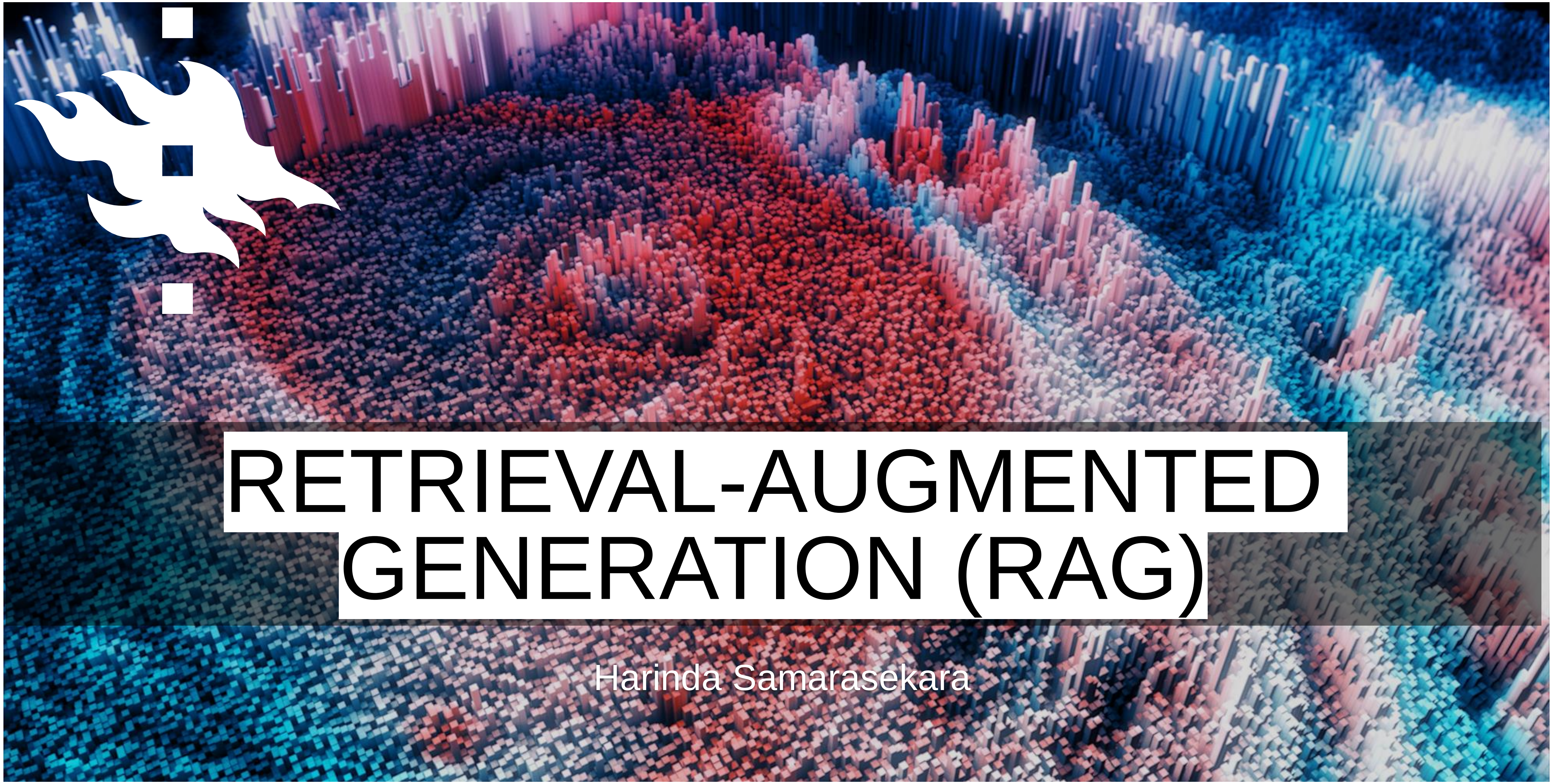


RETRIEVAL-AUGMENTED GENERATION (RAG)

Harinda Samarasekara

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

THE PROBLEM!?

- Large Language Models (LLMs) encounter challenges like;
  - Hallucination
  - Outdated knowledge
  - None transparent, untraceable reasoning process

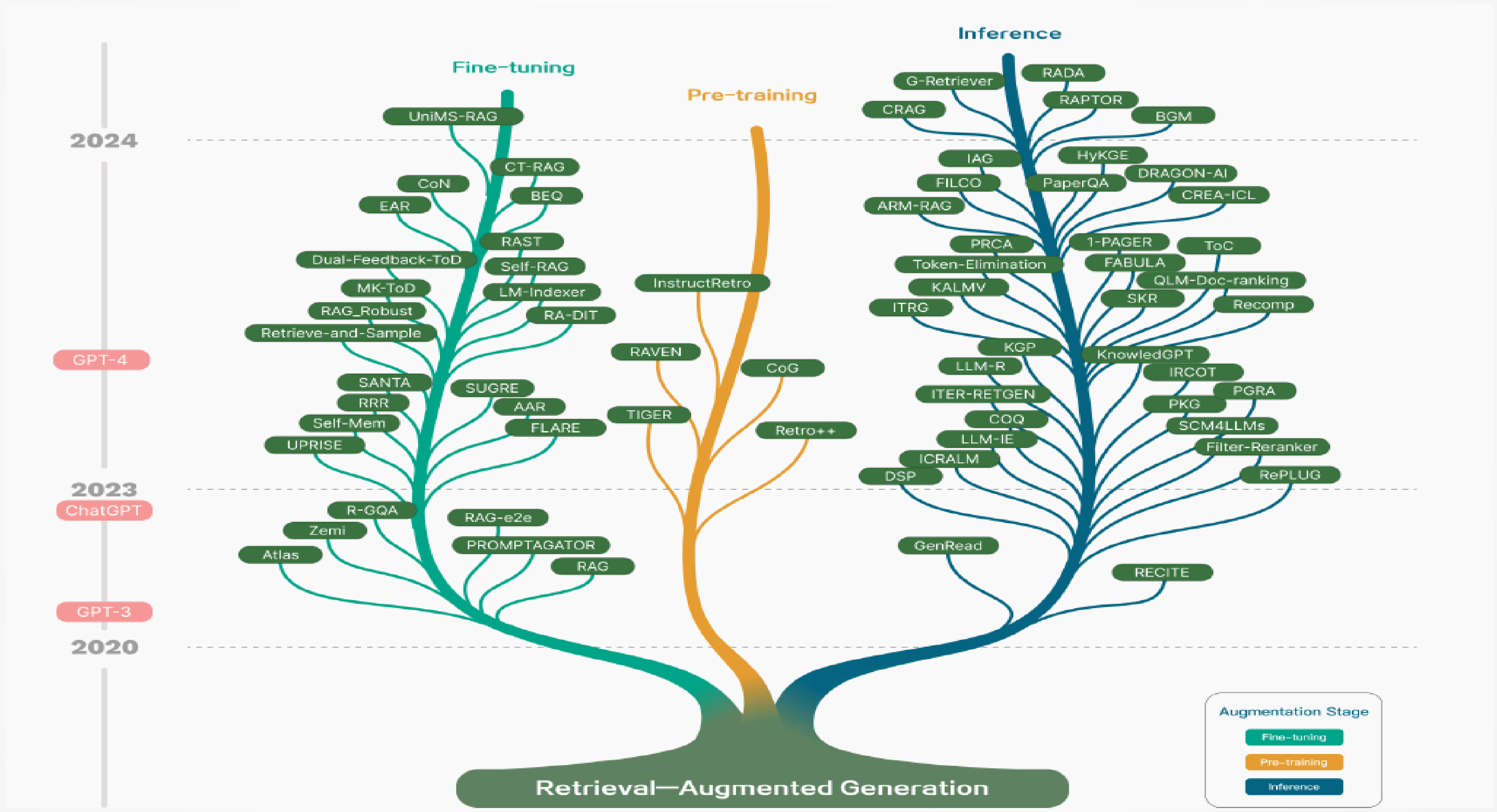
❖ Approach to address these challenges? → Incorporating knowledge from external sources

❖ RAG facilitates this!

WHAT IS RAG?

R	→	Retrieve relevant contextual information from sources outside of the LLM’s memory (Dense Retrievals, Sparse Retrievals)
A	→	Augment the retrieved outside context into the LLM’s memory
G	→	Generate final response using the retrieved and augmented information

RAG – AUGMENTATION STAGES

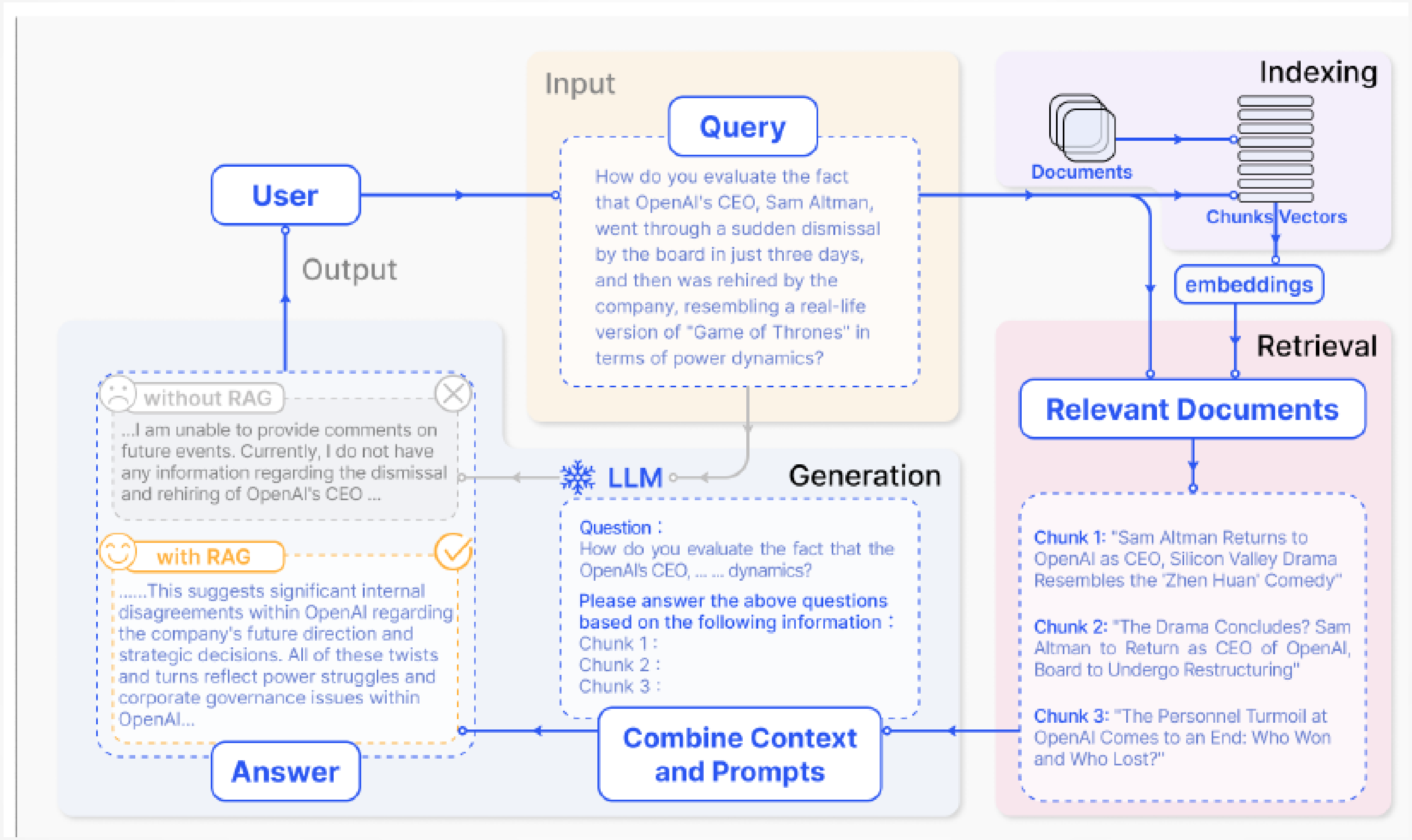


NAÏVE RAG - “RETRIEVE-READ” FRAMEWORK

1. **Indexing** - extraction of raw data in diverse formats, conversion to a uniform plain-text format, chunk, store as vector embeddings
2. **Retrieval** – for a specific user query, find the most relevant information from the database, add the retrieved data to expand and enrich the prompt
3. **Generation** – The large language model is tasked with formulating a response using the context provided

Notable Drawbacks

- Retrieval Challenges
- Generation Difficulties
- Augmentation Hurdles


Input
Query
How do you evaluate the fact that OpenAI's CEO, Sam Altman, went through a sudden dismissal by the board in just three days, and then was rehired by the company, resembling a real-life version of "Game of Thrones" in terms of power dynamics?
Indexing
Documents
Chunks
Vectors
embeddings
User
Output
Retrieval
Relevant Documents
without RAG
...I am unable to provide comments on future events. Currently, I do not have any information regarding the dismissal and rehiring of OpenAI's CEO ...
LLM
Generation
with RAG
......This suggests significant internal disagreements within OpenAI regarding the company's future direction and strategic decisions. All of these twists and turns reflect power struggles and corporate governance issues within OpenAI...
Question :
How do you evaluate the fact that the OpenAI's CEO, ... dynamics?
Please answer the above questions based on the following information :
Chunk 1 :
Chunk 2 :
Chunk 3 :
Chunk 1: "Sam Altman Returns to OpenAI as CEO, Silicon Valley Drama Resembles the 'Zhen Huan' Comedy"
Chunk 2: "The Drama Concludes? Sam Altman to Return as CEO of OpenAI, Board to Undergo Restructuring"
Chunk 3: "The Personnel Turmoil at OpenAI Comes to an End: Who Won and Who Lost?"
Answer
Combine Context and Prompts

ADVANCED RAG

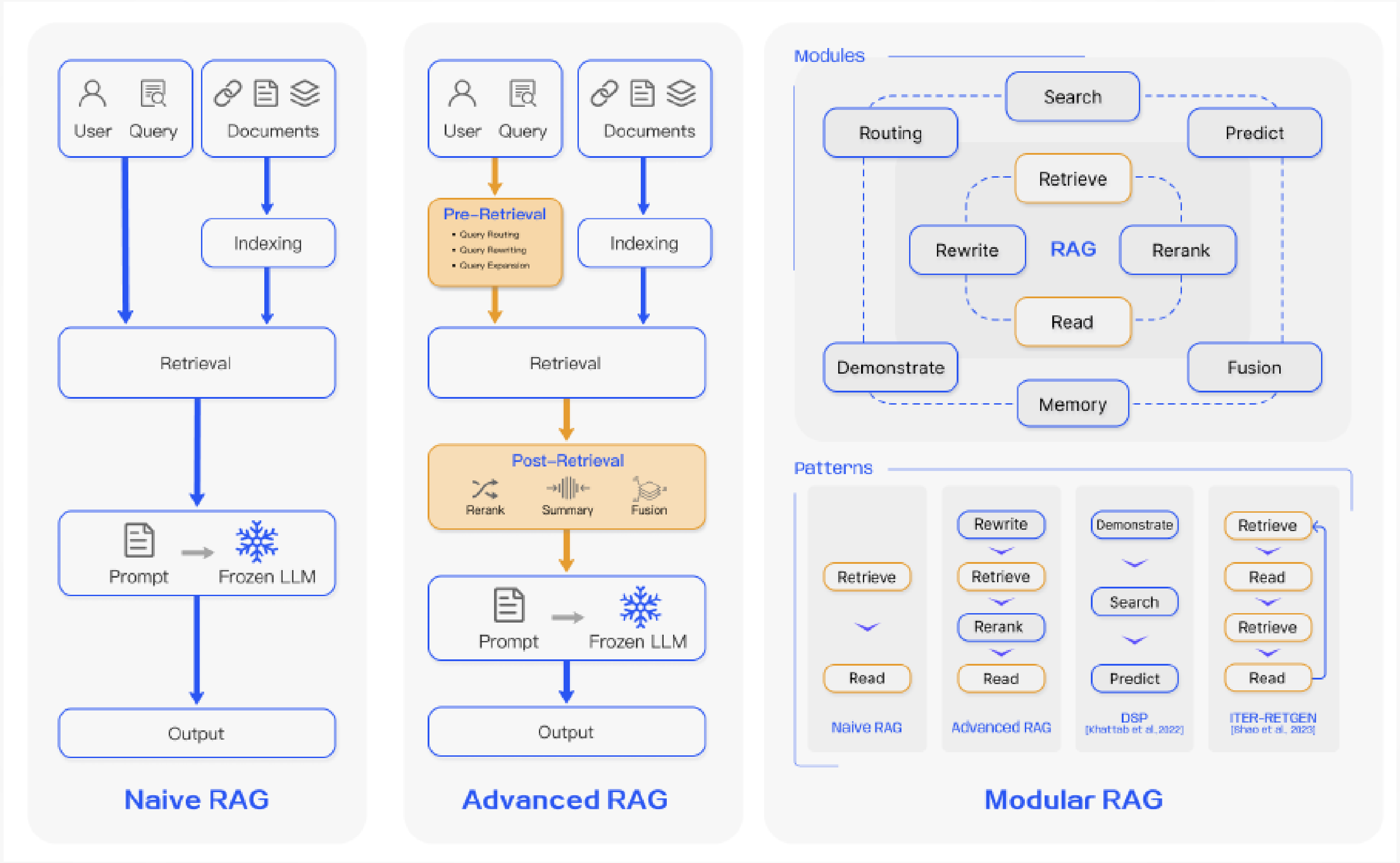
Employs pre-retrieval and post-retrieval strategies.

Pre-retrieval Process	Post-retrieval Process
Primary focus is on optimizing the indexing structure and the original query	After retrieving the query, integrate it effectively with the query.
make the user’s original question clearer and more suitable for the retrieval task	Feeding all relevant documents directly into LLMs can lead to information overload, diluting the focus on key details with irrelevant content.
i.e. - query rewriting query transformation, query expansion and other techniques	i.e. rerank chunks and context compressing.

MODULAR RAG

- Tries to achieve enhanced adaptability and versatility through extra modules and patterns
- Modules –
  - **Search module**
  - **RAGFusion**
  - **Predict module**
- Patterns –
  - **Rewrite-Retrieve-Read**
  - **Generate-Read**
  - **Demonstrate-Search-Predict / ITERRETGEN**

Another benefit of a flexible architecture is that the RAG system can more easily integrate with other technologies (such as fine-tuning or reinforcement learning)



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

AUGMENTATION PROCESS IN RAG

The standard practice of singular retrieval step followed by generation is insufficient in complex problems demanding multi-step reasoning

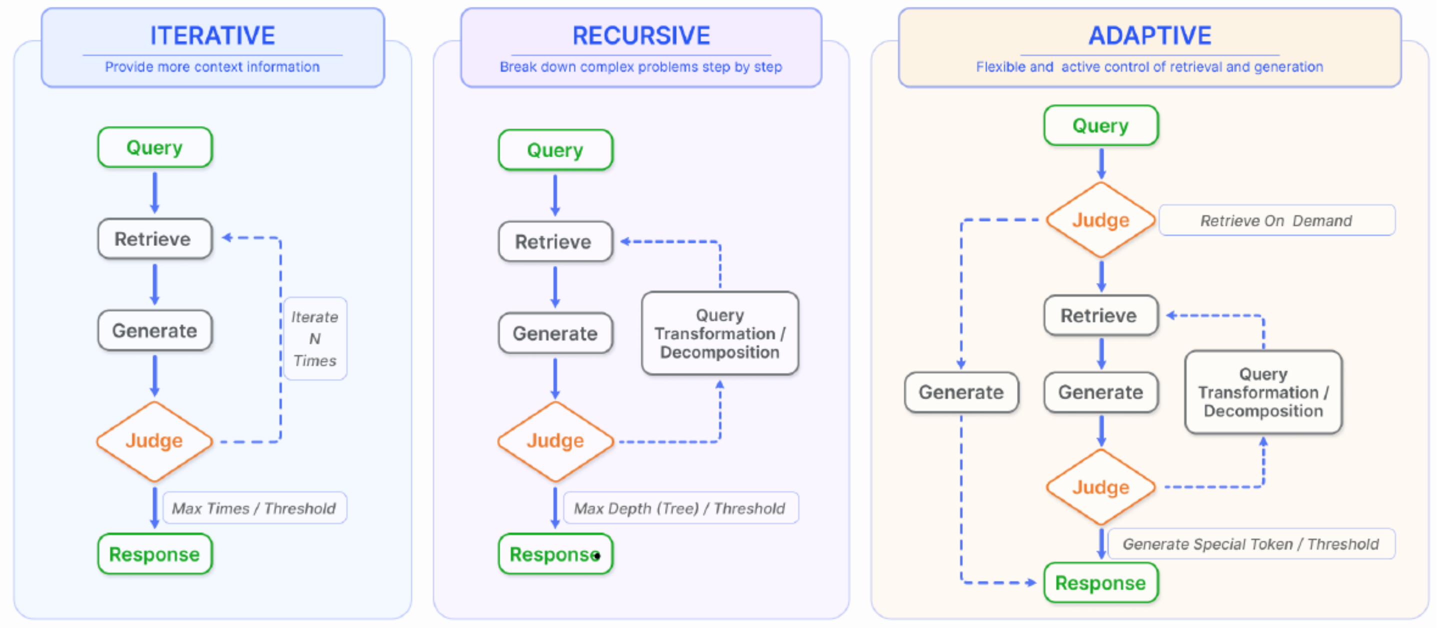
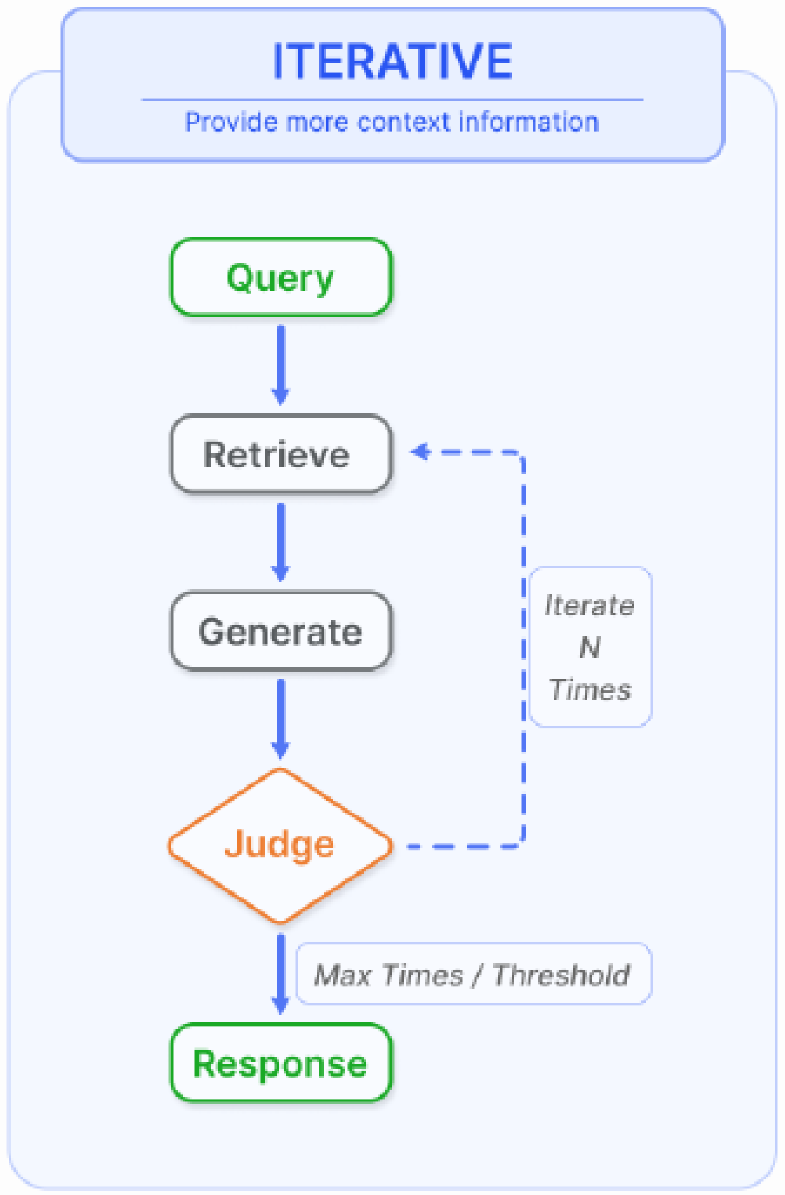

ITERATIVE
Provide more context information
Query
Retrieve
Generate
Judge
Iterate N Times
Max Times / Threshold
Response
RECURSIVE
Break down complex problems step by step
Query
Retrieve
Generate
Query Transformation / Decomposition
Judge
Max Depth (Tree) / Threshold
Response
ADAPTIVE
Flexible and active control of retrieval and generation
Query
Judge
Retrieve On Demand
Retrieve
Generate
Generate
Query Transformation / Decomposition
Judge
Generate Special Token / Threshold
Response

Fig. 5. In addition to the most common once retrieval, RAG also includes three types of retrieval augmentation processes. (left) Iterative retrieval involves alternating between retrieval and generation, allowing for richer and more targeted context from the knowledge base at each step. (Middle) Recursive retrieval involves gradually refining the user query and breaking down the problem into sub-problems, then continuously solving complex problems through retrieval and generation. (Right) Adaptive retrieval focuses on enabling the RAG system to autonomously determine whether external knowledge retrieval is necessary and when to stop retrieval and generation, often utilizing LLM-generated special tokens for control.

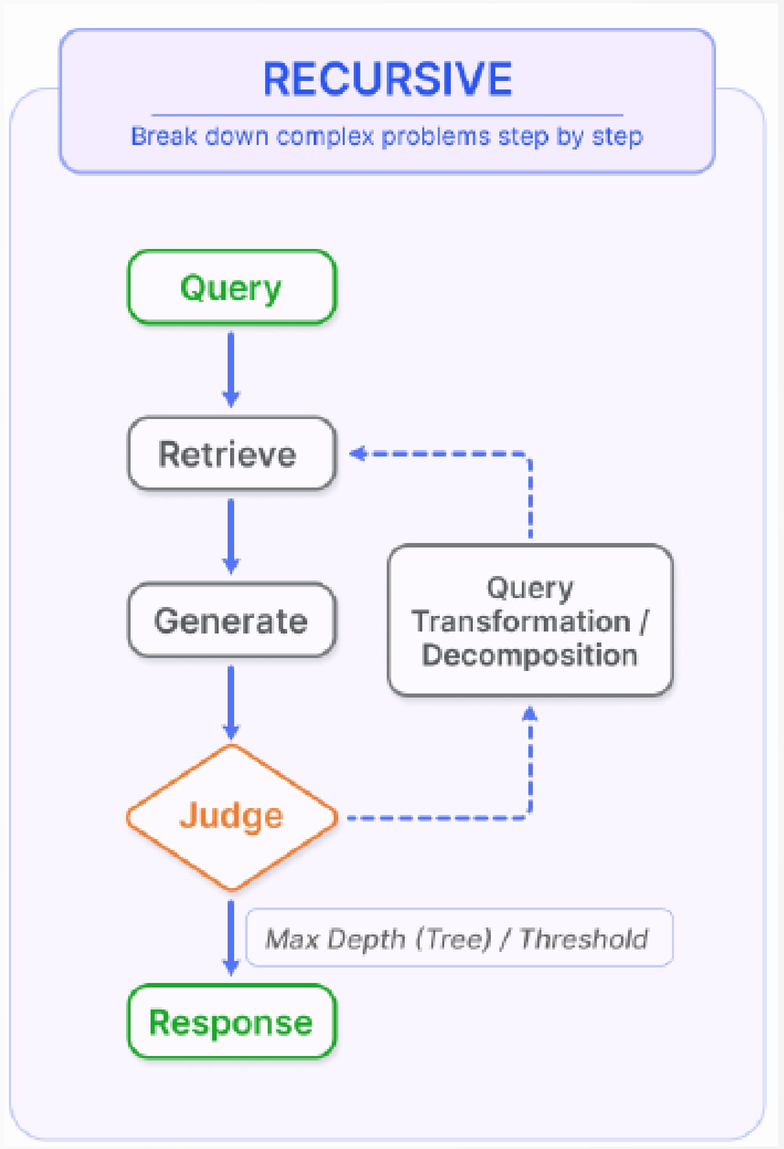
ITERATIVE RETRIEVAL

- The knowledge base is repeatedly searched based on the initial query and the text generated so far
- enhance the robustness of subsequent answer generation by offering additional contextual references through multiple retrieval iterations
- may be affected by semantic discontinuity and the accumulation of irrelevant information.



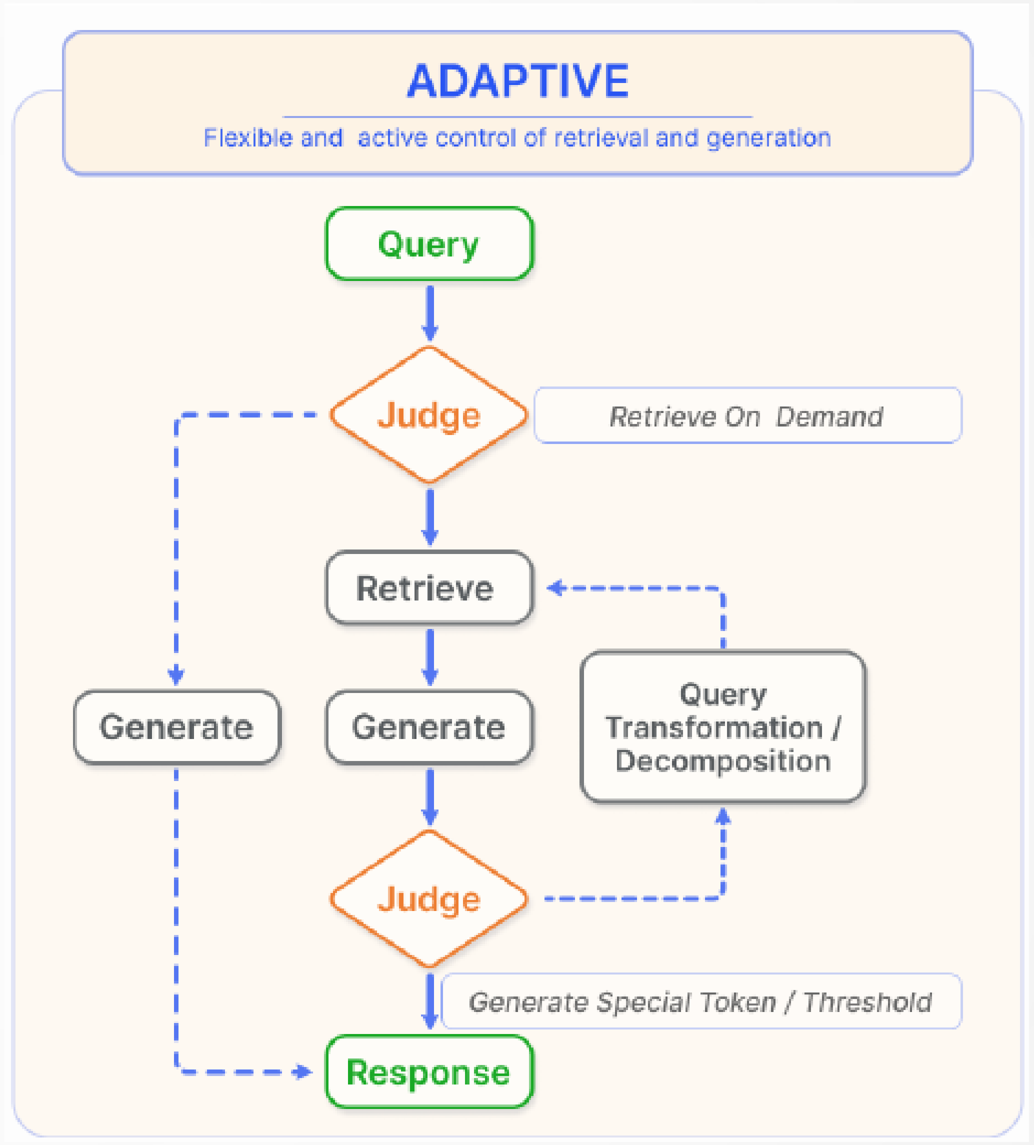
RECURSIVE RETRIEVAL

- The knowledge base is repeatedly searched based on the initial query and the text generated so far
- Beneficial when user's search query is not clear or the sought information is highly nuanced.


RECURSIVE
Break down complex problems step by step
Query
Retrieve
Generate
Query Transformation / Decomposition
Judge
Max Depth (Tree) / Threshold
Response

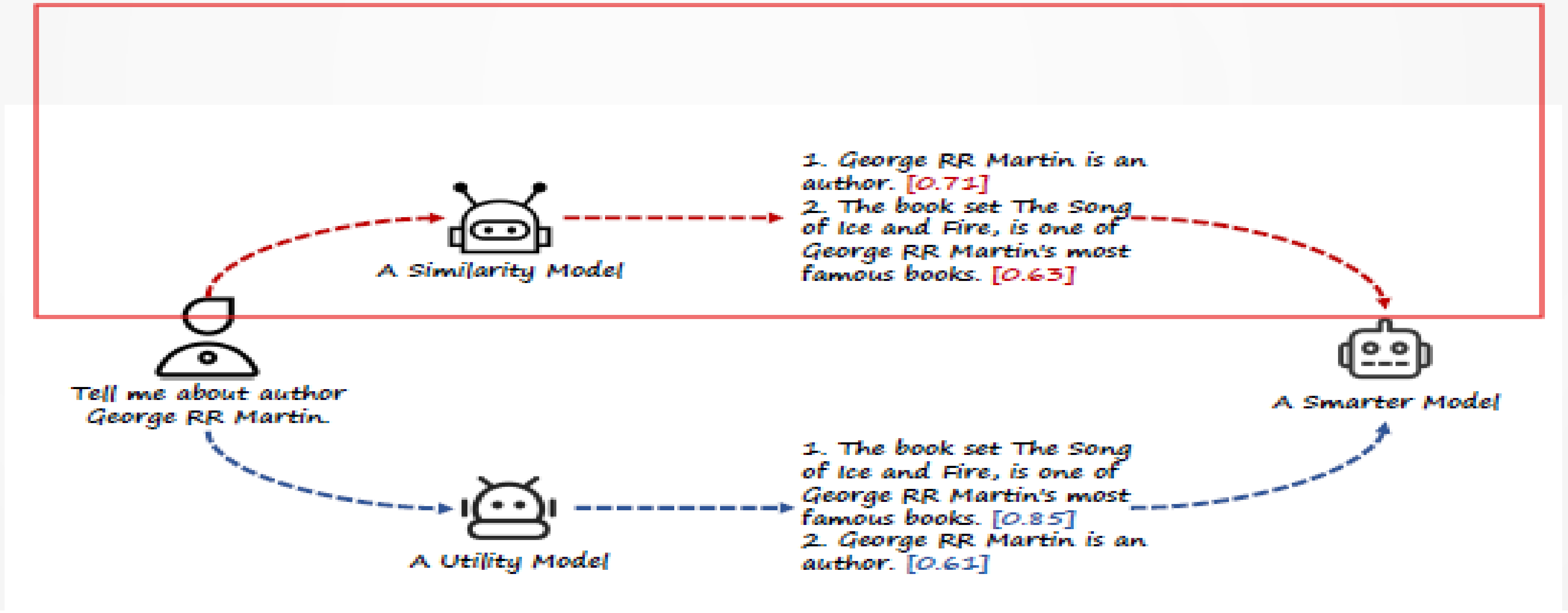
ADAPTIVE RETRIEVAL

- Enables LLMs to actively determine the optimal moments and content for retrieval.
- Retrieval process into distinct steps where LLMs proactively use retrievers, apply Self-Ask techniques, and employ few-shot prompts to initiate search queries
- Improves Model’s autonomous judgement capabilities


ADAPTIVE
Flexible and active control of retrieval and generation
Query
Judge
Retrieve On Demand
Retrieve
Generate
Generate
Query Transformation / Decomposition
Judge
Generate Special Token / Threshold
Response

RETRIEVING RELYING ON SIMILARITY

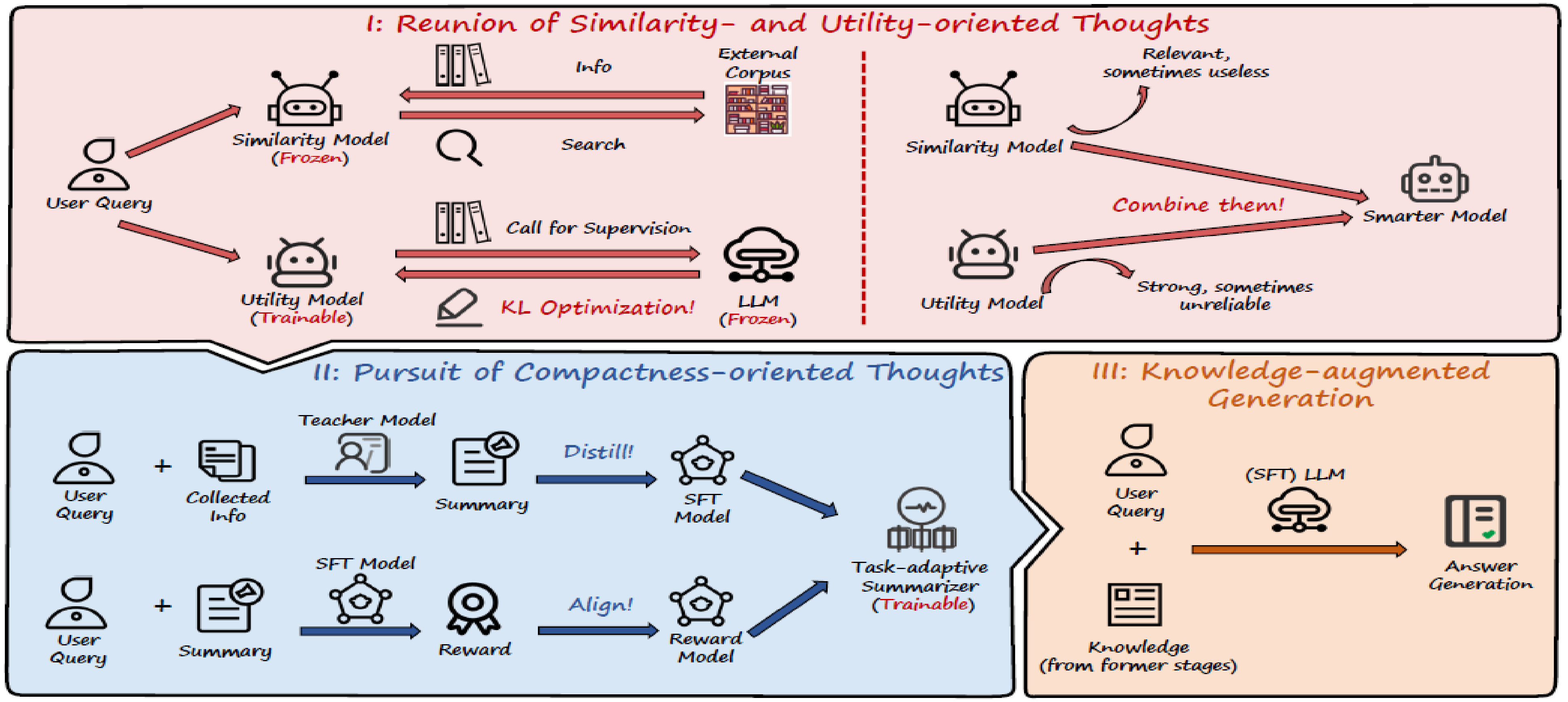
- Existing retrieval-augmented generation approaches are typically similarity-based.
- How ever relying completely on similarity would sometimes degrade the performance of retrieval-augmented generation



Tell me about author George RR Martin
A Similarity Model
1. George RR Martin is an author. [0.71]
2. The book set The Song of Ice and Fire, is one of George RR Martin's most famous books. [0.63]
A Smarter Model
A Utility Model
1. The book set The Song of Ice and Fire, is one of George RR Martin's most famous books. [0.85]
2. George RR Martin is an author. [0.61]

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

METRAG FRAMEWORK

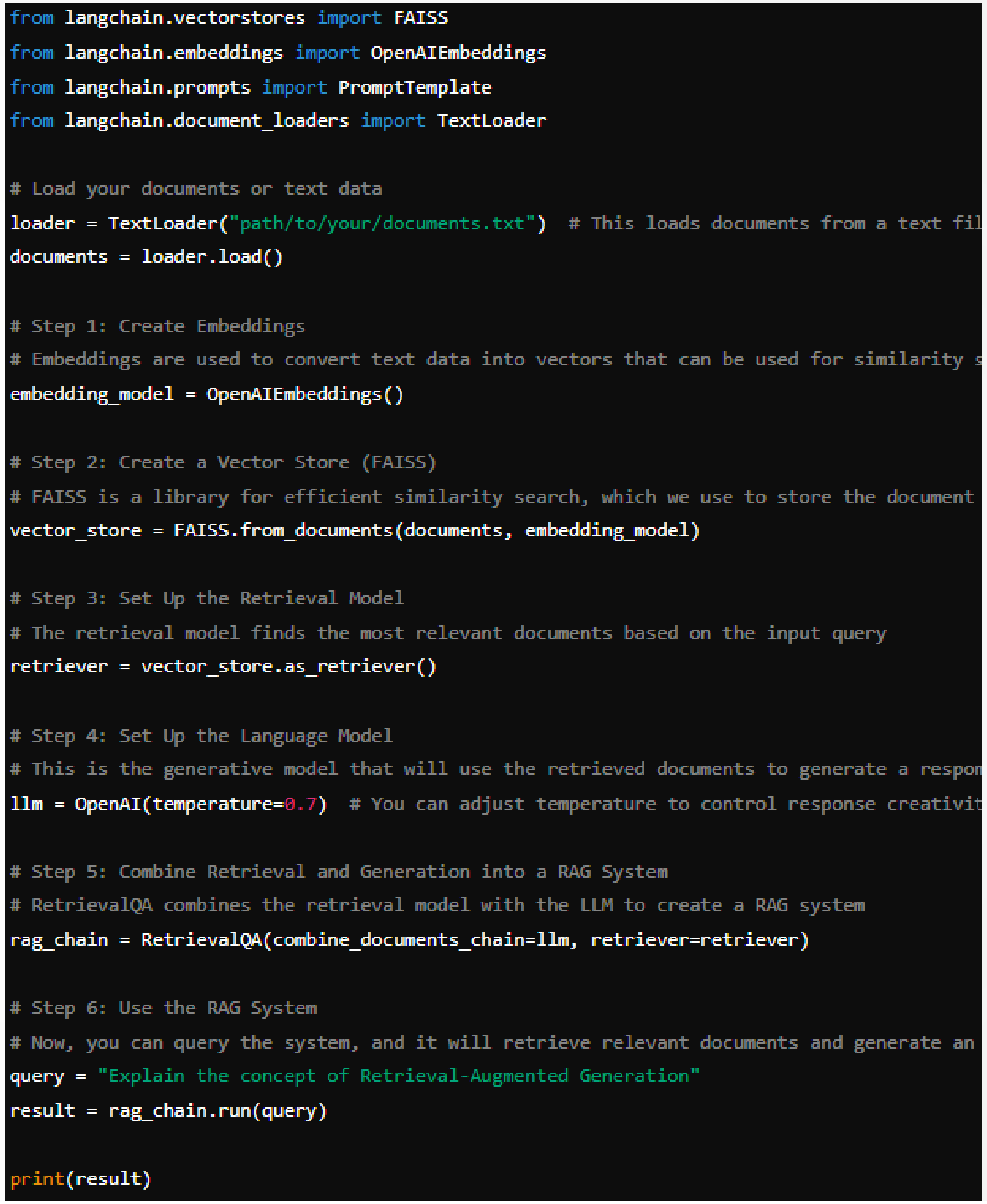


I: Reunion of Similarity- and Utility-oriented Thoughts
User Query
Similarity Model (Frozen)
Info
Search
External Corpus
Utility Model (Trainable)
Call for Supervision
KL Optimization!
LLM (Frozen)
Relevant, sometimes useless
Similarity Model
Combine them!
Smarter Model
Utility Model
Strong, sometimes unreliable
II: Pursuit of Compactness-oriented Thoughts
User Query
+
Collected Info
Teacher Model
Summary
Distill!
SFT Model
Task-adaptive Summarizer (Trainable)
User Query
+
Summary
SFT Model
Reward
Align!
Reward Model
III: Knowledge-augmented Generation
User Query
+
Knowledge (from former stages)
(SFT) LLM
Answer Generation

Figure 2: The proposed METRAG framework, where we endow retrieval-augmented generation with multi-layered thoughts from Stage I and II, and utilize the derived knowledge in Stage III for answer generation.

LANGCHAIN FRAMEWORK FOR RAG

A powerful framework designed to facilitate the development of applications that integrate large language models (LLMs) with external data sources, APIs, and databases.



```python
from langchain.vectorstores import FAISS
from langchain.embeddings import OpenAIEmbeddings
from langchain.prompts import PromptTemplate
from langchain.document_loaders import TextLoader

# Load your documents or text data
loader = TextLoader("path/to/your/documents.txt")  # This loads documents from a text fil
documents = loader.load()

# Step 1: Create Embeddings
# Embeddings are used to convert text data into vectors that can be used for similarity s
embedding_model = OpenAIEmbeddings()

# Step 2: Create a Vector Store (FAISS)
# FAISS is a library for efficient similarity search, which we use to store the document
vector_store = FAISS.from_documents(documents, embedding_model)

# Step 3: Set Up the Retrieval Model
# The retrieval model finds the most relevant documents based on the input query
retriever = vector_store.as_retriever()

# Step 4: Set Up the Language Model
# This is the generative model that will use the retrieved documents to generate a respon
llm = OpenAI(temperature=0.7)  # You can adjust temperature to control response creativit

# Step 5: Combine Retrieval and Generation into a RAG System
# RetrievalQA combines the retrieval model with the LLM to create a RAG system
rag_chain = RetrievalQA(combine_documents_chain=llm, retriever=retriever)

# Step 6: Use the RAG System
# Now, you can query the system, and it will retrieve relevant documents and generate an
query = "Explain the concept of Retrieval-Augmented Generation"
result = rag_chain.run(query)

print(result)
```

LANGCHAIN CONT..

provides a structured approach to building complex AI applications by allowing developers to chain together components such as prompt templates, memory, document retrieval, and response generation

Modular Design
Ease of Use
Extensibility

LANGCHAIN CORE FEATURES

- ❖ Document Loaders
- ❖ Embeddings and Vector Stores
- ❖ Retrievers:
- ❖ LLM Integration:
- ❖ Chains and Prompt Templates:

LangChain’s integrated environment allows for the seamless combination of retrieval and generation

THANK YOU!