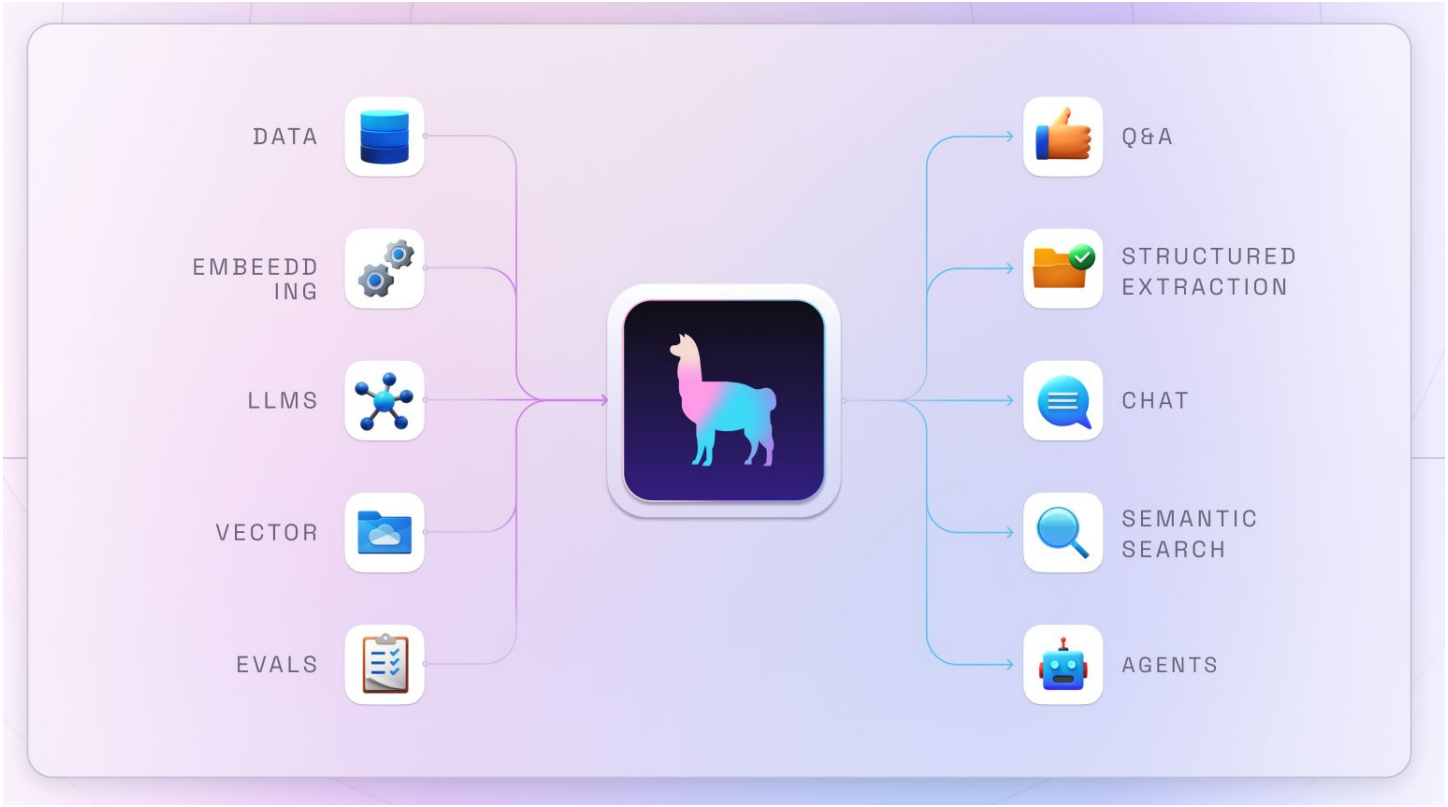
An introduction to RAG in 2024

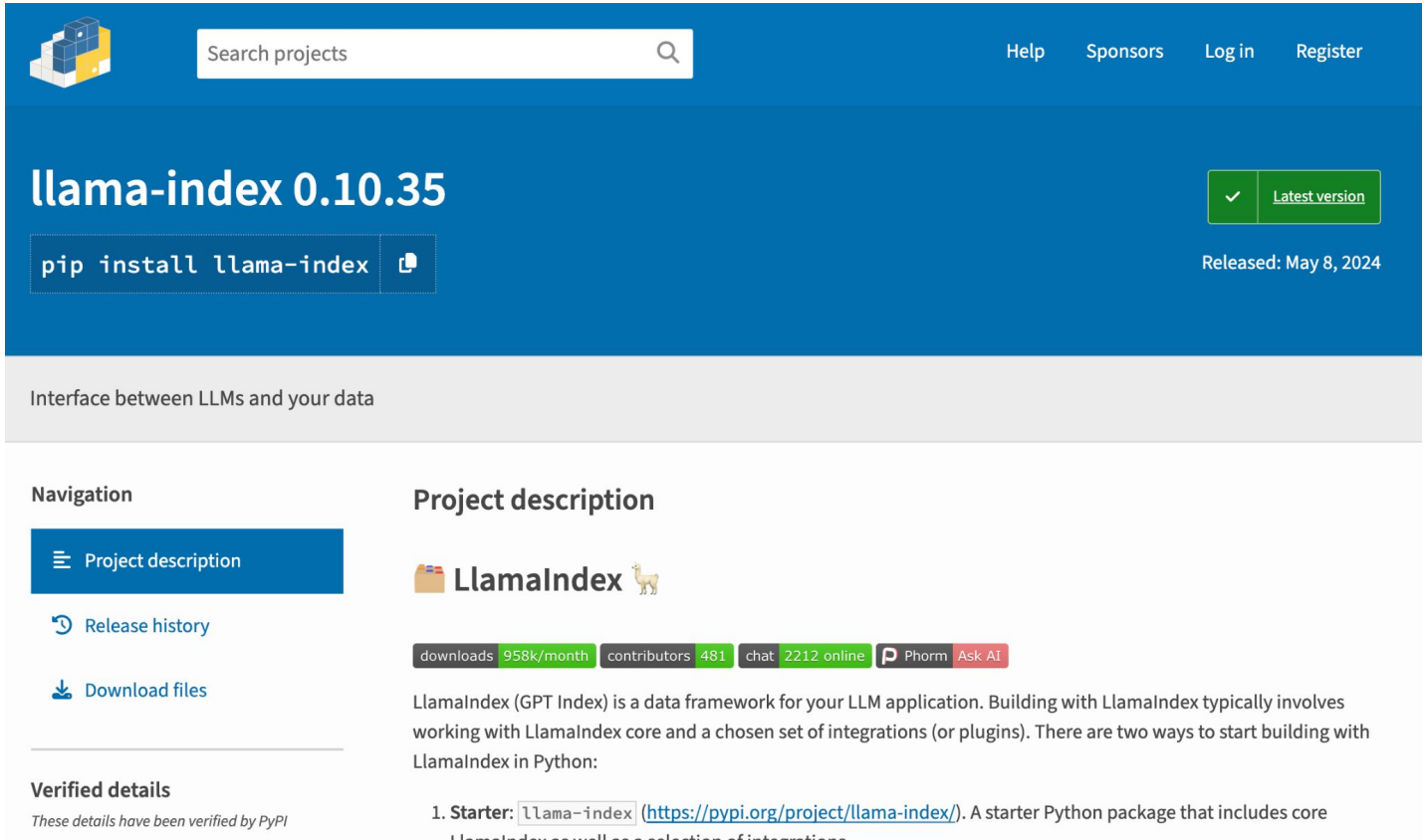
Pierre-Loic Doulcet, Founding AI engineer, LlamaIndex

@EPFL Lauzhack - May 8th 2024

LlamaIndex: Context Augmentation for your LLM app



LlamaIndex python framework



LlamaIndex typescript framework

llamaindex TS

0.3.8 • Public • Published 19 hours ago

Readme | Code Beta | 38 Dependencies | 16 Dependents | 113 Versions

LlamaIndex.TS

npm v0.3.8 | license MIT | downloads 61k/month | chat 2211 online

LlamaIndex is a data framework for your LLM application.

Use your own data with large language models (LLMs, OpenAI ChatGPT and others) in Typescript and Javascript.

Documentation: https://ts.llamaindex.ai/

Install

```
> npm i llamaindex
```

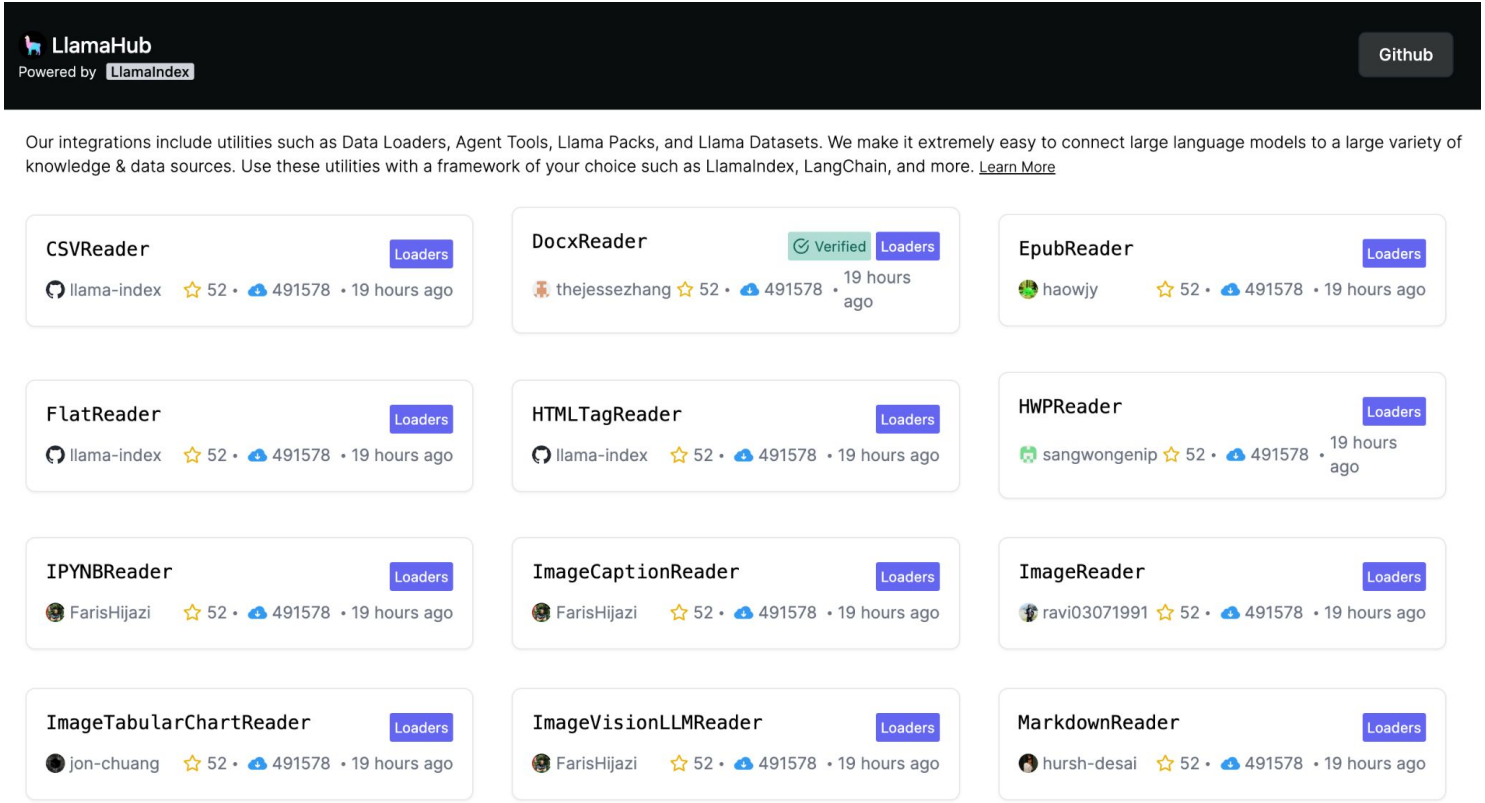
Repository

github.com/run-llama/LlamaIndexTS

Homepage

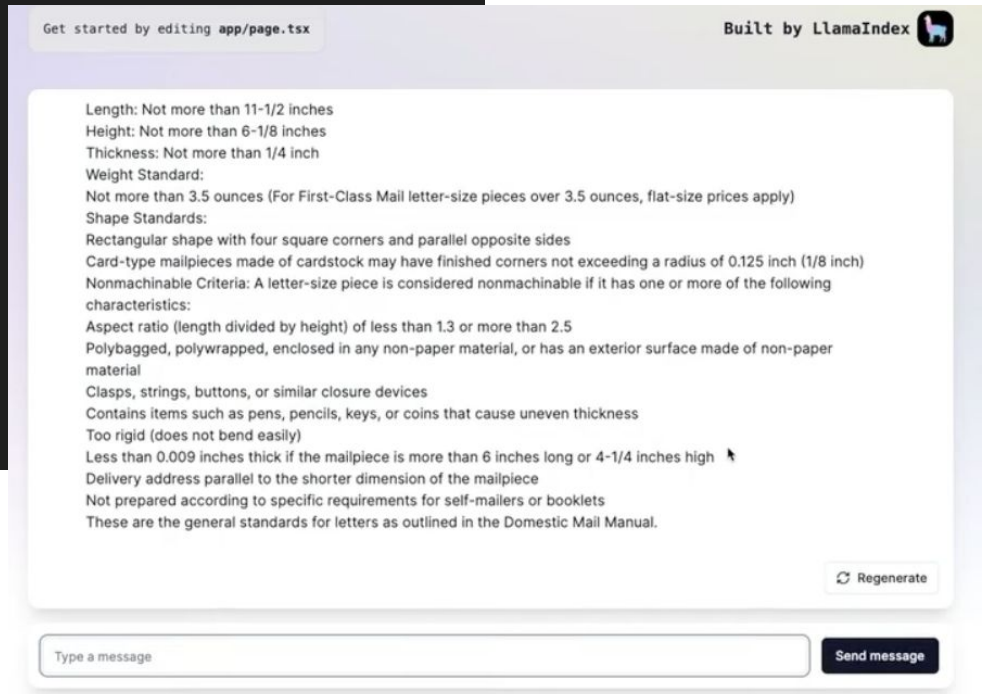
github.com/run-llama/LlamaIndexTS#r...

LlamaHub.ai

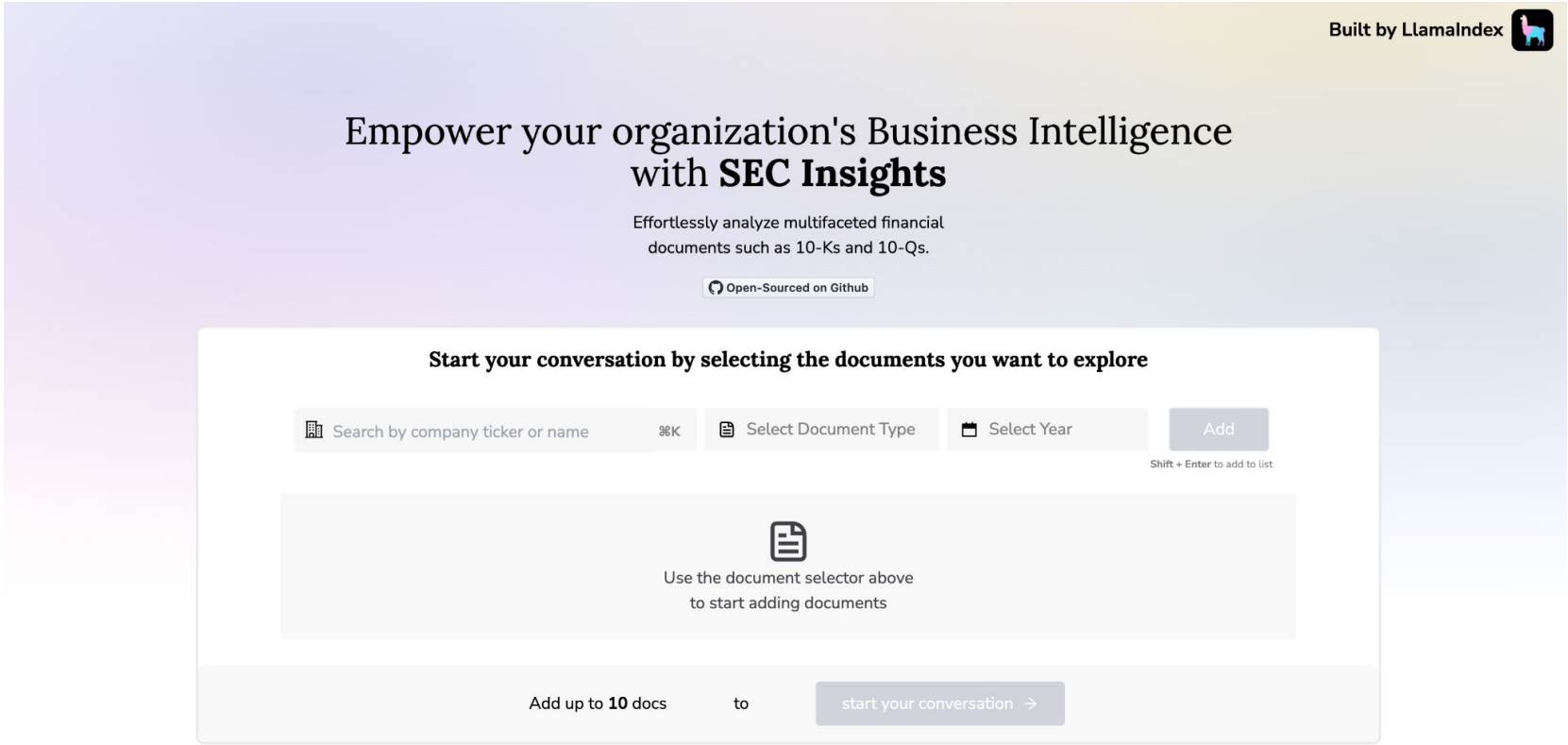

LlamaHub
Powered by LlamaIndex
Github
Our integrations include utilities such as Data Loaders, Agent Tools, Llama Packs, and Llama Datasets. We make it extremely easy to connect large language models to a large variety of knowledge & data sources. Use these utilities with a framework of your choice such as LlamaIndex, LangChain, and more. Learn More
CSVReader
Loaders
llama-index 52 • 491578 • 19 hours ago
DocxReader
Verified
Loaders
thejessezhang 52 • 491578 • 19 hours ago
EpubReader
Loaders
haowjy 52 • 491578 • 19 hours ago
FlatReader
Loaders
llama-index 52 • 491578 • 19 hours ago
HTMLTagReader
Loaders
llama-index 52 • 491578 • 19 hours ago
HWPReader
Loaders
sangwongenip 52 • 491578 • 19 hours ago
IPYNBReader
Loaders
FarisHijazi 52 • 491578 • 19 hours ago
ImageCaptionReader
Loaders
FarisHijazi 52 • 491578 • 19 hours ago
ImageReader
Loaders
ravi03071991 52 • 491578 • 19 hours ago
ImageTabularChartReader
Loaders
jon-chuang 52 • 491578 • 19 hours ago
ImageVisionLLMReader
Loaders
FarisHijazi 52 • 491578 • 19 hours ago
MarkdownReader
Loaders
hursh-desai 52 • 491578 • 19 hours ago

Create llama - jump start your llm app development.

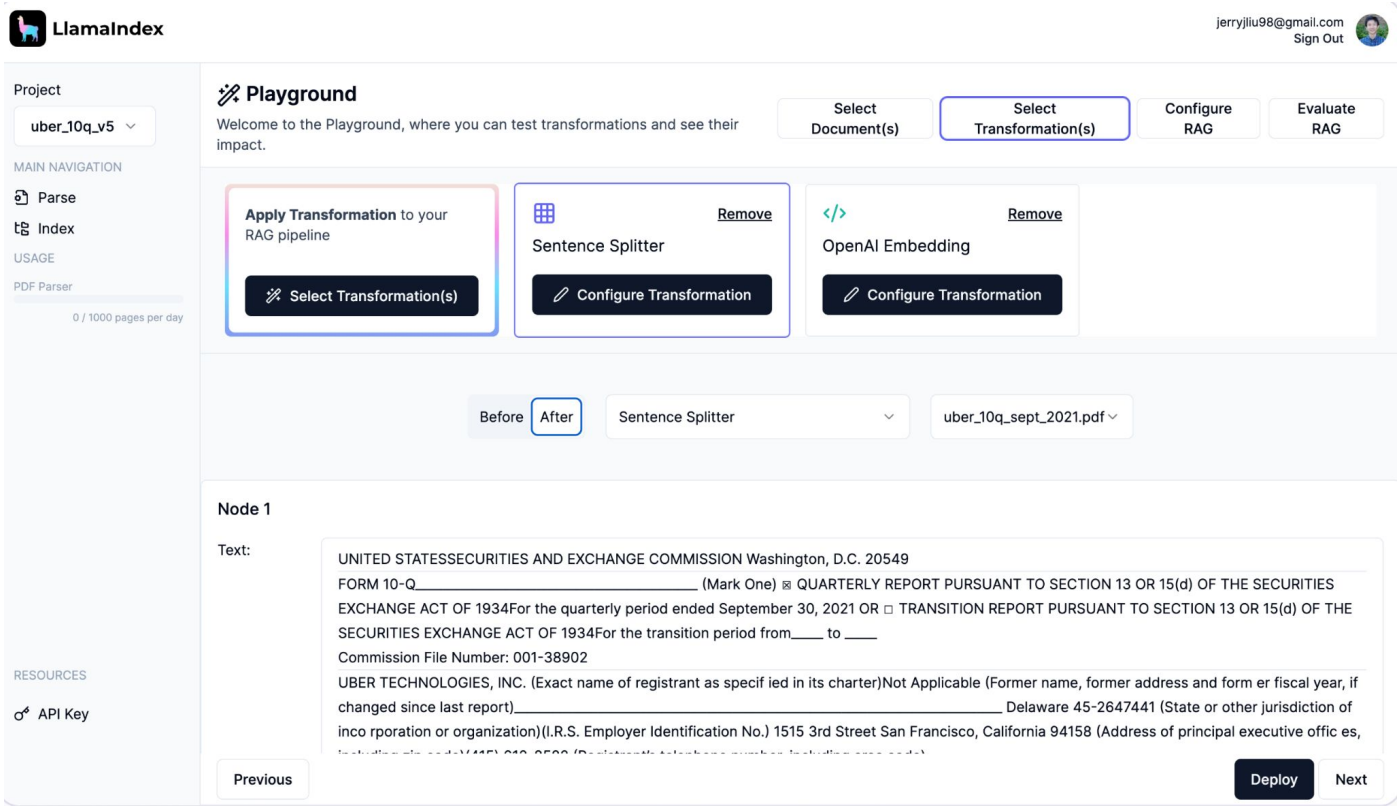
```
>> npm create llama@latest
Need to install the following packages:
  create-llama@latest
Ok to proceed? (y) y
✔ What is your project named? … my-app
✔ Which template would you like to use? › Chat
✔ Which framework would you like to use? › NextJS
✔ Would you like to set up observability? › No
✔ Please provide your OpenAI API key (leave blank to skip):
✔ Which data source would you like to use? › Use an example
✔ Would you like to add another data source? › No
✔ Would you like to use LlamaParse (improved parser for RAG
✔ Would you like to use a vector database? › No, just store
? How would you like to proceed? › - Use arrow-keys. Return
    Just generate code (~1 sec)
❯   Start in VSCode (~1 sec)
    Generate code and install dependencies (~2 min)
    Generate code, install dependencies, and run the app (~2
```



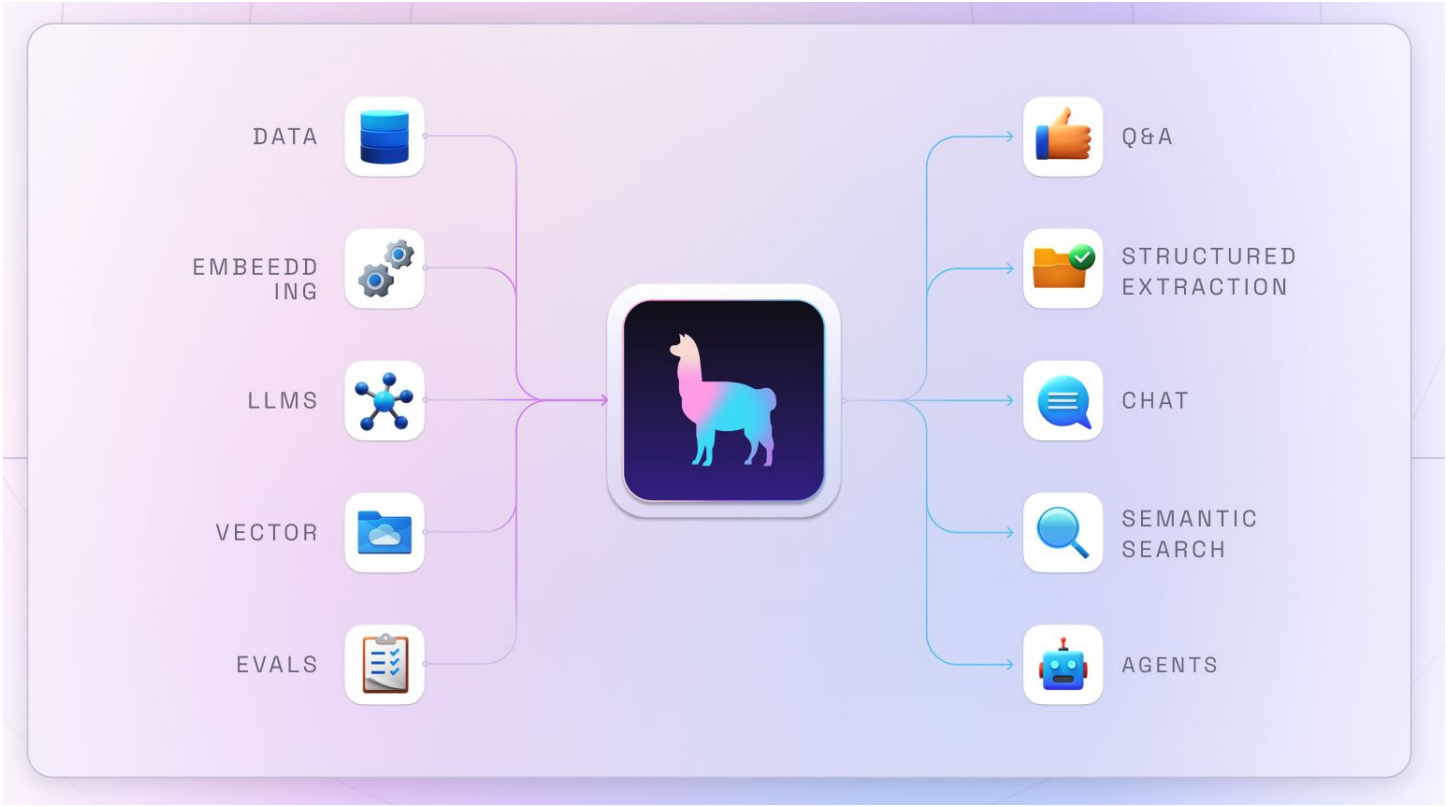
secInsights.ai : Our advanced RAG Demo



LlamaCloud - Managed RAG SAAS
LlamaParse - Doc to structured data Parser



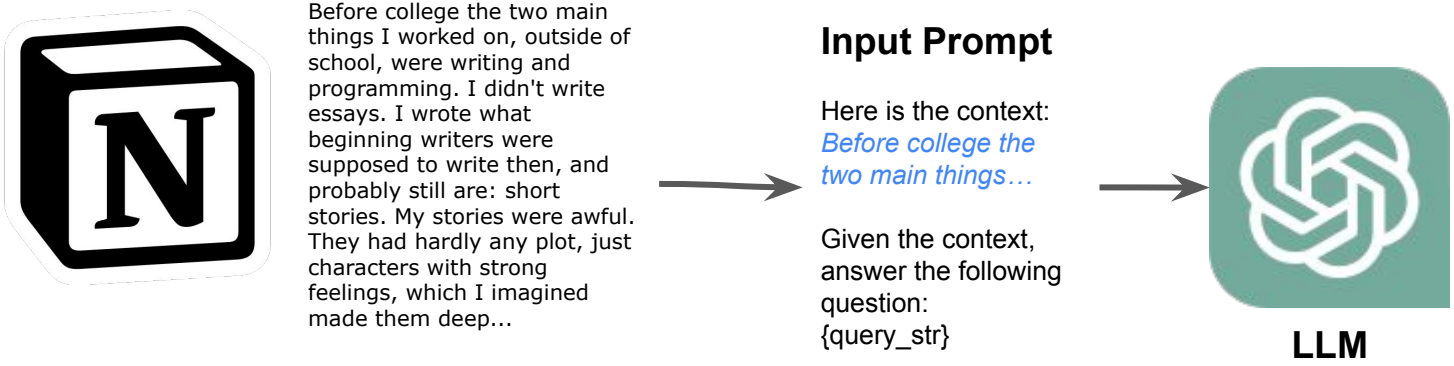
LlamaIndex: Context Augmentation for your LLM app



RAG?
(Retrieval Augmented Generation)

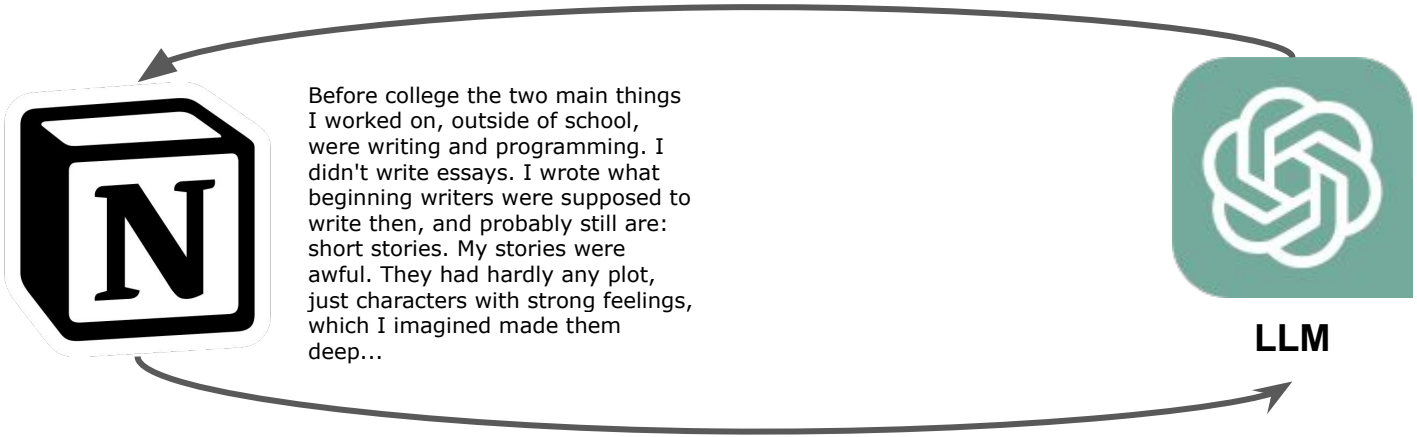
Paradigms for inserting knowledge

Retrieval Augmentation - Fix the model, put context into the prompt



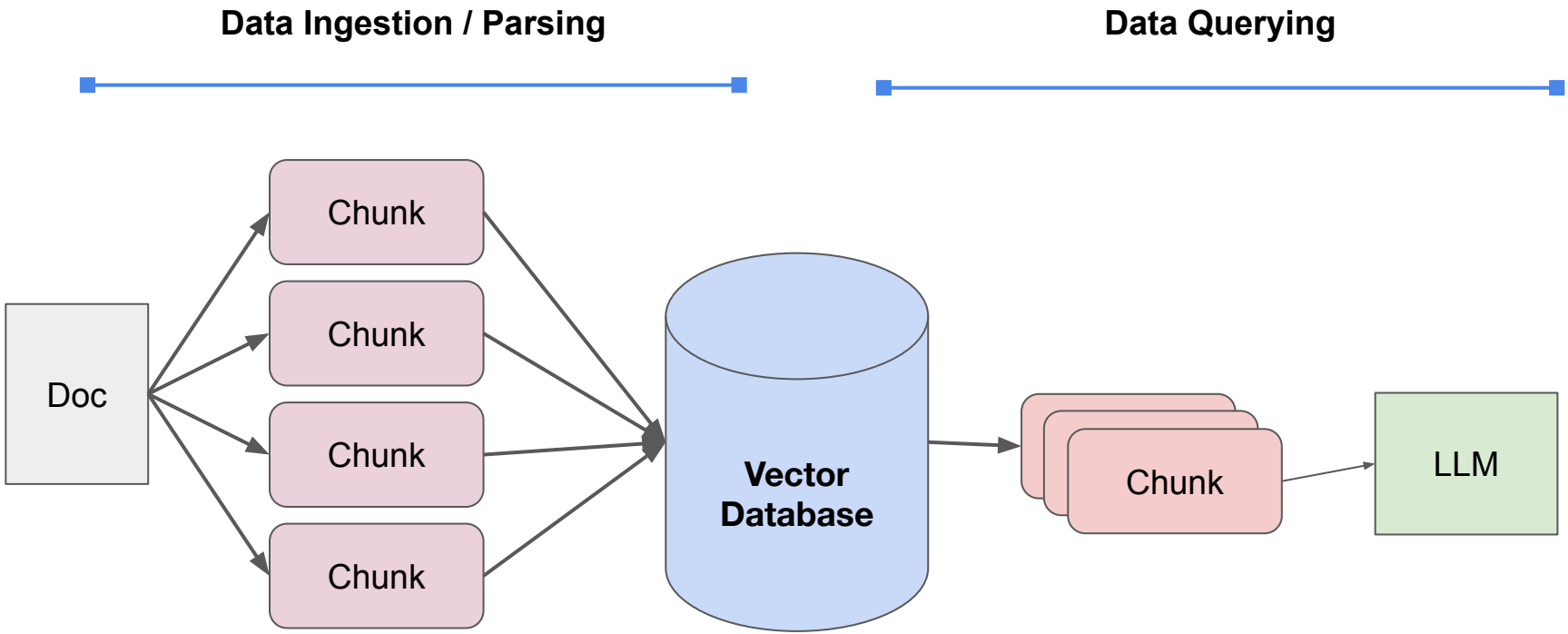
Paradigms for inserting knowledge

Fine-tuning - baking knowledge into the weights of the network



RAG Stack

Current RAG Stack for building a QA System



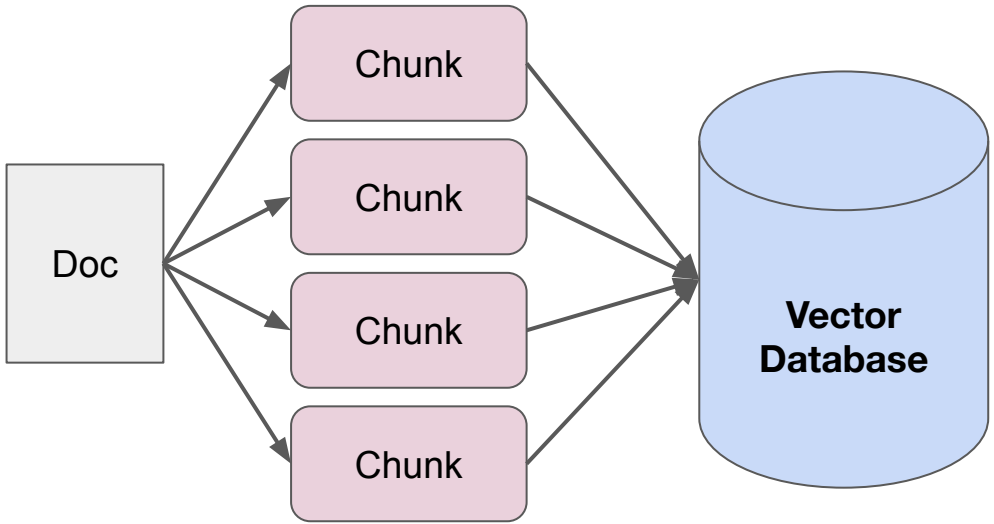
5 Lines of Code in LlamaIndex!

Current RAG Stack for building a QA System

```python
from llama_index.core import VectorStoreIndex, SimpleDirectoryReader

documents = SimpleDirectoryReader("data").load_data()
index = VectorStoreIndex.from_documents(documents)
query_engine = index.as_query_engine()
response = query_engine.query("What did the author do growing up?")

print(response)
```

5 Lines of Code in LlamaIndex!

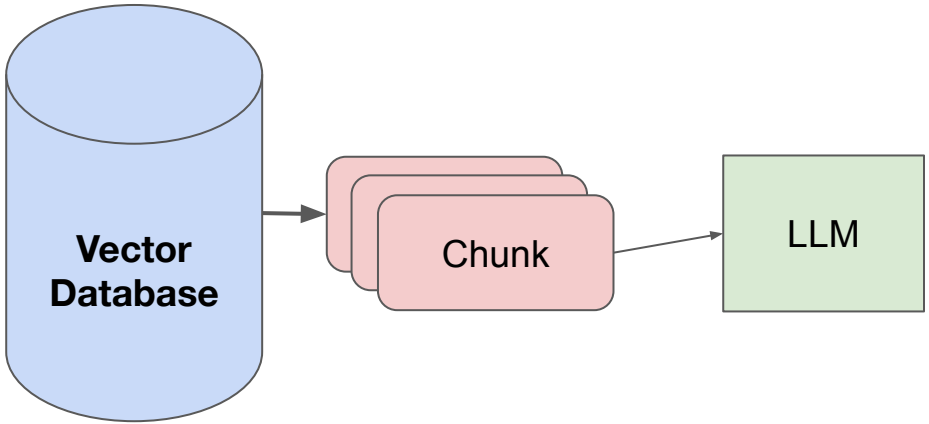
Current RAG Stack (Data Ingestion/Parsing)



Process:

- Split up document(s) into even chunks.
- Each chunk is a piece of raw text.
- Generate embedding for each chunk (e.g. OpenAI embeddings, sentence_transformer)
- Store each chunk into a vector database

Current RAG Stack (Querying)



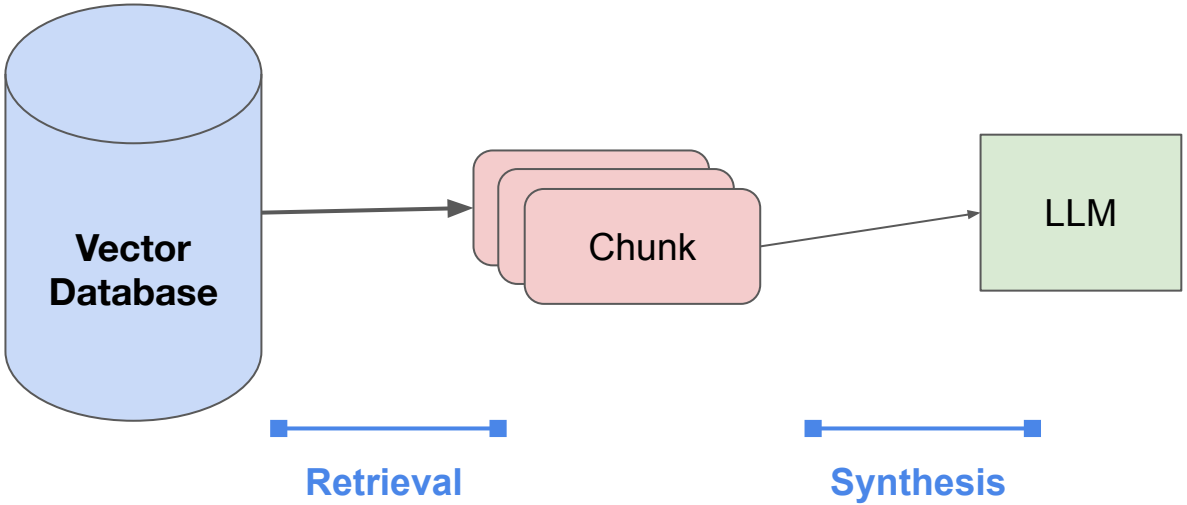
Process:

- Find top-k most similar chunks from vector database collection
- Plug into LLM response synthesis module

Current RAG Stack (Querying)

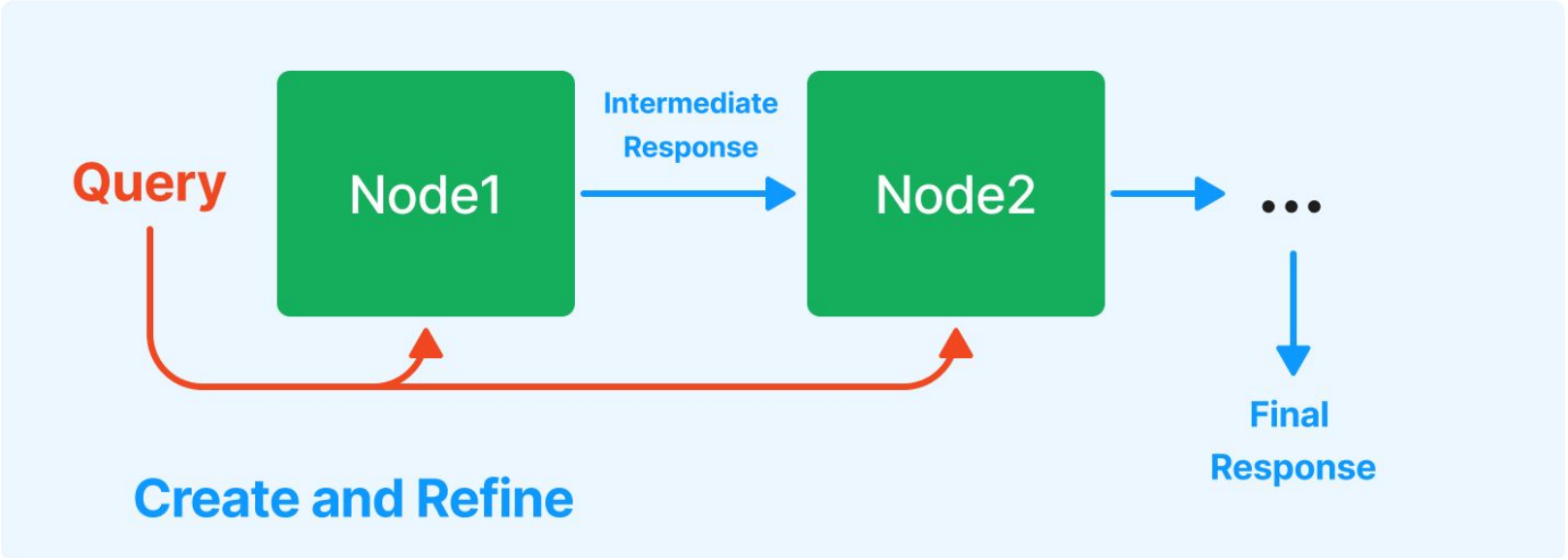
Process:

- Find top-k most similar chunks from vector database collection
- Plug into LLM **response synthesis module**



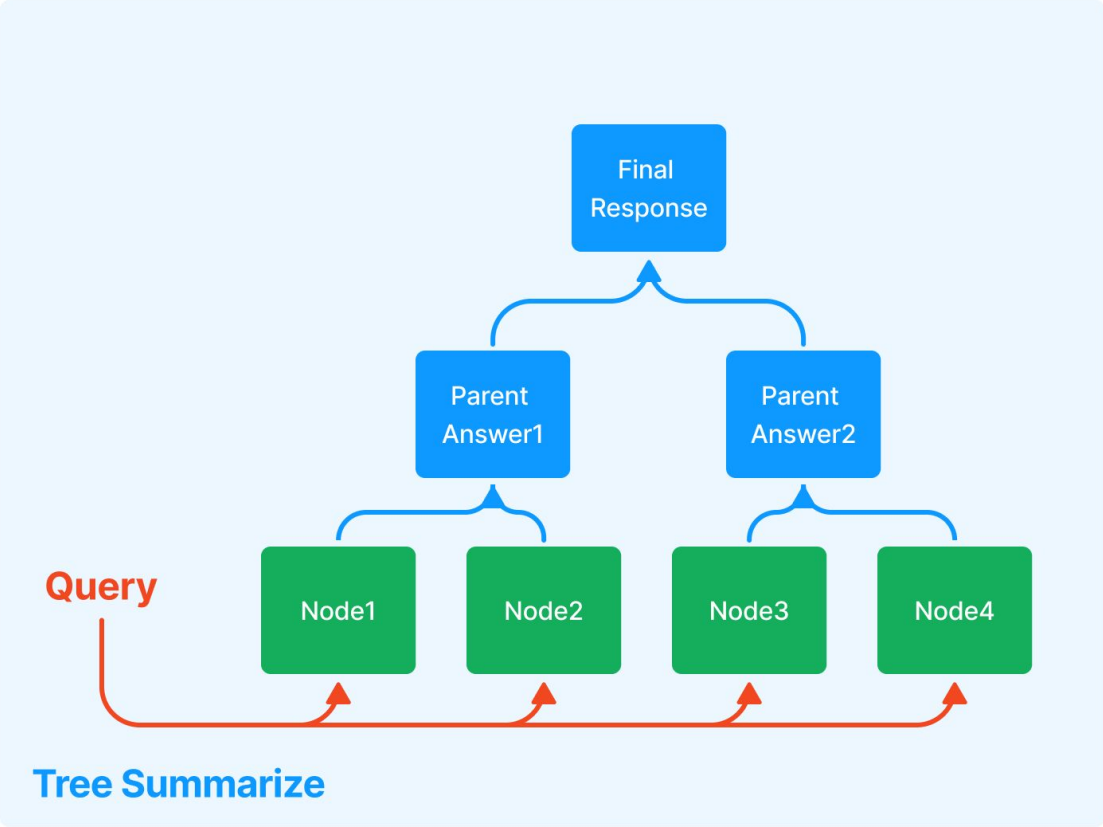
Response Synthesis

Create and refine



Response Synthesis

Tree Summarize

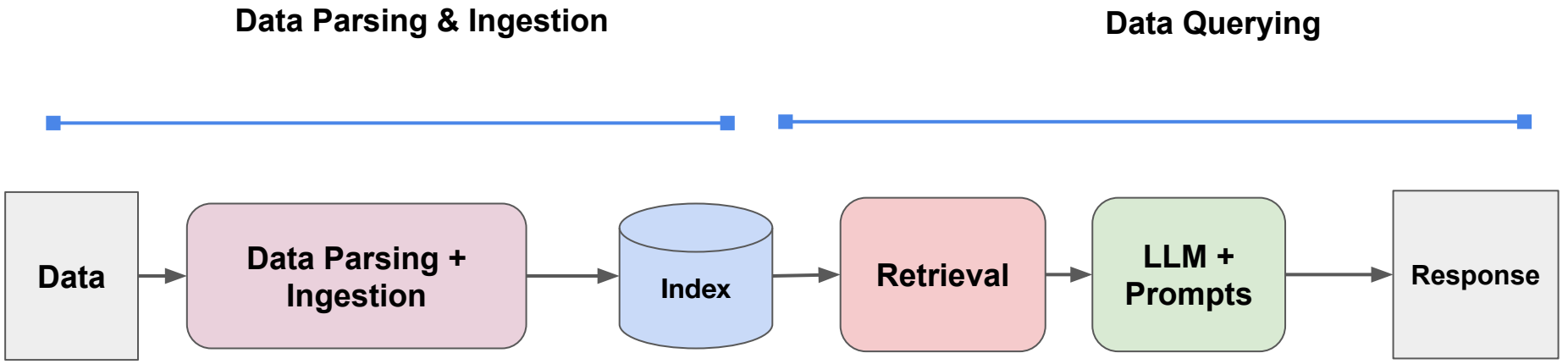


Quickstart

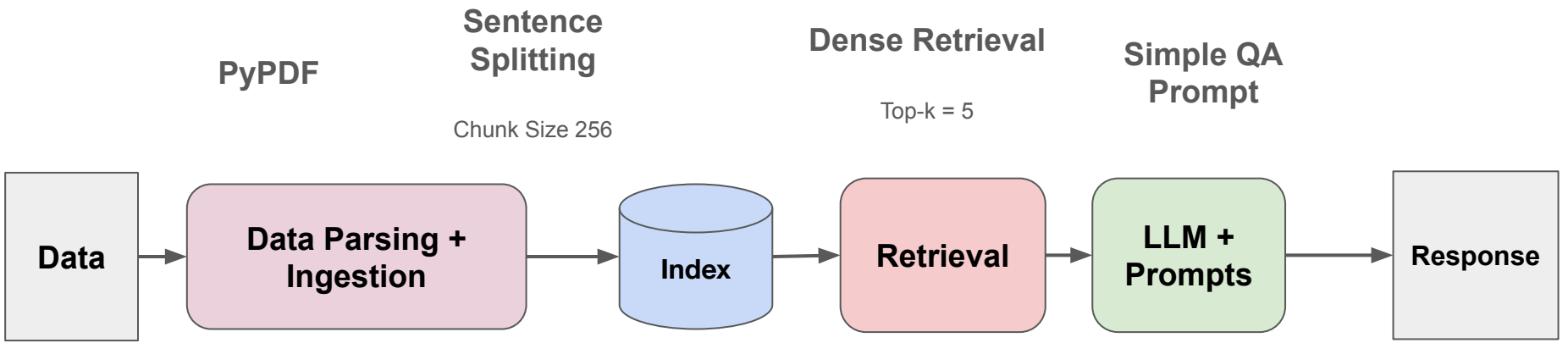
https://colab.research.google.com/drive/1knQpGJLHj-LTTHqlZhgcjDH5F_nJIiY0?usp=sharing

Challenges with “Naive” RAG

RAG


Data Parsing & Ingestion
Data Querying
Data
Data Parsing + Ingestion
Index
Retrieval
LLM + Prompts
Response

Naive RAG


PyPDF
Sentence Splitting
Chunk Size 256
Dense Retrieval
Top-k = 5
Simple QA Prompt
Data
Data Parsing + Ingestion
Index
Retrieval
LLM + Prompts
Response

Easy to Prototype, Hard to Productionize

Naive RAG approaches tend to work well for **simple** questions over a **simple, small** set of documents.

- “What are the main risk factors for Tesla?” (over Tesla 2021 10K)
- “What did the author do during his time at YC?” (Paul Graham essay)

Easy to Prototype, Hard to Productionize

But productionizing RAG over **more questions** and a **larger set of data** is hard!

Failure Modes:

- Response Quality: Bad Retrieval, Bad Response Generation
- Hard to Improve: Too many parameters to tune
- Systems: Latency, Cost, Security

Easy to Prototype, Hard to Productionize

But productionizing RAG over **more questions** and a **larger set of data** is hard!

Failure Modes:

- **Response Quality:** Bad Retrieval, Bad Response Generation
- **Hard to Improve:** Too many parameters to tune
- **Systems:** Latency, Cost, Security

Challenges with Naive RAG (Response Quality)

- Bad Retrieval
  - **Low Precision:** Not all chunks in retrieved set are relevant
    - Hallucination + Lost in the Middle Problems
  - **Low Recall:** Now all relevant chunks are retrieved.
    - Lacks enough context for LLM to synthesize an answer
  - **Outdated information:** The data is redundant or out of date.

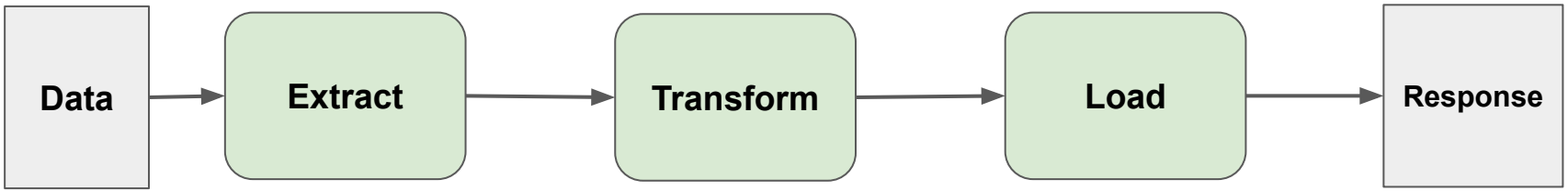
Challenges with Naive RAG (Response Quality)

- Bad Retrieval
  - **Low Precision:** Not all chunks in retrieved set are relevant
    - Hallucination + Lost in the Middle Problems
  - **Low Recall:** Now all relevant chunks are retrieved.
    - Lacks enough context for LLM to synthesize an answer
  - **Outdated information:** The data is redundant or out of date.
- Bad Response Generation
  - **Hallucination:** Model makes up an answer that isn't in the context.
  - **Irrelevance:** Model makes up an answer that doesn't answer the question.
  - **Toxicity/Bias:** Model makes up an answer that's harmful/offensive.

Difference with Traditional Software

Traditional software is defined by a set of programmatic rules.

Given an input, you can easily reason about the expected output.

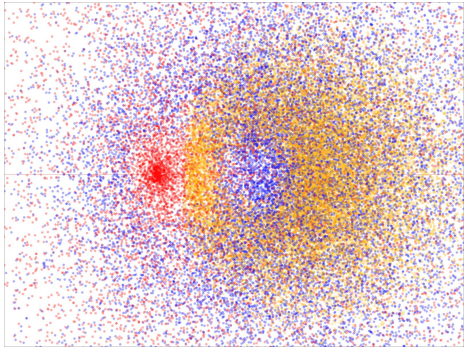


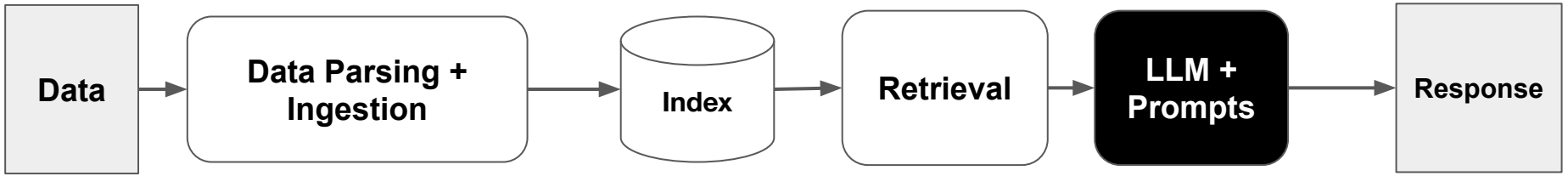
Difference with Traditional Software

AI-powered software is defined by a **black-box set of parameters.**

It is really hard to reason about what the function space looks like.

The model parameters are tuned, the surrounding parameters (prompt templates) are not.

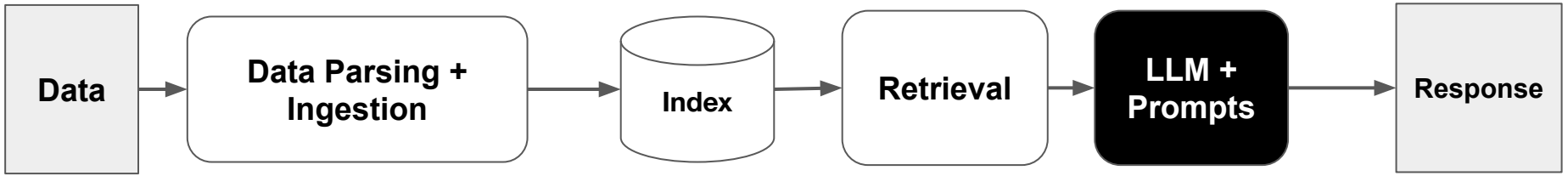




Difference with Traditional Software

If one component of the system is a black-box, all components of the system become black boxes.

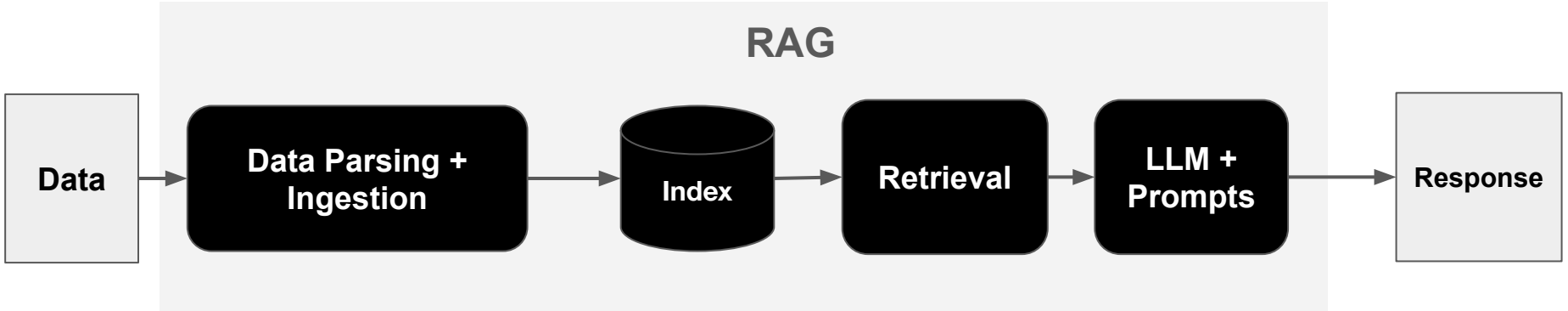
The more components, the more parameters you have to tune.



Difference with Traditional Software

If one component of the system is a black-box, all components of the system become black boxes.

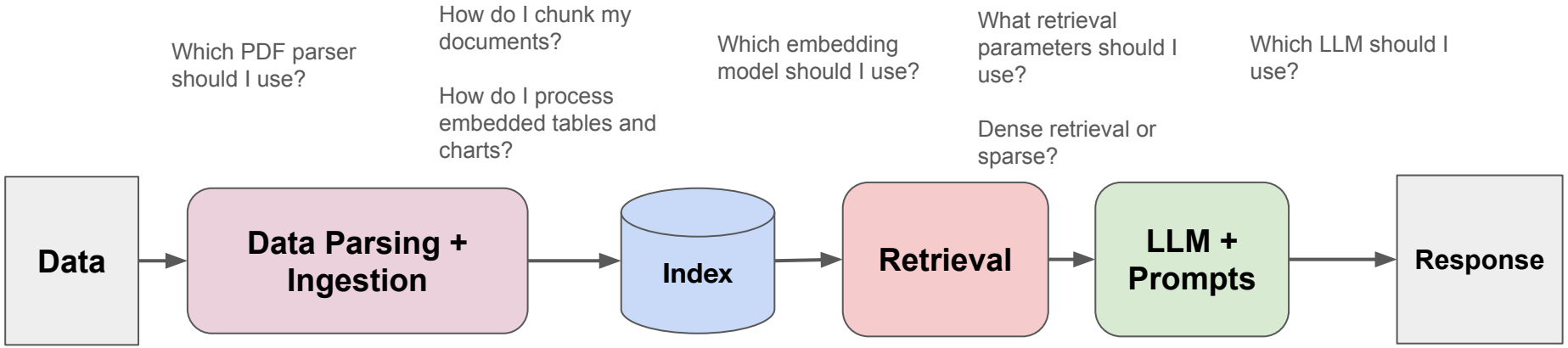
Every parameter affects the performance of the end system.



There's Too Many Parameters

Every parameter affects the performance of the entire RAG pipeline.

Which parameters should a user tune?
There's too many options!



Mapping Pain Points to Solutions

Solution

Categorize by pain point, and establish best practices

Solution

Categorize by pain point, and establish best practices

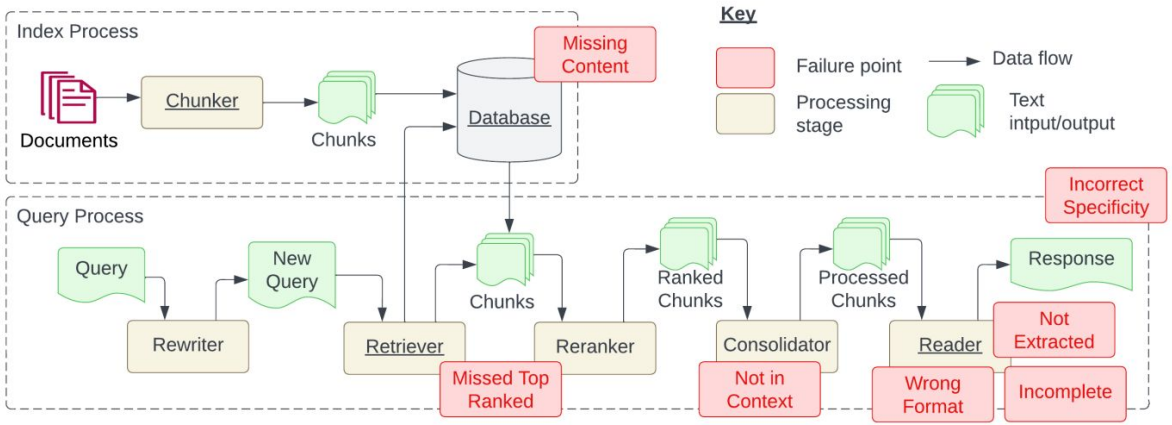
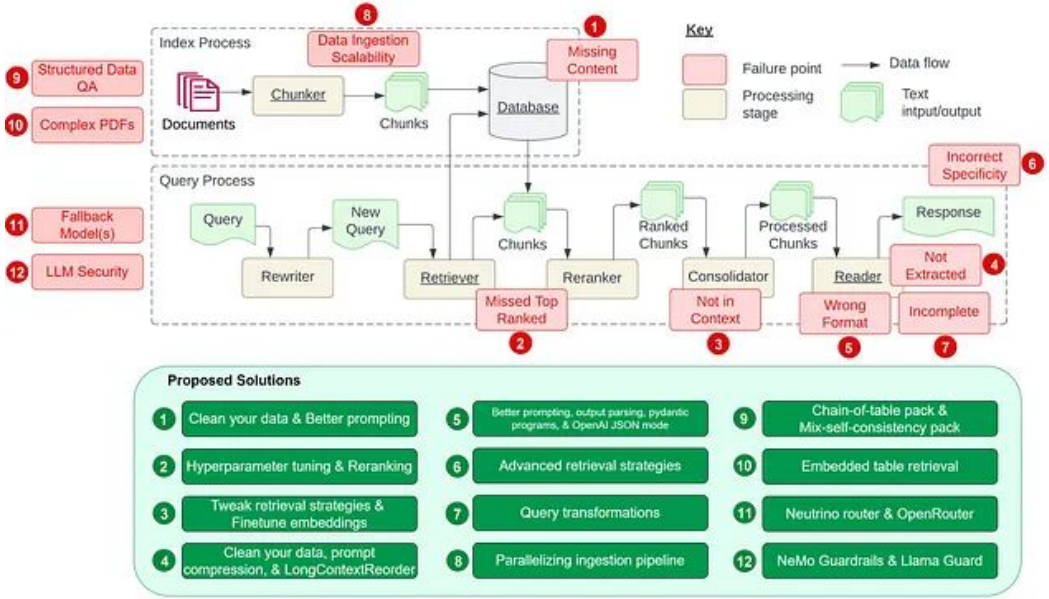


Figure 1: Indexing and Query processes required for creating a Retrieval Augmented Generation (RAG) system. The indexing process is typically done at development time and queries at runtime. Failure points identified in this study are shown in red boxes. All required stages are underlined. Figure expanded from [19].

"Seven Failure Points When Engineering a Retrieval Augmented Generation System", Barnett et al.

Solution

Categorize by pain point, and establish best practices



[“12 RAG Pain Points and Proposed Solutions”, by Wenqi Glantz]

Pain Points

Response Quality Related

1. Context Missing in the Knowledge Base
2. Context Missing in the Initial Retrieval Pass
3. Context Missing After Reranking
4. Context Not Extracted
5. Output is in Wrong Format
6. Output has Incorrect Level of Specificity
7. Output is Incomplete

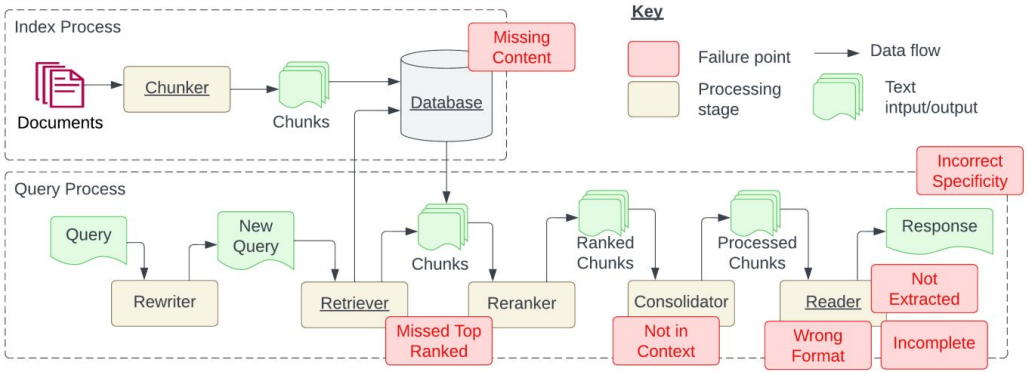


Figure 1: Indexing and Query processes required for creating a Retrieval Augmented Generation (RAG) system. The indexing process is typically done at development time and queries at runtime. Failure points identified in this study are shown in red boxes. All required stages are underlined. Figure expanded from [19].

Pain Points

Scalability

8. Can't Scale to Larger Data Volumes

11. Rate-Limit Errors

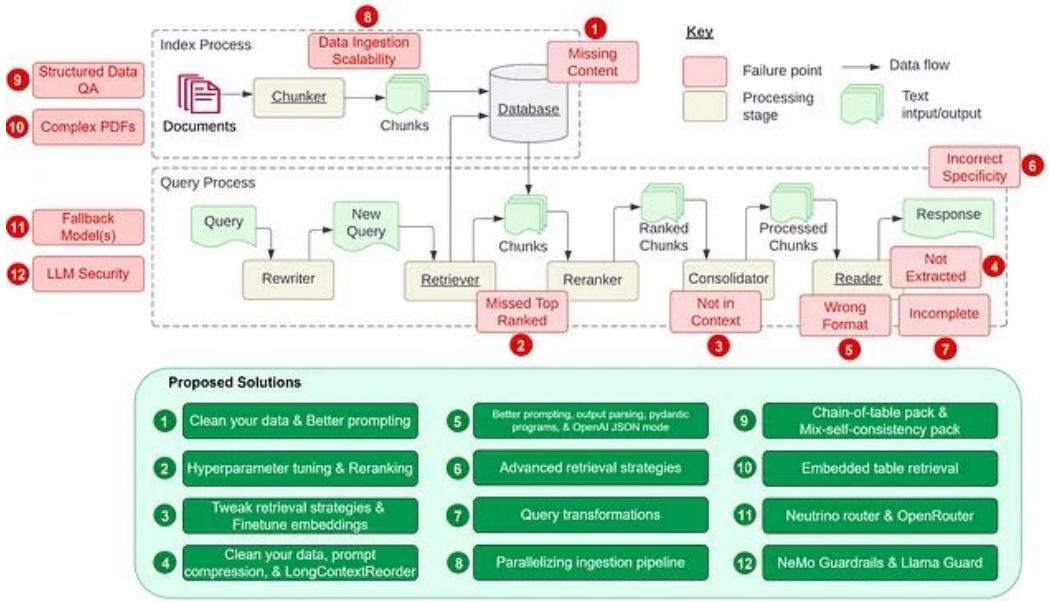
Security

12. LLM Security

Use Case Specific

9. Ability to QA Tabular Data

10. Ability to Parse PDFs



Pain Points

Scalability

8. Can't Scale to Larger Data Volumes

11. Rate-Limit Errors

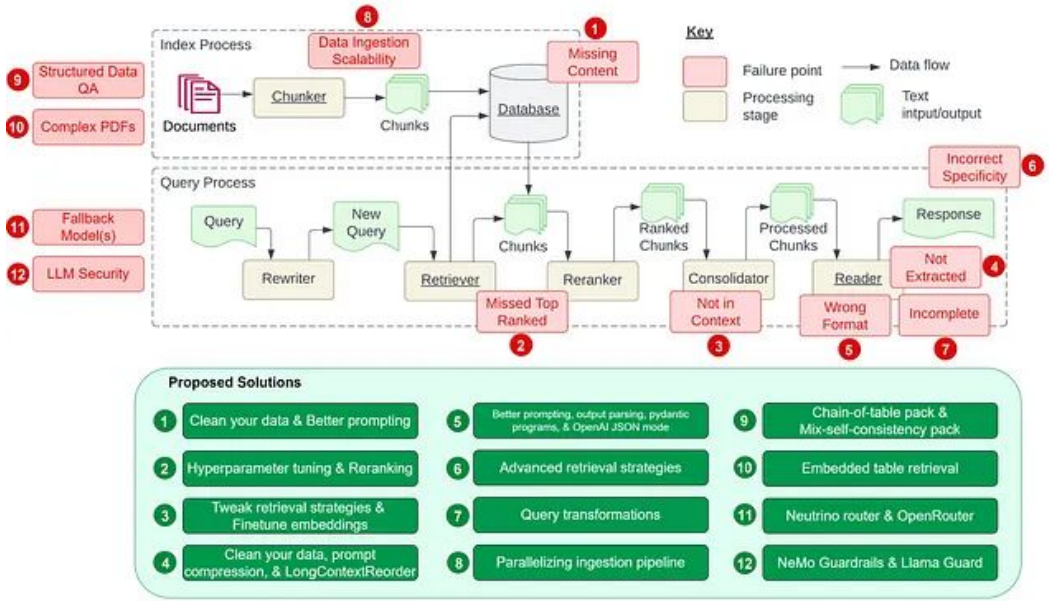
Security

12. LLM Security

Use Case Specific

9. Ability to QA Tabular Data

10. Ability to Parse PDFs



Let’s figure out solutions

1. Context Missing in the Knowledge Base

Clean your data: Pick a good document parser (more on this later!)

Add in Metadata: inject global context to each chunk

Keep your data updated: Setup a recurring data ingestion pipeline. Upsert documents to prevent duplicates.

```python
from llama_index.core.node_parser import SentenceSplitter
from llama_index.core.extractors import (
    SummaryExtractor,
    QuestionsAnsweredExtractor,
    TitleExtractor,
    KeywordExtractor,
)
from llama_index.extractors.entity import EntityExtractor

transformations = [
    SentenceSplitter(),
    TitleExtractor(nodes=5),
    QuestionsAnsweredExtractor(questions=3),
    SummaryExtractor(summaries=["prev", "self"]),
    KeywordExtractor(keywords=10),
    EntityExtractor(prediction_threshold=0.5),
]
```

Metadata Extraction
https://docs.llamaindex.ai/en/stable/module_guides/indexing/metadata_extraction/

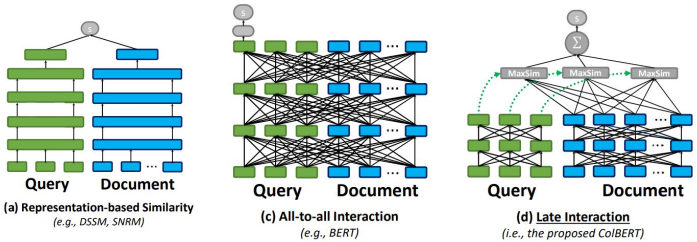
```python
from llama_index.core.ingestion import IngestionPipeline
from llama_index.core.storage.docstore import SimpleDocumentStore

pipeline = IngestionPipeline(
    transformations=[...], docstore=SimpleDocumentStore()
)
```

Ingestion pipeline / DocStore:
https://docs.llamaindex.ai/en/stable/module_guides/loading/ingestion_pipeline/

2. Context Missing in the Initial Retrieval Pass

Solution: Hyperparameter tuning for chunk size and top-k

Solution: Reranking



📄 **Source: ColBERT**

```python
from llama_index.postprocessor.colbert_rerank import ColbertRerank

colbert_reranker = ColbertRerank(
    top_n=5,
    model="colbert-ir/colbertv2.0",
    tokenizer="colbert-ir/colbertv2.0",
    keep_retrieval_score=True,
)

query_engine = index.as_query_engine(
    similarity_top_k=10,
    node_postprocessors=[colbert_reranker],
)
response = query_engine.query(
    "What did Sam Altman do in this essay?",
)
```

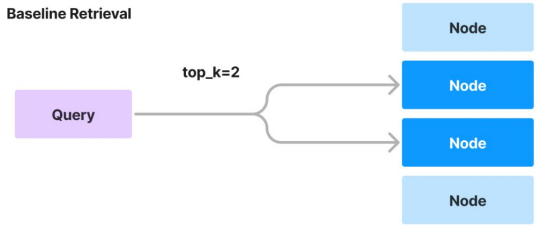
ColBERT Reranking

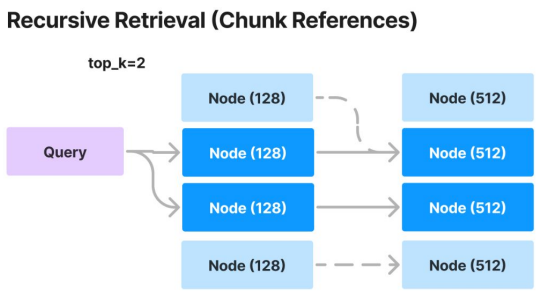
https://docs.llamaindex.ai/en/stable/examples/node_postprocessor/ColbertRerank/

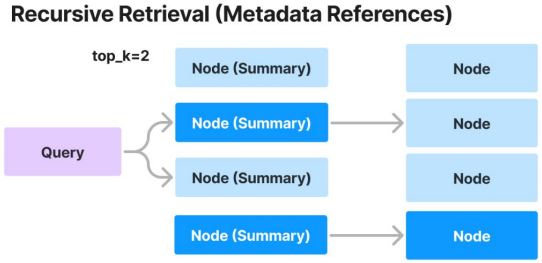
3. Context Missing After Reranking

Solution: try out fancier retrieval methods (small-to-big, auto-merging, auto-retrieval, ensembling, …)

Solution: fine-tune your embedding models to task-specific data





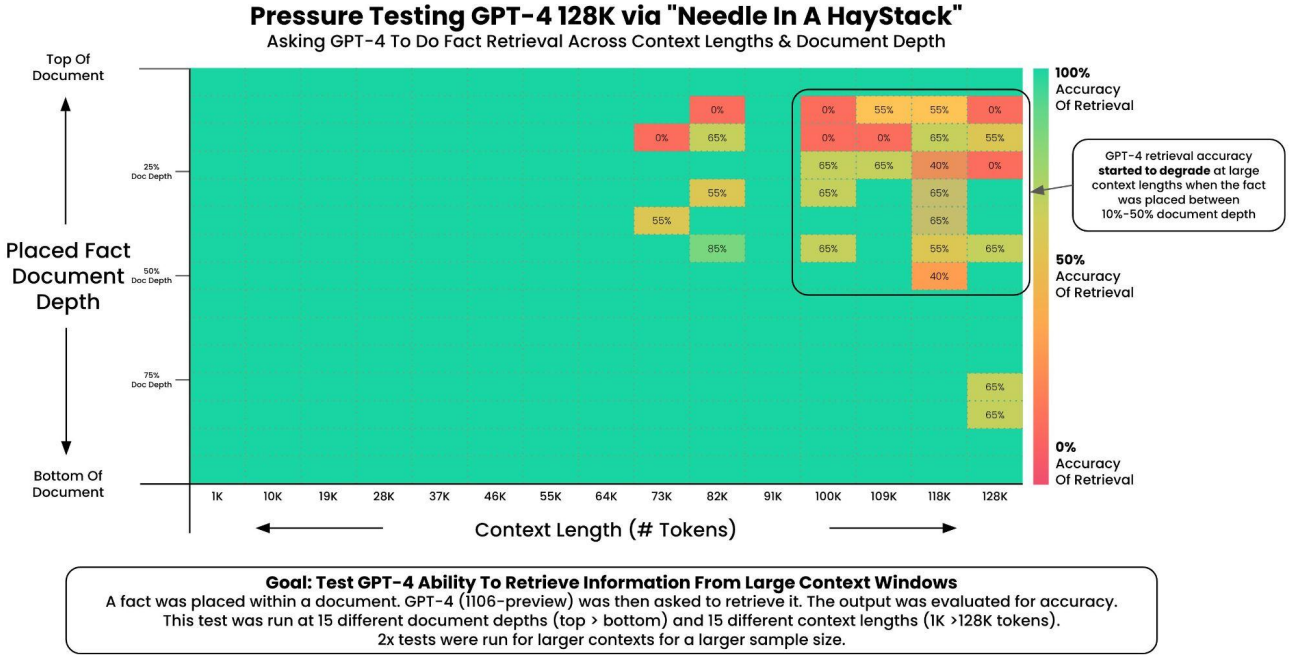


Recursive retrieval
https://docs.llamaindex.ai/en/stable/examples/retrievers/recursive_retriever_nodes/

4. Context is there, but not extracted by the LLM

The context is there, but the LLM doesn't understand it.

"Lost in the middle" Problems.



https://x.com/GregKamradt/status/1722386725635580292?s=20

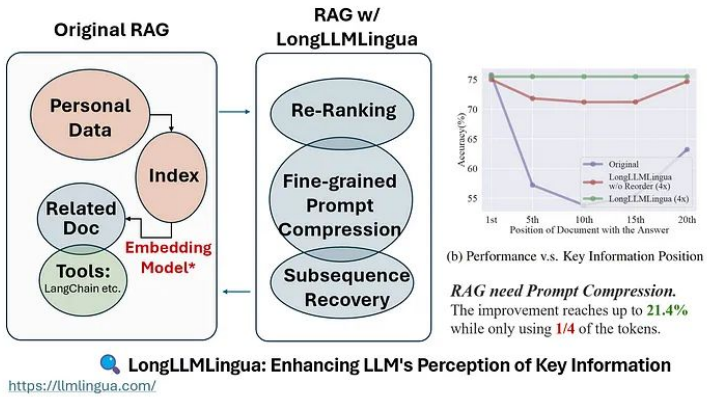
4. Context is there, but not extracted by the LLM

Solution: Prompt Compression

📄https://arxiv.org/abs/2310.06839

LongLLMLingua

https://docs.llamaindex.ai/en/stable/api_reference/postprocessor/longllmlingua/



LongLLMLingua by Jiang et al.

Solution: LongContextReorder

LongContextReorder

https://docs.llamaindex.ai/en/stable/examples/node_postprocessor/LongContextReorder/

Retrieved Set (Reverse Order)

Node 1: (0.98)
Node 2: (0.93)
Node 3: (0.84)
... 4 (0.81)
Node 5
...
...
Node N

Long Context Re-order

Node 1: (0.98)
Node 3: (0.84)
Node 5
...
Node N
...
Node 6
Node 4 (0.81)
Node 2: (0.93)

The ends matter the most

4. Context is there, but not extracted by the LLM

Financial tables - if you need to provide table to the LLM use MD

Markdown

		Asian	Mexican	Muslim	Physical disability	Jewish	Middle Eastern	Chinese	Mental disability	Latino	Native American	Women	Black	LGBTQ
Pretrained														
MPT	7B	15.40	33.55	23.54	17.09	26.12	23.20	16.25	17.63	28.40	19.52	24.34	25.04	20.03
	30B	15.74	31.49	19.04	21.68	26.82	30.60	13.87	24.36	**16.51**	32.68	**15.56**	25.21	20.32
Falcon	7B	**9.06**	**18.30**	**17.34**	**8.29**	**19.40**	**12.99**	**10.07**	**10.26**	18.03	**15.34**	17.32	**16.75**	**15.73**
	40B	19.59	29.61	25.83	13.54	29.85	23.40	25.55	29.10	23.20	17.31	21.05	23.11	23.52
Llama 1	7B	16.65	30.72	26.82	16.58	26.49	22.27	17.16	19.71	28.67	21.71	29.80	23.01	19.37
	13B	18.80	32.03	25.18	14.72	28.54	21.11	18.76	15.71	30.42	20.52	27.15	25.21	21.85
	33B	16.87	32.24	21.53	16.24	28.54	22.04	19.91	18.27	29.88	18.13	25.90	24.53	19.37
	65B	14.27	31.59	21.90	14.89	23.51	22.27	17.16	18.91	28.40	19.32	28.71	22.00	20.03
Llama 2	7B	16.53	31.15	22.63	15.74	26.87	19.95	15.79	19.55	25.03	18.92	21.53	22.34	20.20
	13B	21.29	37.25	22.81	17.77	32.65	24.13	21.05	20.19	35.40	27.69	26.99	28.26	23.84
	34B	16.76	29.63	23.36	14.38	27.43	19.49	**18.54**	17.31	26.38	18.73	22.78	**21.66**	19.04
	70B	21.29	32.90	25.91	16.92	30.60	21.35	16.93	21.47	**30.42**	20.12	31.05	28.43	22.35
Fine-tuned														
ChatGPT		**0.23**	**0.22**	**0.18**	**0**	**0.19**	**0**	**0.46**	**0**	**0.13**	**0**	**0.47**	**0**	**0.66**
MPT-instruct	7B	15.86	28.76	11.31	9.64	18.84	14.62	15.33	16.51	25.3	13.94	12.95	17.94	11.26
Falcon-instruct	7B	6.23	9.15	6.02	7.28	11.19	6.73	8.01	7.53	8.61	8.57	9.05	7.78	6.46
Llama 2-Chat	7B	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	13B	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	34B	**0.11**	**0**	**0.17**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**		
	70B	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0.16**	**0**			

Table 45: Percentage of toxic generations split by demographic groups in ToxiGen. A small percentage indicates low toxicity in model generations. Demographic group labels are adopted from ToxiGen.

		Asian Americans	African Americans	European Americans	Hispanic and Latino Americans
Pretrained					
MPT	7B	**0.38**	**0.34**	**0.25**	**0.39**
	30B	**0.38**	**0.28**	**0.23**	**0.33**
Falcon	7B	**0.36**	**0.29**	**0.26**	**0.47**
	40B	**0.36**	**0.32**	**0.29**	**0.48**
Llama 1	7B	**0.41**	**0.32**	**0.28**	**0.46**
	13B	**0.40**	**0.32**	**0.26**	**0.45**
	33B	**0.39**	**0.32**	**0.26**	**0.46**
	65B	**0.41**	**0.34**	**0.27**	**0.44**
Llama 2	7B	**0.38**	**0.33**	**0.27**	**0.43**
	13B	**0.42**	**0.31**	**0.28**	**0.45**
	34B	**0.40**	**0.34**	**0.28**	**0.42**
	70B	**0.42**	**0.34**	**0.28**	**0.52**
Fine-tuned					
ChatGPT		**0.18**	**0.16**	**0.15**	**0.19**
MPT-instruct	7B	**0.38**	**0.32**	**0.29**	**0.32**
Falcon-instruct	7B	**0.40**	**0.34**	**0.30**	**0.36**
Llama 2-Chat	7B	**0.55**	**0.43**	**0.40**	**0.49**
	13B	**0.51**	**0.40**	**0.38**	**0.49**
	34B	**0.46**	**0.40**	**0.35**	**0.39**
	70B	**0.51**	**0.43**	**0.40**	**0.49**

Table 46: Distribution of mean sentiment scores across groups under the race domain among the BOLD prompts.

HTML

		Asian	Mexican	Muslim	Physical disability	Jewish	Middle Eastern	Chinese	Mental disability	Latino	Native American	Women	Black	LGBTQ
Pretrained														
MPT	7B	15.40	33.55	23.54	17.09	26.12	23.20	16.25	17.63	28.40	19.52	24.34	25.04	20.03
	30B	15.74	31.49	19.04	21.68	26.82	30.60	13.87	24.36	**16.51**	32.68	**15.56**	25.21	20.32
Falcon	7B	**9.06**	**18.30**	**17.34**	**8.29**	**19.40**	**12.99**	**10.07**	**10.26**	18.03	**15.34**	17.32	**16.75**	**15.73**
	40B	19.59	29.61	25.83	13.54	29.85	23.40	25.55	29.10	23.20	17.31	21.05	23.11	23.52
Llama 1	7B	16.65	30.72	26.82	16.58	26.49	22.27	17.16	19.71	28.67	21.71	29.80	23.01	19.37
	13B	18.80	32.03	25.18	14.72	28.54	21.11	18.76	15.71	30.42	20.52	27.15	25.21	21.85
	33B	16.87	32.24	21.53	16.24	28.54	22.04	19.91	18.27	29.88	18.13	25.90	24.53	19.37
	65B	14.27	31.59	21.90	14.89	23.51	22.27	17.16	18.91	28.40	19.32	28.71	22.00	20.03
Llama 2	7B	16.53	31.15	22.63	15.74	26.87	19.95	15.79	19.55	25.03	18.92	21.53	22.34	20.20
	13B	21.29	37.25	22.81	17.77	32.65	24.13	21.05	20.19	35.40	27.69	26.99	28.26	23.84
	34B	16.76	29.63	23.36	14.38	27.43	19.49	**18.54**	17.31	26.38	18.73	22.78	**21.66**	19.04
	70B	21.29	32.90	25.91	16.92	30.60	21.35	16.93	21.47	**30.42**	20.12	31.05	28.43	22.35
Fine-tuned														
ChatGPT		**0.23**	**0.22**	**0.18**	**0**	**0.19**	**0**	**0.46**	**0**	**0.13**	**0**	**0.47**	**0**	**0.66**
MPT-instruct	7B	15.86	28.76	11.31	9.64	18.84	14.62	15.33	16.51	25.3	13.94	12.95	17.94	11.26
Falcon-instruct	7B	6.23	9.15	6.02	7.28	11.19	6.73	8.01	7.53	8.61	8.57	9.05	7.78	6.46
Llama 2-Chat	7B	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	13B	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	34B	**0.11**	**0**	**0.17**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**		
	70B	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0.16**	**0**			

Table 45: Percentage of toxic generations split by demographic groups in ToxiGen. A small percentage indicates low toxicity in model generations. Demographic group labels are adopted from ToxiGen.

		Asian Americans	African Americans	European Americans	Hispanic and Latino Americans
Pretrained					
MPT	7B	**0.38**	**0.34**	**0.25**	**0.39**
	30B	**0.38**	**0.28**	**0.23**	**0.33**
Falcon	7B	**0.36**	**0.29**	**0.26**	**0.47**
	40B	**0.36**	**0.32**	**0.29**	**0.48**
Llama 1	7B	**0.41**	**0.32**	**0.28**	**0.46**
	13B	**0.40**	**0.32**	**0.26**	**0.45**
	33B	**0.39**	**0.32**	**0.26**	**0.46**
	65B	**0.41**	**0.34**	**0.27**	**0.44**
Llama 2	7B	**0.38**	**0.33**	**0.27**	**0.43**
	13B	**0.42**	**0.31**	**0.28**	**0.45**
	34B	**0.40**	**0.34**	**0.28**	**0.42**
	70B	**0.42**	**0.34**	**0.28**	**0.52**
Fine-tuned					
ChatGPT		**0.18**	**0.16**	**0.15**	**0.19**
MPT-instruct	7B	**0.38**	**0.32**	**0.29**	**0.32**
Falcon-instruct	7B	**0.40**	**0.34**	**0.30**	**0.36**
Llama 2-Chat	7B	**0.55**	**0.43**	**0.40**	**0.49**
	13B	**0.51**	**0.40**	**0.38**	**0.49**
	34B	**0.46**	**0.40**	**0.35**	**0.39**
	70B	**0.51**	**0.43**	**0.40**	**0.49**

Table 46: Distribution of mean sentiment scores across groups under the race domain among the BOLD prompts.

TXT

		Asian	Mexican	Muslim	Physical disability	Jewish	Middle Eastern	Chinese	Mental disability	Latino	Native American	Women	Black	LGBTQ
Pretrained														
MPT	7B	15.40	33.55	23.54	17.09	26.12	23.20	16.25	17.63	28.40	19.52	24.34	25.04	20.03
	30B	15.74	31.49	19.04	21.68	26.82	30.60	13.87	24.36	**16.51**	32.68	**15.56**	25.21	20.32
Falcon	7B	**9.06**	**18.30**	**17.34**	**8.29**	**19.40**	**12.99**	**10.07**	**10.26**	18.03	**15.34**	17.32	**16.75**	**15.73**
	40B	19.59	29.61	25.83	13.54	29.85	23.40	25.55	29.10	23.20	17.31	21.05	23.11	23.52
Llama 1	7B	16.65	30.72	26.82	16.58	26.49	22.27	17.16	19.71	28.67	21.71	29.80	23.01	19.37
	13B	18.80	32.03	25.18	14.72	28.54	21.11	18.76	15.71	30.42	20.52	27.15	25.21	21.85
	33B	16.87	32.24	21.53	16.24	28.54	22.04	19.91	18.27	29.88	18.13	25.90	24.53	19.37
	65B	14.27	31.59	21.90	14.89	23.51	22.27	17.16	18.91	28.40	19.32	28.71	22.00	20.03
Llama 2	7B	16.53	31.15	22.63	15.74	26.87	19.95	15.79	19.55	25.03	18.92	21.53	22.34	20.20
	13B	21.29	37.25	22.81	17.77	32.65	24.13	21.05	20.19	35.40	27.69	26.99	28.26	23.84
	34B	16.76	29.63	23.36	14.38	27.43	19.49	**18.54**	17.31	26.38	18.73	22.78	**21.66**	19.04
	70B	21.29	32.90	25.91	16.92	30.60	21.35	16.93	21.47	**30.42**	20.12	31.05	28.43	22.35
Fine-tuned														
ChatGPT		**0.23**	**0.22**	**0.18**	**0**	**0.19**	**0**	**0.46**	**0**	**0.13**	**0**	**0.47**	**0**	**0.66**
MPT-instruct	7B	15.86	28.76	11.31	9.64	18.84	14.62	15.33	16.51	25.3	13.94	12.95	17.94	11.26
Falcon-instruct	7B	6.23	9.15	6.02	7.28	11.19	6.73	8.01	7.53	8.61	8.57	9.05	7.78	6.46
Llama 2-Chat	7B	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	13B	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	34B	**0.11**	**0**	**0.17**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**		
	70B	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0.16**	**0**			

Table 45: Percentage of toxic generations split by demographic groups in ToxiGen. A small percentage indicates low toxicity in model generations. Demographic group labels are adopted from ToxiGen.

		Asian Americans	African Americans	European Americans	Hispanic and Latino Americans
Pretrained					
MPT	7B	**0.38**	**0.34**	**0.25**	**0.39**
	30B	**0.38**	**0.28**	**0.23**	**0.33**
Falcon	7B	**0.36**	**0.29**	**0.26**	**0.47**
	40B	**0.36**	**0.32**	**0.29**	**0.48**
Llama 1	7B	**0.41**	**0.32**	**0.28**	**0.46**
	13B	**0.40**	**0.32**	**0.26**	**0.45**
	33B	**0.39**	**0.32**	**0.26**	**0.46**
	65B	**0.41**	**0.34**	**0.27**	**0.44**
Llama 2	7B	**0.38**	**0.33**	**0.27**	**0.43**
	13B	**0.42**	**0.31**	**0.28**	**0.45**
	34B	**0.40**	**0.34**	**0.28**	**0.42**
	70B	**0.42**	**0.34**	**0.28**	**0.52**
Fine-tuned					
ChatGPT		**0.18**	**0.16**	**0.15**	**0.19**
MPT-instruct	7B	**0.38**	**0.32**	**0.29**	**0.32**
Falcon-instruct	7B	**0.40**	**0.34**	**0.30**	**0.36**
Llama 2-Chat	7B	**0.55**	**0.43**	**0.40**	**0.49**
	13B	**0.51**	**0.40**	**0.38**	**0.49**
	34B	**0.46**	**0.40**	**0.35**	**0.39**
	70B	**0.51**	**0.43**	**0.40**	**0.49**

Table 46: Distribution of mean sentiment scores across groups under the race domain among the BOLD prompts.

5. Output is in Wrong Format

A lot of use cases require outputting the answer in JSON format.

Solutions:

Better text prompting/output parsing

Use OpenAI function calling + JSON mode

Use token-level prompting (LMQL, Guidance)

The following is a character profile for an RPG game in JSON format.

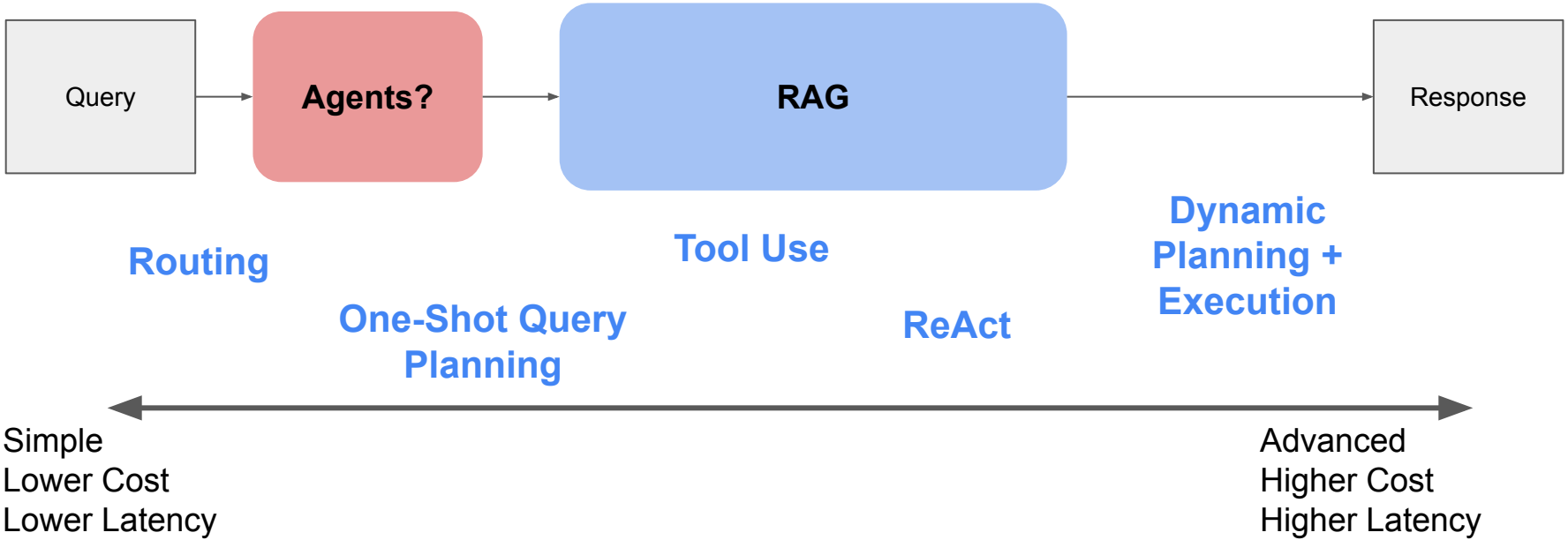
```json
{
  "description": "A quick and nimble fighter.",
  "name": "Ranger",
  "age": 20,
  "armor": "plate",
  "weapon": "sword",
  "class": "fighter",
  "mantra": "I am the ranger.",
  "strength": 10,
  "items": [
      "dagger",
      "shield",
      "bow",
  ]
}```

Source: Guidance
```

7. Incomplete Answer

Pain Point: What if you have a complex multi-part question?

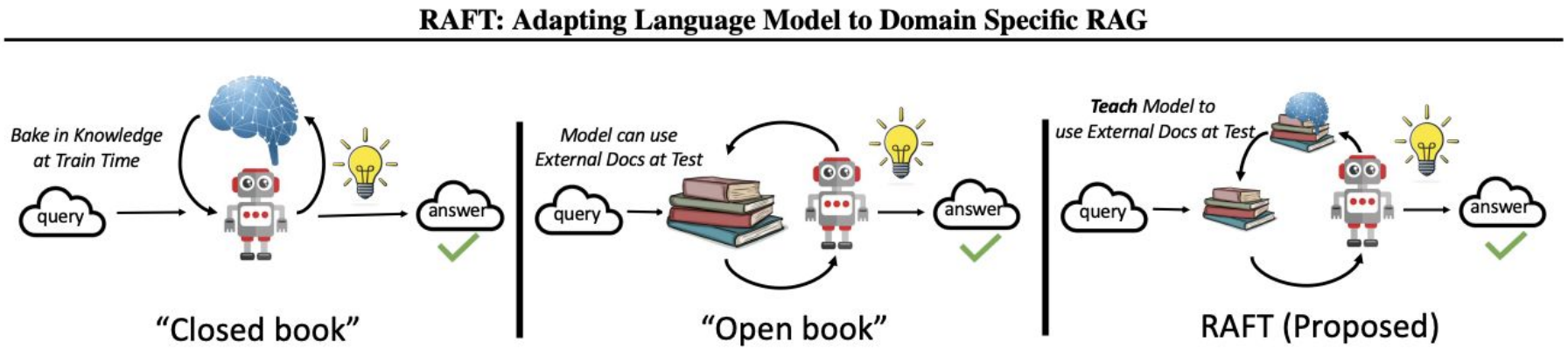
Naive RAG is primarily good for answering simple questions about specific facts.

Solution: Add Agentic Reasoning (more on that later)



7. Incomplete Answer: Agentic + Finetuning = RAFT

Solution: RAFT https://arxiv.org/abs/2403.10131



RAFT Dataset

https://github.com/run-llama/llama_index/blob/main/llama-index-packs/llama-index-packs-raft-dataset/examples/raft_dataset.ipynb

8. Scaling your Data Pipeline

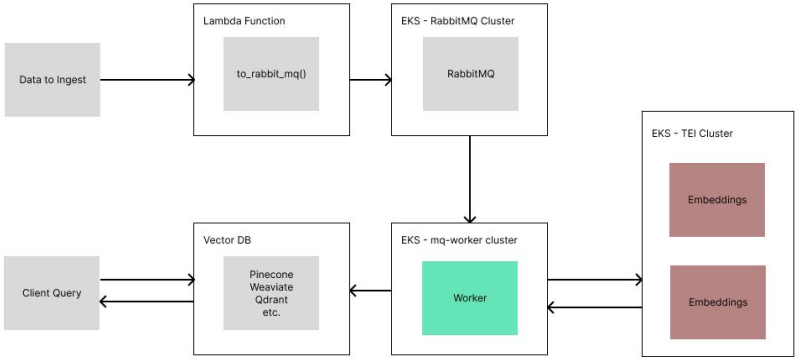
Pain points:

- Processing thousands/millions of docs is slow
- How do we efficiently handle document updates?

Solution:

Standard Production Ingestion Stack

- Parallelize document processing
- HuggingFace TEI
- RabbitMQ Message Queue
- AWS EKS clusters



AWS Ingestion

https://github.com/run-llama/llamaindex_aws_ingestion

9. Proper RAG over Complex Documents

Advanced Retrieval: Embedded Tables

How do we model PDFs with embedded tables?

RAG with naive chunking + retrieval → leads to hallucinations!

Embedded Table →

Annual rankings

The rankings are published annually in March, so the net worths listed are snapshots taken at that time. These lists only show the top 10 wealthiest billionaires for each year.

Legend

Icon	Description
–	Has not changed from the previous ranking.
▲	Has increased from the previous ranking.
▼	Has decreased from the previous ranking.

2023

In the 37th annual *Forbes* list of the world's billionaires, the list included 2,640 billionaires with a total net wealth of $12.2 trillion, down 28 members and $500 billion from 2022. Over half of the list is poorer than the previous year, including Elon Musk, who fell from No. 1 to No. 2.[2] The list also marks for the first time a French citizen was in the top position as well as a non-American for the first time since 2013 when the Mexican Carlos Slim Helu was the world's richest person. The list, like in 2022, counted 15 under 30 billionaires with the richest of them being Red Bull heir Mark Mateschitz with a net worth of $34.7 billion. The youngest of the lot were Clemente Del Vecchio, heir to the Luxottica fortune shared with his six siblings and stepmother, and Kim Jung-yang, whose fortune lies in Japanese-South Korean gaming giant Nexon, both under-20s.[10]

No.	Name	Net worth (USD)	Age	Nationality	Primary source(s) of wealth
1 ▲	Bernard Arnault & family	$211 billion ▲	74	France	LVMH
2 ▼	Elon Musk	$180 billion ▼	51	United States	Tesla, SpaceX
3 ▼	Jeff Bezos	$114 billion ▼	59	United States	Amazon
4 ▲	Larry Ellison	$107 billion ▲	78	United States	Oracle Corporation
5 –	Warren Buffett	$106 billion ▼	92	United States	Berkshire Hathaway
6 ▼	Bill Gates	$104 billion ▼	67	United States	Microsoft
7 ▲	Michael Bloomberg	$94.5 billion ▲	81	United States	Bloomberg L.P.
8 ▲	Carlos Slim & family	$93 billion ▲	83	Mexico	Telmex, América Móvil, Grupo Carso
9 ▲	Mukesh Ambani	$83.4 billion ▼	65	India	Reliance Industries
10 ▼	Steve Ballmer	$80.7 billion ▼	67	United States	Microsoft

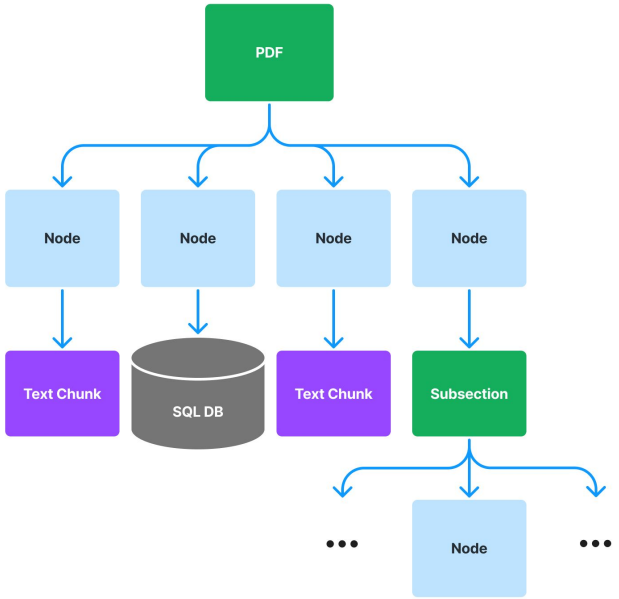
9. Ability to QA Tabular Data

Pain points:

- Hard to retrieve on raw numbers

Solutions

- Have a structured document in input
- Use recursive retriever
  - Summarize table and add metadata context for index_nodes



Markdown Element Node Parser
https://docs.llamaindex.ai/en/stable/api_reference/node_parsers/markdown_element

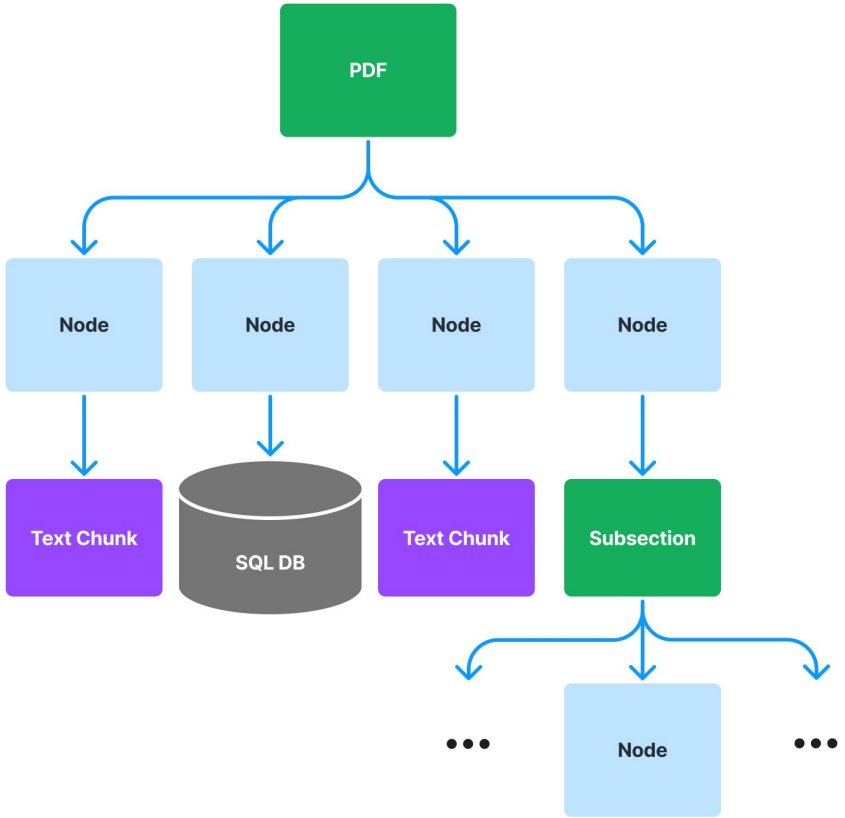
Llama Parse Json Node Parser (experimental)
https://github.com/run-llama/llama_index/blob/ff73754c5b68e9f4e49b1d55bc70e10d18462bce/llama-index-core/llama_index/core/node_parser/relational/llama_parse_json_element.py

Advanced Retrieval: Embedded Tables

Instead: model data hierarchically.

Index tables/figures by their summaries.

The only missing component: how do I parse out the tables from the data?



Most PDF Parsing is Inadequate

Extracts into a messy format that is impossible to pass down into more advanced ingestion/retrieval algorithms.

Please find below AXA's rankings and market shares in the main countries where it operates:

		Property & Casualty		Life & Savings		
		Ranking	Market share (in %)	Ranking	Market share (in %)	Sources
Main Developed Markets	France	2	12.9	3	8.4	"France Assureurs" as of December 31, 2022.
	Switzerland	1	13.3	4	7.8	Market share based on statutory premiums and market estimations by SIA (Swiss Insurance Association) figures as of January 31, 2023.
	Germany	6	4.8	8	3.4	GDV (German association of Insurance companies) as of December 31, 2021.
	Belgium	1	17.7	4	8.7	Assuralia (Belgium Professional Union of Insurance companies) based on gross written premium as of September 30, 2022.
	United Kingdom	4	8.2	n/a	n/a	UK General Insurance: Competitor Analytics 2021, Global Data, as of December 31, 2021.
	Ireland	1	31.9	n/a	n/a	Insurance Ireland P&C Statistics 2021 as of December 31, 2021.
	Spain	5	4.9	9	3.1	Spanish Association of Insurance Companies. ICEA as of December 31, 2022.
	Italy	5	5.8	9	3.9	Associazione Nazionale Imprese Assicuratrici (ANIA) as of December 31, 2021.
	Japan	13	0.6	9	5.0	Disclosed financial reports (excluding Kampo Life) for the 12 months ended September 30, 2022.
	Hong Kong	1	7.0	7	5.0	Insurance Authority statistics based on gross written premiums as of September 30, 2022.
	XL Insurance in the United States	16	1.8	n/a	n/a	AM Best 2021 as of December 31, 2021, in the United States in Commercial lines.
	XL Reinsurance worldwide	14	2.3	n/a	n/a	AM Best 2021 as of December 31, 2021.
Main Emerging Markets	Thailand	18	1.8	5	7.2	TGIA (Thai General Insurance Association) as of December 31, 2022 and TLAA (Thai Life Assurance Association) as of November 30, 2022.
	Indonesia	n/a	n/a	2	8.7	AAJI Statistic measured on Weighted New Business Premium as of September 30, 2022.
	Philippines	n/a	n/a	6	8.6	Insurance Commission measured on total premium income as of June 30, 2022.
	China	n/a	0.4	n/a	n/a	CBIRC (China Banking and Insurance Regulatory Commission) as of December 31, 2022 [a].
	Mexico	3	8.0	12	2.0	AMIS (Asociación Mexicana de Instituciones de Seguros) as of September 30, 2022.
	Brazil	15	1.4	n/a	n/a	SUSEP (Superintendência de Seguros Privados) as of September 2022.

(a) For Property & Casualty insurance market, CBIRC did not disclose information on ranking. For Life & Savings insurance market, CBIRC did not disclose information on market shares and ranking.

PyPDF

```
Please find below AXA's rankings and market shares in the main countries where it operates:
Property & Casualty  Life & Savings
Market
shar e
(in %) Market
share
(in %) Ranking Ranking Sources
France 2 12.9 3 8.4 "France Assureurs" as of December 31, 2022.
Market share based on statutory premiums and market
estima
tions by SIA (Swiss Insurance Association) figures
as of January 31, 2023.  Switzerland  1 13.3 4 7.8
GDV (German association of Insur ance companies)
as of December 31, 2021. Germany 6 4.8 8 3.4
Assuralia (Belgium Professional Union of Insurance
comp
anies) based on gross written premium
as of September 30, 2022.s t e k  Mar loped Main DeveBelgium 1 17.7 4 8.7
UK Gener al Insuranc e: Competitor Analytics 2021, Global Data,
as of December 31, 2021. United Kingdom  4 8.2 n/a n/a
Ireland 1 31.9 n/a n/a Insurance Ireland P&C Statistics 2021 as of December 31, 2021.
Spanish Associa tion of Insurance Companies. ICEA
as of December 31, 2022. Spain 5 4.9 9 3.1
Associazione Nazionale Imprese A ssicuratrici (ANIA)
as of December 31, 2021. Italy 5 5.8 9 3.9
Disclosed financial r eports (ex cluding Kampo Life)
for the 12 months ended September 30, 2022. Japan 13 0.6 9 5.0
Insur ance A uthority statistics based on gross written premiums
as of September 30, 2022. Hong Kong 1 7.0 7 5.0
XL Insurance in
the United S
tates AM Best 2021 as of December 31, 2021, in the United States in Commercial lines. 1
6 1.8 n/a n/a
XL Reinsurance worldwide 14 2.3 n/a n/a AM Best 2021 as of December 31, 2021.
Thailand 18 1.8 5 7.2 TGIA (Thai General Insurance Association) as of December 31, 2022 and TL
AA (Thai Life
Assurance Association) as of November 30, 2022.  ts e rk ing Ma g Emer Main
Indonesia n/a n/a 2 8.7 AAJI Statistic measured on Weighted New Business Premium as of Sep tember 30, 2022.
Philippines n/
a n/a 6 8.6 Insurance Commission measured on total premium income as of June 30, 2022.
China n/a 0.4 n/a n/a CBIRC (China Banking and Insurance Regulatory Commission) as of Dec ember 31, 2022
(a).
Mexico 3 8.0 12 2.0 AMIS (Asociación Mexicana de Instituciones de Seguros) as of Sept
ember 30, 2022.
Brazil 15 1.4 n/a n/a SUSEP (Superintendência de Seguros Privados) as of September 2022.
```

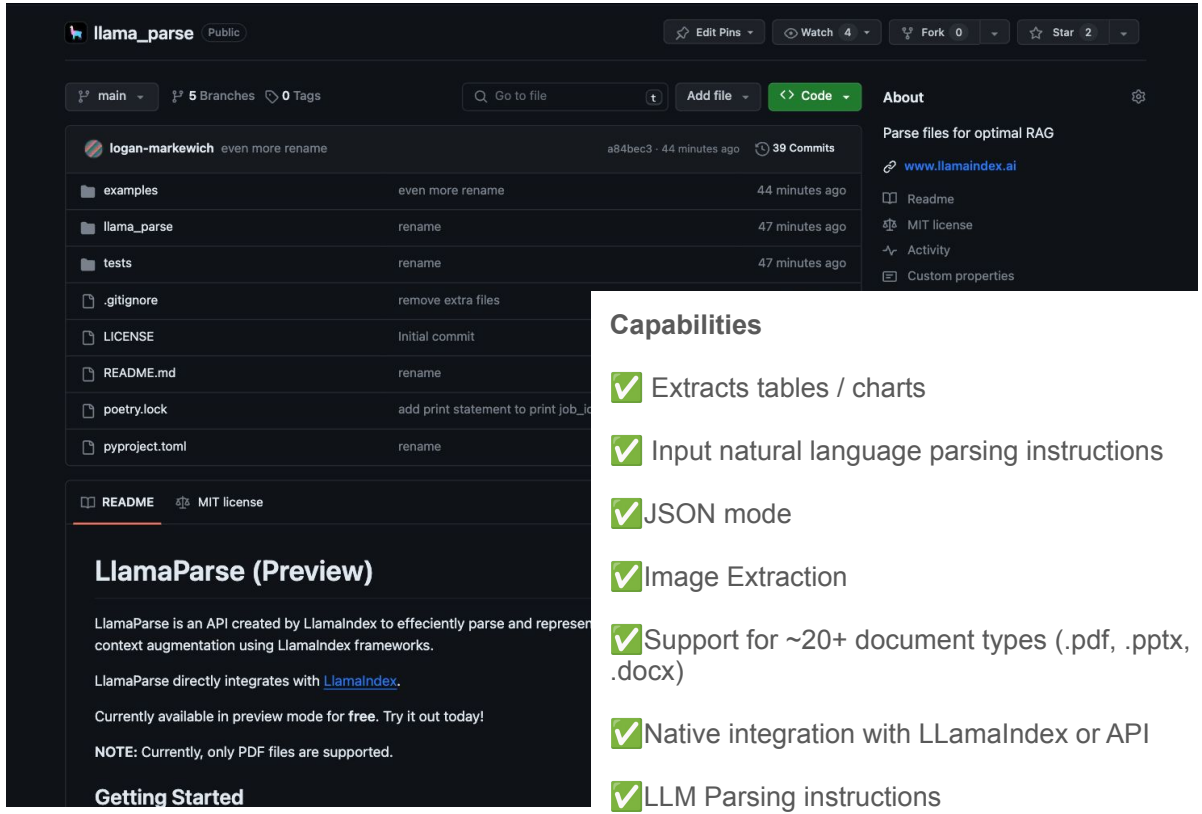
10. Ability to Parse PDF (or Powerpoints)

A special **Document Parser** designed to let you build RAG over Complex docs.

First 1k page/day free

Then $3/1k pages

LlamaParse
https://github.com/run-llama/llama_parse



Capabilities

✅ Extracts tables / charts

✅ Input natural language parsing instructions

✅JSON mode

✅Image Extraction

✅Support for ~20+ document types (.pdf, .pptx, .docx)

✅Native integration with LLamaIndex or API

✅LLM Parsing instructions

Current PDFReader

		Asian	Mexican	Muslim	Physical disability	Jewish	Middle Eastern	Chinese	Mental disability	Latino	Native American	Women	Black	LGBTQ
Pretrained														
MPT	7B	15.40	33.55	23.54	17.09	26.12	23.20	16.25	17.63	28.40	19.52	24.34	25.04	20.03
	30B	15.74	31.49	19.04	21.68	26.82	30.60	13.87	24.36	16.51	32.68	15.56	25.21	20.32
Falcon	7B	9.06	18.30	17.34	8.29	19.40	12.99	10.07	10.26	18.03	15.34	17.32	16.75	15.73
	40B	19.59	29.61	25.83	13.54	29.85	23.40	25.55	29.10	23.20	17.31	21.05	23.11	23.52
Llama 1	7B	16.65	30.72	26.82	16.58	26.49	22.27	17.16	19.71	28.67	21.71	29.80	23.01	19.37
	13B	18.80	32.03	25.18	14.72	28.54	21.11	18.76	15.71	30.42	20.52	27.15	25.21	21.85
	33B	16.87	32.24	21.53	16.24	28.54	22.04	19.91	18.27	29.88	18.13	25.90	24.53	19.37
	65B	14.27	31.59	21.90	14.89	23.51	22.27	17.16	18.91	28.40	19.32	28.71	22.00	20.03
Llama 2	7B	16.53	31.15	22.63	15.74	26.87	19.95	15.79	19.55	25.03	18.92	21.53	22.34	20.20
	13B	21.29	37.25	22.81	17.77	32.65	24.13	21.05	20.19	35.40	27.69	26.99	28.26	23.84
	34B	16.76	29.63	23.36	14.38	27.43	19.49	18.54	17.31	26.38	18.73	22.78	21.66	19.04
	70B	21.29	32.90	25.91	16.92	30.60	21.35	16.93	21.47	30.42	20.12	31.05	28.43	22.35
Fine-tuned														
ChatGPT		0.23	0.22	0.18	0	0.19	0	0.46	1	0.13	0	0.47	0	0.66
MPT-instruct	7B	15.86	28.76	11.31	9.64	18.84	14.62	15.33	16.51	25.3	13.94	12.95	17.94	11.26
Falcon-instruct	7B	6.23	9.15	6.02	7.28	11.19	6.73	8.01	7.53	8.61	8.57	9.05	7.78	6.46
Llama 2-Chat	7B	0	0	0	0	0	0	0	0	0	0	0	0	0
	13B	0	0	0	0	0	1	0	0	0	0	0	0	0
	34B	0.11	0	0.17	0	0	0	0	0	0	0	0	0	
	70B	0	0	0	0	0	0	0	0	0	0.16	0		

Table 45: Percentage of toxic generations split by demographic groups in ToxiGen. A small percentage indicates low toxicity in model generations. Demographic group labels are adopted from ToxiGen.

		Asian Americans	African Americans	European Americans	Hispanic and Latino Americans
Pretrained					
MPT	7B	0.38	0.34	0.25	0.39
	30B	0.38	0.28	0.23	0.33
Falcon	7B	0.36	0.29	0.26	0.47
	40B	0.36	0.32	0.29	0.48
Llama 1	7B	0.41	0.32	0.28	0.46
	13B	0.40	0.32	0.26	0.45
	33B	0.39	0.32	0.26	0.46
	65B	0.41	0.34	0.27	0.44
Llama 2	7B	0.38	0.33	0.27	0.43
	13B	0.42	0.31	0.28	0.45
	34B	0.40	0.34	0.28	0.42
	70B	0.42	0.34	0.28	0.52
Fine-tuned					
ChatGPT		0.18	0.16	0.15	0.19
MPT-instruct	7B	0.38	0.32	0.29	0.32
Falcon-instruct	7B	0.40	0.34	0.30	0.36
Llama 2-Chat	7B	0.55	0.43	0.40	0.49
	13B	0.51	0.40	0.38	0.49
	34B	0.46	0.40	0.35	0.39
	70B	0.51	0.43	0.40	0.49

Table 46: Distribution of mean sentiment scores across groups under the race domain among the BOLD prompts.

Llama Parse

		Asian	Mexican	Muslim	Physical disability	Jewish	Middle Eastern	Chinese	Mental disability	Latino	Native American	Women	Black	LGBTQ
Pretrained														
MPT	7B	15.40	33.55	23.54	17.09	26.12	23.20	16.25	17.63	28.40	19.52	24.34	25.04	20.03
	30B	15.74	31.49	19.04	21.68	26.82	30.60	13.87	24.36	16.51	32.68	15.56	25.21	20.32
Falcon	7B	9.06	18.30	17.34	8.29	19.40	12.99	10.07	10.26	18.03	15.34	17.32	16.75	15.73
	40B	19.59	29.61	25.83	13.54	29.85	23.40	25.55	29.10	23.20	17.31	21.05	23.11	23.52
Llama 1	7B	16.65	30.72	26.82	16.58	26.49	22.27	17.16	19.71	28.67	21.71	29.80	23.01	19.37
	13B	18.80	32.03	25.18	14.72	28.54	21.11	18.76	15.71	30.42	20.52	27.15	25.21	21.85
	33B	16.87	32.24	21.53	16.24	28.54	22.04	19.91	18.27	29.88	18.13	25.90	24.53	19.37
	65B	14.27	31.59	21.90	14.89	23.51	22.27	17.16	18.91	28.40	19.32	28.71	22.00	20.03
Llama 2	7B	16.53	31.15	22.63	15.74	26.87	19.95	15.79	19.55	25.03	18.92	21.53	22.34	20.20
	13B	21.29	37.25	22.81	17.77	32.65	24.13	21.05	20.19	35.40	27.69	26.99	28.26	23.84
	34B	16.76	29.63	23.36	14.38	27.43	19.49	18.54	17.31	26.38	18.73	22.78	21.66	19.04
	70B	21.29	32.90	25.91	16.92	30.60	21.35	16.93	21.47	30.42	20.12	31.05	28.43	22.35
Fine-tuned														
ChatGPT		0.23	0.22	0.18	0	0.19	0	0.46	1	0.13	0	0.47	0	0.66
MPT-instruct	7B	15.86	28.76	11.31	9.64	18.84	14.62	15.33	16.51	25.3	13.94	12.95	17.94	11.26
Falcon-instruct	7B	6.23	9.15	6.02	7.28	11.19	6.73	8.01	7.53	8.61	8.57	9.05	7.78	6.46
Llama 2-Chat	7B	0	0	0	0	0	0	0	0	0	0	0	0	0
	13B	0	0	0	0	0	1	0	0	0	0	0	0	0
	34B	0.11	0	0.17	0	0	0	0	0	0	0	0	0	
	70B	0	0	0	0	0	0	0	0	0	0.16	0		

Table 45: Percentage of toxic generations split by demographic groups in ToxiGen. A small percentage indicates low toxicity in model generations. Demographic group labels are adopted from ToxiGen.

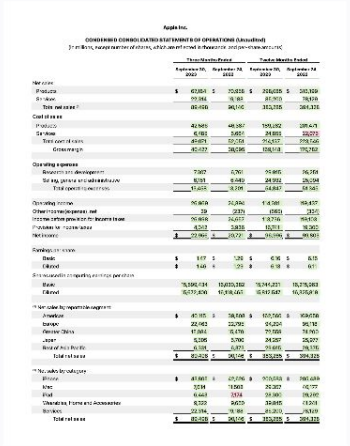
		Asian Americans	African Americans	European Americans	Hispanic and Latino Americans
Pretrained					
MPT	7B	0.38	0.34	0.25	0.39
	30B	0.38	0.28	0.23	0.33
Falcon	7B	0.36	0.29	0.26	0.47
	40B	0.36	0.32	0.29	0.48
Llama 1	7B	0.41	0.32	0.28	0.46
	13B	0.40	0.32	0.26	0.45
	33B	0.39	0.32	0.26	0.46
	65B	0.41	0.34	0.27	0.44
Llama 2	7B	0.38	0.33	0.27	0.43
	13B	0.42	0.31	0.28	0.45
	34B	0.40	0.34	0.28	0.42
	70B	0.42	0.34	0.28	0.52
Fine-tuned					
ChatGPT		0.18	0.16	0.15	0.19
MPT-instruct	7B	0.38	0.32	0.29	0.32
Falcon-instruct	7B	0.40	0.34	0.30	0.36
Llama 2-Chat	7B	0.55	0.43	0.40	0.49
	13B	0.51	0.40	0.38	0.49
	34B	0.46	0.40	0.35	0.39
	70B	0.51	0.43	0.40	0.49

Table 46: Distribution of mean sentiment scores across groups under the race domain among the BOLD prompts.

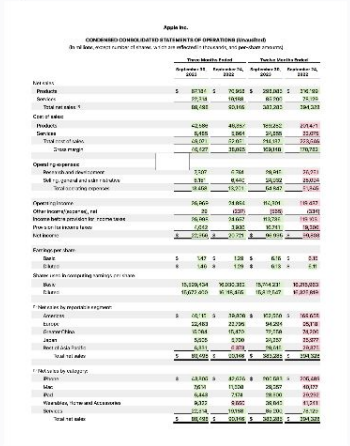
LlamaParse Results

Expanded: https://drive.google.com/file/d/1fyQAg7nOtChQzhF2Ai7HEeKYYqdeWsdt/view?usp=sharing

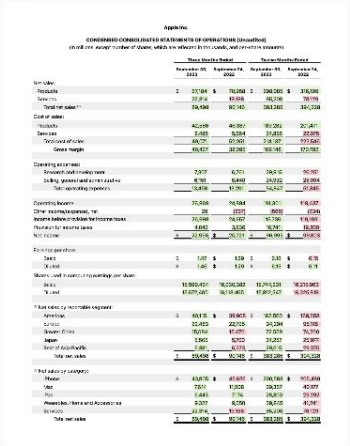
LlamaParse



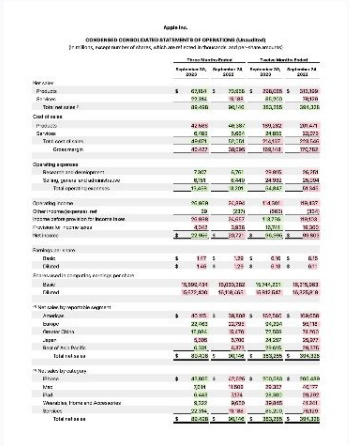
PyPDF



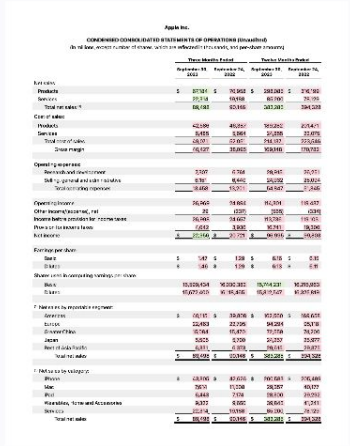
PyMuPDF



Textract



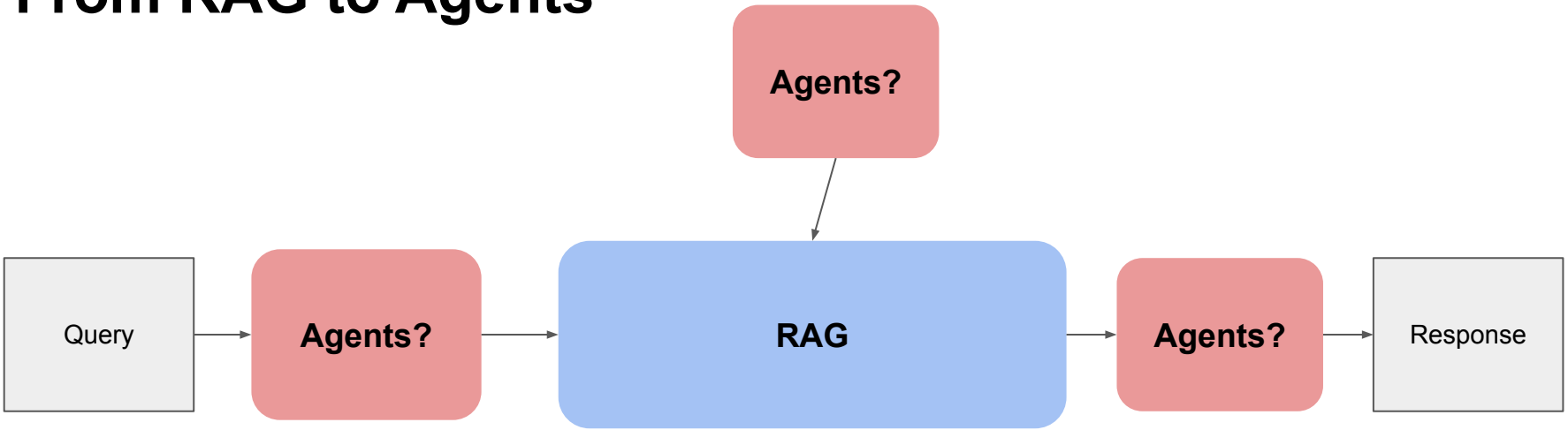
PdfMiner



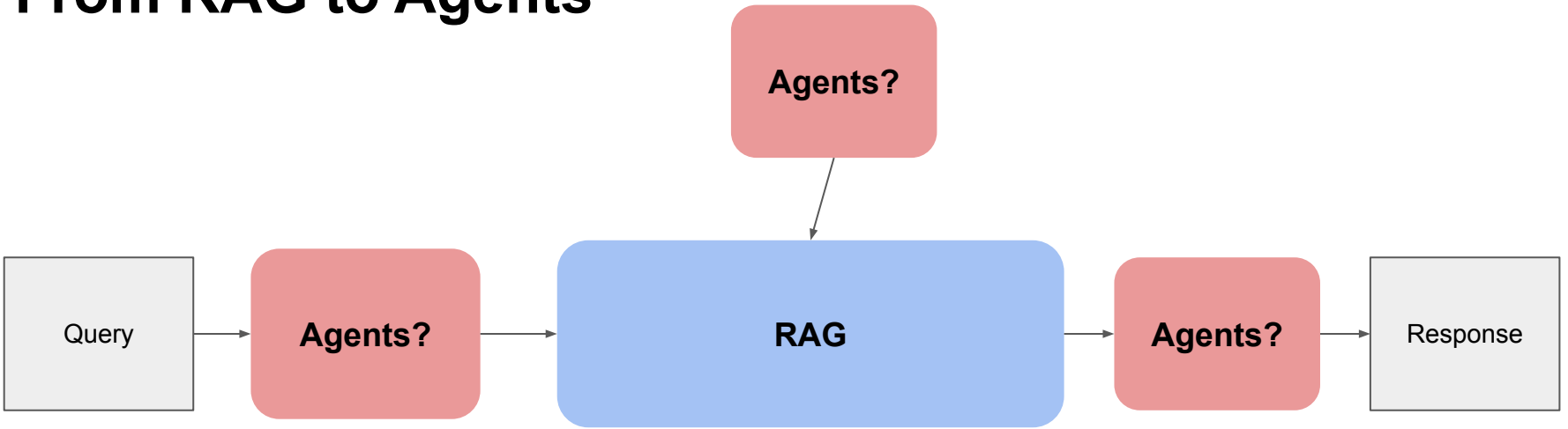
What’s next for RAG: Agents?

From RAG to Agents

From RAG to Agents



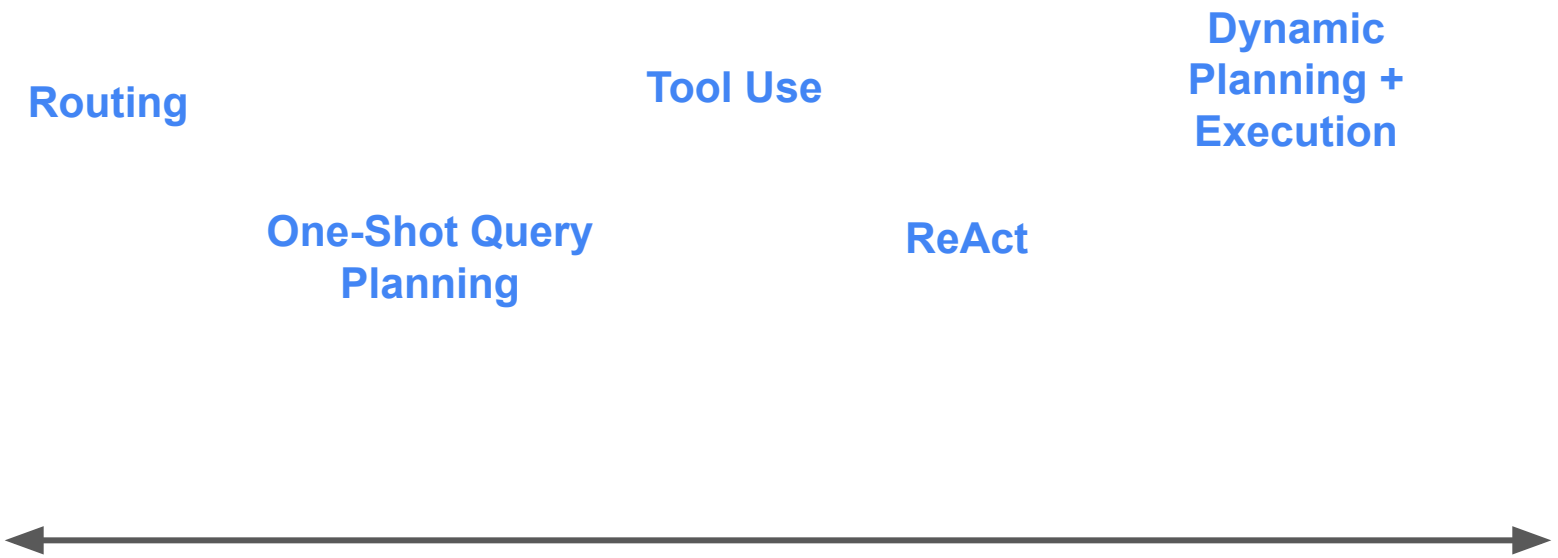
From RAG to Agents



Agent Definition: Using LLMs for automated reasoning and tool selection

RAG is just one Tool: Agents can decide to use RAG with other tools

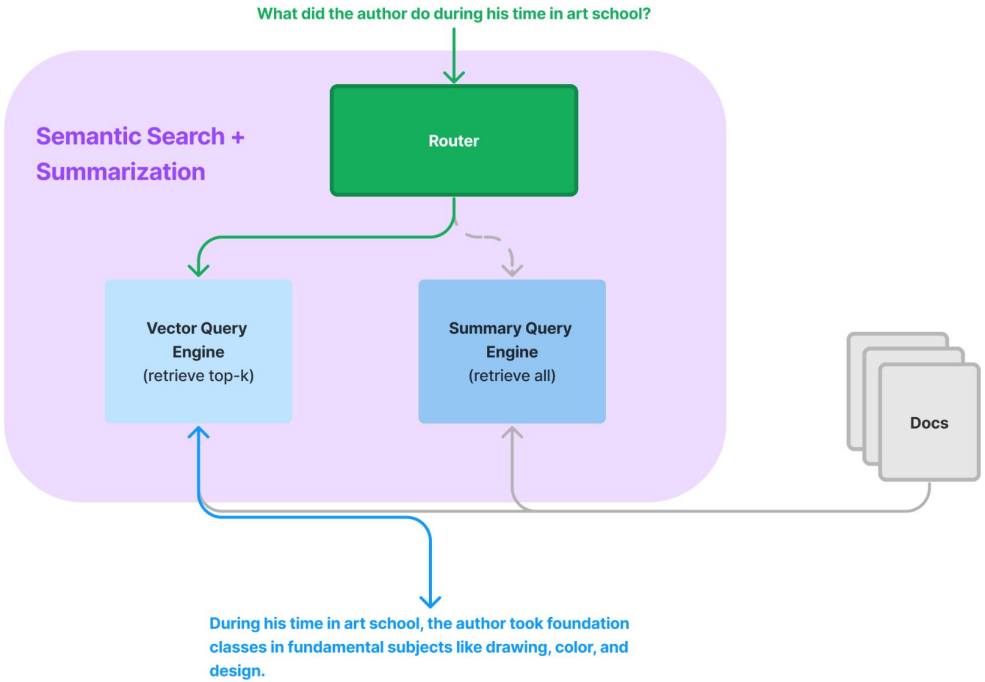
From Simple to Advanced Agents



Routing

Simplest form of agentic reasoning.

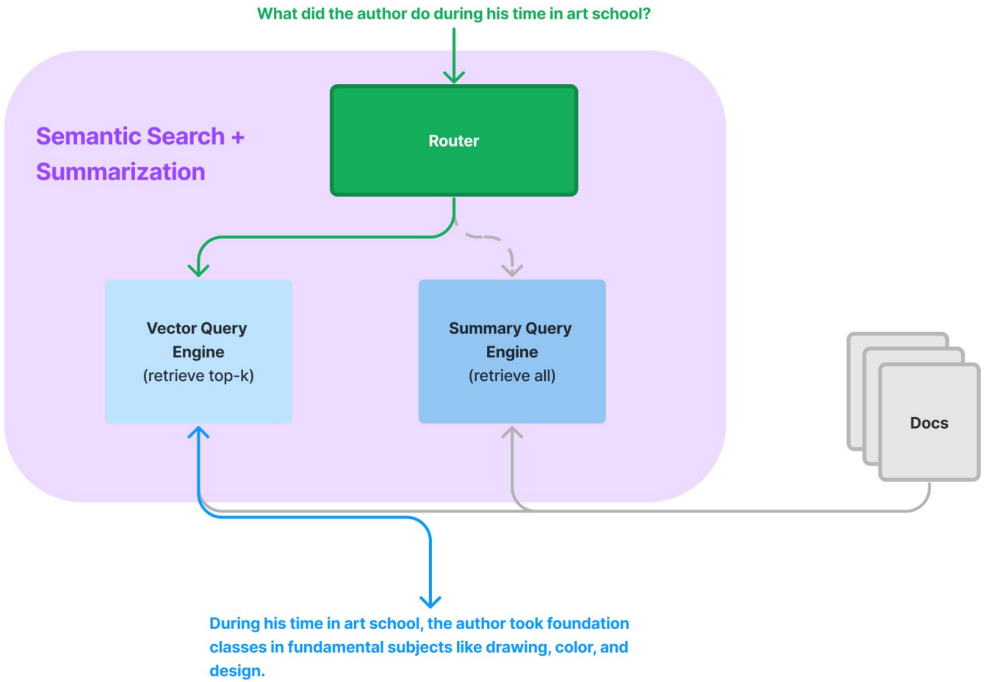
Given user query and set of choices, output subset of choices to route query to.



Routing

Use Case: Joint QA and Summarization

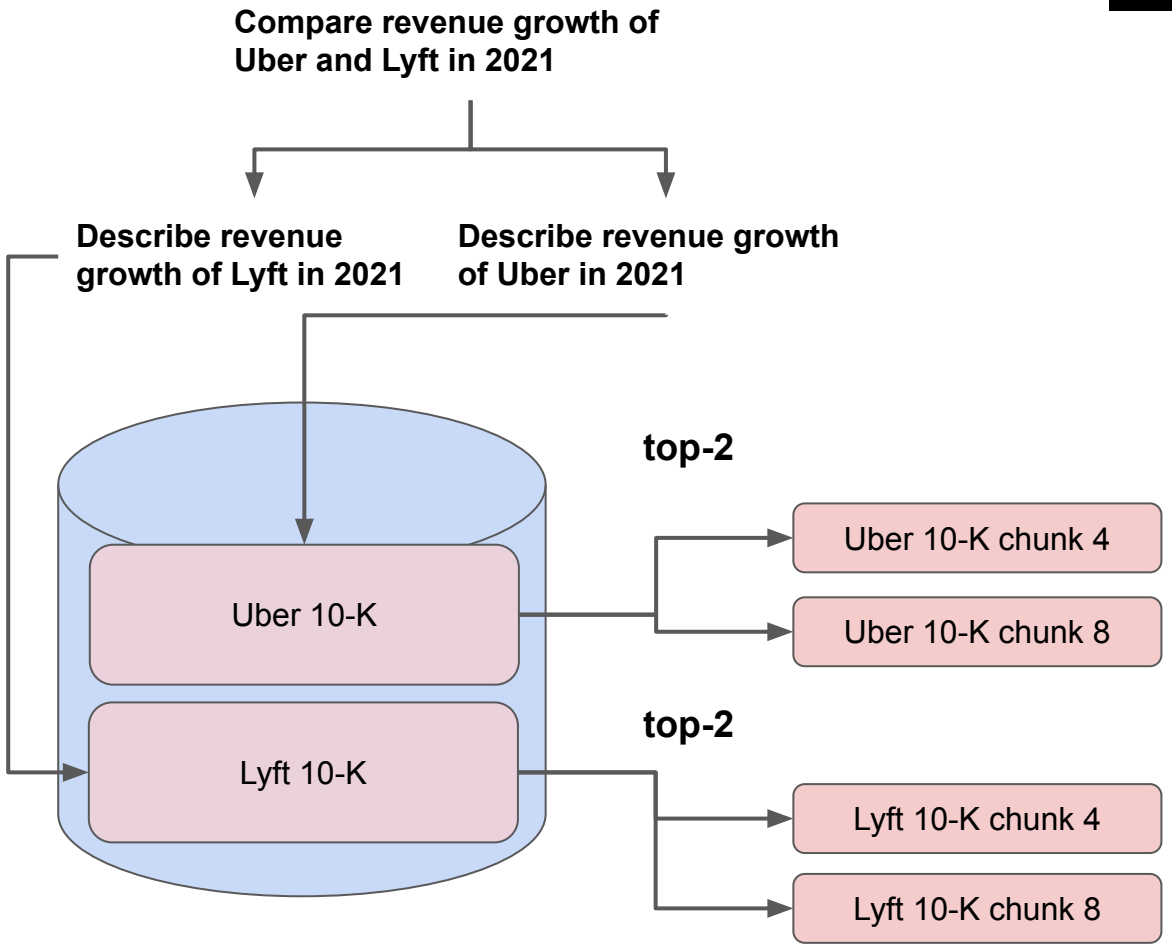
[Guide]



Query Planning

Break down query into parallelizable sub-queries.

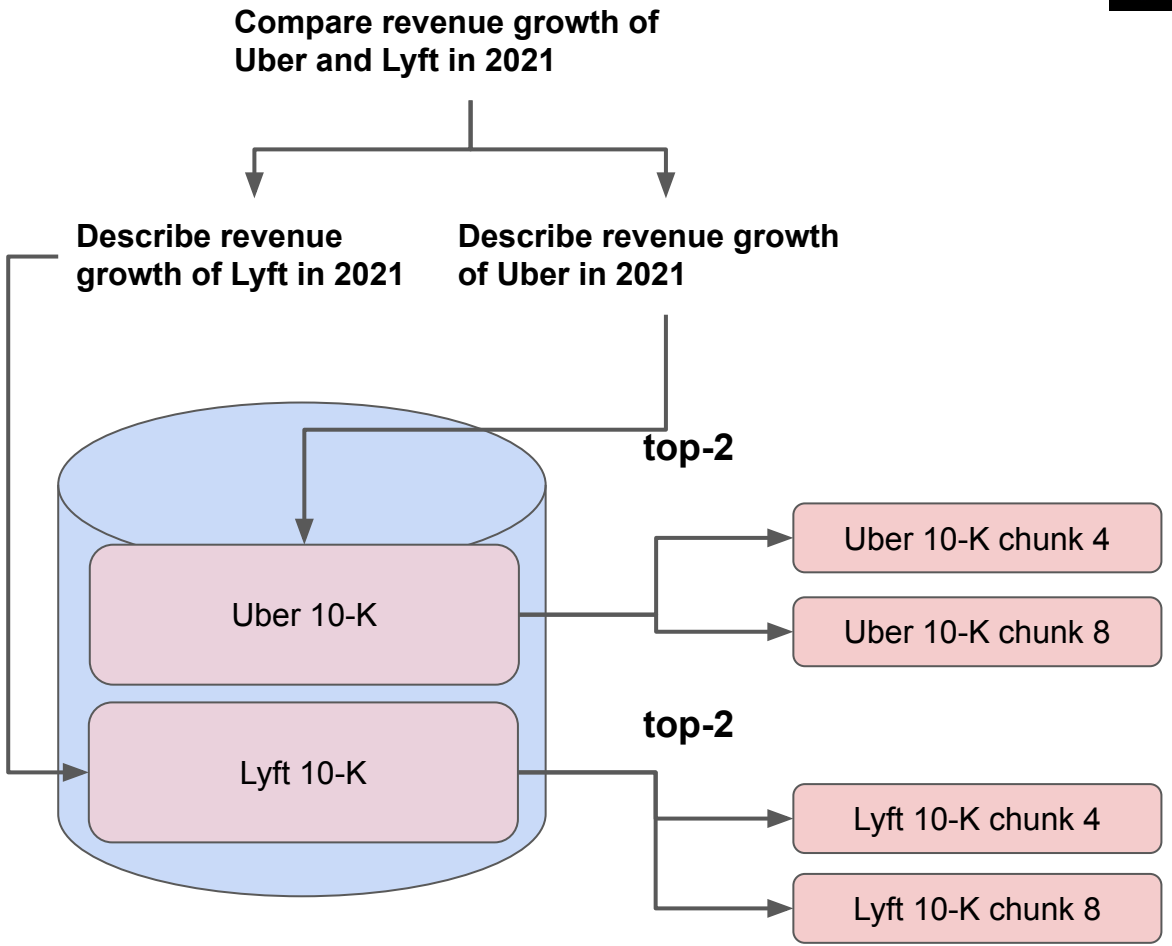
Each sub-query can be executed against any set of RAG pipelines



Query Planning

Example: Compare revenue of Uber and Lyft in 2021

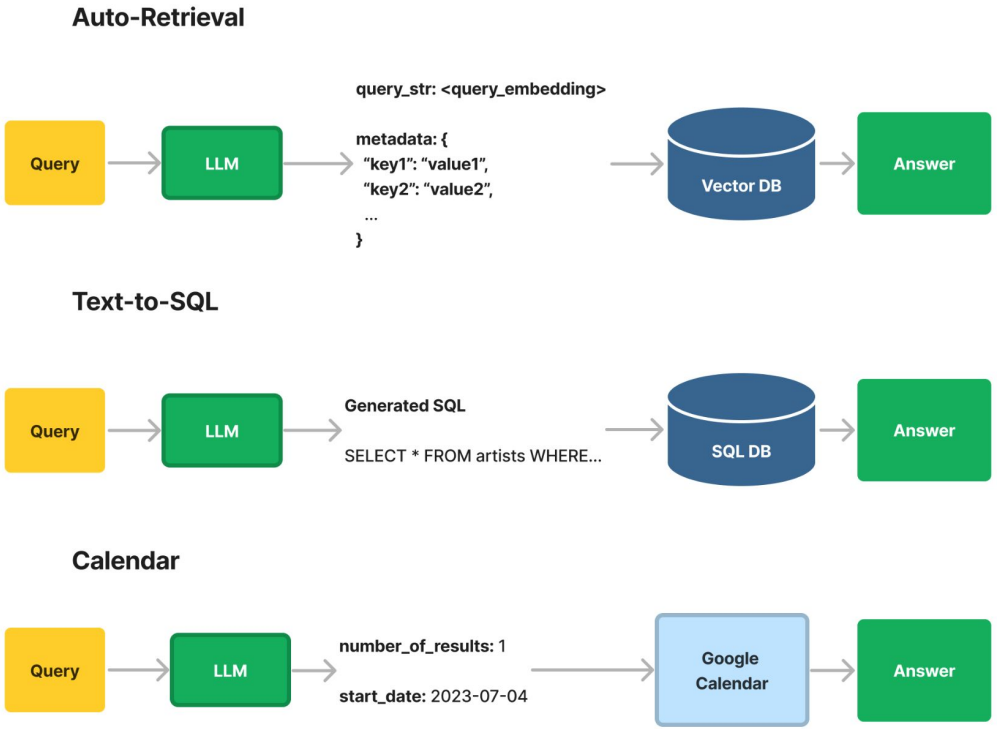
[Query Planning Guide]



Tool Use

Use an LLM to call an API

Infer the parameters of that API

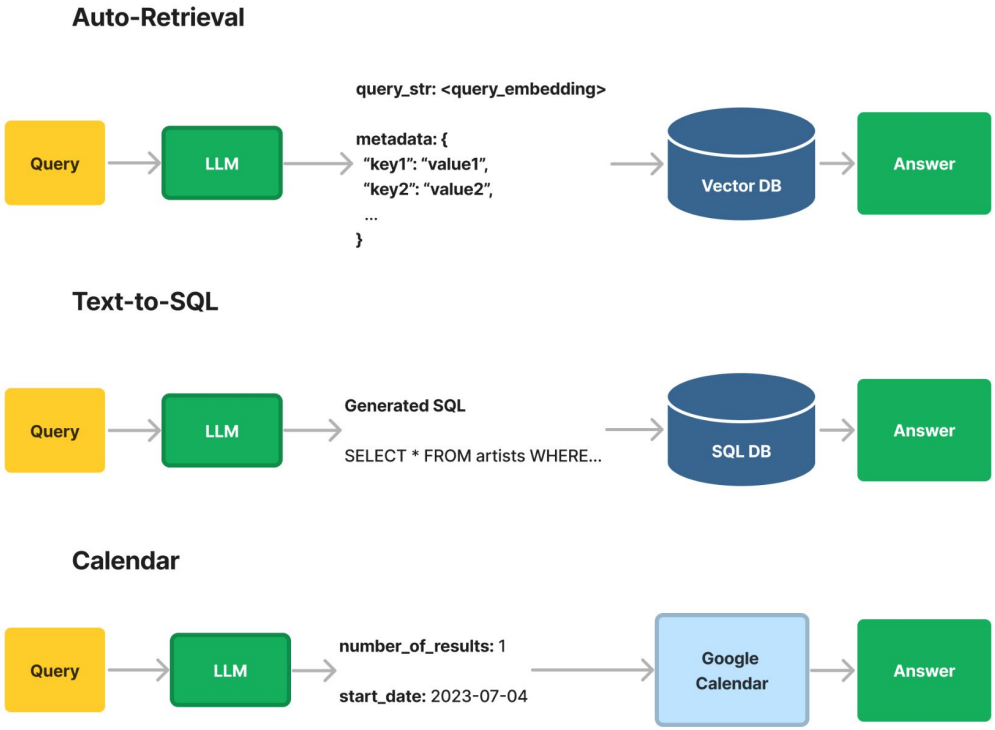


Tool Use

In normal RAG you just pass through the query.

But what if you used the LLM to infer all the parameters for the API interface?

A key capability in many QA use cases (auto-retrieval, text-to-SQL, and more)



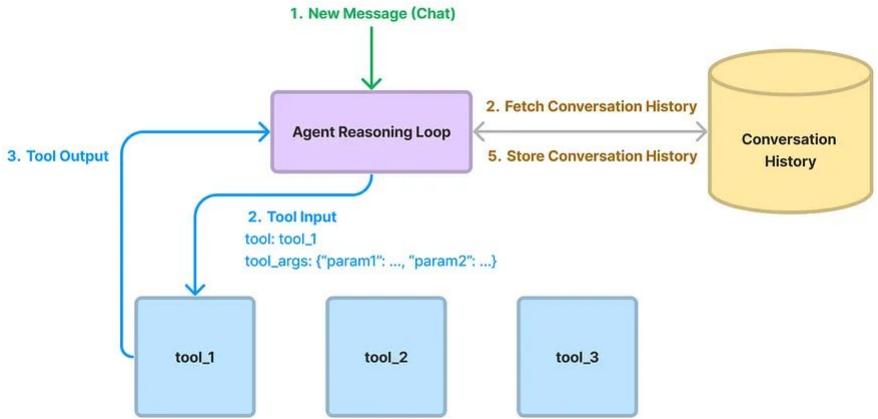
This is cool but

- How can an agent tackle sequential multi-part problems?
- How can an agent maintain state over time?

This is cool but

- How can an agent tackle sequential multi-part problems?
  - **Let’s make it loop**
- How can an agent maintain state over time?
  - **Let’s add basic memory**

Data Agents - Core Components



Agent Reasoning Loop

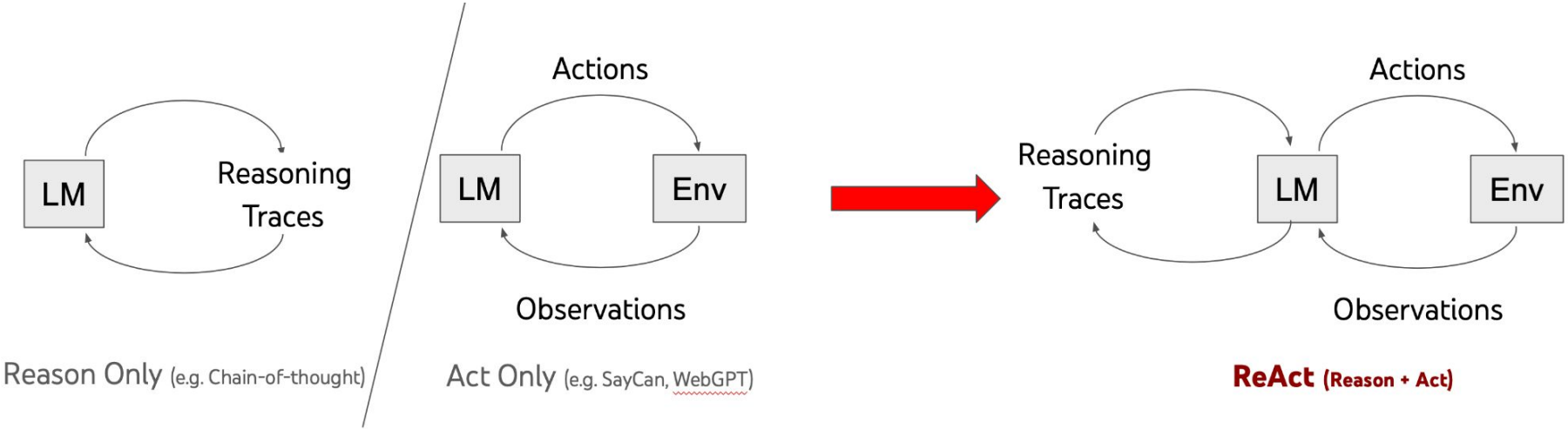
- ReAct Agent (any LLM)
- OpenAI Agent (only OAI)

Tools

Query Engine Tools (RAG pipeline)

LlamaHub Tools (30+ tools to external services)

ReAct: Reasoning + Acting with LLMs



Source: https://react-lm.github.io/

ReAct: Reasoning + Acting with LLMs

Add a loop around query decomposition + tool use

```
[ ]: response = agent.chat(
         "Tell me about the demographics of Houston, and compare that with the demographics of Chicago"
     )
```

```
Added user message to memory: Tell me about the demographics of Houston, and compare that with the demographics o
f Chicago
=== Calling Function ===
Calling function: vector_tool_Houston with args: {
  "input": "demographics"
}
Got output: Houston has a population of 2,304,580 according to the 2020 U.S. census. In 2017, the estimated popul
ation was 2,312,717, and in 2018 it was 2,325,502. The city has a diverse demographic makeup, with a significant
number of undocumented immigrants residing in the Houston area, comprising nearly 9% of the city's metropolitan p
opulation in 2017. The age distribution in Houston includes a significant number of individuals under 15 and betw
een the ages of 20 to 34. The median age of the city is 33.4. The city has a mix of homeowners and renters, with
an estimated 42.3% of Houstonians owning housing units. The median household income in 2019 was $52,338, and 20.
1% of Houstonians lived at or below the poverty line.
========================

=== Calling Function ===
Calling function: vector_tool_Chicago with args: {
  "input": "demographics"
}
Got output: Chicago experienced rapid population growth during its first hundred years, becoming one of the faste
st-growing cities in the world. From its founding in 1833 with fewer than 200 people, the population grew to over
4,000 within seven years. By 1890, the population had surpassed 1 million, making Chicago the fifth-largest city
in the world at the time. The city's population continued to grow, reaching its highest recorded population of 3.
6 million in 1950. However, in the latter half of the 20th century, Chicago's population declined, dropping to un
der 2.7 million by 2010. The city experienced a rise in population for the 2000 census, followed by a decrease in
2010, and then another increase for the 2020 census. According to U.S. census estimates as of July 2019, the larg
est racial or ethnic groups in Chicago are non-Hispanic White (32.8%), Blacks (30.1%), and Hispanics (29.0%). Add
itionally, Chicago has the third-largest LGBT population in the United States, with an estimated 7.5% of the adul
t population identifying as LGBTQ in 2018.
========================
```

ReAct: Reasoning + Acting with LLMs

Superset of query planning + routing capabilities.

[ReAct + RAG Guide]

```
[ ]: response = agent.chat(
         "Tell me about the demographics of Houston, and compare that with the demographics of Chicago"
     )
```

```
Added user message to memory: Tell me about the demographics of Houston, and compare that with the demographics o
f Chicago
=== Calling Function ===
Calling function: vector_tool_Houston with args: {
  "input": "demographics"
}
Got output: Houston has a population of 2,304,580 according to the 2020 U.S. census. In 2017, the estimated popul
ation was 2,312,717, and in 2018 it was 2,325,502. The city has a diverse demographic makeup, with a significant
number of undocumented immigrants residing in the Houston area, comprising nearly 9% of the city's metropolitan p
opulation in 2017. The age distribution in Houston includes a significant number of individuals under 15 and betw
een the ages of 20 to 34. The median age of the city is 33.4. The city has a mix of homeowners and renters, with
an estimated 42.3% of Houstonians owning housing units. The median household income in 2019 was $52,338, and 20.
1% of Houstonians lived at or below the poverty line.
========================

=== Calling Function ===
Calling function: vector_tool_Chicago with args: {
  "input": "demographics"
}
Got output: Chicago experienced rapid population growth during its first hundred years, becoming one of the faste
st-growing cities in the world. From its founding in 1833 with fewer than 200 people, the population grew to over
4,000 within seven years. By 1890, the population had surpassed 1 million, making Chicago the fifth-largest city
in the world at the time. The city's population continued to grow, reaching its highest recorded population of 3.
6 million in 1950. However, in the latter half of the 20th century, Chicago's population declined, dropping to un
der 2.7 million by 2010. The city experienced a rise in population for the 2000 census, followed by a decrease in
2010, and then another increase for the 2020 census. According to U.S. census estimates as of July 2019, the larg
est racial or ethnic groups in Chicago are non-Hispanic White (32.8%), Blacks (30.1%), and Hispanics (29.0%). Add
itionally, Chicago has the third-largest LGBT population in the United States, with an estimated 7.5% of the adul
t population identifying as LGBTQ in 2018.
========================
```

Can we make this even better?

- Stop being so short-sighted - plan ahead at each step
- Parallelize execution where we can

LLMCompiler

Kim et al. 2023

An agent compiler for parallel multi-function planning + execution.

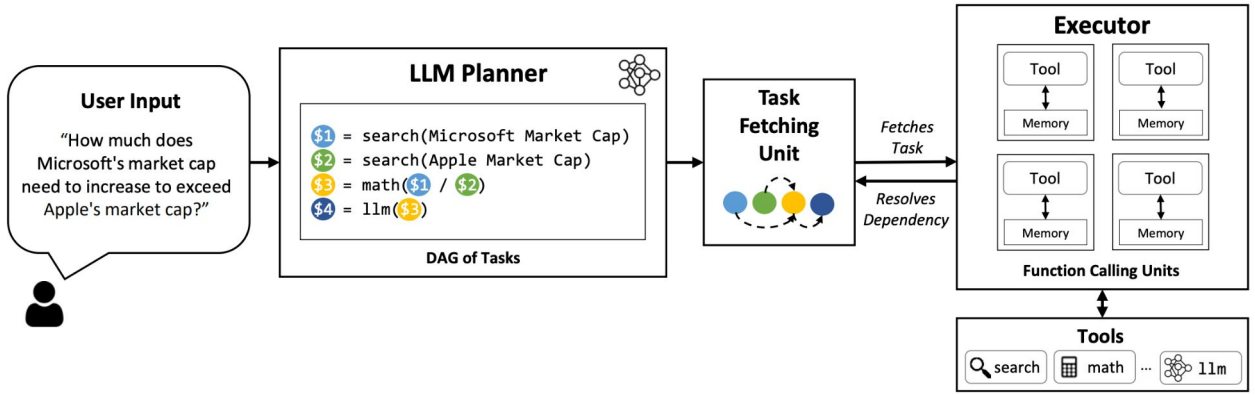


Figure 2: Overview of the `LLMCompiler` framework: the workflow from initial user input to task execution. Beginning with user input, the LLM Planner generates a sequence of tasks with their inter-dependencies. These tasks are then dispatched by the Task Fetching Unit to the Executor based on their dependencies, thus allowing for their parallel executions. For instance, in this example, Task `$1` and `$2` are fetched together for parallel execution of two independent search tasks. After each task is performed, the results (i.e., observations) are forwarded back to the Task Fetching Unit to unblock the dependent tasks after replacing their placeholder variables (e.g., the variable `$1` and `$2` in Task `$3`) with actual values. Once all tasks have been executed, the final answer is delivered to the user.

LLMCompiler

Plan out steps beforehand, and replan as necessary

[LLMCompiler Agent]

[17]:
```
response = agent.chat(
    "Is the climate of Chicago or Seattle better during the wintertime?"
)
print(str(response))
```

```
> Running step f8fdf4cb-9dde-4aba-996d-edbcee53c4c2 for task 20df27d7-cc27-4311-bb57-b1a6f4ad5799.
> Step count: 0
> Plan: 1. vector_tool_Chicago("climate during wintertime")
2. vector_tool_Seattle("climate during wintertime")
3. join()<END_OF_PLAN>
Ran task: vector_tool_Seattle. Observation: During wintertime, Seattle experiences cool, wet conditions. Extreme cold tempe
ratures, below about 15 °F or −9 °C, are rare due to the moderating influence of the adjacent Puget Sound, the greater Paci
fic Ocean, and Lake Washington. The city is often cloudy due to frequent storms and lows moving in from the Pacific Ocean,
and it has many "rain days". However, the rainfall is often a light drizzle.
Ran task: vector_tool_Chicago. Observation: During wintertime, the city experiences relatively cold and snowy conditions. B
lizzards can occur, as they did in winter 2011. The normal winter high from December through March is about 36 °F (2 °C). J
anuary and February are the coldest months. A polar vortex in January 2019 nearly broke the city's cold record of −27 °F
(−33 °C), which was set on January 20, 1985. Measurable snowfall can continue through the first or second week of April. Th
e city's proximity to Lake Michigan tends to keep the lakefront somewhat cooler in summer and less brutally cold in winter
than inland parts of the city and suburbs away from the lake. Northeast winds from wintertime cyclones departing south of t
he region sometimes bring the city lake-effect snow.
Ran task: join. Observation: None
> Thought: Comparing the two climates, Seattle seems to have a milder winter climate than Chicago.
> Answer: Seattle
Seattle
```

Tree-based Planning

Tree of Thoughts (Yao et al. 2023)

Reasoning via Planning (Hao et al. 2023)

Language Agent Tree Search (Zhou et al. 2023)

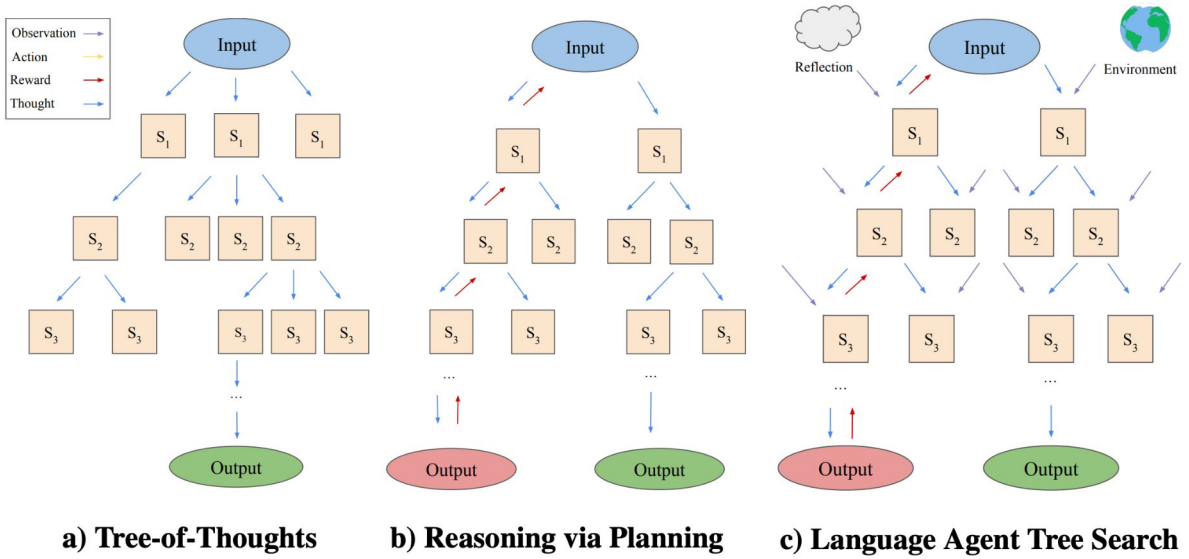


Figure 2: An overview of the differences between LATS and recently proposed LM search algorithms ToT (Yao et al., 2023a) and RAP (Hao et al., 2023). LATS leverages environmental feedback and self-reflection to further adapt search and improve performance.

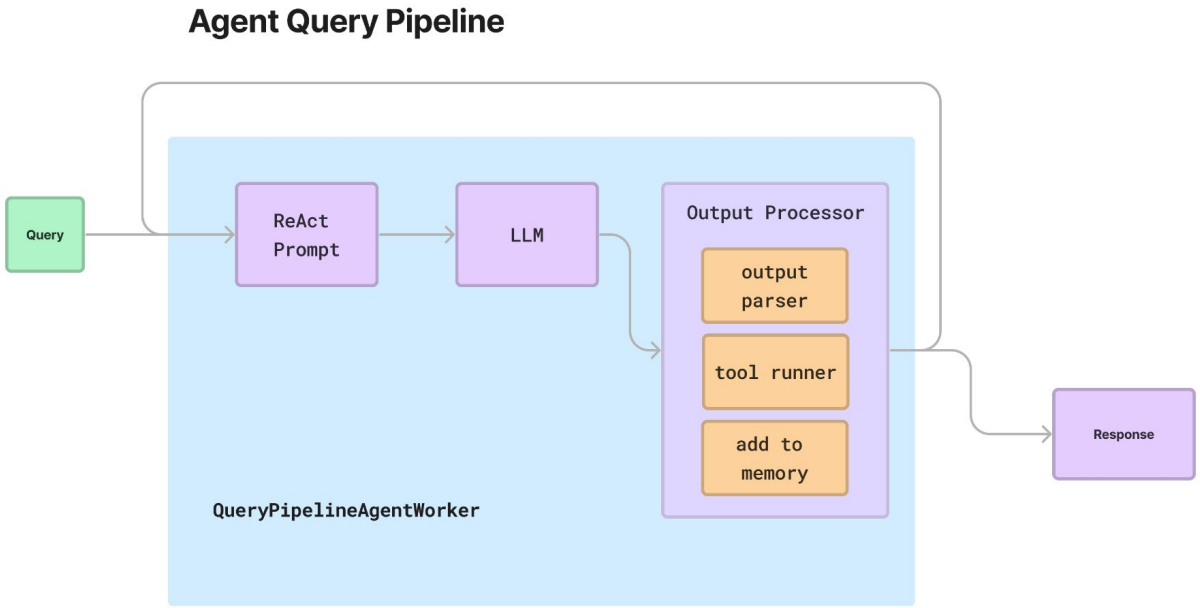
Additional Requirements

- **Observability:** see the full trace of the agent
  - Observability Guide
- **Control:** Be able to guide the intermediate steps of an agent *step-by-step*
  - Lower-Level Agent API
- **Customizability:** Define your own agentic logic around any set of tools.
  - Custom Agent Guide
  - Custom Agent with Query Pipeline Guide

Additional Requirements

Possible through our query pipeline syntax

[Query Pipeline Guide]



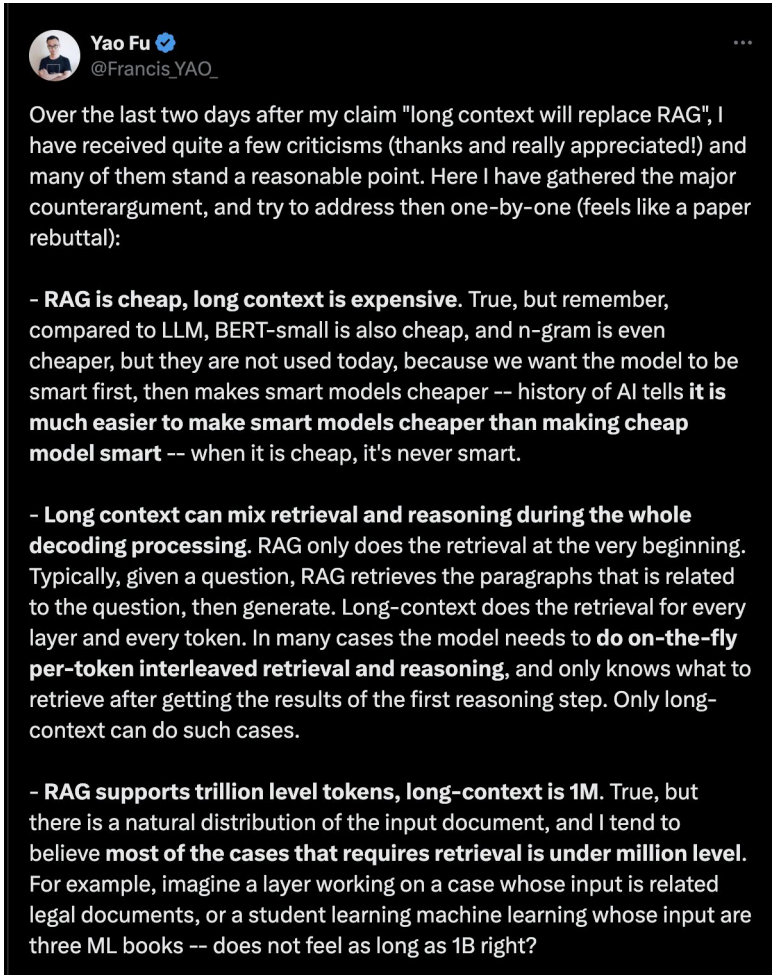
What’s next for RAG: Long Contexts?

Is RAG Dead?

Gemini 1.5 Pro has a 1-10M context window.

What does this mean for RAG?

https://x.com/Francis_YAO_/status/1759962812229800012?s=20



Yao Fu
@Francis_YAO_

Over the last two days after my claim "long context will replace RAG", I have received quite a few criticisms (thanks and really appreciated!) and many of them stand a reasonable point. Here I have gathered the major counterargument, and try to address then one-by-one (feels like a paper rebuttal):

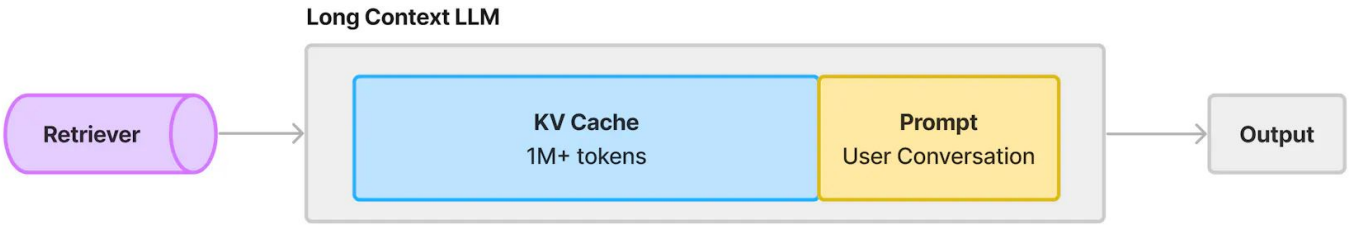
- **RAG is cheap, long context is expensive**. True, but remember, compared to LLM, BERT-small is also cheap, and n-gram is even cheaper, but they are not used today, because we want the model to be smart first, then makes smart models cheaper -- history of AI tells **it is much easier to make smart models cheaper than making cheap model smart** -- when it is cheap, it's never smart.

- **Long context can mix retrieval and reasoning during the whole decoding processing**. RAG only does the retrieval at the very beginning. Typically, given a question, RAG retrieves the paragraphs that is related to the question, then generate. Long-context does the retrieval for every layer and every token. In many cases the model needs to **do on-the-fly per-token interleaved retrieval and reasoning**, and only knows what to retrieve after getting the results of the first reasoning step. Only long-context can do such cases.

- **RAG supports trillion level tokens, long-context is 1M**. True, but there is a natural distribution of the input document, and I tend to believe **most of the cases that requires retrieval is under million level**. For example, imagine a layer working on a case whose input is related legal documents, or a student learning machine learning whose input are three ML books -- does not feel as long as 1B right?

Our Position

1. Frameworks are valuable whether or not RAG lives or dies
2. Certain RAG concepts will go away, but others will remain and evolve



Long Context LLMs will Solve the Following

1. Developers will worry less about tuning chunking algorithms
2. Developers will need to spend less time tuning retrieval and chain-of-thought over single documents
3. Summarization will be easier
4. Personalized memory will be better and easier to build

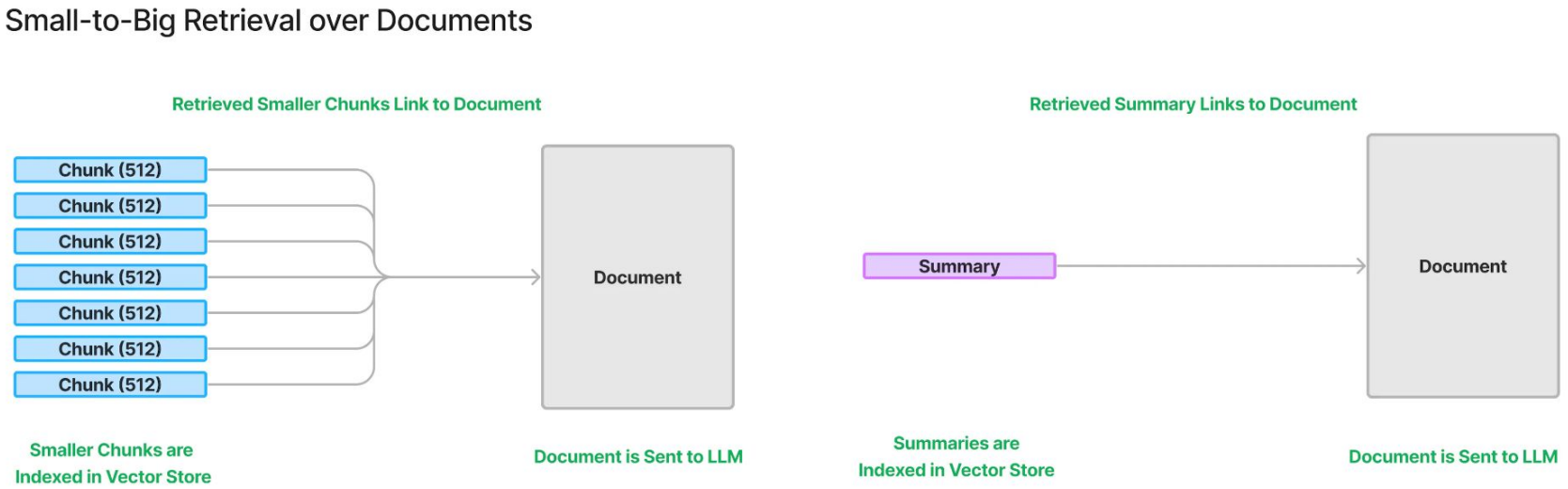
Some Challenges Remain

1. 10M tokens is not enough for large document corpuses (hundreds of MB, GB)
2. Embedding models are lagging behind in context length
3. Cost and Latency
4. A KV Cache takes up a significant amount of GPU memory, and has sequential dependencies

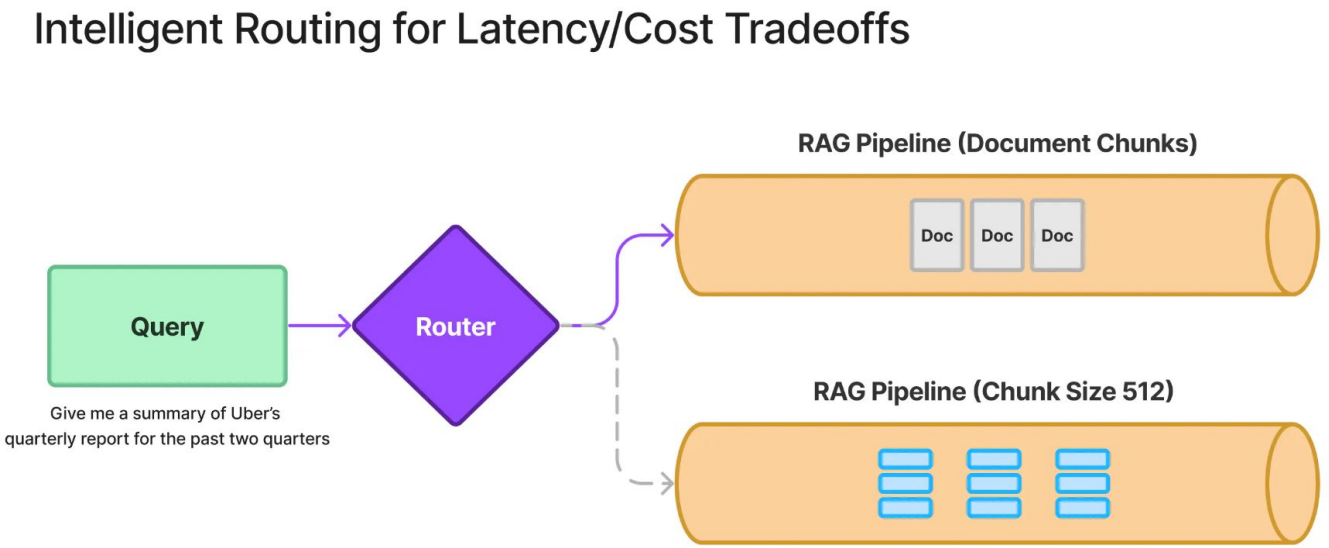
New RAG Architectures

1. Small to Big Retrieval over Documents
2. Intelligent Routing for Latency/Cost Tradeoffs
3. Retrieval Augmented KV Caching

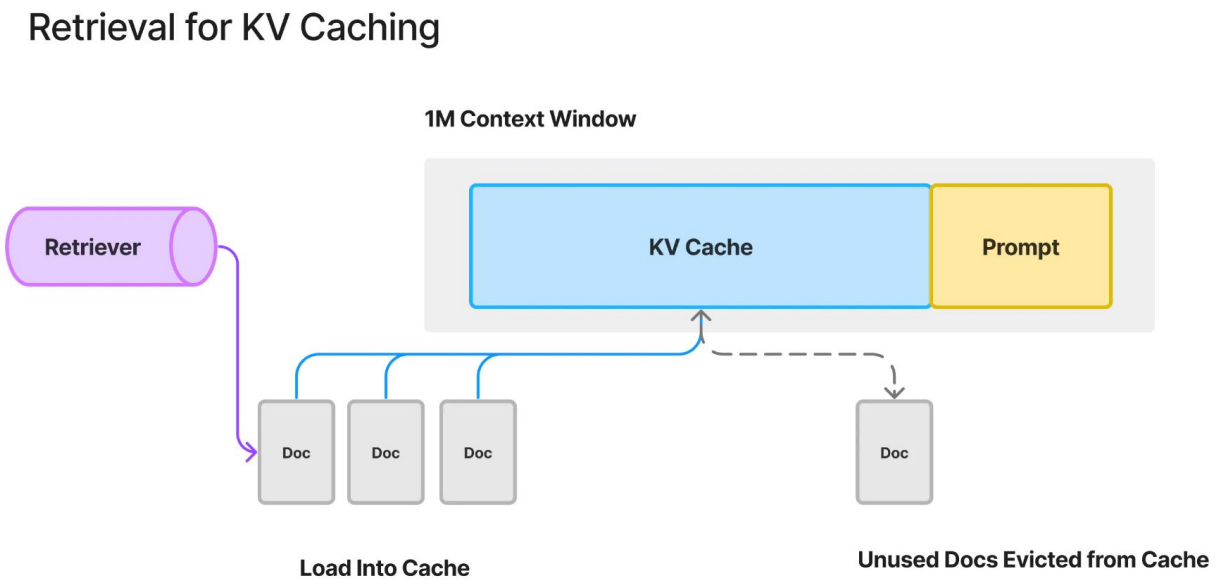
Small to Big Retrieval over Documents



Intelligent Routing for Latency/Cost Tradeoffs



Retrieval Augmented KV Caching



Thanks for your attention!

Hiring!
- AI Engineer
- Applied AI Engineer (customer-facing)
- Backend Engineer
- Frontend/Product Engineer
- Typescript Engineer (for LlamaIndex.TS)