Chapter 8. Semantic Search and Retrieval-Augmented Generation

Introduction

- The ability LLMs add to search engines is called semantic search, which enables searching by meaning, and not simply keyword matching.
- The fast adoption of text generation models led many users to ask the models questions and expect factual answers. And while the models were able to answer fluently and confidently, their answers were not always correct or up-to-date.
- This problem grew to be known as model “hallucinations,” and one of the leading ways to reduce it is to build systems that can retrieve relevant information and provide it to the LLM to aid it in generating more factual answers.
- There’s a lot of research on how to best use language models for search. Three broad categories of these models are dense retrieval, reranking, and RAG.

Dense Retrieval

Dense retrieval systems rely on the concept of embeddings, the same concept we've encountered in the previous chapters, and turn the search problem into retrieving the nearest neighbors of the search query (after both the query and the documents are converted into embeddings). Figure 8-1 shows how dense retrieval takes a search query, consults its archive of texts, and outputs a set of relevant results.

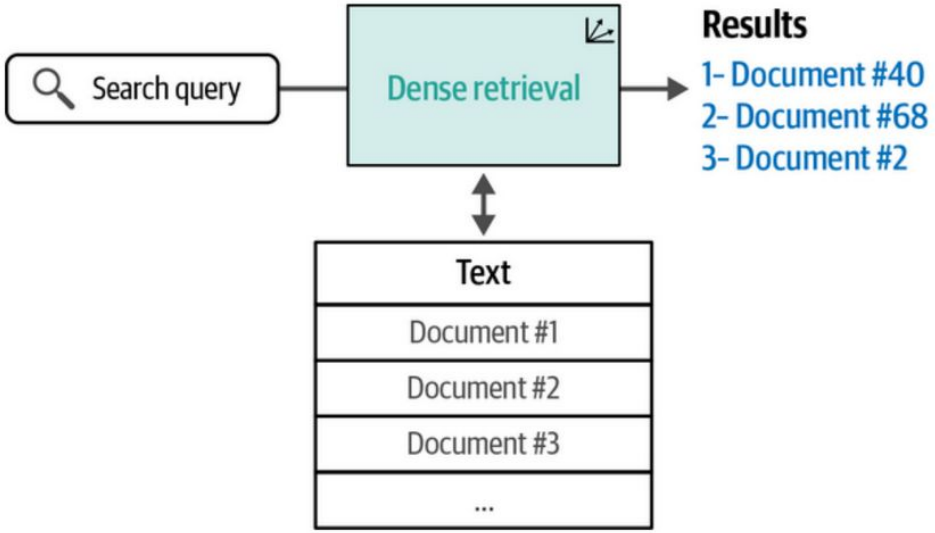


Figure 8-1. Dense retrieval is one of the key types of semantic search, relying on the similarity of text embeddings to retrieve relevant results.

Reranking

Search systems are often pipelines of multiple steps. A reranking language model is one of these steps and is tasked with scoring the

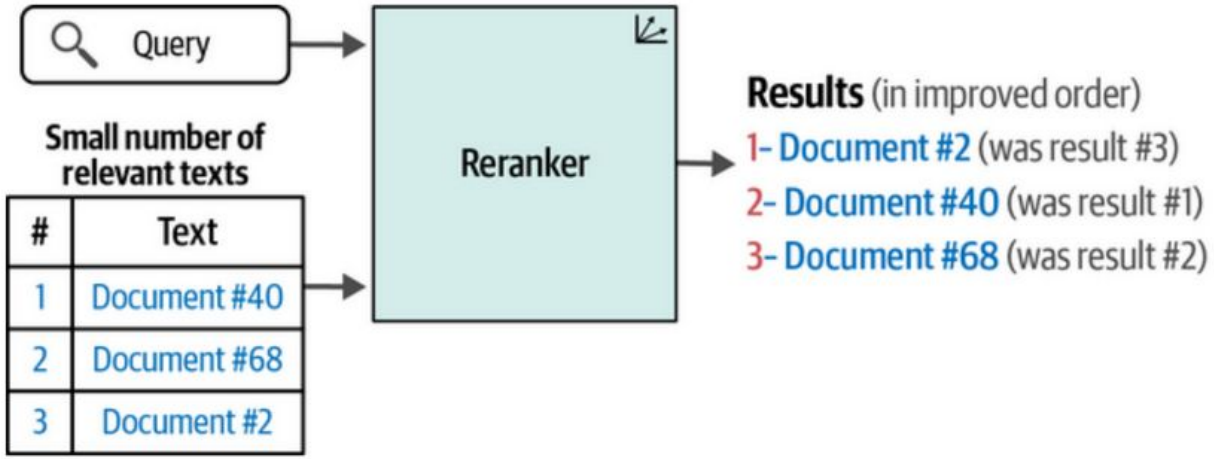


Figure 8-2. Rerankers, the second key type of semantic search, take a search query and a collection of results, and reorder them by relevance, often resulting in vastly improved results.

RAG - Retrieval Augmented Generations

Generative search is a subset of a broader type of category of systems better called RAG systems. These are text generation systems that incorporate search capabilities to reduce hallucinations, increase factuality, and/or ground the generation model on a specific dataset.

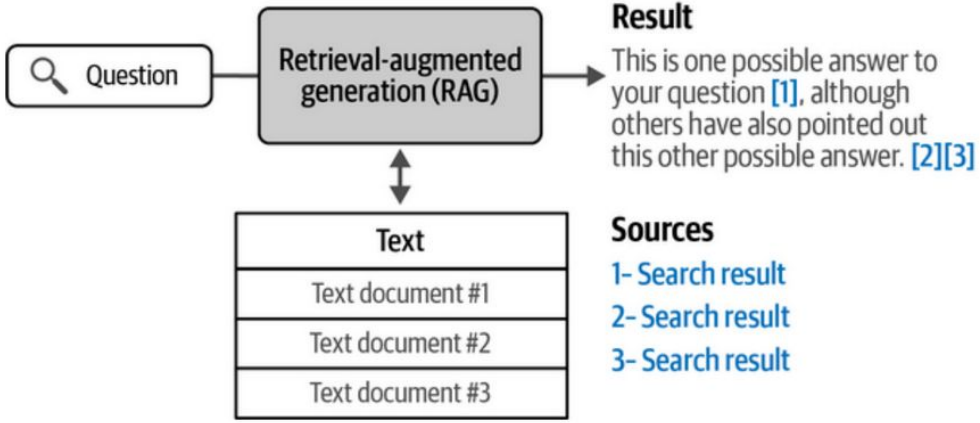


Figure 8-3. A RAG system formulates an answer to a question and (preferably) cites its information sources.

Semantic Search with Language Models using Dense Retrieval

Recall that embeddings turn text into numeric representations. Those can be thought of as points in space, as we can see in Figure 8-4. Points that are close together mean that the text they represent is similar. So in this example, text 1 and text 2 are more similar to each other (because they are near each other) than text 3 (because it's farther away).

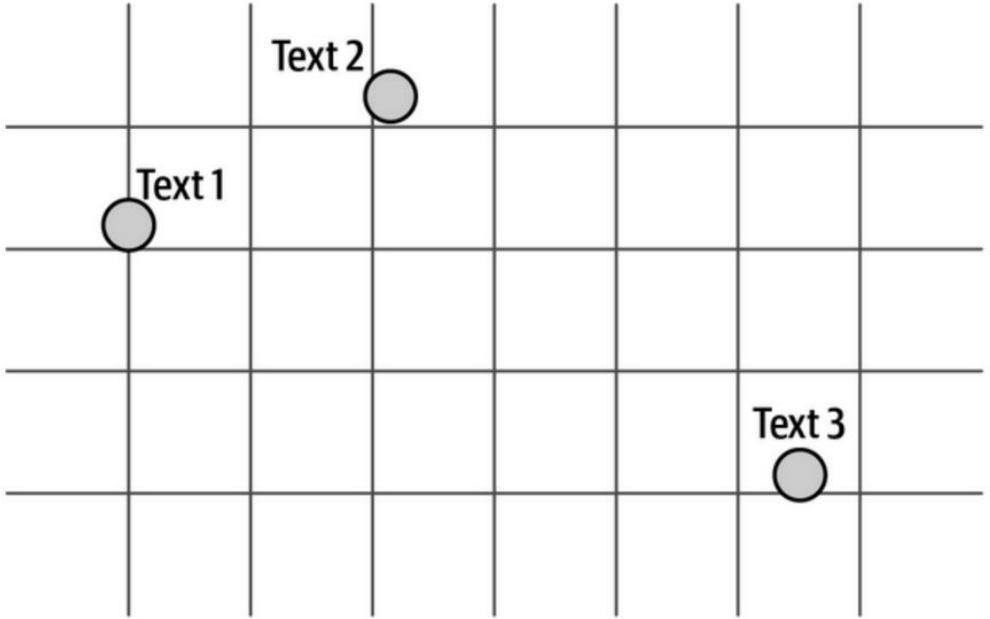


Figure 8-4. The intuition of embeddings: each text is a point and texts with similar meaning are close

Semantic Search with Language Models using Dense Retrieval

This is the property that is used to build search systems. In this scenario, when a user enters a search query, we embed the query, thus projecting it into the same space as our text archive. Then we simply find the nearest documents to the query in that space, and those would be the search results (Figure 8-5).

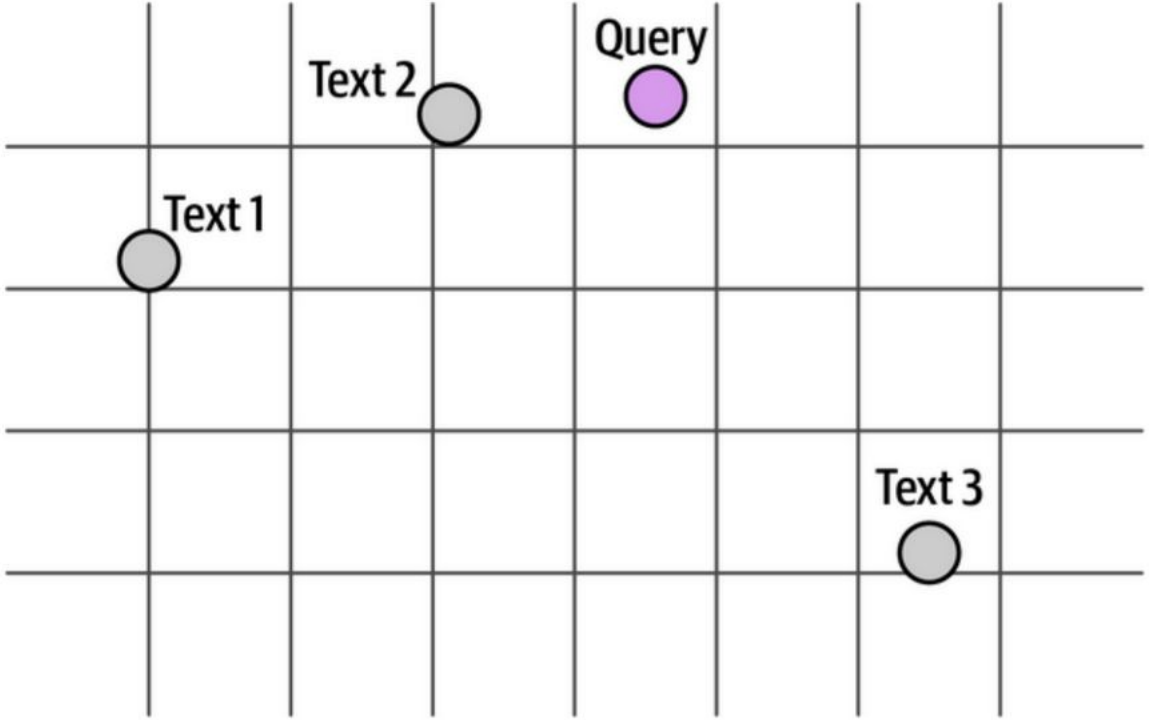


Figure 8-5. Dense retrieval relies on the property that search queries will be close to their relevant results.

Dense Retrieval - Design Objectives

Judging by the distances in Figure 8-5, “text 2” is the best result for this query, followed by “text 1.” Two questions could arise here, however:

- Should text 3 even be returned as a result? That’s a decision for you, the system designer. It’s sometimes desirable to have a max threshold of similarity score to filter out irrelevant results (in case the corpus has no relevant results for the query).

- Are a query and its best result semantically similar? Not always. This is why language models need to be trained on question-answer pairs to become better at retrieval. This process is explained in more detail in Chapter 10.

Dense Retrieval - Steps using Cohere API

Dense retrieval example

Let's take a look at a dense retrieval example by using Cohere to search the Wikipedia page for the film *Interstellar*. In this example, we will do the following:

1. Get the text we want to make searchable and apply some light processing to chunk it into sentences.
2. Embed the sentences.
3. Build the search index.
4. Search and see the results.

Dense Retrieval - Steps using Cohere API

Results:

```
Input question: how precise was the science
Top-3 lexical search (BM25) hits
      1.789 Interstellar is a 2014 epic science fiction film
co-written, directed, and produced by Christopher Nolan
      1.373 Caltech theoretical physicist and 2017 Nobel
laureate in Physics[4] Kip Thorne was an executive producer,
acted as a scientific consultant, and wrote a tie-in book, The
Science of Interstellar
      0.000 It stars Matthew McConaughey, Anne Hathaway,
Jessica Chastain, Bill Irwin, Ellen Burstyn, Matt Damon, and
Michael Caine
```

Note that the first result does not really answer the question despite it sharing the word “science” with the query. In the next section, we’ll see how adding a reranker can improve this search system. But before that, let’s complete our overview of dense retrieval by looking at its caveats and go over some methods of breaking down texts into chunks.

Caveats of dense retrieval

What happens, for example, if the texts don't contain the answer? We still get results and their distances. For example:

```
Query:'What is the mass of the moon?'
Nearest neighbors:
```

	texts	distance
0	The film had a worldwide gross over $677 million (and $773 million with subsequent re-releases), making it the tenth-highest grossing film of 2014	1.298275
1	It has also received praise from many astronomers for its scientific accuracy and portrayal of theoretical astrophysics	1.324389
2	Cinematographer Hoyte van Hoytema shot it on 35 mm movie film in the Panavision anamorphic format and IMAX 70 mm	1.328375

In cases like this, one possible heuristic is to set a threshold level—a maximum distance for relevance.

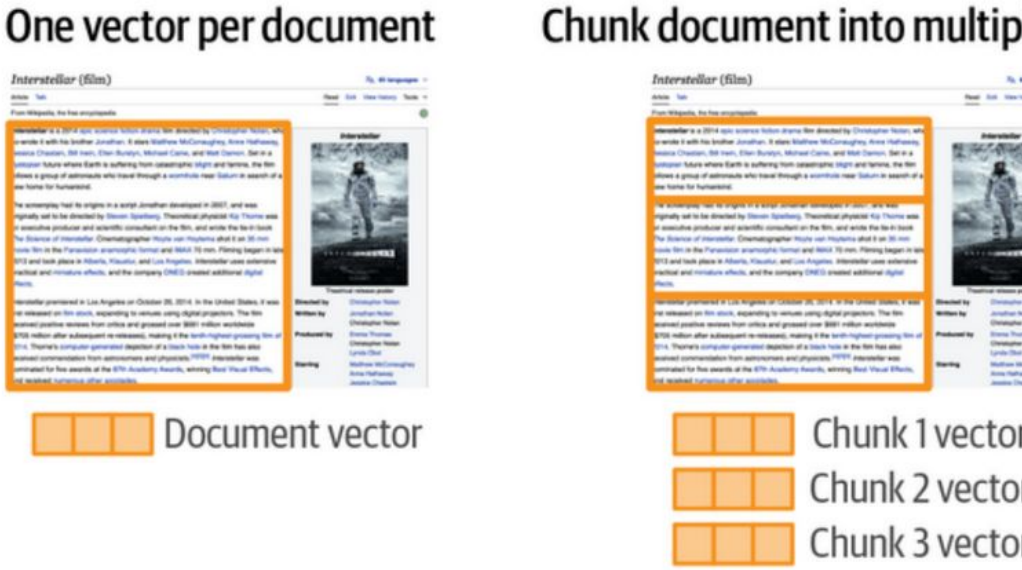
Caveats of dense retrieval

- Another caveat of dense retrieval is when a user wants to find an exact match for a specific phrase.
- Dense retrieval systems also find it challenging to work properly in domains other than the ones that they were trained on. So, for example, if you train a retrieval model on internet and Wikipedia data, and then deploy it on legal texts (without having enough legal data as part of the training set), the model will not work as well in that legal domain.
- What about questions whose answers span multiple sentences? This highlights one of the important design parameters of dense retrieval systems: what is the best way to chunk long texts? And why do we need to chunk them in the first place?

How to Chunk Documents

A downside of this approach is that it results in a highly compressed vector that loses a lot of the information in the document. This approach can satisfy some information needs, but not others. A lot of
the time, a search is for a specific piece of information contained in an
article, which is better captured if the concept had its own vector.

There are several possible ways, and two possible approaches shown in Figure 8-7 include indexing one vector per document and indexing multiple vectors per document.



In this approach, we chunk the document into smaller pieces, and embed those chunks. Our search index then becomes that of chunk embeddings, not entire document embeddings.

Figure 8-7. It's possible to create one vector representing an entire document, but it's better for longer documents to be split into smaller chunks that get their own embeddings.

Chunking a Document into Multiple Chunks

The chunking approach is better because it has full coverage of the text and
because the vectors tend to capture individual concepts inside the text.

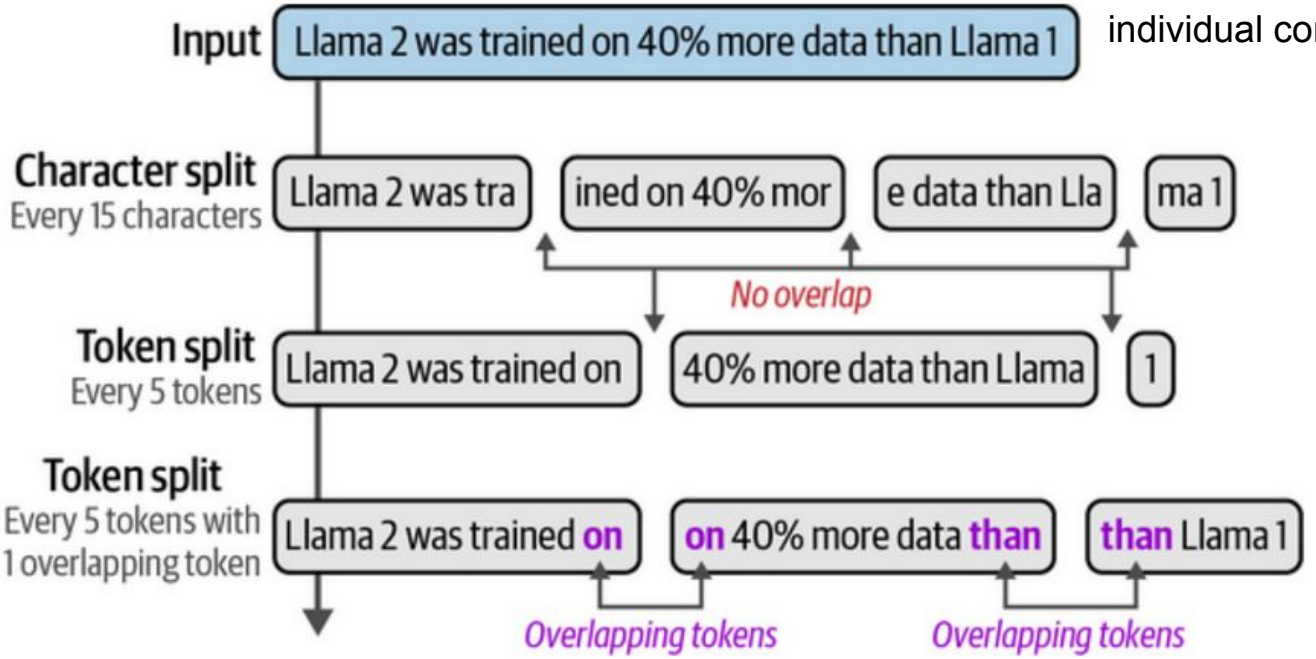


Figure 8-8. Several chunking methods and their effects on the input text. Overlapping chunks can be important to prevent the absence of context.

One vector chunking vs. Multiple vectors chunking

The best way of chunking a long text will depend on the types of texts and queries your system anticipates. Approaches include:

- Each sentence is a chunk. The issue here is this could be too granular and the vectors don’t capture enough of the context.
- Each paragraph is a chunk. This is great if the text is made up of short paragraphs. Otherwise, it may be that every 3–8 sentences is a chunk.
- Some chunks derive a lot of their meaning from the text around them. So we can incorporate some context via:
    - Adding the title of the document to the chunk.
    - Adding some of the text before and after them to the chunk.

This way, the chunks can overlap so they include some surrounding text that also appears in adjacent chunks.

How the nearest vectors are found in Dense Vectors

Once the query is embedded, we need to find the nearest vectors to it from our text archive as we can see in Figure 8-11. The most straightforward way to find the nearest neighbors is to calculate the distances between the query and the archive.

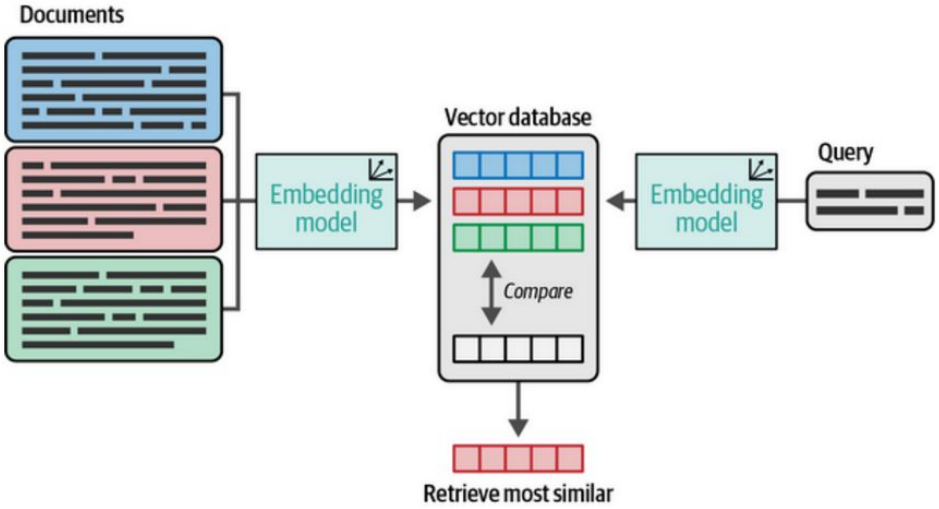


Figure 8-11. As we saw in Chapter 3, we can compare embeddings to quickly find the most similar documents to a query.

As you scale beyond to the millions of vectors, an optimized approach for retrieval is to rely on approximate nearest neighbor search libraries like Annoy or FAISS.

Another class of vector retrieval systems are vector databases like Weaviate or Pinecone.

Reranking

- A lot of organizations have already built search systems. For those organizations, an easier way to incorporate language models is as a final step inside their search pipeline.
- The first-stage retriever can be keyword search, dense retrieval, or better yet —hybrid search that uses both of them.
- This step is tasked with changing the order of the search results based on relevance to the search query. This one step can vastly improve search results and it's in fact what Microsoft Bing added to achieve the improvements to search results using BERT-like models.

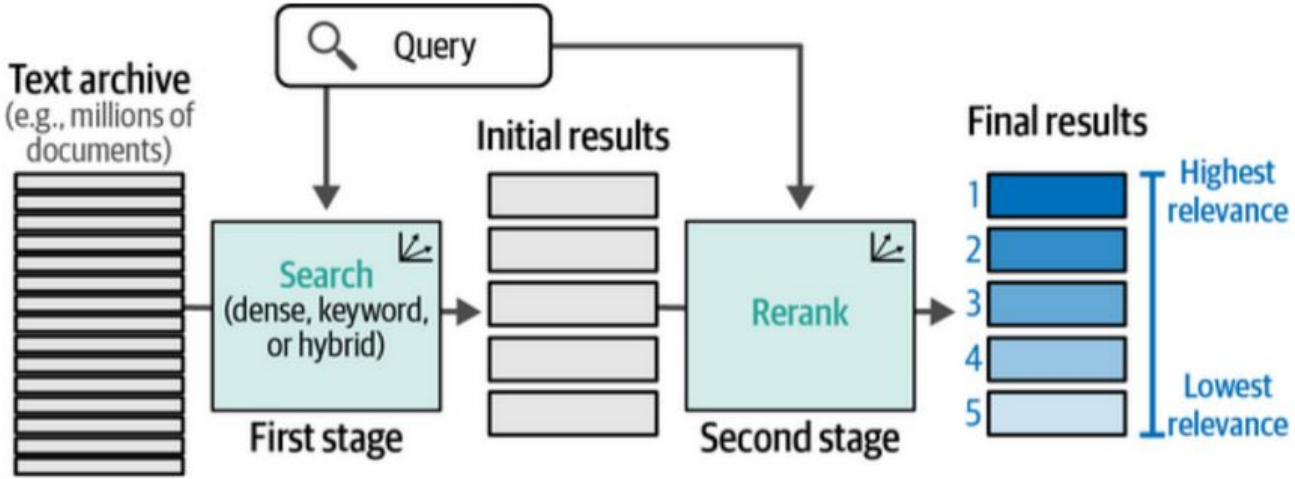


Figure 8-14. LLM rerankers operate as part of a search pipeline with the goal of reordering a number of shortlisted search results by relevance.

Improvements to the dense retrieval results using Reranking

```
query = "how precise was the science"
results = co.rerank(query=query, documents=texts, top_n=3,
return_documents=True)
results.results
```

Output:

```
0 0.1698185 It has also received praise from many astronomers
for its scientific accuracy and portrayal of theoretical
astrophysics
1 0.07004896 The film had a worldwide gross over $677 million
(and $773 million with subsequent re-releases), making it the
tenth-highest grossing film of 2014
2 0.0043994132 Caltech theoretical physicist and 2017 Nobel
laureate in Physics[4] Kip Thorne was an executive producer,
acted as a scientific consultant, and wrote a tie-in book, The
Science of Interstellar
```

This shows the reranker is much more confident about the first result, assigning it a relevance score of 0.16, while the other results are scored much lower in relevance.

How reranking models work

One popular way of building LLM search rerankers is to present the query and each result to an LLM working as a cross-encoder. This means that a query and possible result are presented to the model at the same time allowing the model to view both these texts before it assigns a relevance score, as we can see in Figure 8-15.

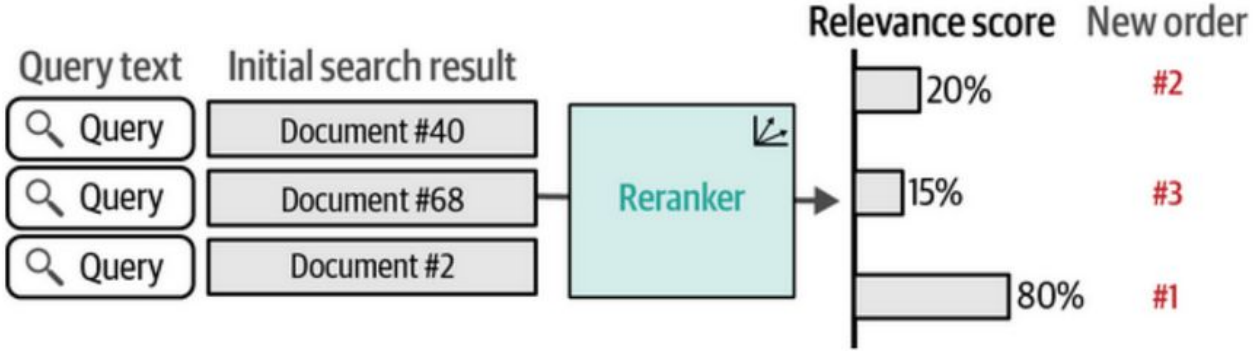


Figure 8-15. A reranker assigns a relevance score to each document by looking at the document and the query at the same time.

This formulation of search as relevance scoring basically boils down to being a classification problem. Given those inputs, the model outputs a score from 0–1 where 0 is irrelevant and 1 is highly relevant.

Retrieval Evaluation Metrics – Mean Average Precision (MAP).

Evaluating search systems needs three major components: a text archive, a set of queries, and relevance judgments indicating which documents are relevant for each query.

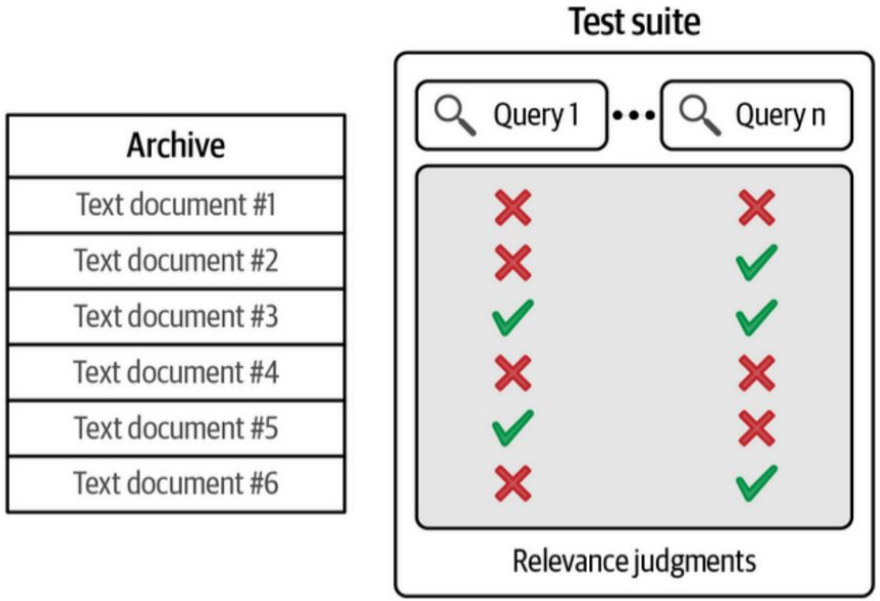


Figure 8-16. To evaluate search systems, we need a test suite including queries and relevance judgments indicating which documents in our archive are relevant for each query.

Retrieval Evaluation Metrics - Mean Average Precision (MAP).

Let's assume we pass query 1 to two different search systems. And get two sets of results. To tell which is a better system, we turn to the relevance judgments that we have for the query.

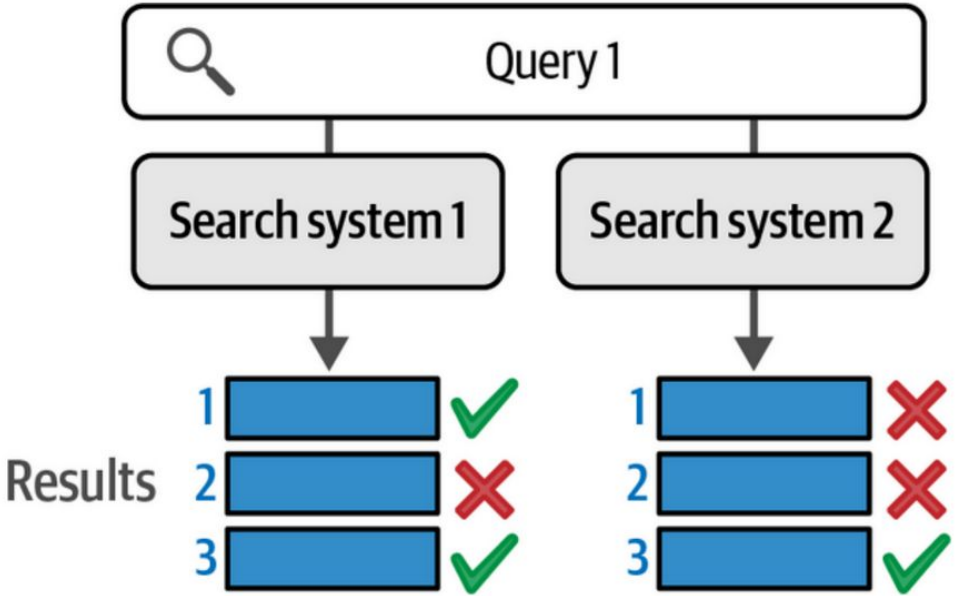


Figure 8-18. Looking at the relevance judgments from our test suite, we can see that system 1 did a better job than system 2.

This shows us a clear case where system 1 is better than system 2.

How MAP scoring is done?

- How can we assign a number or score to how much better that result is? Mean average precision is a measure that is able to quantify this distinction.

- One common way to assign numeric scores in this scenario is average precision, which evaluates system 1's result for the query to be 1 and system 2's to be 0.3. So let's see how average precision is calculated

Scoring a single query with average precision for System 1

- The first one is easy: the search system placed the relevant result (the only available one for this query) at the top. This gets the system the perfect score of 1.

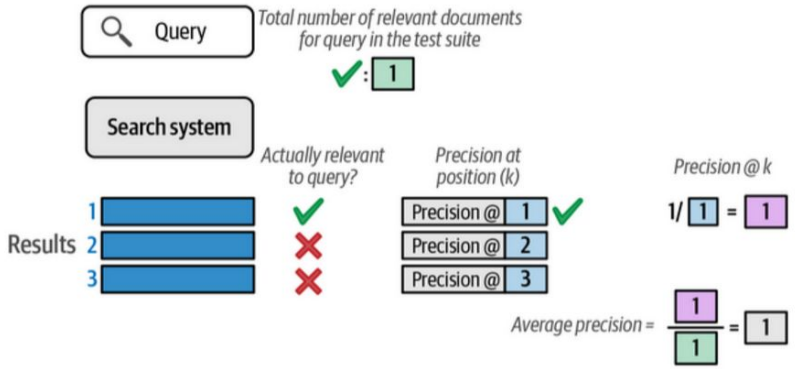


Figure 8-20. To calculate mean average precision, we start by calculating precision at each position, starting with position 1.

How MAP scoring is done?

- **Scoring a single query with average precision for System 2**

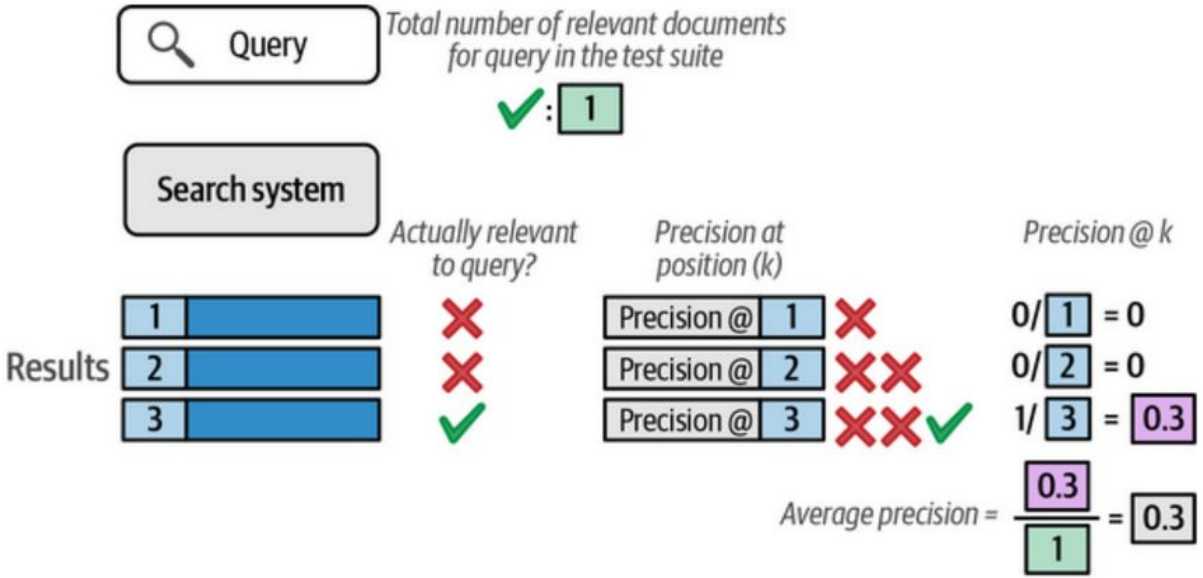


Figure 8-21. If the system places nonrelevant documents ahead of a relevant document, its precision score is penalized.

How MAP scoring is done?

Let's now look at a query with more than one relevant document. Figure 8-22 shows that calculation and how averaging now comes into the picture.

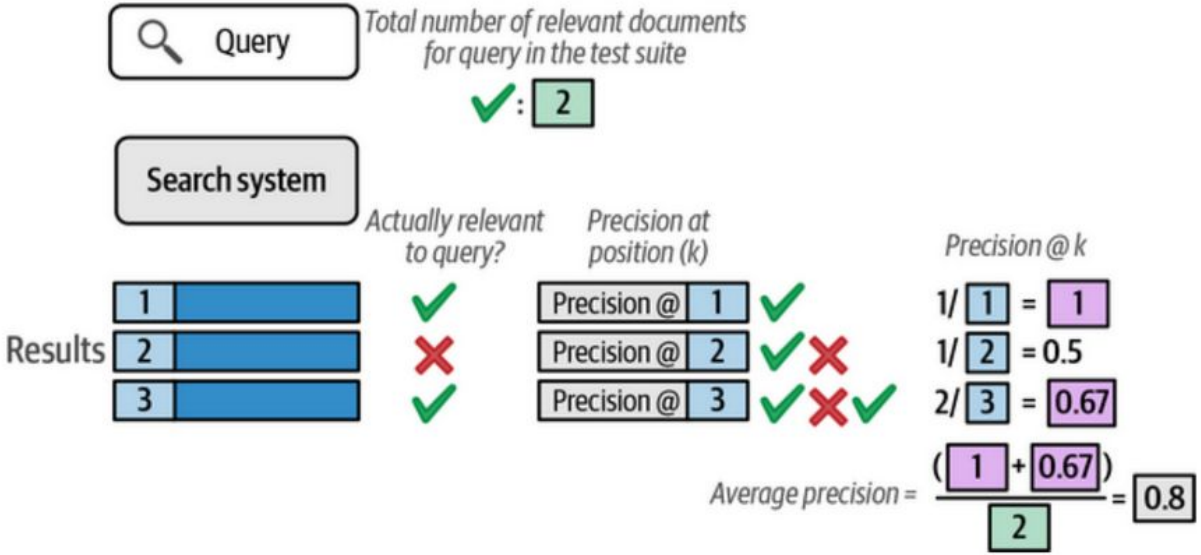


Figure 8-22. Average precision of a document with multiple relevant documents considers the precision at k results of all the relevant documents.

How MAP scoring is done?

Scoring across multiple queries with mean average precision

Figure 8-23 shows how to calculate this metric by taking the mean of the average precisions of each query.

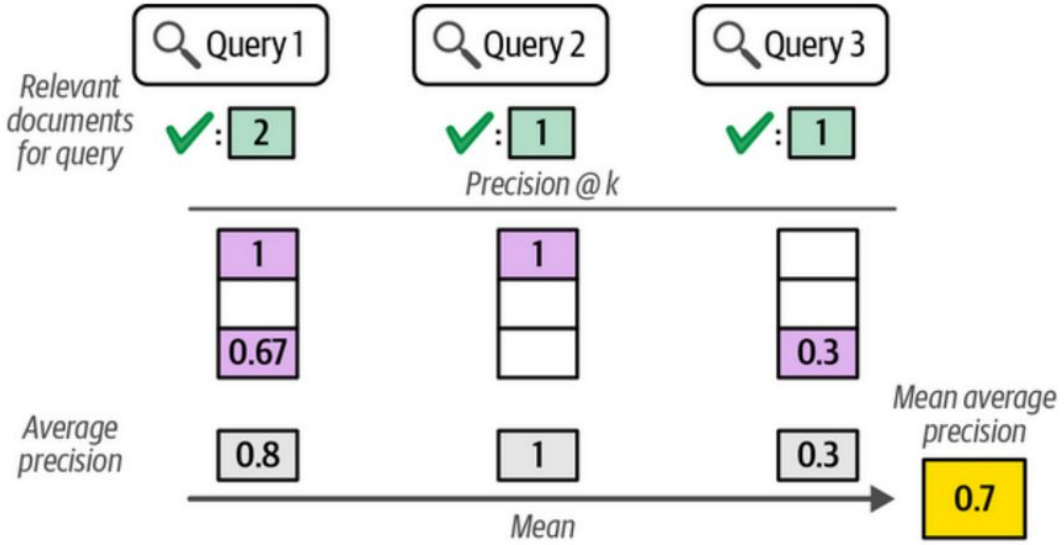


Figure 8-23. The mean average precision takes into consideration the average precision score of a system for every query in the test suite. By averaging them, it produces a single metric that we can use to compare a search system against another.

Retrieval-Augmented Generation (RAG)

- The mass adoption of LLMs quickly led to people asking them questions and expecting factual answers. While the models can answer some questions correctly, they also confidently answer lots of questions incorrectly.
- RAG systems incorporate search capabilities in addition to generation capabilities. They can be seen as an improvement to generation systems because they reduce their hallucinations and improve their factuality.

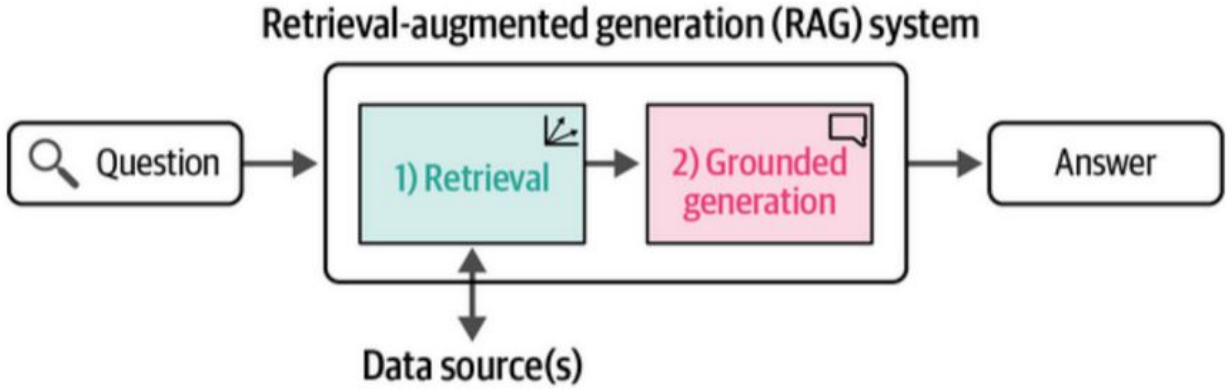


Figure 8-24. A basic RAG pipeline is made up of a search step followed by a grounded generation step where the LLM is prompted with the question and the information retrieved from the search step.

They also enable use cases of “chat with my data” that consumers and companies can use to ground an LLM on internal company data, or a specific data source of interest (e.g., chatting with a book).

From Search to RAG

- Let's now turn our search system into a RAG system. We do that by adding an LLM to the end of the search pipeline.

- We present the question and the top retrieved documents to the LLM, and ask it to answer the question given the context provided by the search results.

- This generation step is called grounded generation because the retrieved relevant information we provide the LLM establishes a certain context that grounds the LLM in the domain we're interested in.

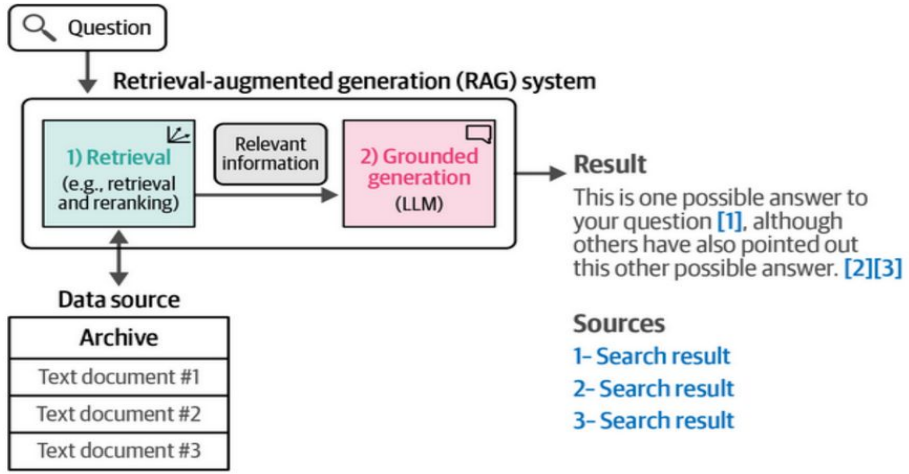


Figure 8-25. Generative search formulates answers and summaries at the end of a search pipeline while citing its sources (returned by the previous steps in the search system).

From Search to RAG

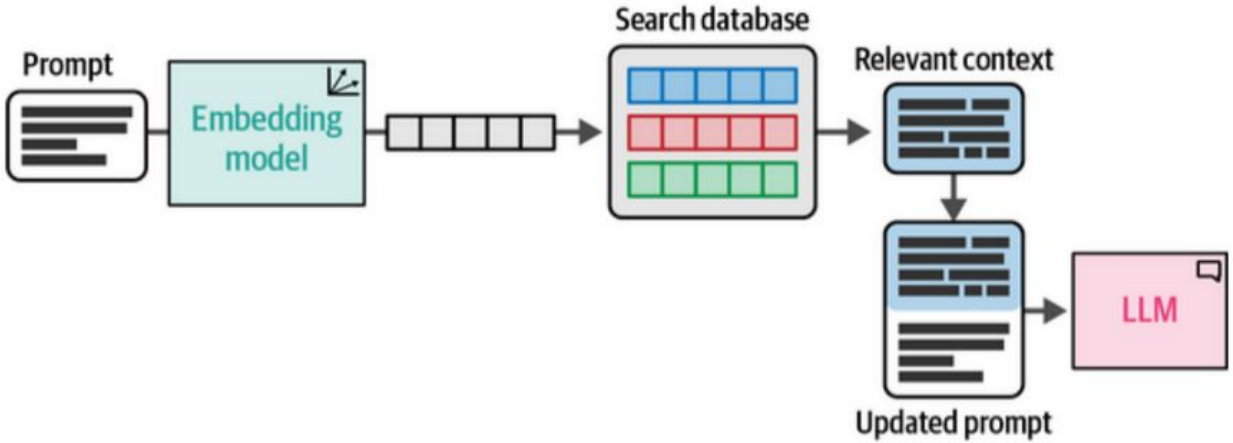


Figure 8-26. Find the most relevant information to an input prompt by comparing the similarities between embeddings. The most relevant information is added to the prompt before giving it to the LLM.

Grounded Generation with an LLM API

we'll use Cohere's managed LLM, which builds on the search systems we've seen earlier in the chapter. We'll use embedding search to retrieve the top documents, then we'll pass those to the co.chat endpoint along with the questions to provide a grounded answer:

```
rag.invoke('Income generated')
```

Result:

```
The Income generated by the film in 2014 was over $677 million
worldwide. This made it the tenth-highest grossing film of that
year. It should be noted, however, this figure includes both
initial ticket sales as well as any subsequent re-releases.
With these additional releases, total earnings surged to
approximately $773 million. The release format transitioned
```

```
from traditional film stock projection in theaters to digital
projectors once it was expanded to various venues in the United
States. This shift might have contributed to wider audience
reach and potentially higher grossing figures over time.
However, specific [illegible] this affected total earnings
isn't provided in the information above.
```

Advanced RAG Techniques

Query rewriting:

If the RAG system is a chatbot, the preceding simple RAG implementation would likely struggle with the search step if a question is too verbose, or to refer to context in previous messages in the conversation.

> *User Question: "We have an essay due tomorrow. We have to write about some animal. I love penguins. I could write about them. But I could also write about dolphins. Are they animals? Maybe. Let's do dolphins. Where do they live for example?"*

This should actually be rewritten into a query like:

> *Query: "Where do dolphins live"*

This rewriting behavior can be done through a prompt (or through an API call). Cohere's API, for example, has a dedicated query-rewriting mode for `co.chat`.

Advanced RAG Techniques

Multi-query RAG:

The next improvement we can introduce is to extend the query rewriting to be able to search multiple queries if more than one is needed to answer a specific question.

User Question: "Compare the financial results of Nvidia in 2020 vs. 2023"

We may find one document that contains the results for both years, but more likely, we're better off making two search queries:

Query 1: "Nvidia 2020 financial results"

Query 2: "Nvidia 2023 financial results"

We then present the top results of both queries to the model for grounded generation. An additional small improvement here is to also give the query rewriter the option to determine if no search is required and if it can directly generate a confident answer without searching.

Advanced RAG Techniques

Multi-hop RAG:

A more advanced question may require a series of sequential queries. Take for example a question like:

> *User Question: "Who are the largest car manufacturers in 2023? Do they each make EVs or not?"*
>
> To answer this, the system must first search for:
>
> *Step 1, Query 1: "largest car manufacturers 2023"*
>
> Then after it gets this information (the result being Toyota, Volkswagen, and Hyundai), it should ask follow-up questions:
>
> *Step 2, Query 1: "Toyota Motor Corporation electric vehicles"*
>
> *Step 2, Query 2: "Volkswagen AG electric vehicles"*
>
> *Step 2, Query 3: "Hyundai Motor Company electric vehicles"*

Advanced RAG Techniques

Query routing:

An additional enhancement is to give the model the ability to search multiple data sources. We can, for example,

- specify for the model that if it gets a question about HR, it should search the company's HR information system (e.g., Notion) but if the question is about customer data, that it should search the customer relationship management (CRM) (e.g., Salesforce).

Advanced RAG Techniques

Agentic RAG:

You may be able to now see that the list of previous enhancements slowly delegates more and more responsibility to the LLM to solve more and more complex problems. This relies on the LLM's capability to gauge the required information needs as well as its ability to utilize multiple data sources. This new nature of the LLM starts to become closer and closer to an agent that acts on the world. The data sources can also now be abstracted into tools.

RAG Evaluation

It evaluates results along four axes:

- Fluency - Whether the generated text is fluent and cohesive.
- Perceived utility - Whether the generated answer is helpful and informative.
- Citation recall - The proportion of generated statements about the external world that are fully supported by their citations.
- Citation precision - The proportion of generated citations that support their associated statements.

Ragas is a software library that does exactly this. It also scores some additional useful metrics like:

- Faithfulness - Whether the answer is consistent with the provided context
- Answer relevance - How relevant the answer is to the question