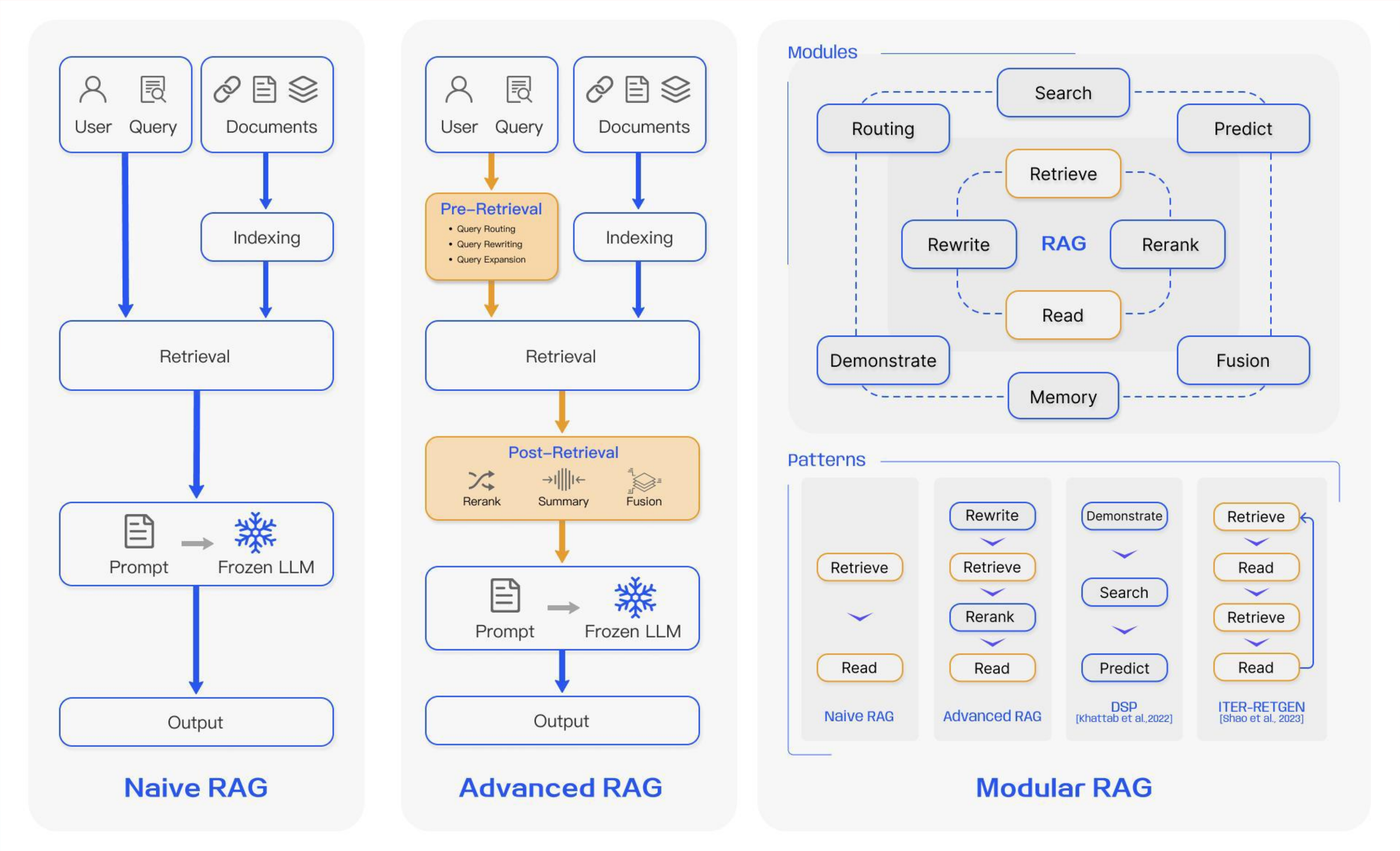
- **Rewrite-Retrieve-Read**

Rewrite → Retrieve → Read

- **Retrieve-then-read**

Retrieve → Read → Generate

Different RAG Paradigms



Key problems in RAG

- **How to retrieve**
- **When to retrieve**
- **How to use the retrieved information**

How to retrieve

- **By using the information on different structuration levels**

- **Token level**
- **Phrase level**
- **Chunk level**
- **Entity level**
- **Knowledge level**

It excels in handling long-tail and cross-domain issues with high computational efficiency, but it requires significant storage.

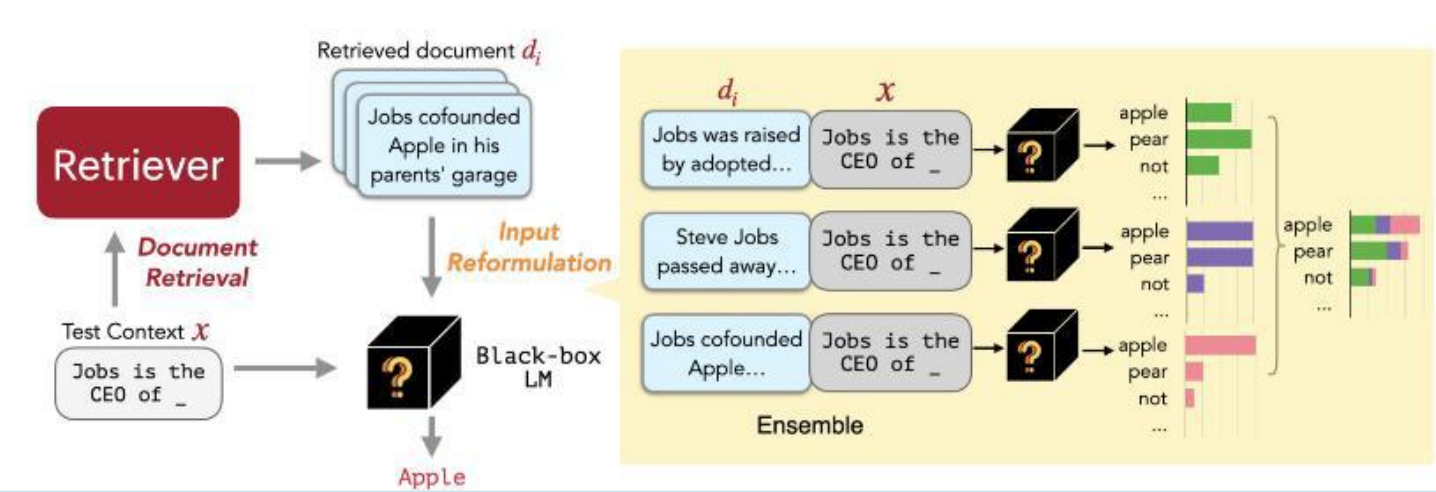
The search is broad, recalling a large amount of information, but with low accuracy, high coverage but includes much redundant information.

Richer semantic and structured information, but the retrieval efficiency is lower and is limited by the quality of KG.

When to retrieve

- **Two questions:**
- **When we need to retrieve information to support the QA**
- **How many times we need to retrieve the information**

- **Solution#1: Conducting once search during the reasoning process.**

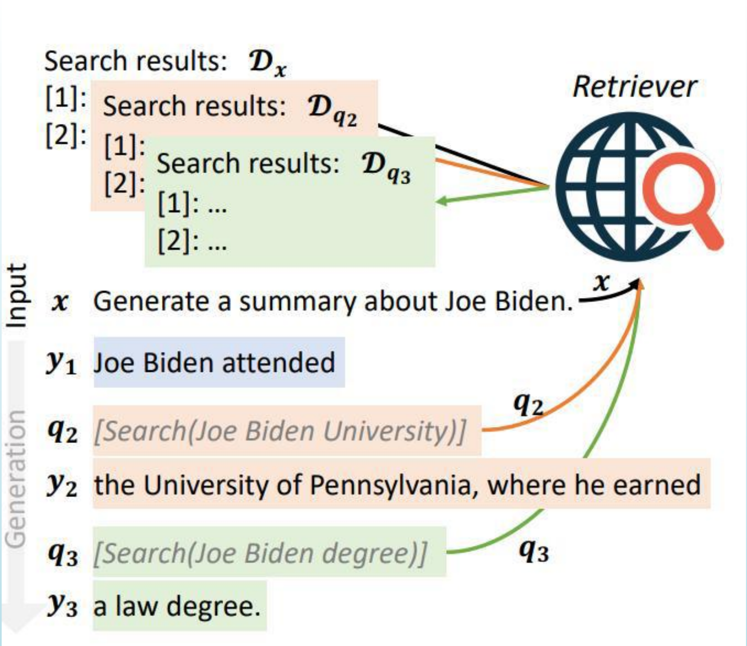


High efficiency, but low relevance of the retrieved documents

When to retrieve

- **Two questions:**
- **When we need to retrieve information to support the QA**
- **How many times we need to retrieve the information**

- **Solution#2: Adaptively conduct the search.**

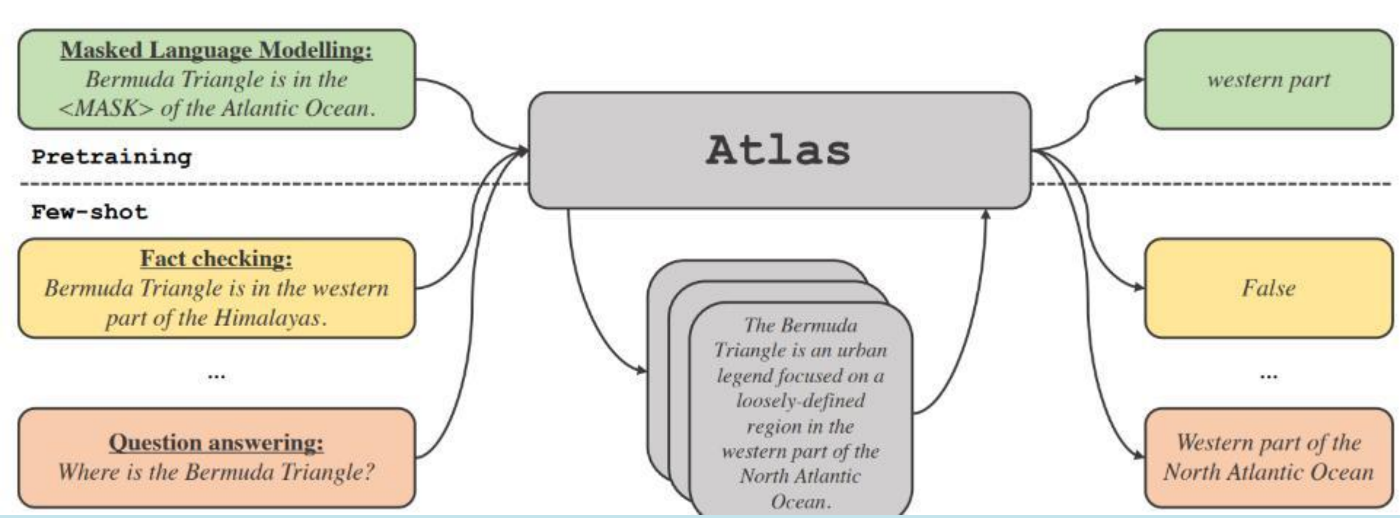


Balancing efficiency and information might not yield the optimal solution

When to retrieve

- **Two questions:**
- **When we need to retrieve information to support the QA**
- **How many times we need to retrieve the information**

- **Solution#3: Retrieve once for every N tokens generated.**



A large amount of information with low efficiency and redundant information.

Contents

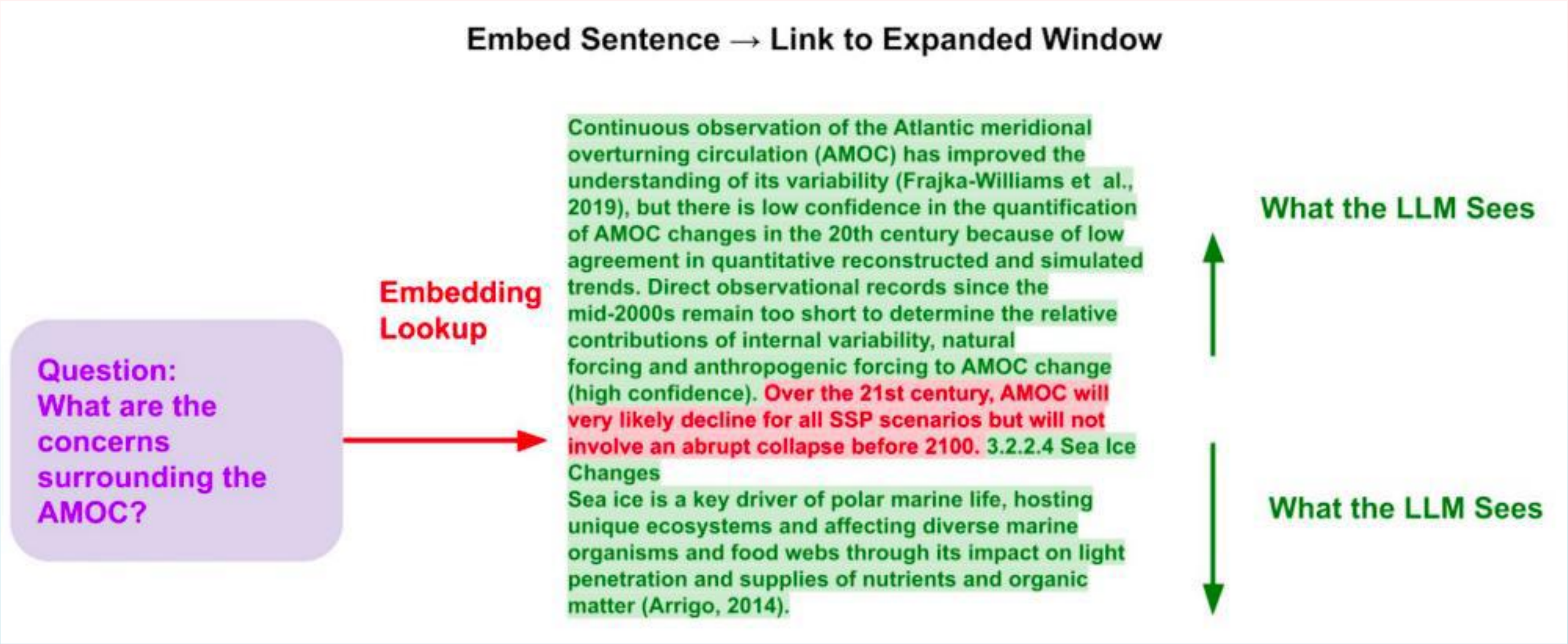
- RAG overview
- RAG Paradigms Shifting
- **Key Technologies and Evaluation**
- Applications

Key Technologies

- Data indexing optimization
- Structured Corpus
- Retrieval Source Optimization
- KG as a Retrieval Data Source
- Query Optimization
- Embedding Optimization
- Fine-tuning on RAG

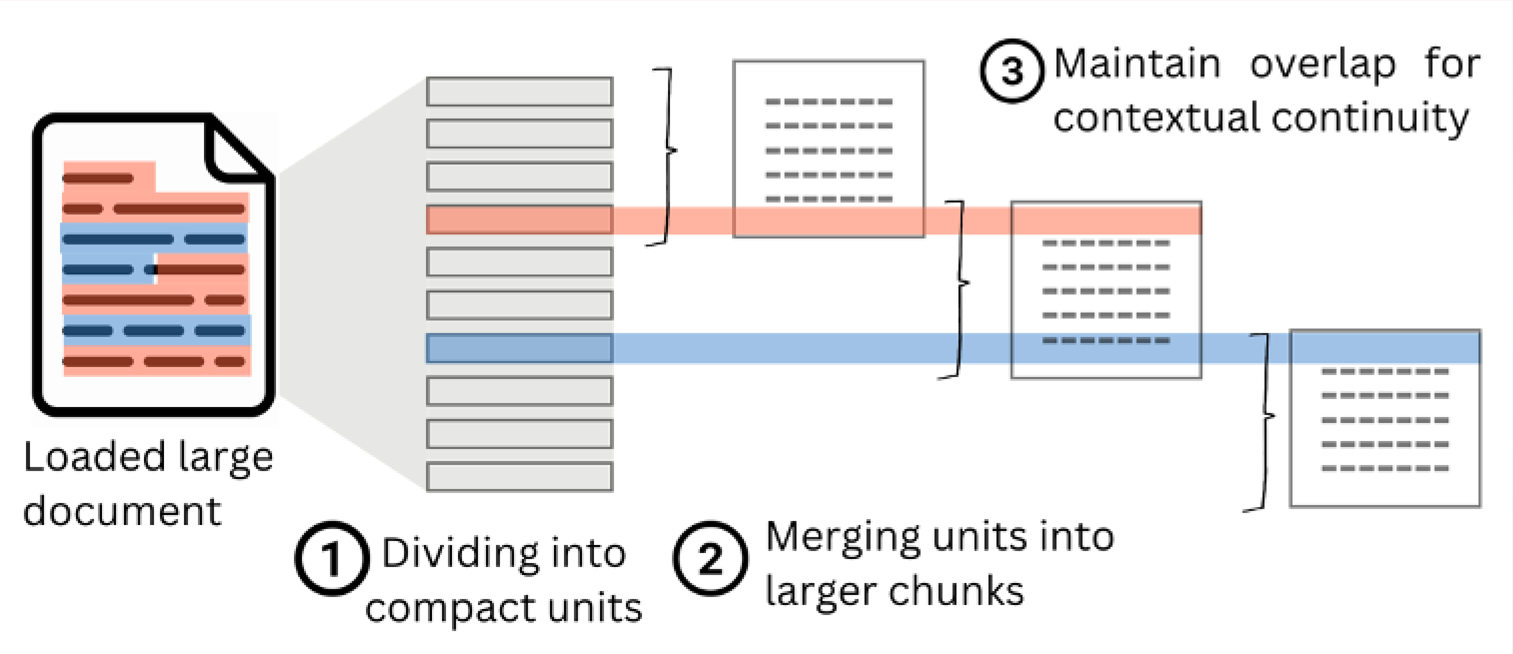
Data indexing optimization

- Chunk optimization
- Small-2-big: Embedding at sentence level expand the window during generation process.



Data indexing optimization

- Chunk optimization
- Sliding window: sliding chunk covers the entire text, avoiding semantic ambiguity.

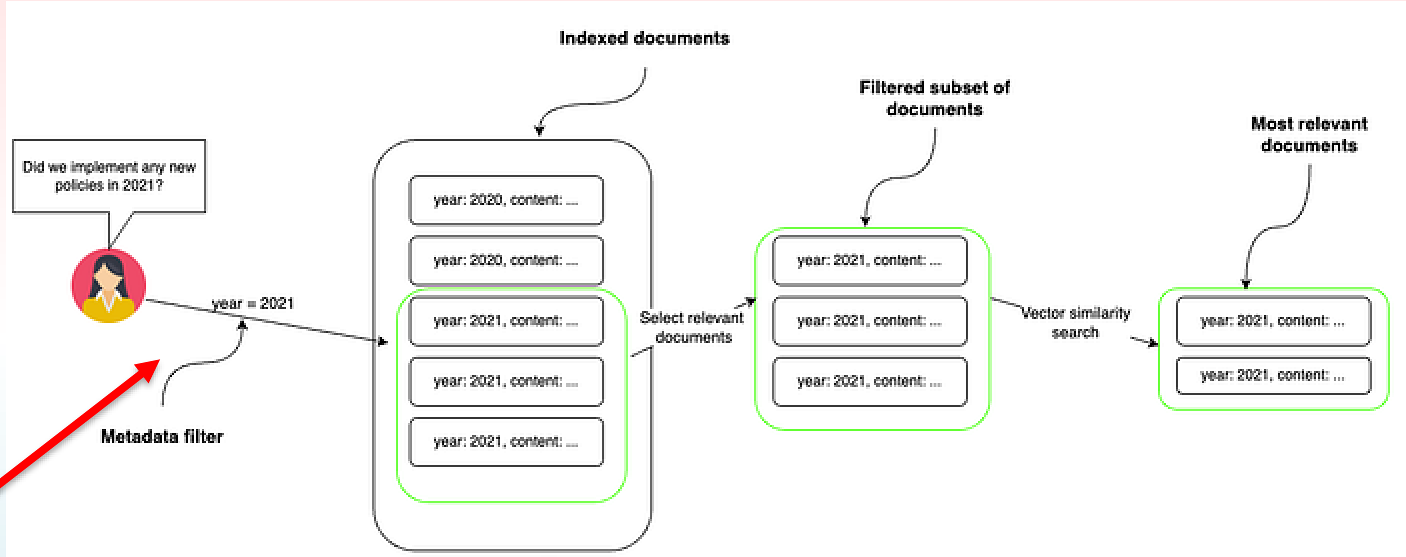


Data indexing optimization

- Chunk optimization
- Two-stage method: Retrieve documents through summaries, then retrieve text blocks from the documents.
  - Step 1 — Search Summaries (Coarse Retrieval) You maintain a summary index, where each summary represents a larger document, chapter, or cluster.
  - Step 2 — Search Related Chunks (Fine Retrieval) Once you find the top summaries, you only search inside their associated chunks.

Structured Corpus

- Adding meta-data: adding meta-data in the query searching to improve retrieval accuracy, provide context during chunking, and enables filtering



Filter the irrelevant docs

Ensure each chunk contains the metadata

Retrieval Source Optimization

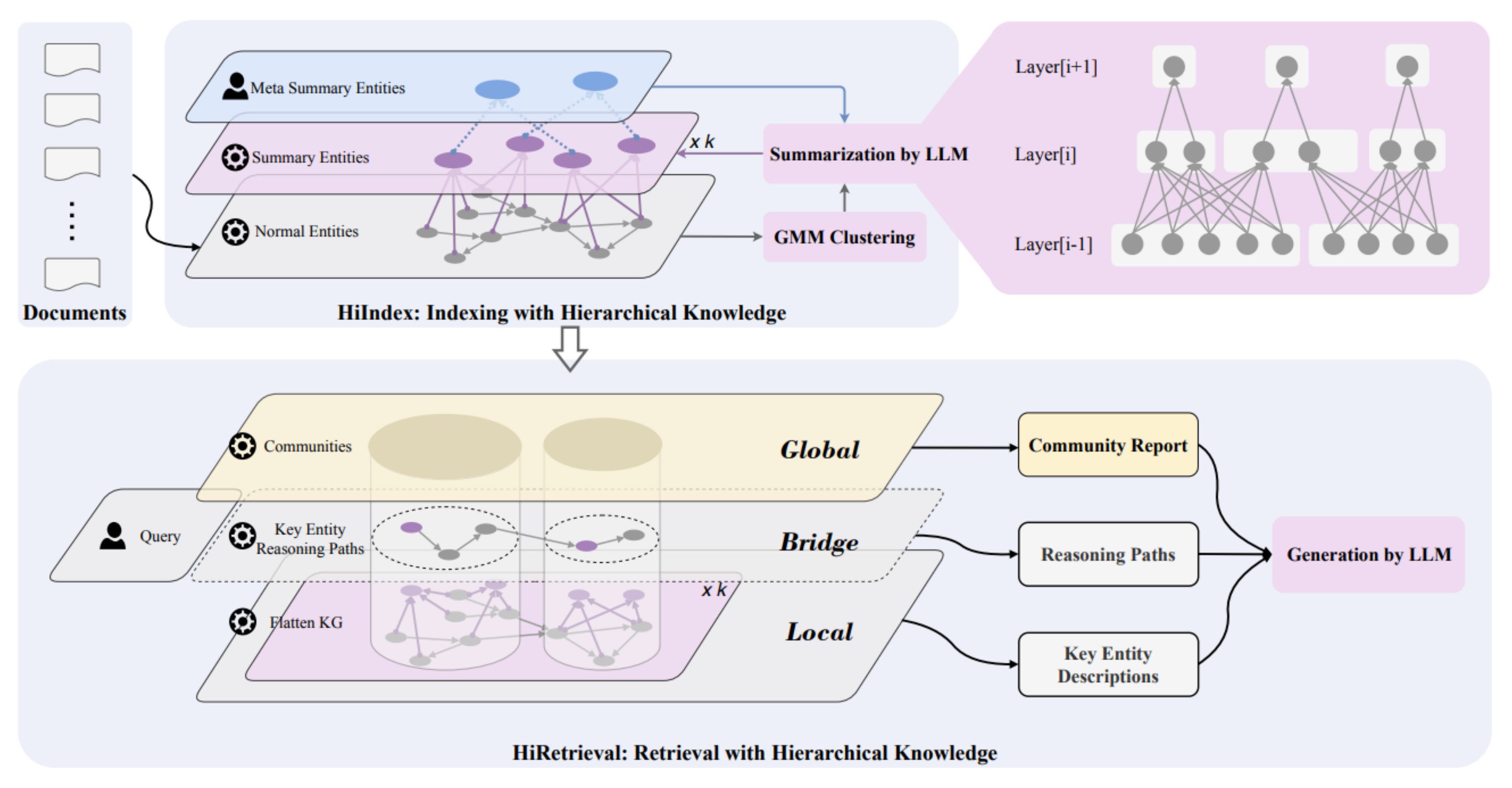
- Adding meta-data
- Two-stage method: Retrieve documents through summaries, then retrieve text blocks from the documents.
  - Step 1 — Search Summaries (Coarse Retrieval) You maintain a summary index, where each summary represents a larger document, chapter, or cluster.
  - Step 2 — Search Related Chunks (Fine Retrieval) Once you find the top summaries, you only search inside their associated chunks.

KG as a Retrieval Data Source

- Extract entities from the user's input query, then construct a subgraph to form context, and finally feed it into the large model for generation.
  - Use LLM (or other models) to extract key entities from the question.
  - Retrieve subgraphs based on entities, delving to a certain depth, such as 2 hops or even more.
  - Utilize the obtained context to generate answers through LLM.Two-stage method: Retrieve documents through summaries, then retrieve text blocks from the documents.

KG as a Retrieval Data Source

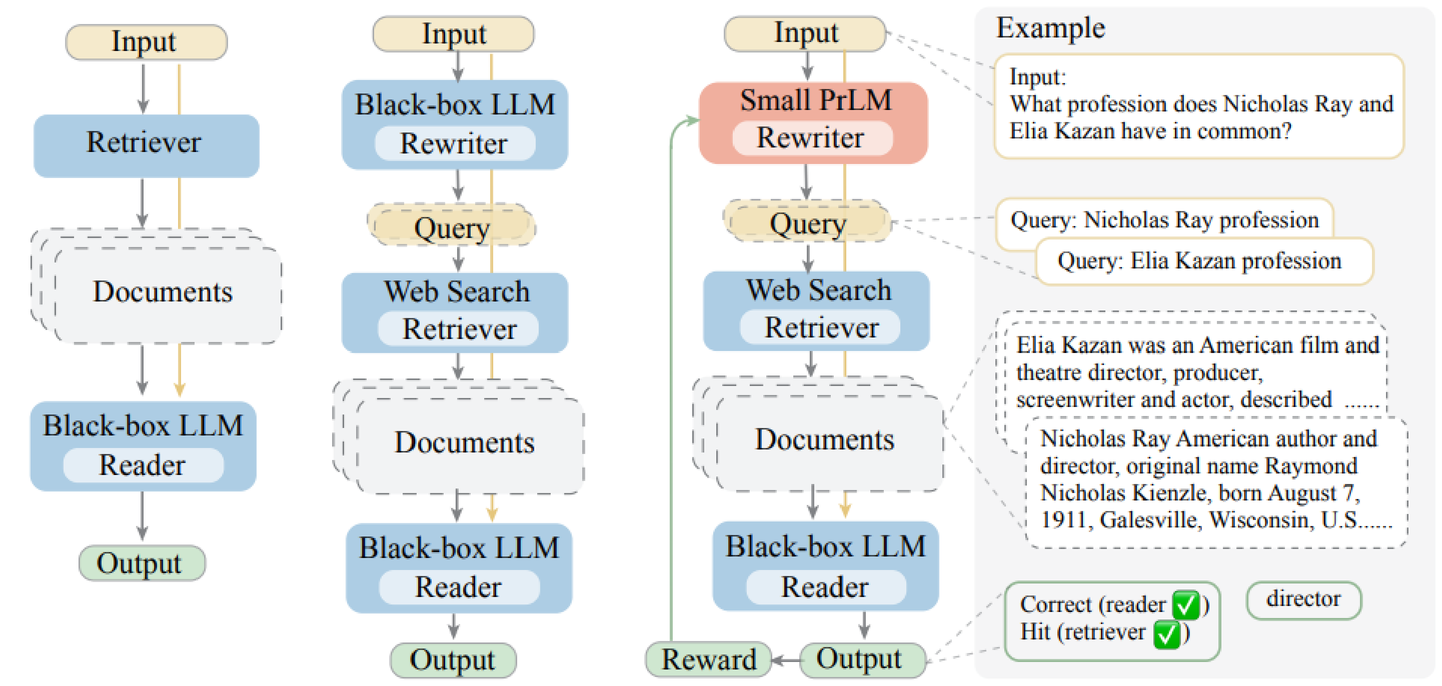
- Extract entities from the user's input query, then construct a subgraph to form context, and finally feed it into the large model for generation.



https://arxiv.org/pdf/2503.10150

Query Optimization

- Questions and answers do not always possess high semantic similarity; adjusting the Query can yield better retrieval results.
- Rewrite query:

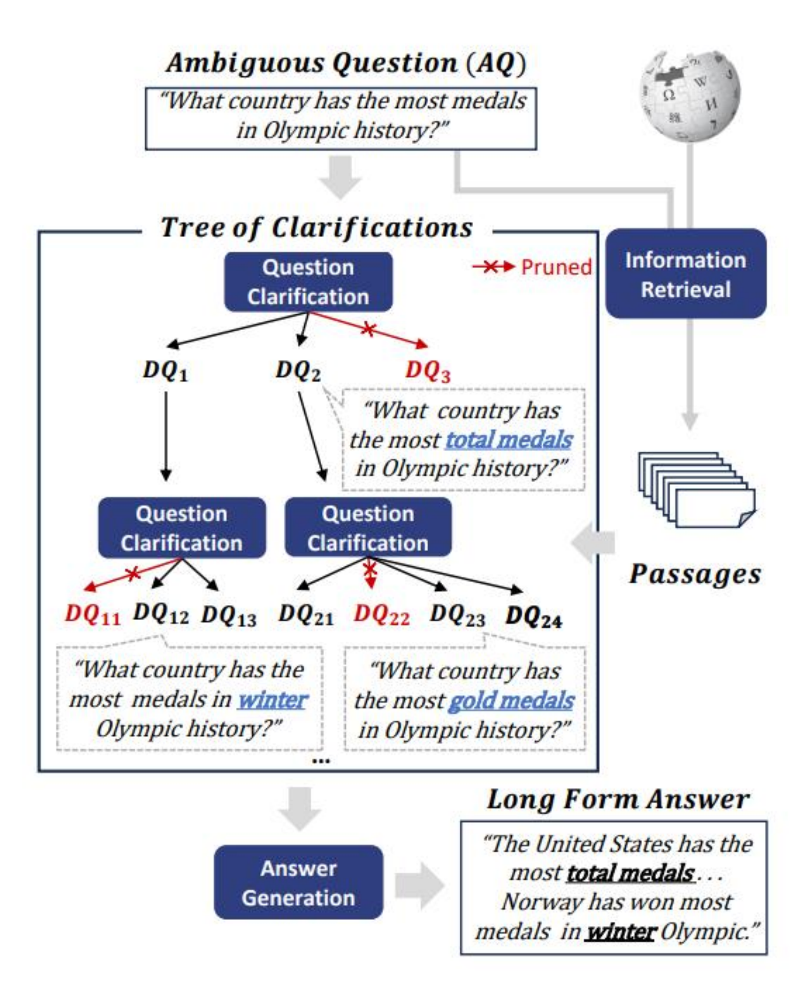


(a) Retrieve-then-read (b)Rewrite-retrieve-read (c) Trainable rewrite-retrieve-read

https://arxiv.org/pdf/2305.14283

Query Optimization

- Questions and answers do not always possess high semantic similarity; adjusting the Query can yield better retrieval results.
- Clarify the query:



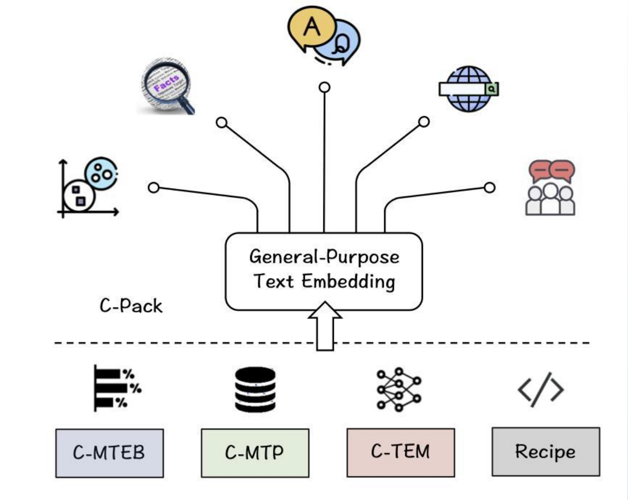
https://aclanthology.org/2023.emnlp-main.63.pdf

Embedding Optimization

- Better embedding always indicate a better retrieval results:
  - **Selecting a more suitable embedding method**
  - Fine-tuning the embedding model

Model	Retriever & Framework	HotpotQA EM	HotpotQA F1	2Wiki EM	2Wiki F1	NQ EM	NQ F1	WebQ EM	WebQ F1
LLaMA3.1 8B-Ins	BM25	25.4	37.2	16.6	21.1	26.0	32.8	22.2	31.2
	+SuRe	38.8	53.5	23.8	31.0	36.6	47.9	34.4	48.5
	+EmbQA (ours)	**42.0**	**55.8**	**27.4**	**36.6**	**42.2**	**54.4**	**38.2**	**52.1**
	DPR	20.6	21.7	10.8	13.5	25.0	34.2	23.8	34.4
	+SuRe	25.0	31.9	14.2	16.0	38.8	52.3	36.0	49.6
	+EmbQA (ours)	**29.8**	**36.3**	**16.8**	**21.0**	**43.0**	**54.4**	**38.0**	**52.0**
	Contriever	22.6	35.4	16.6	20.7	25.8	32.8	25.2	34.2
	+SuRe	33.8	50.6	21.0	29.3	39.0	52.8	34.4	48.5
	+EmbQA (ours)	**36.6**	**52.7**	**26.4**	**34.2**	**42.2**	**53.6**	**36.0**	**49.6**
Mistral v0.2 7B-Ins	BM25	21.2	29.2	13.8	21.7	18.8	25.3	19.0	26.1
	+SuRe	32.2	**46.1**	17.8	30.1	35.2	45.1	31.6	45.7
	+EmbQA (ours)	**34.8**	44.3	**18.6**	**30.5**	**35.8**	**46.0**	**35.8**	**48.1**
	DPR	7.8	11.0	3.8	4.5	22.2	26.7	18.8	27.7
	+Sure	15.0	21.8	6.4	8.5	40.0	**51.8**	32.6	**47.7**
	+EmbQA (ours)	**16.2**	**23.3**	**7.6**	**9.6**	**40.2**	49.4	**33.4**	46.0
	Contriever	19.4	28.6	13.6	20.7	21.8	27.4	17.8	24.4
	+SuRe	28.0	41.6	17.2	25.4	39.8	51.6	30.2	**45.0**
	+EmbQA (ours)	**29.8**	**42.3**	**17.4**	**26.2**	**40.6**	**51.8**	**31.6**	43.0
Qwen 2.5 7B-Ins	BM25	28.6	37.1	20.2	24.1	24.0	29.4	22.6	31.4
	+Sure	43.6	54.7	28.4	**34.1**	41.6	49.0	36.6	47.3
	+EmbQA (ours)	**44.6**	**55.6**	**28.8**	33.8	**42.4**	**49.2**	**38.2**	**48.7**
	DPR	8.8	9.8	5.6	7.1	29.2	32.6	25.6	31.1
	+Sure	21.8	27.3	12.2	16.1	45.4	54.6	38.4	49.6
	+EmbQA (ours)	**22.6**	**29.1**	**13.8**	**17.3**	**45.8**	**54.7**	**38.6**	**50.1**
	Contriever	27.0	34.0	17.6	20.0	26.6	31.9	21.0	29.1
	+Sure	38.8	**50.3**	23.8	30.4	44.0	**52.9**	36.4	48.1
	+EmbQA (ours)	**39.0**	50.2	**24.4**	**30.9**	**45.2**	50.5	**37.0**	**48.6**

Try different embedding methods in the RAG

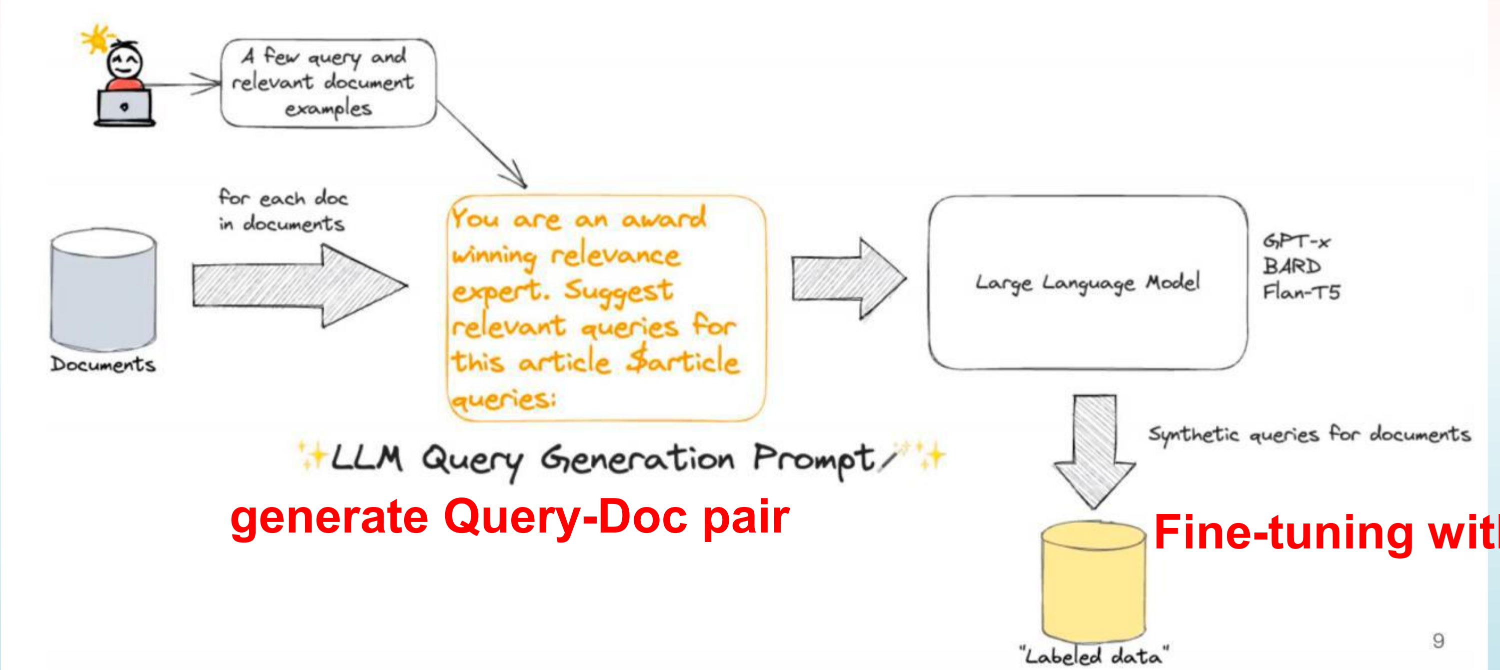


https://arxiv.org/pdf/2503.01606

Embedding Optimization

- Better embedding always indicate a better retrieval results:
  - Selecting a more suitable embedding method
  - **Fine-tuning the embedding model**

An in-context learning based method to generate prompt



generate Query-Doc pair

Fine-tuning with pseudo data

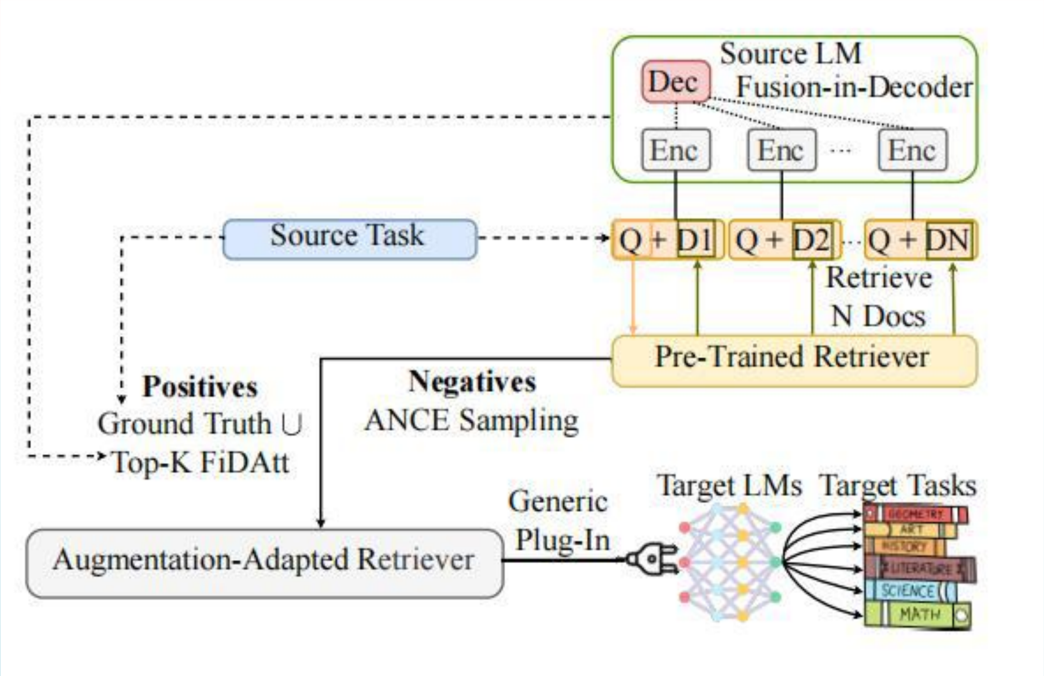
Fine-tuning on RAG

- It is also possible to train the RAG framework by fine-tuning, including:
    - Retriever fine-tuning
    - Generator fine-tuning

Fine-tuning on RAG

- It is also possible to train the RAG framework by fine-tuning, including:
  - **Retriever fine-tuning**
  - Generator fine-tuning

A small LM

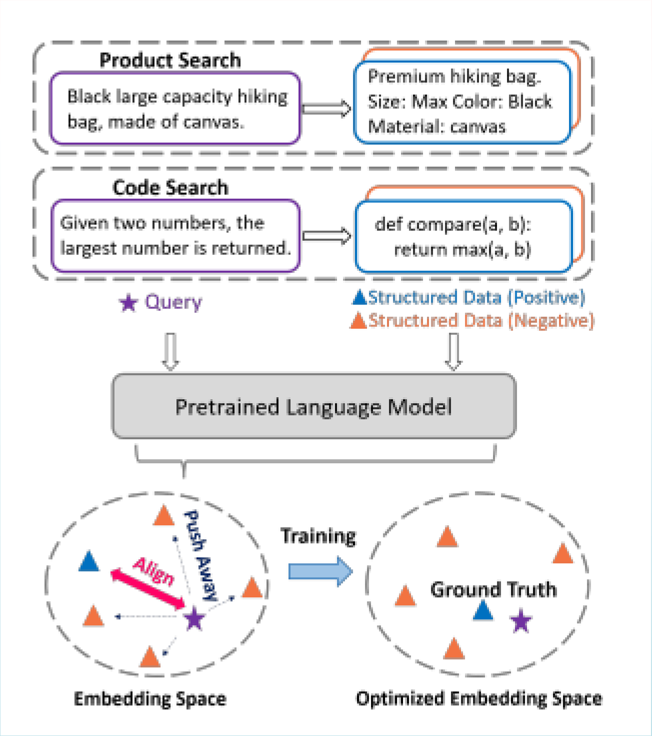


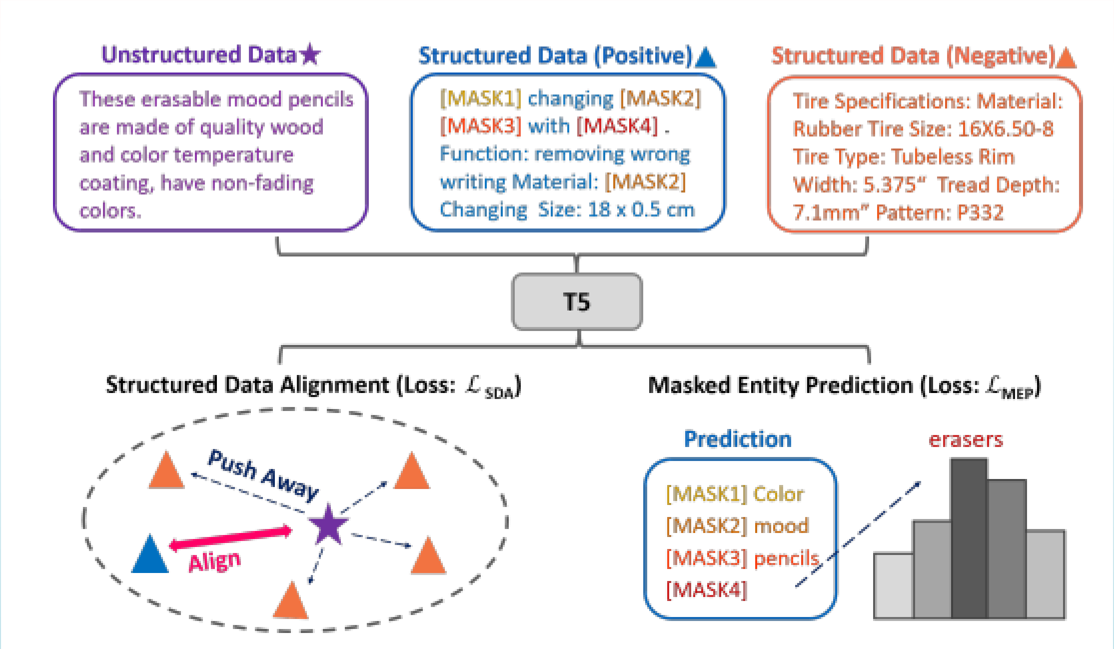
Using the attention scores annotate which documents the LM “prefers”.

https://aclanthology.org/2023.acl-long.136.pdf

Fine-tuning on RAG

- It is also possible to train the RAG framework by fine-tuning, including:
  - Retriever fine-tuning
  - **Generator fine-tuning**





Add an entity prediction loss in the fine-tuning

Thank you

Lin.1.Gui@kcl.ac.uk

www.kcl.ac.uk/people/lin-gui