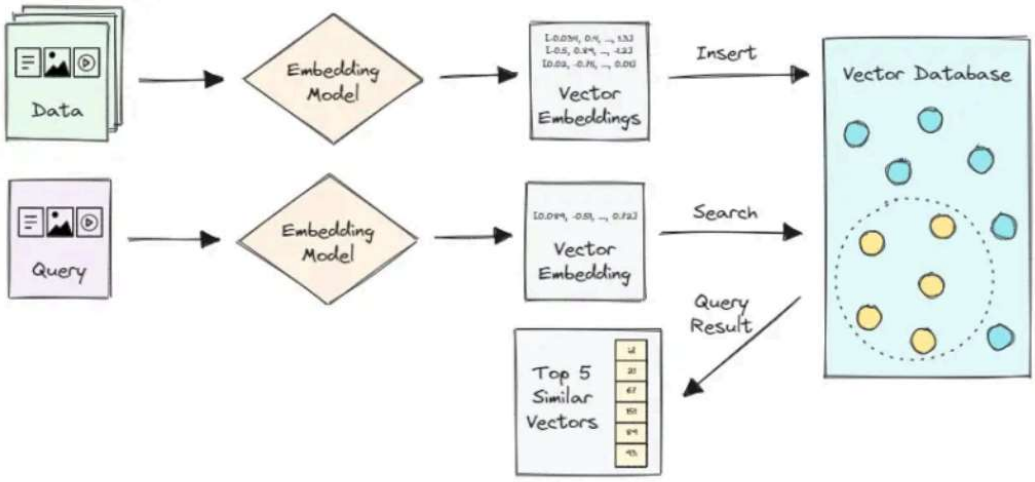
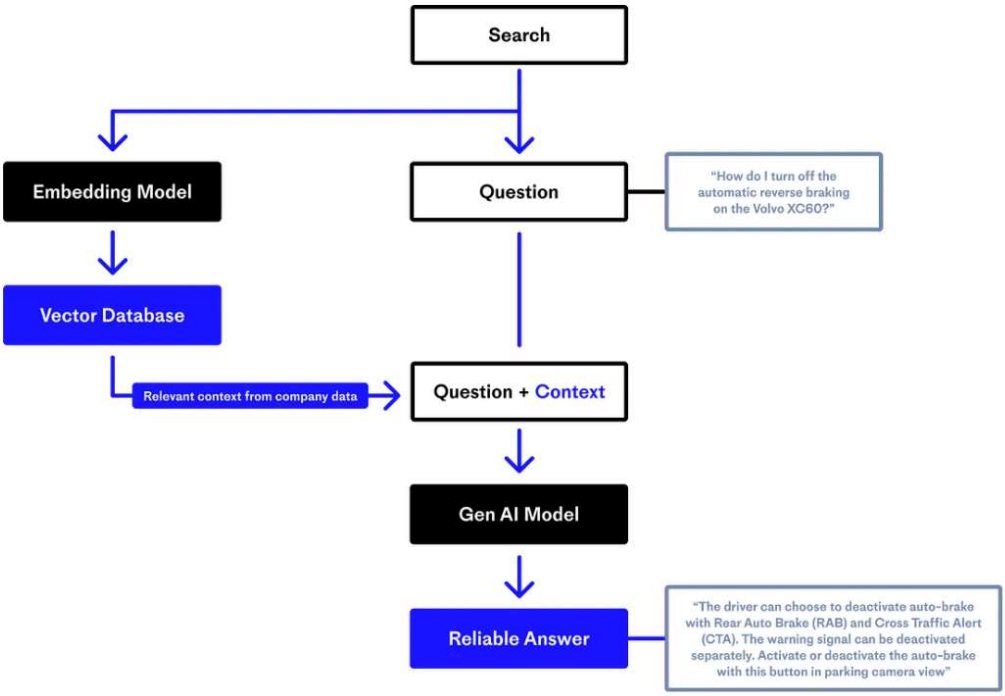
PART 5: SEARCH AND RETRIEVAL



Vector DB

When a user queries, their input is tokenized and embedded using the same model used for raw data. Relevant segments from the knowledge base are then selected based on their similarity to the user's query.

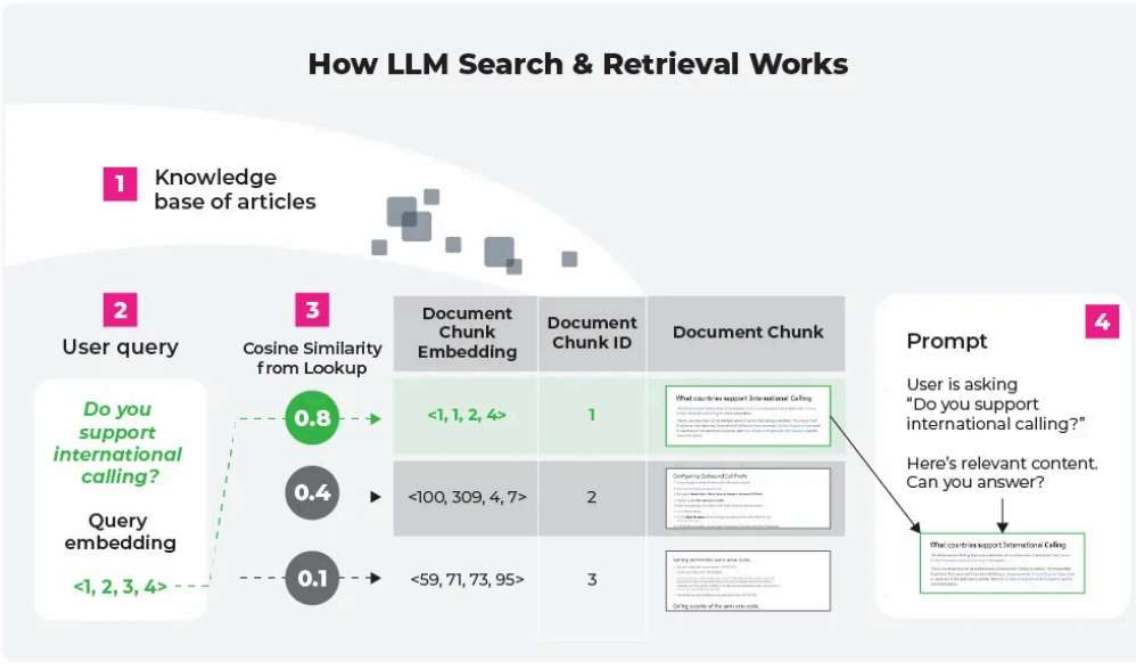


Retrieval Augmented Generation reduces the likelihood of hallucinations by providing domain-specific information through an LLM's context window.

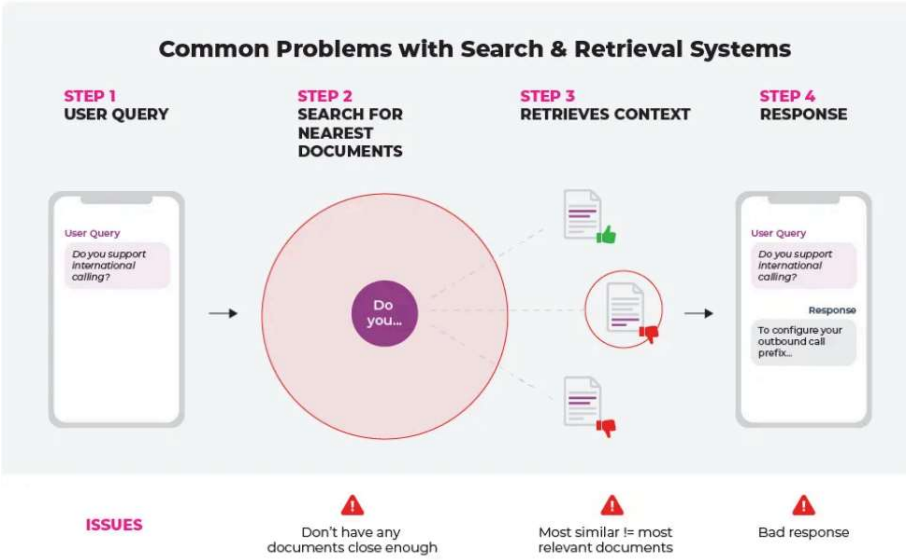
Lets say u want to build a customer care chatbot for your company.
You have tons of documentation for it.

This data is broken down into pieces through a process called "chunking." After the data is broken down, each chunk is assigned a unique identifier and embedded into a high-dimensional space within a vector database.

When a customer's question comes in, the LLM uses a retrieval algorithm to quickly identify and fetch the most relevant chunks from the vector database. This retrieval is based on the semantic similarity between the query and the chunks, not just keyword matching.



- Efficiency: Reduces time and computational resources by interacting only with relevant chunks.
- Real-Time Updates: Ensures the chatbot provides up-to-date information as documentation evolves.
- Precision: Improves customer satisfaction by offering precise and contextually appropriate responses.



How can we prevent these issues?

1. Query density:
   Measures how well user queries are covered by the vector store. A significant drift in query density indicates that the vector store may not be fully capturing user queries, leading to fewer closely related chunks.
   Regular monitoring of query density helps identify gaps, allowing us to enhance the vector store with more relevant chunks or refine existing ones, thus improving data retrieval.
2. Ranking metrics:
   Evaluate a search system's effectiveness in selecting relevant chunks. A decline in these metrics signals that the system needs improvement in distinguishing relevant from irrelevant chunks.
3. User Feedback

Optimizing Search & Retrieval

To optimize search and retrieval processes in your LLM-powered application, we need to focus on both the building and post-production phases.

1. Building Phase:
   a. Chunking Strategy:
      → Refine how information is broken down and processed for better performance

   b. Retrieval Performance
      → Evaluate and improve the system's ability to retrieve information, using tools like context ranking or HYDE.

2. Post-Production:
   a. Expand Knowledge Base:
      → Continuously add documentation to enhance response quality.

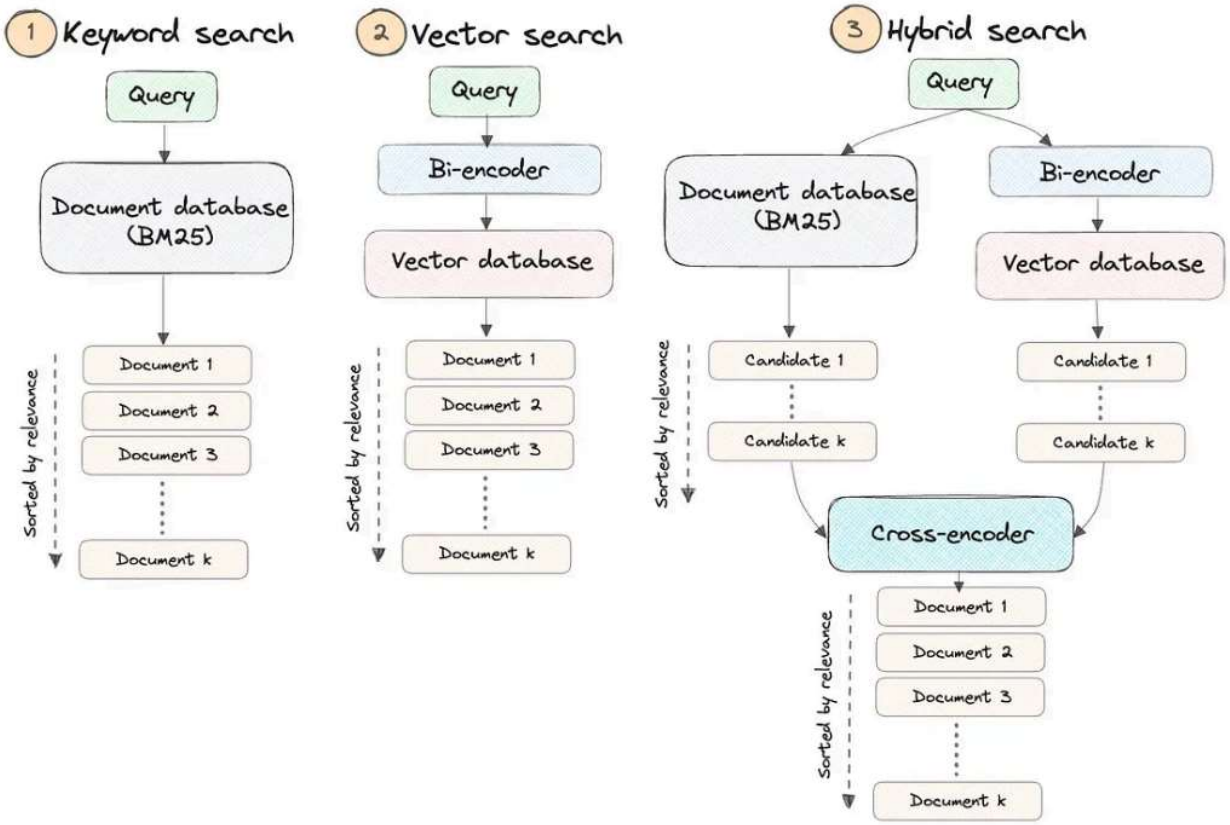
   b. Refine Chunking:
      → Further adjust how information is chunked for optimal processing

   c. Enhance Context Understanding
      → Implement a context evaluation step to ensure relevant context is incorporated into responses.

Types of Search

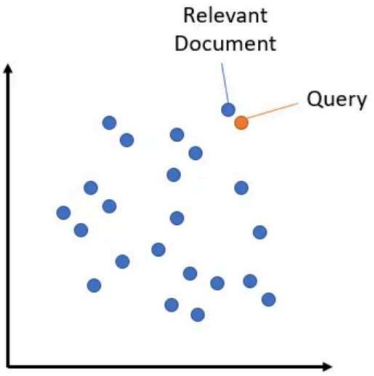
a. Keyword search
   ⇒ Finds exact matches, ideal if the user knows what they want. No vector database needed.
b. Vector search
   ⇒ Useful when users are uncertain, needs a vector database for relevance.
c. Hybrid search

⇒ Blends keyword and vector approaches, enhancing relevance with cross-encoder models. Requires both types of databases.



Semantic Search:

Semantic search involves embedding all entries in a corpus into a vector space. During search, queries are also embedded into this space to find entries with high semantic similarity.



1. Symmetric Semantic Search
    a. Query and corpus entries are similar in length and content amount.
    b. Example: Query "How to learn Python online?" matches corpus entry "How to learn Python on the web?"
    c. Query and corpus entries can potentially be flipped.

2. Asymmetric Semantic Search

    a. Query is typically short (e.g., a question or keywords).
    b. Corpus entries are longer paragraphs or documents providing detailed answers.
    c. Example: Query "What is Python?" matches corpus entry describing Python's features and philosophy.
    d. Flipping the query and corpus entries usually doesn't make sense in asymmetric tasks.
- Suitable models for **asymmetric semantic search**: Pre-Trained MS MARCO Model
- Suitable models for **symmetric semantic search**: Pre-Trained Sentence Embedding Models

Retrieval Algorithms

Similarity Search (Vanilla Search) & Maximum Marginal Relevance(MMR)

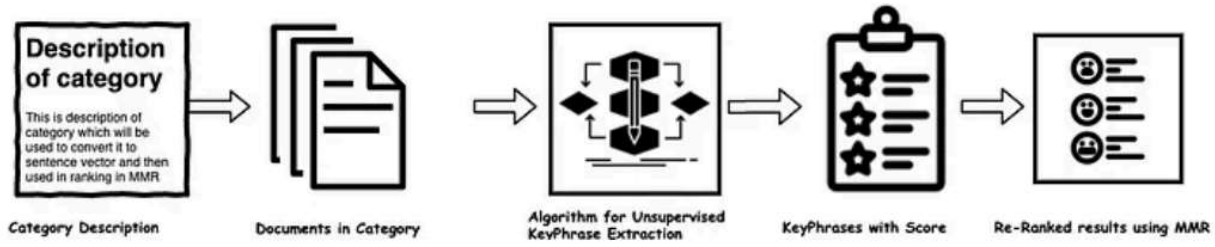
⇒ Traditional methods like cosine similarity prioritize retrieving documents most similar to the query.
⇒ Maximum Marginal Relevance (MMR) enhances diversity by considering both relevance to the query and dissimilarity to already retrieved documents, ensuring a balanced and varied selection of documents.

In case of **unsupervised learning**, let's say our final keyPhrases are ranked like `Good Product, Great Product, Nice Product, Excellent Product, Easy Install, Nice UI, Light weight etc.` But there is an issue with this approach, all the phrases like `good product, nice product, excellent product` are similar and define the same property of the product and are ranked higher.

To maximize space efficiency and ensure diverse information from keyphrases about documents:

1. Cosine Similarity Removal: Eliminate redundant phrases by comparing their semantic similarity using cosine similarity.

2. MMR Re-ranking: Employ Maximal Marginal Relevance (MMR) to re-rank keyphrases. This method prioritizes phrases that add new information, reducing redundancy and enhancing diversity.

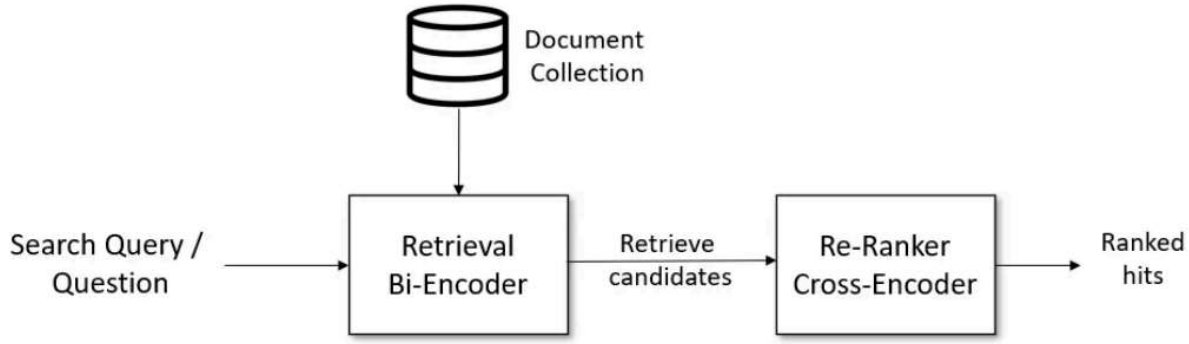


MMR

Retrieve and Re-Rank:

For complex search tasks, for example, for question-answering retrieval, the search can significantly be improved by using **Retrieve & Re-Rank**.

Retrieve & Re-Rank Pipeline



By SBERT

Semantic Search: Bi-Encoder

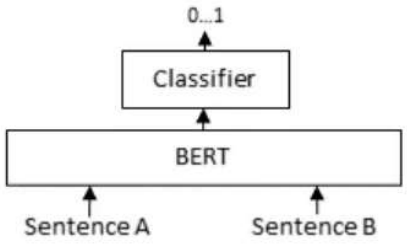
In the retrieval stage, the goal is to narrow down a large set of potential documents (or hits) that are likely relevant to the user's query. This is typically done using a bi-encoder approach

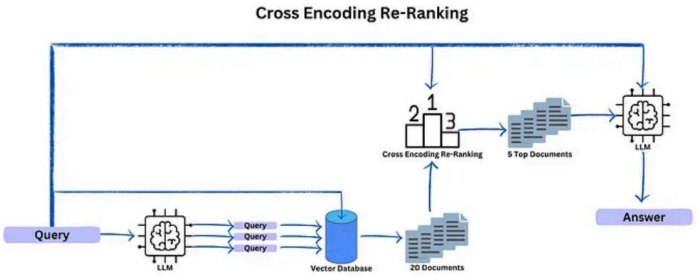
a. Uses a dense representation of both queries and documents.
b. Encodes each query and each document into fixed-dimensional vectors in a semantic space using deep learning models like BERT (Bidirectional Encoder Representations from Transformers) or its variants.
c. Retrieves a set of documents that have high cosine similarity scores with the query vector.

Re-Ranking Stage (Cross-Encoder)

After retrieving a preliminary set of documents (e.g., 100 candidates) from the bi-encoder, the re-ranking stage aims to refine this list to prioritize the most relevant documents using a cross-encoder:

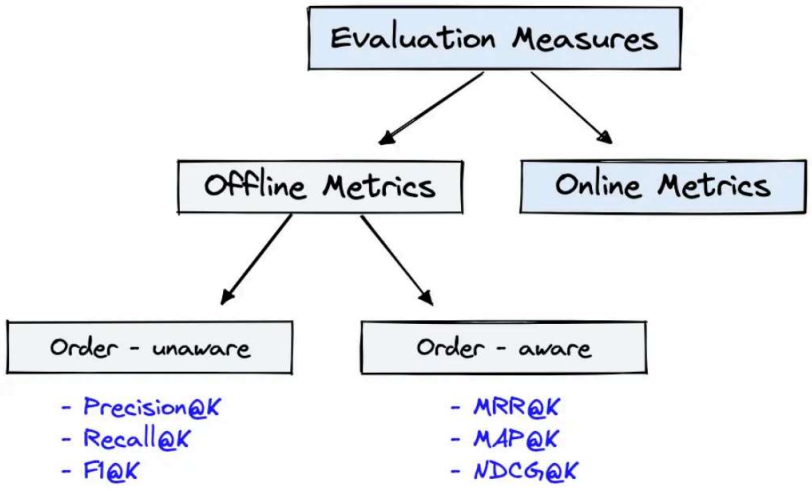
a. Takes in a query and a candidate document as input simultaneously.
b. Computes a single score (between 0 and 1) that reflects the overall relevance of the document to the query.
c. Unlike bi-encoders that compare query and document embeddings separately, cross-encoders perform attention across both the query and document simultaneously, capturing more nuanced interactions.
d. This approach involves a transformer network (similar to BERT) that is fine-tuned on a question-answering or information retrieval task.





Evaluation of Information Retrieval

- Online Metrics:
  a. Captured during actual usage of the IR system.
  b. Measure user interactions like click-through rates (CTR) on recommendations or ads.
  c. Reflect how users engage with the system in real-time.
  d.
- Offline Metrics:
  a. Measured in an isolated, controlled environment pre-deployment.
  b. Assess the relevance and accuracy of retrieved results.
  c. Provide insights into system performance before it goes live.



1. Recall@K

$$Recall@K = \frac{truePositives}{truePositives + falseNegatives} = \frac{p\hat{p}}{p\hat{p} + p\hat{n}}$$

The *K* in this and all other offline metrics refers to the number of items returned by the IR system.
Recall@K is straightforward but has limitations. Perfect scores can be deceptive with large K values, and it ignores result order. Returning relevant items early doesn't improve scores, undermining its utility.

2. Mean Reciprocal Rank (MRR)

The Mean Reciprocal Rank (MRR) is an order-sensitive metric where finding a relevant result sooner ranks higher. Unlike recall@K, relevance at rank one is more favorable than at rank four. It's calculated across multiple queries as:

$$RR = \frac{1}{Q}\sum_{q=1}^{Q}\frac{1}{rank_q}$$

Q is the number of queries, *q* is a specific query, and *rank-q* is the rank of the first *actual relevant* result for query *q*.

MRR's advantage lies in its focus on the rank of the first relevant result, crucial for applications like chatbots. However, it overlooks subsequent relevant items, making it unsuitable for multi-item recommendations or search engines. Its interpretability is also lower compared to simpler metrics like recall@K, though it remains more understandable than many others.

3. Mean Average Precision (MAP)

→ *order-aware* metric

$$Precision@K = \frac{truePositives}{truePositives + falsePositives} = \frac{p\hat{p}}{p\hat{p} + n\hat{p}}$$

$$AP@K = \frac{\sum_{k=1}^{K}(Precision@k * rel_k)}{number\ of\ relevant\ results}$$

Each of these individual *AP@Kq* calculations produces a single **A**verage **P**recision@K score for each query q.

To get the **M**ean **A**verage **P**recision@K (*MAP@K*) score for all queries, we simply divide by the number of queries *Q*.

4. Normalized Discounted Cumulative Gain (NDCG@K)

Starting with **C**umulative **G**ain (*CG@K*) calculated like so:

$$CG@K = \sum_{k=1}^{K} rel_k$$

The *rel-K* variable is different this time. It is a range of relevance ranks where *0* is the least relevant, and some higher value is the most relevant.

CG@K is *not* order-aware