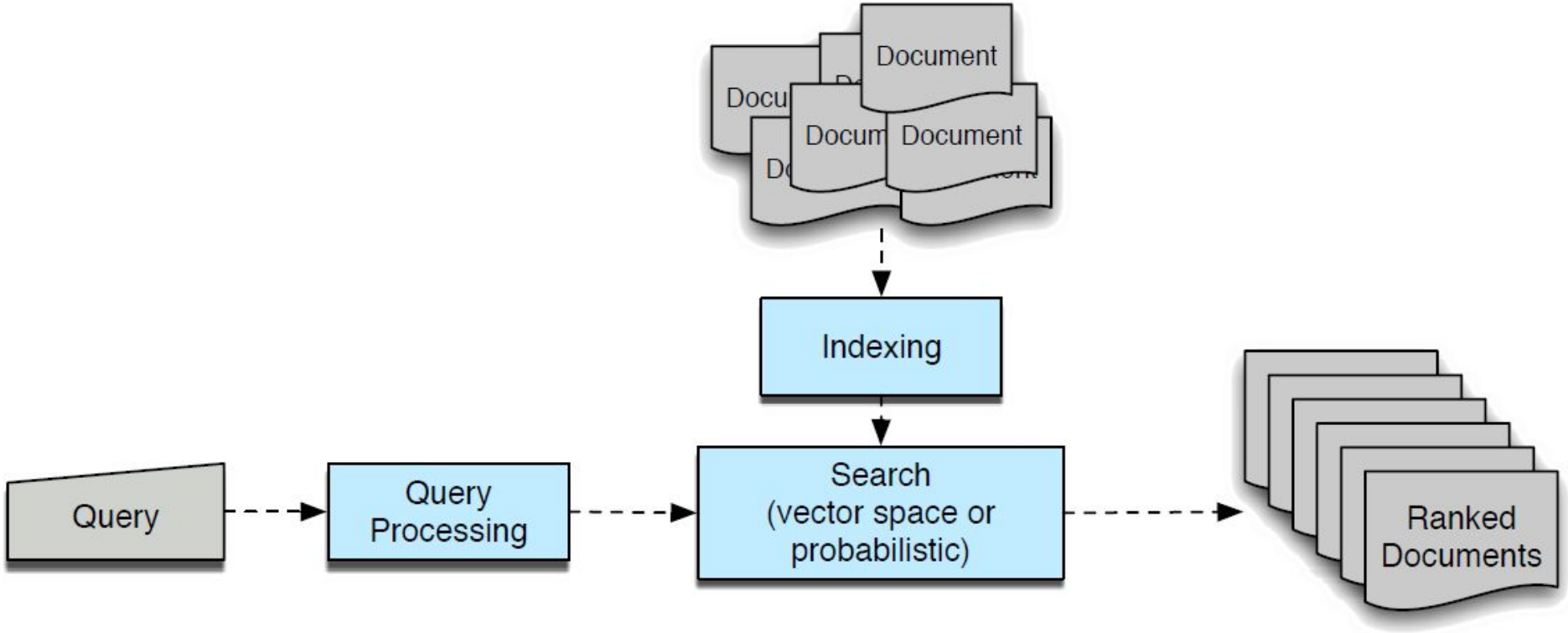
RETRIEVAL AUGMENTED GENERATION

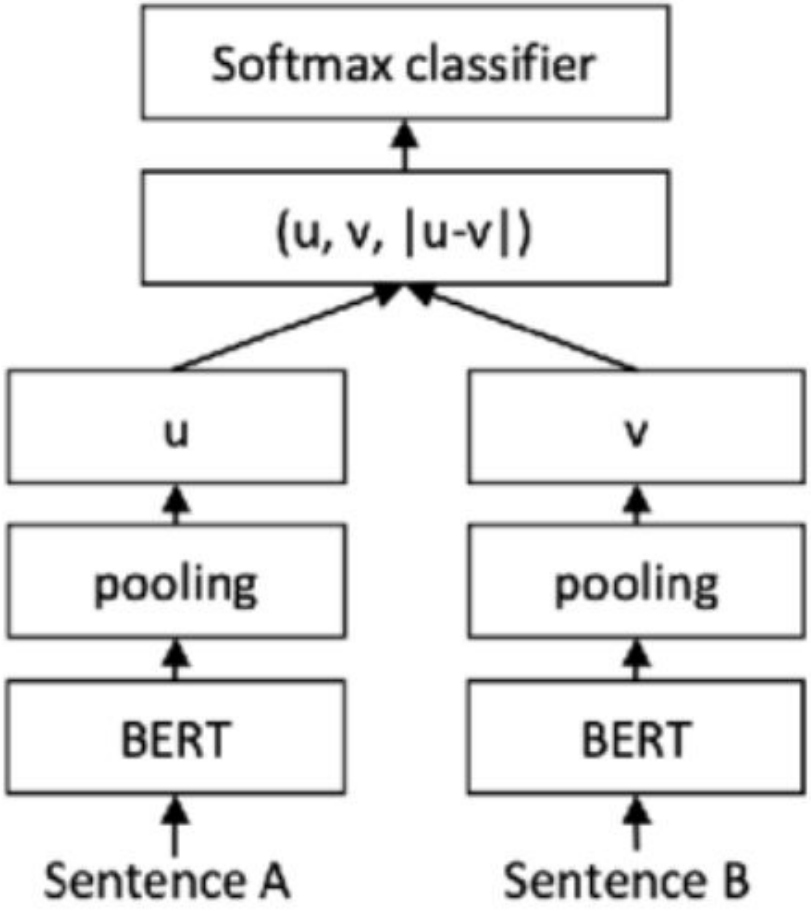
[Modern Information Retrieval]

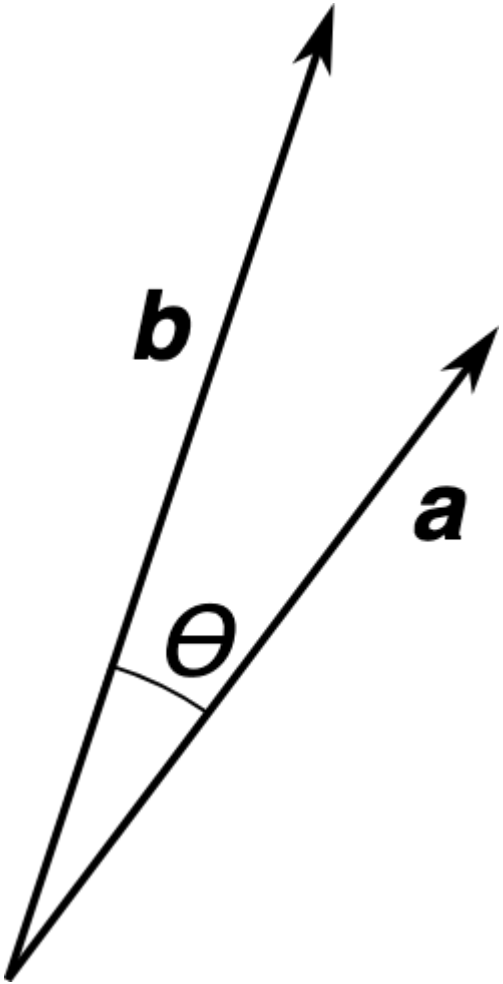
Open problem with a billion dollar potential!

INFORMATION RETRIEVAL



SBERT Fine-Tuning





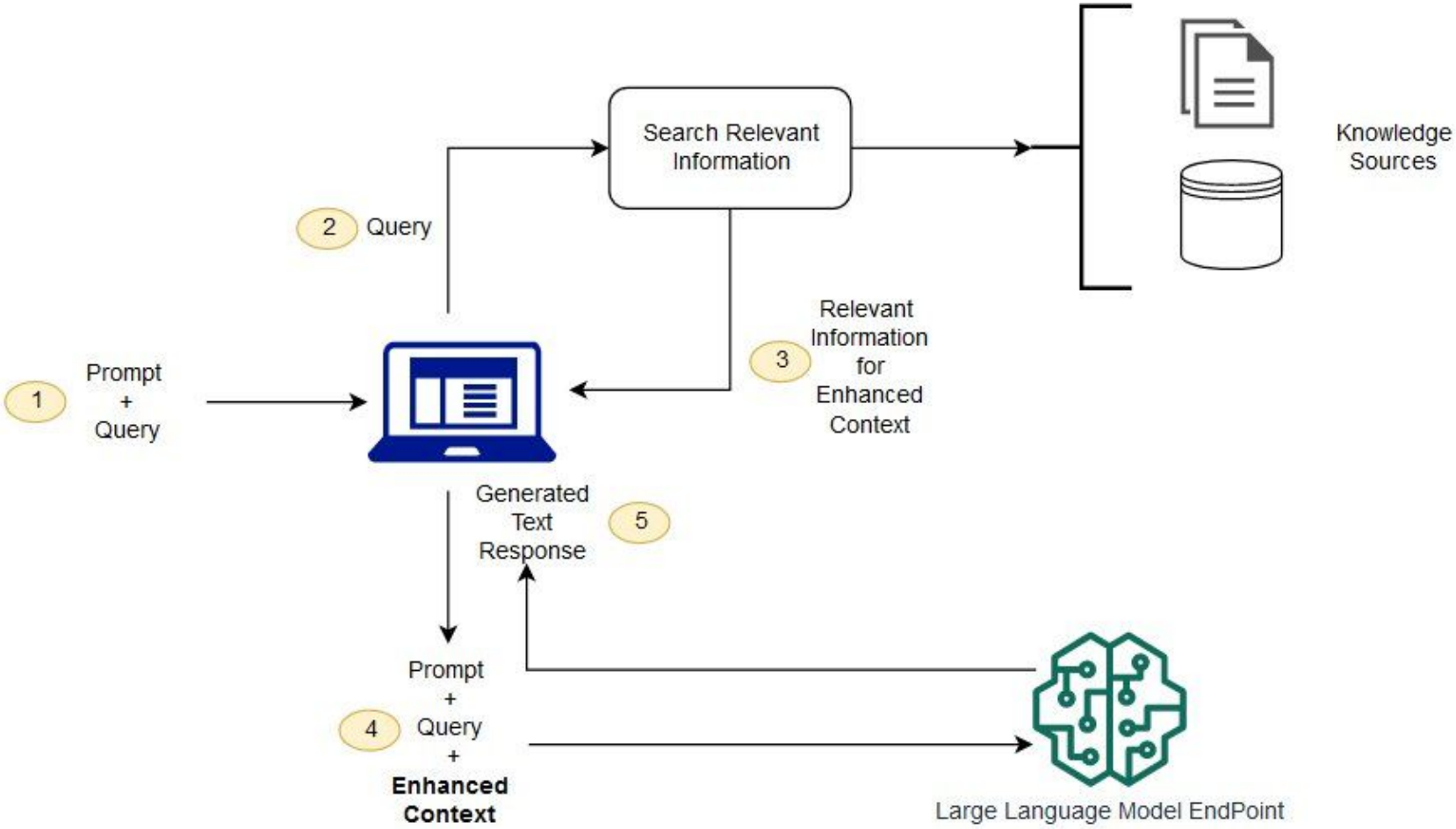
- The query has a vector representation using embeddings
- Documents in the database stored as embeddings

- **Brute Force Approach:**
  Do a dot product of the query vector with the embeddings of all the documents, and choose the one that gives the closest match

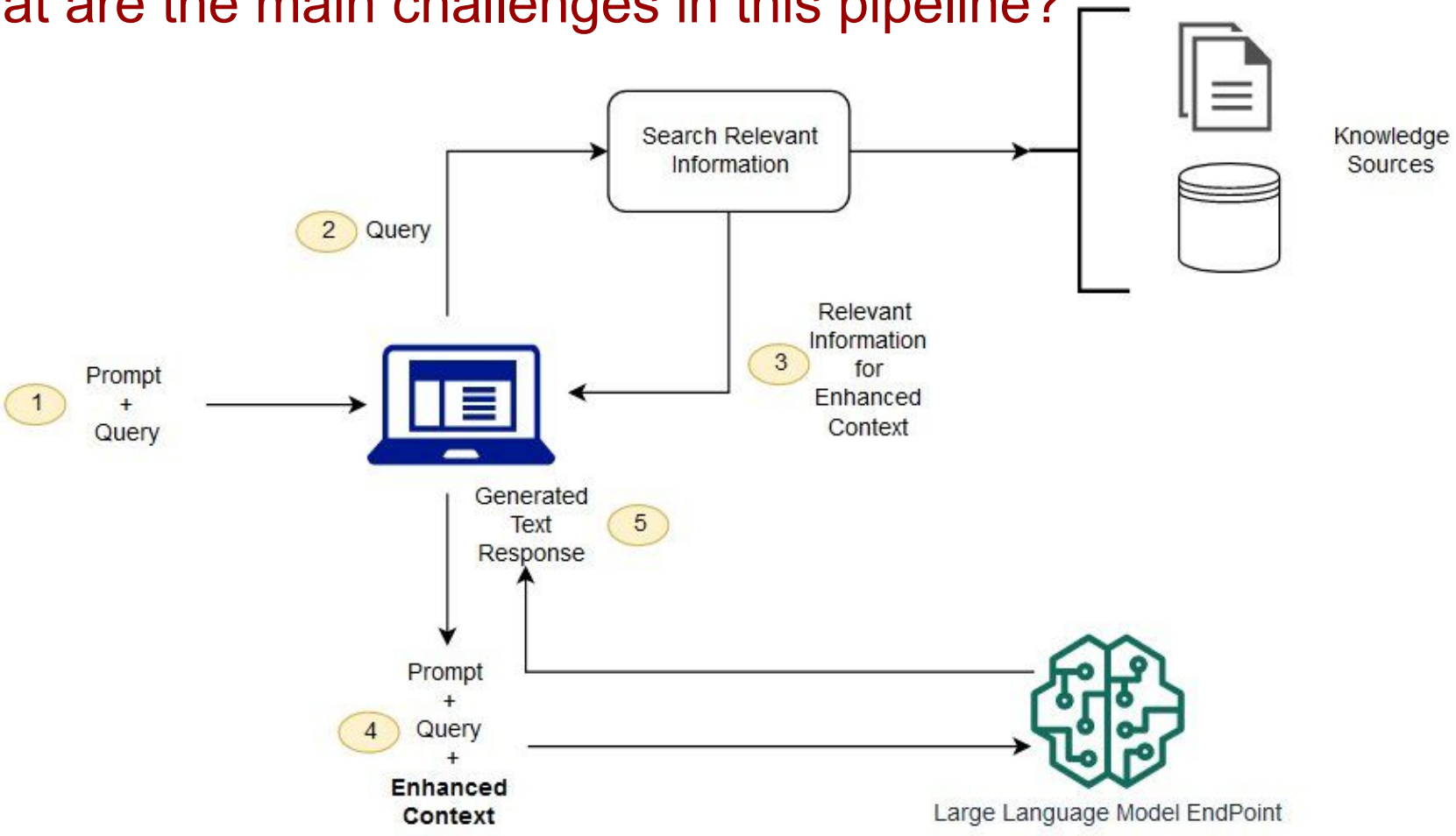
- **Hierarchical Navigable Small World (HNSW):**
  Create a layered graph structure of the document embedding vectors so that the search process is made much faster

Retrieval Augmented Generation [RAG]

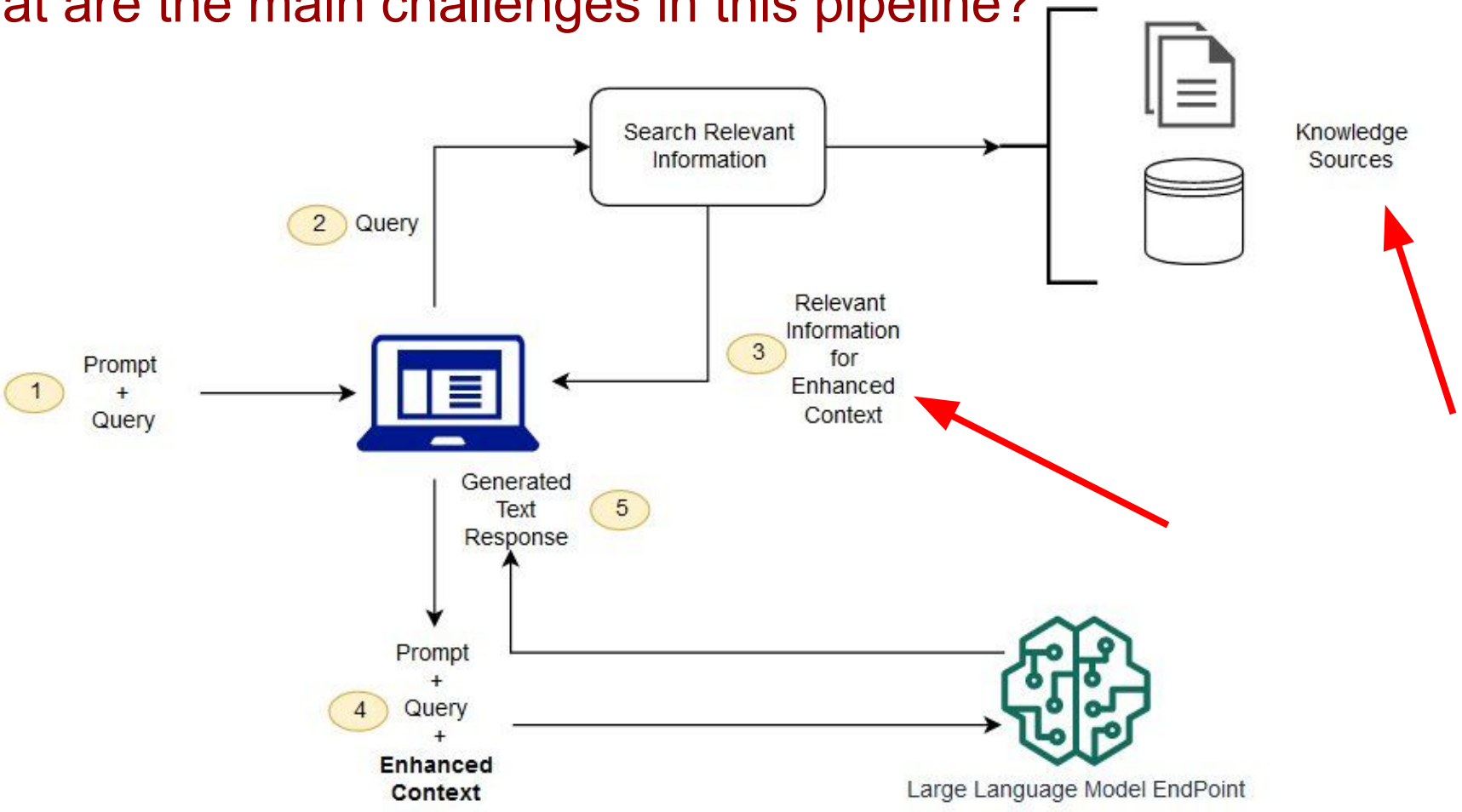
- New data gets generated at a rapid pace
- Re-Training LLMs is expensive
- LLM Hallucinations
- Solution is to extract relevant information from trusted sources and then use LLMs to generate a concise response


Search Relevant Information
Knowledge Sources
2 Query
Relevant Information for Enhanced Context
3
1 Prompt + Query
Generated Text Response
5
Prompt + Query + Enhanced Context
4
Large Language Model EndPoint

What are the main challenges in this pipeline?



What are the main challenges in this pipeline?



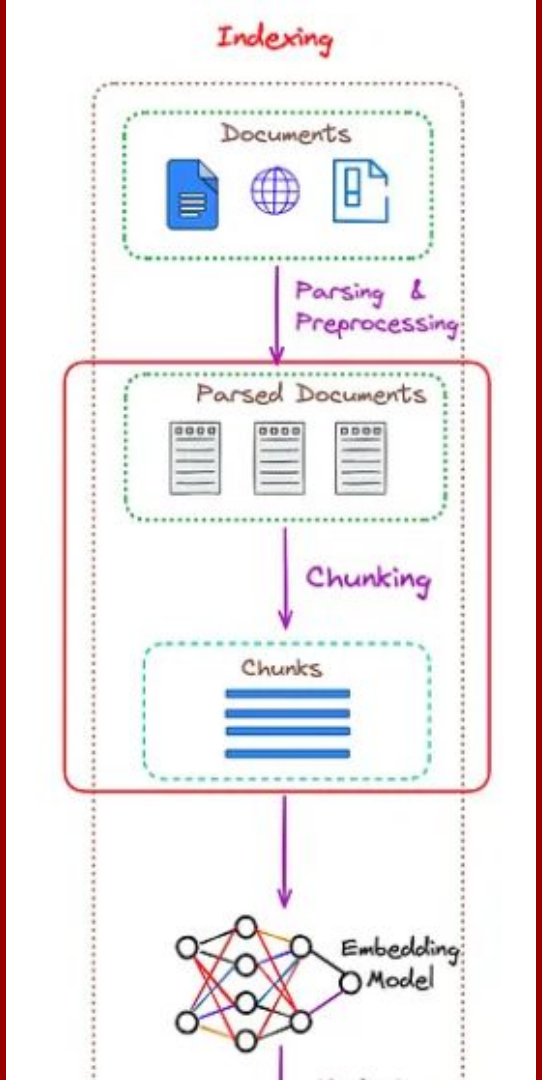
Keyword Search

vs.

Semantic Search

vs.

RAG


Indexing
Documents
Parsing & Preprocessing
Parsed Documents
Chunking
Chunks
Embedding Model

CHUNKING

In the context of building LLM-related applications, chunking is the process of breaking down large pieces of text into smaller segments.

It’s an essential technique that helps optimize the relevance of the content we get back from a vector database once we use the LLM to embed content.

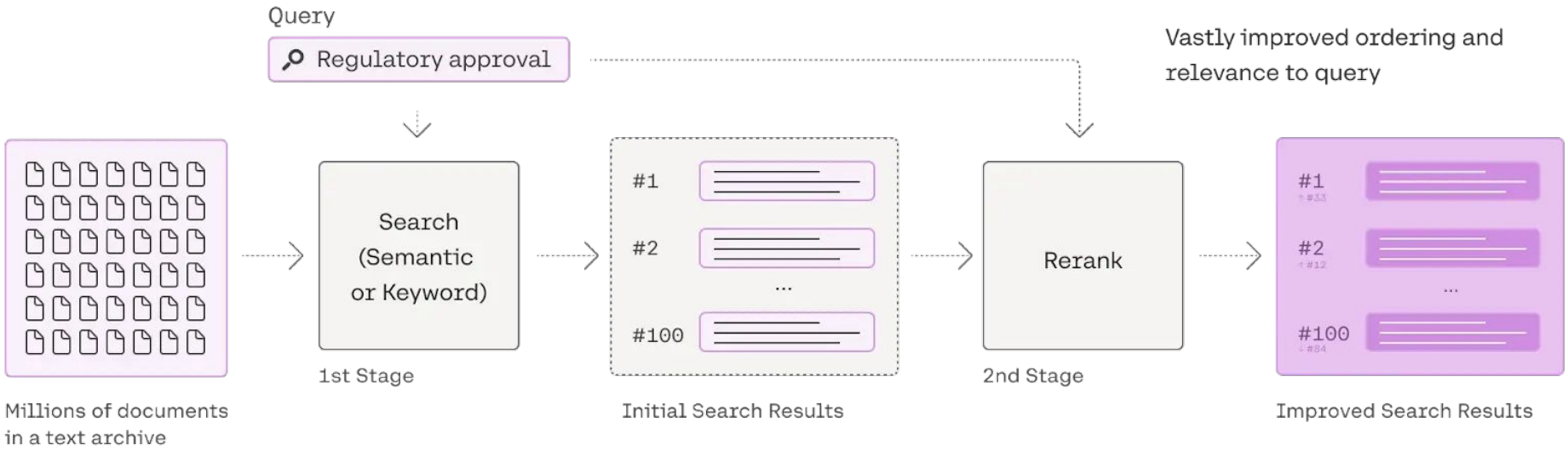
SEMANTIC CHUNKING

- Break up the document into sentences.
- Create sentence groups: for each sentence, create a group containing some sentences before and after the given sentence.
- Generate embeddings for each sentence group.
- Compare distances between each group sequentially: as long as the topic or theme is the same, the distance between the sentence group embedding for a given sentence and the sentence group preceding it will be low. On the other hand, higher semantic distance indicates that the theme or topic has changed.

Generally, a combination of different techniques is used and so the same sentence can appear in multiple chunks.

This leads to larger size of the database and more computations while retrieval, but improves RAG performance.

RE-RANKING


Query
Regulatory approval
Vastly improved ordering and relevance to query
Search (Semantic or Keyword)
#1
#2
...
#100
Rerank
#1
#2
...
#100
1st Stage
2nd Stage
Millions of documents in a text archive
Initial Search Results
Improved Search Results

Rerankers and Two-Stage Retrieval

For vector search to work, we need vectors. These vectors are essentially compressions of the "meaning" behind some text into (typically) 768 or 1536-dimensional vectors. There is some information loss because we're compressing this information into a single vector.

Because of this information loss, we often see that the top three (for example) vector search documents will miss relevant information. Unfortunately, the retrieval may return relevant information below our **top_k** cutoff.

What do we do if relevant information at a lower position would help our LLM formulate a better response?

The easiest approach is to increase the number of documents we're returning (increase **top_k**) and pass them all to the LLM.

Unfortunately, we cannot return everything.

LLMs have limits on how much text we can pass to them — we call this limit the *context window*.

Some LLMs have huge context windows, like Anthropic's Claude, with a context window of 100K tokens.

With that, we could fit many tens of pages of text — so could we return many documents (not quite all) and "stuff" the context window to improve recall?

Again, no. We cannot use context stuffing because this reduces the LLM's *recall* performance.

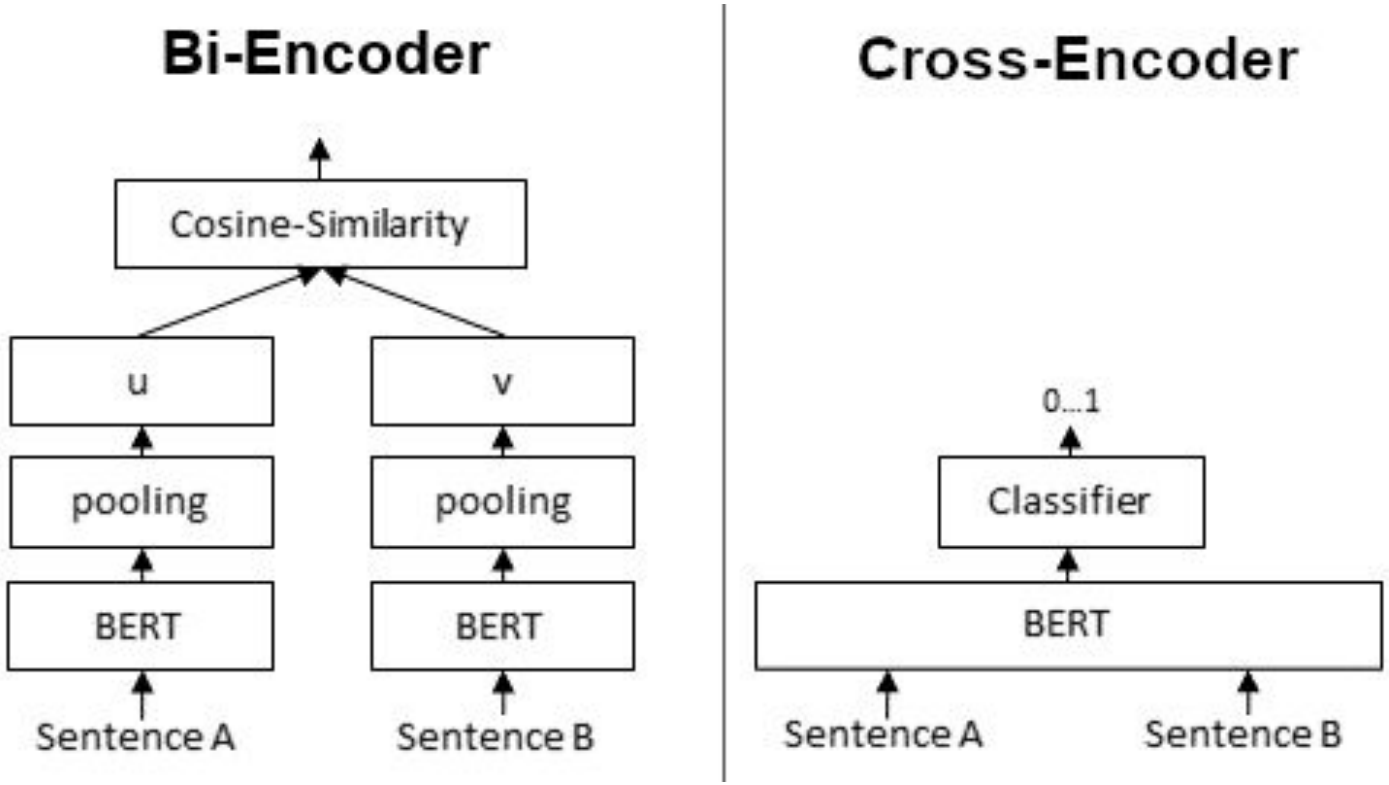
The solution to this issue is to maximize retrieval recall by retrieving plenty of documents and then maximize LLM recall by *minimizing* the number of documents that make it to the LLM.

To do that, we reorder retrieved documents and keep just the most relevant for our LLM

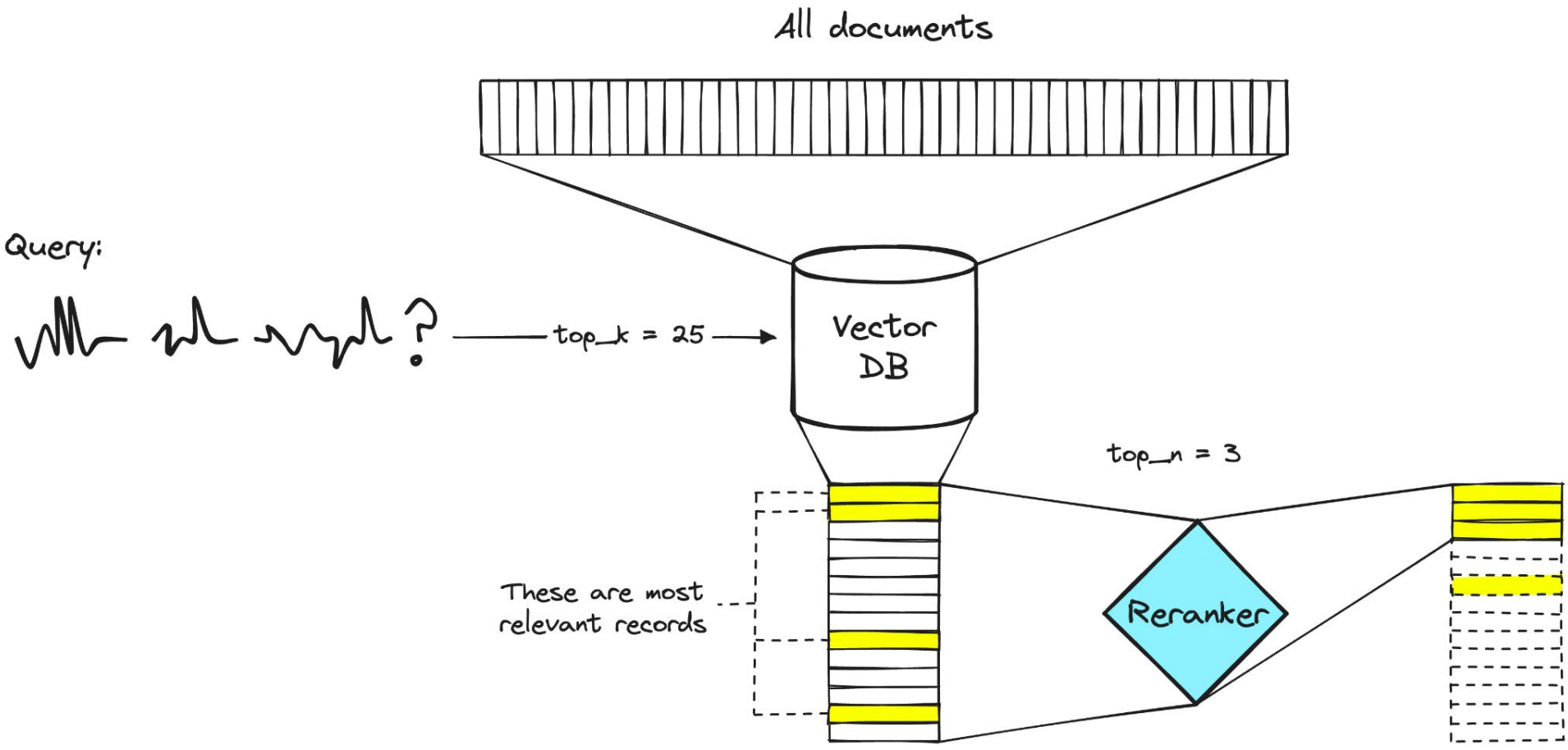
— to do that, we use *reranking*.

A reranking model
— also known as a **cross-encoder**
— is a type of model that,
given a query and document pair,
will output a similarity score.

We use this score to reorder the documents
by relevance to our query.



For retrieval, is it better to use bi-encoders or cross-encoders?
Is one of them going to be much slower than the other?


All documents
Query:
?
top_k = 25
Vector DB
top_n = 3
These are most relevant records
Reranker

Search engineers have used rerankers in two-stage retrieval systems for *a long time*.

In these two-stage systems, a first-stage model (an embedding model/retriever) retrieves a set of relevant documents from a larger dataset.

Then, a second-stage model (the reranker) is used to rerank those documents retrieved by the first-stage model.

We use two stages because retrieving a small set of documents from a large dataset is much faster than reranking a large set of documents — we'll discuss why this is the case soon — but TL;DR,

rerankers are slow, and retrievers are *fast*.

If a reranker is so much slower,
why bother using them?

Model	Type	Performance	Cost	Example
Cross encoder	Open source	Great	Medium	BGE, sentence, transformers, Mixedbread
Multi-vector	Open source	Good	Low	ColBERT
LLM	Open source	Great	High	RankZephyr, RankT5
LLM API	Private	Best	Very High	GPT, Claude
Rerank API	Private	Great	Medium	Cohere, Mixedbread, Jina

Cross-Encoders

Cross-encoder models redefine the conventional approach by employing a classification mechanism for pairs of data.

The model takes a pair of data, such as two sentences, as input, and produces an output value between 0 and 1, indicating the similarity between the two items.

This departure from vector embeddings allows for a more nuanced understanding of the relationships between data points.

It's important to note that cross-encoders require a pair of "items" for every input

Ensemble Models

One approach to re-ranking involves using an ensemble of multiple language models or ranking algorithms.

By combining the strengths of different models, ensemble-based re-ranking can provide more accurate and diverse results compared to relying on a single model.

Contextual Re-Ranking

This technique involves incorporating contextual information, such as user preferences or interaction history, into the re-ranking process.

By personalizing the ranking criteria based on the user’s context, the system can deliver more relevant and engaging responses.

Query Expansion

Query expansion is a re-ranking technique that involves modifying or expanding the user's initial query to better capture their intent.

This can be achieved by adding related terms, synonyms, or even paraphrasing the query.

By broadening the scope of the search, query expansion helps retrieve more relevant and diverse candidates for re-ranking.

Feature-based Re-Ranking

In this approach, the system assigns scores to the top-k candidates based on a set of predefined features, such as term frequency, document length, or entity overlap.

These scores are then used to re-rank the candidates, ensuring that the most relevant and informative responses are selected for the generation step.

Re-Ranking is an open problem
and lot of work is needed to improve its accuracy

Install Dependencies

```
!pip install --quiet langchain_experimental langchain_openai
```

Load Example Data

```
# This is a long document we can split up.
with open("../../state_of_the_union.txt") as f:
    state_of_the_union = f.read()
```

Is RAG a fool proof solution to augment LLMs with new information?

What do you think are the challenges?

Non-Graded Task

Build a RAG pipeline (with re-ranking)
using books from any domain

Explore LangChain and LlamaIndex in detail,
and use them for your RAG pipeline

Compare with classical SEIR algorithms