Analytics Vidhya | DATAHACK SUMMIT

GenAI Hack Session

Improving Real-World RAG Systems

Key Challenges & Practical Solutions

Speaker

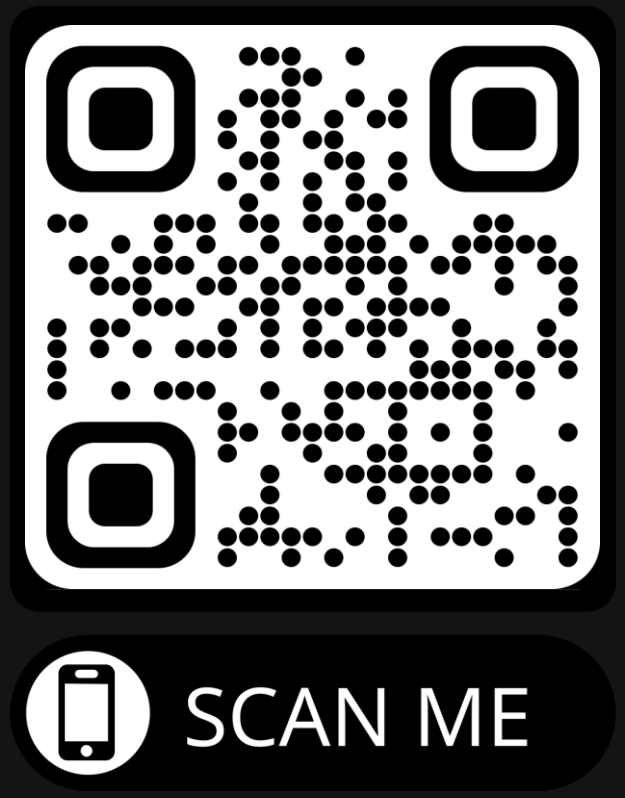
Dipanjan (DJ) Sarkar

Head of Community & Principal AI Scientist at Analytics Vidhya

Published Author, Google Developer Expert & Cloud Champion Innovator

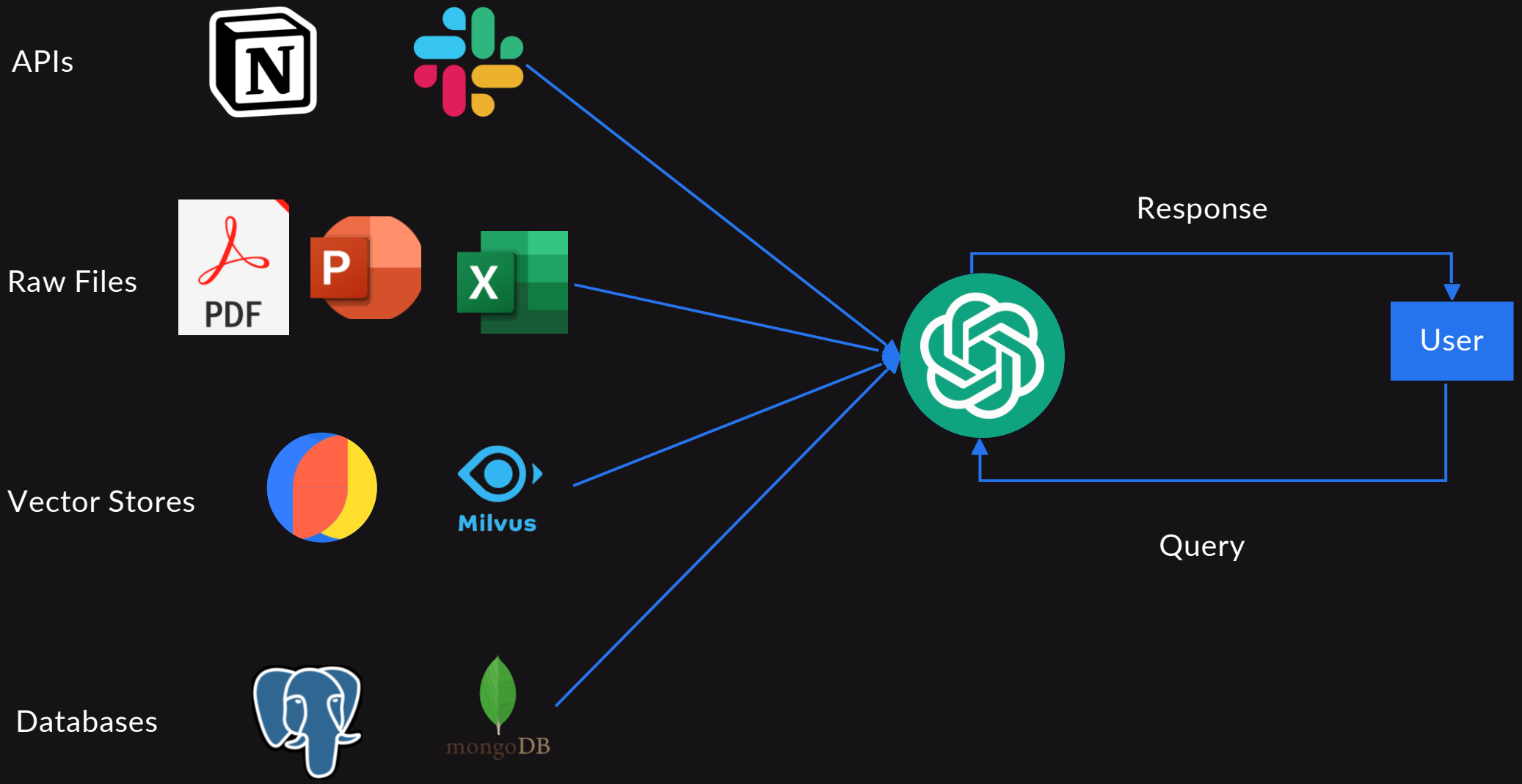
Slides & Code

https://github.com/dipanjanS/improving-RAG-systems-dhs2024

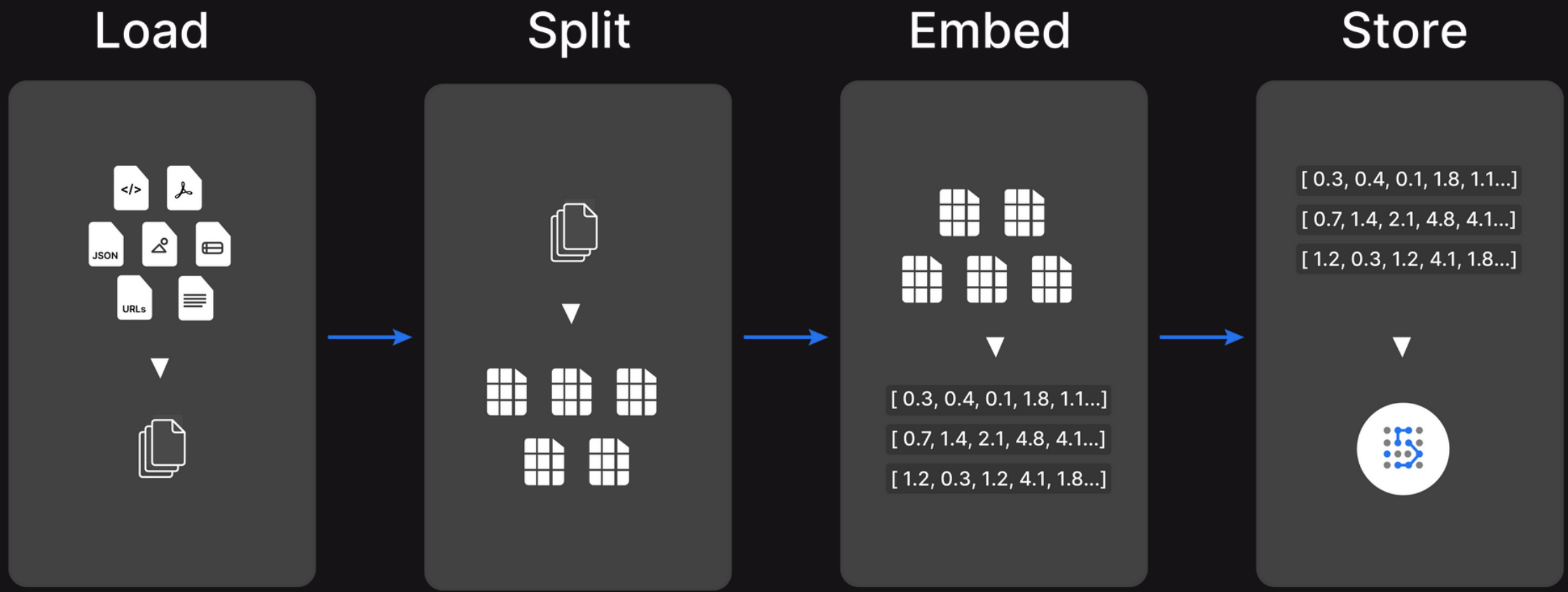


Understanding RAG Systems

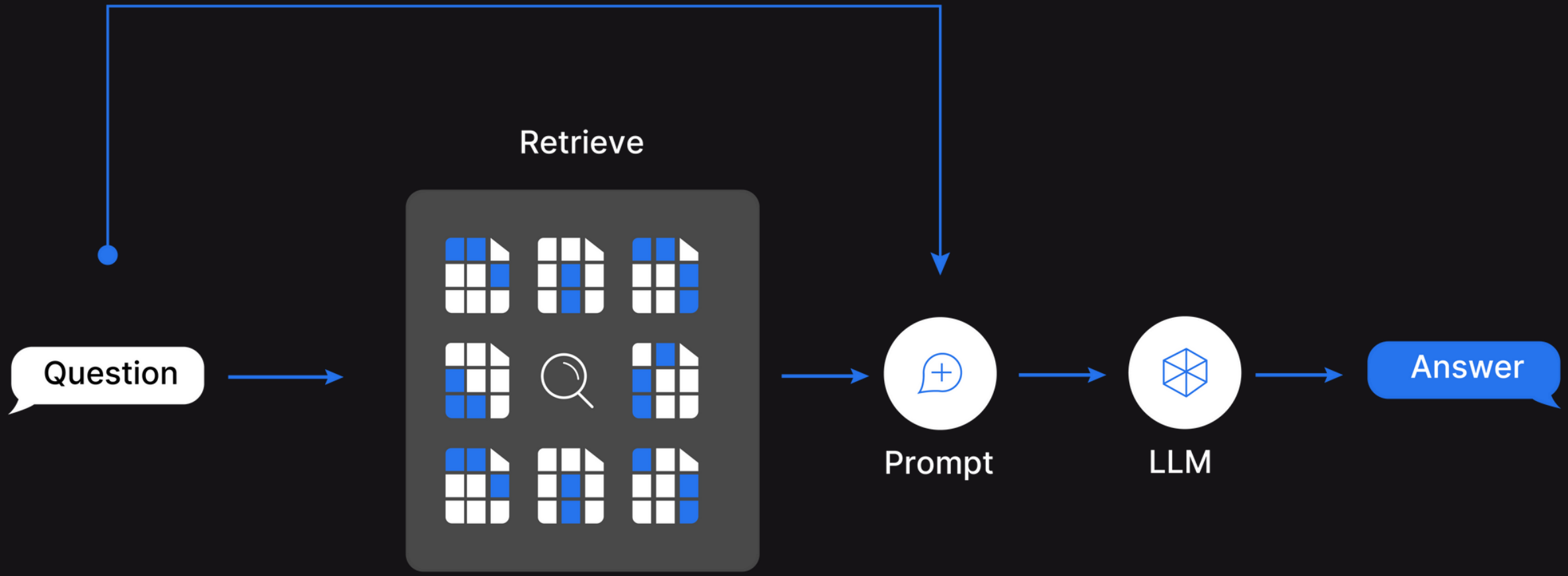
What is a RAG System?



RAG System Architecture - Data Indexing

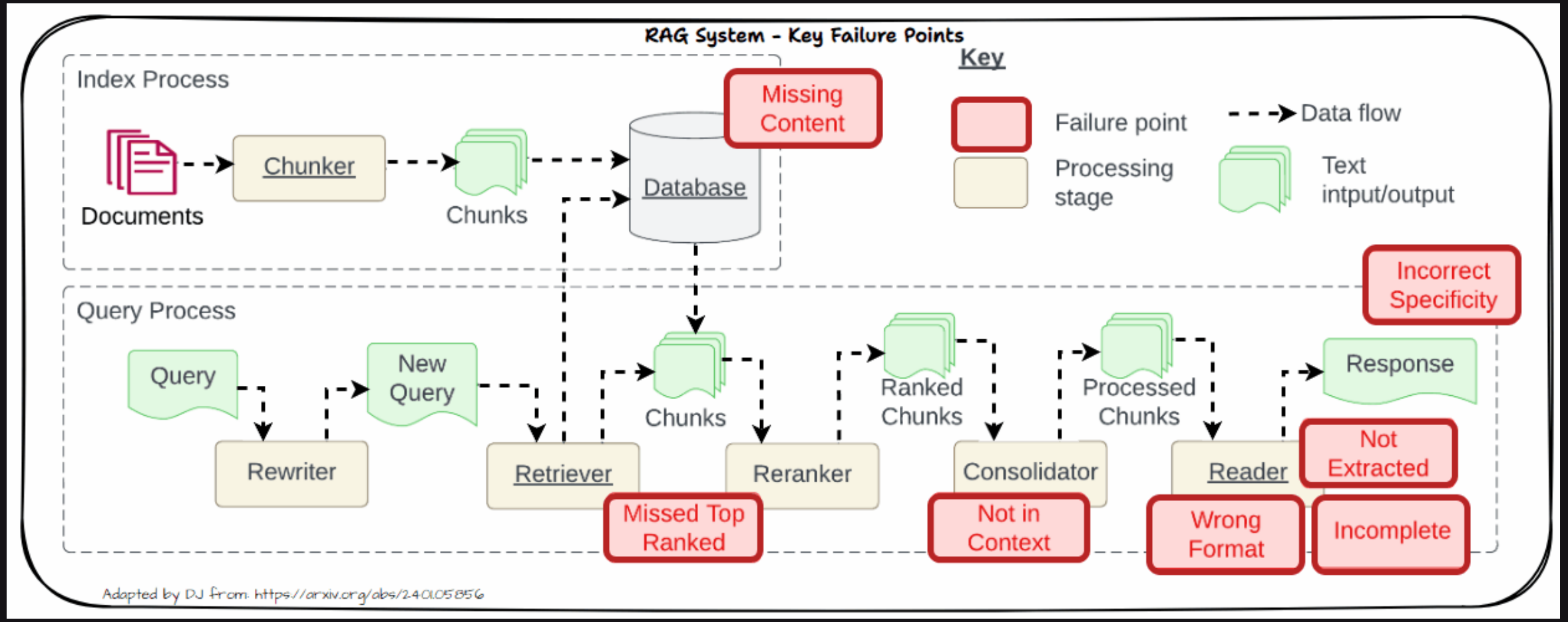


RAG System Architecture - Search and Generation



RAG System Challenges & Practical Solutions

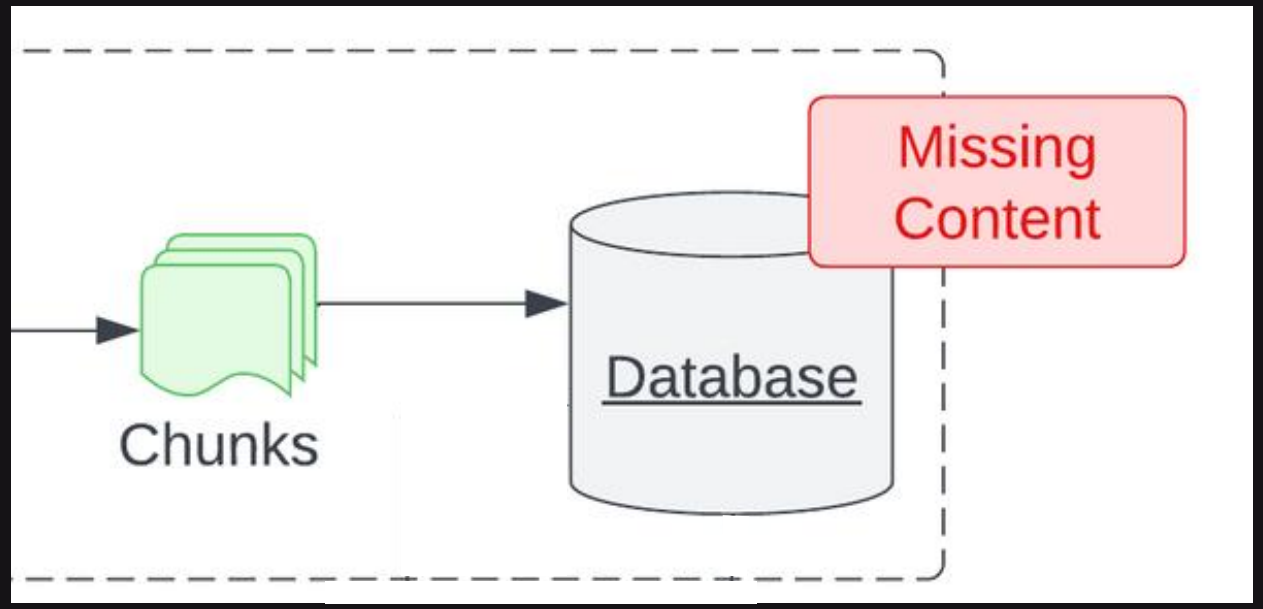
Key Failure or Pain Points in a RAG System



Source: Seven Failure Points When Engineering a Retrieval Augmented Generation System

Problem: Missing Content

- Missing Content means the relevant context to answer the question is not present in the database
- Leads to the model giving a wrong answer and hallucinating
- End users end up being frustrated with irrelevant or wrong responses

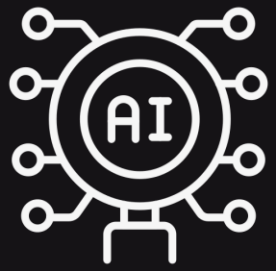


Source: Seven Failure Points When Engineering a Retrieval Augmented Generation System

Solutions for Missing Content

Better data cleaning using tools like unstructured.io to ensure we extract good quality data

Better prompting to constrain the model to NOT answer the question if the context is irrelevant



Agentic RAG with search tools to get live information for question with no context data

Source: Seven Failure Points When Engineering a Retrieval Augmented Generation System

Hands-on Demo

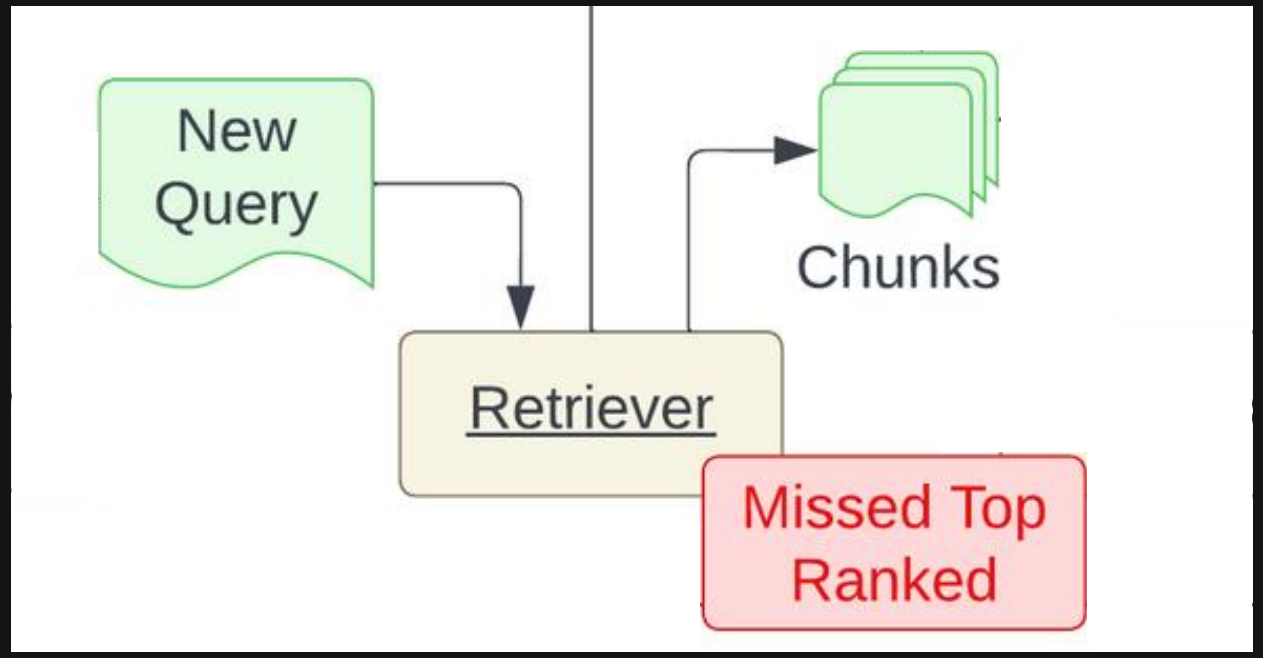
- Better Data Cleaning
- Better Prompting
- Agentic RAG with Tools

- Get the notebook from HERE

Source: Seven Failure Points When Engineering a Retrieval Augmented Generation System

Problem: Missed Top Ranked

- Missed Top Ranked means context documents don’t appear in the top retrieval results
- Leads to the model not able to answer the question
- Documents to answer the question are present but failed to get retrieved due to poor retrieval strategy

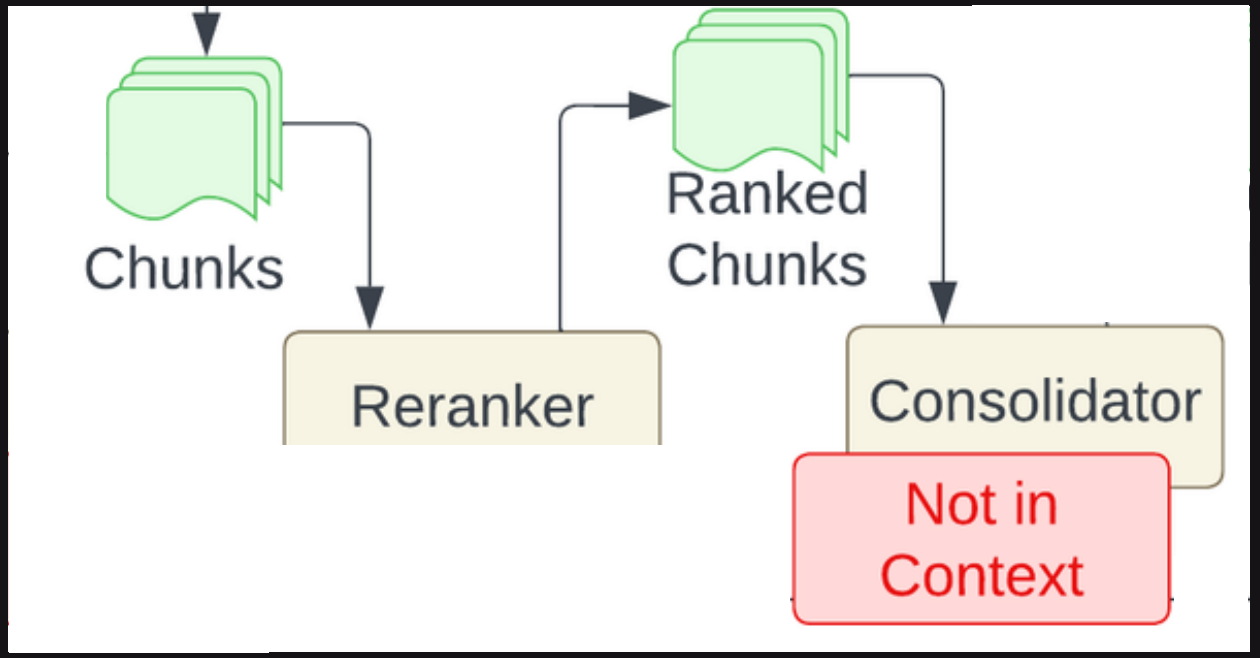


Source: Seven Failure Points When Engineering a Retrieval Augmented Generation System

Problem: Not in Context

- Not in Context means documents with the answer are present during initial retrieval but did not make it into the final context for generating an answer

- Bad retrieval, reranking and consolidation strategies lead to missing out on the right documents in context

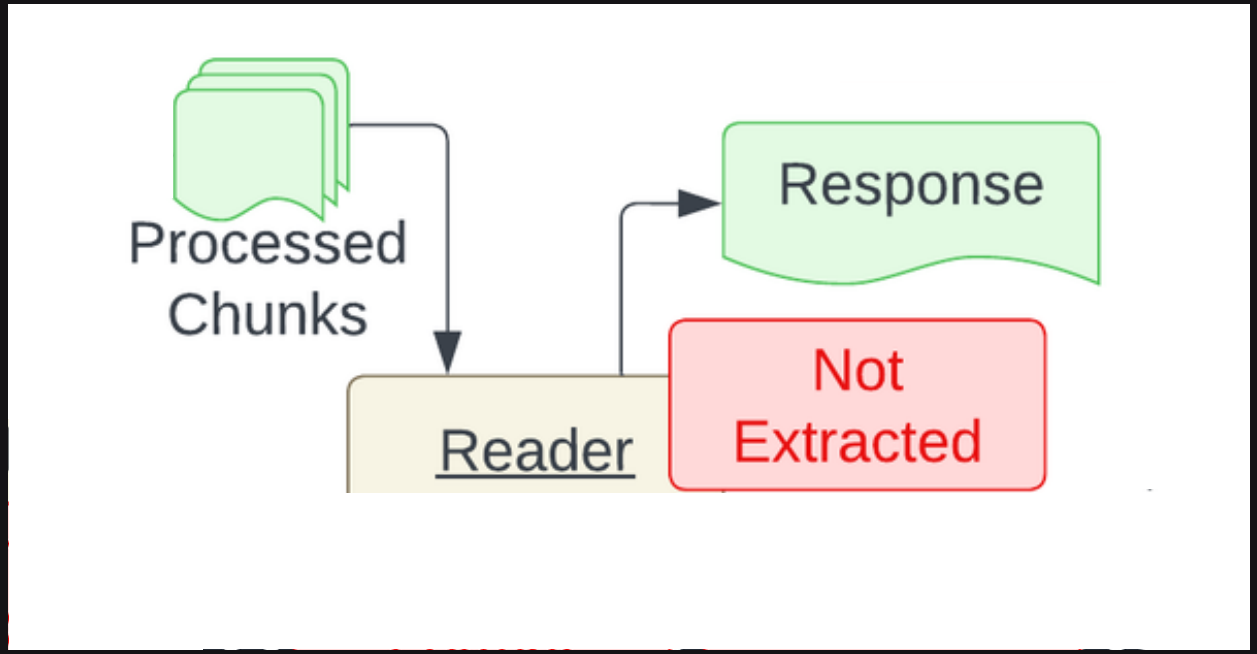


Source: Seven Failure Points When Engineering a Retrieval Augmented Generation System

Problem: Not Extracted

- Not extracted means the LLM struggles to extract the correct answer from the provided context even if it has the answer

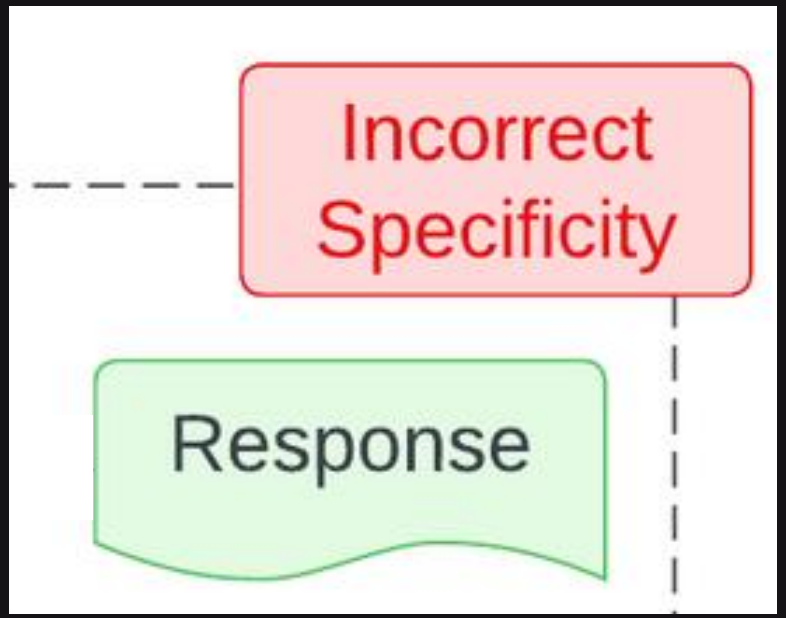
- This occurs when there is too much unnecessary information, noise or contradicting information in the context



Source: Seven Failure Points When Engineering a Retrieval Augmented Generation System

Problem: Incorrect Specificity

- Output response is too vague and is not detailed or specific enough
- Vague or generic queries might lead to not getting the right context and response
- Wrong chunking or bad retrieval can lead to this problem



Source: Seven Failure Points When Engineering a Retrieval Augmented Generation System

Solutions for Missed Top Ranked, Not in Context & Incorrect Specificity

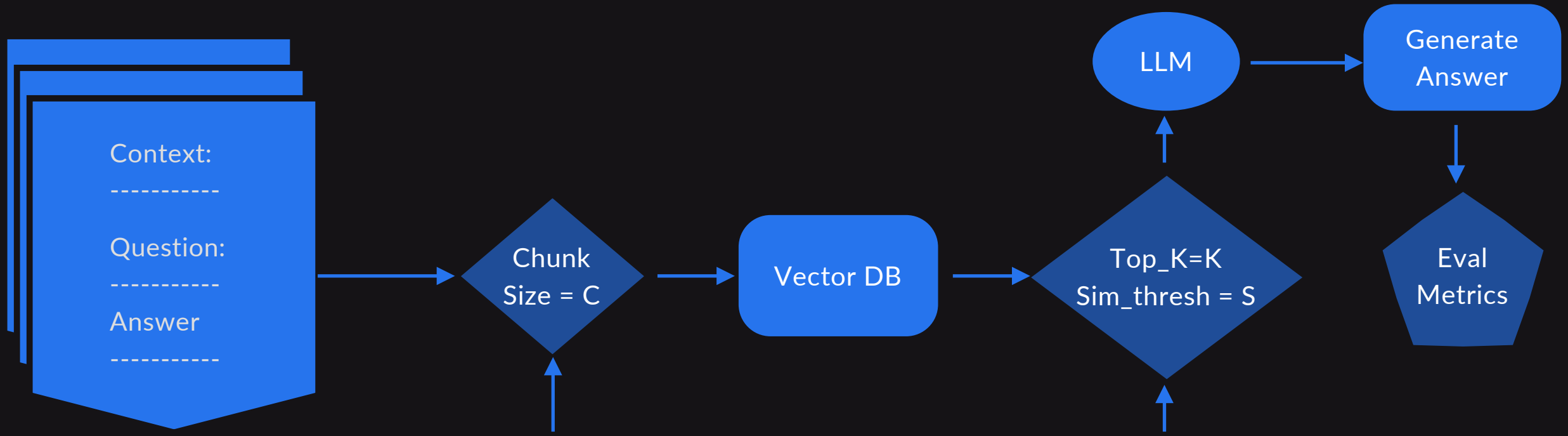
- Use Better Chunking Strategies
- Hyperparameter Tuning - Chunking & Retrieval
- Use Better Embedder Models
- Use Advanced Retrieval Strategies
- Use Context Compression Strategies
- Use Better Reranker Models

Source: Seven Failure Points When Engineering a Retrieval Augmented Generation System

Experiment with Various Chunking Strategies

Splitter Type	Description
RecursiveCharacterText Splitter	Recursively splits text into larger chunks based on several defined characters. Tries to keep related pieces of text next to each other. LangChain's recommended way to start splitting text
CharacterTextSplitter	Splits text based on a user defined character. One of the simpler text splitters
tiktoken	Splits text based on tokens using trained LLM tokenizers like GPT-4
spaCy	Splits text using the tokenizer from the popular NLP library - spaCy
SentenceTransformers	Splits text based on tokens using trained open LLM tokenizers available from the popular sentence-transformers library
unstructured.io	The unstructured library allows various splitting and chunking strategies including splitting text based on key sections and titles

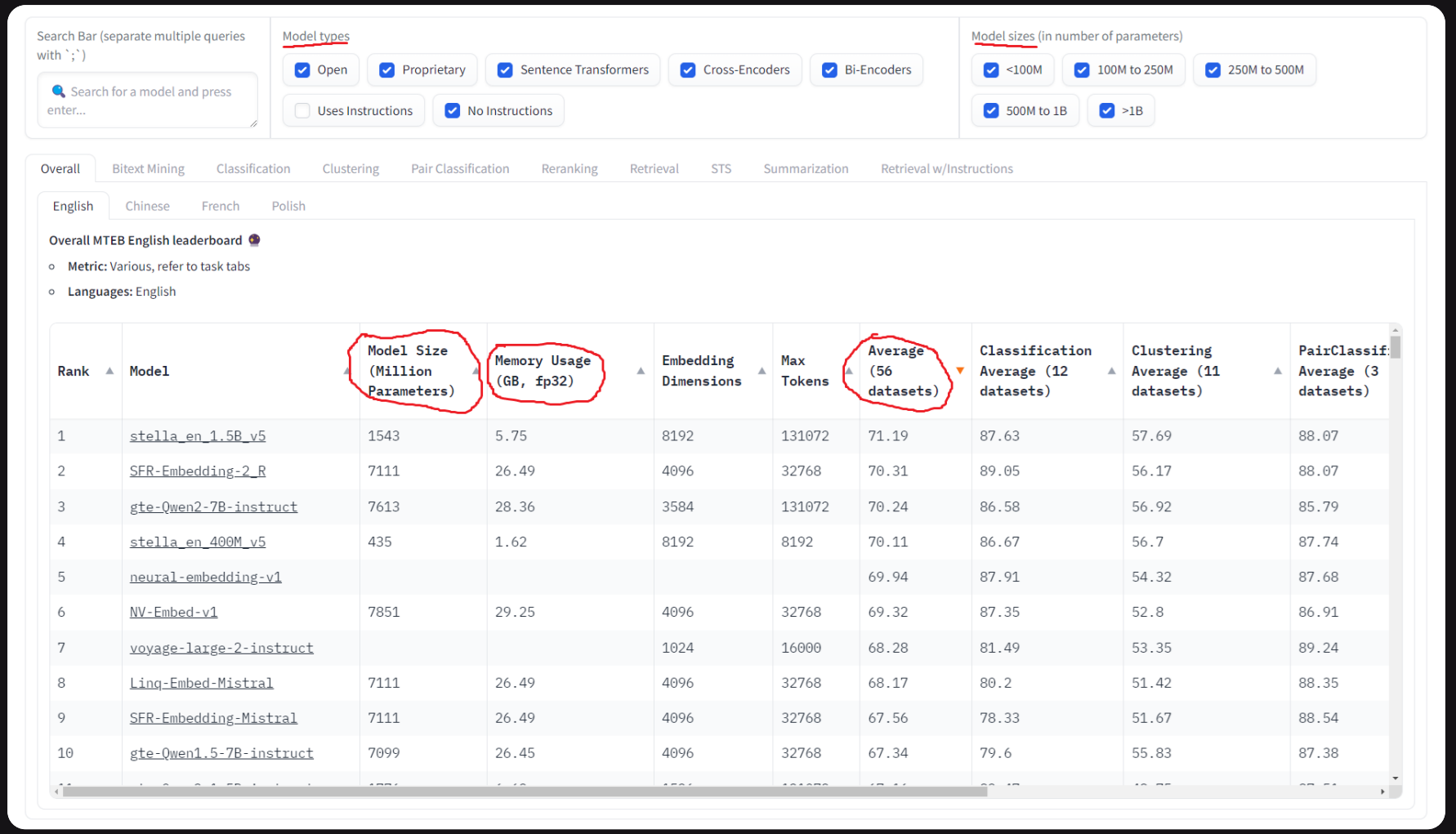
Hyperparameter Tuning - Chunking & Retrieval



C Value Space
500
1,000
2,000

K Value Space	S Value Space
5	0.2
8	0.3
10	0.5

Better Embedder Models - MTEB Leaderboard



Search Bar (separate multiple queries with `;`)

Search for a model and press enter...

Model types

Open | Proprietary | Sentence Transformers | Cross-Encoders | Bi-Encoders | Uses Instructions | No Instructions

Model sizes (in number of parameters)

<100M | 100M to 250M | 250M to 500M | 500M to 1B | >1B

Overall | Bitext Mining | Classification | Clustering | Pair Classification | Reranking | Retrieval | STS | Summarization | Retrieval w/Instructions

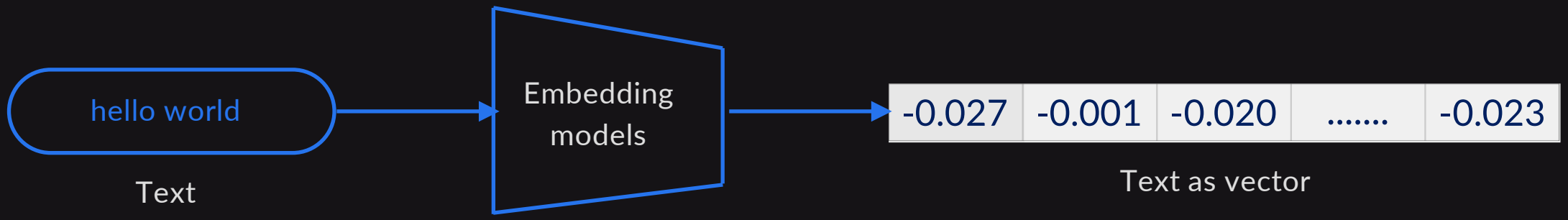
English | Chinese | French | Polish

Overall MTEB English leaderboard

- **Metric:** Various, refer to task tabs
- **Languages:** English

Rank	Model	Model Size (Million Parameters)	Memory Usage (GB, fp32)	Embedding Dimensions	Max Tokens	Average (56 datasets)	Classification Average (12 datasets)	Clustering Average (11 datasets)	PairClassif Average (3 datasets)
1	stella_en_1.5B_v5	1543	5.75	8192	131072	71.19	87.63	57.69	88.07
2	SFR-Embedding-2_R	7111	26.49	4096	32768	70.31	89.05	56.17	88.07
3	gte-Qwen2-7B-instruct	7613	28.36	3584	131072	70.24	86.58	56.92	85.79
4	stella_en_400M_v5	435	1.62	8192	8192	70.11	86.67	56.7	87.74
5	neural-embedding-v1					69.94	87.91	54.32	87.68
6	NV-Embed-v1	7851	29.25	4096	32768	69.32	87.35	52.8	86.91
7	voyage-large-2-instruct			1024	16000	68.28	81.49	53.35	89.24
8	Linq-Embed-Mistral	7111	26.49	4096	32768	68.17	80.2	51.42	88.35
9	SFR-Embedding-Mistral	7111	26.49	4096	32768	67.56	78.33	51.67	88.54
10	gte-Qwen1.5-7B-instruct	7099	26.45	4096	32768	67.34	79.6	55.83	87.38

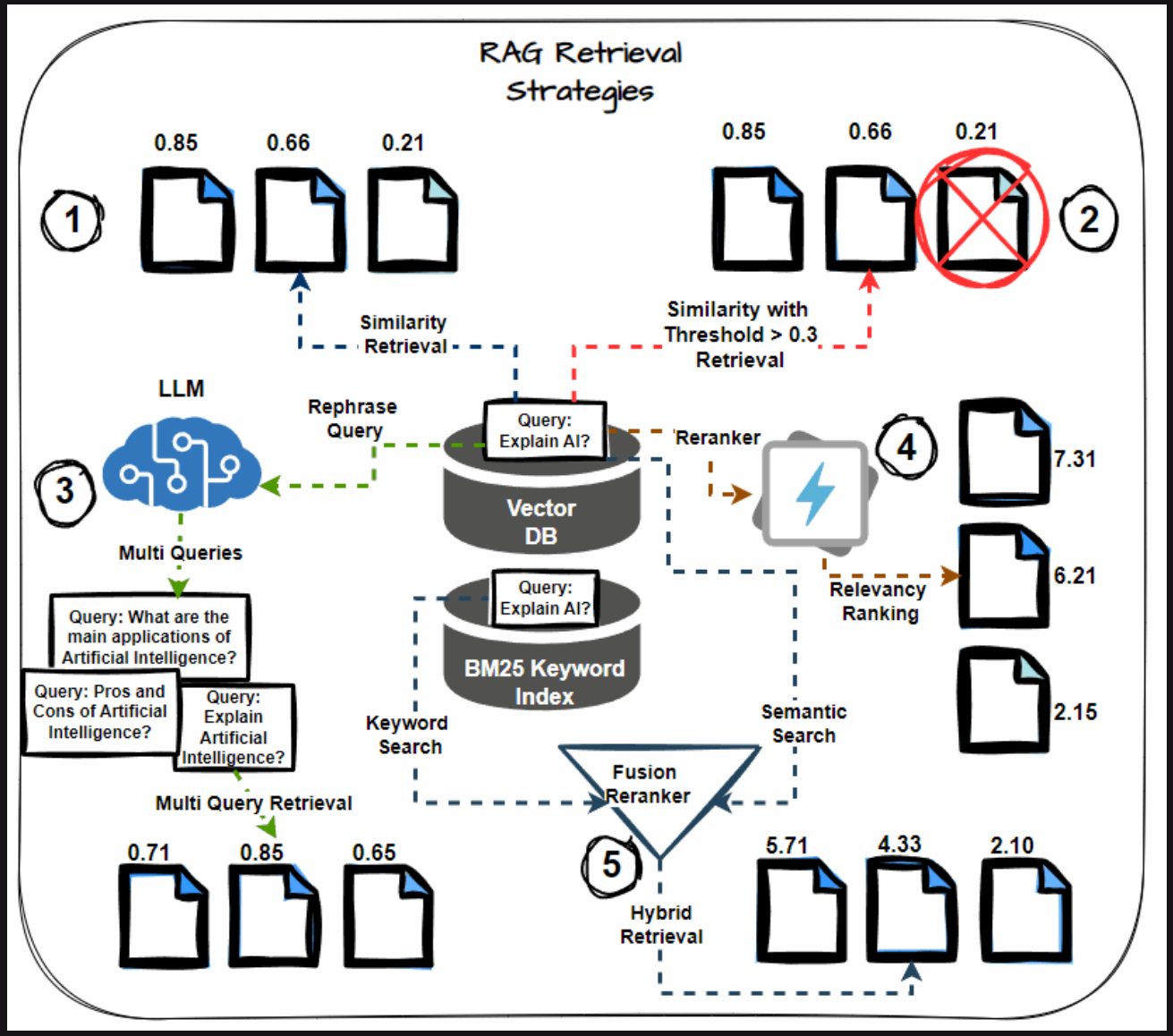
Better Embedder Models - Experiment Yourself



- Newer Embedder Models will be trained on more data and often better
- Don’t just go by benchmarks, use and experiment on your data
- Do not use commercial models if data privacy is important

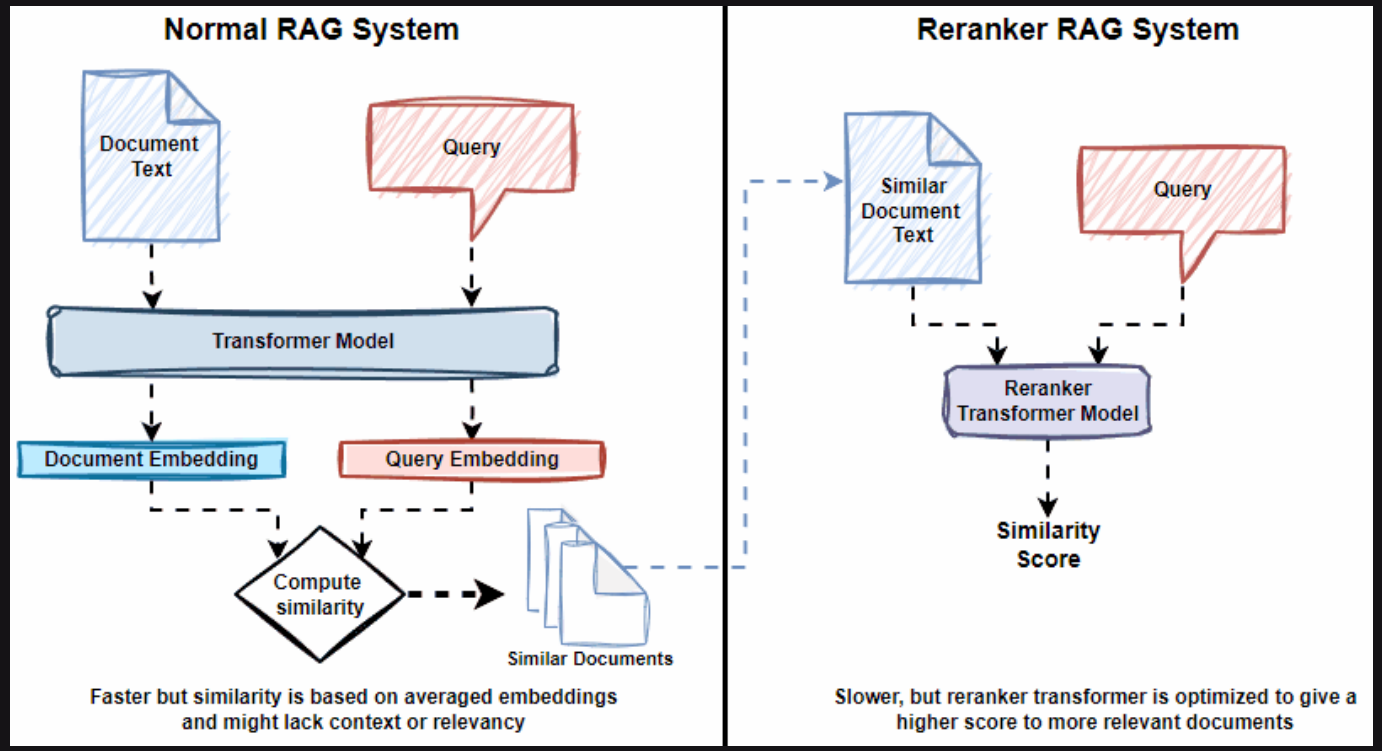
Advanced Retrieval Strategies

- Semantic Similarity Thresholding
- Multi-query Retrieval
- Hybrid Search (Keyword + Semantic)
- Reranking
- Chained Retrieval



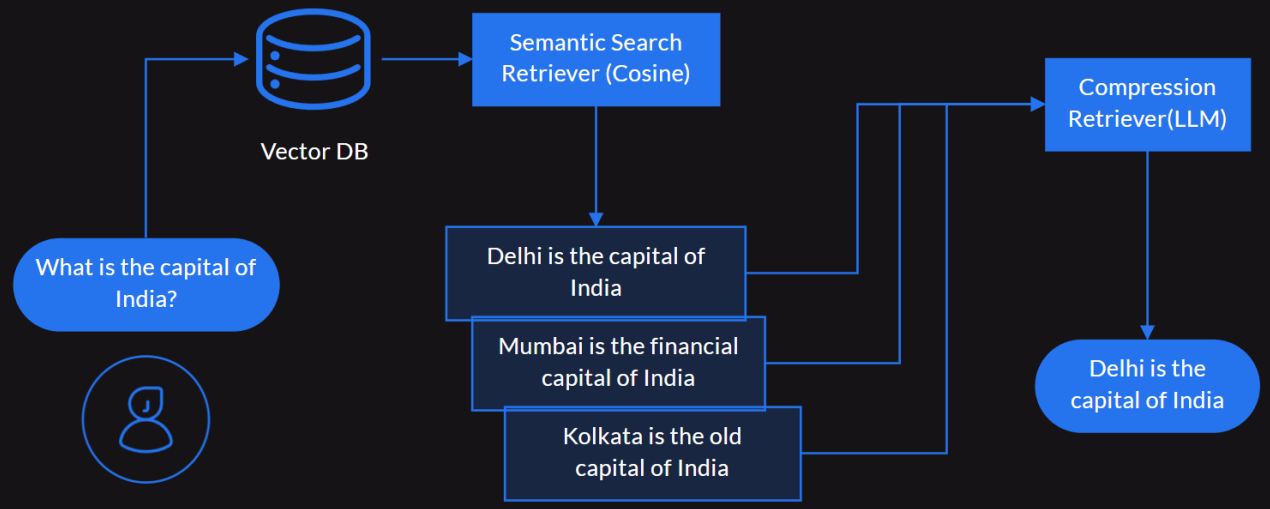
Better Reranker Models

- Rerankers are fine-tuned cross-encoder transformer models
- These models take in a pair of documents (Query, Document) and return back a relevance score
- Models fine-tuned on more pairs and released recently will usually be better



Context Compression Strategies

- LLM prompt-based Context Compression
  - Extractor: Filters out content from context document not related to query
  - Filter: Filters out whole context documents not related to query
- Microsoft LLMLingua Prompt Compression



Hands-on Demo

Solutions for Missed Top Ranked, Not in Context, Not Extracted & Incorrect Specificity

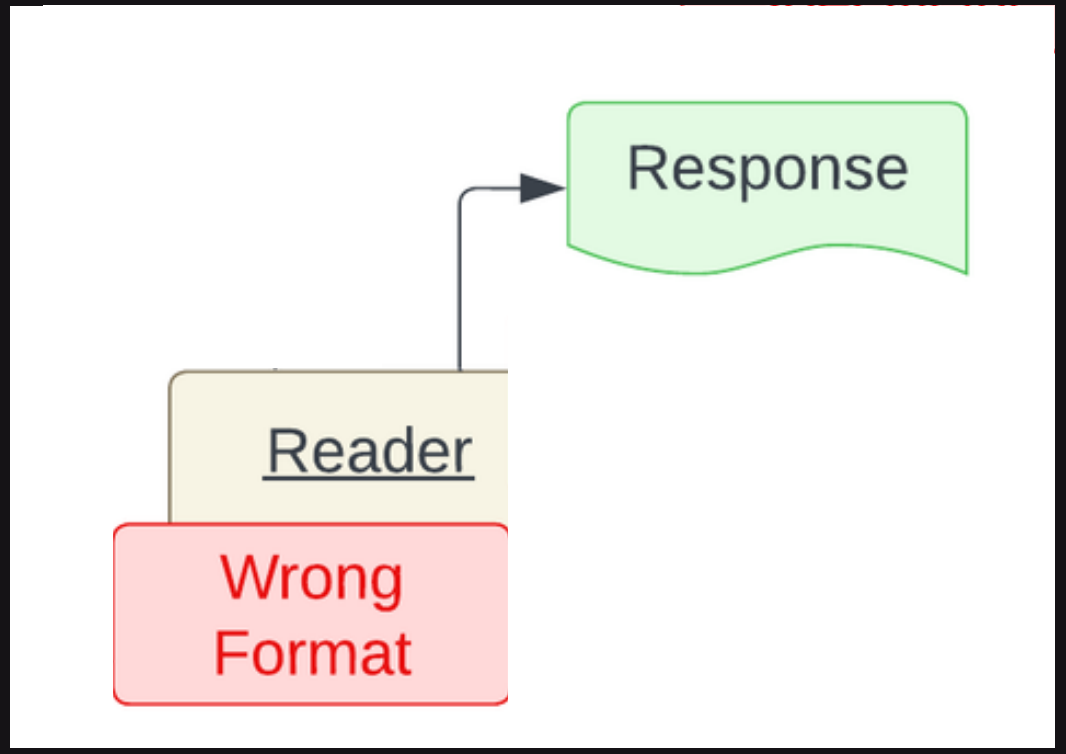
- Effect of Embedder Models
- Advanced Retrieval Strategies
- Chained Retrieval with Rerankers
- Context Compression Strategies

- Get the notebook from HERE

Source: Seven Failure Points When Engineering a Retrieval Augmented Generation System

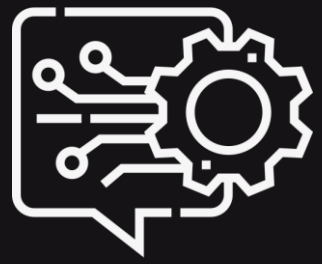
Problem: Incorrect Specificity

- The output response is in the wrong format
- It happens when you tell the LLM to return the response in a specific format e.g JSON and it fails to do so



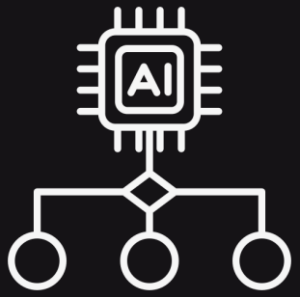
Source: Seven Failure Points When Engineering a Retrieval Augmented Generation System

Solutions for Wrong Format



Powerful LLMs have native support for response formats e.g OpenAI supports JSON outputs

Better Prompting and Output Parsers



Structured Output Frameworks

Source: Seven Failure Points When Engineering a Retrieval Augmented Generation System

Solutions for Wrong Format - Native LLM Support

```python
from langchain_openai import ChatOpenAI

chatgpt = ChatOpenAI(model_name="gpt-4o-mini", temperature=0,
                     model_kwargs={"response_format": {"type": "json_object"}})
```

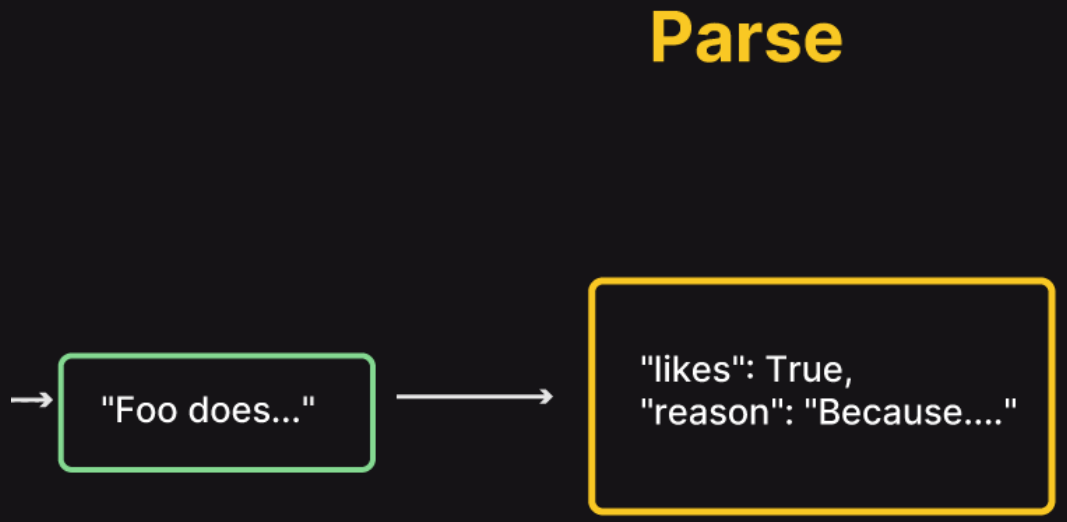
```python
prompt = """Who won the Champions league in 2023,
            Output should be in JSON and have following fields:
            win_team, lose_team, venue, date, score
         """
response = chatgpt.invoke(prompt)
```

```python
print(response.content)
```

```
{
  "win_team": "Manchester City",
  "lose_team": "Inter Milan",
  "venue": "Atatürk Olympic Stadium",
  "date": "June 10, 2023",
  "score": "1-0"
}
```

Solutions for Wrong Format - Output Parsers & Better Prompting

- LangChain allows to convert the raw LLM response into a more consumable format using Output Parsers.
- There exists a variety of parsers including:
  - String parser
  - CSV parser
  - Pydantic parser
  - JSON parser



Solutions for Wrong Format - Structured Output Frameworks

LLM Structured Output Benchmarks

python 3.11.9 | DOI 10.5281/zenodo.12674272 | license Apache-2.0

Followers 88 | Marie Stephen Leo | Medium

Benchmark various LLM Structured Output frameworks: Instructor, Mirascope, Langchain, LlamaIndex, Fructose, Marvin, Outlines, LMFormatEnforcer, etc on tasks like multi-label classification, named entity recognition, synthetic data generation, etc.

Benchmark Results [2024-07-20]

1. Multi-label classification

Framework	Model	Reliability	Latency p95 (s)
Instructor	gpt-4o-mini-2024-07-18	1.000	1.096
Mirascope	gpt-4o-mini-2024-07-18	1.000	1.523
Fructose	gpt-4o-mini-2024-07-18	1.000	2.256
Outlines	unsloth/llama-3-8b-Instruct-bnb-4bit	1.000	1.891*
LMFormatEnforcer	unsloth/llama-3-8b-Instruct-bnb-4bit	1.000	2.994*
Llamaindex	gpt-4o-mini-2024-07-18	0.999	0.936
Modelsmith	gpt-4o-mini-2024-07-18	0.999	1.333
Marvin	gpt-4o-mini-2024-07-18	0.998	1.722

Hands-on Demo

Solutions for Wrong Format

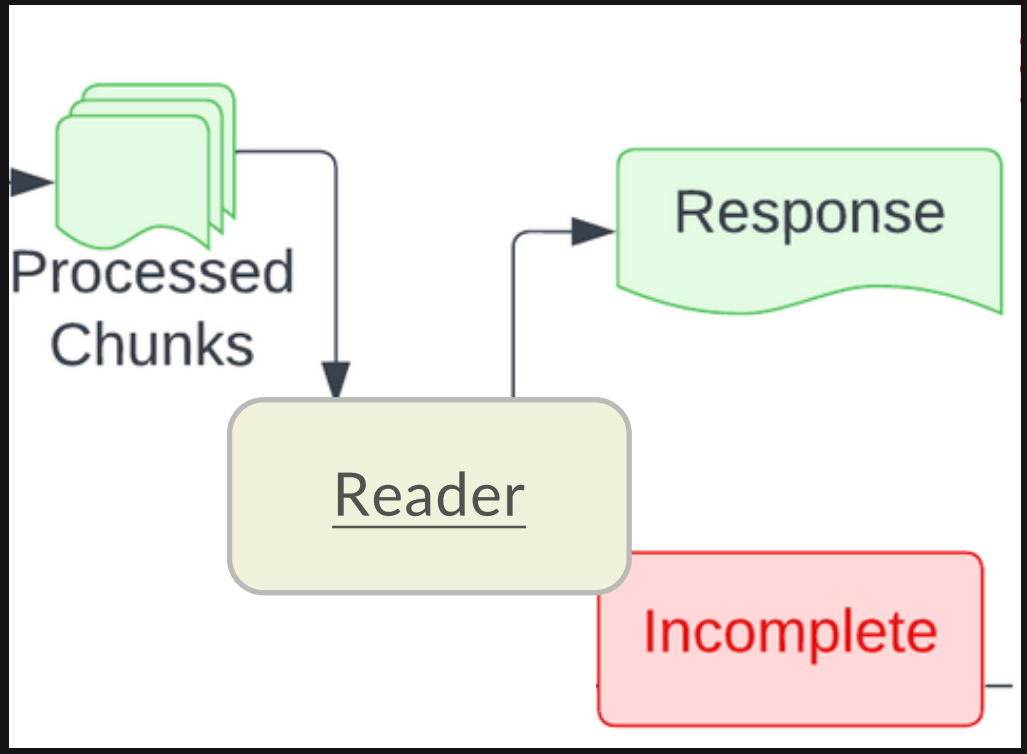
- Native LLM Support
- Output Parsers

- Get the notebook from HERE

Source: Seven Failure Points When Engineering a Retrieval Augmented Generation System

Problem: Incomplete

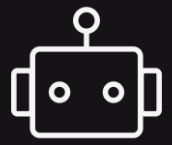
- Incomplete means the generated response is incomplete
- This could be because of poorly worded questions, lack of right context retrieved, bad reasoning



Source: Seven Failure Points When Engineering a Retrieval Augmented Generation System

Solutions for Incomplete

Use Better LLMs like GPT-4o, Claude 3.5 or Gemini 1.5



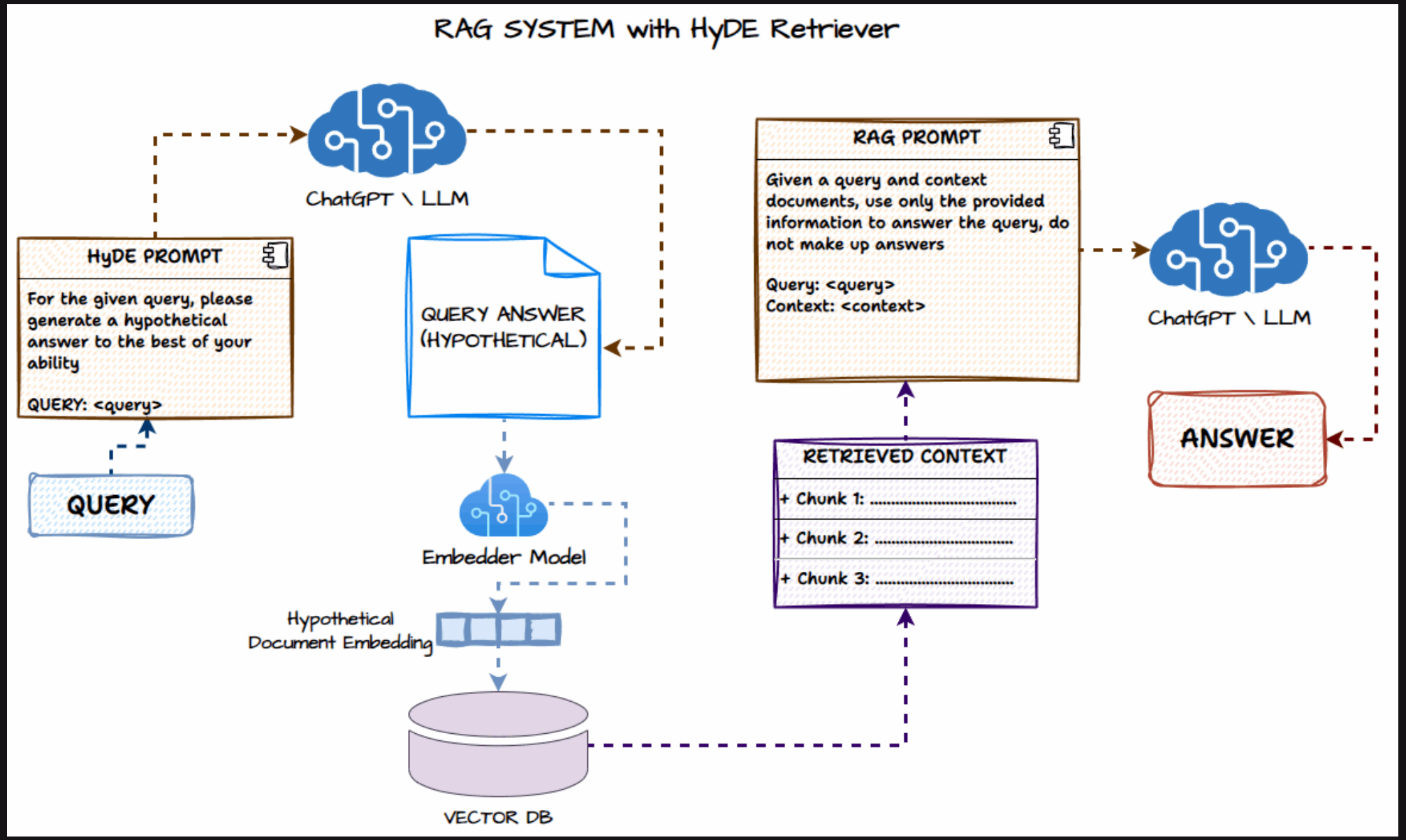
Build Agentic Systems with Tool Use if necessary

Use Advanced Prompting Techniques like Chain-of-Thought, Self-Consistency

Rewrite User Query and Improve Retrieval - HyDE

Source: Seven Failure Points When Engineering a Retrieval Augmented Generation System

HyDE - Hypothetical Document Embedding



Other Practical Solutions from recent Research Papers which actually work!

RAG vs. Long Context LLMs

- Long Context LLMs often outperform RAG but are very expensive in terms of computing and cost
- Hybrid approach where you can use an LLM to reflect and see if the RAG answer is good enough or route to Long Context LLM

Retrieval Augmented Generation or Long-Context LLMs? A Comprehensive Study and Hybrid Approach

Zhuowan Li[1] **Cheng Li**[1] **Mingyang Zhang**[1]
Qiaozhu Mei[2*] **Michael Bendersky**[1]
[1] Google DeepMind [2] University of Michigan
[1] {zhuowan,chgli,mingyang,bemike}@google.com [2] qmei@umich.edu

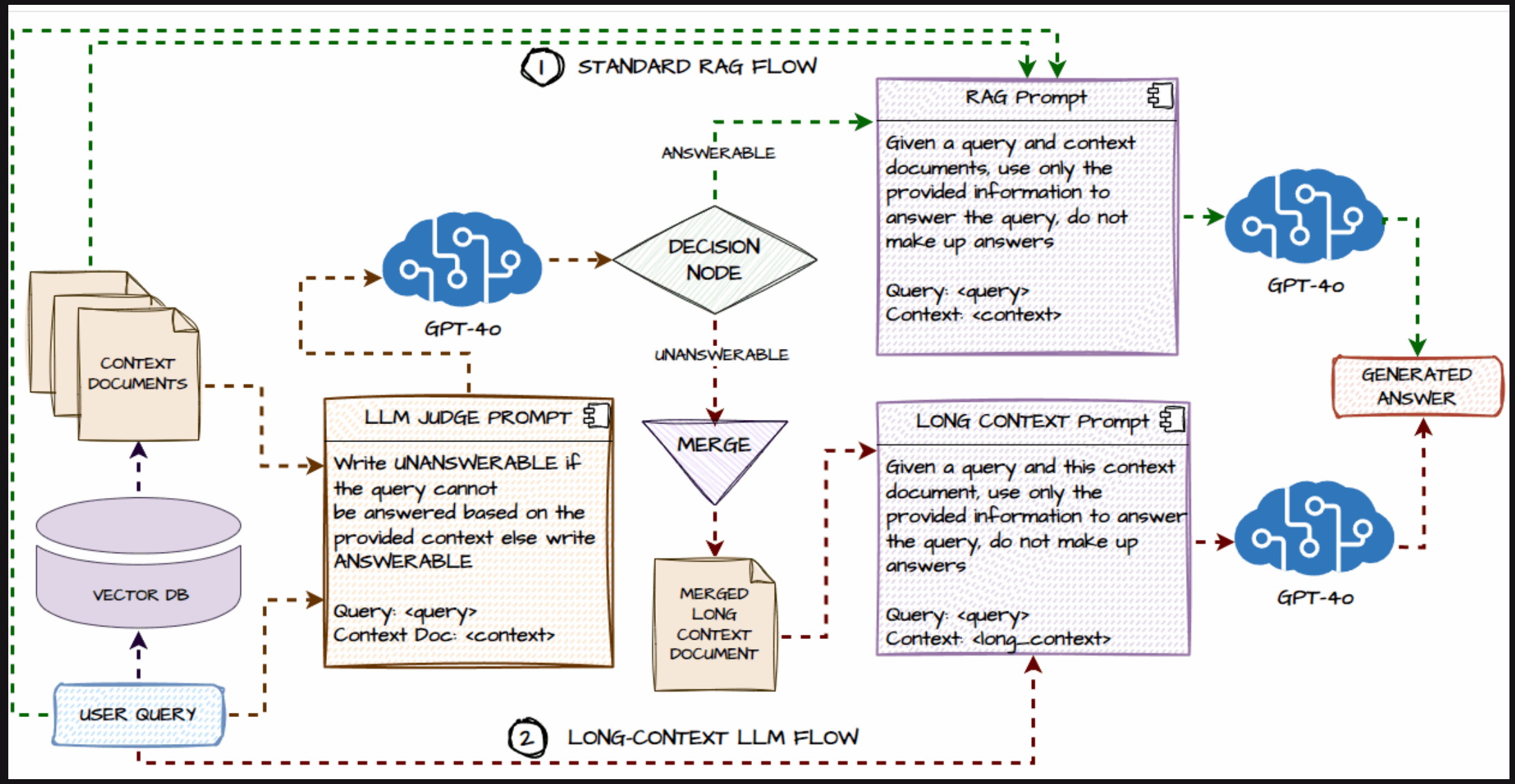
Abstract

Retrieval Augmented Generation (RAG) has been a powerful tool for *Large Language Models (LLMs)* to efficiently process overly lengthy contexts. However, recent LLMs like Gemini-1.5 and GPT-4 show exceptional capabilities to understand long contexts directly. We conduct a comprehensive comparison between RAG and long-context *(LC)* LLMs, aiming to leverage the strengths of both. We benchmark RAG and LC across various public datasets using three latest LLMs. Results reveal that when resourced sufficiently, LC consistently outperforms RAG in terms of average performance. However, RAG's significantly lower cost remains a distinct advantage. Based on this observation, we propose SELF-ROUTE, a simple yet effective method that routes queries to RAG or LC based on model self-reflection. SELF-ROUTE significantly reduces the computation cost while maintaining a comparable performance to LC. Our findings provide a guideline for long-context applications of LLMs using RAG and LC.

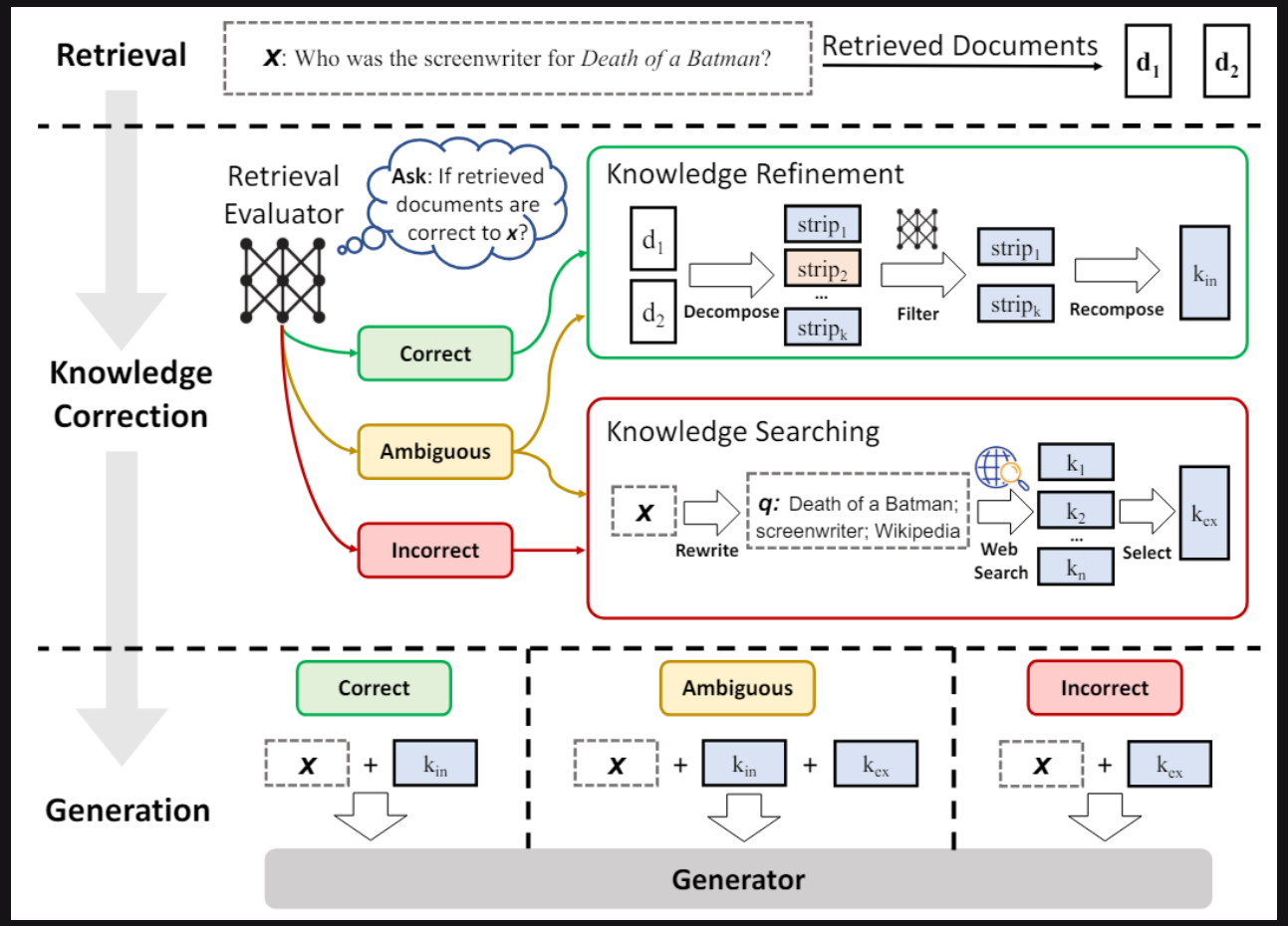
Figure 1: While long-context LLMs (LC) surpass RAG in long-context understanding, RAG is significantly more cost-efficient. Our approach, SELF-ROUTE, combining RAG and LC, achieves comparable performance to LC at a much lower cost.

and RAG: on one hand, RAG conceptually acts as a prior, regularizing the attention of LLMs onto retrieved segments, thus avoiding the distraction of the irrelevant information and saving unnecessary attention computations; on the other hand, large-scale pretraining may enable LLMs to develop even stronger long-context capabilities. Therefore, we

RAG vs Long Context LLMs - Self-Router RAG



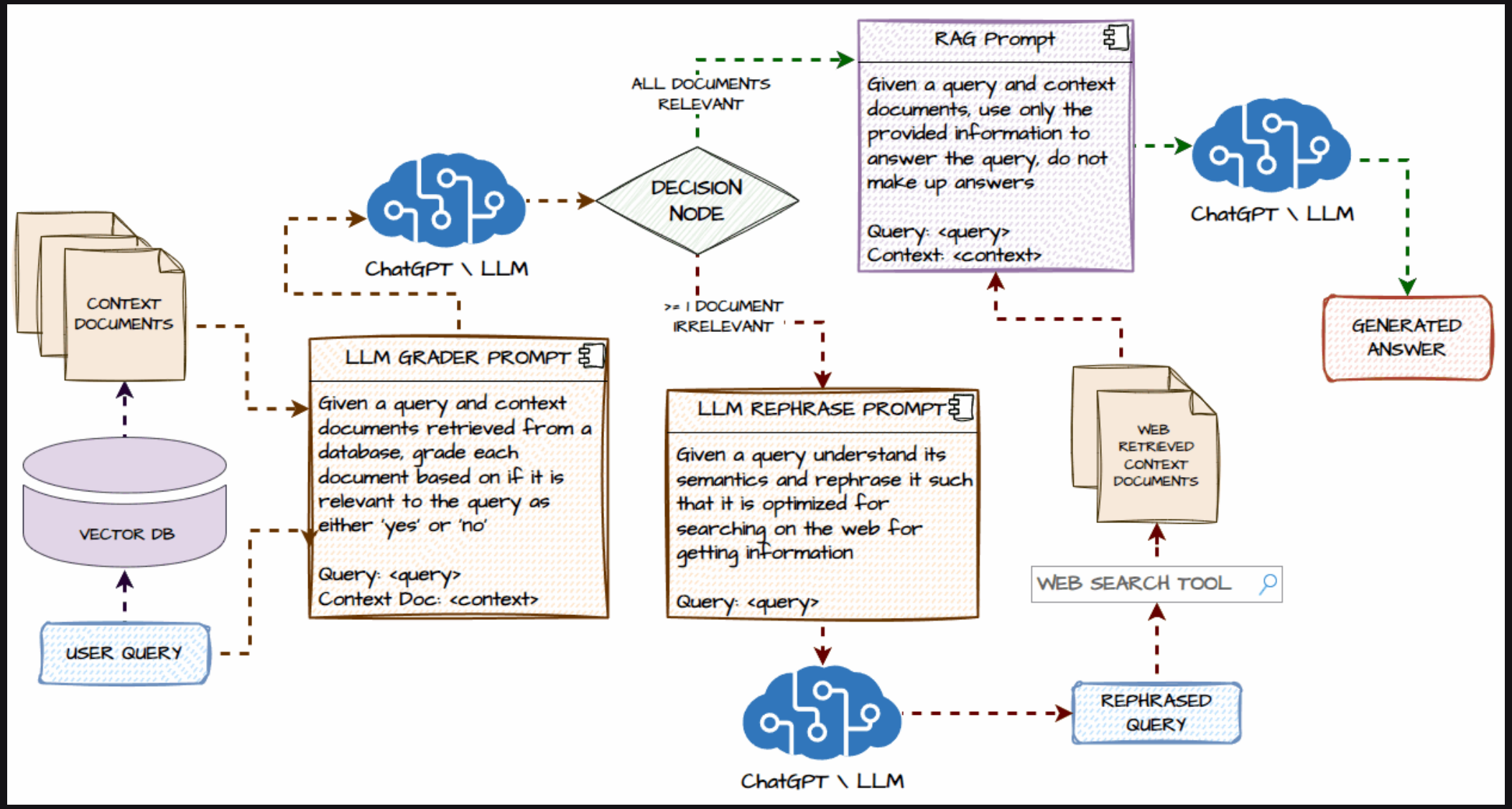
Agentic Corrective RAG



- Step 1:
  - Retrieve context documents from vector database from the input query
- Step 2:
  - Use an LLM to check if retrieved documents are relevant to input question
- Step 3:
  - If all documents are relevant (Correct), no specific action needed
- Step 4:
  - If some or all documents are not relevant (Ambiguous OR Incorrect), rephrase the query and search the web to get relevant context information
- Step 5:
  - Send rephrased query and context documents or information to the LLM for response generation

Source: Corrective Retrieval Augmented Generation; https://arxiv.org/abs/2401.15884

Agentic Corrective RAG



Source: https://www.analyticsvidhya.com/blog/2024/07/building-agentic-rag-systems-with-langgraph/

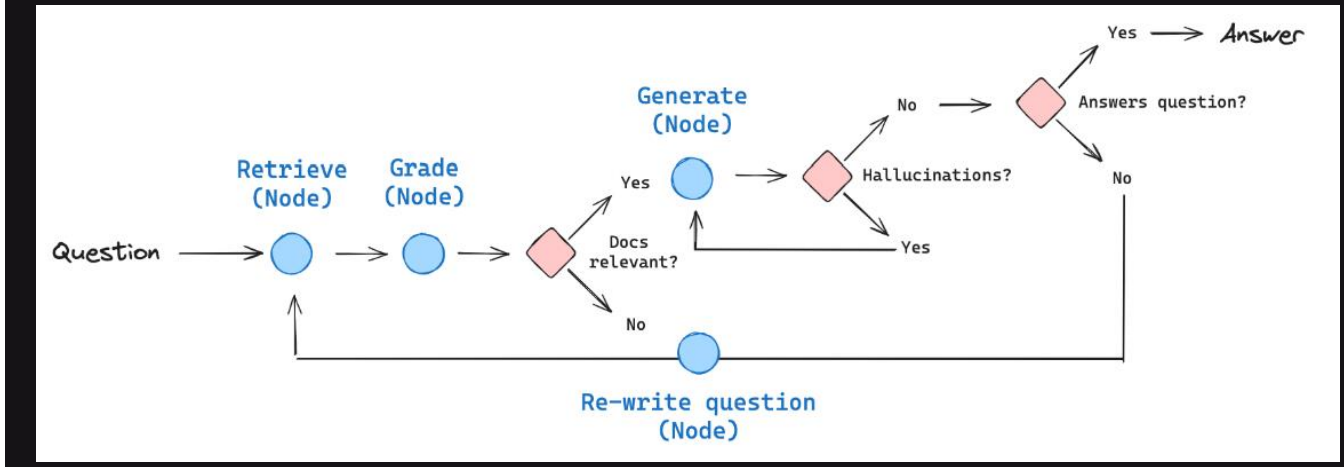
Agentic Self-Reflection RAG

SELF-RAG: LEARNING TO RETRIEVE, GENERATE, AND CRITIQUE THROUGH SELF-REFLECTION

Akari Asai[†], **Zeqiu Wu**[†], **Yizhong Wang**[†§], **Avirup Sil**[‡], **Hannaneh Hajishirzi**[†§]
[†]University of Washington [§]Allen Institute for AI [‡]IBM Research AI
{akari,zeqiuwu,yizhongw,hannaneh}@cs.washington.edu, avi@us.ibm.com

ABSTRACT

Despite their remarkable capabilities, large language models (LLMs) often produce responses containing factual inaccuracies due to their sole reliance on the parametric knowledge they encapsulate. Retrieval-Augmented Generation (RAG), an ad hoc approach that augments LMs with retrieval of relevant knowledge, decreases such issues. However, indiscriminately retrieving and incorporating a fixed number of retrieved passages, regardless of whether retrieval is necessary, or passages are relevant, diminishes LM versatility or can lead to unhelpful response generation. We introduce a new framework called **Self-Reflective Retrieval-Augmented Generation (SELF-RAG)** that enhances an LM's quality and factuality through retrieval and self-reflection. Our framework trains a single arbitrary LM that adaptively retrieves passages on-demand, and generates and reflects on retrieved passages and its own generations using special tokens, called *reflection* tokens. Generating reflection tokens makes the LM controllable during the inference phase, enabling it to tailor its behavior to diverse task requirements. Experiments show that SELF-RAG (7B and 13B parameters) significantly outperforms state-of-the-art LLMs and retrieval-augmented models on a diverse set of tasks. Specifically, SELF-RAG outperforms ChatGPT and retrieval-augmented Llama2-chat on Open-domain QA, reasoning and fact verification tasks, and it shows significant gains in improving factuality and citation accuracy for long-form generations relative to these models.[1]



Source: Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection; https://arxiv.org/abs/2310.11511

Retrieval Augmented Fine-tuning (RAFT)

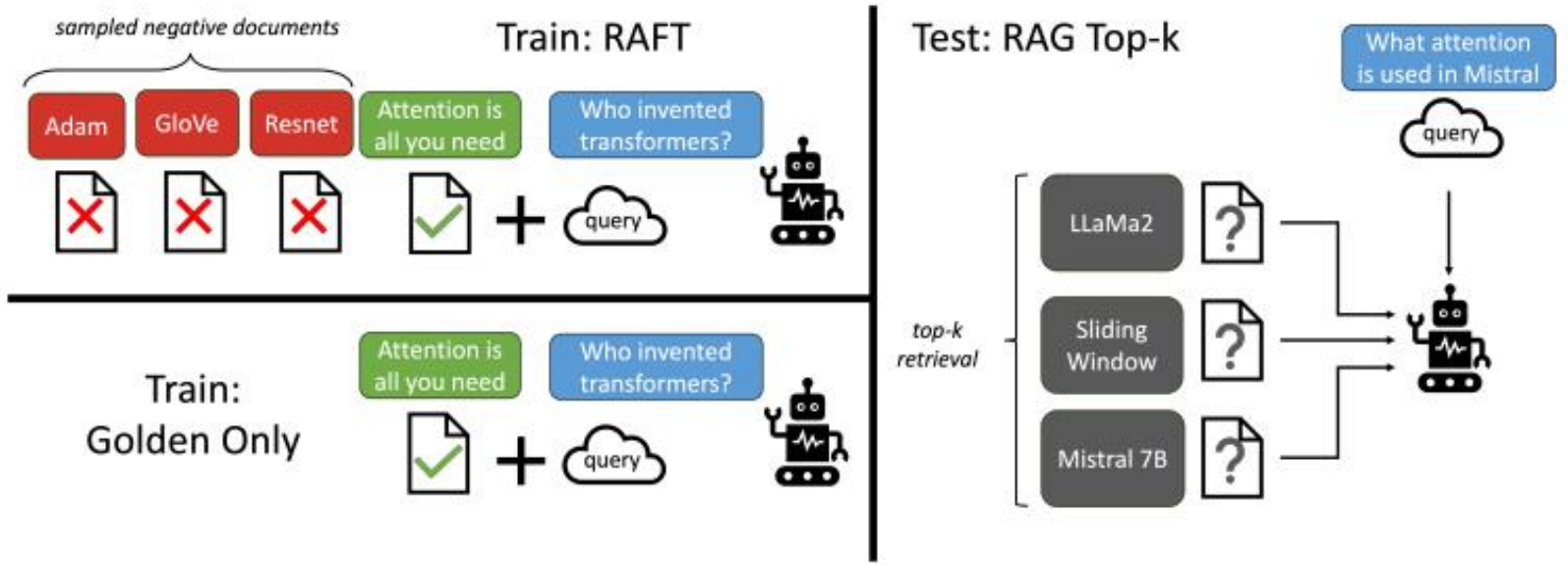


Figure 2: **Overview of our RAFT method.** The top-left figure depicts our approach of adapting LLMs to *reading* solution from a set of positive and distractor documents in contrast to standard RAG setup where models are trained based on the retriever outputs, which is a mixture of both memorization and reading. At test time, all methods follow the standard RAG setting, provided with a top-k retrieved documents in the context.

Source: RAFT: Adapting Language Model to Domain Specific RAG; https://arxiv.org/abs/2403.10131

Recent LLMs to Check Hallucinations

```python
from transformers import pipeline

pipe = pipeline("text-generation",
                model="PatronusAI/Llama-3-Patronus-Lynx-8B-Instruct",
                return_full_text=False, max_new_tokens=200, device='cuda')

context = """New Delhi is the capital of India and ......
             Until December 1911, Calcutta was the capital of India ..."""
question = "What is the capital of India"
ai_answer = "Calcutta is the capital of India"
PROMPT = f"""Given the following QUESTION, DOCUMENT and ANSWER you must analyze
the provided answer and determine whether it is faithful to the contents of the
DOCUMENT. The ANSWER must not offer new information beyond the context provided
in the DOCUMENT. The ANSWER also must not contradict information provided in the
DOCUMENT. Output your final verdict by strictly following this format:
"PASS" if the answer is faithful to the DOCUMENT and "FAIL" if the answer is not
faithful to the DOCUMENT.
Show your reasoning.
            --
            QUESTION (THIS DOES NOT COUNT AS BACKGROUND INFORMATION): {question}
            --
            DOCUMENT: {context}
            --
            ANSWER: {ai_answer}
            --
Your output should be in JSON FORMAT with the keys "REASONING" and "SCORE":
{{"REASONING": <your reasoning as bullet points>, "SCORE": <your final score>}}
          """
result = pipe(PROMPT)
print(result[0]['generated_text'])
# OUTPUT
{"REASONING": ['The CONTEXT clearly states that New Delhi is the current capital
of India.', ..., 'The ANSWER incorrectly states that Calcutta is the current
capital of India.', 'This is not faithful to the CONTEXT ...'],
 "SCORE": FAIL}
```

- GPT-4o from OpenAI

- Lynx from PatronusAI

Source: Lynx: An Open Source Hallucination Evaluation Model; https://arxiv.org/abs/2407.08488

Key Takeaways

RAG is still very much a retrieval problem

Build an evaluation dataset and always evaluate your RAG system

Explore various chunking and retrieval strategies, don’t stick to default settings

Even with Long Context LLMs, RAG isn’t going anywhere (for now)

Agentic RAG systems and domain-specific fine-tuned RAG systems are the future

Thank You