Open in app ↗

Search Write

Modular RAG and RAG Flow: Part II

Save 0:41 by listening

0:00 10:39

How to design your own RAG Flow?

Yunfan Gao · Follow

12 min read · Jan 29, 2024

10:39

112 3

In Part I, we primarily discussed the three-tier structure of modular RAG (Module Type - Module- Operator) and briefly mentioned the concept of RAG Flow.

Modular RAG and RAG Flow: Part I

A compressive and high-level summarization of RAG .

medium.com

After defining Module and Operator, they can help us to view various RAG methods from a flow perspective. Each RAG can be arranged with a set of operators.

Alt+Q

Framework of Modular RAG

So, under the paradigm of modular RAG, how should we design our RAG system?

In Part II, we will delve into the **typical RAG Flow pattern**, **specific RAG Flow implementation**, and **best industry case.**

Typical RAG Flow Pattern and Implementation

First, let's explore the prominent patterns for RAG flow, along with the specific flows under each template, illustrating how different modules and operators are orchestrated.

In the context of RAG Flow, we will delineate three distinct flows for the fine-tuning stage and four flows for the inference stage.

Tuning Stage

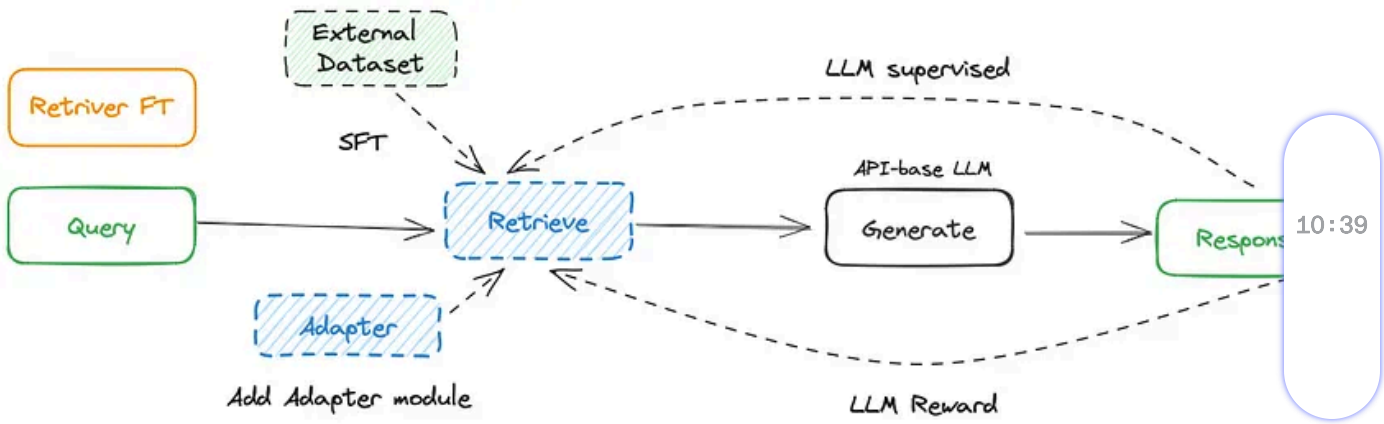
> *Retriever Fine-tuning, Generator Fine-tuning, and Dual Fine-tuning.*

Retriever FT

In the RAG Flow, common methods for fine-tuning the retriever include:

- **Direct fine-tuning of the retriever.** Constructing a specialized dataset for retrieval and fine-tuning the dense retriever. For example, using open-source retrieval datasets or constructing one based on your domain-specific data.
- **Adding trainable Adapter modules.** Sometimes, direct fine-tuning of the API-base embedding model (e.g., OpenAI Ada-002 and Cohere) is not feasible. Incorporating an Adapter module can enhance the representation of your data. Additionally, the adapter module facilitates better alignment with downstream tasks, whether for task-specific (e.g., PCRA) or general purposes (e.g., AAR).

- **LM-supervised Retrieval (LSR).** Fine-tuning the retriever based on the results generated by LLM.
- **LLM Reward RL :** Still using the LLM output results as the supervisory signal. Employing reinforcement learning to align the retriever with the generator. The whole retrieval process is disassembled in the form of a generative Markov chain.



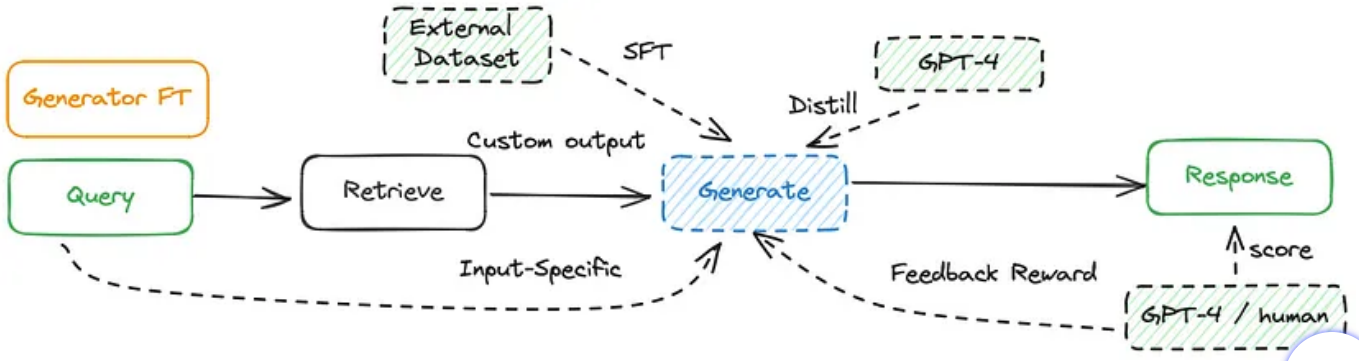
Typical RAG Flow Pattern for Retriever FT

Generator FT

The primary methods for fine-tuning a generator in RAG Flow include:

- **Direct fine-tuning.** Fine-tuning through an external dataset can supplement the generator with additional knowledge. Another benefit is the ability to customize input and output formats. By setting theQ&A format, LLM can understand specific data formats and output according to instructions.
- **GPT-4 distillation.** When using on-premise deployment of open-source models, a simple and effective method is to use GPT-4 to batch construct fine-tuning data to enhance the capabilities of the open-source model.

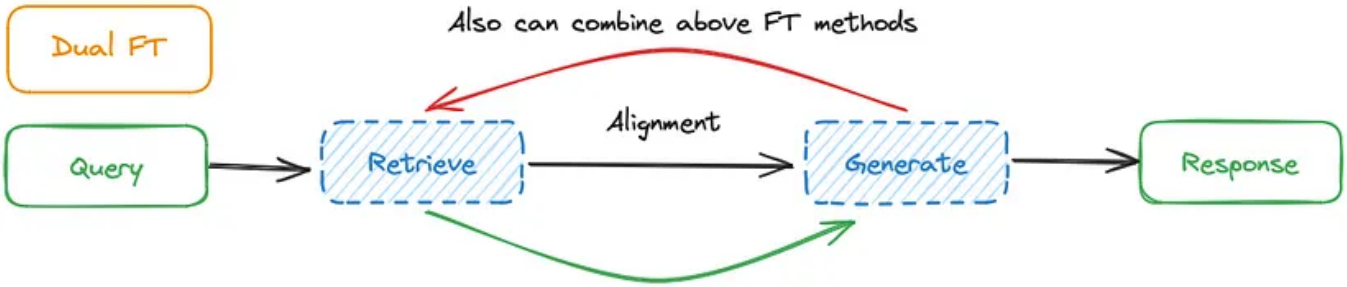
- **Reinforcement Learning from LLM/Human Feedback.** Reinforcement learning based on feedback from the final generated answers. In addition to using human evaluations, GPT-4 can also serve as an evaluative judge.



Typical RAG Flow Pattern for Generator FT

Dual FT

In the RAG system, fine-tuning both the retriever and the generator simultaneously is a unique feature of the RAG system. It is important to note that the emphasis of system fine-tuning is on the coordination between the retriever and the generator. Fine-tuning the retriever and the generator separately separately belongs to the combination of the former two, rather than being part of Dual FT.



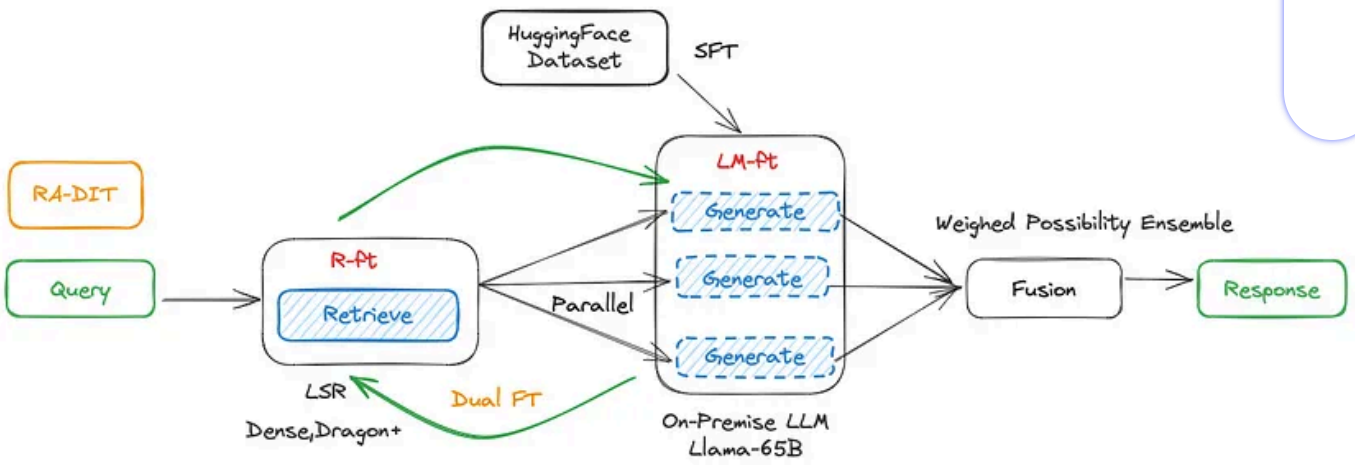
Typical RAG Flow Pattern for Dual FT

An exemplary implementation is RA-DIT, which fine-tunes both the LLM and the retriever. The LM-ft component updates the LLM to maximize the

likelihood of the correct answer given the retrieval-augmented instructions while the R-ft component updates the retriever to minimize the KL-Divergence between the retriever score distribution and the LLM preference.

The framework employs a on-premises Llama as the generator and a state-of-the-art dual-encoder based dense retriever, DRAGON+, as the retriever.

Following REPLUG, RA-DIT retrieve relevant text chunks based on the language model prompt. Each retrieved chunk is prepended to the prompt, and the predictions from multiple chunks are computed in parallel and ensembled by weighted possibilty to produce the final output.



RAG Flow in RA-DIT

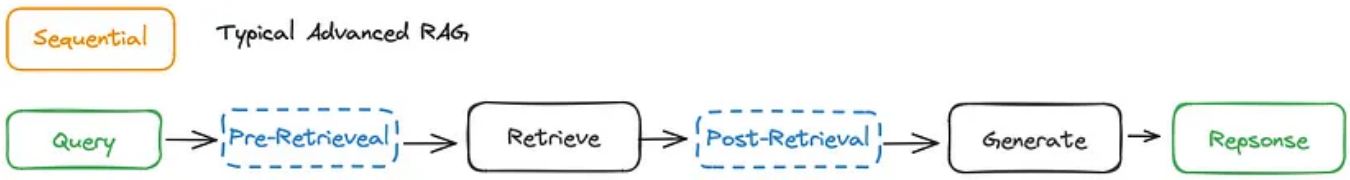
Inference Stage

In the inference stage, we have distilled four typical RAG Flow patterns.

Sequential

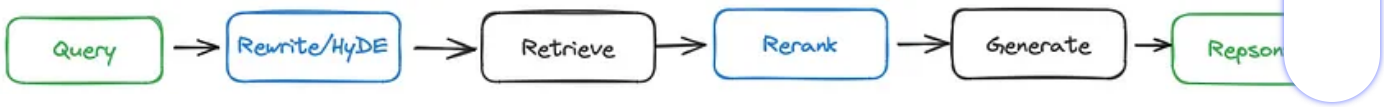
The sequential structure of the RAG Flow organizes the modules and operators of RAG in a linear pipeline, as depicted in the following diagram. If it includes both Pre-Retrieval and Post-Retrieval module types, it represents

the typical Advanced RAG paradigm; otherwise, it embodies the typical Naive RAG paradigm.



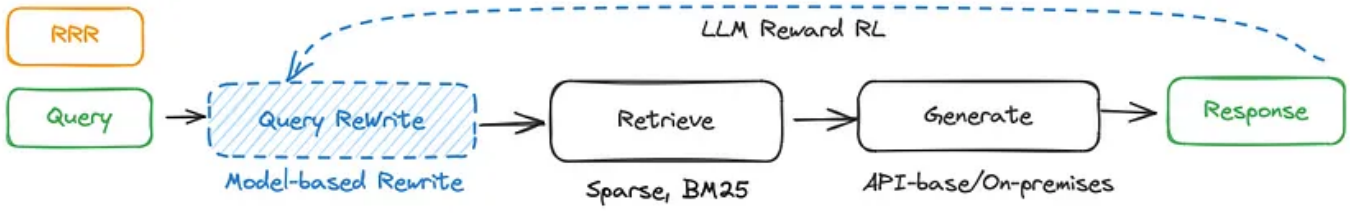
Sequential RAG Flow Pattern

The most widely used RAG Pipeline currently is the Sequential, which commonly includes Query Rewrite or HyDE before retrieval and Rerank operator after retrieval, such as in the case of QAnything.



The most commonly used sequential RAG Flow

Rewrite-Retrieve-Read (RRR) is also a typical sequential structure. The Query Rewrite module is a smaller trainable language model, and in the context of reinforcement learning, the optimization of the rewriter is formalized as a Markov decision process, with the final output of the LLM serving as the reward. The retriever utilizes a sparse encoding model, BM25.

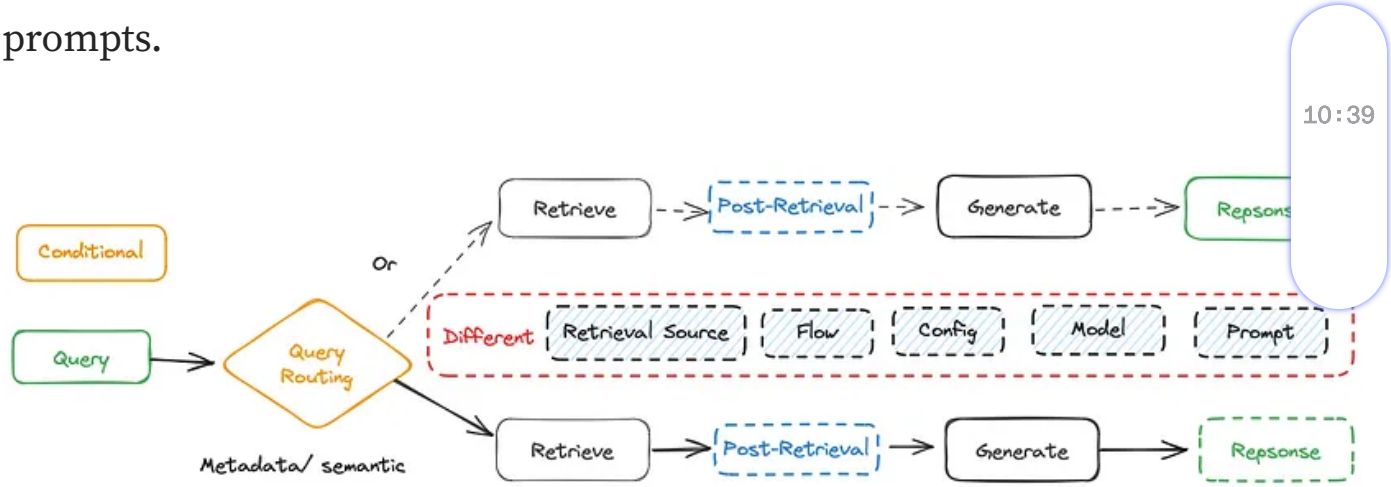


RAG Flow in RRR

Conditional

The RAG Flow with conditional structure involves selecting different RAG pathways based on different conditions. Typically, this is accomplished through a **Routing module** that determines the route based on query **keywords** or **semantics**.

Different routes are chosen based on the type of question, directing to different flows for specific scenarios. For instance, when users inquire about serious issues, political matters, or entertainment topics, the tolerance for answers from large models varies. Different routing branches usually differ in retrieval sources, retrieval processes, configuration , model , and prompts.



Conditional RAG Flow Pattern

A classic implementation of Conditional RAG is the Semantic Router.

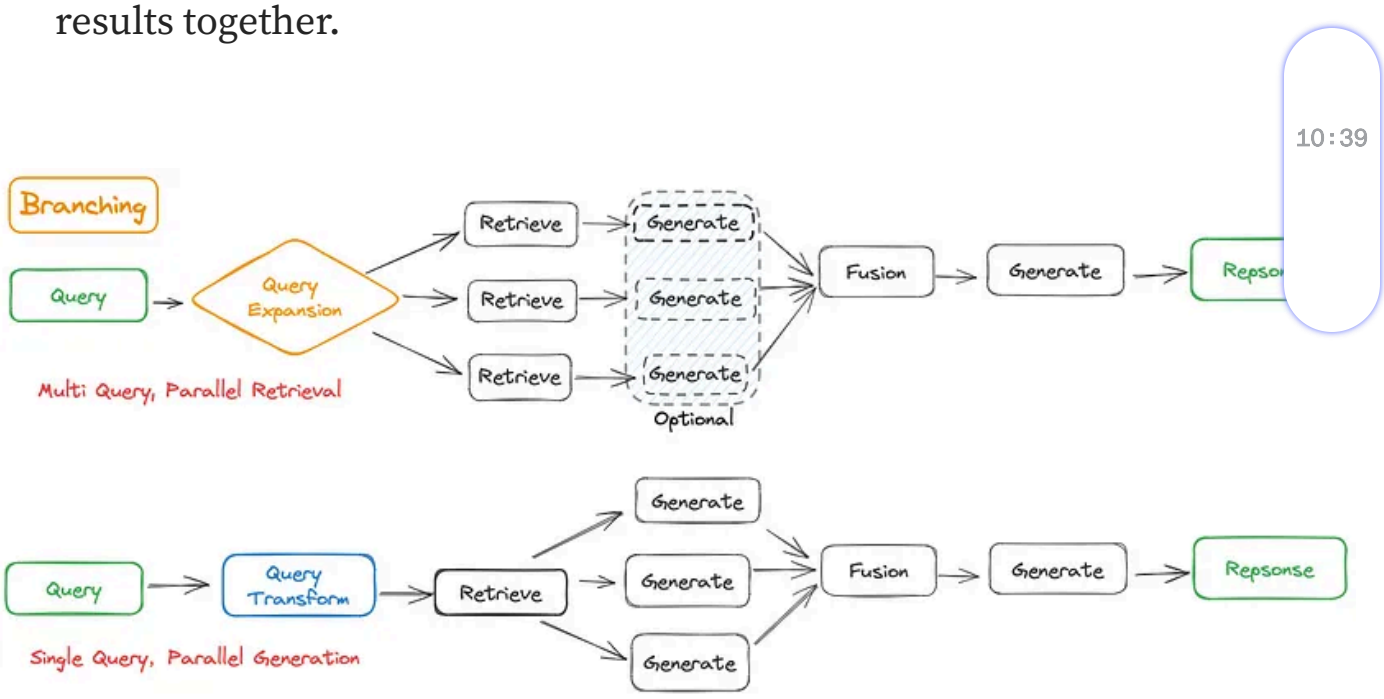
Branching

The RAG Flow with a branching structure differs from the conditional approach in that it involves multiple parallel branches, as opposed to selecting one branch from multiple options in the conditional approach. Structurally, it can be categorized into two types:

- **Pre-Retrieval Branching (Multi-Query, Parallel Retrieval).** This involves expanding the original query to obtain multiple sub-queries, and then

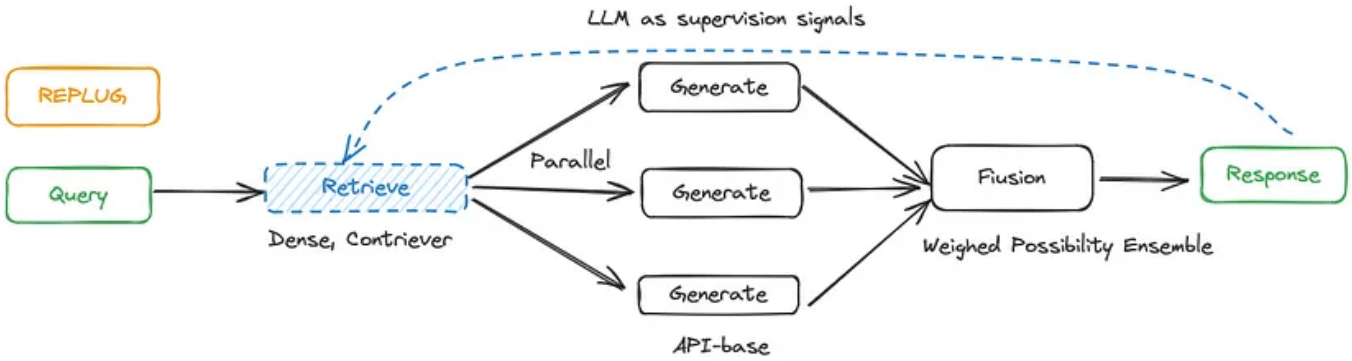
conducting separate retrieval for each sub-query. After retrieval, the approach allows for immediate answer generation based on the sub-questions and the corresponding retrieval content. Alternatively, it may involve using only the expanded retrieval content and merging it into a unified context for generation.

- **Post-Retrieval Branching (Single Query, Parallel Generation).** This approach maintains the original query and retrieves multiple document chunks. Subsequently, it concurrently uses the original query and each document chunks for generation, and finally merges the generated results together.



Branching RAG Flow Pattern

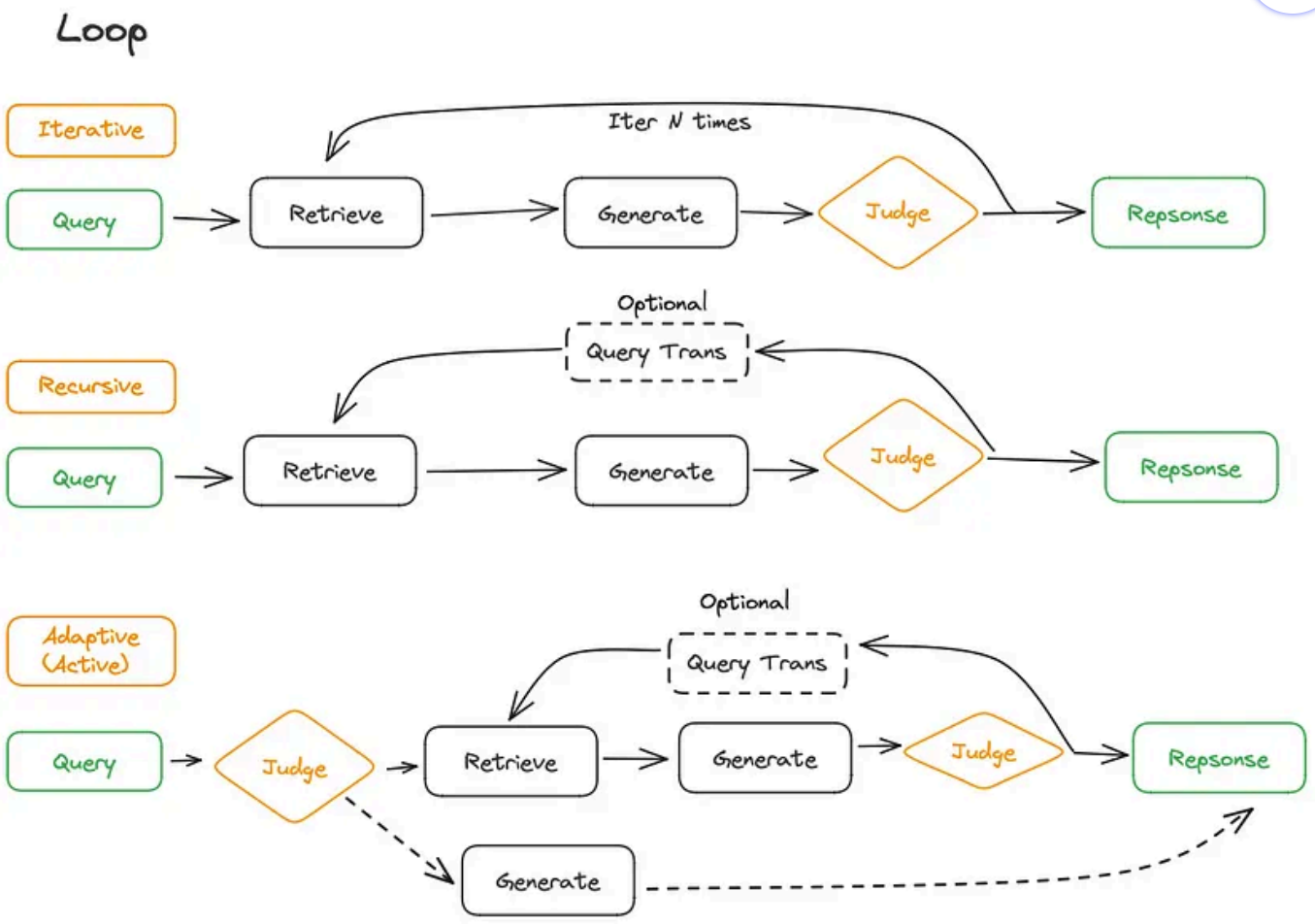
REPLUG embodies a classic post-retrieval branching structure, wherein the probability of each token is predicted for each branch. Through weighted possibility ensemble, the different branches are aggregated, and the final generation result is used to fine-tune the retriever, known as Contriever, through feedback.



RAG Flow in REPLUG

Loop

The RAG Flow with a loop structure, an important characteristic of Modular RAG, involves interdependent retrieval and reasoning steps. It typically includes a **Judge module** for flow control.This can be further categorized into iterative, recursive, and adaptive (active) retrieval approaches.

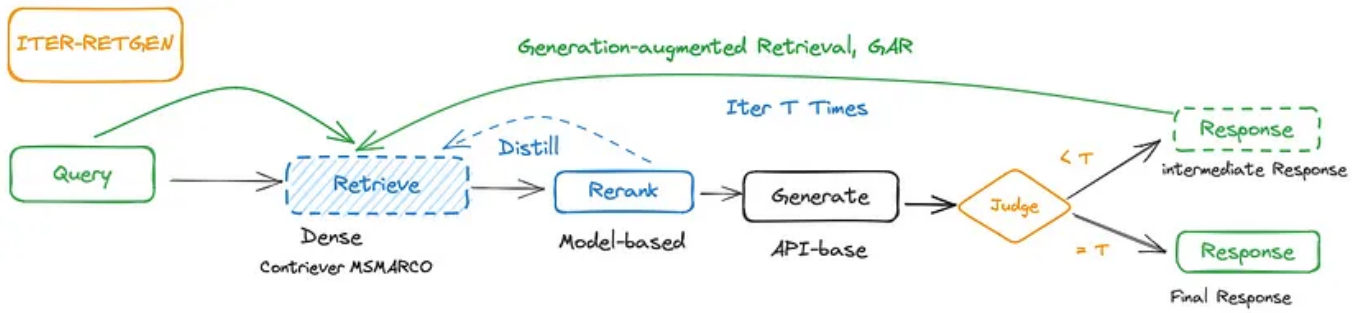


Loop RAG Flow Pattern

Iterative Retrieval

At times, a single retrieval and generation may not effectively address complex questions requiring extensive knowledge. Therefore, an iterative approach can be used in RAG, typically involving a fixed number of iterations for retrieval.

An exemplary case of iterative retrieval is ITER-RETGEN, which iterates retrieval-augmented generation and generation-augmented retrieval. Retrieval-augmented generation outputs a response to a task input based on all retrieved knowledge. In each iteration, ITER-RETGEN leverages the model output from the previous iteration as a specific context to help retrieve more relevant knowledge. Termination of the loop is determined a predefined number of iterations.

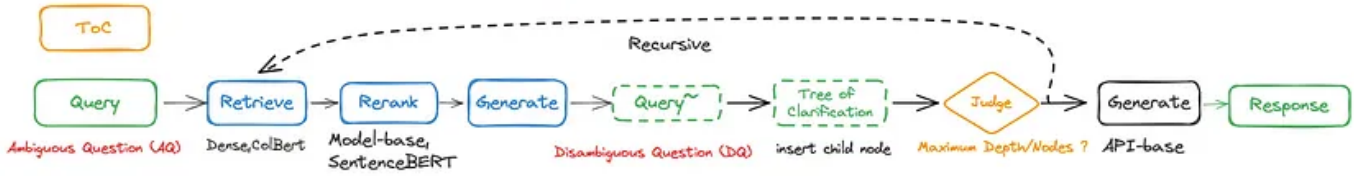


RAG Flow in ITER-RETGEN

Recursive Retrieval

The characteristic feature of recursive retrieval, as opposed to iterative retrieval, is its clear dependency on the previous step and its continuous deepening of retrieval. Typically, there is a termination mechanism as an exit condition for recursive retrieval. In RAG systems, recursive retrieval

usually involves Query Transformation, relying on the newly rewritten query for each retrieval.



RAG Flow in ToC

A typical implementation of recursive retrieval, such as ToC, involves recursively executing RAC (Recursive Augmented Clarification) to gradually insert sub-nodes into the clarification tree from the initial ambiguous question (AQ). At each expansion step, paragraph re-ranking is performed based on the current query to generate a disambiguous Question (DQ). The exploration of the tree concludes upon reaching the maximum number of valid nodes or the maximum depth. Once the clarification tree is constructed, ToC gathers all valid nodes and generates a comprehensive long-text answer to address AQ.

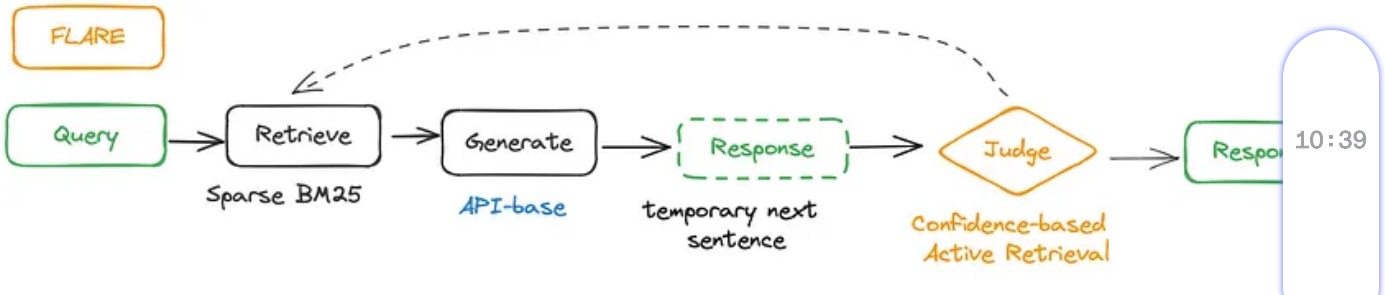
10:39

Adaptive(Active) Retrieval

With the advancement of RAG, there has been a gradual shift beyond passive retrieval to the emergence of adaptive retrieval, also known as proactive retrieval, which is partly attributed to the powerful capabilities of LLM. This shares a core concept with LLM Agent.

RAG systems can actively determine the timing of retrieval and decide when to conclude the entire process and produce the final result. Based on the criteria for judgment, this can be further categorized into Prompt-based and Tuning-based approaches.

- **Prompt-base.**The Prompt-based approach involves controlling the flow using Prompt Engineering to direct LLM. A typical implementation example is FLARE. Its core concept is that the language model should only retrieve when essential knowledge is lacking, to avoid unnecessary or inappropriate retrieval in an enhanced LM. FLARE iteratively generates the next provisional sentence and checks for the presence of low-probability tokens. If found, the system retrieves relevant documents and regenerates the sentence.



RAG Flow in FLARE

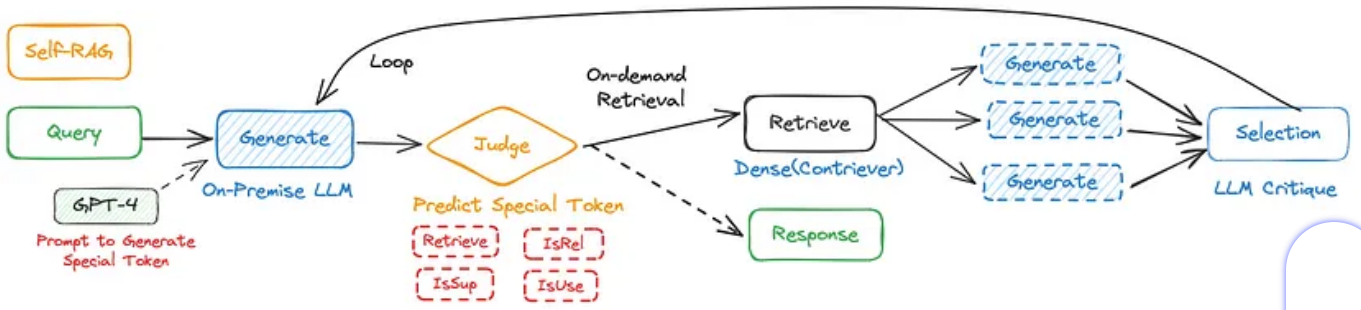
- **Tuning-base.** The Tuning-based approach involves fine-tuning LLM to generate **special tokens**, thereby triggering retrieval or generation. This concept can be traced back to **Toolformer**, where the generation of specific content assists in invoking tools. In RAG systems, this approach is used to control both retrieval and generation steps. A typical case is Self-RAG. Specifically:

1.Given an input prompt and the preceding generation result, first predict whether the special token “Retrieve” is helpful for enhancing the continued generation through paragraph retrieval.

2.If retrieval is needed, the model generates: a critique token to evaluate the retrieved passage’s relevance, the next response segment, and a critique

token to evaluate if the information inthe response segment is supported by the passage.

3.Finally, a critique token evaluates the overall utility of the response and selects the optimal result as the final output.



RAG Flow in Self-RAG

10:39

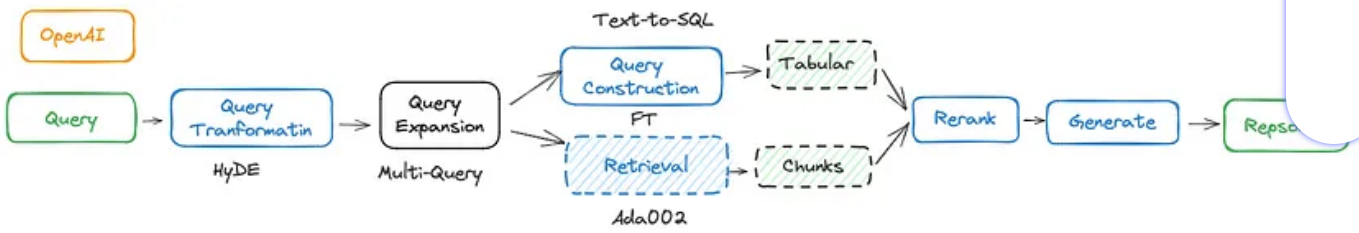
Best Industry Case

In the preceding sections, we have delved into various research papers, with their distinctive feature being an emphasis on addressing specific details and intricacies. RAG, on the other hand, stands out as a technology that shines brightly in the industrial domain, enabling LLM to be applied across a wide range of task scenarios. This chapter will shed light on several industry-leading RAG practices from the perspective of RAG Flow, offering insights into how to effectively combine and construct the flow of RAG in real-world application scenarios.

OpenAI

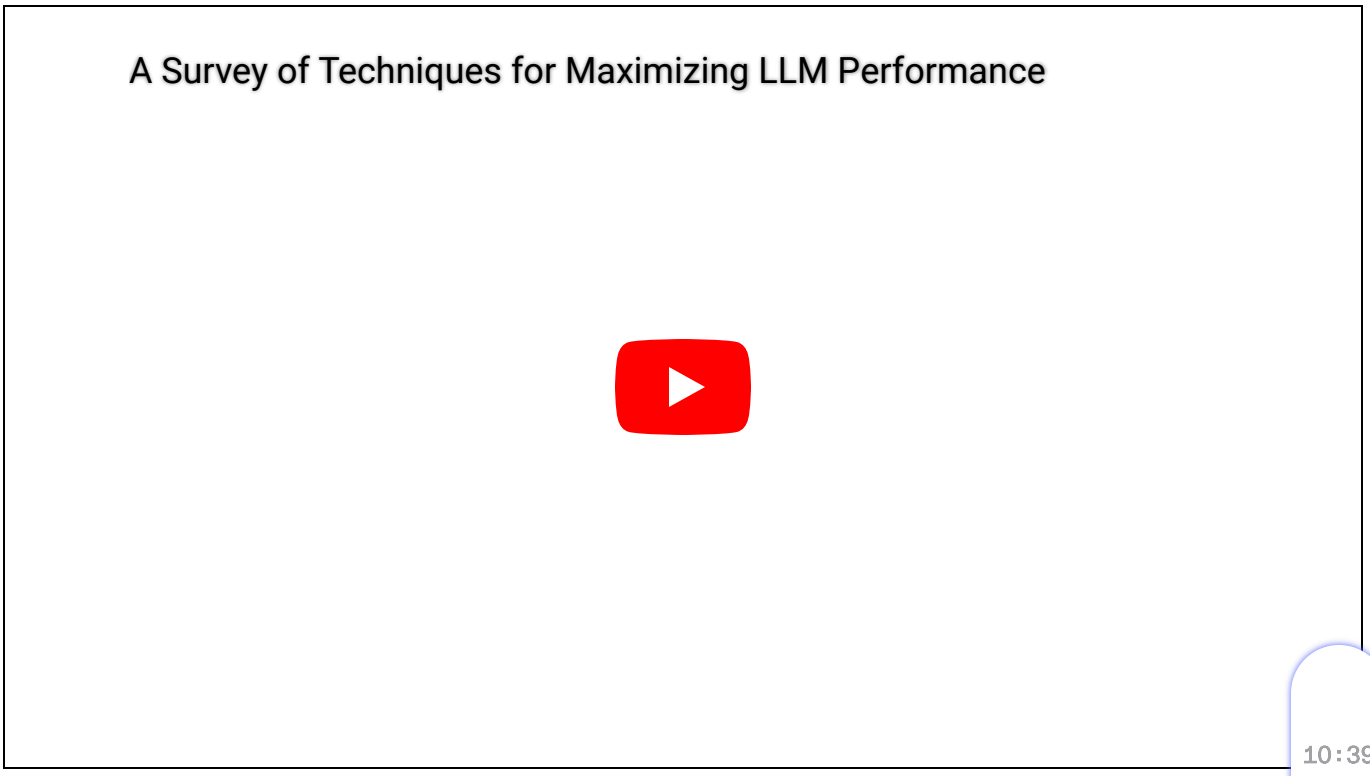
The insights from OpenAI's Demo Day presentation do not fully represent the actual operations of OpenAI.

In their efforts to enhance the success of RAG, the OpenAI team started with a 45% accuracy rate and experimented with various methods, identifying which methods were ultimately adopted for production. They explored hypothetical document embeddings (HyDE), fine-tuning embeddings, and other methods, but the results were not satisfactory. By experimenting with different-sized chunks of information and embedding different content sections, they were able to increase the accuracy to 65%. Through reranking and methods tailored to handle different types of questions, they further improved the accuracy to 85%. Ultimately, by combining prompt engineering, query expansion, and other methods, they achieved a 98% accuracy rate.



OpenAI RAG Flow

The team emphasized the powerful potential of model fine-tuning and the integration of RAG, particularly in approaching industry-leading levels without the use of complex techniques, solely through simple model fine-tuning and prompt engineering.



The Origin OpenAI Demo

Baichuan

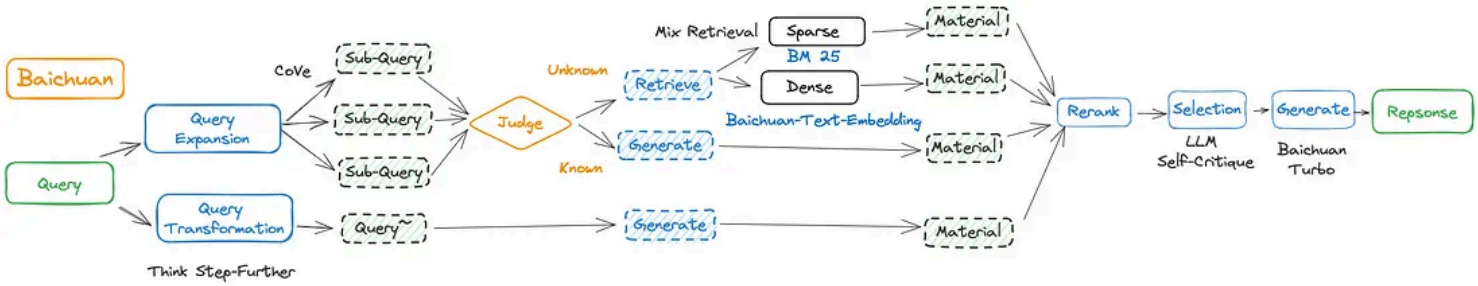
Based on the publicly available information from various sources, the available data is limited, and the author has made some speculative assumptions about certain details. See the original (in Chinese)

Baichuan, drawing inspiration from Meta's **CoVe,** has devised a method to deconstruct complex prompts into multiple independent and parallel retrievable search-friendly queries. This enables large models to conduct targeted knowledge base searches for each sub-query, thereby providing more accurate and detailed answers and reducing spurious outputs. Additionally, they have leveraged their proprietary **TSF (Think-Step Further)** to infer and unearth the deeper underlying questions behind user input, allowing for a more precise and comprehensive understanding of user intent. While the technical details of TSF have not been disclosed, it is speculated to be an enhancement of the Step-back prompting method.

In the retrieval step, Baichuan Intelligence has developed the Baichuan-Text-Embedding vector model, pre-trained on high-quality Chinese data comprising over 1.5 trillion tokens. They have addressed the issue of batch size dependency in contrastive learning through a proprietary loss function. This vector model has **surpassed the C-MTEB.**

Additionally, they have introduced **sparse** retrieval and **rerank** models (not disclosed.), forming a **hybrid retrieval** approach that combines vector retrieval with sparse retrieval in parallel, significantly enhancing the recall rate to 95%.

Furthermore, they have introduced **self-critique**, enabling large models t introspect on the retrieved content based on prompt, relevance, and utili and undergo a secondary review to select the most matching and high-quality candidate content.



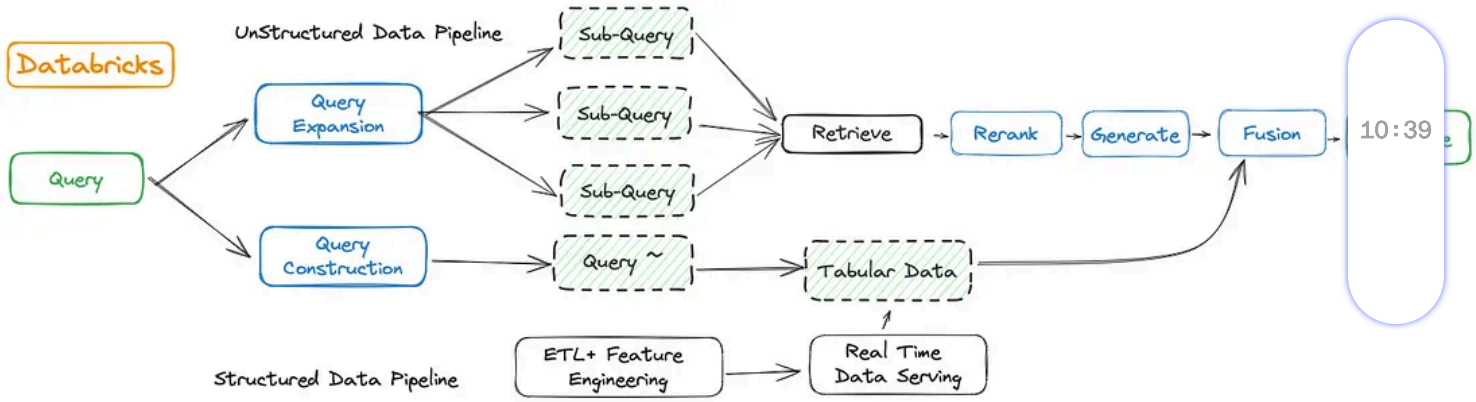
Baichuan RAG Flow

Given the numerous branches in the entire Baichuan RAG Flow and the lack of specific disclosure, it is reasonable to speculate that reranking and selection entail reordering and screening of all materials, whether retrieved or generated from other branches.

Databricks

Databricks, as a leading service provider in the big data domain, has maintained its distinctive features and advantages in RAG design.

When a user inputs a question, the system retrieves relevant information from pre-processed text vector indices, incorporating prompt engineering to generate responses. The upper half, the **Unstructured Data Pipeline**, follows the mainstream RAG approach and does not exhibit any particular uniqueness.



Databricks RAG Flow

The lower half, the **Structured Data Pipeline**, represents Databricks' feature engineering process and is the most significant aspect of Databricks' RAG implementation. Leveraging its expertise in big data, Databricks conducts additional retrieval from its highly accurate data storage, fully utilizing its advantage in **Real Time Data Serving.** It is evident that Databricks' strategy in the era of GenAI is to empower RAG applications with broad market demand, integrating its robust Delta lake processing capabilities with generative AI technology to build an integrated solution, and promoting this unified service to its customers.

Retrieval Augmented Generation (RAG) on Databricks

Learn about retrieval augmented generation (RAG) on Databricks to achieve greater large language model (LLM) accuracy...

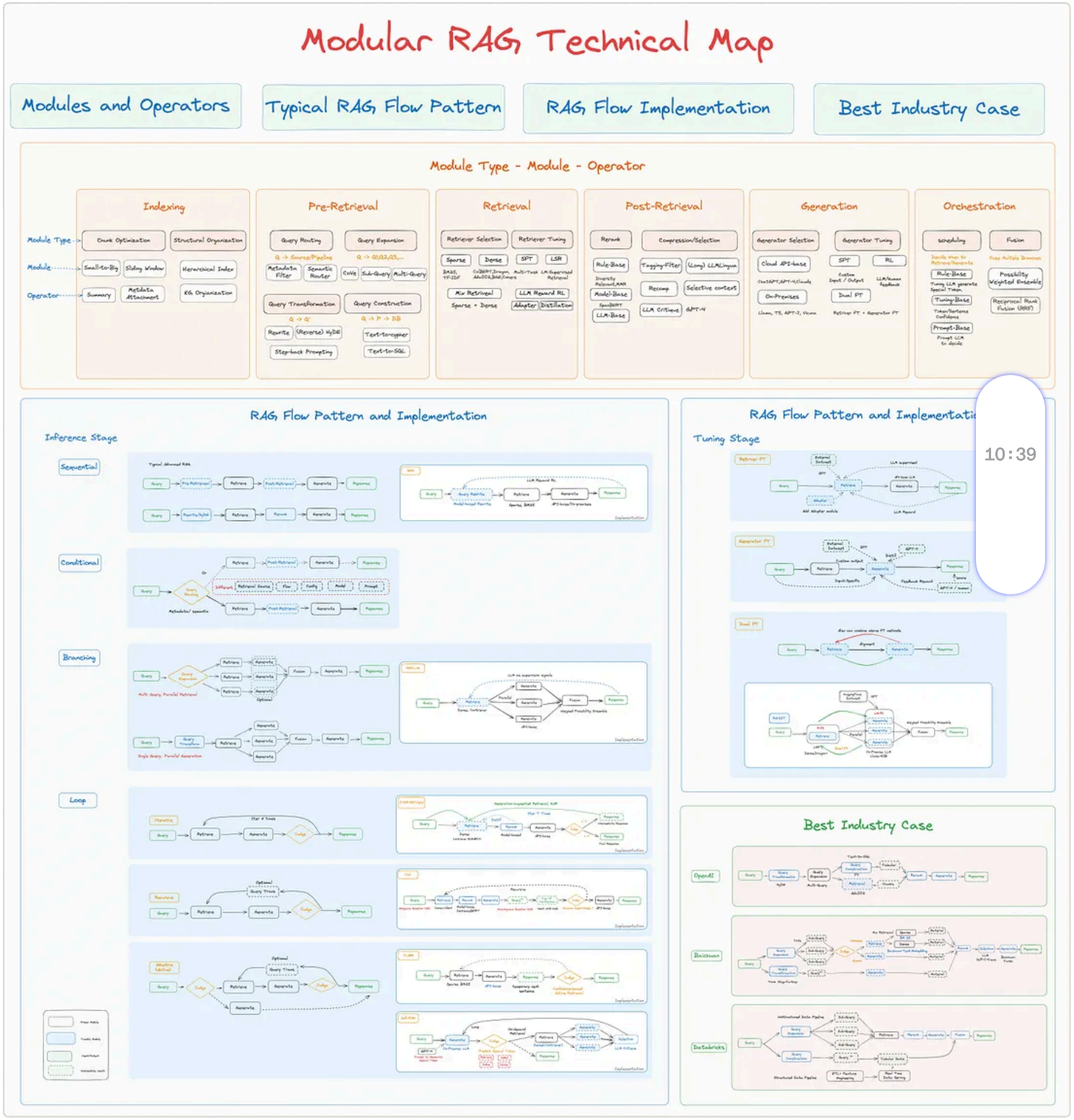
docs.databricks.com

Conclusion

The article delineates three patterns of fine-tuning stages, four patterns of inference stages, as well as the specific flow implementations in seven papers and three industrial best practices. The overall framework is illustrated as follows.

10:39

As we also mentioned in Part 1, summarization and abstraction of the RA paradigm are crucial in this era of rapid technological advancement. It is essential to transcend specific implementations and comprehend the current technological features and trends from a higher dimension, in order to grasp the direction of future development.



Modular RAG Technical Map

Large Language Models | Rag | Modular Rag

Written by Yunfan Gao

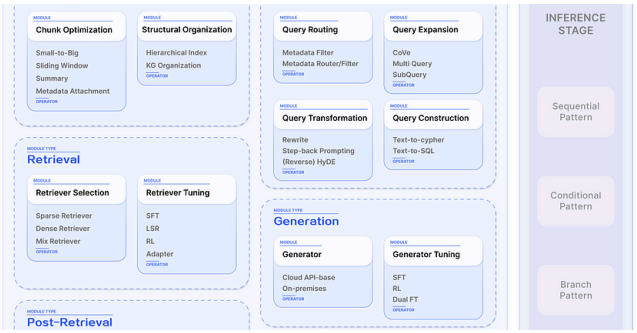
89 Followers

Cs PhD Candidate in Tongji University

Follow

10 : 39

More from Yunfan Gao

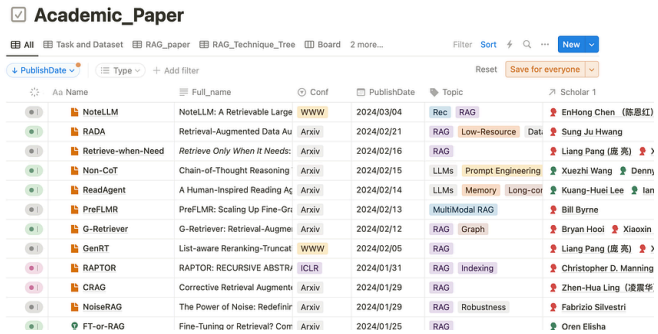


Yunfan Gao

Modular RAG and RAG Flow: Part I

A compressive and high-level summarization of RAG .

18 min read · Jan 24, 2024



Yunfan Gao

OpenRAG Base: Your individual RAG Knowledge Base

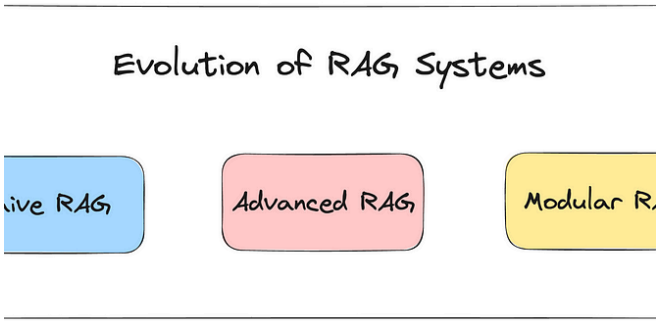
We’re officially launching the RAG Knowledge Base: OpenRAG Base 🎉✨

9 min read · Apr 3, 2024

171 3 16

See all from Yunfan Gao

Recommended from Medium



Dr Julija Bainiaksina

What are Naive RAG, Advanced RAG & Modular RAG Paradigms?

My key learnings on how RAG systems evolved over the years. I share an overview o...

6 min read · Mar 10, 2024

50 1

Tejpal Kumawat

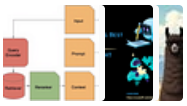
Retrieval-Augmented Generation (RAG) from basics to advanced

Introduction:

12 min read · Feb 14, 2024

81

Lists



Natural Language Processing

1417 stories · 910 saves

AI Regulation

6 stories · 430 saves

ChatGPT prompts

47 stories · 1496 saves

Generative AI Recommended Reading

52 stories · 984 saves

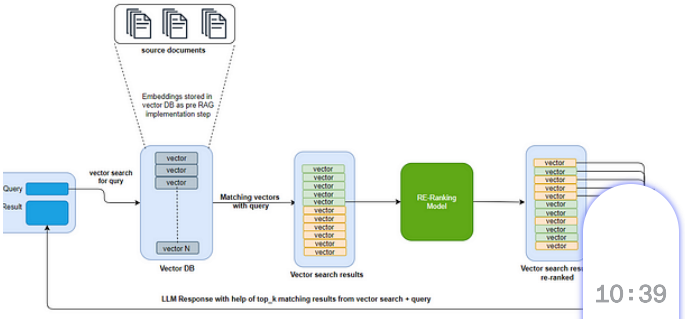
IVAN ILIN in Towards AI

Advanced RAG Techniques: an Illustrated Overview

A comprehensive study of the advanced retrieval augmented generation techniques...

19 min read · Dec 17, 2023

5.5K 36



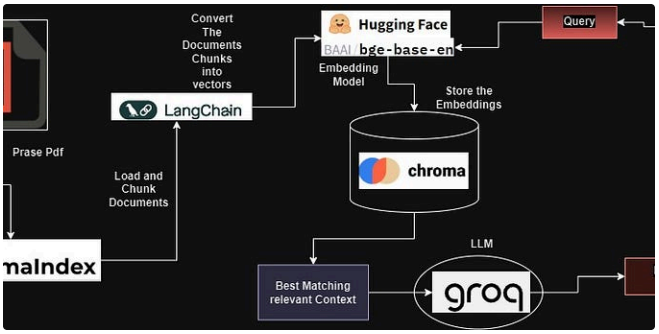
ASHPAK MULANI

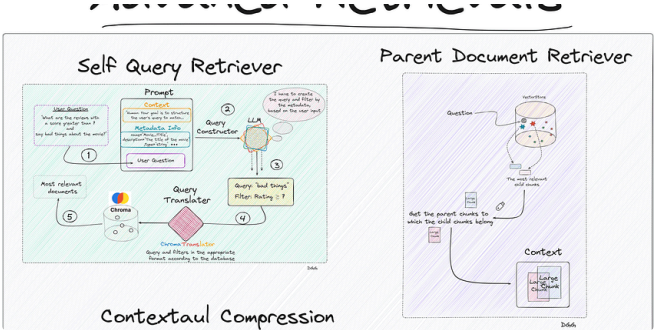
Improve Retrieval Augmented Generation (RAG) with Re-ranking

In the world of GenAI, you'll often come across the term RAG (Retrieval augmented...

7 min read · Feb 24, 2024

13 1





Plaban Nayak in The AI Forum

RAG on Complex PDF using LlamaParse, Langchain and Groq

Damian Gil in Towards Data Science

Advanced Retriever Techniques to Improve Your RAGs

Retrieval-Augmented Generation (RAG) is a new approach that leverages Large Languag...

13 min read · Apr 7, 2024

340 3

Master Advanced Information Retrieval: Cutting-edge Techniques to Optimize the...

18 min read · Apr 17, 2024

620 4

See more recommendations