Novel Trends: Low Rank, RAGs.

Roberto Basili, Danilo Croce
Deep Learning, 2023/2024

Outline

- How to fine tune Large Scale Decoder-only architectures
  - Scale problems
  - Adapters for LLMs
- Alignment through External Sources
- Retrieval Augmented LLMs
  - RAG: the architecture
- Applications of RAGs
  - Vector Databases
  - Knowledge Distillation

How to train a large scale encoder?

Challenge: 16GB GPU resources

ChatGPT's resources: 10-30.000 GPUs

1xTesla T4

Render Config	
Shading Units:	2560
TMUs:	160
ROPs:	64
SM Count:	40
Tensor Cores:	320
RT Cores:	40
L1 Cache:	64 KB (per SM)
L2 Cache:	4 MB

Theoretical Performance	
Pixel Rate:	101.8 GPixel/s
Texture Rate:	254.4 GTexel/s
FP16 (half):	65.13 TFLOPS (8:1)
FP32 (float):	8.141 TFLOPS
FP64 (double):	254.4 GFLOPS (1:32)

Scale: impact

ChatGPT uses 10000 graphic cards and 285000 processor chips to process the data.

- How to **cool** them?
- Currently, to cool the systems used for GPT, **water is being used by Microsoft and OpenAI** (the method of cooling is called "*evaporative cooling*"). As reported in the research paper [1], Microsoft's state-of-the-art data center in the US can easily consume 700,000 litres of clean fresh water (potable water).
- To compare the metrics of water usage, the same amount of water can be used to
  - manufacture 370 BMW cars
  - manufacture 320 Tesla Evs
  - could quench the thirst of 2,30,000 people (considering an avg of 3 litres of water drunk in a day by a person) in one single day

Pengfei Li et al., 2023, Making AI Less "Thirsty": Uncovering and Addressing the Secret Water Footprint of AI Models.

Pretrainable Adapters: the idea

$$h_x^{(A)} = \mathrm{MLP}_u(g_2(\mathrm{Enc}(g_1(\mathrm{MLP}_d(h_x))))). \quad (1)$$

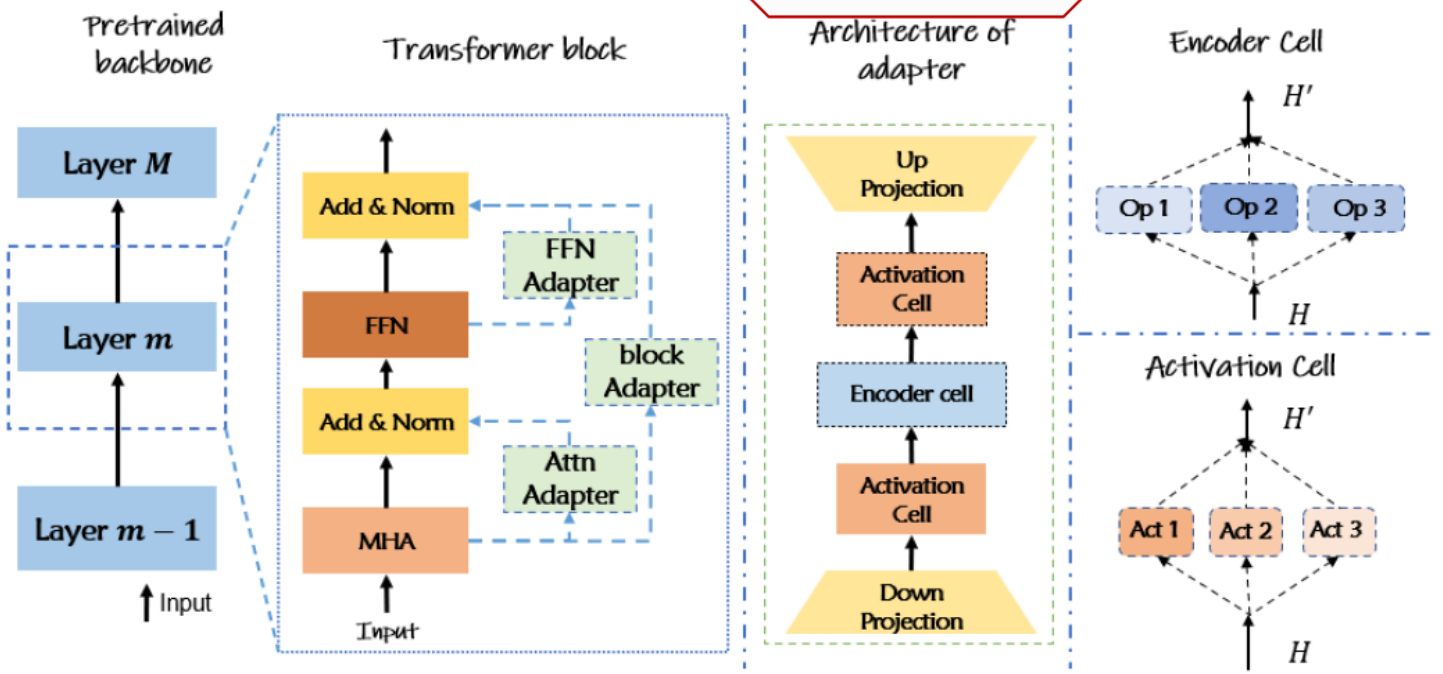


Figure 2: The overall framework of our Learned Adapter.

What are the optimal architectures for adapters?

Architecture Search methods

- Architectural Choices:
  - Which Activation function?
  - Which Encoder?
  - Which Adapter Placement?
- DART algorithm for architecture search
  - Define the search options O and design an *hypernetwork* where layer weights and architectural parameters O are trainable in an end-to-end fashion
  - Extracts the final sub-network a posteriori by selecting the best operation on each edge and dropping the lower-score operations
  - Retrain from scratch the final sub-network on the original train set with randomly initialized parameters.

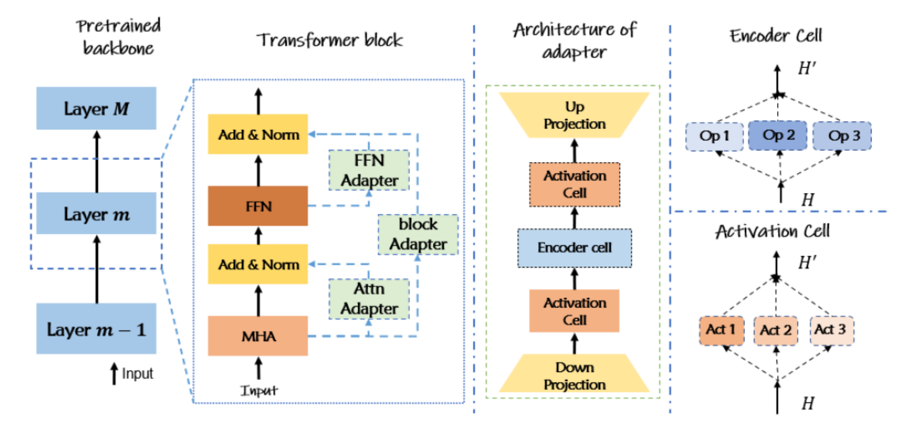


Figure 2: The overall framework of our Learned Adapter.

Architecture Search: Outcomes

Layer index	Adapter placement	Activation g_1	Activation g_2	Encoder operation 1
1	FFN	elu	null_act	mha_8
2	-	-	-	-
3	-	-	-	-
4	FFN	elu	null_act	mha_2
5	Block	tanh	null_act	mha_2
6	FFN	gelu_new	leaky_relu	conv_3
7	FFN	null_act	tanh	mha_2
8	-	-	-	-
9	Attn	elu	relu	conv_3
10	-	-	-	-
11	Attn	gelu_new	relu	conv_1
12	Attn	gelu_new	relu	conv_3
13	Block	relu	leaky_relu	conv_1
14	Attn	swish	relu	conv_3
15	FFN	leaky_relu	relu	conv_3
16	Block	leaky_relu	relu	conv_5
17	FFN	leaky_relu	relu	conv_1
18	Attn	leaky_relu	null_act	skip_connect
19	FFN	relu	relu	conv_3
20	FFN	gelu_new	null_act	conv_3
21	-	-	-	-
22	Block	tanh	null_act	mha_8
23	Attn	tanh	null_act	conv_3
24	FFN	tanh	null_act	mha_2

Table 9: The learned adapter architectures on the RTE task when the PTM backbone is RoBERTa-large. If an adapter's architecture contains only "-", it means our Learned Adapter framework choose the **null** encoder operation, and equivalently, dropping this layer's adapter.

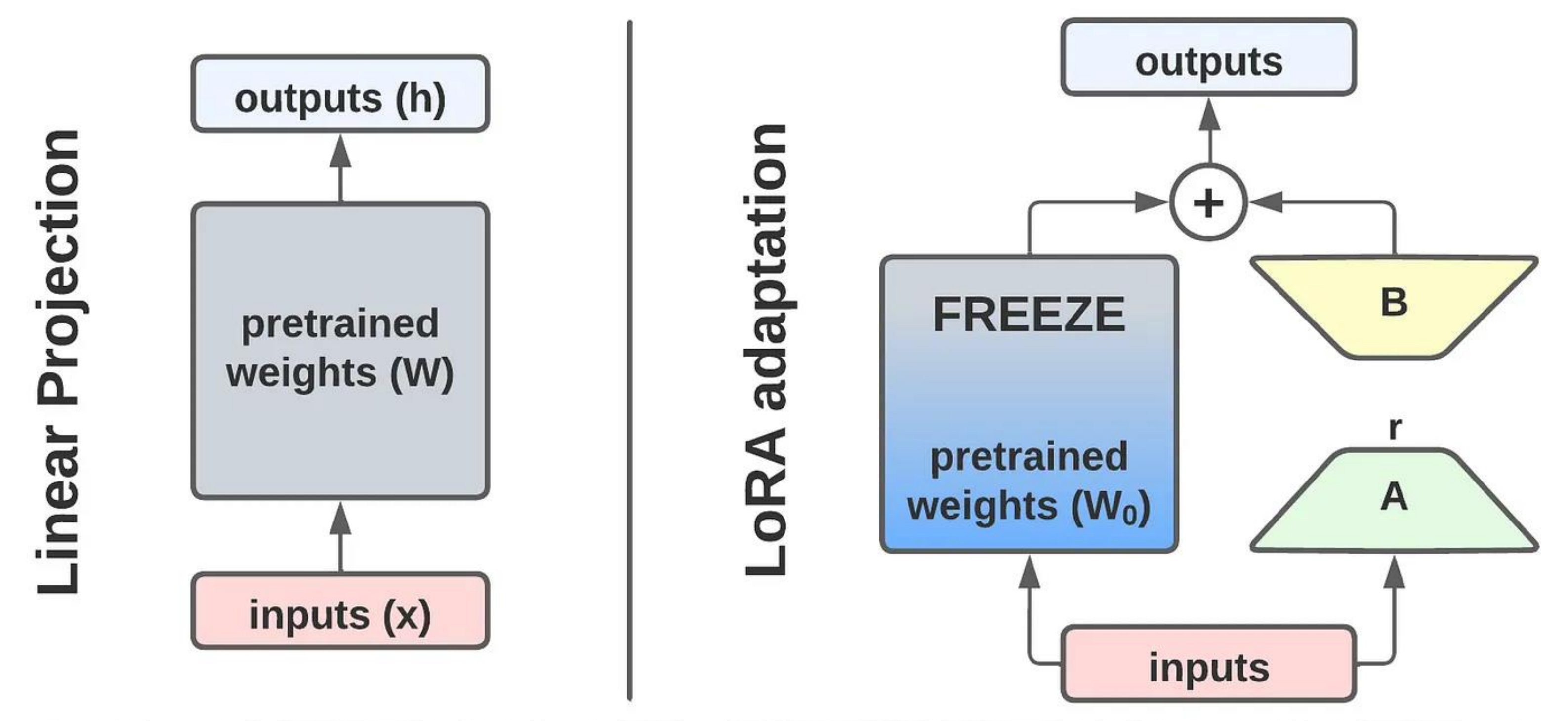
Low Rank Adaptation: Motivations

- Fine-tuning is computationally challenging when applied to large pre-trained models as it involves the adjustment of millions of parameters. Effective traditional fine-tuning demands huge computational resources and time, so that it has a limited applicability to model adaption for specific tasks.

- In traditional fine-tuning, the adjustment involves altering the original weight matrix ***W*** of the network. The changes made to ***W*** during fine-tuning can be collectively represented by ***ΔW***, such that the updated weights can be expressed as ***W*+*ΔW***

- As *intrinsic rank hypothesis* may suggest, the significant changes (i.e. ***ΔW***) to the neural network can be captured just relying on a small lower-dimensional representation.

Adapters: the idea

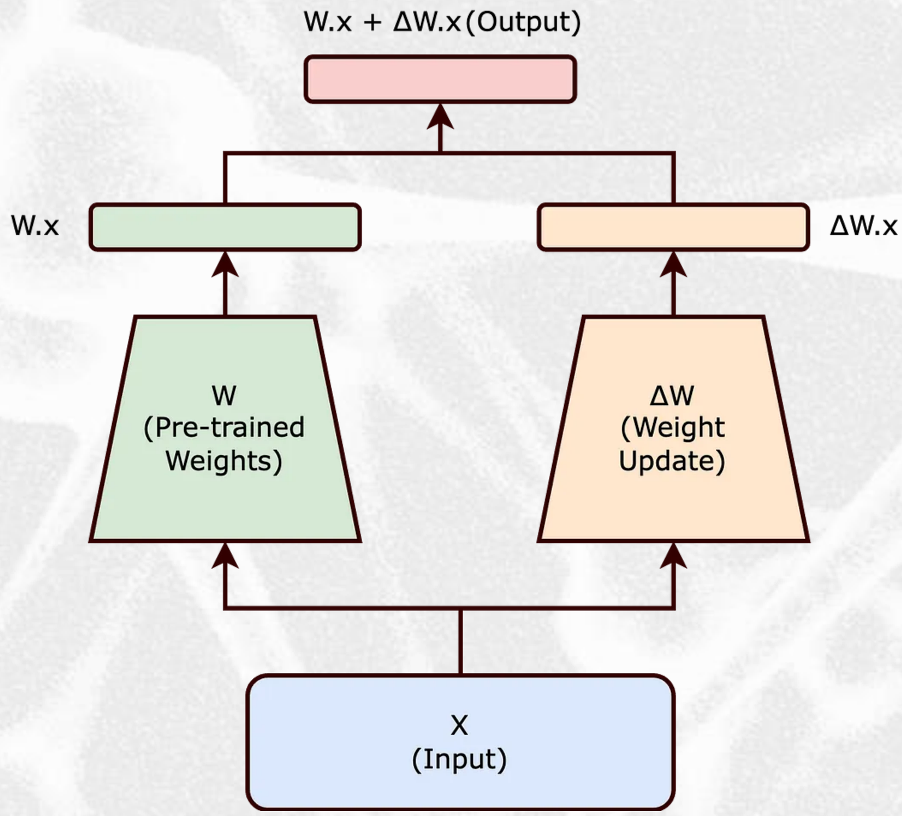


LoRA: aims

- A pre-trained model can be shared and used to build many small LoRA modules for different tasks.
  - The shared model can be freezed and efficient switching among tasks is achieved by replacing the matrices A and B, reducing the storage requirement and task-switching overhead significantly.
- LoRA makes training more efficient and lowers the hardware barrier to entry by up to 3 times when using adaptive optimizers since we do not need to calculate the gradients or maintain the optimizer states for most parameters.
  - Only the injected, much smaller low-rank matrices are optimized.
- A simple linear design allows us to merge the trainable matrices with the frozen weights when deployed, that does not introduce any inference latency compared to a fully fine-tuned model, by construction.
- LoRA is orthogonal to many prior fine-tuning methods and can be combined with many of them, such as prefix-tuning.

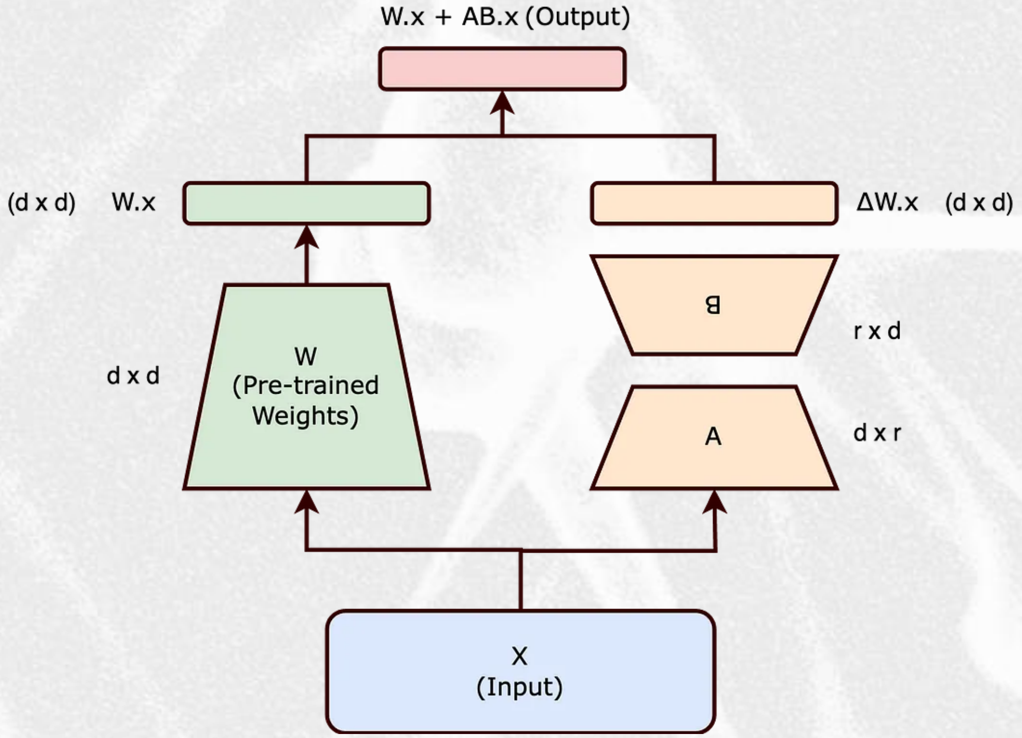
Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen (2021). LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685

Exploting implicit Low Rank



Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen (2021). LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685

Exploting implicit Low Rank (2)



Low Rank of *A* and *B* implies a rank *r* (with $r \ll d$) significantly reduces the number of trainable parameters.

If *W* is a *dxd* matrix, standard *W* updating involves d^2 parameters.

With *B* and *A* of sizes *dxr* and *rxd* respectively, the total number of parameters reduces to $2dr$, which is much smaller when $r \ll d$.

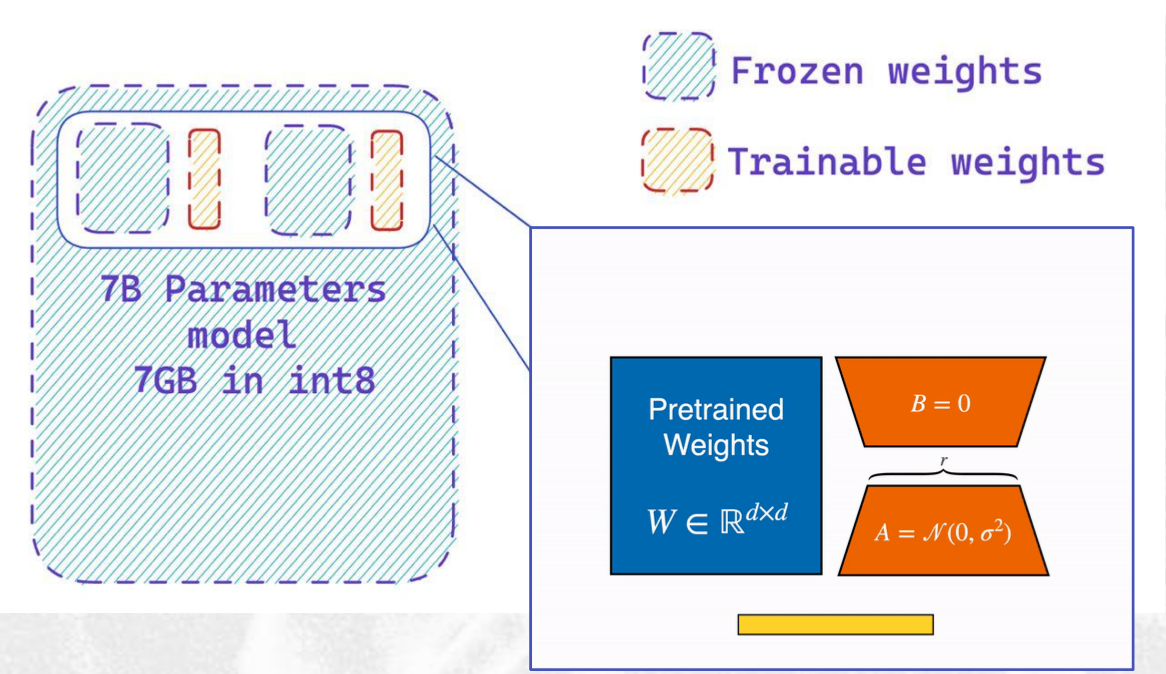
Low-Rank Adaptation (LoRA) (Hu et al., 2021)

Low Rank Adaptation (LoRA: Hu et al., 2021): create the parallel (fine-tunable) adapters as smaller matrices:

- add the adapters to the base model while keeping the base model frozen

LoRA is **NOT** learning any parameter, but the **changes in the parameters**!

$$\underbrace{W_0 + \Delta W}_{\text{Traditional FT}} = \underbrace{W_0 + BA}_{\text{LoRA FT}}$$



Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen (2021). LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685

Advantages

- When applied to very large language models, the Low-Rank Adaptation (LoRA) method largely reduces the number of trainable parameters, offering several benefits, :

1. **Reduced Memory Requirements**: LoRA decreases memory needs by lowering the number of parameters to update, aiding in the management of large-scale models.
2. **Faster Adaptation/Training**: By simplifying computational demands, LoRA accelerates the training and fine-tuning of large models for new tasks.
3. **Lower HW requirements**: LoRA’s lower parameter count enables the fine-tuning of substantial models on less powerful hardware, like modest GPUs or CPUs.
4. **Larger Scale Models**: LoRA facilitates the expansion of AI models without a corresponding increase in computational resources, making the management of growing model sizes more practical.

LoRA trends: ALoRA

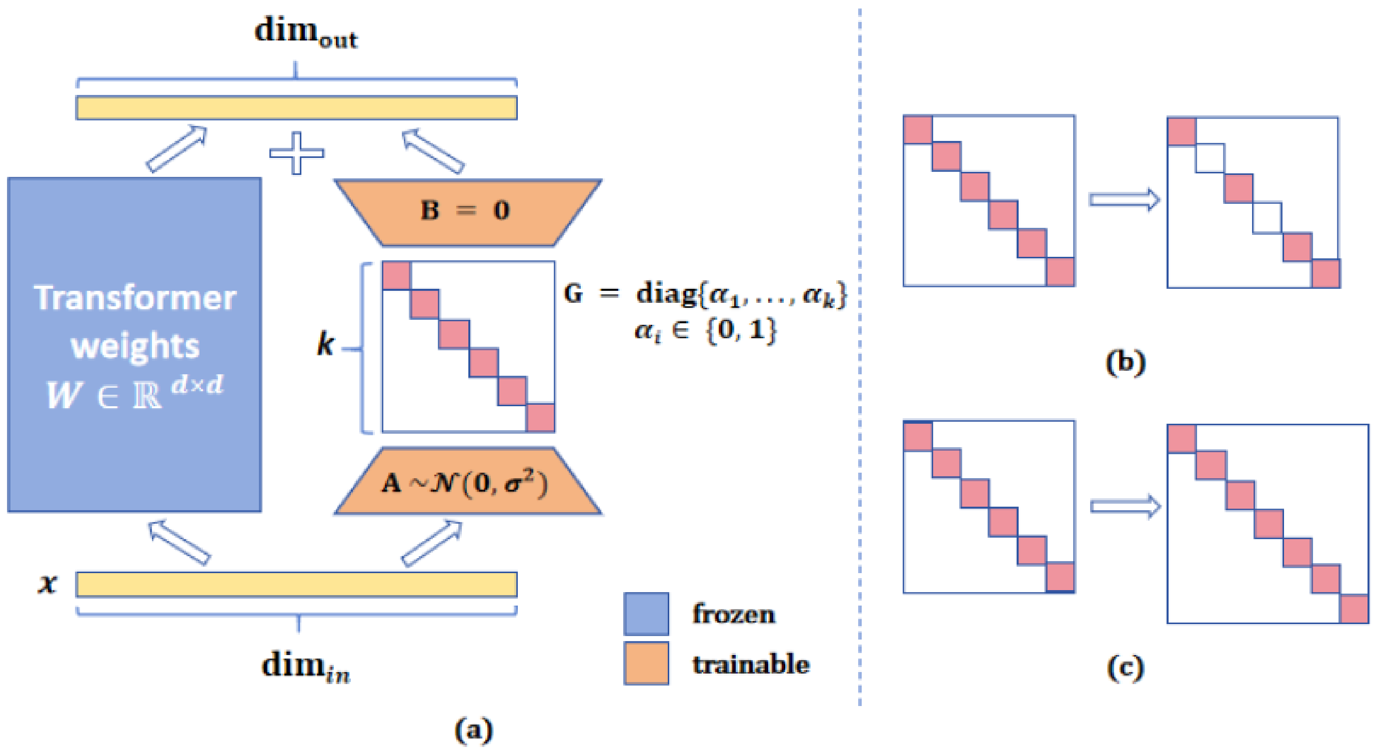


Figure 1: Schematic illustration of our ALoRA. Left (a): ALoRA follows LoRA to update the weight matrix W by fine-tuning the low-rank matrices A and B with intermediate rank k. Matrix G is a diagonal matrix where each diagonal element is the gate unit α_i for each LoRA rank $i < k$. Each α_i is set to 1 at initialization. Right upper (b): Some abundant LoRA ranks are pruned by setting the corresponding gate α_i to zeros. Right lower (c): For weight matrix W whose LoRA ranks are not pruned, we will assign additional LoRA ranks to enhance reparameterization.

ALoRA: algorithm

Algorithm 1: Workflow of ALoRA

Input: A super-network M, with R^{target} LoRA ranks uniformly distributed in modules of M;

Output: A new allocation of R^{target} LoRA ranks.

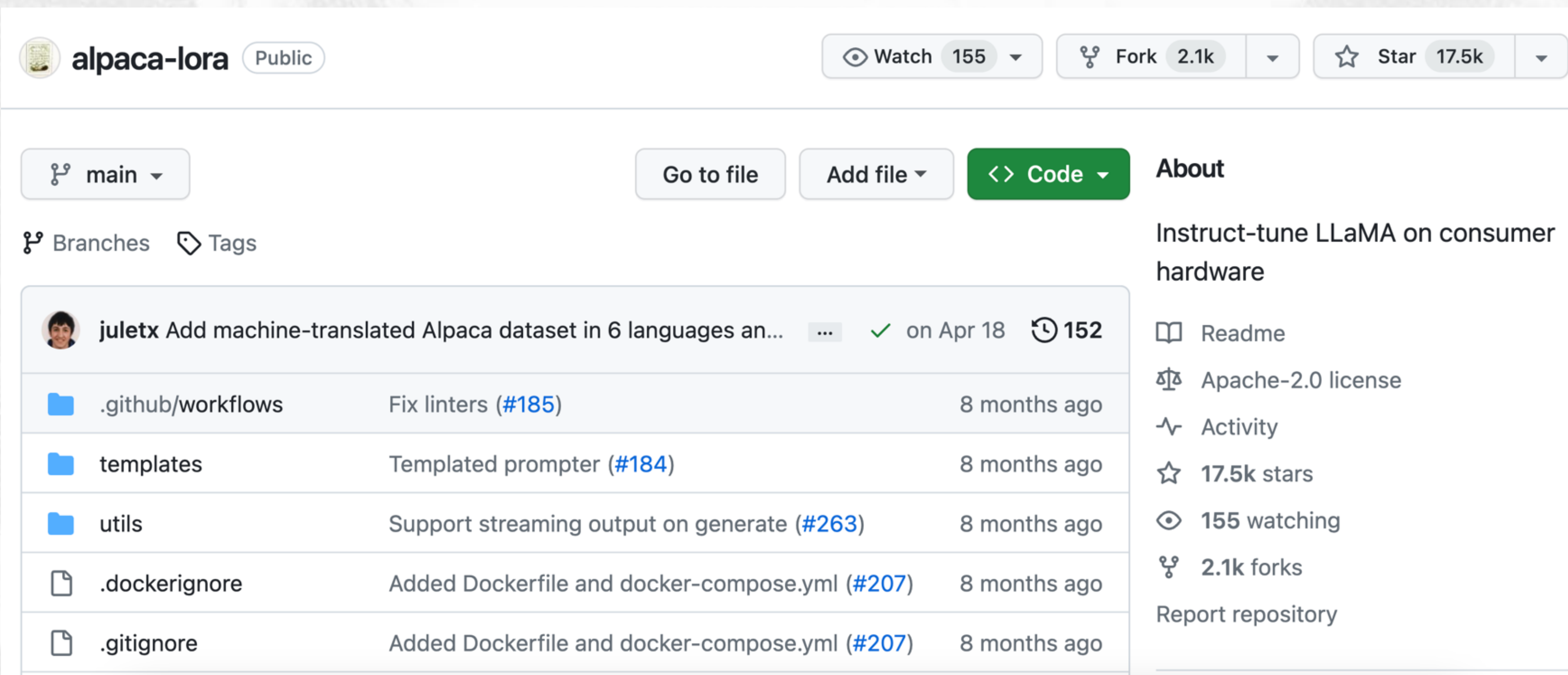
Data: Training set D_{train}, a batch of validation data B_{val}

```
Train super-network M on the training set D_train for K_1 epochs;
for n = 0; n < N_A do
    for a single LoRA rank r_{m,i} on M do
        Calculate the importance score IS(r_{m,i}) on B_val;
    Prune n_0 LoRA ranks with lowest importance scores;
    if there are modules not pruned then
        Add n_0 LoRA ranks to the un-pruned modules;
    Further train the Super-network M on D_train for K_2 epochs;
```

Alpaca LoRA

Within just a few days following the release of Alpaca's training material, LoRA was utilized to fine-tune LLaMa into Alpaca efficiently, using only a «small» GPU:

- https://github.com/tloen/alpaca-lora



Aligning LLMs

RAG: motivations

- Large pre-trained language models have been shown to **store factual knowledge** in their parameters, and **achieve state-of-the-art results** when fine-tuned on downstream NLP tasks.
- However, **their ability to access and precisely manipulate knowledge is still limited**, and hence on knowledge-intensive tasks, their performance lags behind task-specific architectures.
- Additionally, **providing provenance for their decisions** and **updating their world knowledge** remain open research problems.

Knowledge Integration and LLMs: RAG Models

- Retrieval Augmented Generation (Lewis et al., 2020)
  - At *generation time* contextual information able to qualify the LLM response is made available
  - It is essential for knowledge intensive tasks
  - It is possible to apply RAG either to the *pre-training* or to the *fine-tuning* and *prompting* stage
  - It has been shown to impact positively onto hallucinations

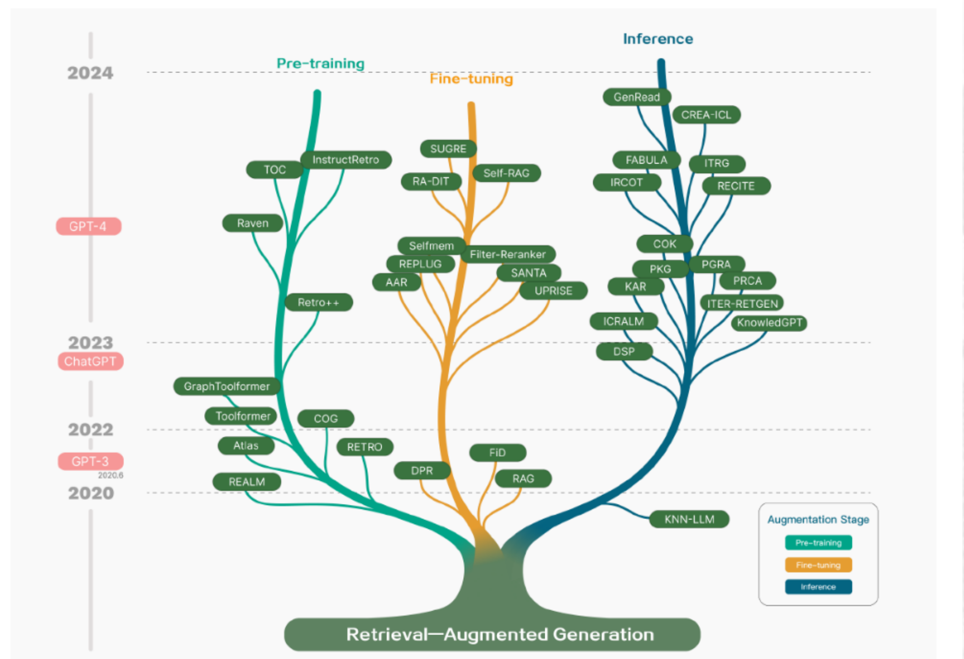
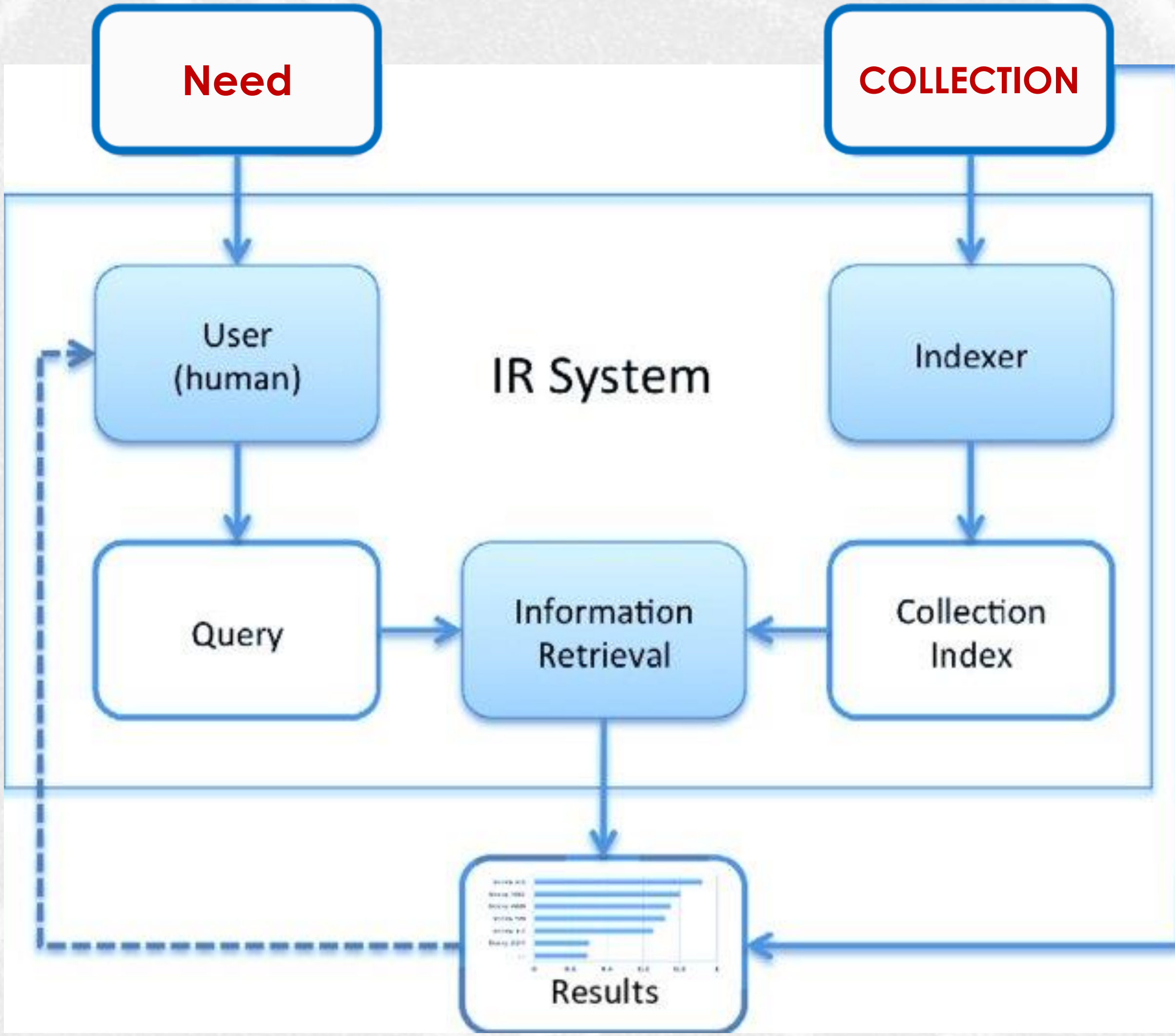


Figure 1: Technology tree of RAG research development featuring representative works

(Lewis et al, 2020) Retrieval-augmented generation for knowledge-intensive NLP tasks. Proceedings of NIPS, Advances in Neural Information Processing Systems, 33 (2020): 9459-9474.

The basic Retrieval workflow



RAG: the steps

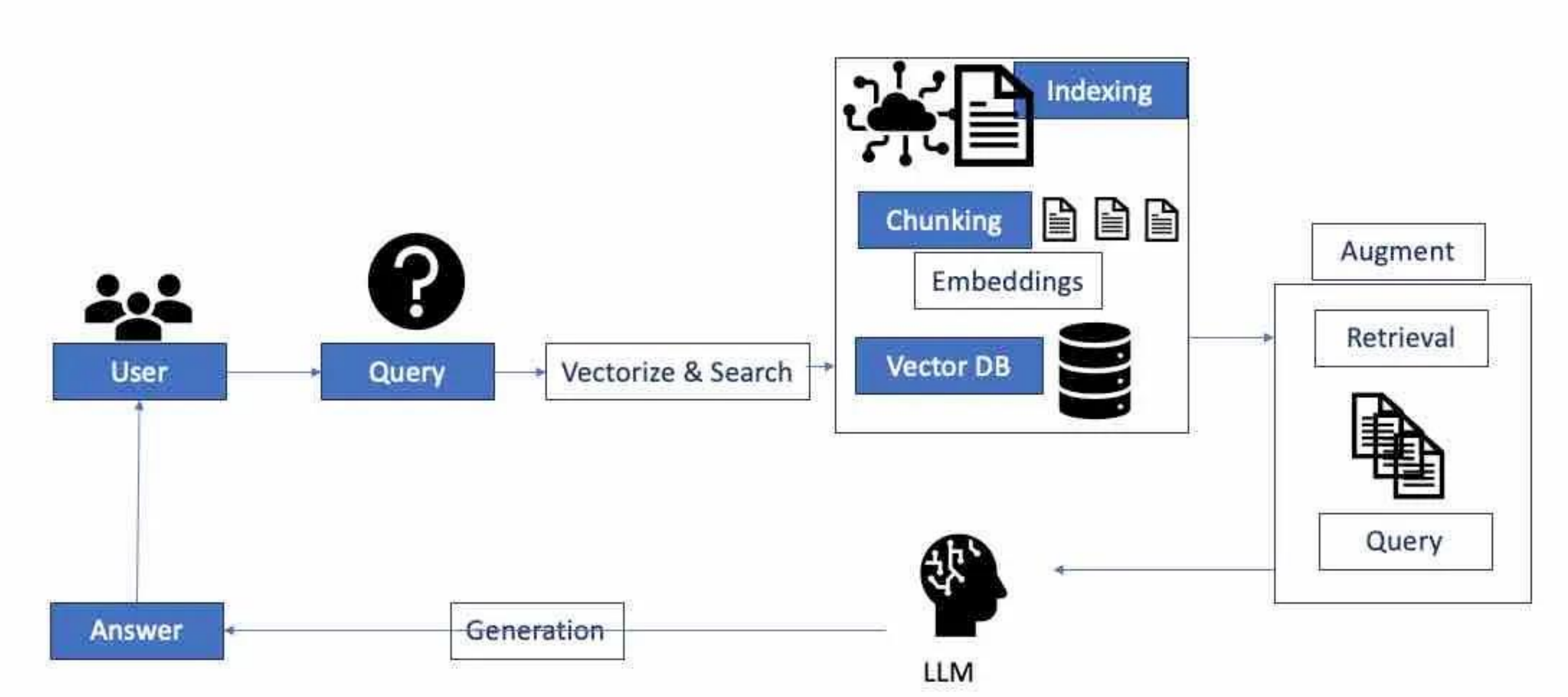
1. INPUT: It corresponds to the question posed to an LLM system. If no RAG is applied, LLM responds to the question through standard decoding

2. INDEXING: To employ RAG, a set of reference documents are to be indexed.
   - It involves chunking the documents, embeddings these chunks, and then indexing embeddings into a *vector store*.
   - The input query is also embedded.

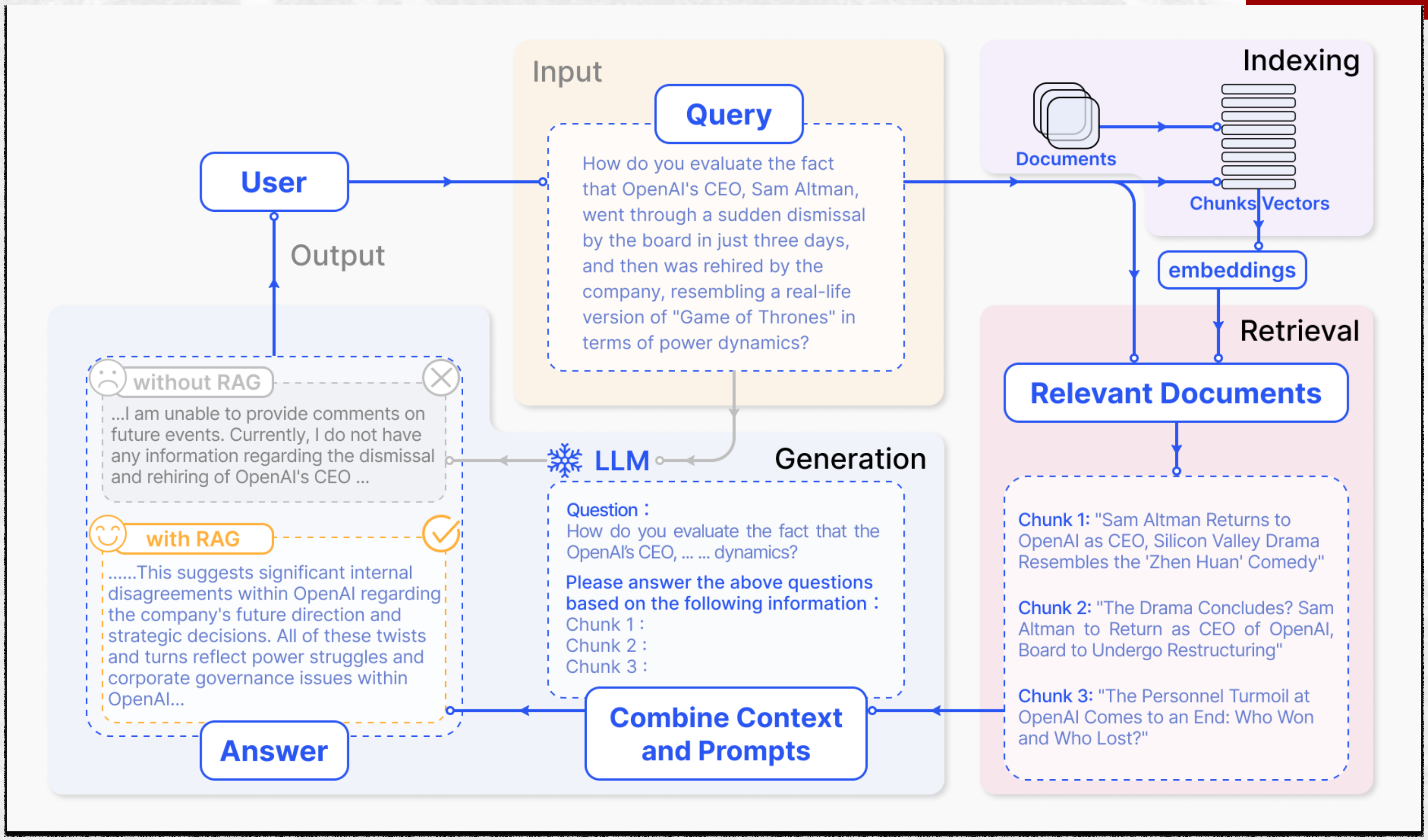
3. RETRIEVAL: Relevant documents are retrieved by comparing the query embedding against the document vectors.

4. GENERATION: Retrieved documents are first merged with the original prompt to provide additional context and then the LLM response generation is triggered:
   - This combined text and prompt is the input for response generation, that produces the final output provided to the user.

The RAG architecture



RAG models: the information flow



RAG models: the training task

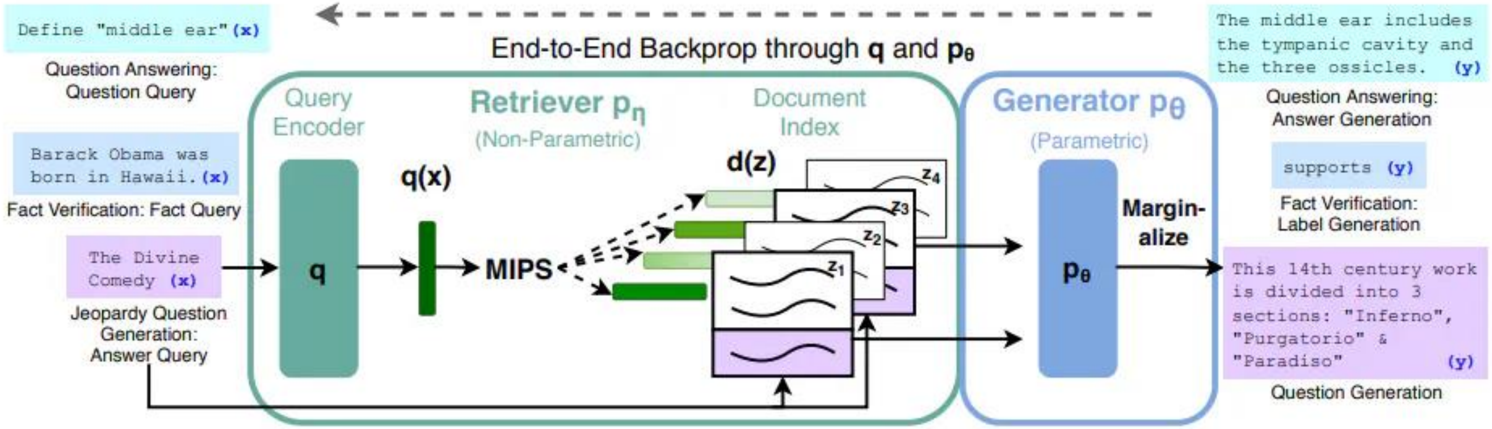
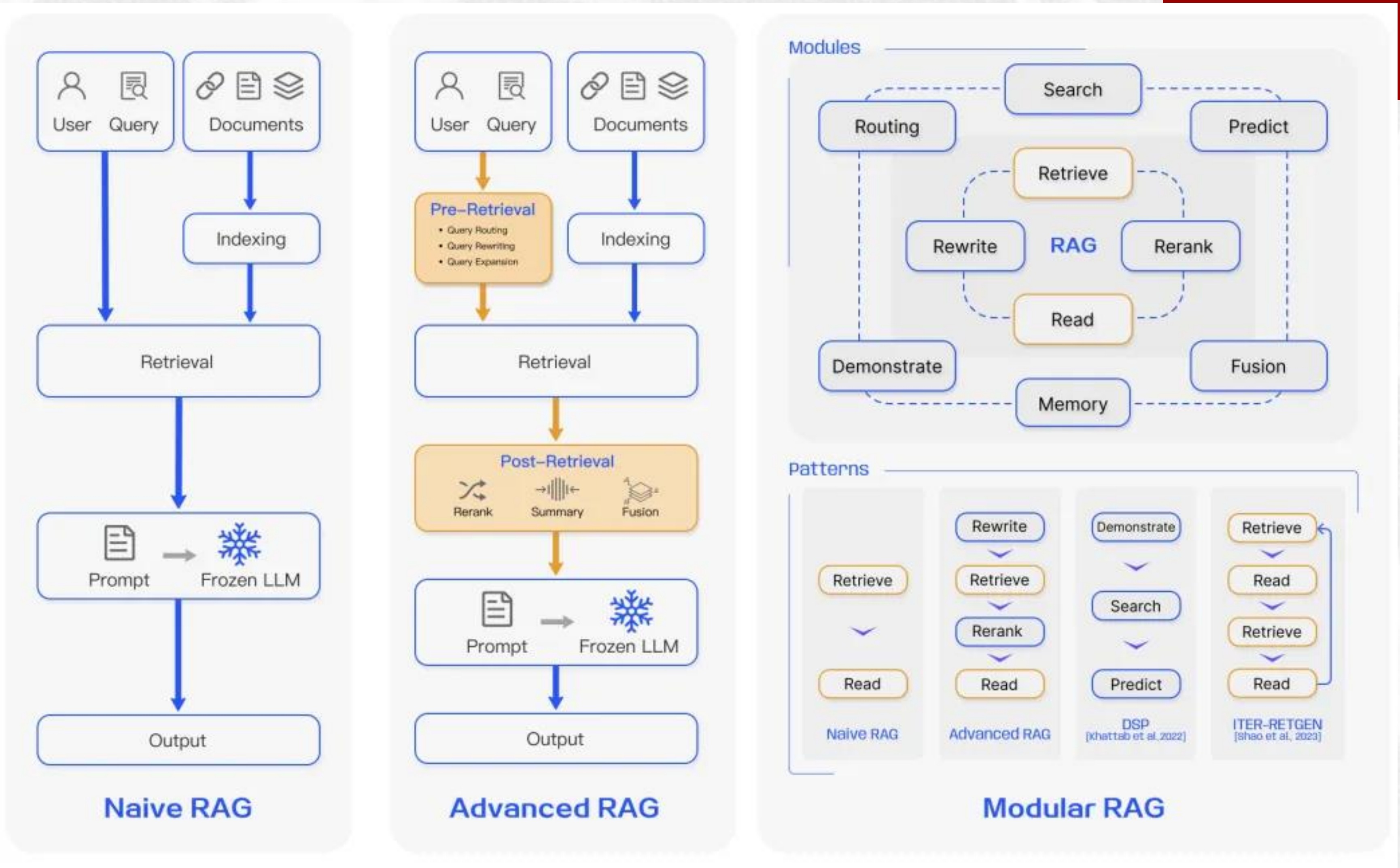


Figure 1: Overview of our approach. We combine a pre-trained retriever (*Query Encoder* + *Document Index*) with a pre-trained seq2seq model (*Generator*) and fine-tune end-to-end. For query x, we use Maximum Inner Product Search (MIPS) to find the top-K documents z_i. For final prediction y, we treat z as a latent variable and marginalize over seq2seq predictions given different documents.

Types of RAG

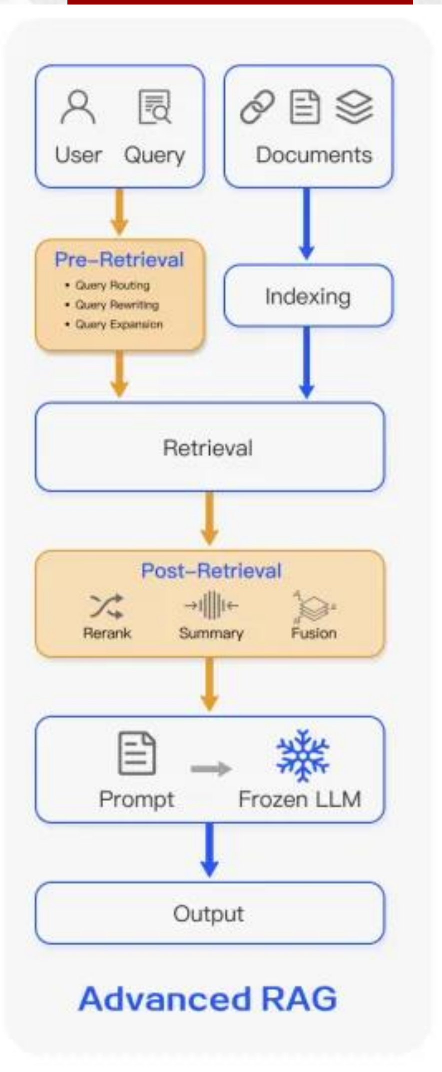


User
Query
Documents
Indexing
Retrieval
Prompt
Frozen LLM
Output
Naive RAG
User
Query
Documents
Pre-Retrieval
• Query Routing
• Query Rewriting
• Query Expansion
Indexing
Retrieval
Post-Retrieval
Rerank
Summary
Fusion
Prompt
Frozen LLM
Output
Advanced RAG
Modules
Search
Routing
Predict
Retrieve
Rewrite
RAG
Rerank
Read
Demonstrate
Fusion
Memory
Patterns
Retrieve
Read
Naive RAG
Rewrite
Retrieve
Rerank
Read
Advanced RAG
Demonstrate
Search
Predict
DSP
[Khattab et al. 2022]
Retrieve
Read
Retrieve
Read
ITER-RETGEN
[Shao et al., 2023]
Modular RAG

Advanced RAGs

- It employs optimization across the (A) pre-retrieval, (B) retrieval, and (C) post-retrieval processes.

A. The **pre-retrieval phase** involves refining data indexing through five key stages:
   - enhancing data granularity,
   - optimizing index structures,
   - adding metadata,
   - alignment optimization, and
   - mixed retrieval

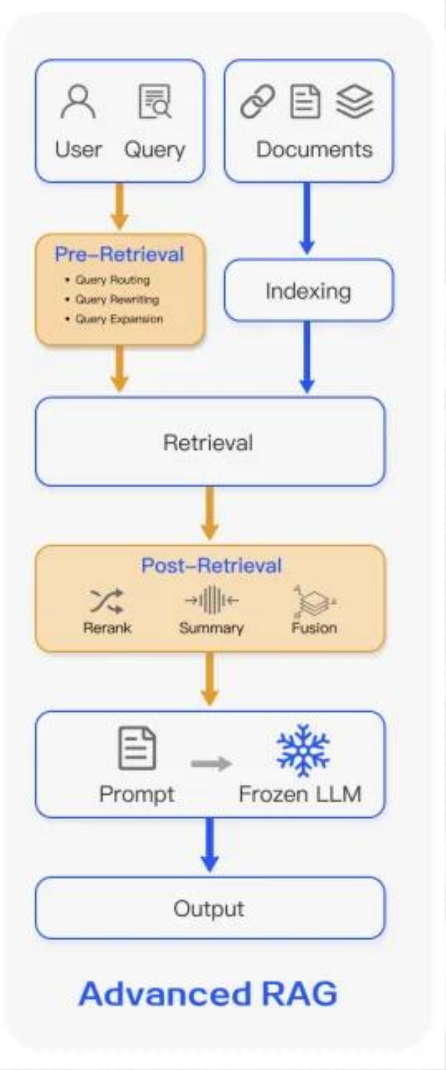


Advanced RAGs

- It employs optimization across the (A) pre-retrieval, (B) retrieval, and (C) post-retrieval processes.

B. The **retrieval phase involves optimizing the embedding model itself** to maximize the quality of context chunks. Strategies may include:

- **fine-tuning embeddings** to improve retrieval relevance or
- **employing dynamic embeddings** that better capture contextual nuances (e.g., OpenAI's embeddings-ada-02 model)

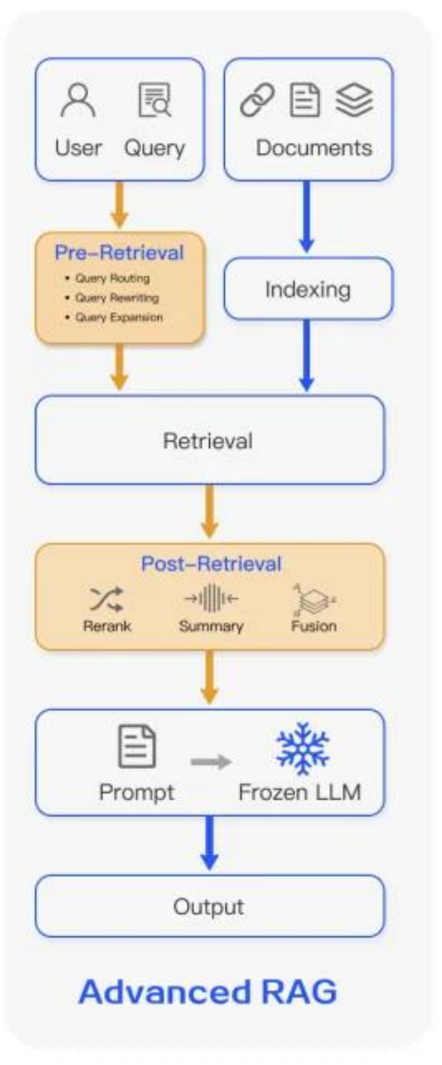


Advanced RAGs

- It employs optimization across the (A) pre-retrieval, (B) retrieval, and (C) post-retrieval processes.

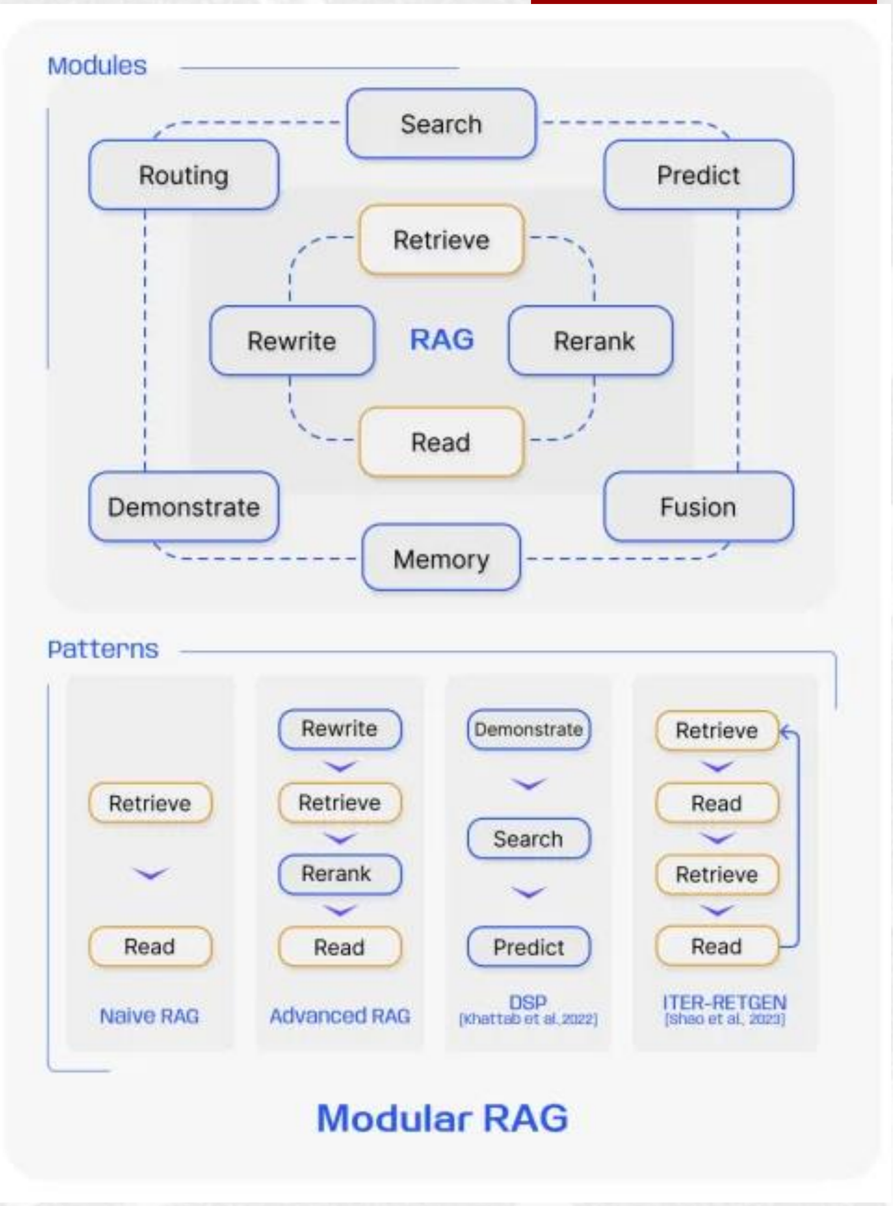
C. The **post-retrieval phase** focuses on **circumventing context window limitations** and **managing noisy or distracting information**. **Re-ranking is a common approach** to address these challenges, involving techniques such as

- relocating relevant context to the edges of the prompt or
- **recalculating semantic similarity between the query and relevant text chunks**.
- **Prompt compression techniques** may also aid

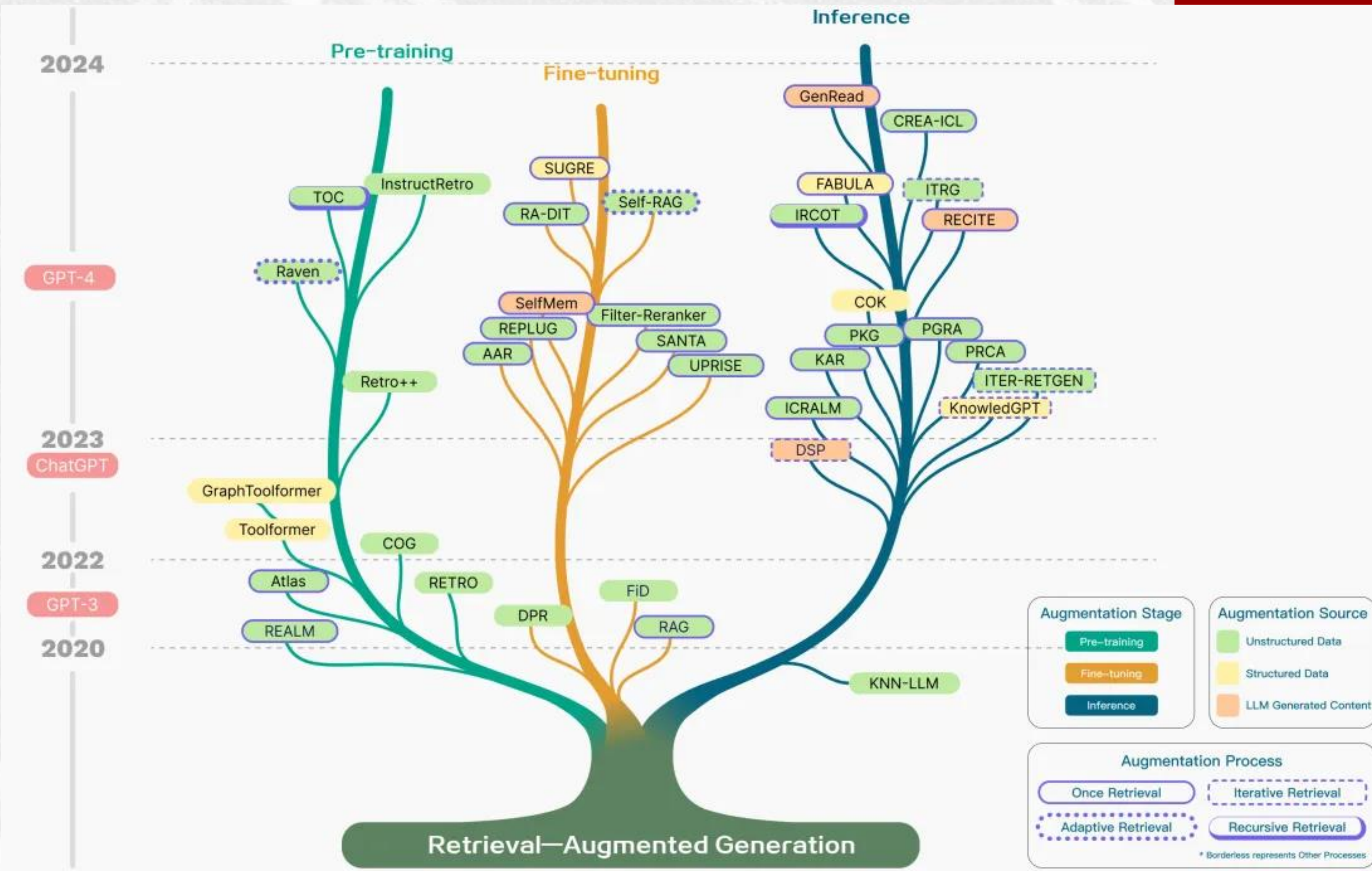


Modular RAG

- SEARCH MODULE: Tailored for specific use-cases, it can perform direct searches on various corpora, utilizing LLM-generated code and query languages.
- MEMORY MODULE: Uses the LLM's memory for retrieval, improving alignment with data distributions.
- FUSION: Expands user queries into diverse perspectives, improving search results through multi-query approaches and re-ranking.
- ROUTING: Determines actions for queries, selecting the appropriate data source for retrieval.
- PREDICT: Uses the LLM to generate context instead of direct retrieval to reduce redundancy and noise.
- TASK ADAPTER: Adapts RAG to various tasks, enhancing universality and creating task-specific retrievers.



The fondational RAGs



RAG evaluation

- The evaluation of a RAG framework focuses on **three primary quality scores and four abilities**.
- QUALITY SCORES encompass measuring
  - context relevance (precision and specificity of retrieved context),
  - answer faithfulness (faithfulness of answers to retrieved context), and
  - answer relevance (relevance of answers to posed questions).
- Additionally, four abilities measure ADAPTABILITY AND EFFICIENCY of a RAG system:
  - noise robustness,
  - negative rejection,
  - information integration, and
  - counterfactual robustness.

RAG evaluation: DEFs

- *Context Relevance* - Precision and Specificity of retrieved context (*How much does the context actually relate to the query?*)
- *Answer Faithfulness* - *Is the answer true to the retrieved context*? *Is it making anything up that isn't within the context*?
- *Answer Relevance* - *Is the answer actually relevant to the core meaning of the query*?
- *Noise Robustness* - *How well can the model ignore useless information that is retrieved*?
- *Negative Rejection* - *How well can the model refrain from responding when the context does not have the necessary information included*?
- *Information Integration* - *How well can the model combine all of the information into a clean and summarized answer*?
- *Counterfactual Robustness* - *How well can the model recognize that the provided context is actually wrong, and discard the information*?

RAG evaluation

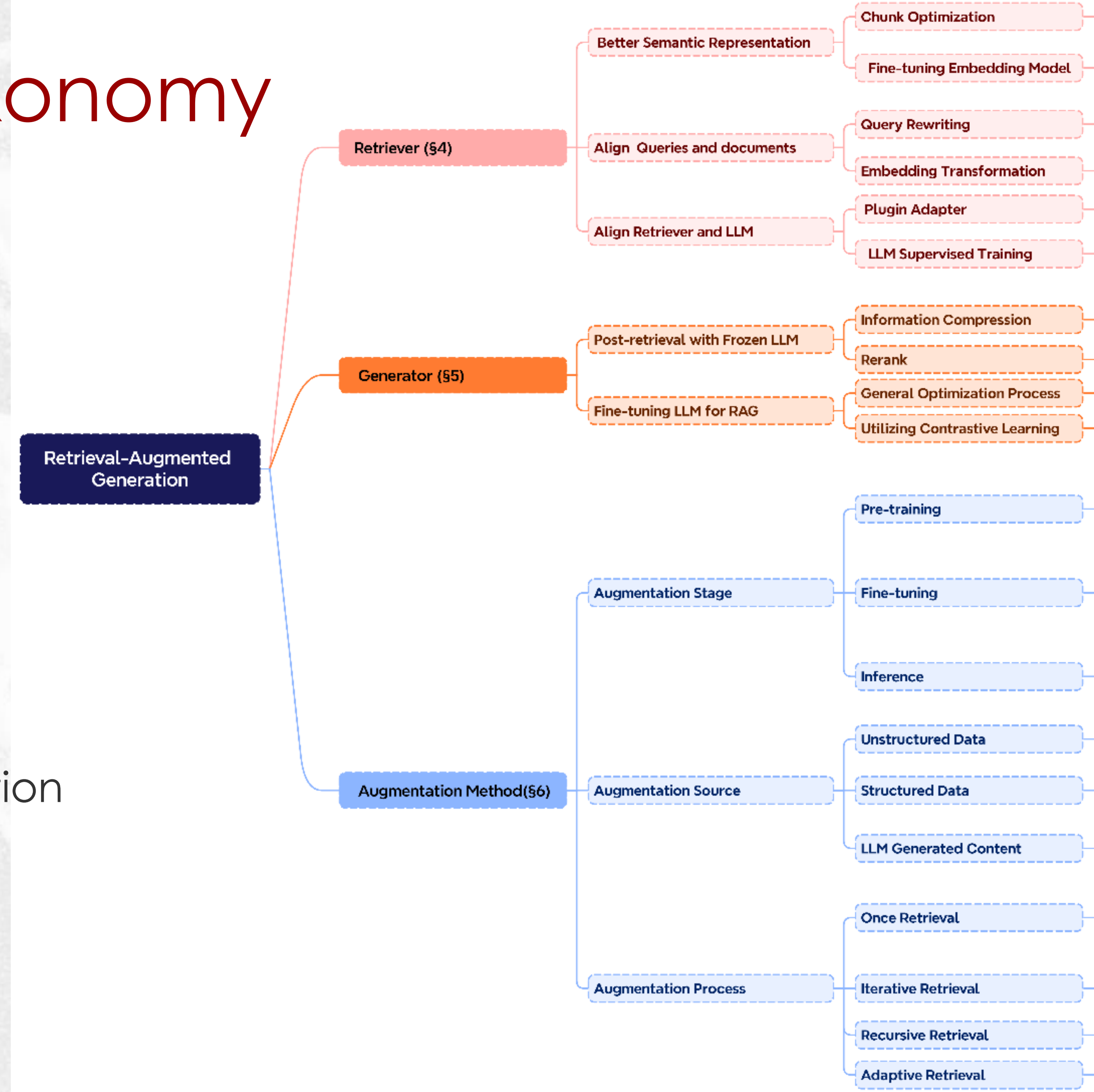
	Context Relevance	Faithfulness	Answer Relevance	Noise Robustness	Negative Rejection	Information Integration	Counterfactual Robustness
Accuracy	✓	✓	✓	✓	✓	✓	✓
EM					✓		
Recall	✓						
Precision	✓			✓			
R-Rate							✓
Cosine Similarity			✓				
Hit Rate	✓						
MRR	✓						
NDCG	✓						

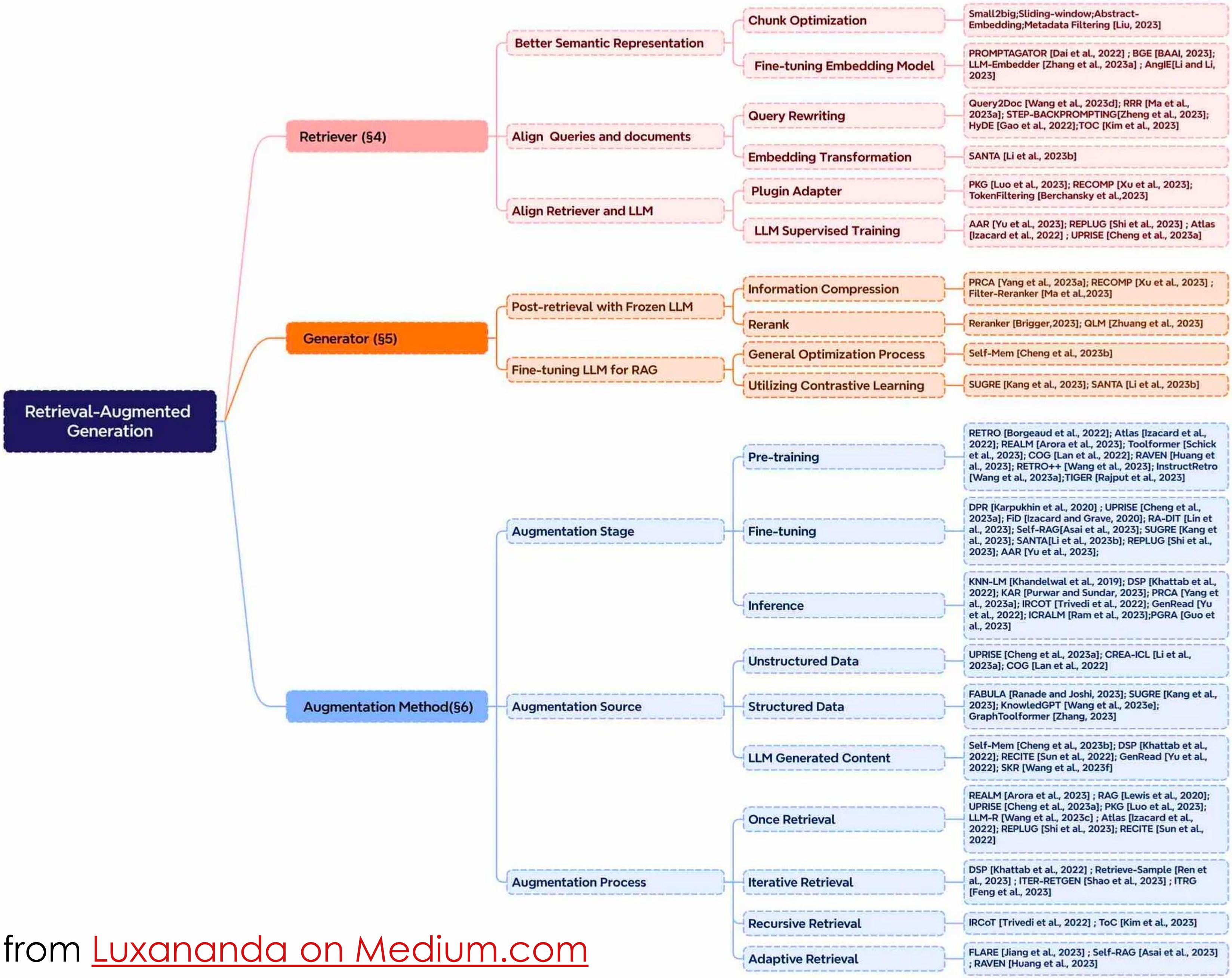
Table 3: Summary of evaluation frameworks

Evaluation Framework	**Evaluation Targets**	**Evaluation Aspects**	**Quantitative Metrics**
RGB†	Retrieval Quality Generation Quality	Noise Robustness Negative Rejection Information Integration Counterfactual Robustness	Accuracy EM Accuracy Accuracy
RECALL†	Generation Quality	Counterfactual Robustness	R-Rate (Reappearance Rate)
RAGAS‡	Retrieval Quality Generation Quality	Context Relevance Faithfulness Answer Relevance	* * Cosine Similarity
ARES‡	Retrieval Quality Generation Quality	Context Relevance Faithfulness Answer Relevance	Accuracy Accuracy Accuracy
TruLens‡	Retrieval Quality Generation Quality	Context Relevance Faithfulness Answer Relevance	* * *

A RAG Taxonomy

- Research is active in different directions
  - Retrieval
  - Generation
  - Textual, Logical and Procedural Augmentation
- DBs or KG are often explored as information sources

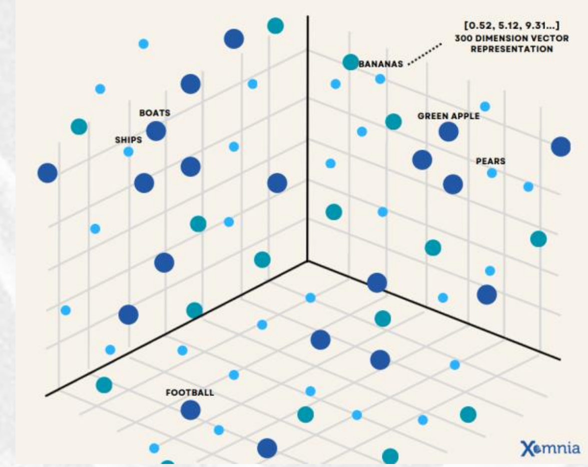




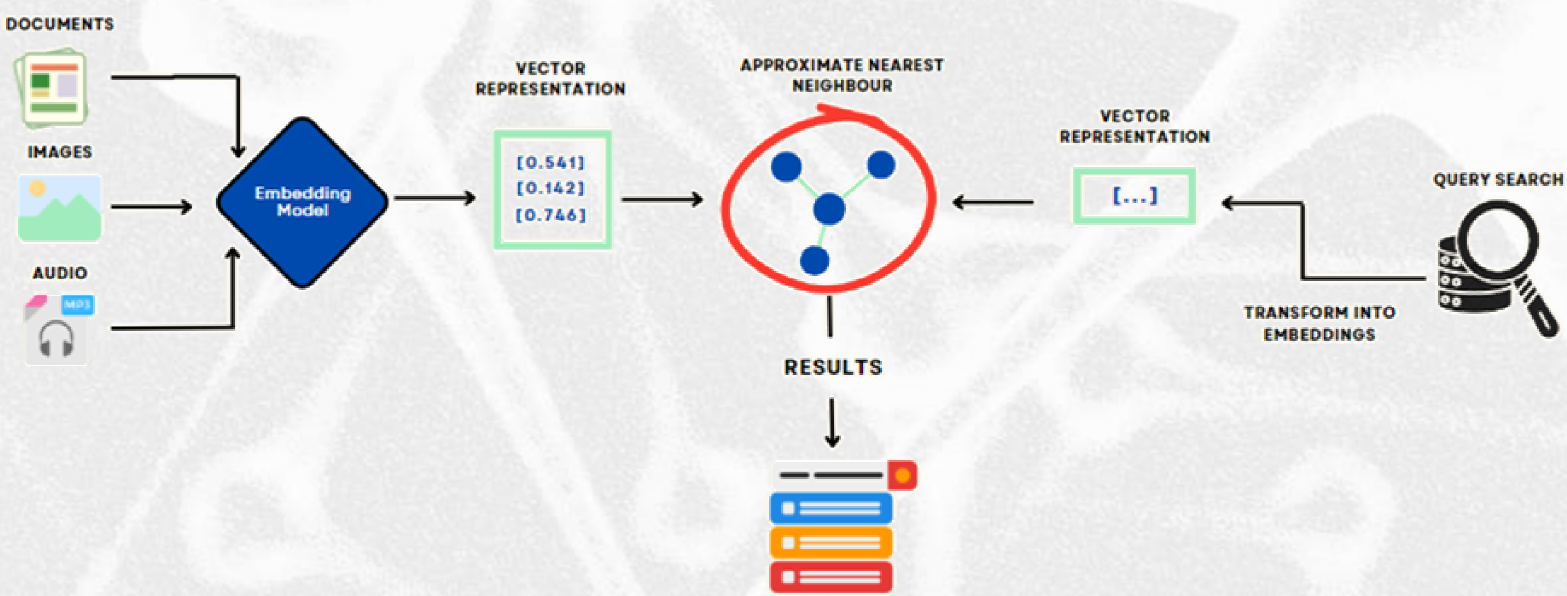
from [Luxananda on Medium.com]

Applications of RAGs

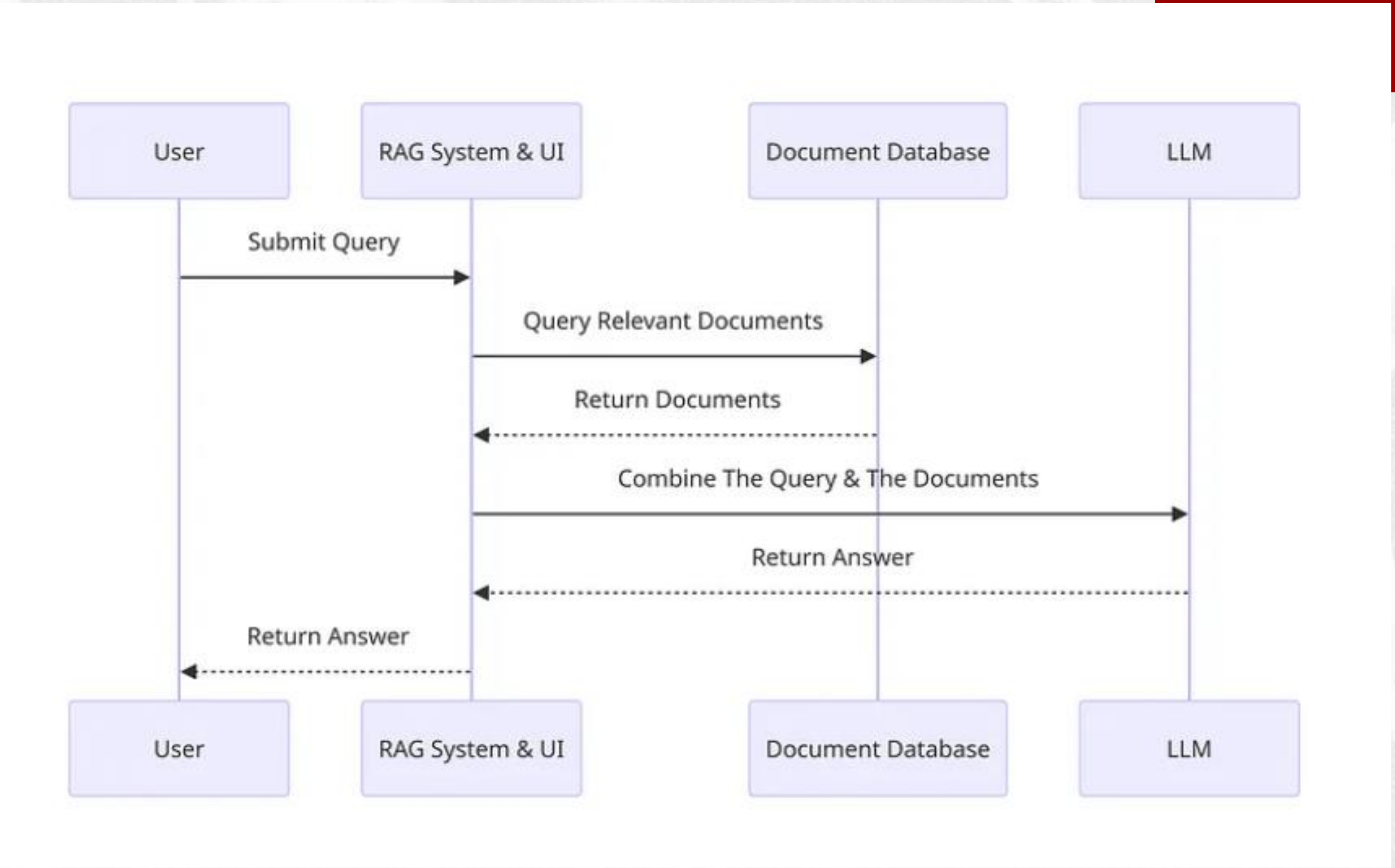
Vector Databases



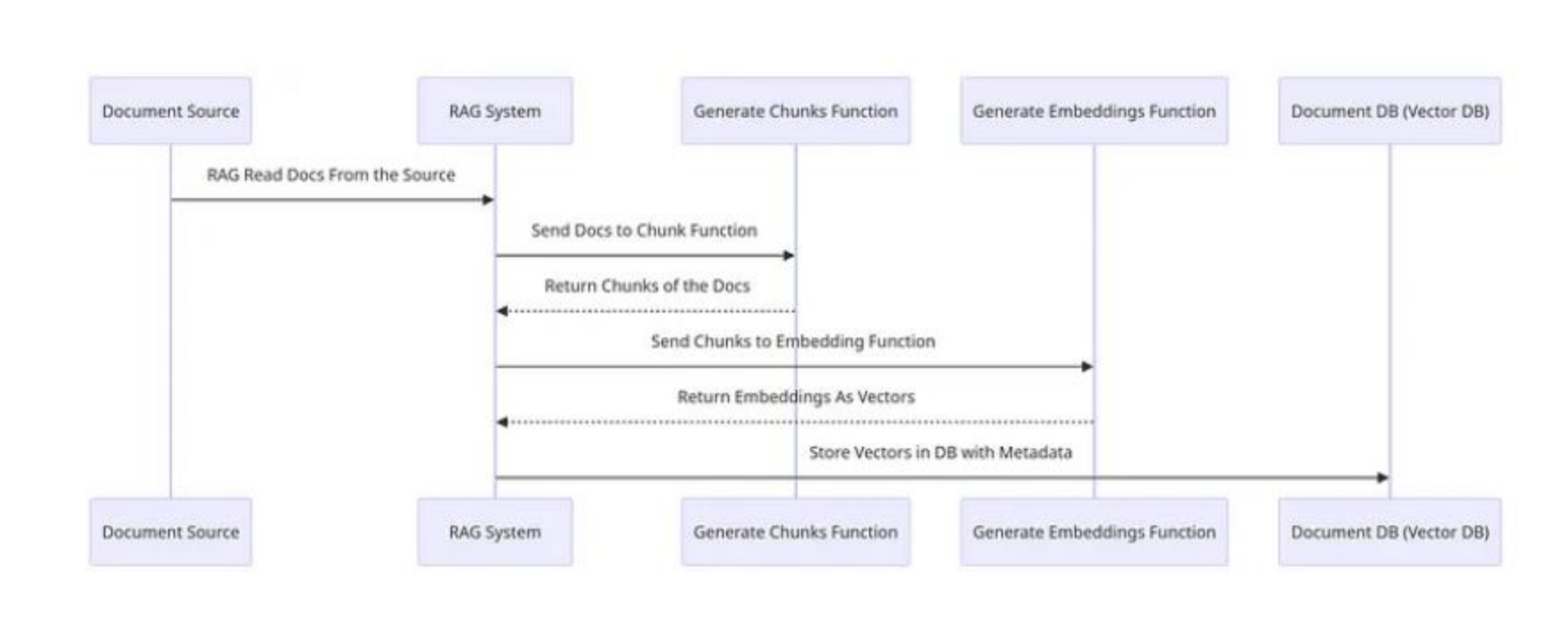
- A vector database is a type of database that stores and manages unstructured data, such as
  - texts, images, or audio,
- in vector embeddings (high-dimensional vectors) to make it easy to find and retrieve similar objects quickly.



RAG: workflows



RAG: data gathering



RAG potential applications

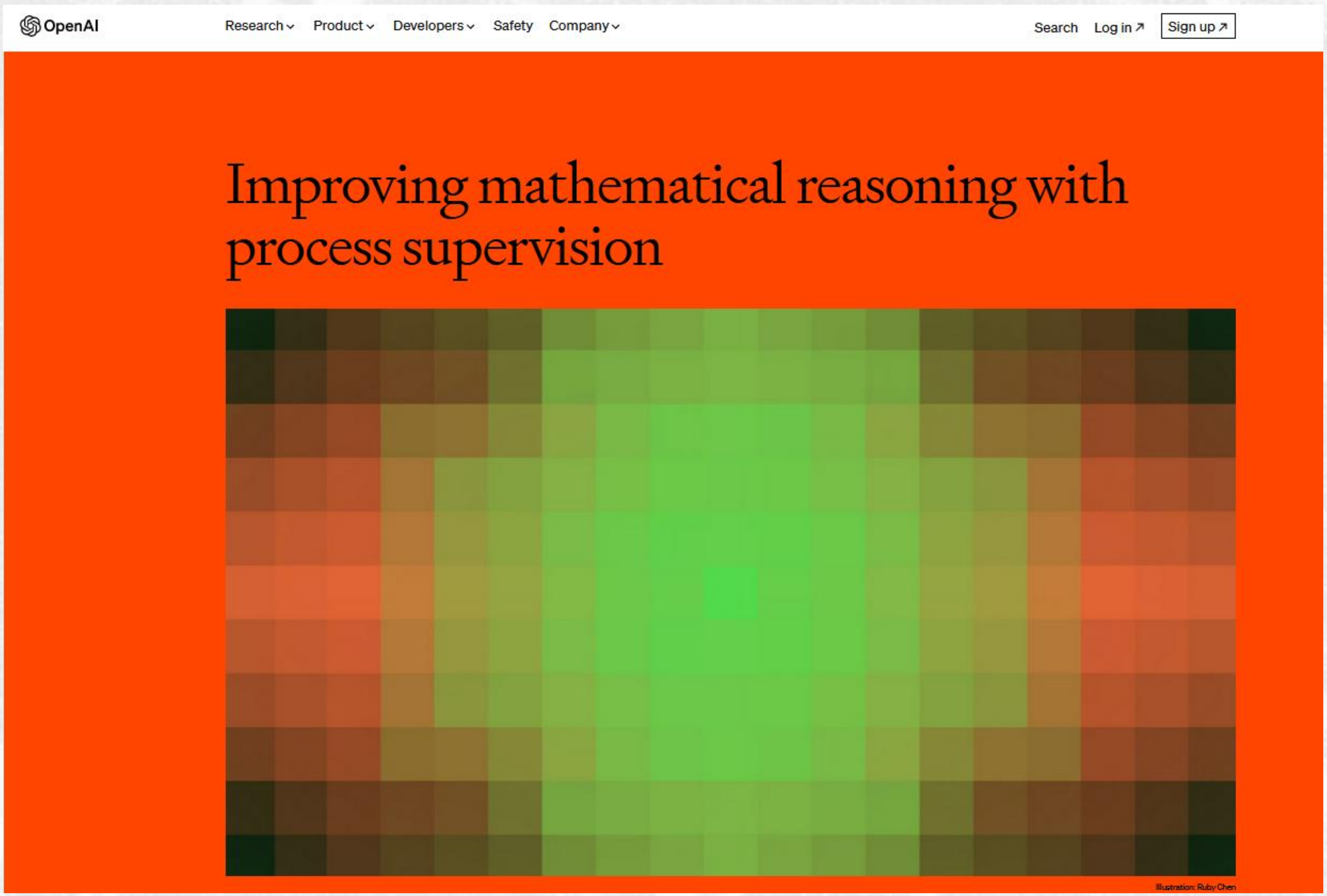
- **Question Answering** where facts are derived from the retrieved texts that represent up-to-date information (in IR style)
- **Summarization**, where on-the-fly retrieval of supporting documents is carried out
- **DB query in NL**, as individual DB records can be seen as texts
- **KB retrieval** and **alignment to specific user' needs**
- …

RAG: business applications

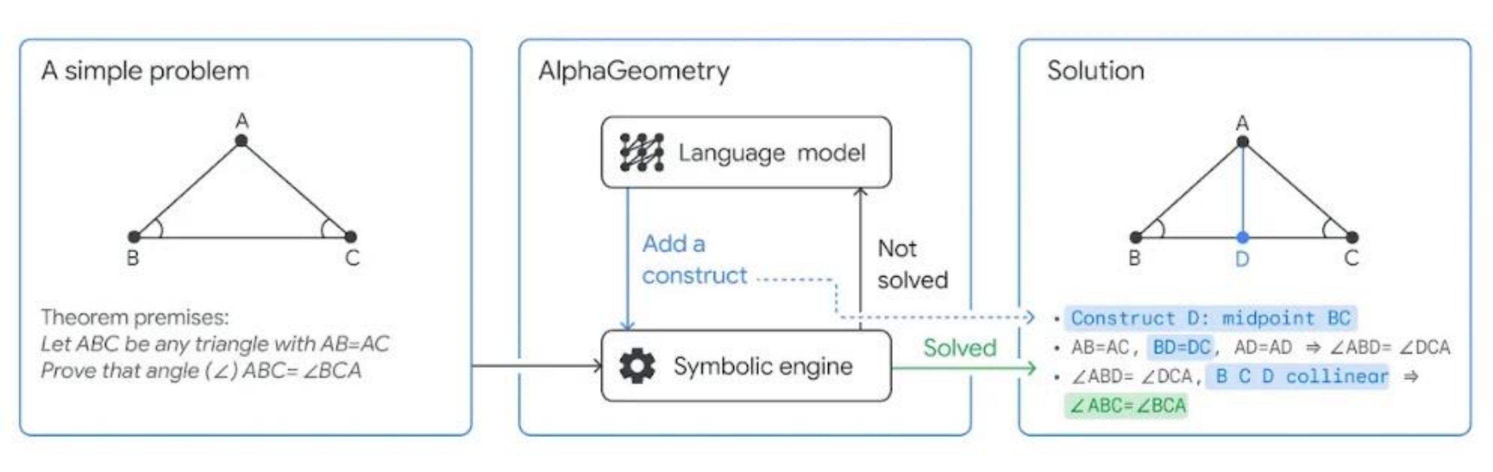
Practical applications of RAG include for exa,ple:

- **Customer support**: RAG can be used to build chatbots or AI assistants that provide personalized assistance across various questions and issues.
- **Content generation**: RAG enables the automation of content creation tasks, such as writing aids or content curation apps.
- **Education**: RAG can serve as a learning assistant, providing explanations and summaries of educational content.
- **Research**: RAG can assist researchers in obtaining relevant information and insights from large document collections.

Future directions

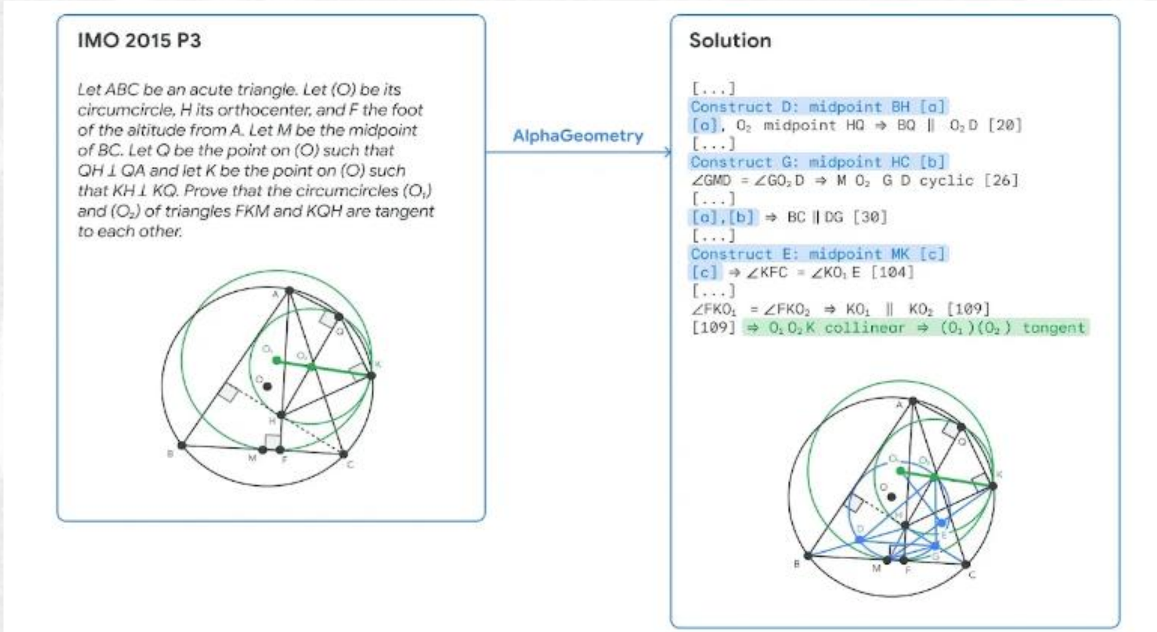


AlphaGeometry (Google DeepMind, Jan 2024)



[Trinh, Trieu H., Wu Yuhuai, Le Quoc V., He He, Luong Thang, Solving olympiad geometry without human demonstrations, Nature, 625, 2024.]()

AlphaGeometry (Google DeepMind, Jan 2024)



Problem 3 of the 2015 International Mathematics Olympiad (left) and a condensed version of AlphaGeometry's solution (right). The blue elements are added constructs. AlphaGeometry's solution has 109 logical steps.

[Trinh, Trieu H., Wu Yuhuai, Le Quoc V., He He, Luong Thang, Solving olympiad geometry without human demonstrations, Nature, 625, 2024.]

LORA and RAG: bibliography

- Pengfei Li et al., 2023, Making AI Less "Thirsty": Uncovering and Addressing the Secret Water Footprint of AI Models.
- LoRA: LoRA: Low-Rank Adaptation of Large Language Models, Edward Hu et al., 2021
- Learned Adapters Are Better Than Manually Designed Adapters, Zhang et al., 2023
- ALoRA: Allocating Low-Rank Adaptation for Fine-tuning Large Language Models, liu et al., 2024
- RAG:
  - (Lewis et al, 2020) Retrieval-augmented generation for knowledge-intensive NLP tasks. Proceedings of NIPS, Advances in Neural Information Processing Systems, 33 (2020): 9459-9474.
- RAG surveys & tutorial:
- Retrieval-Augmented Generation for Large Language Models: A Survey, Gao et al, 2023
- A Survey on Retrieval-Augmented Text Generation for Large Language Models, Yizheng Huang and Jimmy X. Huang, 2024
- "Towards LLM #8: Techniques of Prompt Engineering — Retrieval-Augmented Generation (Part 1)", LAKSHMI VENKATESH on Luxananda - Medium.com