* RAGs (Retrieval Augmented Generation):

- LLMs face significant limitations especially in domain-specific or knowledge-intensive tasks, notably producing "hallucinations".
- RAGs overcome these challenges by retrieving relevant document chunks from external knowledge base through semantic similarity calculation.
- It is a key technology in advancing chatbots and enhancing the suitability of LLMs for real world applications.
- RAG has three core parts:
  - Retrieval
  - Generation
  - Augmentation
- Based on development of RAG it is of 3 types: Naive RAG, Advanced RAG & Modular RAG.

- LLMs are great for generating text, however that does not guarantee that the text is also factually accurate, thus RAGs come in.

* Naive RAG: (Retrieve Read)

→ INDEXING:
cleaning & extracting raw data (from HTML, PDF, Word, Markdown) → convert to plain text format → these chunks are then encoded to vector representations using embeddings & database
(crucial to enable similarity searches.

→ RETRIEVAL:
gets input user query → converts to vector rep. using same encoder of indexing → computes similarity score → prioritizes & retrieves top K chunks that demonstrate greatest similarity

→ GENERATION:
posed query & selected documents → synthesised into a coherent prompt to which LLM is tasked

* Difficulties / Disadvantages of Naive RAG:

- Retrieval: struggle with precision & recall
- Generation: may still face hallucination
- Augmentation: integration may result in disjointed or incoherent outputs

- Moreover, concern that generations models overly rely on retrieved info, thus not adding insightful or synthesised information.

* Advanced RAG: (Retrieve Read)

- Specific improvements to overcome the limitations of Naive RAG.

- Enhancing RETRIEVAL: (optimise indexing structure & og query)
  - → pre - retrieval
  - → post - retrieval (crucial to integrate retrieval with query).

- Enhancing INDEXING:
  - → sliding window approach
  - → fine-grained segmentation
  - → incorporation of metadata

* Modular RAG: (Rewrite Retrieve Read)

Many modifications that include:

1) New Modules:

- Search Modules: enable direct search across various data sources like search engines, databases & knowledge graphs.

- RAG-fusion addresses search limitations by employing a multi query strategy that expands user query into diverse perspectives, utilising parallel vector searches and intelligent re-ranking

- Memory Module: leverages LLM's memory to guide retrieval, thus aligning text closely with data distribution through iterative self enhancement.

- Predict Module aims to reduce the redundancy & noise by generating content directly through the LLM.

- Task Adapter : tailors RAG to various downstream tasks, automating prompt retrieval for zero shot inputs
  → tasks applied on a trained model.

- The distinguishing property of Modular RAG is that it allows module substitution or reconfiguration to address specific challenges.
  → leveraging LLM's capability of refined retrieval
  → RAG system can more easily integrate with other technologies (such as fine tuning or reinforcement learning).

* RAGs vs	Fine Tuning	vs Prompt Engg
like providing a model with a tailored textbook for information retrieval	comparable to a student internalising knowledge overtime, good for scenarios involving replication.	leverages model's with min necessity for external knowledge.

→ not mutually exclusive, can complement each other
→ RAGs outperform FT on various knowledge intensive tasks across various topics.
→ above classification based off:
external knowledge & model adaption requirements.

A * Retrieval

1. Retrieval Source :

1. Data Structure :

→ Unstructured data (text) Wikipedia.

Text2 sql, Table QA → Semi - Structured data (pdf)

→ Structured data (knowledge graphs)

→ Graph neural network (GNN), LLMs & RAG integration.

2. Retrieval Granularity :

→ Coarse Grained : can provide more relevant info for the problem but may also contain redundant content which could distract retriever & language models in downstream tasks

→ Fine Grained : increases the burden of retrieval & does not guarantee semantic integrity & meeting req. knowledge

Token, Phrase, Sentence, Proposition, Chunk, Document

Fine to Coarse

Entity, Triplet, Subgraph

→ Granularity of text & knowledge graph

B * INDEXING Optimisation :

→ determines whether correct context can be obtained.

1. Chunking Strategy
→ leads to truncation of sentences, prompting the optimisation of a recursive splits & sliding window methods, then enabling layered retrieval by merging globally related info.
→ still, not the perfect balance.
→ long chunks → more data more noise
small chunks → less data less noise.

2. Meta Data Attachments:
→ enriching with metadata info such as page no, file name, author, category, timestamp, limiting scope of retrieval.
→ can be artificially constructed, for eg: adding summaries of paragraph.

3. Structural Index:
→ hierarchical structure for documents
↳ parent-child relationships.
→ knowledge Graph Index:
Use KGs where nodes (text) and edges (show relation of any sort among the text)
→ Thus, maintains consistency, and describes the connection b/w different entities & concepts, markedly reducing potential illusions

C* Query Optimisation:
1. Query Expansion
2→ Multi Query
3→ Sub Query
→ Chain of Verification (CoVe)

} enriches content of query & enable to fully answer the asked qs.

2. Query Transformation:
→ to retrieve chunks based on transformed query over user query as they may not always be optimal. Thus, we can either prompt the LLM to rewrite or use another smaller model.

3. Query Routing:
→ Metadata routing: extracting keyword & apply filter on metadata.
→ Semantic Router: leveraging semantic info of the query.
→ A hybrid approach can also be employed.

D * EMBEDDINGS
1. Mix/Hybrid Retrieval
2. Fine-Tuning Embedding Model (especially when used for specific domains such as health, legal, etc).
→ capture different relevance features and can benefit from each other by leveraging complementary relevance info.

E * Adapter:
Use pretrained models to help bridge the gap which could not achieved by finetuning.

IV GENERATION.

A* Context Curation
1. Re Ranking (Assigning weightage to retrieved)
2. Context Selection / Compression
(Small Language Model (SLM) & LLM can be used in tandem to remove unnecessary tokens).

B* LLM Fineturning:
→ allows the model to adjust to data & learn more.
→ Fineturning of LLM & Retriever can also be coordinated.

V AUGMENTATION PROCESS.

In order to present the best generated text some augmentation is done.

→ Iterative Retrieval
→ Recursive Retrieval
→ Adaptive Retrieval

Iterative: enhance robustness of subsequent answer generation by offering additional contextual references through multiple iterations. After each round, retrieved info is used to refine search in next round.

Recursive: To improve depth & relevance of search results. Retrieves info, processes & then uses this output as new input, creating a layered/hierarchical process. (~~test~~ nested loops)

LVLM → Large Vision language Model
(generate captions)
Whisper → audio to text.

Adaptive : adjusts its strategy based on the input or task. Dynamically changes retrieval approach, such as how it queries the database or which sources it uses.

* RAGs Use Cases:
→ customer support Q & A chatbot
→ Email chain analysis,
→ company internal documentation
→ textbook Q & A

* Why RUN RAG locally?
→ Privacy, Speed, cost

MULTIMODAL RAG

A multimodal RAG system should utilize both visual & language content in a video including:
1. Utilize the written text in the video.
2. Utilize pure visual info in the video.
3. Utilize the info being spoken about in the video.
4. Be able to hold multi-turn conversation with content of the video.

For similarity search: COSINE SIMILARITY, as it is computationally cheap & used widely.

Multimodal

① Embeddings ⇒ Bridge Transformer.

② Video Processing ⇒ ① Transformer (audio + image)
(no transcript all.) ② Whisper model
(only image) ③ LVLM.

③ Database for Embeddings:
→ storing the multimodal embeddings using Lance DB (data can be private).

④ Retrieval ⇒ use similarity search.

⑤ LVLM ⇒ take as input both vision & text to generate textual outputs.