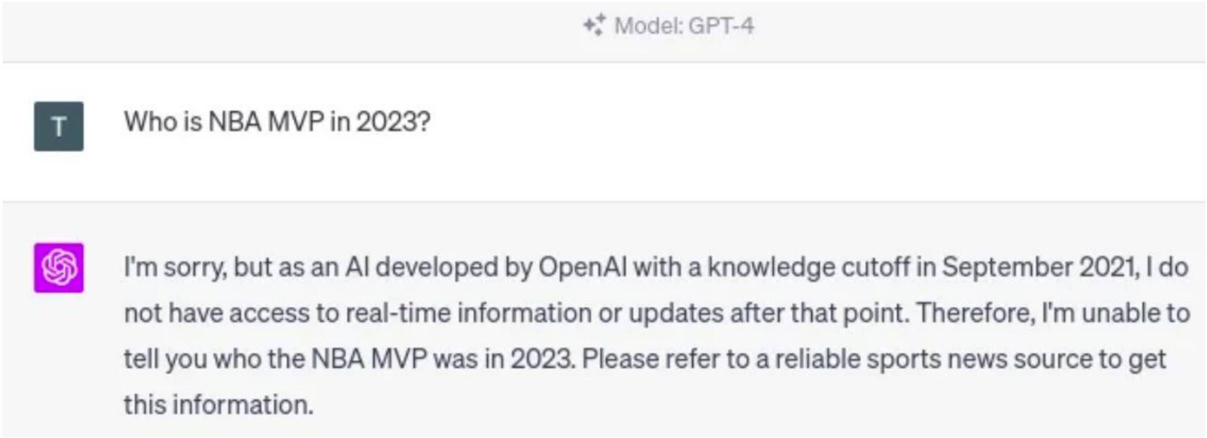
Retrieval Augmented LLMs

A.k.a Retrieval Augmented Generation (RAG)

Motivation: Limitations of pretrained LLMs

- Knowledge cutoff:
  - Lack up-to-date knowledge
  - Lack of knowledge outside the training dataset
  - Lack of specific knowledge (e.g., private or confidential data)
- Hallucination
  - Inaccurate, invalid information

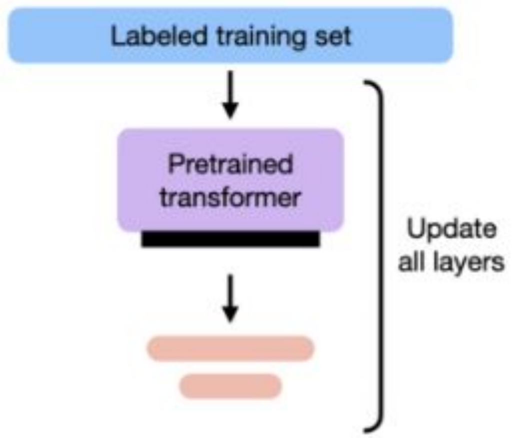


Source : Knowledge Graphs & LLMs: Fine-Tuning Vs. Retrieval-Augmented Generation

Motivation: Available solutions

- **Knowledge cutoff, Hallucination -> Supervised Finetuning**:
  - Time consuming
  - Data generation (prompt-completion pairs) is time consuming and expensive
  - Dynamic data sources (i.e., changes quickly)
- **Hallucination -> RLHF**
  - Resource intensive
  - Human and manual input and supervision
  - Works only during training and not inference

Supervised finetune
(from Understanding Parameter-Efficient Finetuning of Large Language Models: From Prefix Tuning to LLaMA-Adapters)



Context window

- Incorporate knowledge into LLMs using context window:

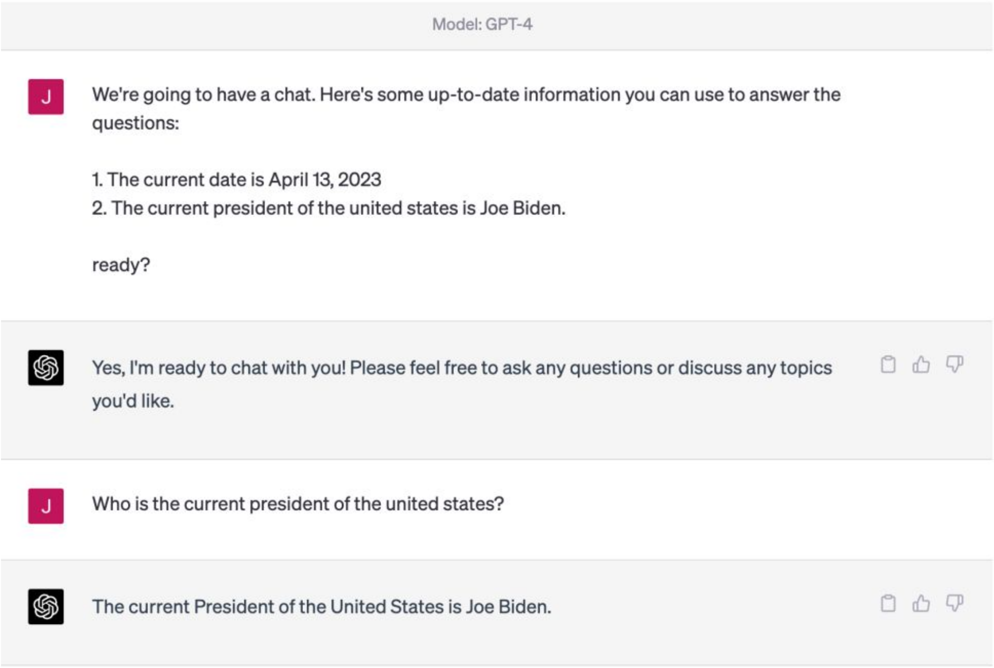
  (+) No need to finetune

  (-) the context window is limited

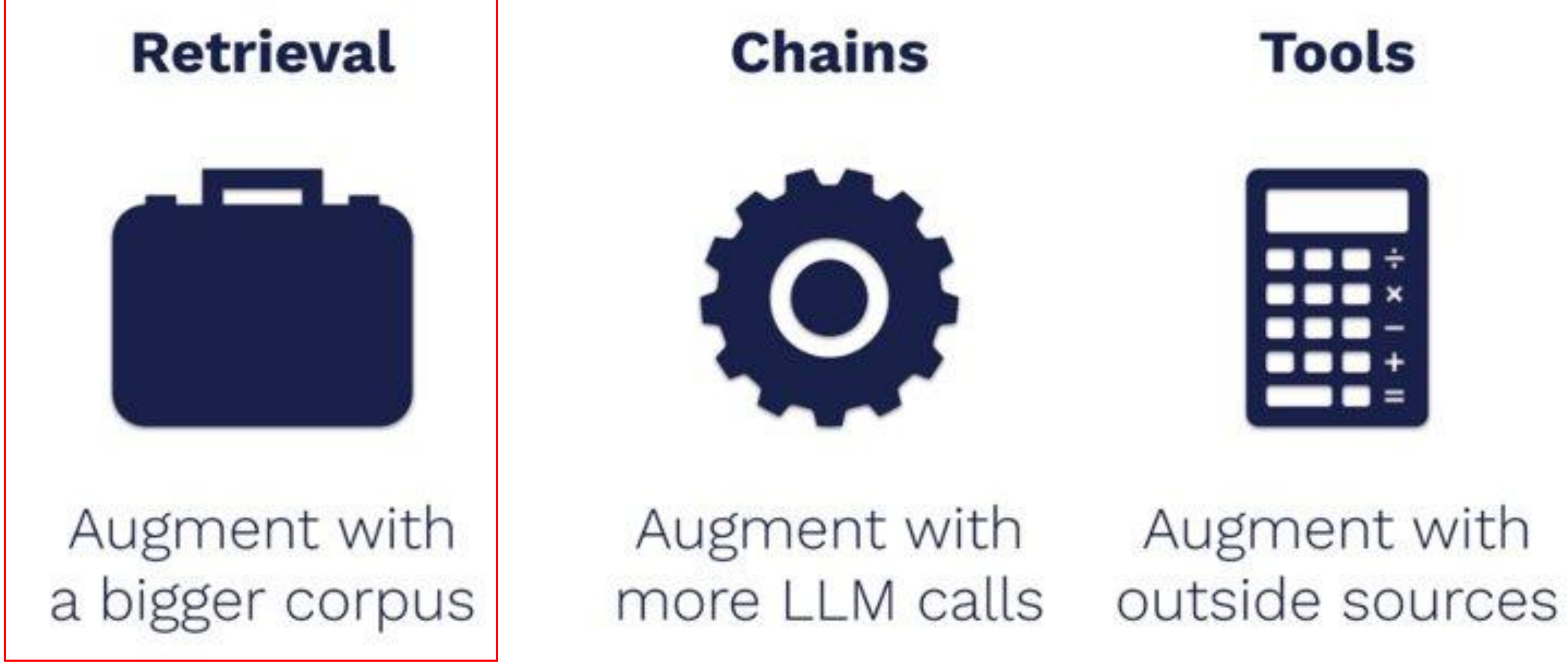
  (-) cost increases as the size of the context increases

  -> How to make the most of the limited context window

Feeding up-to-date information to the LLM
source: <u>The full stack LLM bootcamp</u>



Augmented LLMs

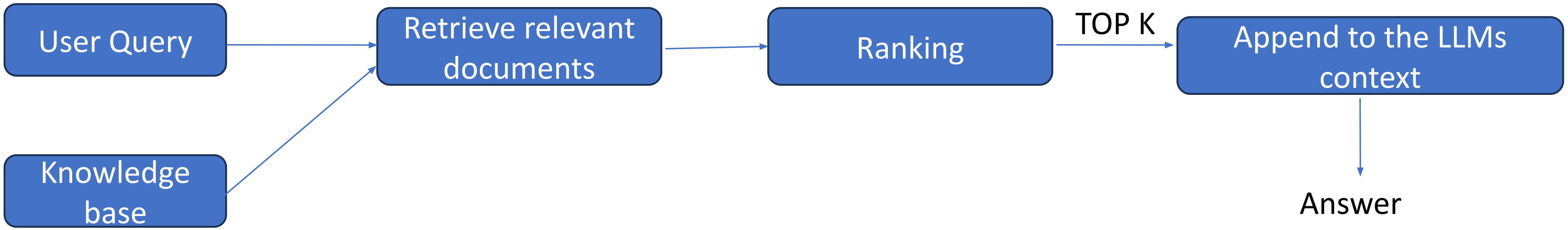


source: The full stack LLM bootcamp

Retrieval augmentation

- Why?
  - The process of building the context for LLMs == Information retrieval
  - **Search** for right data to put in the context window

- Retrieval Pipeline: Given a user query, search for all the relevent objects (e.g., documents) and rank them



Retrievers

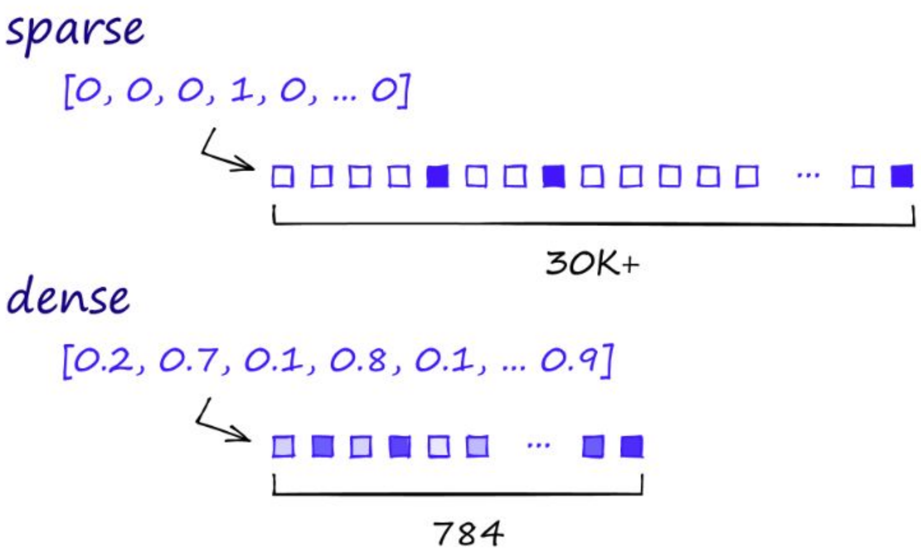
- **Sparse** : sparse bag-of-words representations of the documents and the queries

-> checking for precise term syntax overlap

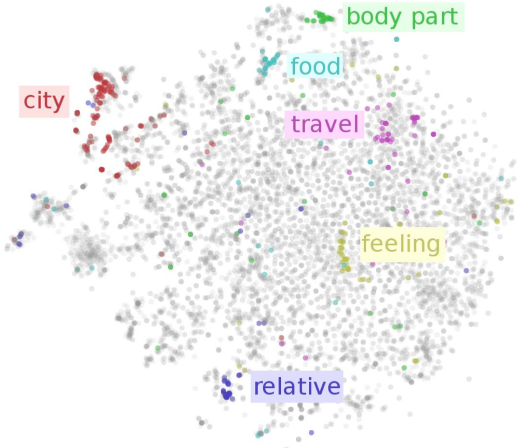
-> Doesn't capture semantic information, correlation information

- **Dense**: dense query and document vectors obtained from neural networks (a.k.a embeddings)

-> computing the semantic similarity of related topics



source: Pinecone-dense-vectors



source: on word embeddings

Which embedding model?

- Popular but not free:

OpenAI text-embedding-ada-002

- Open source: see MTEB leaderboard on HF

- For better retrieval quality, training your own embedding model is necessary

Overall | Bitext Mining | Classification | Clustering | Pair Classification | Retrieval | Reranking | STS | Summarization

Overall MTEB English leaderboard

- **Metric:** Various, refer to task tabs
- **Languages:** English, refer to task tabs for others

Rank	Model	Embedding Dimensions	Average (56 datasets)	Classification Average (12 datasets)	Clustering Average (11 datasets)	Pair Classification Average (3 datasets)	Reranking Average (4 datasets)	Retrieval Average (15 datasets)	STS Average (10 datasets)	Summarization Average (1 dataset)
1	instructor-xl	768	61.79	73.12	44.74	86.62	57.29	49.26	83.06	32.32
2	instructor-large	768	61.59	73.86	45.29	85.89	57.54	47.57	83.15	31.84
3	e5-large	1024	61.42	73.14	43.33	85.94	56.53	49.99	82.06	30.97
4	text-embedding-ada-002	1536	60.99	70.93	45.9	84.89	56.32	49.25	80.97	30.8
5	e5-base	768	60.44	72.63	42.11	85.09	55.7	48.75	80.96	31.01
6	instructor-base	768	59.54	72.36	41.9	83.51	56.2	45.12	82.29	29.85
7	sentence-t5-xxl	768	59.51	73.42	43.72	85.06	56.42	42.24	82.63	30.08

Massive Text Embedding Benchmark (MTEB) Leaderboard.

How to chunk data?

- Like LLMs, embedding models have limited context

  -> Split the documents to multiple chunks

- Things to consider:
  - Natural structure, semantic content

- Tools: Langchain, NLTK, LLamaIndex

- Ideas:
  - Perplexity decreases at a semantic boundary
  - Use distance between embedding (higher -> chunk)
  - Summarize -> embed

Pinecone-chuncking strategy

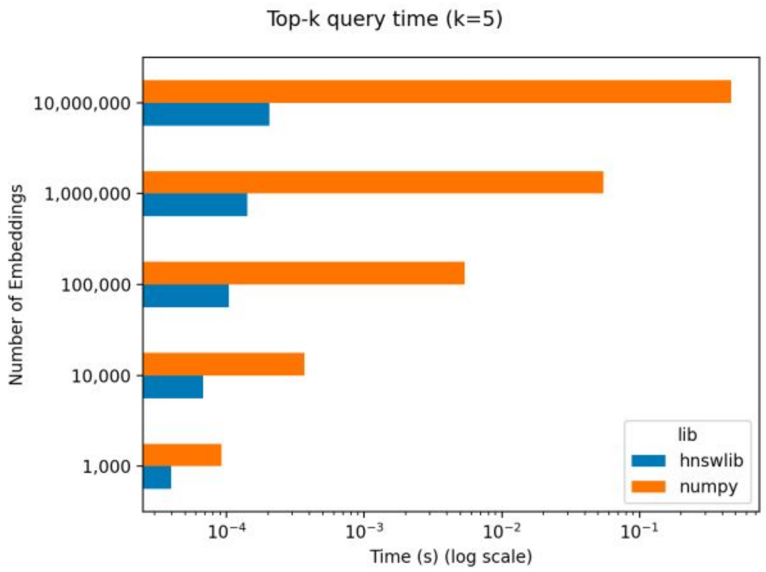
Embedding Retrieval: KNN & Flat index

- Embed your corpus
- Store embeddings as an array
- Embed the query, compute dot product with the array

```
# vec -> 1D numpy array of shape D
# mat -> 2D numpy array of shape N x D
# k -> number of most similar entities to find.
similarities = vec @ mat.T
partitioned_indices = np.argpartition(-similarities, kth=k)[:k]
top_k_indices = partitioned_indices[np.argsort(-similarities[partitioned_indices])]
```

- Works for < 100K vectors (difference in speed not noticeable)

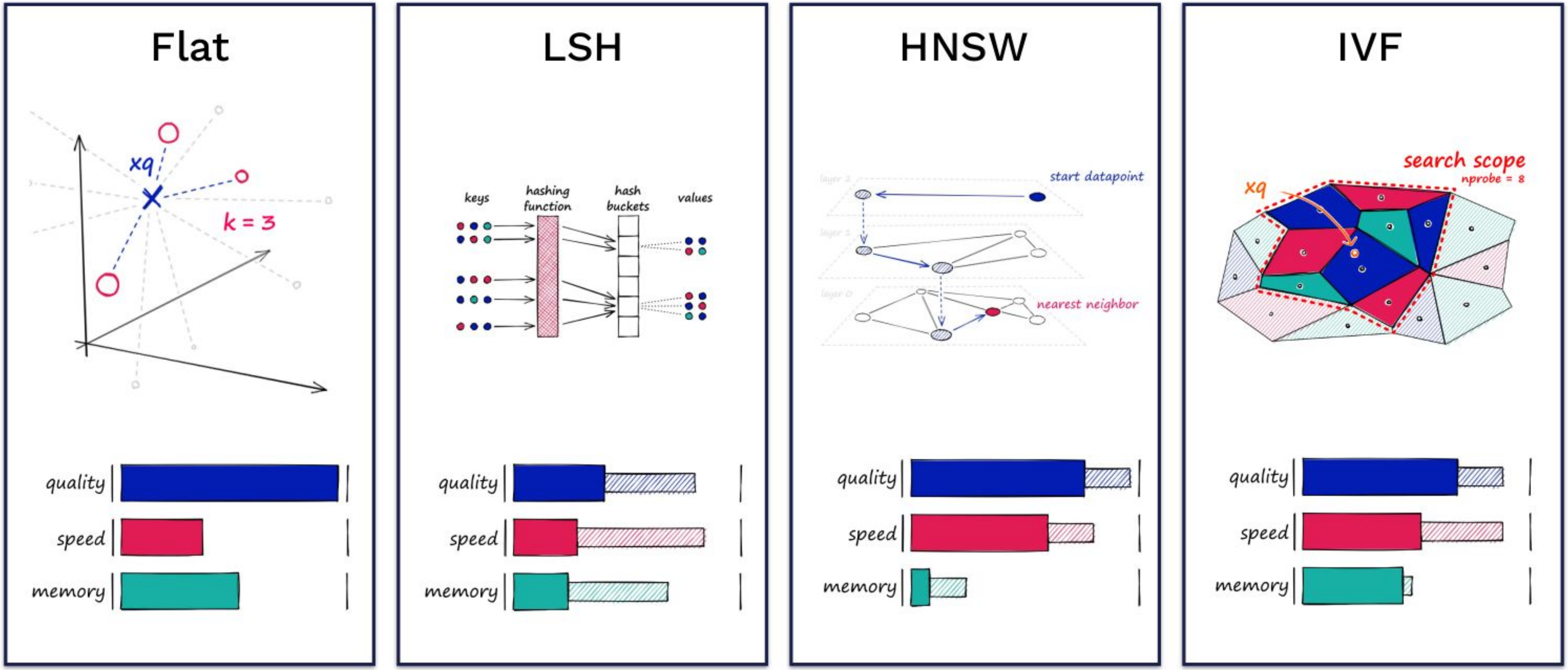
- Not scalable (of course)



source: Do you actually need a vector database?

Approximate nearest neighbor (ANN) index

- **Embedding Indexes** : data structure enabling efficient and fast retrieval
- **Approximate nearest neighbor (ANN) index**
- The indexing process :
  - partitioning the vector space
  - creating data structures to enable efficient traversal and search operations
  - storing the necessary metadata for each indexed vector.


Flat
xq
k = 3
quality
speed
memory
LSH
keys
hashing function
hash buckets
values
quality
speed
memory
HNSW
start datapoint
nearest neighbor
quality
speed
memory
IVF
search scope
nprobe = 8
xq
quality
speed
memory

Vector databases

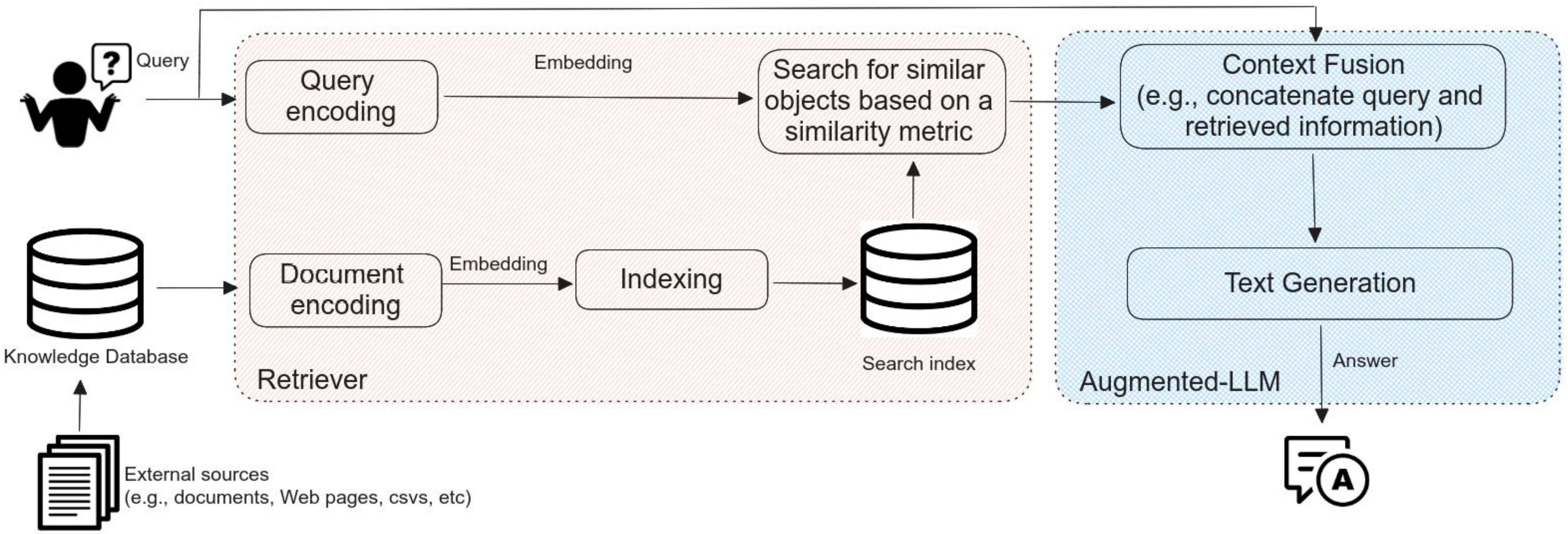
- ANN indexes are just data structure, they do not offer:
  - Hosting
  - Storing data/metadata alonside vectors
  - Combining sparse + dense retrieval
  - Managing embedding functions themselves
  - Scale
  - …...

Vector database

Tool	Prominent users	DB features	Embedding mgmt	Sql-like Filtering	Full text search	It's for...
Chroma	N/A	✓	✓	✓	✗	Betting on the most "AI-native" tool in the category
Milvus	ebay, Walmart	✓	✗	✓	✗	Scale & enterprise
Pinecone	shopify, GONG	✓	✗	✓	✗	Fastest to get started
vespa	yahoo!, Spotify R&D	✓	✓	✓	✓	Battle-tested; most powerful
weaviate	N/A	✓	✓	✓	✓	Embedding mgmt and flexible GraphQL-like query interface

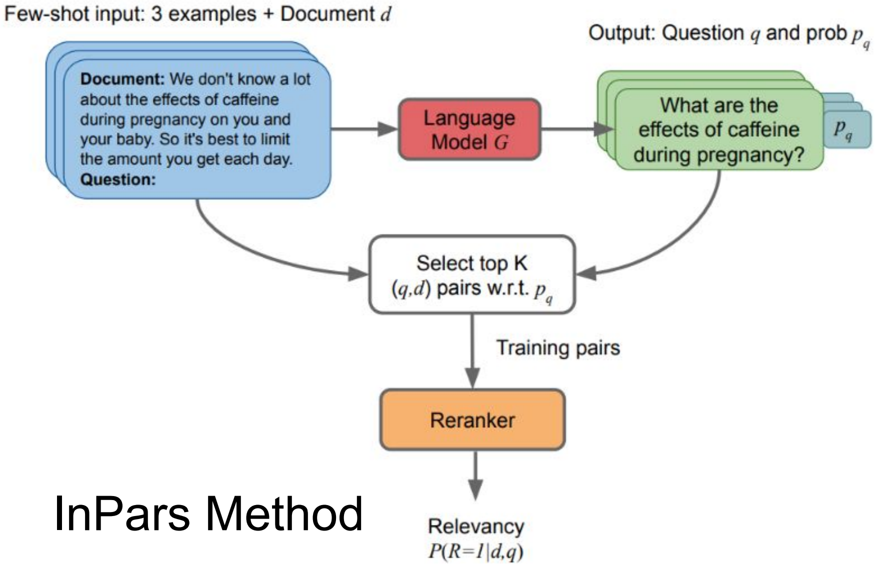
source: The full stack LLM bootcamp

Retrieval Augmented LLMs- Full pipeline



Retrieval Relevance : NN way

- Re-ranking: further re-fined the retrieved documents
  - **Cross-encoder**: takes a query and a document vector as the input and calculates the relevance scores as the maximum inner product over it.
  - **Bi-encoder**: takes a query and a document vector as the input and calculates the relevance scores as the maximum inner product over it.
  - **Cascaded pipeline**: cheap algorithm for retrieval (e.g, ElasticSearch, BM25, bi-encoder), more complex model for re-ranking (cross-encoder)
  - **Using LLMs** : to generate synthetic data in few-shot manner and then finetune a re-ranker model
  - Read more here



InPars Method

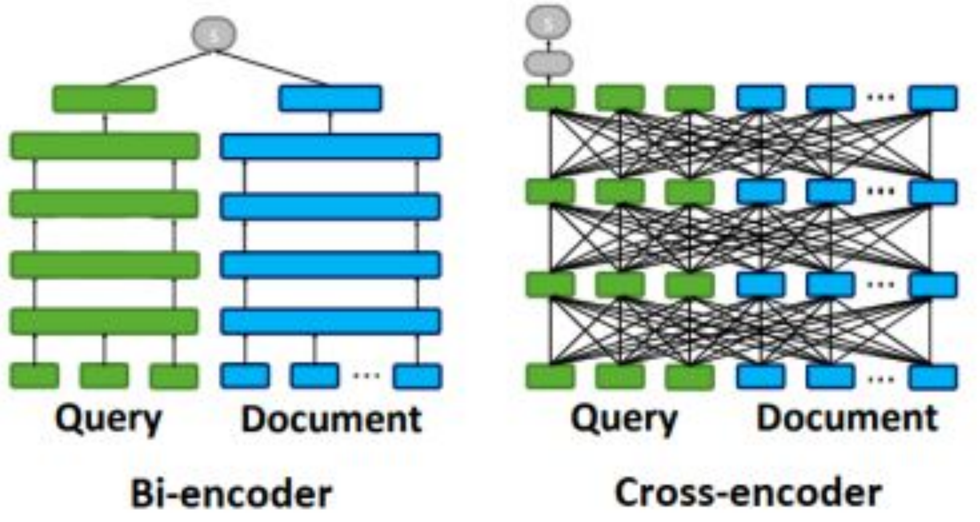


Figure adapted from ColBERT paper

Retrieval Relevance : Non-NN ways

- **Maximal marginal relevance** (MMR): selects diverse and representative documents from a larger set of search results.

- FIltering: based on metadata or keywords

Thank you