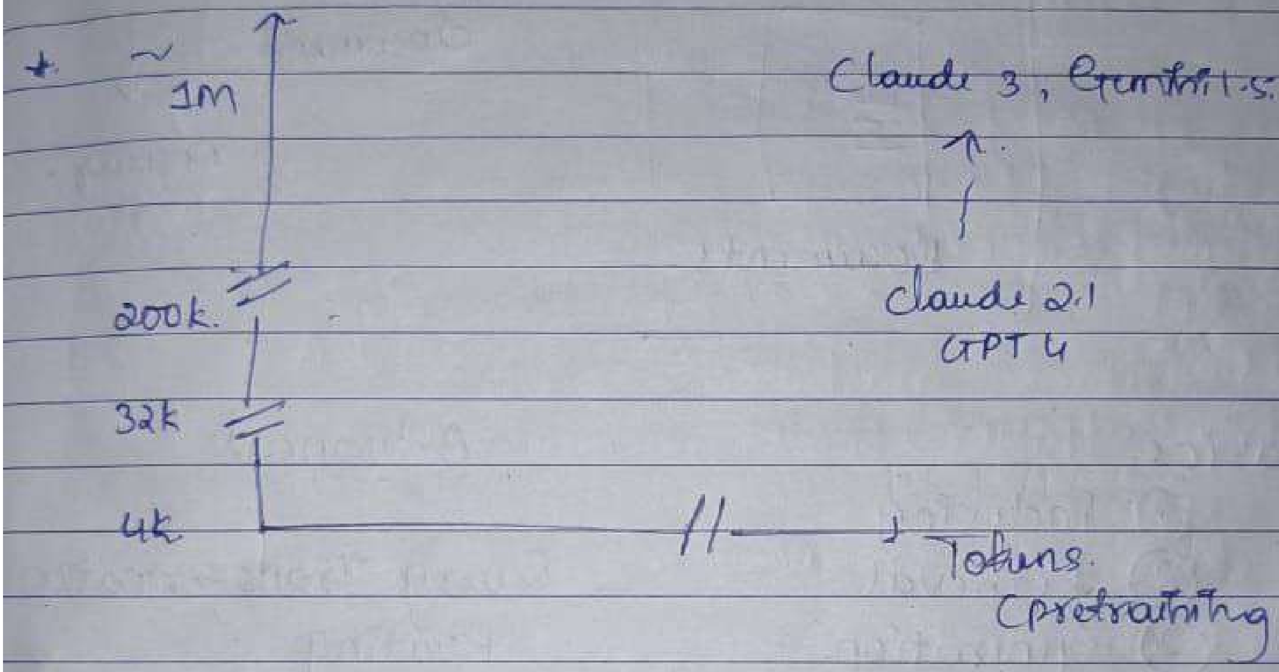
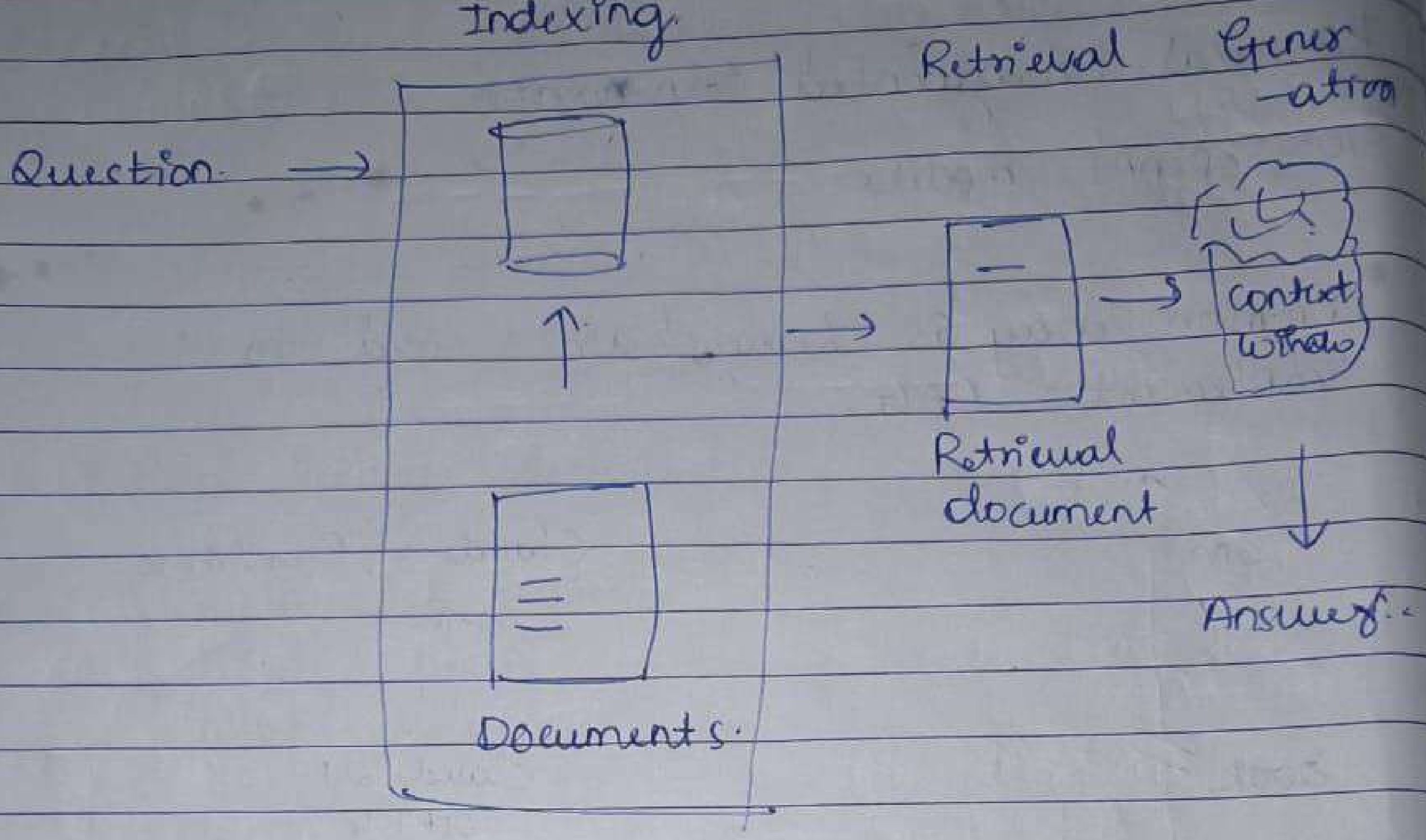
* RAG
* Retrieval Augmented Generation.
70% openAI models.

* common way is Langchain used to implement RAG.



* Connecting LLMs to External data is a central need

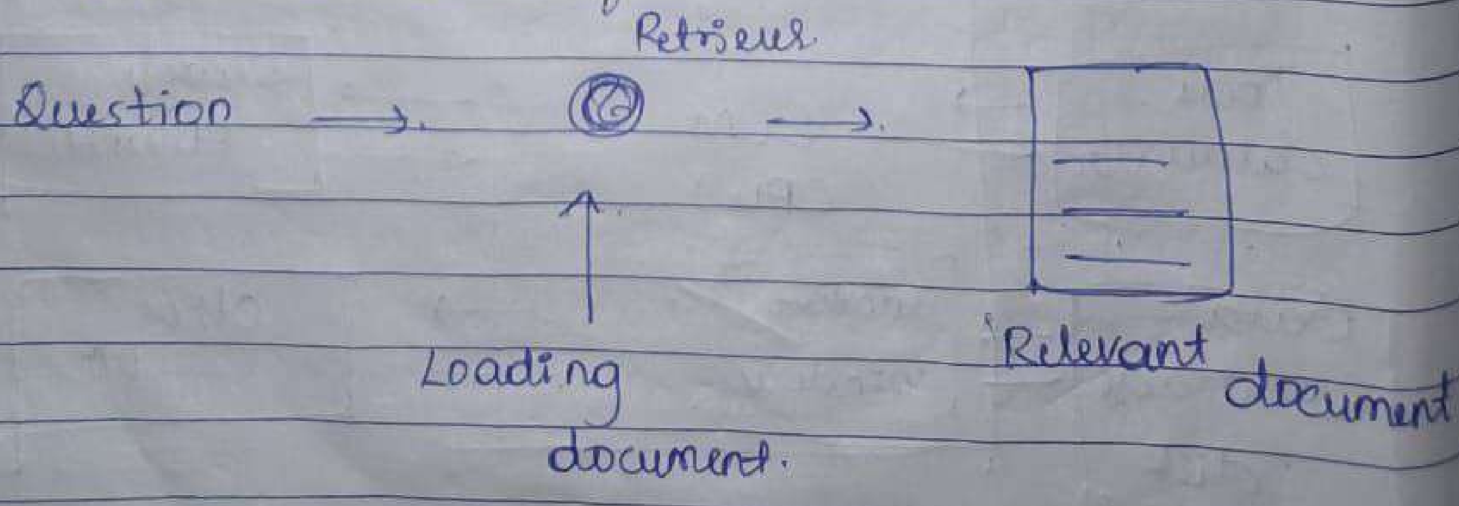


* <u>Basics</u>
  1. indexing
  2. Retrieval
  3. Generation

<u>Advanced</u>

Query Transformation
Routing
Query construction
Indexing
Retrieval
Generation

* document loading

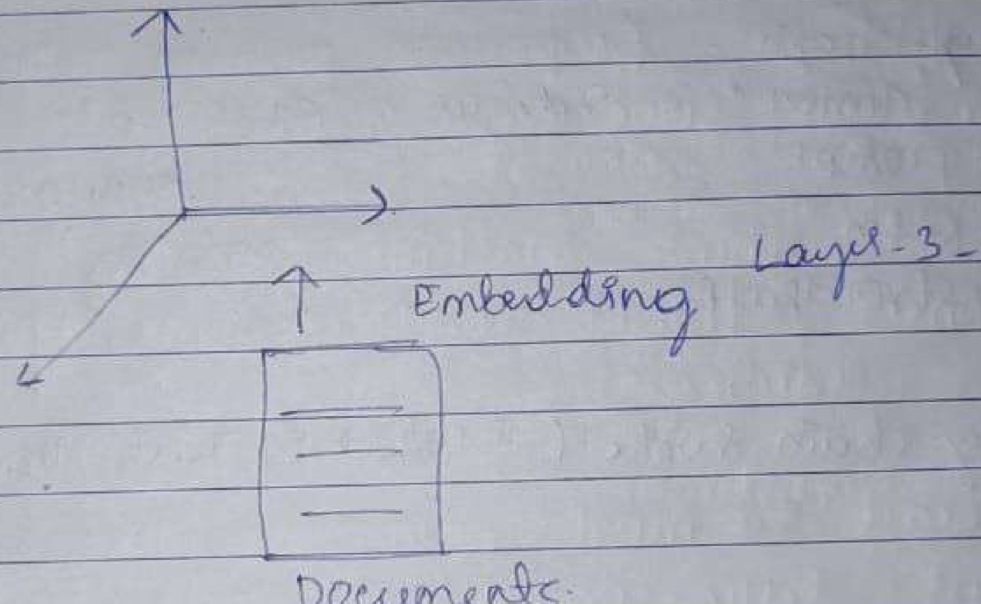


* Comparing Vectors.
(Embeddings).

* Take documents
Loading, Splitting and Embedding.

* Index makes documents Easy to retrieve

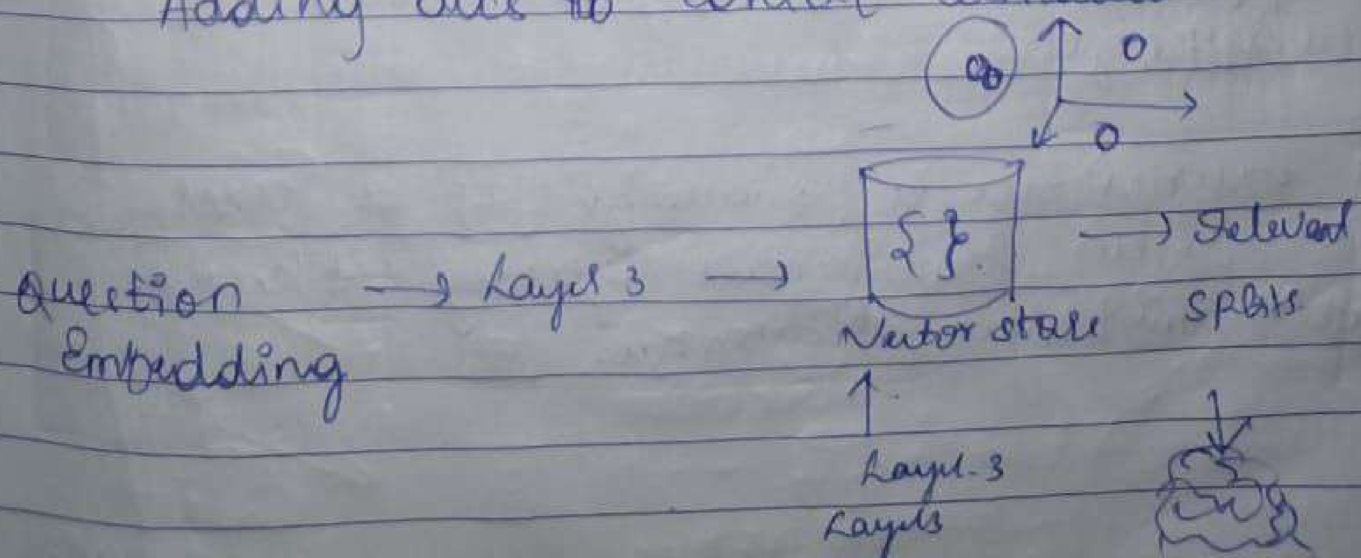
* Retrieval powered via similarity search



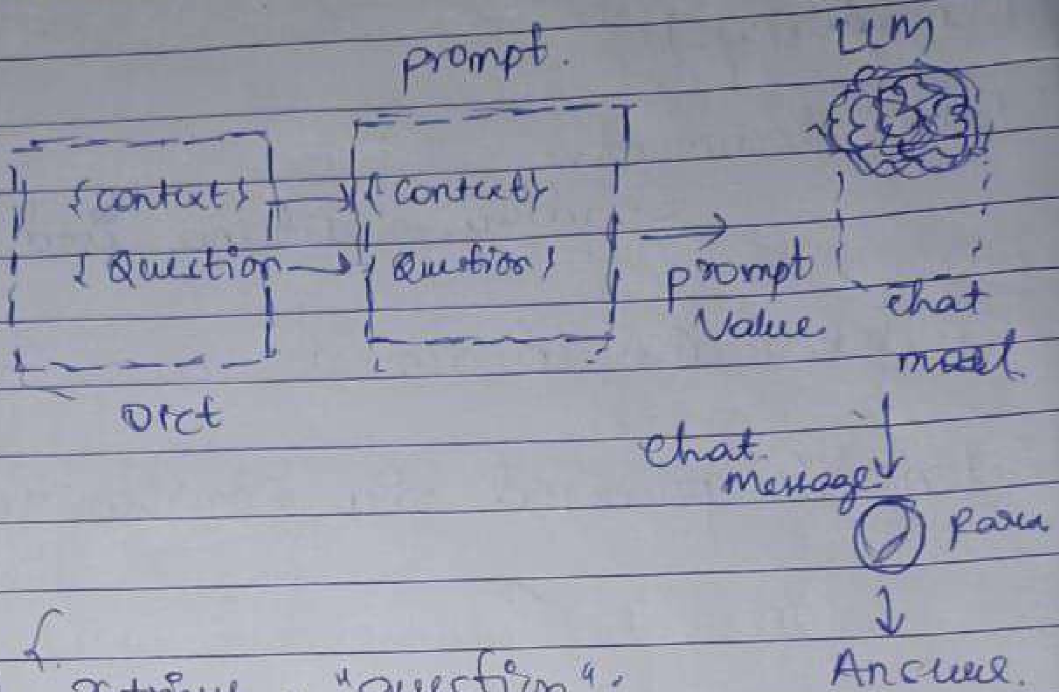
* Langchain has Many Integration

* Generation.

Adding docs to context window



connecting retrieval with LLMs via prompt.




```
* rag-chain = {
  { "context": retriever, "question":
                                   RunnablePassthrough
                                                   ()}.
  | prompt
  | llm
  | StrOutputParser()
  }
* rag-chain.invoke ("What is Task Decomposition")
```

---

* Query Translation
  (Multi query).

Semantic search on Embeddings is hard to get right Embedding long document is a challenge.

User queries are a challenge, if a user provides an ambiguous query they'll get ambitious challenges

General approaches to transform questions.

More abstraction

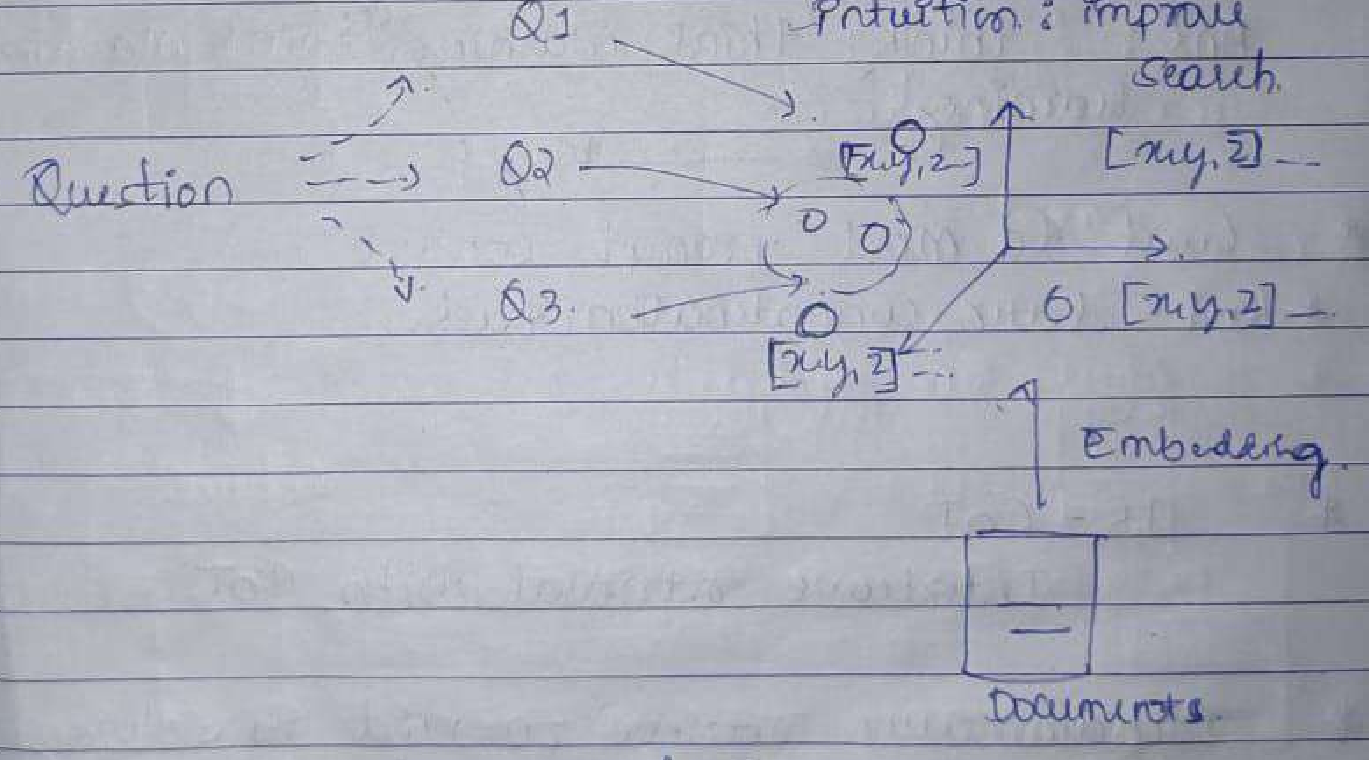
step-back question. (step-back prompting).

Question --→ Rewritten (RAG-fusion, Multi-Query).

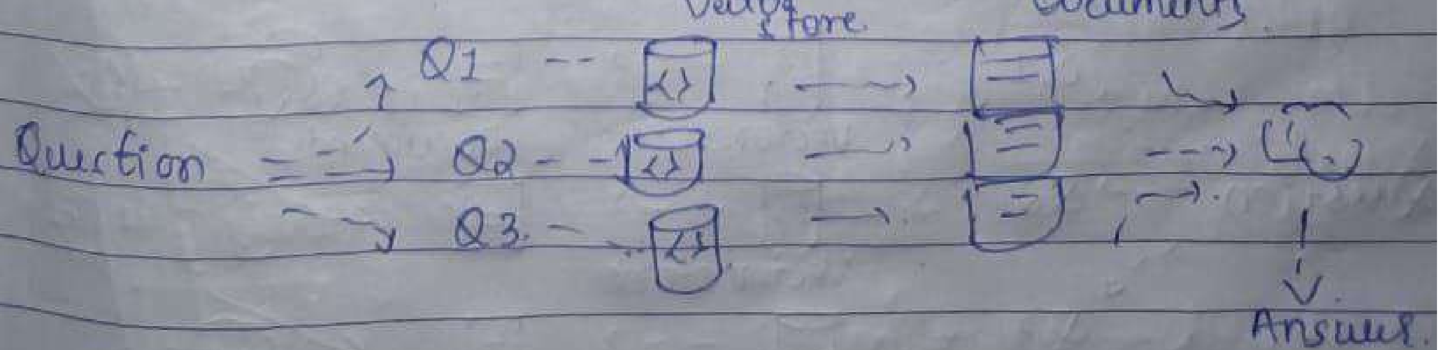
sub-question (Least-to-Most)

Less abstraction

* Transform a question into multiple perspective.



* Use this with Parallelized Retrieval



* Query Translation
RAG-Fusion.

* Intuition. → Improve search.
→ produce consolidated ranking

* Decomposition

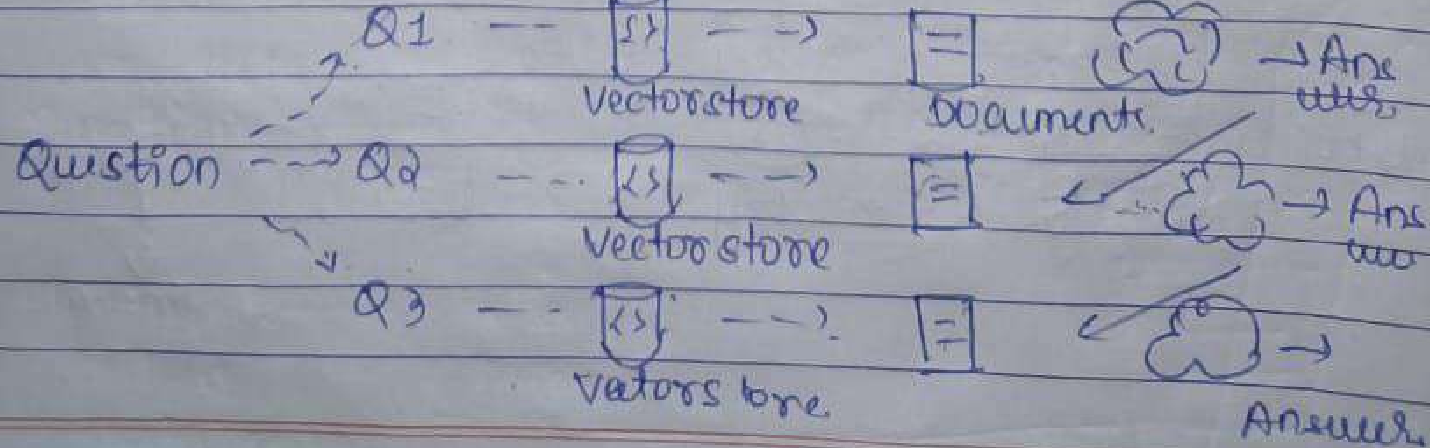
Least to Most :
Decompose problem into sub problems, solve sequentially

Q: "think, Machine, Learning"
Ans : 'think, "think machine", "think machine learning"'
key.

* Least to most prompt context
* last-letter concatination task.

* IR-CoT
Interleave retrieval with CoT.

* Dynamically retrieve to aid in solving the subproblems.



* Template
  (Question + Context)

* step-back
  prompting   Independent retrieval

* Main idea is to take the question and modify in the way that Enhances the retriever.

* different approach suggested by Google where try to ask more abstract questions

* Few shot prompt to produce more abstract step-back question.

  * normal-context

  * stepback-context

* HyDE

$$sim(q,d) = \langle Enc_{q}(q), Enc_{d}(d) \rangle = \langle v_q, v_d \rangle$$

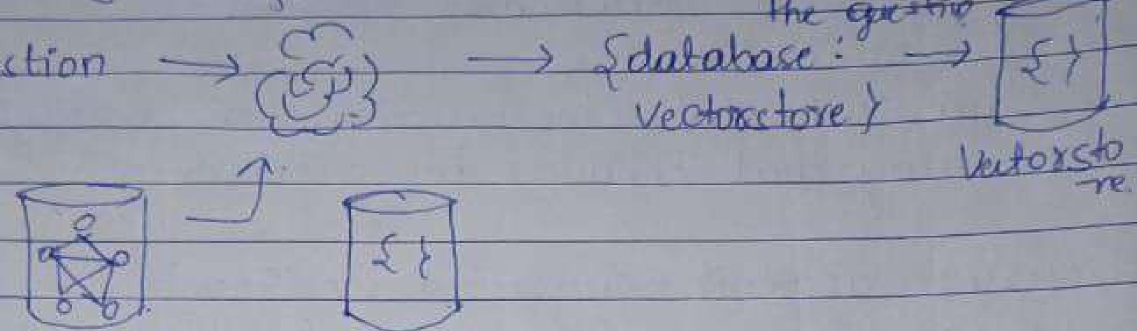
* Translating raw questions into hypothetical documents and that are better suited for retrieval

↑ Taking question & route to the appropriate data source

* Routing → Critical process that
(Logical + semantic) directs user queries to the most appropriate data source.

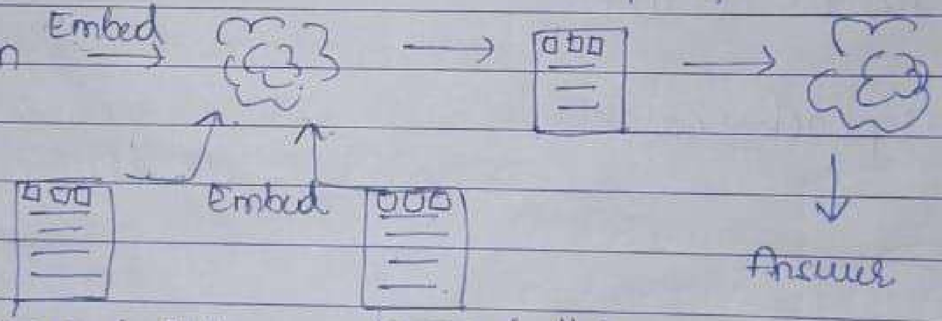
* Logical Routing

LLM for classification

Question →  → {database: vectorstore} → Vectorstore

Available retriever.

* Semantic routing

Question $\xrightarrow{\text{Embed}}$ 

* LLM with structured output (object). Apply parser

Bind function to LLM, LLM calls function. LLM returns a JSON string

Question → LLM → {'datasource': 'vectorstore'}

structured output → Function schema

datasource: [...]

Query

* Query construction

content search:
Chat langchain
Earliest publish date
: 2023-01-01

* Query structuring

Bind function
to LLM

LLM calls function.

Apply
param

videos on
chat langchain
published after
2024 →

→ {"content search
:
"chat langchain"
Earliest publish date:
"2024-01-10"}

Function schema

Vector DB metadata
Content search : [...]
Earliest publish date : [...]

* unstructured input to structured query

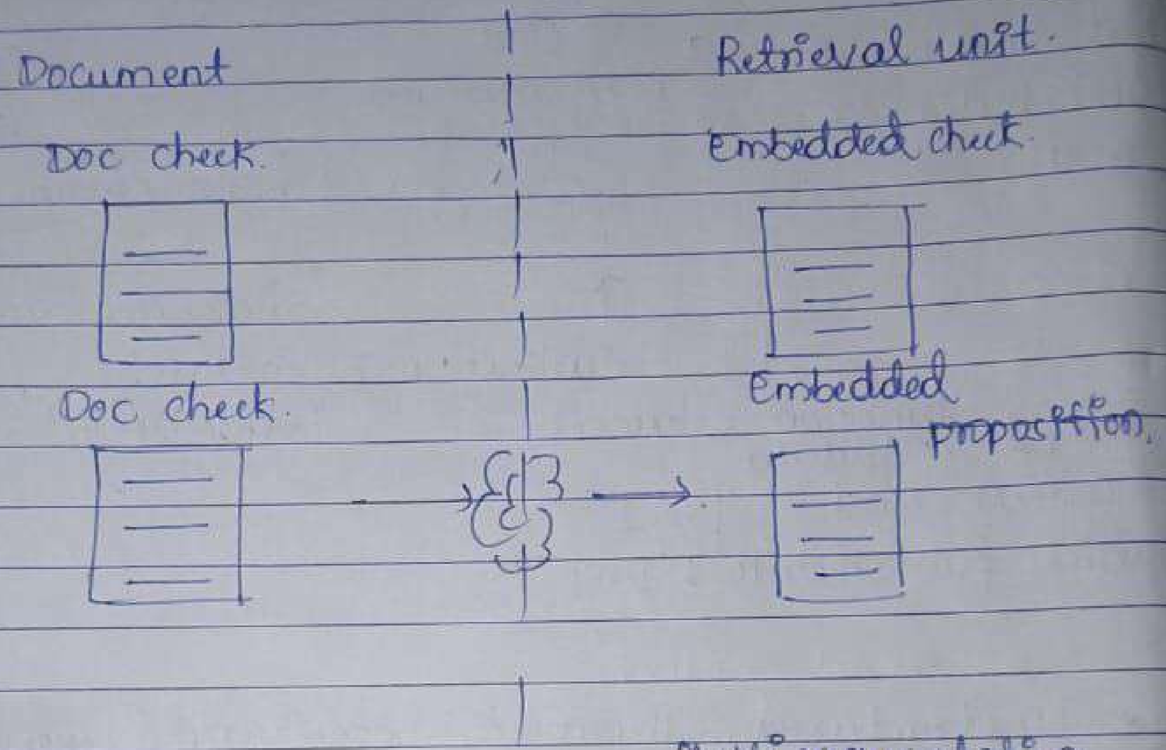
Input → " - - - - - "
↓
o/p → Context search - " "
title search - " "
Max_length-sec 300.

Meta data filtering.

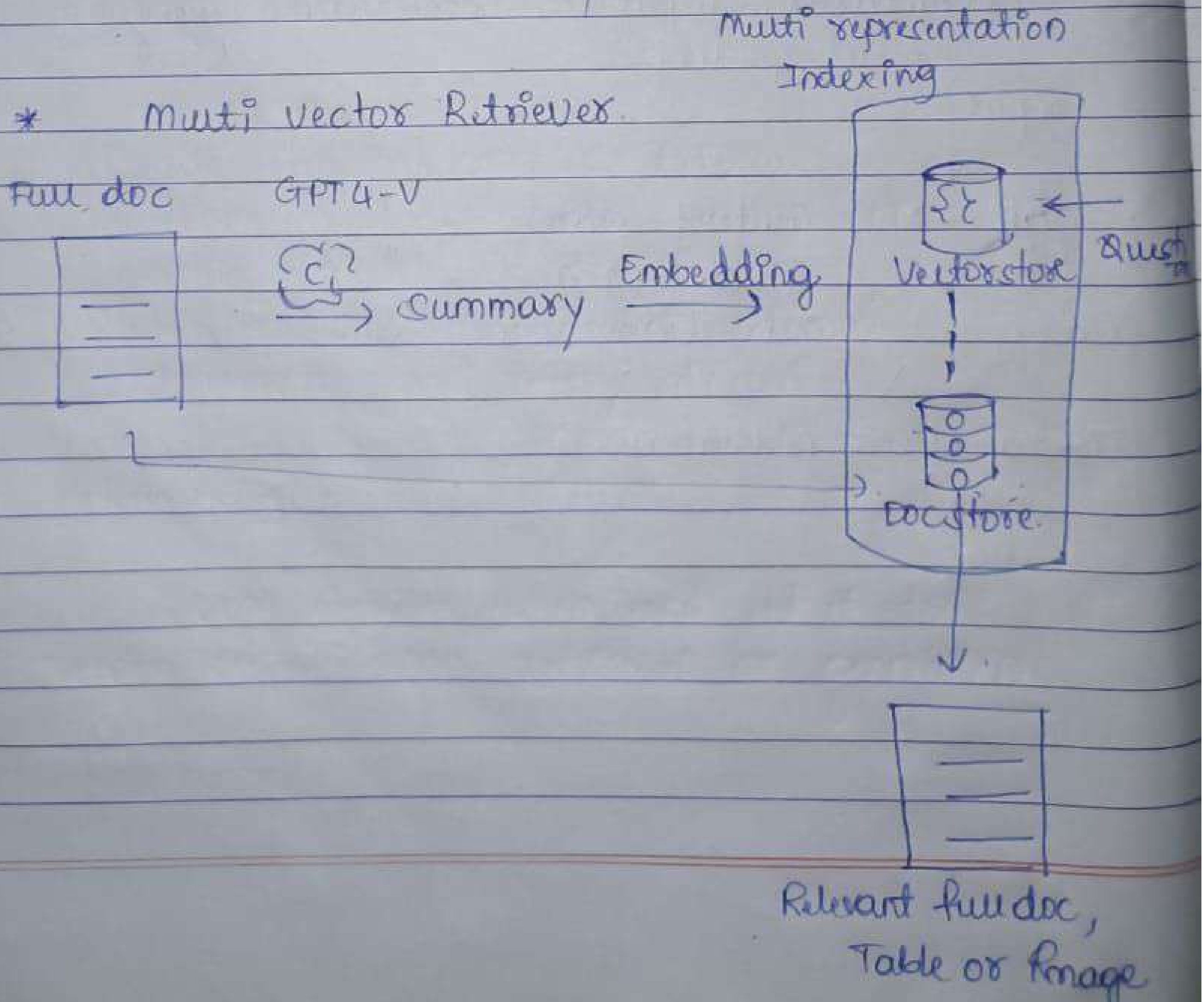
* Indexing (Multi- representation)

* for vector store.

Proposition Indexing.



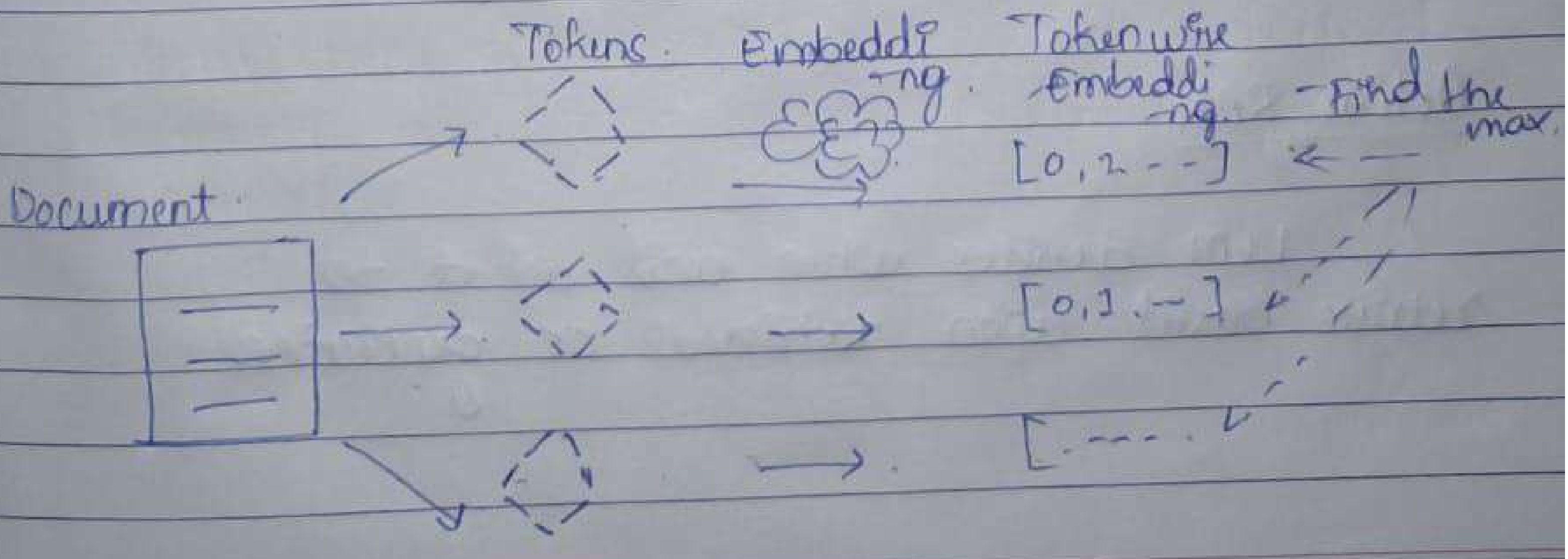
* Multi vector Retriever.



* Indexing (RAPTOR)

* Build a hierarchical index.

* Indexing (COlBERT)
Specialized Embedding



Tokenwise Embedding — Embedding — Tokens

{ }

[0,1 --] ←

[0.1, 0.2 --] ← ← Question

[0.1, 0.2 --] ←

Doc score.

Sum of max similarity of Each query Embedding to any document embedding.

Library - ragatouille.

CRAG (Corrective RAG)

Challenges

1. we can ask when to Retrieve
2. when to re-write the question for better Retrieval
3. when to discard irrelevant retrieved documents and re-try retrieval.

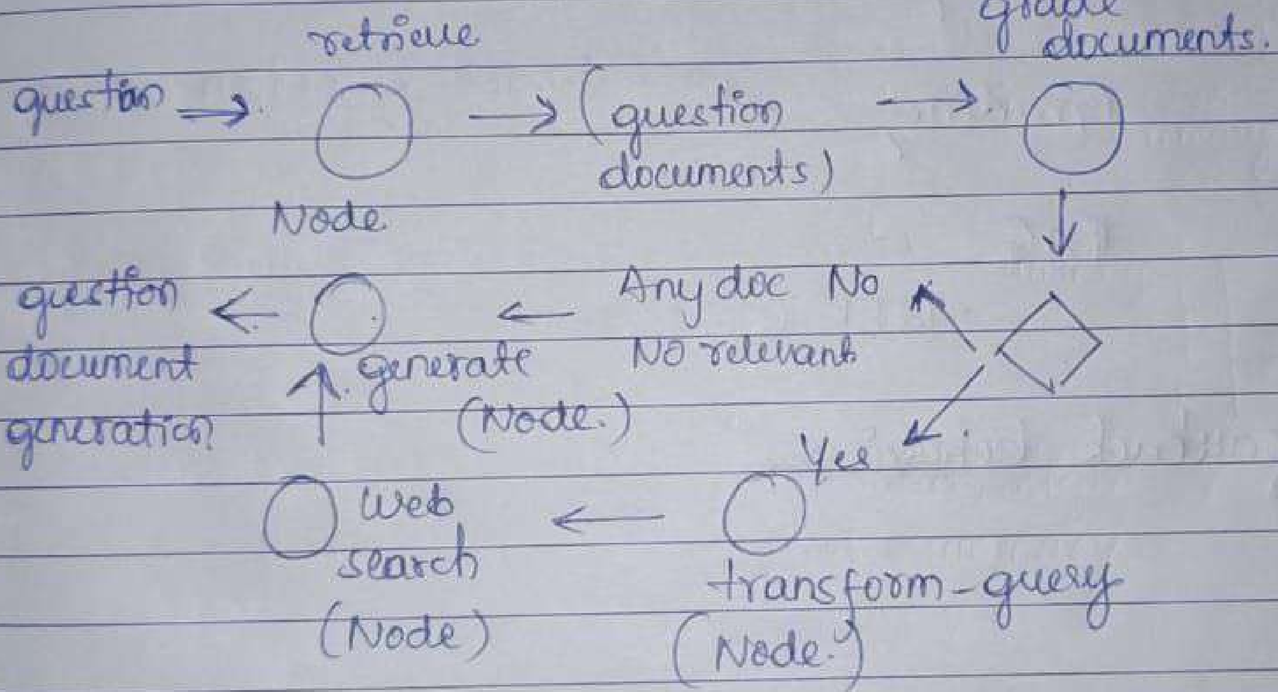
Active RAG

LLM decides when and what to retrive based upon retrieval or generation

* Levels of control

Cognitive architecture.

* State machines are good way to implement active RAG
* Knowledge Refinement stage.
* using Langchain than the Langgraph is relevant here.



* Adaptive RAG
Query Analysis
RAG + Self reflection
→ related to index → vectore store

Question – Query Analysis → current Events
→ anything else

chroma (Vector DB).

* Command - R
Routing

Grading → relevant question or not

RAG
- 128K context

* LLM fallback behaviour.