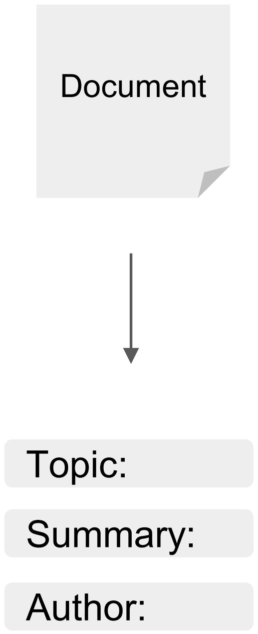
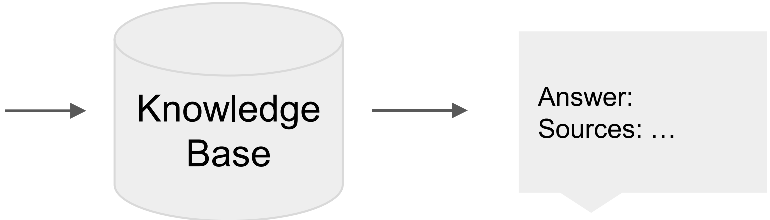
Building and Productionizing RAG

Jerry Liu, LlamaIndex co-founder/CEO

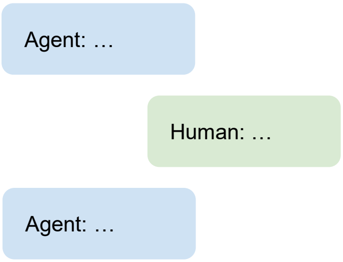
GenAI - Enterprise Use-cases



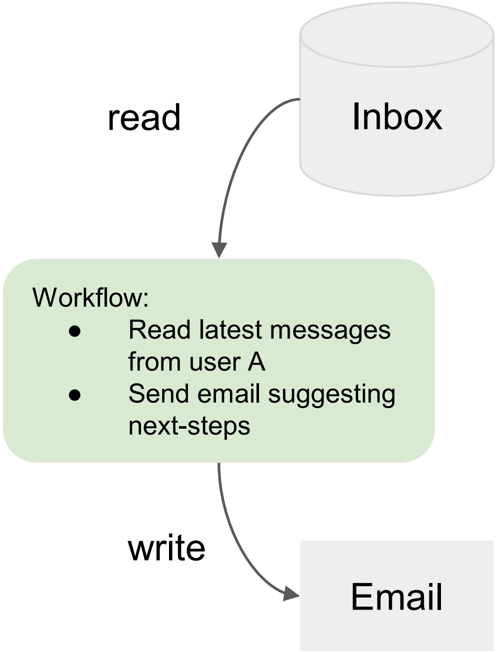
Document Processing
Tagging & Extraction



Knowledge Search & QA



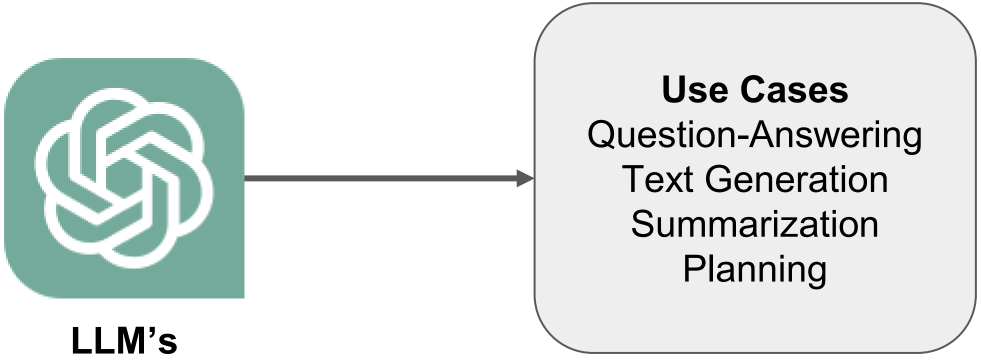
Conversational Agent



Workflow Automation

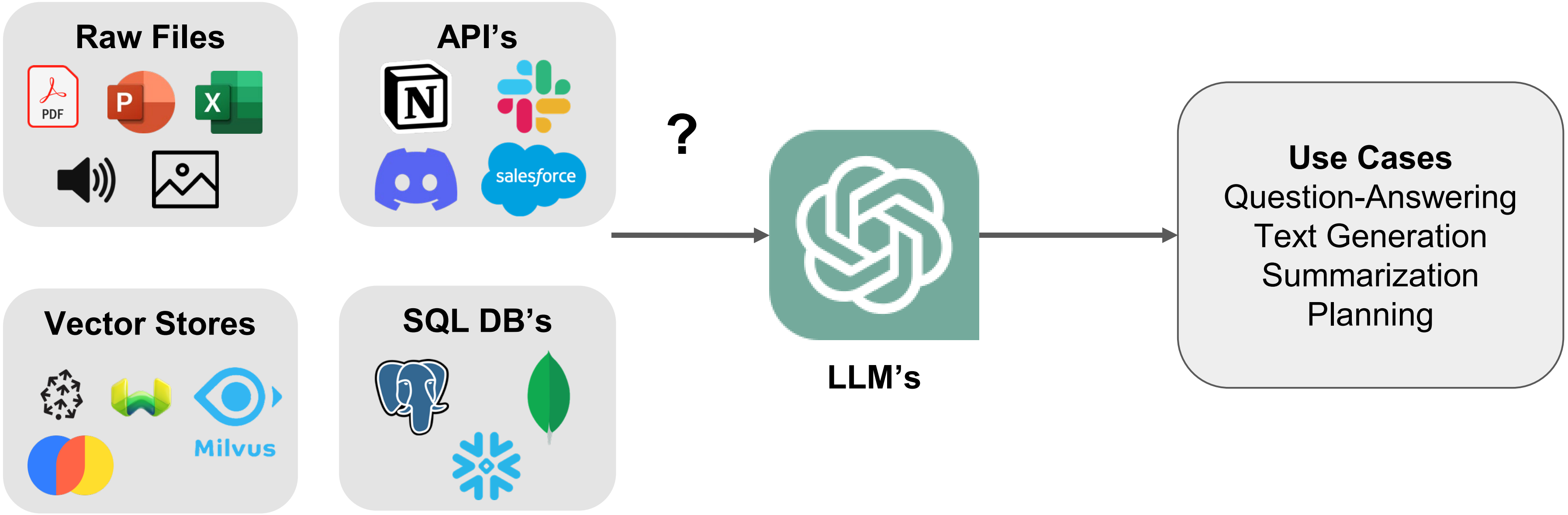
Context

- LLMs are a phenomenal piece of technology for knowledge generation and reasoning. They are pre-trained on large amounts of **publicly available data**.



Context

- How do we best augment LLMs with our own **private data**?



Paradigms for inserting knowledge

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

Before college the two main things I worked on, outside of school, were writing and programming. I didn't write essays. I wrote what beginning writers were supposed to write then, and probably still are: short stories. My stories were awful. They had hardly any plot, just characters with strong feelings, which I imagined made them deep...

Input Prompt

Here is the context:
Before college the two main things…

Given the context, answer the following question:
{query_str}

LLM

Paradigms for inserting knowledge

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

Before college the two main things I worked on, outside of school, were writing and programming. I didn't write essays. I wrote what beginning writers were supposed to write then, and probably still are: short stories. My stories were awful. They had hardly any plot, just characters with strong feelings, which I imagined made them deep...

LLM

RLHF, Adam, SGD, etc.

LlamaIndex: A data framework for LLM applications

- Data Management and Query Engine for your LLM application
- Offers components across the data lifecycle: ingest, index, and query over data

Data Ingestion (LlamaHub 🦙) → **Data Structures** → **Queries**

Data Ingestion (LlamaHub)

- Connect your existing data sources and data formats (API's, PDF's, docs, SQL, etc.)

Data Structures

- Store and index your data for different use cases. Integrate with different db's (vector db, graph db, kv db)

Queries

- Retrieve and query over data
- Includes: QA, Summarization, Agents, and more

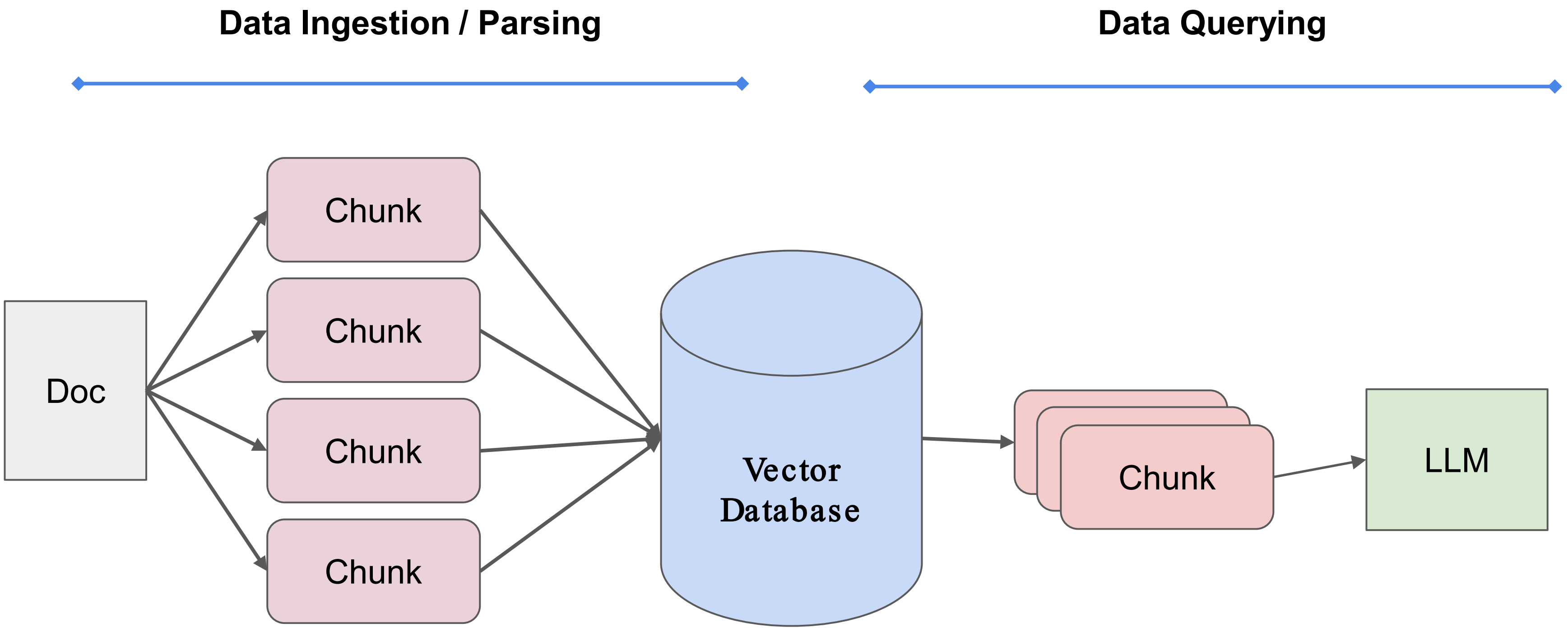
quickstart.py

```python
from llama_index 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(str(response))
```

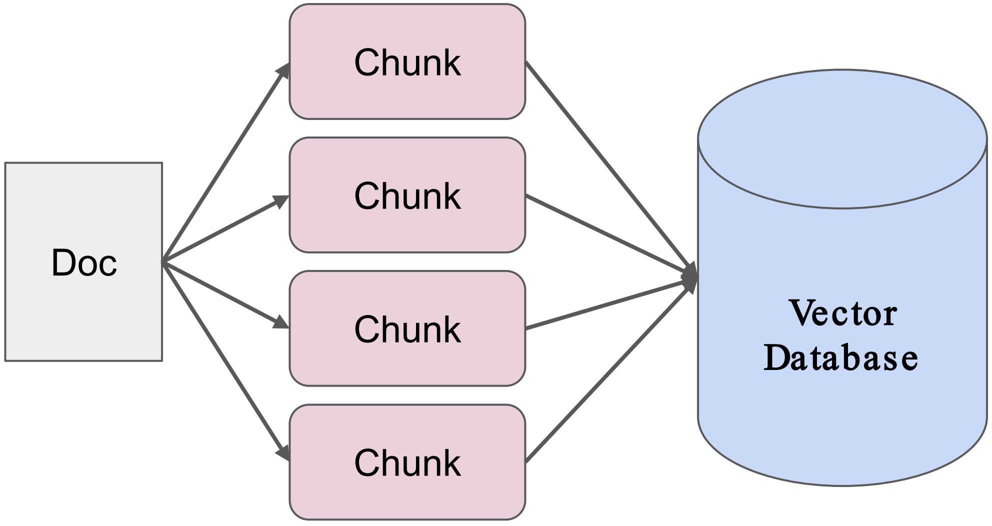
RAG Stack

Current RAG Stack for building a QA System



5 Lines of Code in LlamaIndex!

Current RAG Stack (Data Ingestion/Parsing)



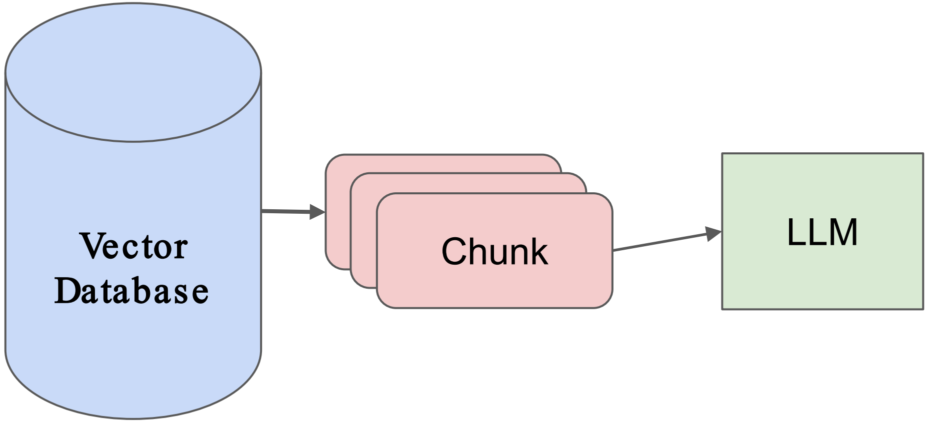
Process:

- 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)

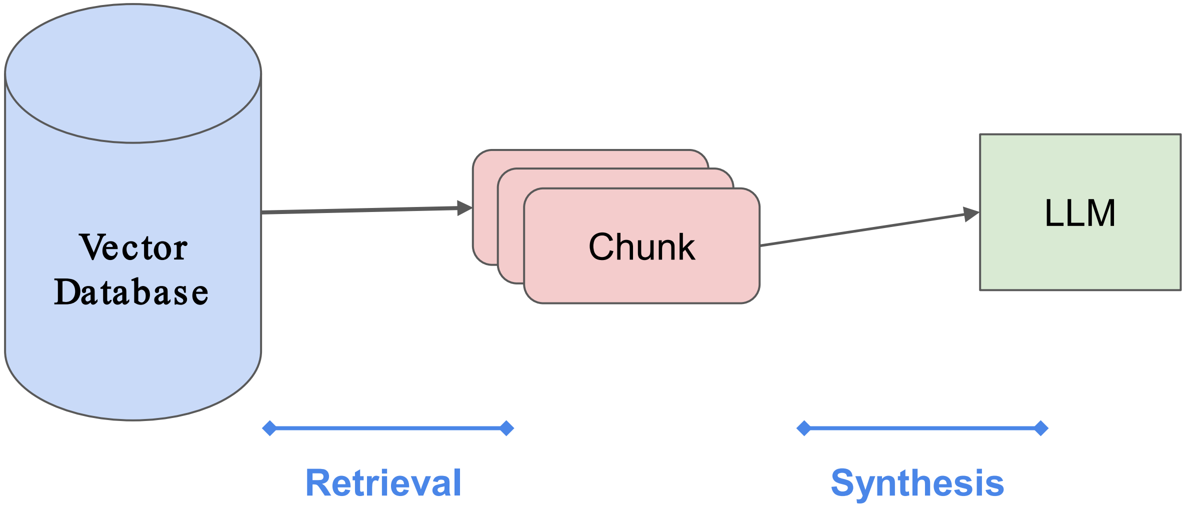
Process:

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



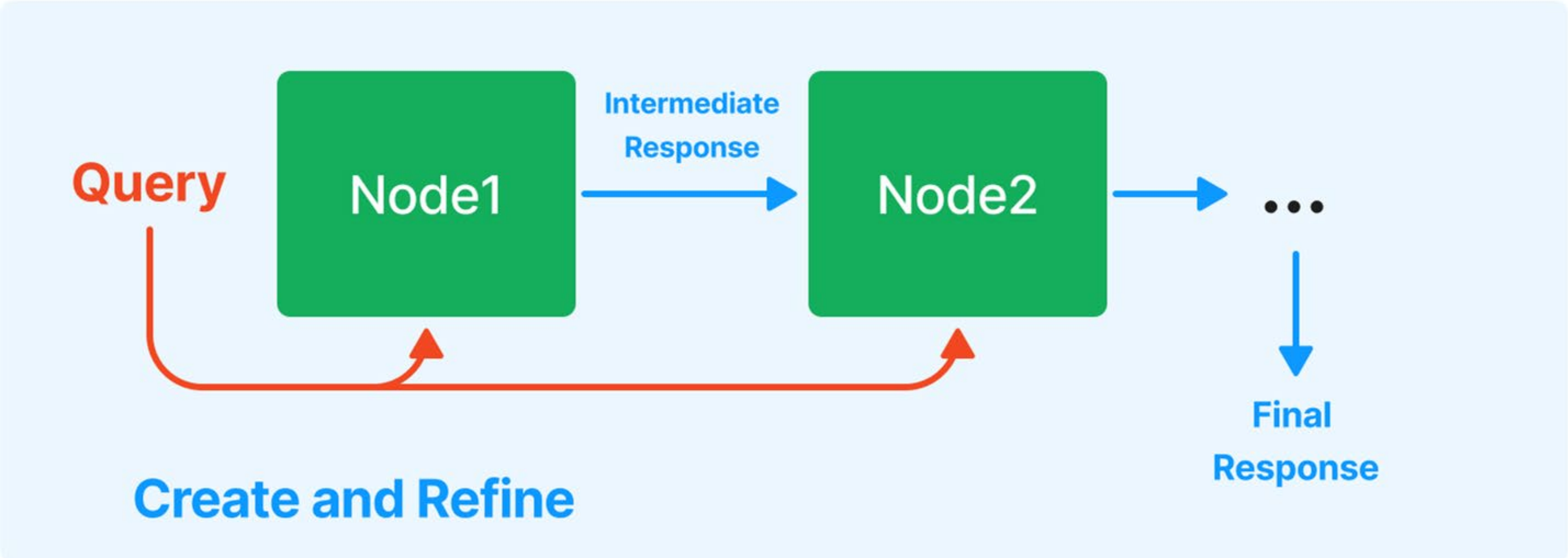
Current RAG Stack (Querying)

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



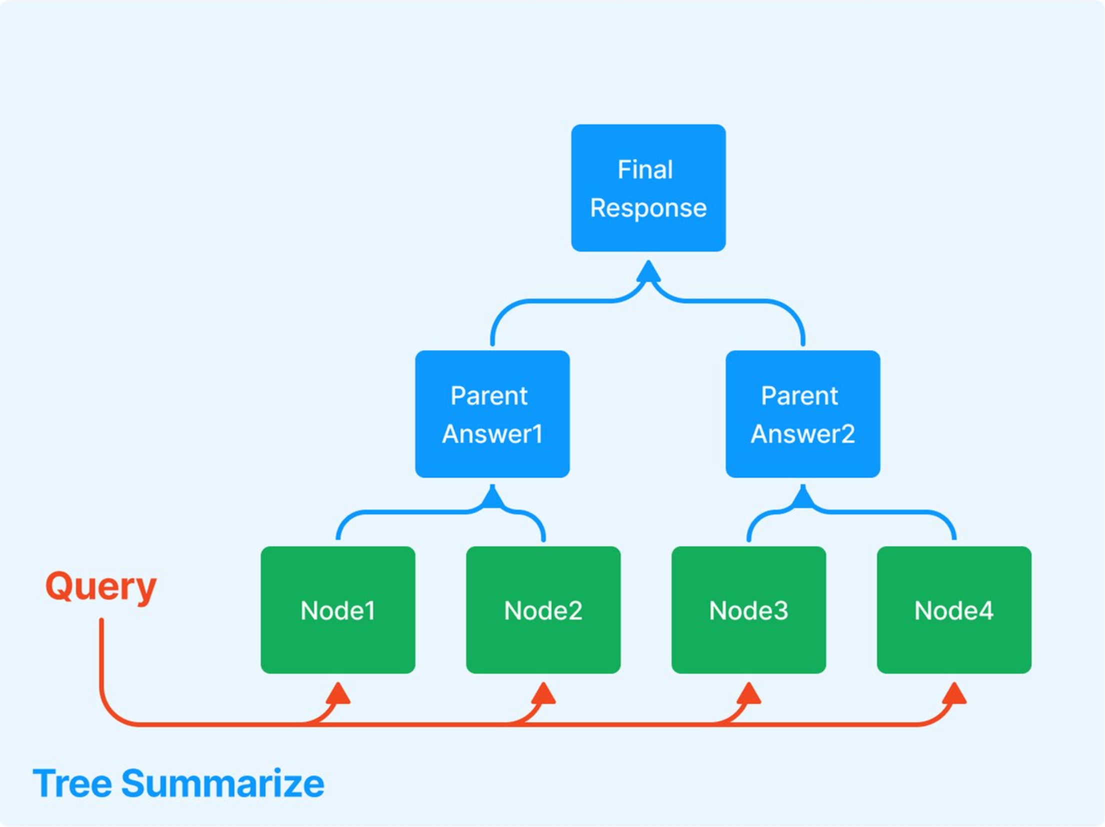
Response Synthesis

Create and refine



Response Synthesis

Tree Summarize



Quickstart

https://colab.research.google.com/drive/1knQpGJLHj-LTTHqIZhgcjDH5F_nJIiY0?usp=sharing

Challenges with “Naive” RAG

Challenges with Naive RAG

- Failure Modes
  - Quality-Related (Hallucination, Accuracy)
  - Non-Quality-Related (Latency, Cost, Syncing)

Challenges with Naive RAG (Response Quality)

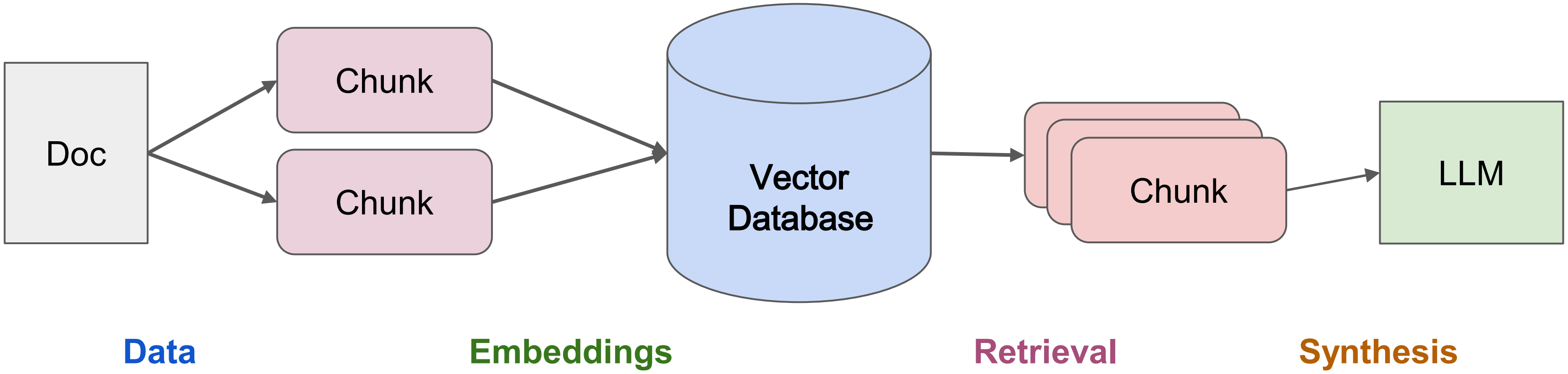
- Bad Retrieval
  - **Low Precision:** Not all chunks in retrieved set are relevant
    - Hallucination + Lost in the Middle Problems
  - **Low Recall:** Now all relevant chunks are retrieved.
    - Lacks enough context for LLM to synthesize an answer
  - **Outdated information:** The data is redundant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

What do we do?

- **Data**: Can we store additional information beyond raw text chunks?
- **Embeddings**: Can we optimize our embedding representations?
- **Retrieval**: Can we do better than top-k embedding lookup?
- **Synthesis**: Can we use LLMs for more than generation?



What do we do?

- **Data:** Can we store additional information beyond raw text chunks?
- **Embeddings:** Can we optimize our embedding representations?
- **Retrieval:** Can we do better than top-k embedding lookup?
- **Synthesis:** Can we use LLMs for more than generation?

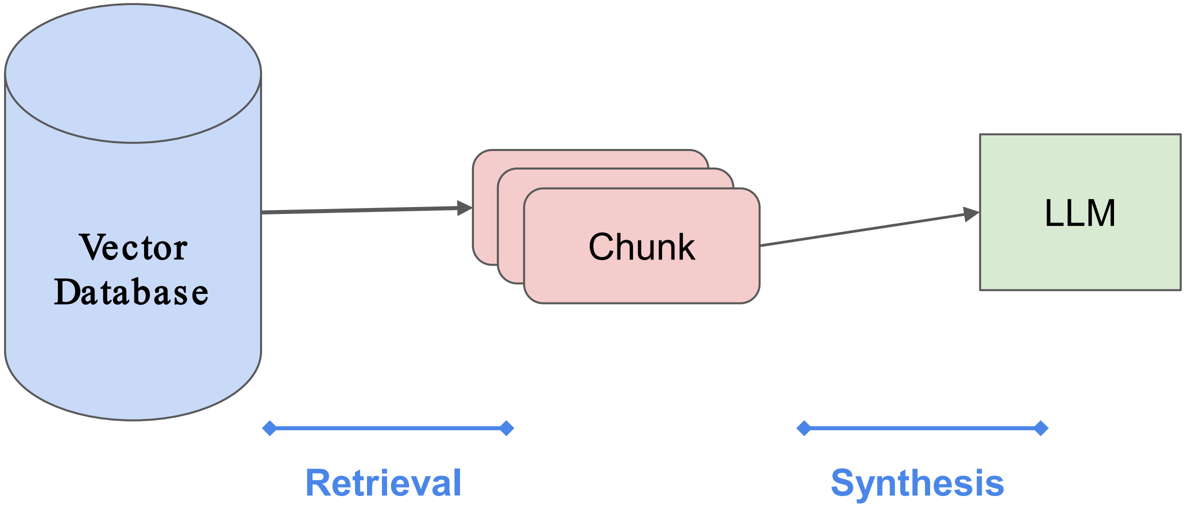
But before all this…

We need a way to measure performance

Evaluation

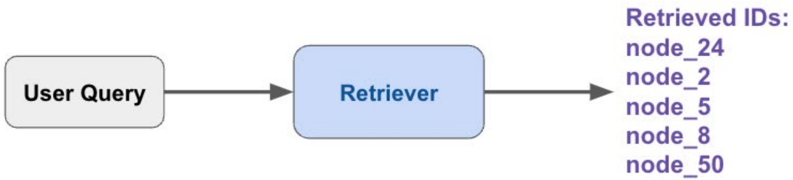
Evaluation

- How do we properly evaluate a RAG system?
  - Evaluate in isolation (retrieval, synthesis)
  - Evaluate e2e
- Open question: which one should we do first?



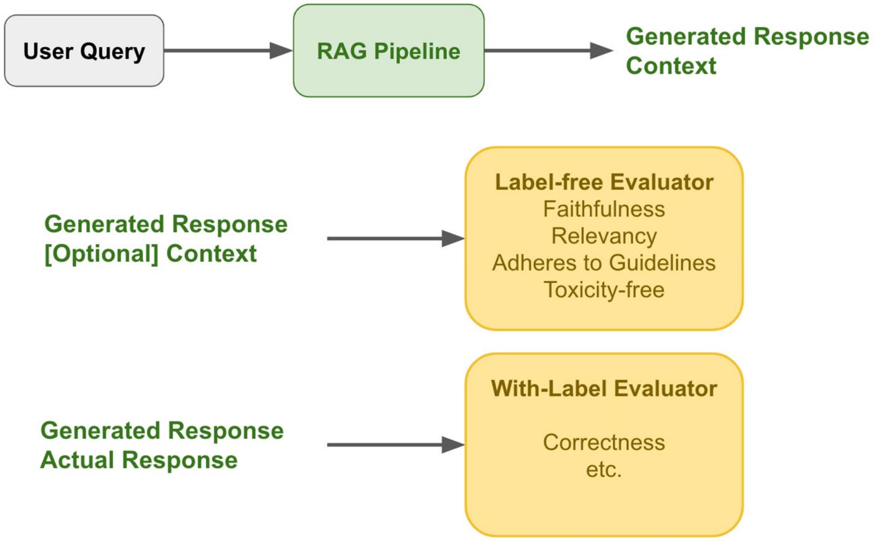
Evaluation in Isolation (Retrieval)

- **Details:** Evaluate quality of retrieved chunks given user query
- **Create dataset**
  - Input: query
  - Output: the “ground-truth” documents relevant to the query
- Run retriever over dataset
- Measure **ranking metrics**
  - Success rate / hit-rate
  - MRR
  - Hit-rate



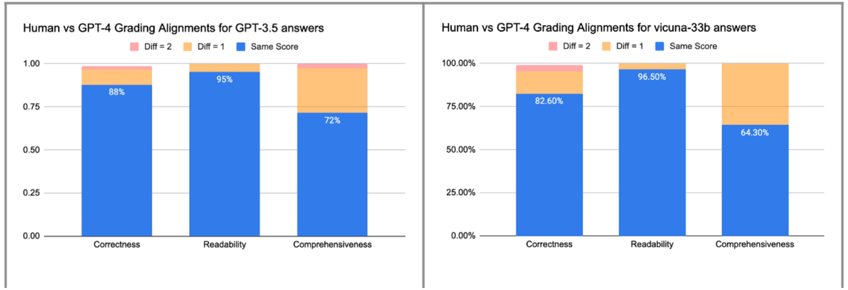
Evaluation E2E

- **Details:** Evaluation of final generated response given input
- **Create Dataset**
  - Input: query
  - [Optional] Output: the "ground-truth" answer
- Run through full RAG pipeline
- Collect evaluation metrics:
  - **If no labels:** label-free evals
  - **If labels:** with-label evals

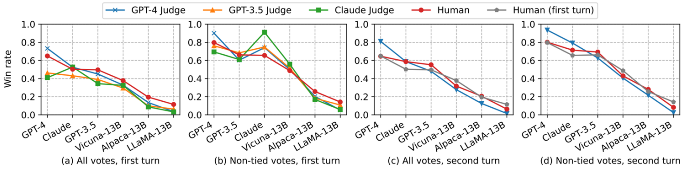


LLM-based Evaluation Modules

- GPT-4 is a good human grader
- **Label-free Modules**
    - **Faithfulness**: whether response matches retrieved context
    - **Relevancy**: whether response matches query
    - **Guidelines**: whether response matches guidelines
- **With-Labels**
    - **Correctness**: whether response matches “golden” answer.



https://www.databricks.com/blog/LLM-auto-eval-best-practices-RAG



https://arxiv.org/pdf/2306.05685.pdf

Optimizing RAG Systems

From Simple to Advanced

Fine-tuning
Embedding fine-tuning
LLM fine-tuning

Table Stakes
Better Parsers
Chunk Sizes
Hybrid Search
Metadata Filters

Advanced Retrieval
Reranking
Recursive Retrieval
Embedded Tables
Small-to-big Retrieval

Agentic Behavior
Routing
Query Planning
Multi-document Agents

Less Expressive
Easier to Implement
Lower Latency/Cost

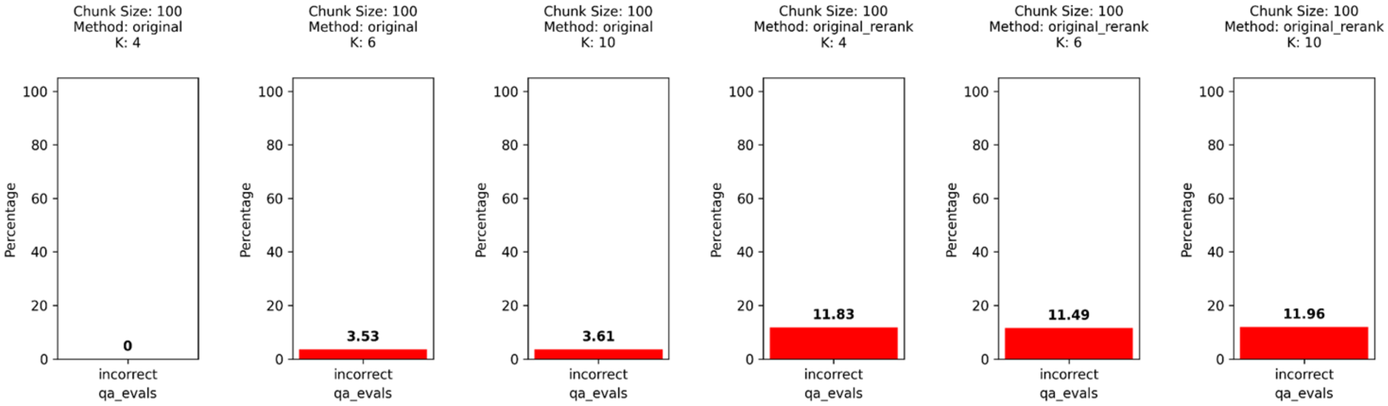
More Expressive
Harder to Implement
Higher Latency/Cost

Table Stakes: Chunk Sizes

Tuning your chunk size can have outsized impacts on performance

Not obvious that more retrieved tokens == higher performance!

Note: Reranking (shuffling context order) isn't always beneficial.



Source:
Arize Phoenix + LlamaIndex Workshop: https://colab.research.google.com/drive/1Siufl13rLI-kII1liaNfvf-NniBdwUpS?usp=sharing#scrollTo=_as7h-u1IlwR

Table Stakes: Prompt Engineering

RAG uses core Question-Answering (QA) prompt templates

Ways you can customize:

- Adding few-shot examples
- Modifying template text
- Adding emotions

Accessing Prompts

Here we get the prompts from the query engine. Note that *all* prompts are returned, including ones used in sub-modules in the query engine. This allows you to centralize a view of these prompts!

```
prompts_dict = query_engine.get_prompts()
```

```
display_prompt_dict(prompts_dict)
```

Prompt Key: response_synthesizer:summary_template

Text:

```
Context information from multiple sources is below.
---------------------
{context_str}
---------------------
Given the information from multiple sources and not prior knowledge, answer the query.
Query: {query_str}
Answer:
```

Table Stakes: Customizing LLMs

Task performance on easy-to-hard tasks (RAG, agents) varies wildly among LLMs

Paid LLM APIs

Model Name	Basic Query Engines	Router Query Engine	Sub Question Query Engine	Text2SQL	Pydantic Programs	Data Agents	Notes
gpt-3.5-turbo (openai)	✅	✅	✅	✅	✅	✅	
gpt-3.5-turbo-instruct (openai)	✅	✅	✅	✅	✅	⚠️	Tool usage in data-agents seems flakey.
gpt-4 (openai)	✅	✅	✅	✅	✅	✅	
claude-2 (anthropic)	✅	✅	✅	✅	✅	⚠️	Prone to hallucinating tool inputs.
claude-instant-1.2 (anthropic)	✅	✅	✅	✅	✅	⚠️	Prone to hallucinating tool inputs.

Open Source LLMs

Since open source LLMs require large amounts of resources, the quantization is reported. Quantization is just a method for reducing the size of an LLM by shrinking the accuracy of calculations within the model. Research has shown that up to 4Bit quantization can be achieved for large LLMs without impacting performance too severely.

Model Name	Basic Query Engines	Router Query Engine	SubQuestion Query Engine	Text2SQL	Pydantic Programs	Data Agents	Notes
llama2-chat-7b 4bit (huggingface)	✅	🔴	🔴	🔴	🔴	⚠️	Llama2 seems to be quite chatty, which makes parsing structured outputs difficult. Fine-tuning and prompt engineering likely required for better performance on structured outputs.
Mistral-7B-instruct-v0.1 4bit (huggingface)	✅	🔴	🔴	⚠️	⚠️	⚠️	Mistral seems slightly more reliable for structured outputs compared to Llama2. Likely with some prompt engineering, it may do better.
zephyr-7b-alpha (huggingface)	✅	✅	✅	✅	✅	⚠️	Overall, `zyphyr-7b` is appears to be more reliable than other open-source models of this size. Although it still hallucinates a bit, especially as an agent.

Table Stakes: Customizing Embeddings

Your embedding model + reranker affects retrieval quality

Embedding	WithoutReranker		bge-reranker-base		bge-reranker-large		Cohere-Reranker	
	Hit Rate	MRR	Hit Rate	MRR	Hit Rate	MRR	Hit Rate	MRR
OpenAI	0.876404	0.718165	0.91573	0.832584	0.910112	0.855805	0.926966	0.86573
bge-large	0.752809	0.597191	0.859551	0.805243	0.865169	0.816011	0.876404	0.822753
llm-embedder	0.814607	0.587266	0.870787	0.80309	0.876404	0.824625	0.882022	0.830243
Cohere-v2	0.780899	0.570506	0.876404	0.798127	0.876404	0.825281	0.876404	0.815543
Cohere-v3	0.825843	0.624532	0.882022	0.806086	0.882022	0.834644	0.88764	0.836049
Voyage	0.831461	0.68736	0.926966	0.837172	0.91573	0.858614	0.91573	0.851217
JinaAI-Small	0.831461	0.614045	0.91573	0.843071	0.926966	0.857303	0.926966	0.868633
JinaAI-Base	0.848315	0.68221	0.938202	0.846348	0.938202	0.868539	0.932584	0.873689
Google-PaLM	0.865169	0.719476	0.910112	0.833708	0.910112	0.85309	0.910112	0.855712

Source: https://blog.llamaindex.ai/boosting-rag-picking-the-best-embedding-reranker-models-42d079022e83

Table Stakes: Metadata Filtering

- **Metadata:** context you can inject into each text chunk
- Examples
  - Page number
  - Document title
  - Summary of adjacent chunks
  - Questions that chunk can answer (reverse HyDE)
- **Benefits**
  - **Can Help Retrieval**
  - **Can Augment Response Quality**
  - **Integrates with Vector DB Metadata Filters**

Example of Metadata

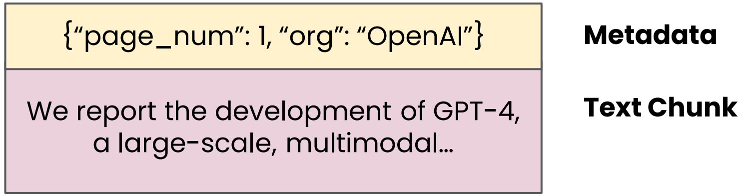
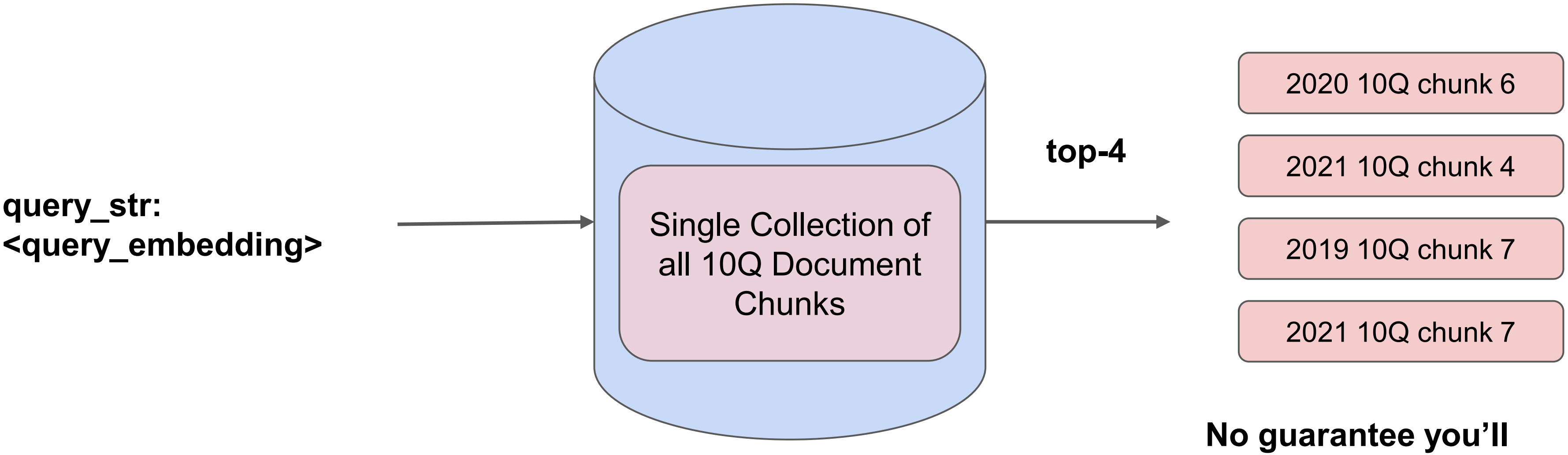


Table Stakes: Metadata Filtering

Question: “Can you tell me the risk factors in 2021?”

Raw Semantic Search is **low precision.**



No guarantee you’ll return the relevant document chunks!

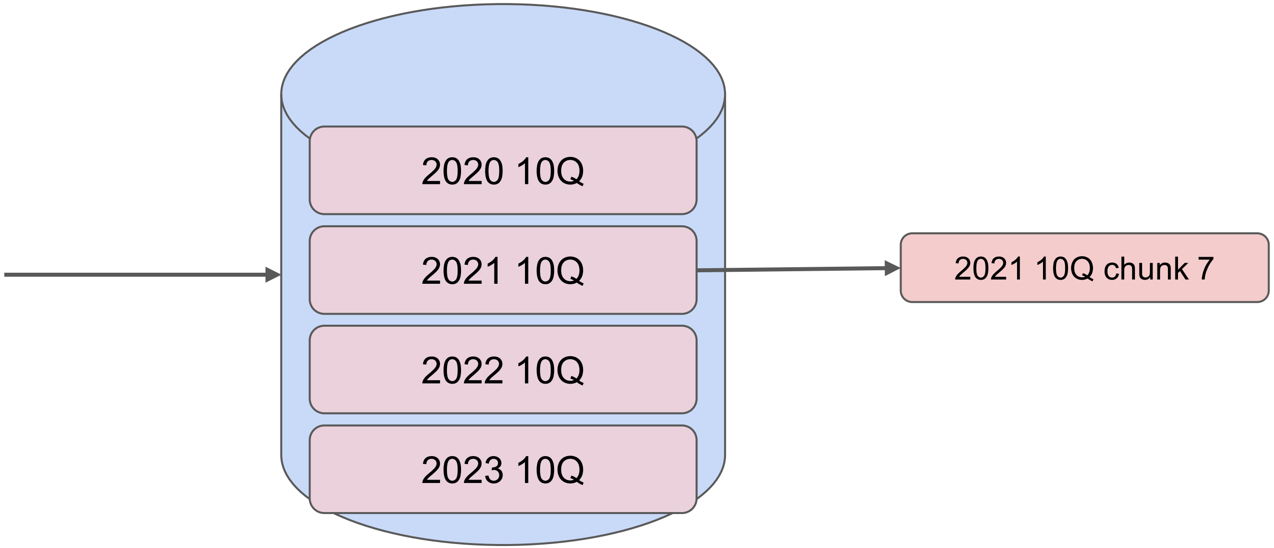
Table Stakes: Metadata Filtering

Question: “Can you tell me the risk factors in 2021?”

If we can *infer* the metadata filters (year=2021), we remove irrelevant candidates, **increasing precision**!

query_str:
<query_embedding>

Metadata tags:
{“year”: 2021}

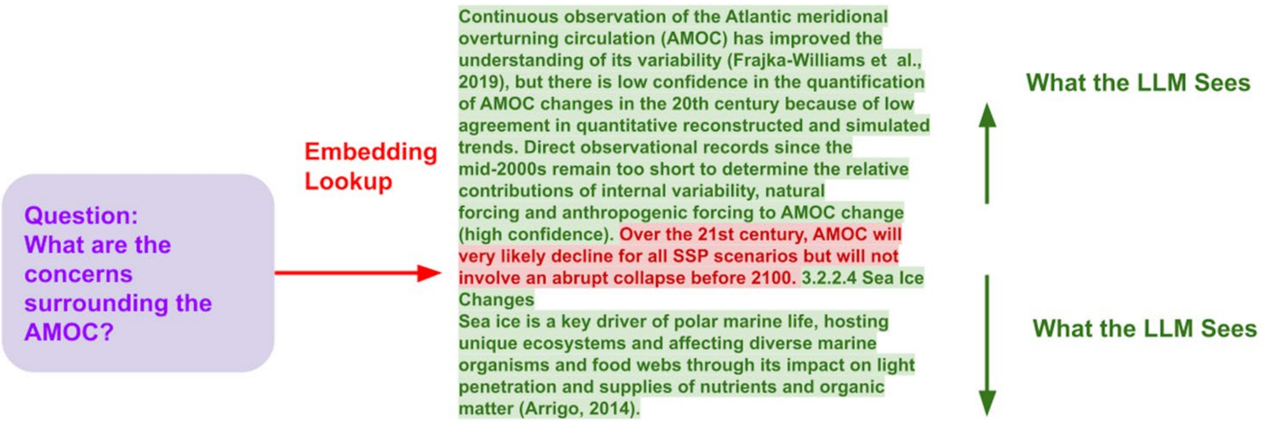


Advanced Retrieval: Small-to-Big

Intuition: Embedding a big text chunk feels suboptimal.

Solution: Embed text at the sentence-level - then **expand** that window during LLM synthesis

Embed Sentence → Link to Expanded Window



Advanced Retrieval: Small-to-Big

Leads to more **precise** retrieval.

Avoids “lost in the middle” problems.

```
There is low confidence in the quantification of AMOC changes in the 20th ce
ntury due to low agreement in quantitative reconstructed and simulated trend
s. Additionally, direct observational records since the mid-2000s remain too
short to determine the relative contributions of internal variability, natur
al forcing, and anthropogenic forcing to AMOC change. However, it is very li
kely that AMOC will decline over the 21st century for all SSP scenarios, but
there will not be an abrupt collapse before 2100.
```

Sentence Window Retrieval (k=2)

```
I'm sorry, but the concerns surrounding the AMOC (Atlantic Meridional Overtu
rning Circulation) are not mentioned in the provided context.
```

Naive Retrieval (k=5)

Only one out of the 5 chunks is relevant - “lost in the middle” problem

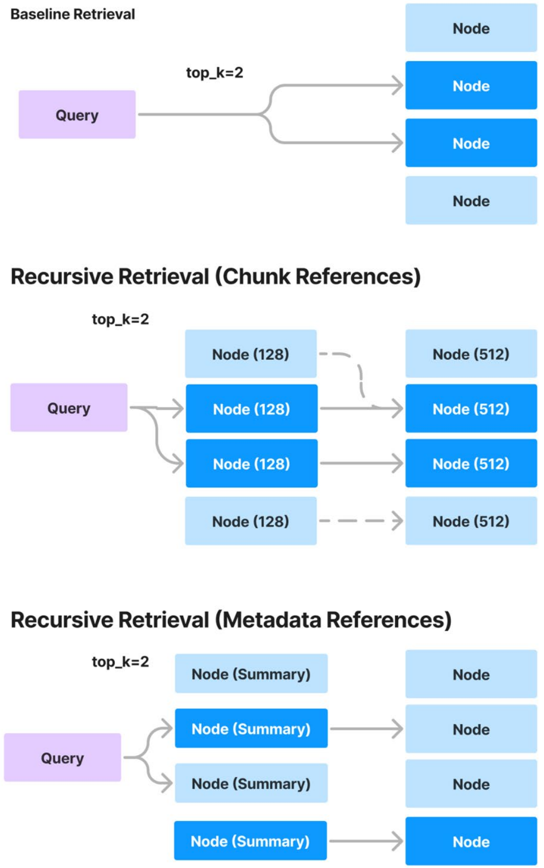
Advanced Retrieval: Small-to-Big

Intuition: Embedding a big text chunk feels suboptimal.

Solution: Embed a smaller **reference** to the parent chunk.
Use parent chunk for synthesis

Examples: Smaller chunks, summaries, metadata

	retrievers	hit_rate	mrr
0	Base Retriever	0.796407	0.605097
1	Retriever (Chunk References)	0.892216	0.739179
2	Retriever (Metadata References)	0.916168	0.746906

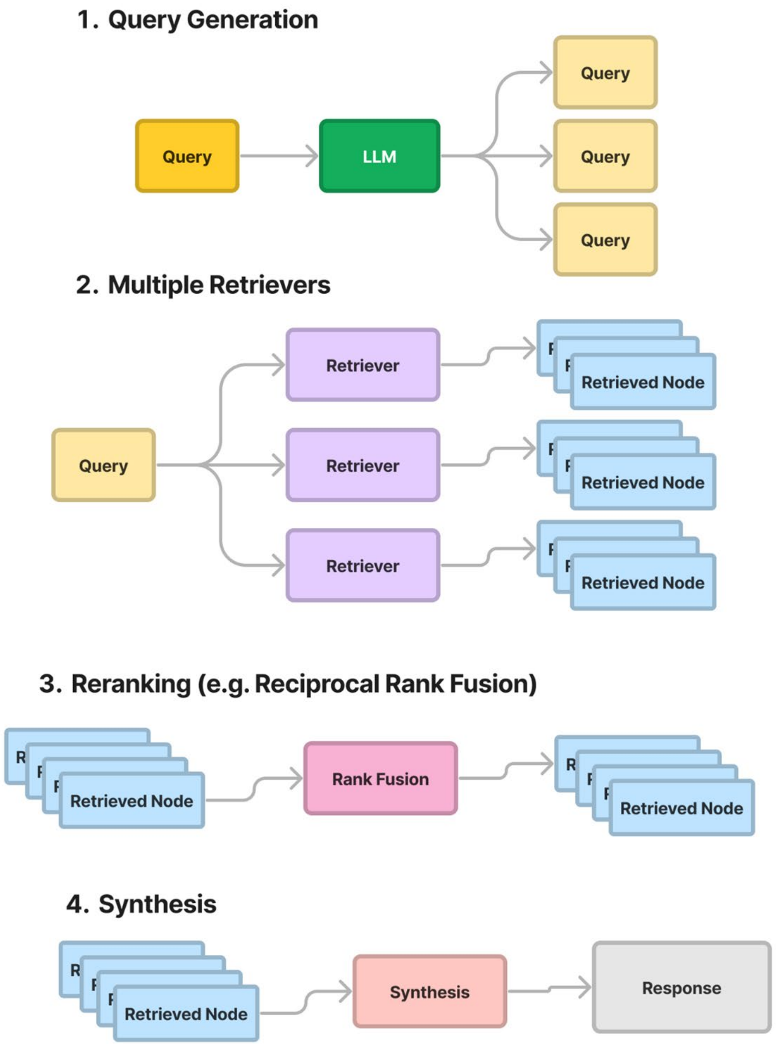


Advanced Retrieval Architecture

RAG Fusion Retriever

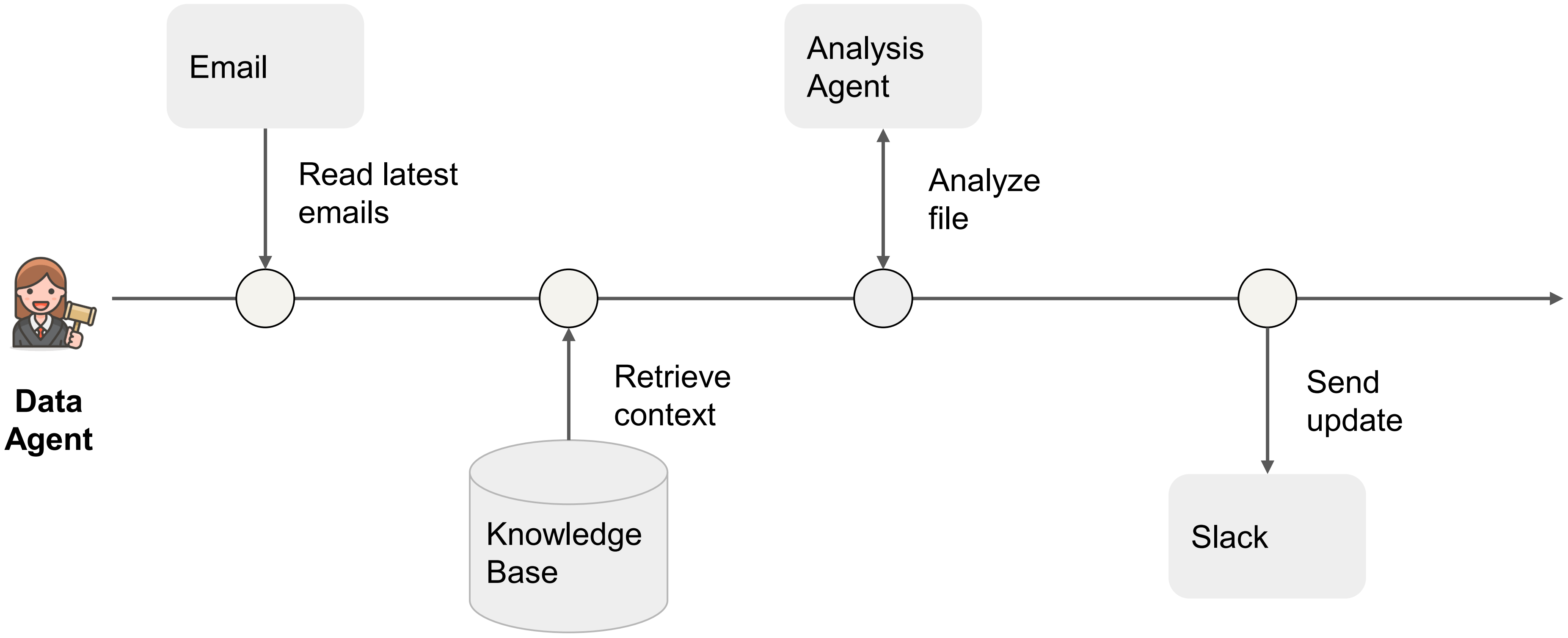
Combines best practices around retrieval:

1. Query Generation/Rewriting
2. Ensemble Retrieval
3. Reranking (with Reciprocal Rank Fusion)
4. Synthesis

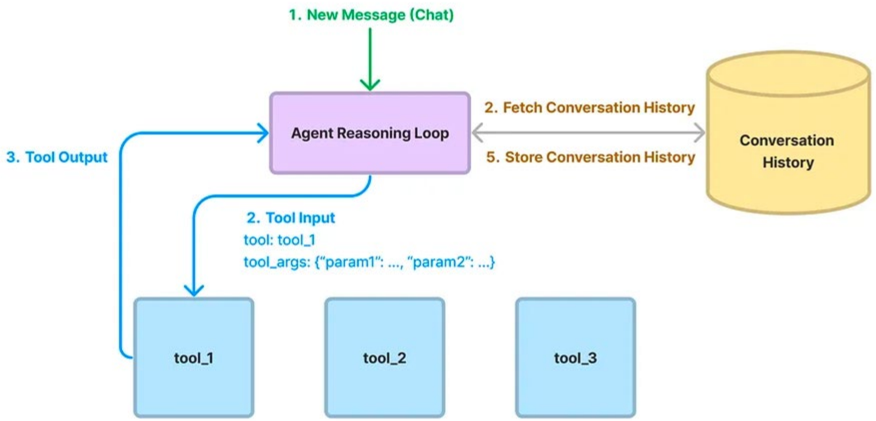


Agents

Data Agents - LLM-powered knowledge workers



Data Agents - Core Components



Agent Reasoning Loop

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

Tools

Query Engine Tools (RAG pipeline)

LlamaHub Tools

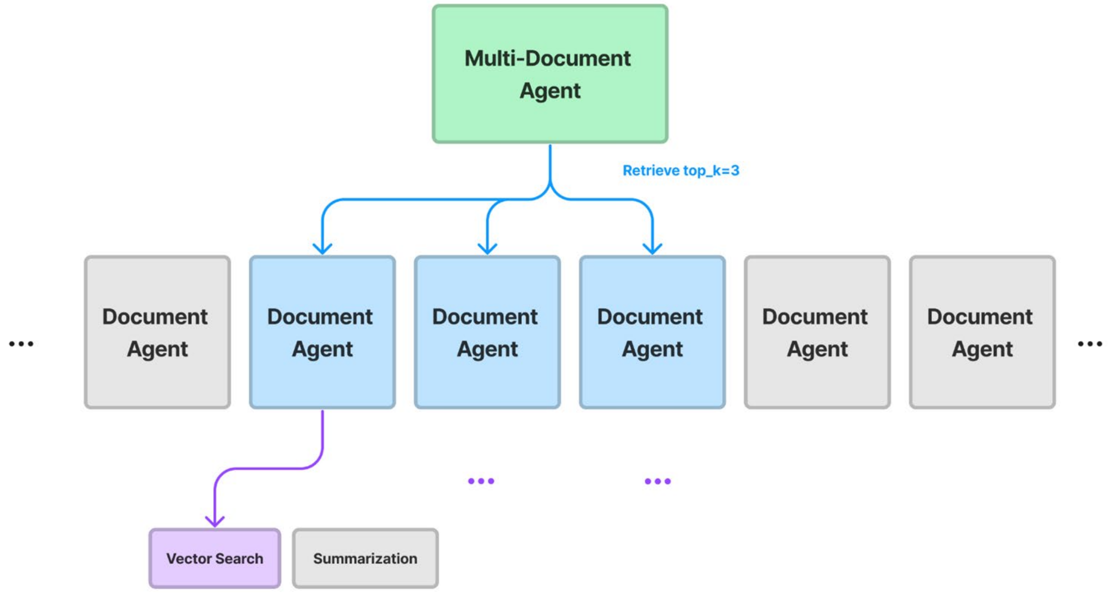
- Code interpreter
- Slack
- Notion
- Zapier
- … (15+ tools)

Agentic Behavior: Multi-Document Agents

Intuition: There's certain questions that "top-k" RAG can't answer.

Solution: Multi-Document Agents

- Fact-based QA and Summarization over any subsets of documents
- Chain-of-thought and query planning.



How to retrieve & analyze data from knowledge base?

Use our query engines as “data tools” over your agent:

- Semantic search
- Summarization
- Text-to-SQL
- Document comparisons
- Combining Structured Data w/ Unstructured

“Simple” Interface - all agent has to infer is a query string!

Example Notebook:

- OpenAI Agent + query engines (as tools)
- Analyzing structured + unstructured data

```
query_engine_tools = [
    QueryEngineTool(
        query_engine=lyft_engine,
        metadata=ToolMetadata(
            name='lyft_10k',
            description="Provides information about Lyft financials for year 2021. "
            "Use a detailed plain text question as input to the tool."
        )
    ),
    QueryEngineTool(
        query_engine=uber_engine,
        metadata=ToolMetadata(
            name='uber_10k',
            description="Provides information about Uber financials for year 2021. "
            "Use a detailed plain text question as input to the tool."
        )
    ),
]
```

Example: Financial Analysis with Agents + RAG

Question: “Compare and contrast Uber and Lyft’s revenue growth”

- **Agent:** breaks down question into sub-questions over tools
- **Per-Document RAG:** Answer question over a given document via top-k retrieval.

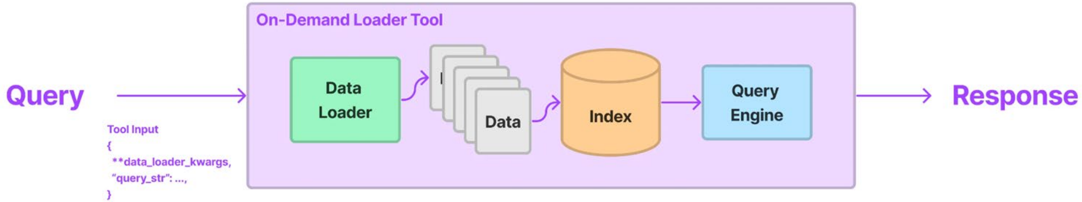
Let's Try It Out!

```
agent.chat_repl()
```

```
===== Entering Chat REPL =====
Type "exit" to exit.

=== Calling Function ===
Calling function: lyft_10k with args: {
  "input": "What was Lyft's revenue growth in 2021?"
}
Got output:
Lyft's revenue grew by 36% in 2021 compared to the prior year.
========================
=== Calling Function ===
Calling function: uber_10k with args: {
  "input": "What was Uber's revenue growth in 2021?"
}
Got output:
Uber's revenue growth in 2021 was 57%.
========================
Assistant: In 2021, Lyft's revenue grew by 36% compared to the previous year, while Uber's revenue growth was highe
r at 57%. This indicates that Uber experienced a faster rate of revenue growth than Lyft in 2021.
```

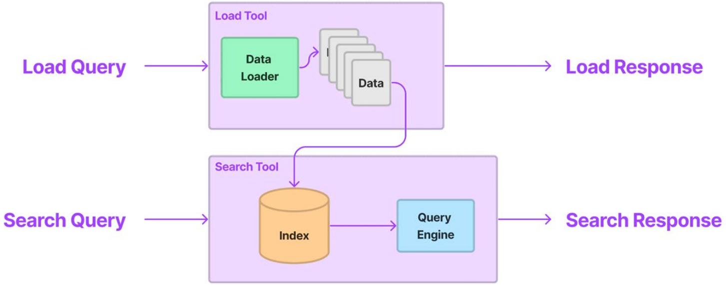
How to handle large responses from tools?




```python
from llama_hub.wikipedia.base import WikipediaReader
from llama_index.tools.on_demand_loader_tool import OnDemandLoaderTool

tool = OnDemandLoaderTool.from_defaults(
        reader,
        name="Wikipedia Tool",
        description="A tool for loading data and querying articles from Wikipedi
)
```

[OnDemandLoaderTool]()




```python
from llama_hub.tools.wikipedia.base import WikipediaToolSpec
from llama_index.tools.tool_spec.load_and_search import LoadAndSearchToolSpec

wiki_spec = WikipediaToolSpec()
# Get the search wikipedia tool
tool = wiki_spec.to_tool_list()[1]

# Create the Agent with load/search tools
agent = OpenAIAgent.from_tools(
 LoadAndSearchToolSpec.from_defaults(
    tool
 ).to_tool_list(), verbose=True
)
```

[LoadAndSearchToolSpec]()

How to handle large number of tools?

- Build an index over your tools, and retrieve the most relevant ones to pass to your agent.
- Example Notebook

```python
# define an "object" index over these tools
from llama_index import VectorStoreIndex
from llama_index.objects import ObjectIndex, SimpleToolNodeMapping

tool_mapping = SimpleToolNodeMapping.from_objects(all_tools)
obj_index = ObjectIndex.from_objects(
    all_tools,
    tool_mapping,
    VectorStoreIndex,
)
```

```python
from llama_index.agent import FnRetrieverOpenAIAgent
```

```python
agent = FnRetrieverOpenAIAgent.from_retriever(obj_index.as_retriever(), verbose=True)
```

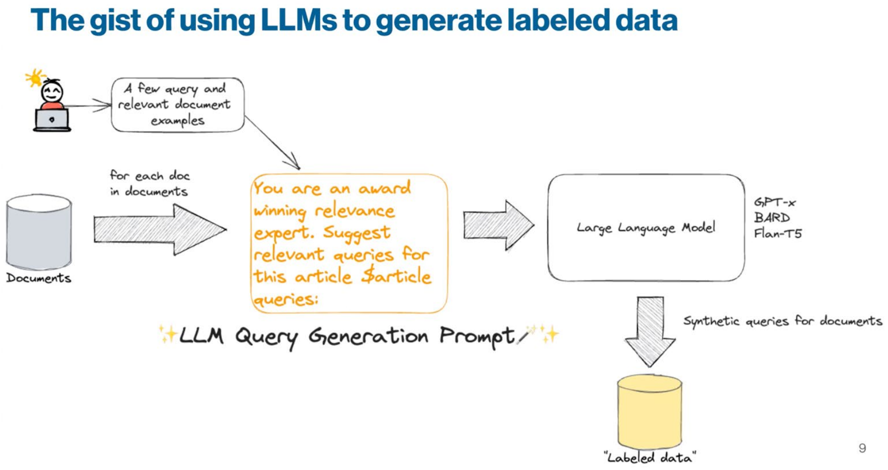
Fine-Tuning

Fine-Tuning: Embeddings

Intuition: Embedding Representations are not optimized over your dataset

Solution: Generate a synthetic query dataset from raw text chunks using LLMs

Use this synthetic dataset to finetune an embedding model.

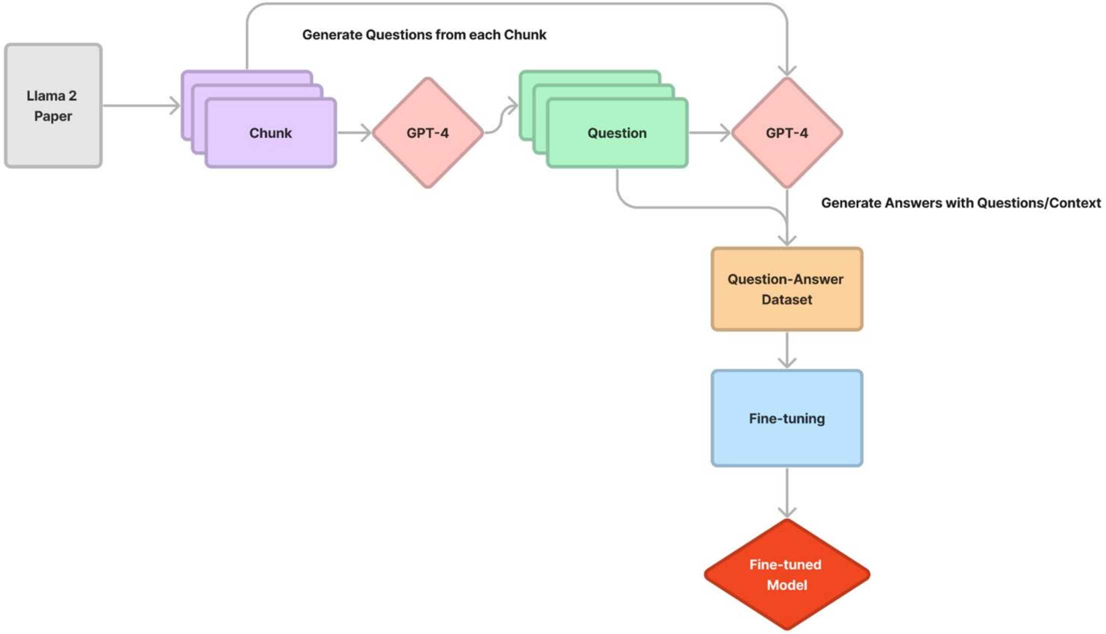


Credits: Jo Bergum, vespa.ai

Fine-Tuning: LLMs

Intuition: Weaker LLMs are not bad at response synthesis, reasoning, structured outputs, etc.

Solution: Generate a synthetic dataset from raw chunks (e.g. using GPT-4). Help fix all of the above!



Production RAG Guide

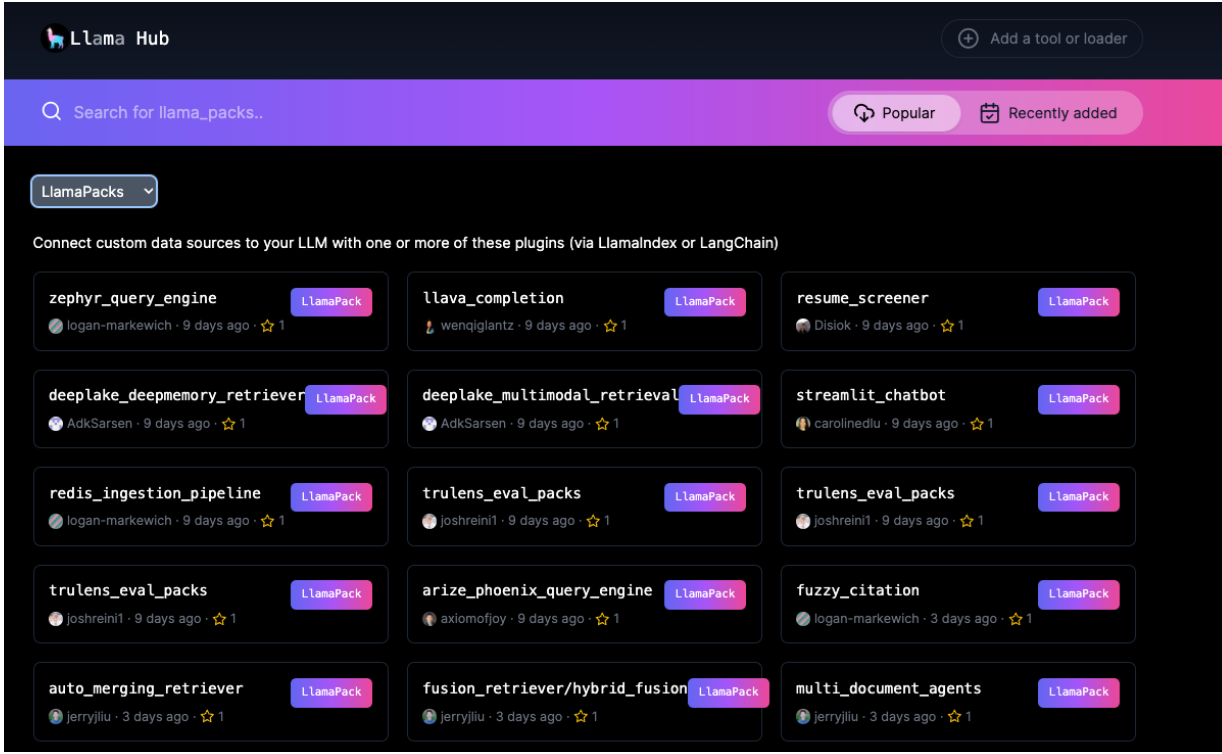
https://gpt-index.readthedocs.io/en/latest/end_to_end_tutorials/dev_practices/production_rag.html

Let’s Put it all Together

Kickstart your LLM app

Llama Packs 🦙📦

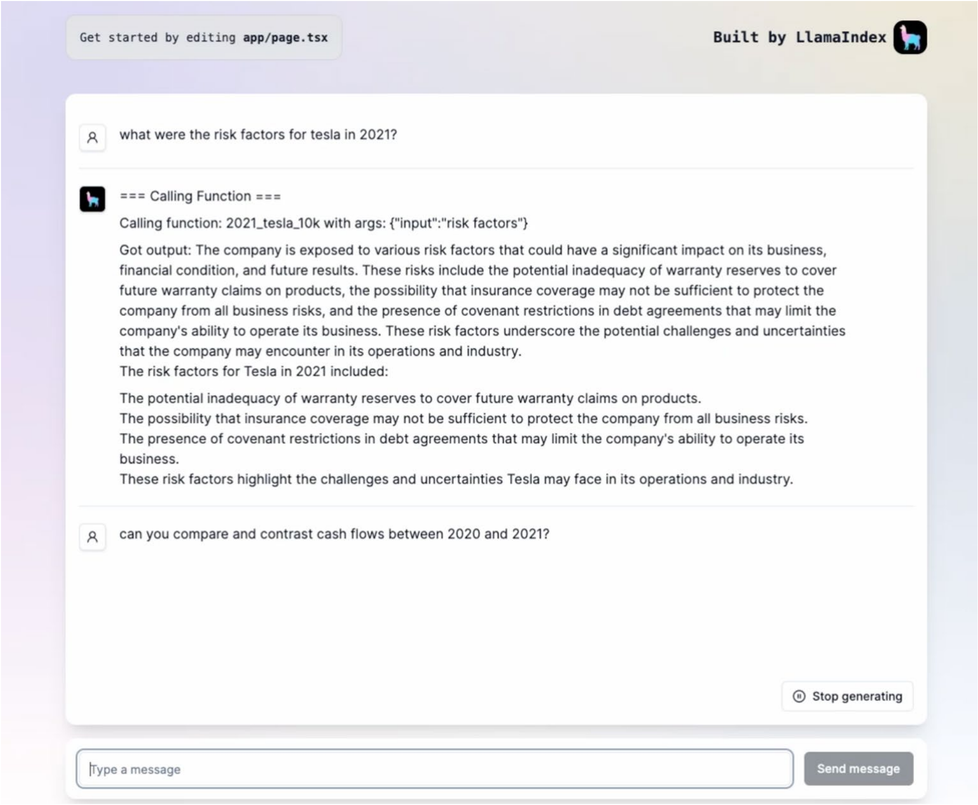
- A community-driven hub of prepackaged modules
- 25+ diverse packs to get started
- Use it out of the box, or inspect the code



Build a Full-Stack Application

create-llama

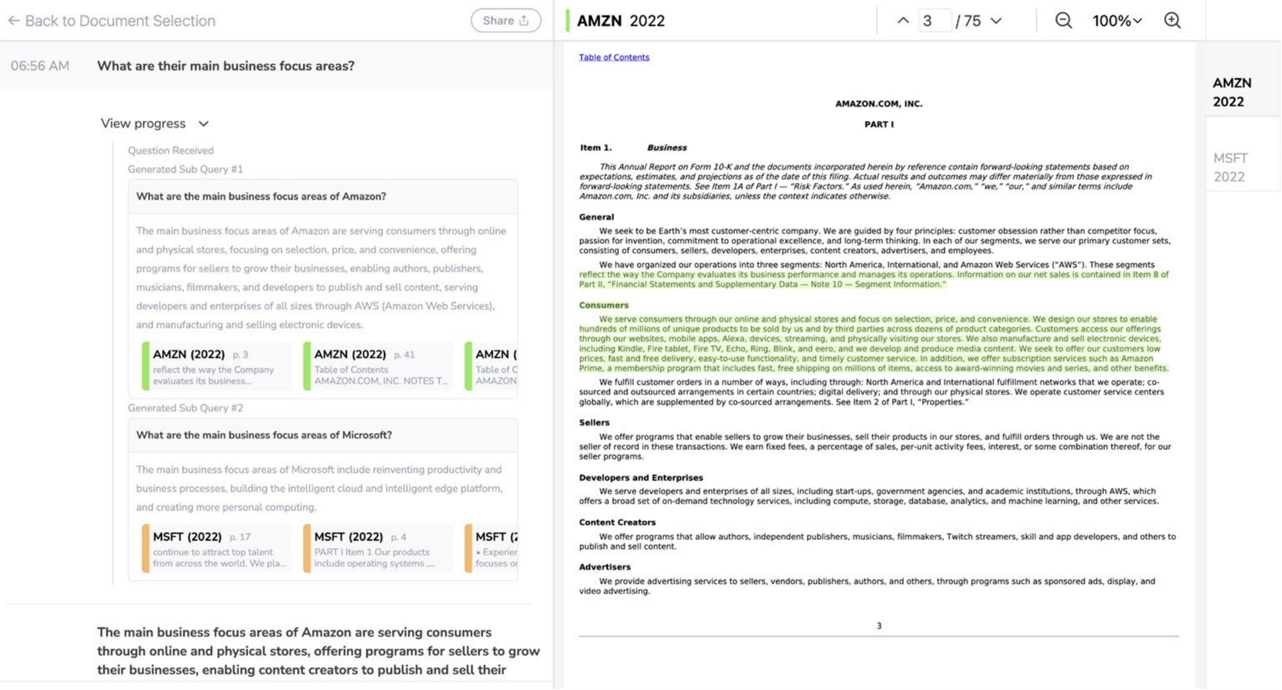
- `create-react-app` but for AI engineers
- Scaffold a full-stack template (choose 3 different backends) with one line of code



SEC Insights

https://secinsights.ai

Github:
https://github.com/run-llama/sec-insights



Thanks!

Building LLMs in an enterprise setting? We'd love to chat!