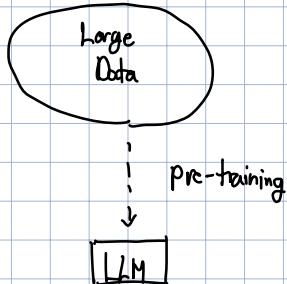


## Before RAG (traditional LLM models)



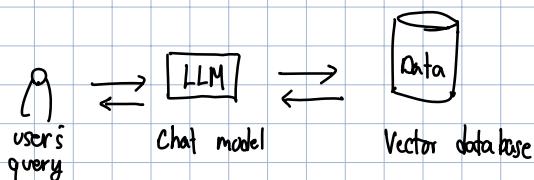
Problems :

② knowledge cut-off : If user's query required the most recent news where the pre-trained model haven't been exposed to, the model simply didn't know or "hallucinated"

③ hallucination : LLMs are "probability machines". They prioritize smoothness over accuracy. Without a source to check, they often made up fake facts.

④ Privacy & Business data : inability to teach entire new large sets of data without retraining the entire model.

## Solution : RAG : Retrieval - Augmented Generation



① Up-to minute accuracy : No need to train the LLM. only upload documents that the LLM needs to reference.

② Citations : tracking sources is possible (seeing if hallucination occurred or not). by referencing a tool online called Langsmith.

③ Cost-effective : Since retraining the model itself is not required, uses much less computational energy cost.

## RLG Implementation

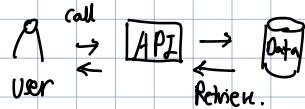
- 3 key words

- ① API
- ② Huggingface
- ③ Langchain

### ① API: Application Programming Interface

- set of rules and protocols allowing different software applications to communicate, share data, and utilize the functionality.

ex)



ex.) repo\_id = "google/gemma-2-qb-it" ← one of huggingface's free models.

### ② Huggingface

- Online opensource for pre-trained ML models.

can be used simply for 2 types of model.

- ① Embedding model
- ② LLM model.

ex.)



} has different types of models  
that performs embedding, NLP,  
Object detection, etc.

### ③ Langchain.

- Opensource Framework for the ease of application development.

3 main key highlights about Langchain.

### ① Abstraction

- Simplifies complex RAG development tasks into few lines of code.

- ex.) developing RAG : 1. Upload document  
2. Splitting text  
3. Embedding to vector  
4. Saving to Vector Data Base.

↳ traditionally, this would take many lines of code, but using langchain tools, it makes it much quicker.

### ② Standardization

- creates a "standard shape" for different AI models.

#### LLM sources

- Chat GPT
- Grog
- Gemini
- Claude
- ⋮

Standardization  
→

#### Chat model

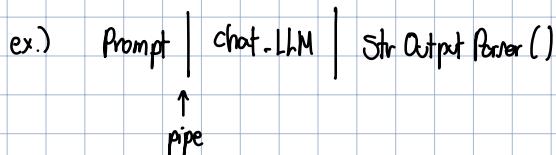
- Chat Anthropic
- Chat Grog
- Chat Google Generative AI.
- ⋮

- Since most LLMs have different endpoints, APIs, changing one model would require many changes in the code.

↳ Langchain allows to change single line of code

### ③ Chaining

- Most Components are designed with input → output structure and using Langchain can "chain" multiple sources under few/ single of code.

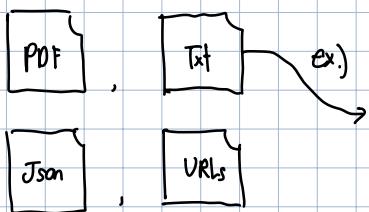


## Implementation Flow diagram -

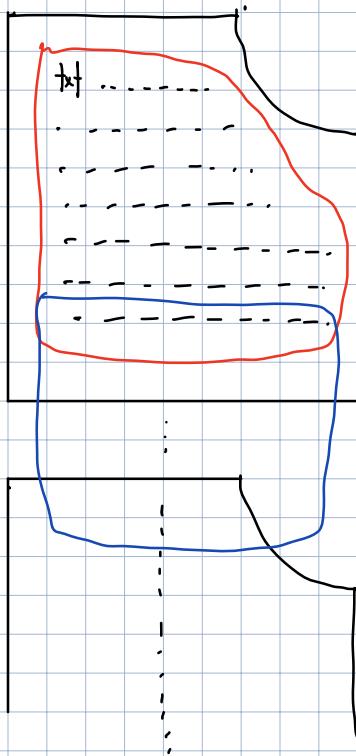
### ② Indexing.

- a.) Upload documents
- b.) Split text within the document
- c.) Embed each splits
- d.) Save to a vector database.

a.) Load.



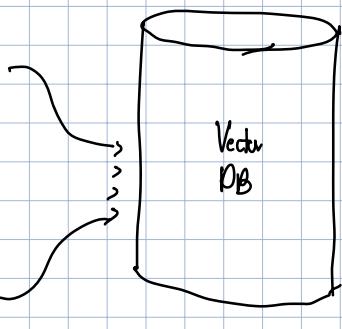
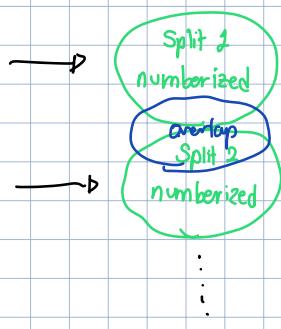
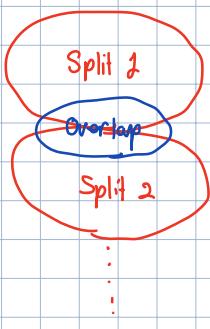
b.) Split text within document



(can use free LLM from HF)

c.) Embed splits.

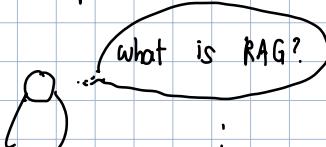
d.) Store to Vector database



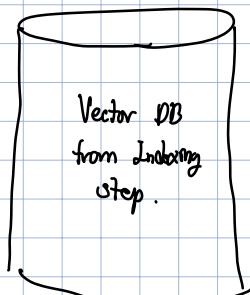
## ② Query

- a.) Input user question
- b.) Retrieve from Vector DB
- c.) Send step a), b) into LLM
- d.) LLM outputs answer.

a.) Input user question



b.) Retrieve related docs from DB

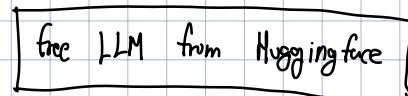


Finds related documents  
ex.) search-keywords  $\{k=2\}$

Find 2 documents / splits



c.) Send a.), b.) into LLM.



d.) LLM outputs answer using Str OutputParser()

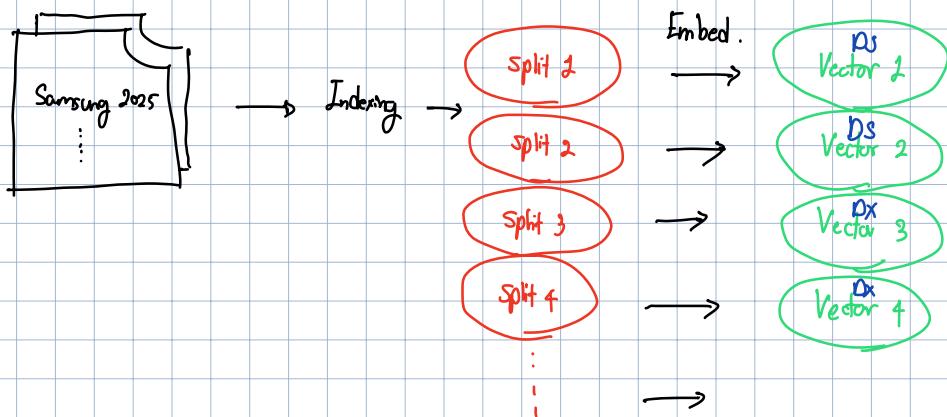


↓  
answer      AIG is Retrieval Augmented Generation...

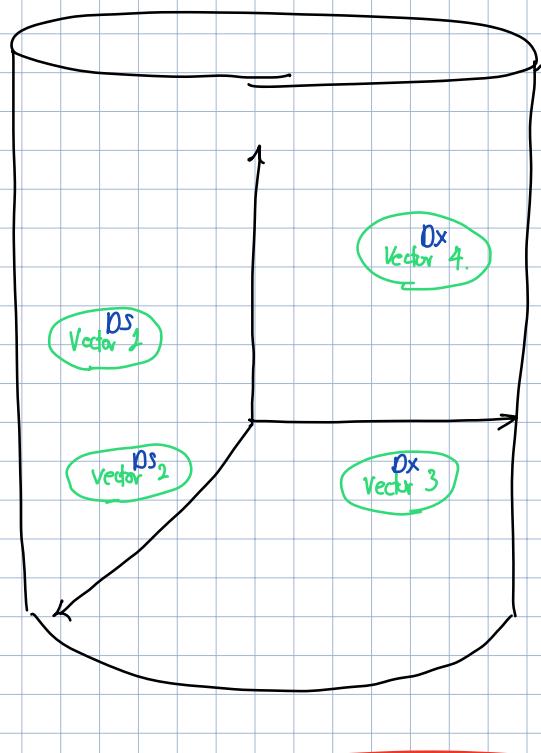
### Visualized simple example.

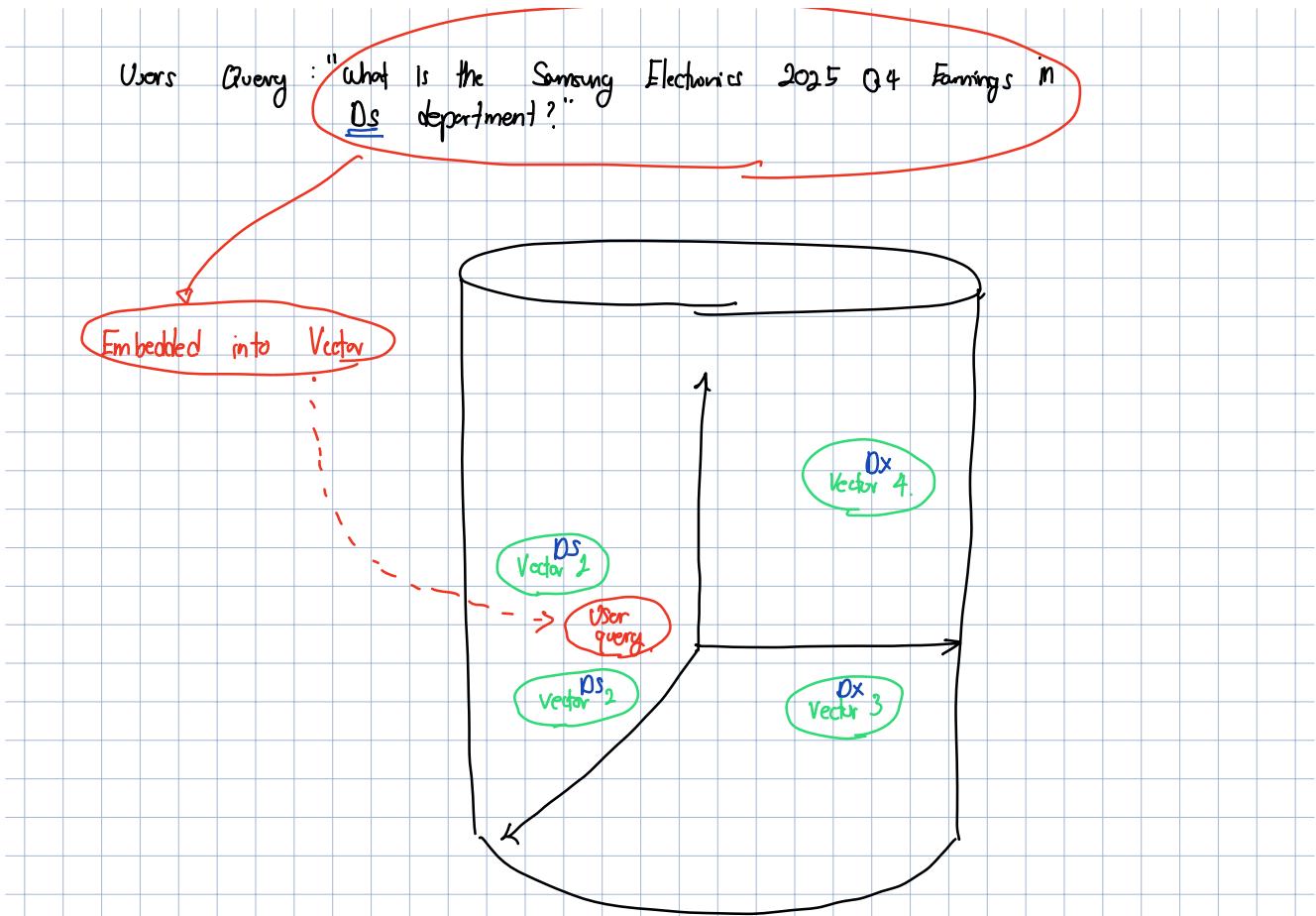
Documents : ① "Samsung 2025 Q4 earnings call PDF"  
 ② "Samsung 2025 Q4 earnings call Presentation"

① ② Documents.

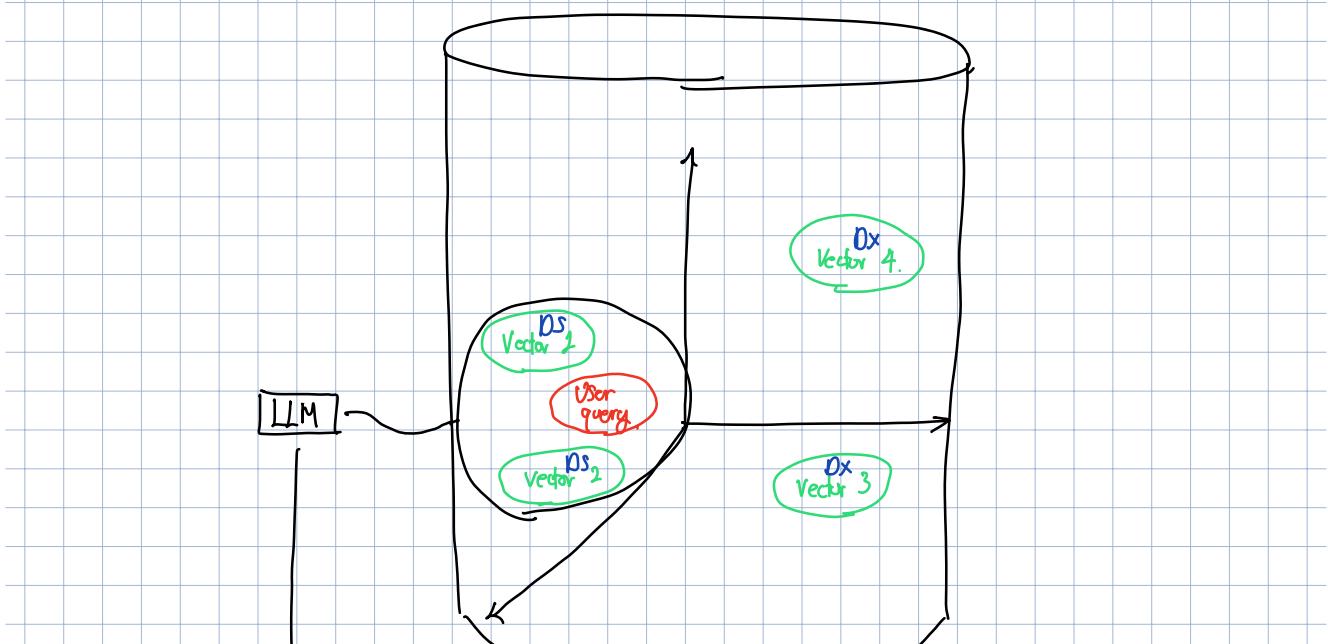


Inside Vector DB





- LLM finds the two closest neighbors (splits) with regards to where the user's query is located and fetches them.



↓  
Summarizes in natural language  
and returns answer