My Handwritten RAG Notes

By Piyush Pant

A Summary of RAG Course by DeepLearning.AI

Acknowledgments & Credits

A special thank you to DeepLearning.AI and instructor Zain Hasan for creating and teaching this excellent course on Retrieval Augmented Generation. These notes represent my personal summaries, diagrams, and takeaways from their educational curriculum. All credit for the original course structure belongs entirely to them.

Community Usage & Non-Commercial Policy

These notes are shared openly as a free resource to support the AI developer community and anyone who is just starting their AI Journey!

- Free to Share: You are welcome to download, use, and share these notes with study groups or colleagues.
- Strictly Not For Sale: This document is a free community asset. It must not be sold, monetized, or hosted behind any commercial paywalls under any circumstances.

Author's Note

Because these notes were written by hand in real-time while watching the lectures, please keep the following in mind:

- Handwriting: I apologize in advance if my handwriting is messy or difficult to read in certain sections! I did my best to keep it as clear as possible.
- Potential Errors: These are personal study materials, meaning there may be occasional typos, spelling mistakes, or minor omissions.

Disclaimer

R.A.G.

1. RAG → Retrieval Augmented Generation

2. • LLMs are already pretty powerful now, they can answer you almost anything but, sometimes your knowledge (Model's) isn't enough or is outdated.

   Eg. "Who is Goku?" → You get a response from model.

   "What's Goku's newest form in upcoming series?" → The model doesn't know, so it searches online, gets info, reason over it and finally gives a response.

- So, we saw that there are 2 steps.

  Prompt → [Collect Information] → [Reason & Respond]

- Retrieval → The process of collecting information.
  More precisely, searching & fetching the most relevant information from a database to give LLM the facts it needs to answer a query.
- Generation → The process of reasoning over the collected information and responding is called generation.
- RAG is useful because —
  - Accurate responses (Factual Grounding), Reduces hallucinations
  - Up-to-date & Fresh responses, we can update it cheaply also!
  - RAG provides citation = which doc used to answer this query.
  - Significantly cheaper than fine-tuning.
  - Better security by implementing access control over information.
- LLMs were trained on massive open dataset on the internet however LLMs still don't know many things because
  - Private databases = Government, Companies, etc.
  - Hard to access information
  - Real-time data = News, stock update, etc.

How to solve it? = Just put it in the prompt!
(that's called the Augmented prompt)

- "Goku's newest form" ⟶ goes to RAG system
  ↓
  Retriever ⇌ Knowledge base
  ↓
  Augmented prompt (original prompt + Retrieved information)
  ↓
  LLM

- Retriever
  - An interface that,
    → Manages knowledge base of "trusted" information.
    → Finds the most relevant info & shares with the LLM.

- Prompt ⟶ (Normal LLM usecase) ⟶ LLM ⟶ Response

  Prompt → RAG system → Retriever ⇅ Knowledge base → Relevant Documents → Augmented Prompt → LLM

3. LLMs

- Autoregressive → LLMs are autoregressive or "self-influencing".
  → They predict the next token by treating its own previous output as an input for the next step.
  → Running the same prompt leads to different completions.

  e.g. The sun → The sun shines → The sun shines in the sky
  The sun → The sun rises → The sun rises from the east.

- LLMs generate probable word sequence not truthful text
  so Truthful ≠ probable.
  And knowledge gaps causes the model to generate inaccurate responses, which are referred to as Hallucinations. These are not actual hallucination, they are just knowledge gaps so the responses can sound right but are not true.

Metaphor *

	Collection	Organisation	Search
Library	Books on many Topics	Different sections and shelves	Librarian helps you find best sections & books
Retriever	Documents in a Database	"Indexes" for search	Retriever searches the index.

5. Keyword Search → How does it works?

- Traditionally used BoW & TF-IDF to identify how important a word is in a document & how often does it appear across the collection of docs.
- BM25 (Best Matching 25) - The newer approach.

Imp * A Search is not just about getting all relevant data/docs but it also decides which data/doc should come first (Ranking)

- let's see why we need BM25 over BoW or TF-IDF
  - BoW → Ignores grammar & word order. Just counts the words! say, you asked about 'Goku' in your query, and a spam doc the looks like this → "Goku Goku Goku Goku ---1000 times". It would get the highest score! or maybe a doc is too long & naturally contains more of that word. This is kinda dumb!
  - TF-IDF → Some words are more special than others so,
    more apperance = more important. → TF
    more rare = also important → IDF
    so score = frequency in this Doc × Rarity across all doc.
    But still it favours longer docs more & "Goku Goku --" problem exists here also.

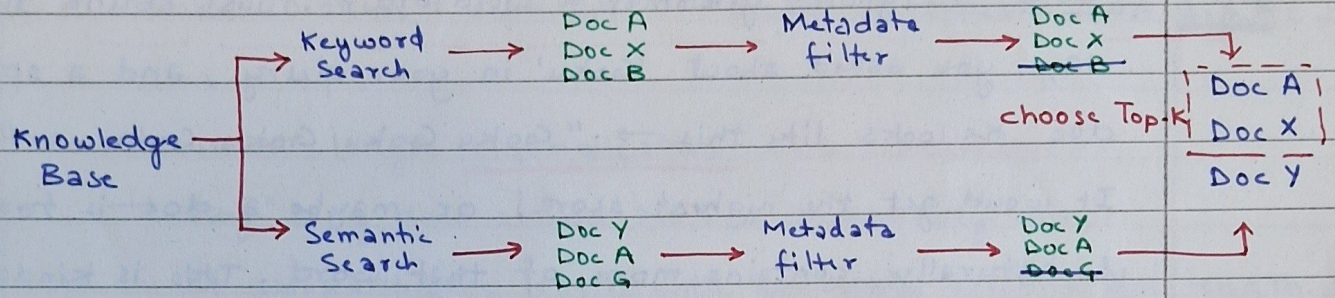
so limitations →

	BoW	TF-IDF
Context	No, only counts words	No, still count-based model
Repetition	Goku appears 100 times so score is 100x	Same, 20 mentions of a word = 20x score
length	long docs get higher score	While it penalizes common words, still rewards longer docs
Importance	Equal weight, "The" is equal to "Transformers"	Solves this with IDF, common words get downranked.

2. Keyword Search (The traditional approach)
- Retriever looks for documents containing the exact words found in the prompt.

b. Semantic Search (The modern approach)
- Retriever looks for documents with similiar meaning to the prompt, even if keywords don't match (more flexible)

c. Hybrid Search (The combination) - More modern approach
- Uses both Keyword & Semantic search to get 2 different lists of documents, applies meta-data filtering to both lists & selects top-k docs after combining both.



* Metadata Filtering → After docs are selected from search, some docs are removed who don't the "rigid criteria' e.g., only IT department data required, so we remove all other department's data, this stage is called metadata filtering.
- Uses metadata like Title, author, creation, etc to narrow down documents.

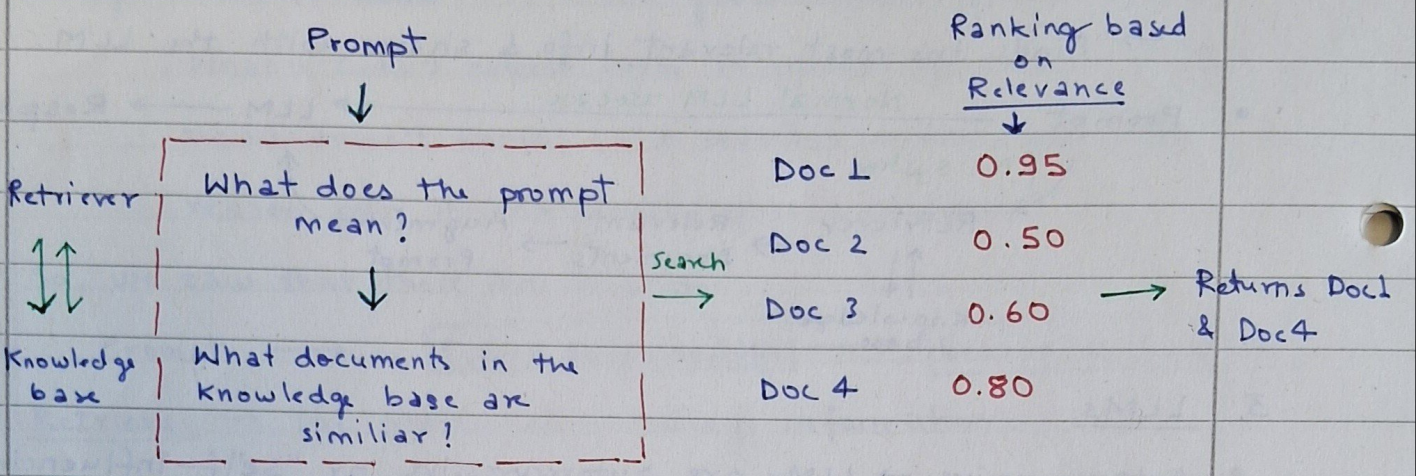
* Bag of Words (BoW) → Used in Keyword Search
- A technique that throws all unique words of all the documents/sentences in a bag & ignores word order, grammar & context, and only focuses on word frequency.

* Term Frequency - Inverse Document frequency (TF-IDF) Keyword Filtring
TF → How often does the word appear in this document?
IDF → How unique/rare is this word across all docs?

* Search with a librarian

- Understands the meaning of your question.
- The ability to interpret the meaning of your question allows them to identify the right sections/shelves of the library to search.
- Eventually find relevant books

Search with Retriever



- Understands the meaning of question
- Searches millions of docs which are similiar to user prompt e.g cosine similarity, to get a candidate pool.
- Ranks each doc in candidate based on relevance to the prompt by using a smarter model/encoder.
- Selects Top-k docs to be sent to LLM.

Tradeoff (Relevance vs Irrelevance)

- if we send all docs, we will waste context window & Money
- if we send Top-1 doc, we miss valuable info

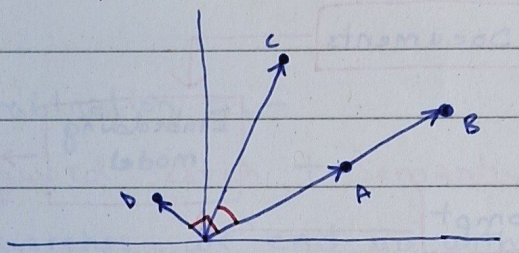
4. Search foundations → When we go to the retreiver, it goes through bunch of data and get docs which are most relevant based on search strategies.

There are 2 kinds of searches. —

[Techniques Used]

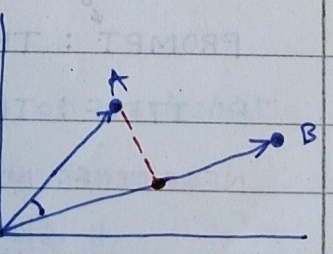
- Euclidean distance → can be used to find the distance between two points in the * latent space. BUT, things (similiar ones) are far away in latent space so,
- Cosine Similarity (comonly used)
  - → Measures the angle between 2 vectors, not length.
  - → Smaller angle = pointing is similiar direction
    larger angle = pointing in opposite/different direction.
  - → Valu of Cos = [-1, 1]
    where, cos 0° = 1 (if angle betw two vectors is 0°, they are in exact same direction.)
    cos 90° = 0 (Unrelated, orthogonal)
    cos 180° = -1 (Exact opposites)

$$\text{cossim} = \frac{A \cdot B}{\|A\| \|B\|}$$



cossim (A, B) = 1, (A, C) = 0.5 - 0.7, (A, D) = 0

- Dot Product
  - → Measure both direction & length betw 2 vectors.
  - → It is length sensitive so if we compare a sentence & a doc (many sentences), the doc will automatically get a huge score as it would have huge length even tho it may not be really related.
  - → So we have prompt as vec $\vec{A}$ & a doc as vec $\vec{B}$, we get a score on how related they are.
  - → Dot product = $[-\infty, \infty]$



so Dot (A, B) = -ve (opposite dir)
Dot (A, B) = +ve (same dir or similiar)
Dot (A, B) = 0 (Orthogonal)

- Simple pipeline (Almost same as keyword search)
  - ◇ Prompt & all documents gets a vector (or converted to)
  - ◇ Vectors are compared to get a score
  - ◇ Difference b/w K.S & V.S is how vectors are assigned or converted
    - → Keyword Search = count words
    - → Semantic Search = Use embedding model.

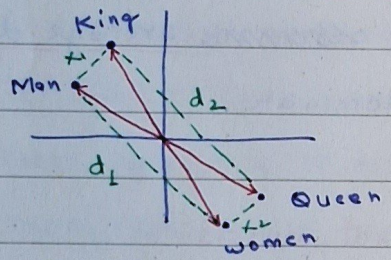
- Embedding model → NN that converts text, images or audio into vector that captures the underlying meaning.

* Linear Representation Hypothesis

→ concepts like truthfulness, languages etc are encoded as specific linear direction.

→ This theory explain why embedding models work.

→ So the idea is → similiar things are grouped together (2D) or are in ~~same~~/similiar directions.

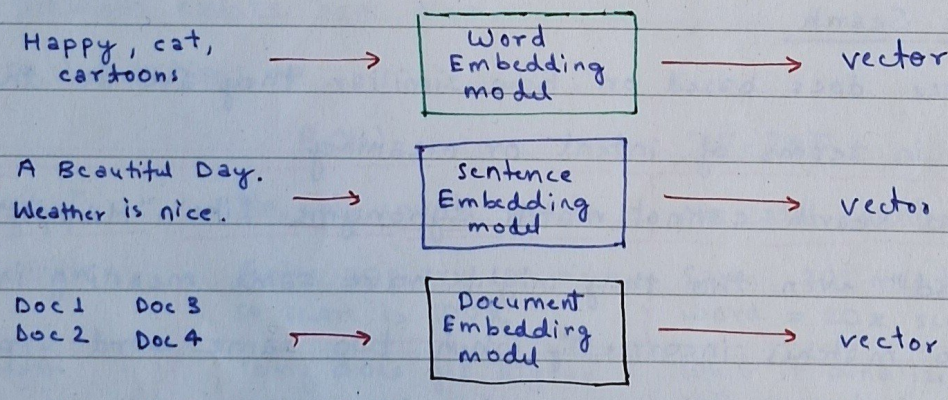


→ $d_1 \approx d_2$ (In theory)

→ Queen − Women = King − Man ($x_1 \approx x_2$)

→ They are not opposite direction, just example

- There are many types of embedding models



so the idea is similiar things are kind of together in latent space.

BM25 → called Best Matching 25 as it is the 25th iteration of the Ranking function in the series.

So TF-IDF has 2 big problems

1. More apparance of a word = higher score
2. longer docs automatically has higher score as they naturally have more words.

* Solution by BM25 (Smarter TF-IDF)

1. Term Frequency Saturation (K1 Parameter)
   → seeing a word 100 times is Not more useful than seeing it 5 times

TF-IDF	BM25
"Goku" 10 times = score X	"Goku" 10 times = Score X
"Goku" 20 times = score 2X	"Goku" 20 times = Score 1.3 X

2. Document length Normalization (b parameter)
   → say "long docs should NOT automatically win."
   It compares doc length with ~~automatic~~ average doc length.
   so huge noisy docs don't dominate.

so, BoW → Counts words
TF-IDF → Count important words
BM25 → Count important words, Intelligently!

6. Semantic Search
- Retrieves docs based on how similiar they are to the prompt in terms of intent or meaning.
- Keyword search cannot match synonyms like "Happy" & "Glad" even tho they might have same meaning in context. It also matches incorrectly when two same word appear but with different meaning eg. 'Python' snake or Python programming language.

- Score points are reciprocal so, for a list A (Assume K=0)

RANK 1 = Doc X = $\frac{1}{k+1} = \frac{1}{0+1} = \frac{1}{1}$ = 1 score

RANK 2 = Doc Y = $\frac{1}{k+2} = \frac{1}{2}$ = 0.5 score

RANK 3 = Doc Z = $\frac{1}{k+3} = \frac{1}{3}$ = 0.33 score

so if there are 2 lists & a doc is at 1st & 10th rank in both.

doc X = $\frac{1}{k+1} + \frac{1}{k+10} = \frac{1}{1} + \frac{1}{10}$ = 1.1 score

Note → It is rare to go above 1, due to k

- Same way, we rank all the documents & get a list

- **k-parameter** → controls the impact of highest rank doc.
  - If k=0, The Top ranked doc in any list will instantly go the top of final ranking, even it performed bad in others.

so we need to control that single high rank doesn't dominate final ranking. so say for same example K = 50

doc X = $\frac{1}{k+1} + \frac{1}{k+10} = \frac{1}{50+1} + \frac{1}{50+10} = \frac{1}{51} + \frac{1}{60}$ = 0.036

so,

lower k → More value to Top rank

higher k → Less value to Top rank

- **Beta Parameter** → control dominance of Vector & Keyword Search.
  - If $\beta = 0.8$ ⇒ 80% importance to Semantic search & 20% importance to keyword
  - This val depends on applications, if keyword matching is more imp, then give it a higher score, etc. eg for medical database, you need exact medicine name.

$$\text{Score} = \sum_{\cdots} \left(\beta \times \frac{1}{k + \text{rank}} \right)$$

SAMPLE is called as Anchor in this process.

- So basically we create pairs like RLHF

  CHOSEN = { SAMPLE * POSITIVE }

  REJECTED = { SAMPLE * NEGATIVE }

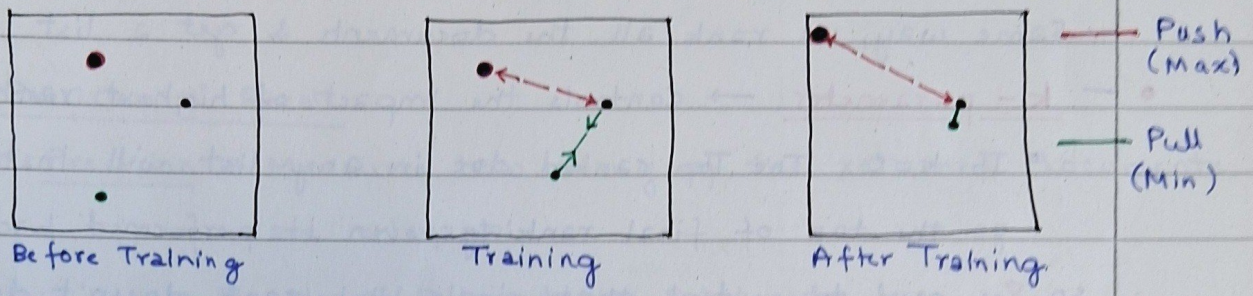
  * is not multiplication Just represents Pair

- We train the embedding model on these contrastive pair.

  An example.

  1 : { ["Good Morning", "Hello"], ["Good Morning", "WTF"] }

  ---250--- N samples for training.



8 Hybrid Search

We saw that it is a combination as -

Metadata filtering → Keyword Search + Semantic Search.

Say after M.F. & the searches, we end up with 2 lists of 30 and 40 documents. So, how do we combine them?

- Reciprocal Rank Fusion (RRF)

- Technique / Algorithm to merge N no. of lists into one list with fairness.
- It solves the "apple vs orange" problem, how do you compare or combine a BM25 score of 25 to vector score of 0.8.
- RRF does 2 main thing
  ① Reward docs for being highly ranked on each list
  ② Control weigth of Keyword vs Semantic search.
- Formula → (K is a hyperparam) see later for it.

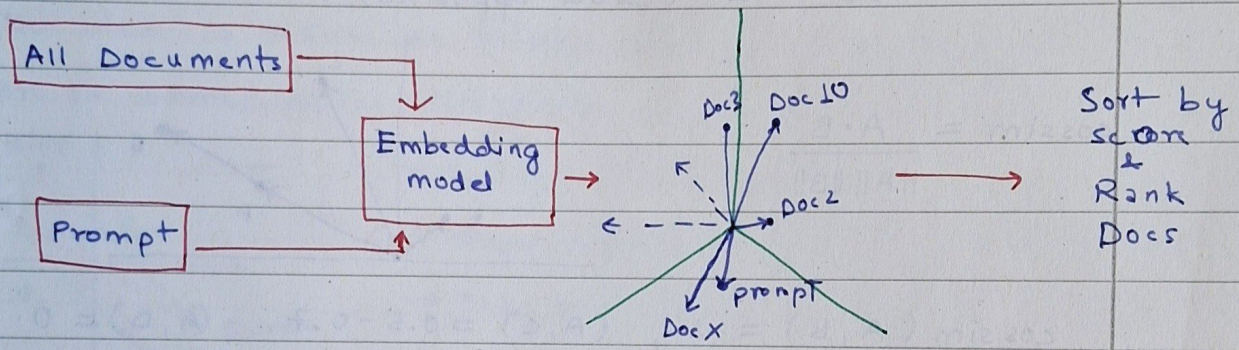
$$\frac{1}{k + \text{rank in list } 1} + \frac{1}{k + \text{Rank in list } 2} + \cdots \frac{1}{k + \text{Rank in List } N} = \sum_{\text{reRanking}} \frac{1}{k + \text{rank}(r)}$$

→ So we get dot products of prompt with each doc & get a score. We choose the top-k scores as they are the strongest match.

→ So, cosine similarity = Only angle

Dot product = Angle + length.

* But larger docs get biased scores so Dot product might not be ideal. BUT, Modern approaches normalize the dot product to remove length bias, so they become almost like cossim & are preferred over cossim as they are computationally cheaper.



7. Training an Embedding model

— Training an Embedding model is very similiar to RLHF training. The training is done using a Process called as Contrastive Learning.

PROMPT (or sample) : The weather is sunny. (Target sample) - T)

POSITIVE: It is a bright day. (Sample similiar in meaning to T)

NEGATIVE: My notebook is full. (Sample different in meaning to T)

Loss function → Minimize distance bet^{n} T & +ve, and
Maximize distance bet^{n} T & -ve.

Cosine Similarity → Measures angle similarity (whether vectors point in similar direction)

Euclidean Distance → Measures physical distance b/w 2 vectors.

10. Information Retrieval with Vector Databases

We saw how we use "Hybrid Search" to retrieve documents. While Relational databases like SQL might be good for Keyword search, they perform very poorly for semantic search (or they can't do it at all).

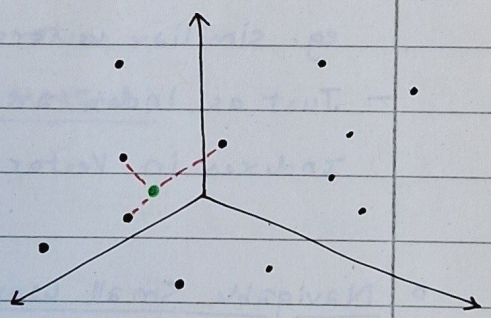
- Relational databases are used for
  - → Exact keyword matching, structured data

but they fail for semantic search. (similiar meaning, different words)

so, we need a special database → Vector Database.
A database that stores embeddings of docs but also do indexing for querying & CRUD.

- If you store vector embeddings of docs in Vector DBs then how to do searches to get similiar docs back?

- KNN (K-Nearest Neighbours) — Basic vector retrieval
  - Vectorize all docs & prompt.
  - Calculate distance between prompt & ALL doc vectors
  - Sort by distance (closest - farthest)
  - Return k closest docs eg k=3 ---



It's good & easy to get closest semantically similiar docs but
Limitations → Compares query against ALL vectors.
- so, good for 100 or so docs but if DB has 100 Million docs, it will be very very very slow. so NOT scalable!

Final list After RRF

RANK	RELEVANT?
1	Relevant
2	—
3	—
4	Relevant
5	—
6	Relevant
7	Relevant
8	—
9	Relevant
10	Relevant

Assume for prompt X, Retriever returned these 10 Docs. Ground Truth for X = 8 Docs.

→ Precision@5 = 2 out of 5 = 40%. Recall@5 = 2 out of 8 = 25%.

Precision@10 = 6 out of 10 = 60%. Recall@10 = 6 out of 8 = 75%.

- We can see how value of Top-k affects Eval. Generally Top-k is used betw [5-15]
- We can calculate Recall@K & Precision@K for every row in the Table to see best K

- **MAP@K (Mean Average Precision)**
  - Evaluates avg precision for relevant documents in first k-docs.

- **MRR (Mean Reciprocal Rank)**
  - Measures how well the retriever placed/Ranked a right or relevant doc in the list
  - if First Relevant (as per ground truth) is at Rank 1 so score = 1.0
    if First Relevant is at Rank 2 = 0.5
    ——— " ——— at Rank 3 = 0.25
  - Formula. = Reciprocal Rank = 1 / Rank

so, the later the first Relevant doc appears, the worst MRR it will have.

MRR just averages over many prompt's Reciprocal Rank.

9. Evaluation of "Retriever"

first, we require common ingredients:

Prompt — Ranked Results (Documents returned by Retriever) — Ground Truth (All docs labelled as Relevant or Irrelevant for this prompt)

It is just like supervised learning, you need to know the correct answers if you want to evaluate generated results.

- Precision & Recall

- Precision = Measures how many of the returned documents are relevant.

  $= \frac{\text{No. of Relevant Doc Retrieved}}{\text{Total Doc Retrieved}}$ e.g. $\frac{8}{12} = 66\%$ Precision

- Recall = Measures how many of the Relevant documents are returned.

  $= \frac{\text{No. of Relevant Doc Retrieved}}{\text{Total Relevant Doc}}$ e.g $\frac{8}{10} = 80\%$ Recall

  (for a prompt as per Ground truth)

so,

Precision → Penalizes for returning irrelevant documents.

Recall → Penalizes for Leaving out relevant documents.

- Top-k → Retrieval eval is influenced by value of Top k.

  that is — How many docs the retriever returns

  → Metrics are discussed in term of Top-k docs.

  e.g. Recall @ K or Precision @ K → metric at value of k.

(Don't worry, see next page ☺)

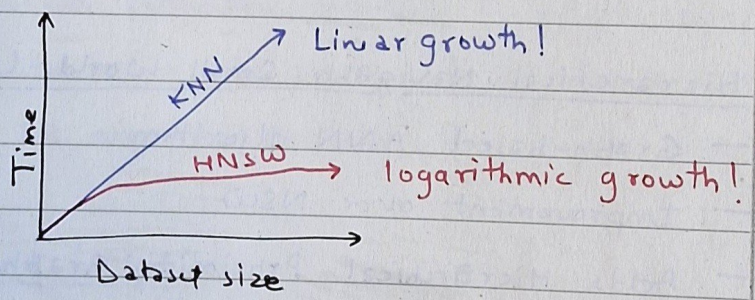
- layer 1 contain all docs, then 100 Random are chosen for layer 2 and new proximity graph is formed then same for layer 3 with 10 Random nodes.

Working (Search)

- Chooses random entry point in Layer 3
  → Search for closest candidate in layer 3
  ↓
- Starts at best candidate from layer 3,
  → Search for closest candidate in layer 2
  ↓
- Starts at best candidate from layer 2
  → Search for closest candidate in layer 1
  = This is the closest candidate to Query vector

It is really efficient as in layer 3 & 2, the algorithm makes big jumps & ignores unnecessary nodes.

Scalability



HNSW scales well with Billions of vectors!

so, KNN is exact but very slow
ANN (NSW → HNSW) are significantly faster but can't gurantee best matches.

* Creating Proximity Graphs is Expensive! but it is pre-computed!

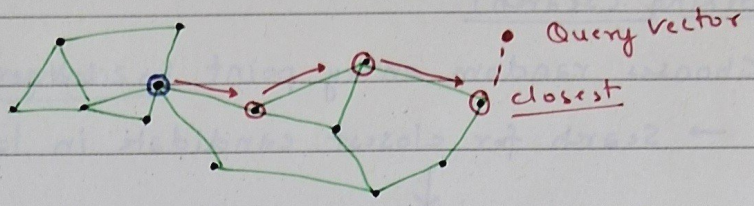
① – Compute distances between all document vectors.

② – Each doc is represented as a node in graph

③ – Connect each node to its nearest neighbour.

This graph is called as Proximity graph

Working (Search)



– Vectorize query

– Randomly chooses an 'Entry point' (any doc or node = candidate vector)

– Looks at each neighbour of the candidate node eg- 4 nodes

– Calculate which of the candidate's neighbour closest to query vector.

– Closest node to query vector becomes new candidate

– Process repeats until closest is found.

It is more efficient than KNN but

Limitation → Graphs can become dense with increasing docs.

so there is something even more efficient.

- Hierarchical Navigable Small World (HNSW)

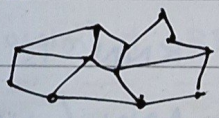
– Graph-based ANN algorithm.

– Improvement over NSW

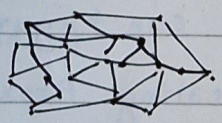
– Adds Hierarchical Proximity Graphs.

Say, there are 1000 docs so 1000 vectors

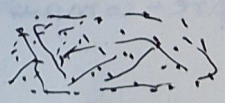
Layer 3 – 10 vectors kept Randomly & new P-graph



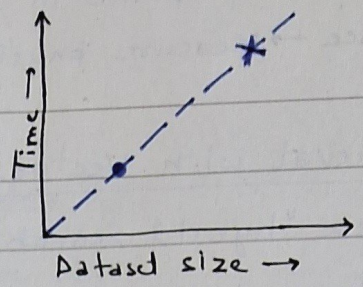
Layer 2 – 100 vectors kept Randomly & new P. graph



Layer 1 – All 1000 vectors with complete proximity. graph



- Scaling challenge



linear Growth ↓
=
Not Scalable

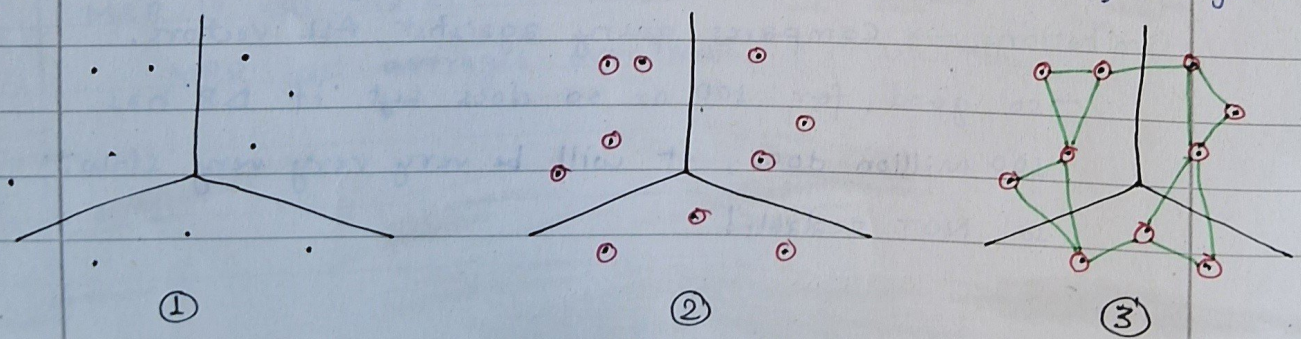
- • → 1000 docs = 1000 distance calc. per search
- x → 1 Billion docs = 1 Billion distance calc. per search.

Solution?

- **ANN (Approximate Nearest Neighbours)**
  - Don't find exact or perfect nearest neighbours but find very close neighbours (approximate)
  - Significantly faster than KNN
  - Tradeoff = Tiny accuracy loss But massive speed gains.
  - Relies on additional new data structure.

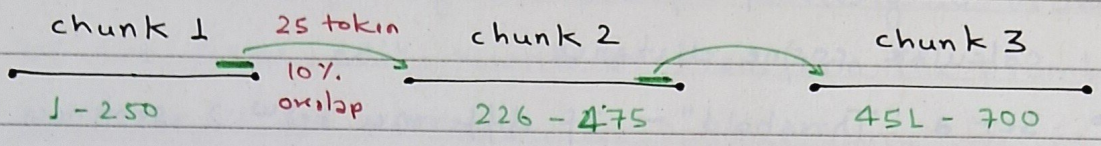
- **Index**
  - Smart Data Structure ~~for~~ a faster search.
  - Indexes organize vectors in a smart way.
    eg. similiar vectors are connected together through edges, etc.
  - Just as index are there in library eg. Transformers = ID350
    Indexes in Vector DB organize ~~do~~ vectors in smart way.

- **Navigable Small world (NSW)**
  - Graph-based ANN algorithm.
  - Idea: Similiar vectors are connected together through edges.



b. Overlapping chunks

- The chunks are fixed size but there is overlapping in content.
- Overlapping → Include a small portion (eg 10%) of previous chunk at the start of the next chunk to preserve context.



c. Recursive character Chunking

- Splitting text into chunks at a specified char. eg Newline.
- It uses a hierarchy to keep related text together.
  - Chunk at double newline → "\n\n"
  - If para too big → chunk at single newline → "\n"
  - If line too big → chunk at space → " "
  - If space too big → chunk at word → [RARE]

There are still issues with above method, they still break context.

"She dreamed that night, that, she is an Olympic champion."

- The second chunk would make the model think that she is already champion and loose meaning that it is a dream.

so we need more advanced chunking methods.

d. Semantic Chunking

- Groups sentences together based on similiar meaning

"Goku is pure of heart and also strong. Saiyans, on the other hand, were killers."

- Pros: → Higher recall & Precision
  → Smaller chunk boundaries
- Cons → Computationally Expensive
  → Slow
  → Threshold tuning.

11. Chunking

— Process of breaking larger documents into smaller, manageable pieces (chunks) before turning them into vectors

Why do we need chunking?

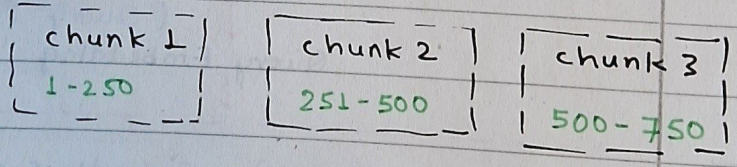
1. Token limit → Embedding models have limit of tokens around (512 - 1024 Tokens).
2. Precision → smaller chunks gives exact para needed so improved relevancy
3. Cost → sending precise 10 docs (chunks) is better than sending 5 large docs.

say, you have 1000 books into your knowledge base.
you vectorize the books directly so now you have 1000 vectors.
Problem is → This compressed entire book meaning into single vector.
→ can't sharply represent specific topic, section, etc.
we get very poor search relevance with this.

— chunking can be (for a Book let's say)
- Page level
- Paragraph level
- Sentence level

— Too large or Too small chunks, both are bad.

- Types of chunking.

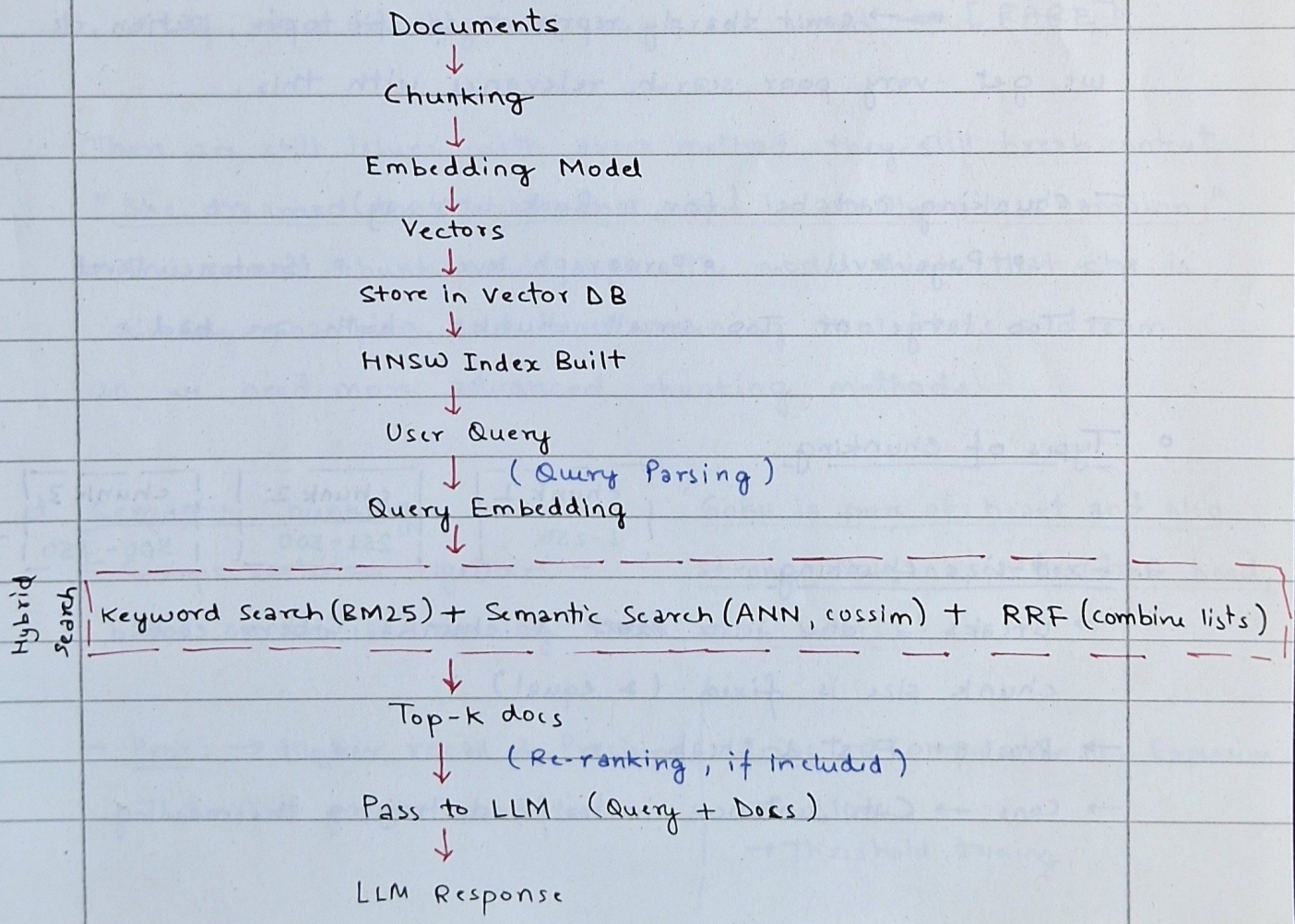


2. Fixed-size chunking.
→ Breaks a doc into ~~exact~~ X chunks where each chunk size is fixed (& equal)
→ Pros → Fast & cheap
→ Cons → Cuts sentences in half, destroying the meaning

- **Vector Database**
  - Database designed to store high-dimensional vectors
  - Perform vector search using ANN algorithms.
  - Designed to build HNSW indexes & compute vector distances.
    - stores text & its vector.

- **Vector Database Operations**
  - Database setup
  - Load Documents
  - Create Sparse vectors for Keyword Search.
  - Create Dense vectors for Semantic Search.
  - Create HNSW index to power ANN algorithm.
  - Run Searches

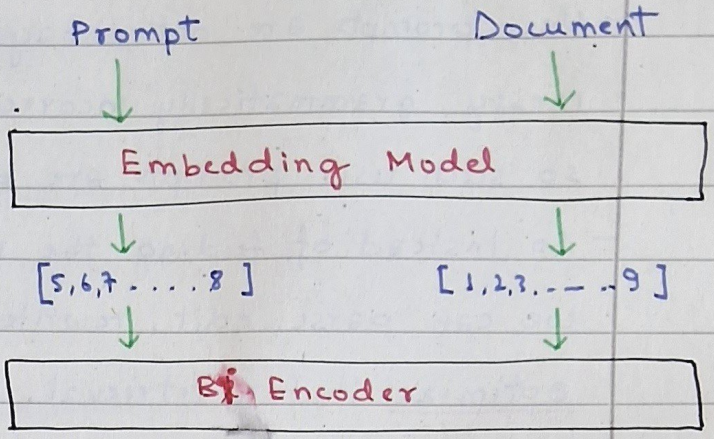
* **RAG Pipeline**



* Bi-Encoders & Cross-Encoders & ColBERT

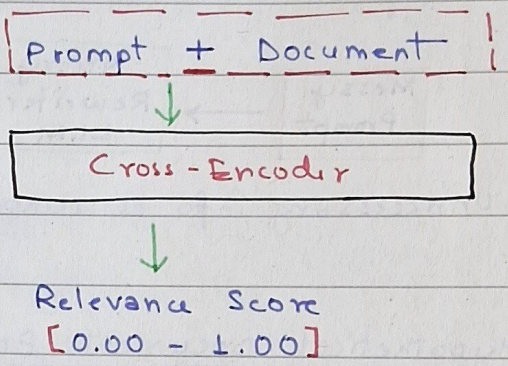
∘ Bi-Encoders (This is what we saw so far)

- Docs & prompt are embedded separately.
- Bi-enc = ANN search by vector DB to find doc vector closer to prompt.
- Doc embeddings are pre-computed so it is faster.



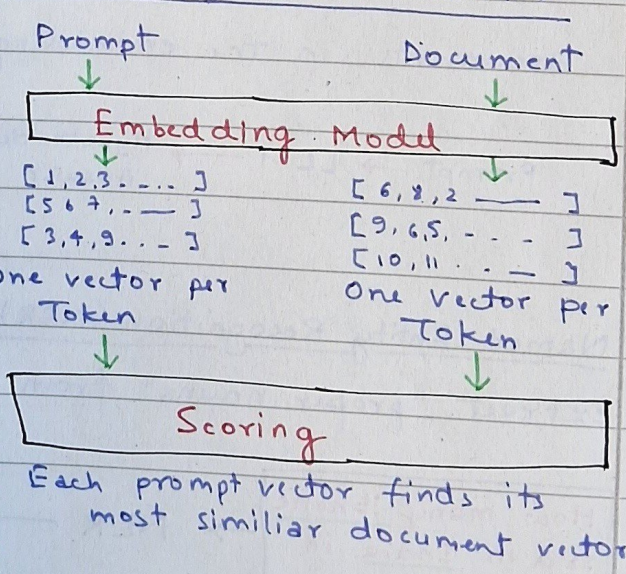
∘ Cross-Encoder

- Concatenate Doc & prompt
- Feed to Cross-enc = It develops deep contextual meaning of interaction between prompt + doc.
- Generates Relevance Score.
- Does this for each Prompt + Doc pair.
- Pros → Way better search results than Bi-encoder (vector search)
  Cons → Too slow & scales terribly., can't preprocess



∘ ColBERT (Contextualized Late Interaction Over BERT)

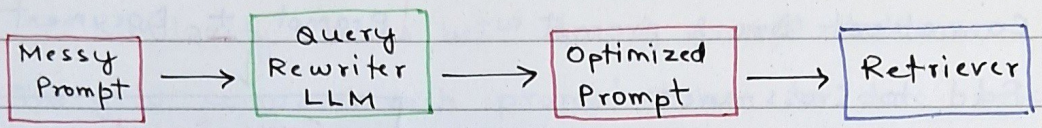
- Hybid of Bi & Cross enc.
- Pre-compute doc vectors just like Bi-enc
- Instead of generating just 1 vector for doc, it turns every single word (token) into its own vector, same for prompt = Dense vector
- Scores →



12. Query Parsing

- User prompts are not always the "ideal" language. They are messy, grammatically incorrect sometimes or too vague. so such user prompts are bad for search queries with vector DB.
- So instead of feeding the prompts directly to Vector DB, we can parse, edit, rewrite or transform the prompt to optimize it for retrieval. We optimize query for searching!

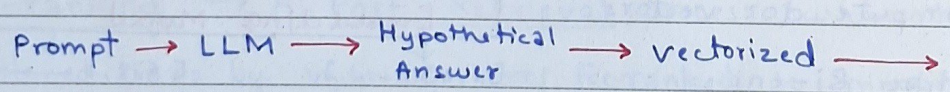
- Query Rewriting (Most commonly used)
  - Using an LLM to rewrite the query before sending to retriever.



  - Unnecessary info is removed & prompt becomes direct & optimized.

- Hypothetical Document Embeddings (HyDE)
  - Sometimes User's query is too short or very vague eg- "Why are my hands sore?" or "Why my remote doesn't work?".
  - HyDE solves the problem by using an LLM to generate a Fake or Hypothetical answer/document for user query.
  - This answer/document is vectorized & used to search the D.B.
  - Idea is - The answer/doc is likely to land near the "true" documents in the embedding space

'True' does nearby the 'Fake'



- Named Entity Recognition (NER) — Not direct query parsing but extract "proper nouns" from query to apply Metadata filters.

How many iphone sold in India in 2027? → NER → Filter = Show me chunks where country = "India" & year = "2027".

- Working
  - Break entire document into individual, complete sentences.
  - First sentences becomes our first chunk.
  - Vectorize the first chunk & the next sentence then calculate cosine distance
  - Set a "Threshold" — If difference betw 2 sentences is smaller than Threshold, they stay in same chunk, else not.
  - Now new chunk & next sentence is rectorized & compared.
  - This process continues until next sentence is semantically different than the chunk.

e. Language or LLM-based Chunking (Agentic Chunking)
- We give our document & an instruction prompt to an LLM.
- We ask LLM to chunk the document in a certain way.

Pros → High Retrieval accuracy,

Cons → Expensive (becoming cheaper tho), latency, prompt-dependence.

f. context-aware Chunking or other chunking.
- Same as LLM-based chunking but LLM is asked to write a context for each chunk.
- We send both context & chunk to LLM for response in final.
- cons → Costly Pre-processing (LLM adds context chunk by chunk)
- Pros → Better searches
- Can be applied over any chunking method.

Quick Revision!

° Attention Head (Multi-head Attention)

Attention → How much every word in a sentence depends on or relate to every other word.

→ Helps a model to decide - which words are important for understanding the current word?

→ Eg - "Goku is strong because he trains hard!".

Attention helps model to understand that he = Goku (Refer to)

Attention Head → One independent "focus" mechanism.

→ Head 1 = Grammar, Head 2 = Action, Head 3 = Reason, etc.

so multiple attention heads = Multi-head Attention (in parallel)

10 Tokens = 10 QKV pairs (single head)
10 Attention head = 10 x 10 Tokens = 100 QKV pairs
so 1 Giant QKV matrix

Core Components of Attention →

Every token creates 3 vectors

1. Query (Q) - "What am i looking for?"
   - Current word asks = Which other words are important to me?
   - word = "It" → Query asks - what does it refers to?
2. Key (K) - "What information do i contain?"
   - Every word exposes a 'key' describing itself.
   - word = "Dog" → key has info like - noun, animal, etc.
3. Value (V) - "what information should i pass forward?"
   - contain actual information, if attention selects this word, this value is passed forward for generation.

How Q, K, V work together → Using Youtube, you type a Query (Q) in search bar, Youtube compares Q against the keys (K) (video titles) and then plays Value (V) (actual video) for best match.

→ Query word searches the sentence against Keys of all other words, gets a similarity score & decide which word's value (meaning) to use.

13 Quick Transformer Overview

ENCODER	DECODER
- The Reader.	- The Writer
- Reads the raw text & turn it into deep mathematical meaning.	- Designed for Generation. Generates text token by Token. They predict next word.
- Bidirectional → looks at the whole sentence all at once. left-2-Right & Right-2-left.	- Autoregressive → Generates next word & then uses its previous output to generate next token.
- Self-Attention → Every word in sentence looks at every other word to calculate how they relate to one another.	- Causal Masking → Only see previous tokens, not future! When generating word #4, it is forbidden to see word #5.
- Outputs a dense grid of vectors.	- Outputs the next token
- Develops deep contextual understanding of the text's meaning.	- Self-Attention → Modern decoder uses this. Each token looks at every other previous tokens.
- Best for understanding tasks	- Cross-Attention → Older model used this. Decoder looks back at Encoder's map, to match original.
- Ex → BERT, ROBERTA	- Ex → GPT, Llama, Gemini
- Used in RAG systems for Embeddings, Reranking.	- Modern LLM only use decoder.
- Also used for classfications tasks & sentiment analysis.	- Used for generation tasks like chatbots, coding, etc. RAG Generation.

- Bi-enc looks at similarity score betw Prompt vector & Doc vector only! but ColBERT looks at similarity score betw each prompt vectors to each doc vectors

		Document Tokens					
		Rice	is	a	famous	and	----
Prompt tokens	How	0.1	-0.6	-0.2	0.3	-	—
	to	0.1	-0.1	-0.6	-0.2	-	-
	make	0.2	-0.1	-0.1	-0.2	-	-
	Rice	0.9	0.2	0.1	-0.1	-	.

similarity scores betw doc & prompt tokens.

The grid compares each prompt token to doc token

The max scores are summed up to get final score.

eg. 0.9 + 0.2 + --- = 3.6 (Relevancy score = Maxsim)

Pros → Scalable like Bi-enc & almost rich interaction as Cross.

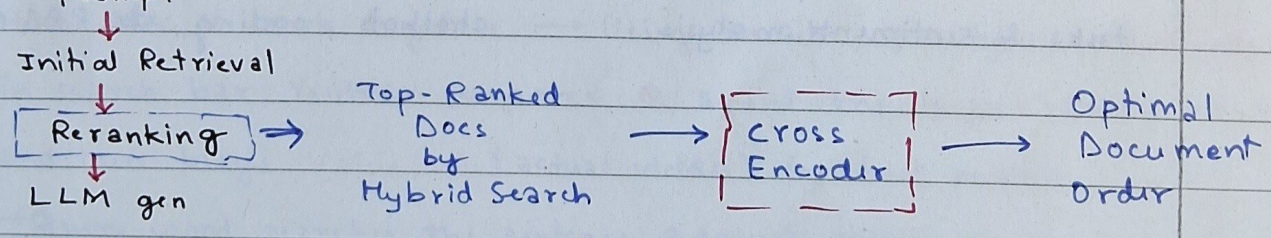
Cons → significant vector storage.

Bi-encoders → Good quality, Great Speed, Minimal Storage.

Cross-enc → Best quality, Extremly slow, Minimal Storage.

ColBERT → Great quality, Decent speed, Significant Storage.

* Re-ranking → A Post-Retrieval process, where the initially retrieved docs by VectorDB are Reranked to improve order.



→ Adds minor latency but better results.

—Cost
—Speed
—Quality

3 Main Aspects ←

17 RAG in Production

* Challenges

- Scaling Performance (More traffic, higher memory & compute).
- Unpredictability of prompts from users.
- Messy real-world data. (PDF, slides, Images).
- Security & Privacy.

* Key Metrics

- Software Performance Metrics
  - → Latency, throughput, memory, token/sec, compute, Traces.
- Quality metrics
  - → User satisfaction, system output quality.

Follows a prompt's path through the entire RAG pipeline

* Eval scope

- ~~Component~~ System
  - → Overall RAG system (end-to-end)
  - → Shows you what problems exist
- Component
  - → Individual parts of your RAG system
  - → Shows you where & why problems occur

* Eval types

- Code-based → cheap, faster, unit-testing
- LLM-as-a-Judge → LLM judges different component.
- Human feedback → costly, real-world quality/issues from users.

* Libraries for Eval

- RAGAS
- TruLens

* Platform for LLM/RAG observability

- Phoenix

15 Choose the right LLM!

a. - For a RAG application, don't look at general leaderboard for choosing an LLM but for following benchmarks →
   1. Instruction following, eg. IFEval leaderboard
   2. Long context accuracy, eg LongBench v2
   3. Multi-hop reasoning eg. MRCR v2 — connect multiple docs to solve complex problem

b. Architecture : Standard vs Reasoning models
   - Standard (Non-reasoning) → Best for majority of RAG apps. They are fast & good for summarizing
   - Reasoning models → for complex docs & Tasks.

c. Task Type → Assistant, qna, summarization? look at leaderboard.

d. Context window

e. Cost

f. latency

g. Open source vs closed model
   - Open source → Privacy, flexibility, cheaper
   - Closed → stronger performance, easier setup

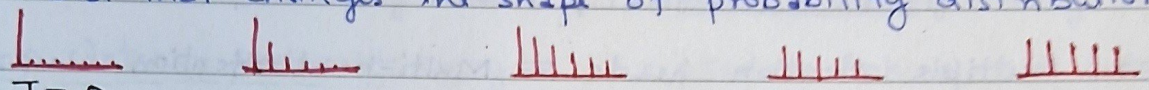
h. Fine-tuning Support ? — PEFT (LoRA, QLoRA) available?

i. Hallucinating model ? — check ALCE benchmark leaderboard.

16. Reducing Hallucinations
   - There's no perfect solution for LLM hallucination, even for RAG.
   - Update system prompt to ground answers with Retrieved info only.
   - Citation Generation — cite sources of responses.
   - Use ACLE Benchmark on your RAG system.

14. LLM Generation Parameters (Sampling strategies):

- **Greedy decoding**
  - → When True, or Temperature = 0 or Top-k = 1,
  - → Strictly selects the highest probability word everytime
  - → Deterministic (same prompt = same response)
  - → Generic sounding text (Predictable), can be repetitive
  - → Useful for → facts, code completion, debugging
- **Temperature**
  - → Control's LLM's Randomness.
  - → Dial that changes the shape of probability distribution.



T = 0	T = 0.5	T = 1	T = 1.2	T = 5
Greedy Decoding	Most likely Tokens will be generated	Original Distribution	'Creative' chances for unlikely Tokens = more variety	'non-sense' everything has almost equal chance to be chosen.

- **Top-K** → Chooses only from Top-k most likely tokens
- **Top-P** → Picks from tokens whose cumulative probability is below some threshold.
  - → Top-p = 70, includes tokens until their probability is 70%.
- **Repetition Penalty**
  - → Reduce the probability of already used tokens
  - → Prevents → loops (I think, I think) → overuse of specific words
- **Logit bias**
  - → Param that manually forces an LLM to either use or completely avoid specific words.
  - → Filters profanity, boosts categories in classifier LLMs.

Facts or code → low temp & Top-p

Creative domain → higher temp & top p.