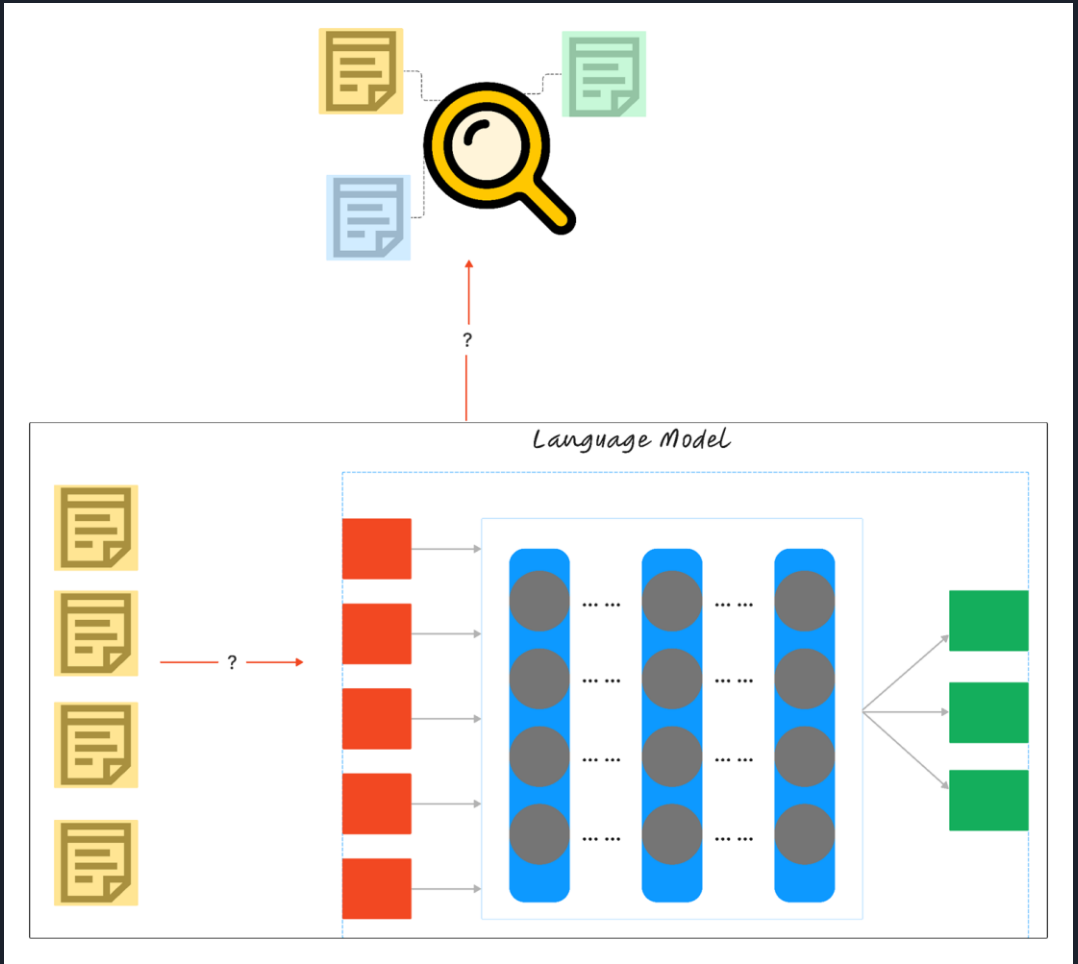
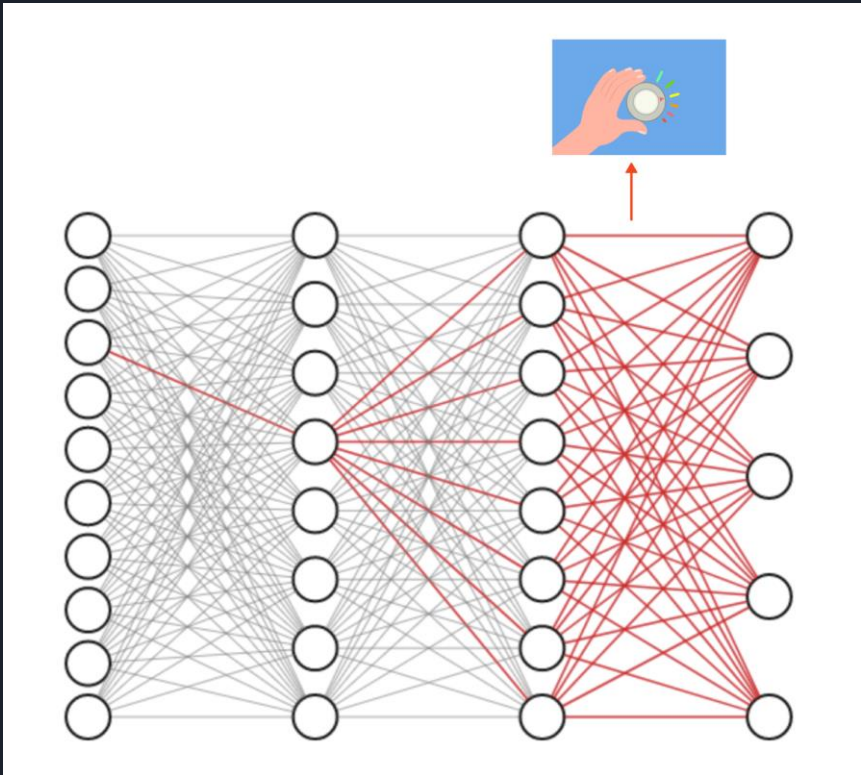
How to ingest and query your own data to LLMs using LlamaIndex

Problem Statement



Fine-tuning



Fine-tuning

1. Time-consuming
2. Resource intensive & computationally costly
3. Opaque Transformations

Prompting

Original Input $\boldsymbol{x}_{\text{inp}}$

a real joy.

Trigger Tokens $\boldsymbol{x}_{\text{trig}}$

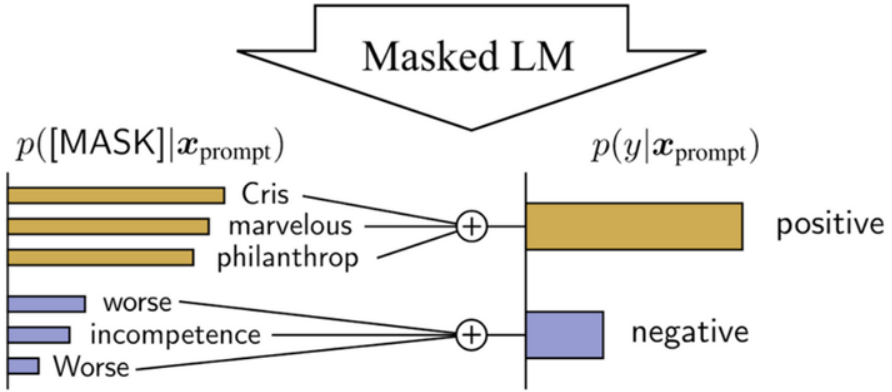
atmosphere, alot, dialogue, Clone...

Template $\lambda(\boldsymbol{x}_{\text{inp}}, \boldsymbol{x}_{\text{trig}})$

{sentence}[T][T][T][T][T][P].

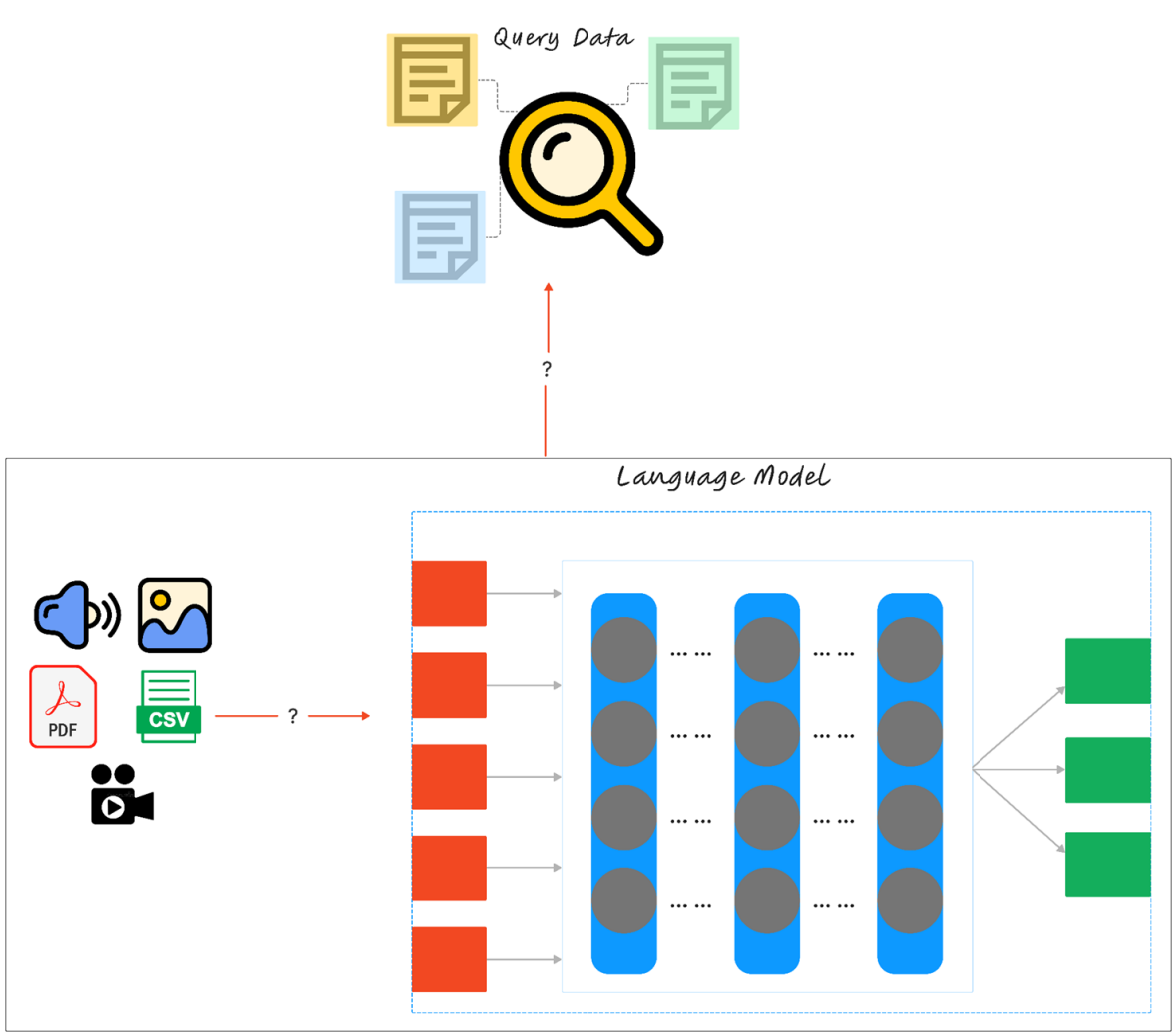
AUTOPROMPT $\boldsymbol{x}_{\text{prompt}}$

a real joy. atmosphere alot dialogue Clone totally [MASK].



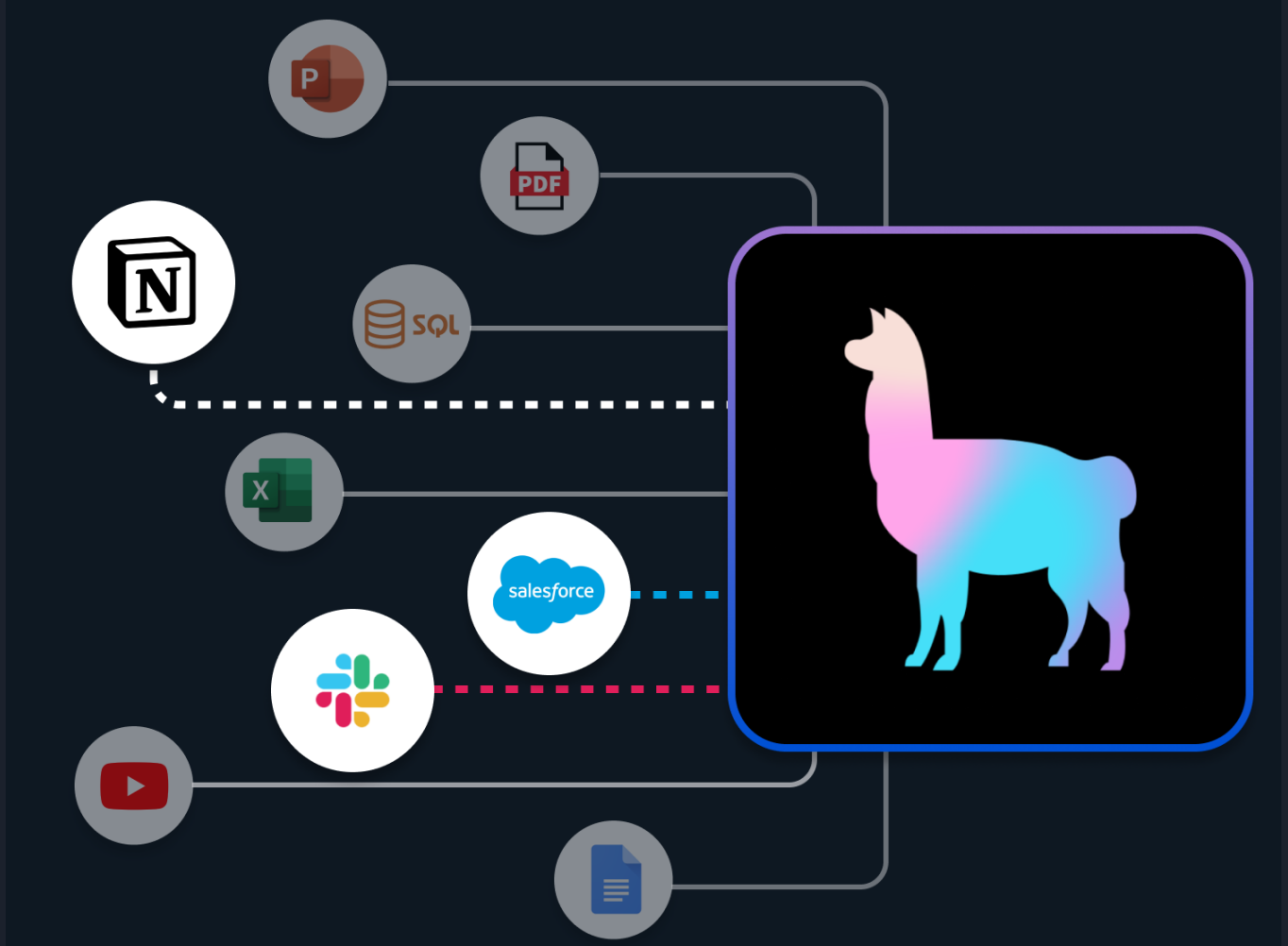
Prompting

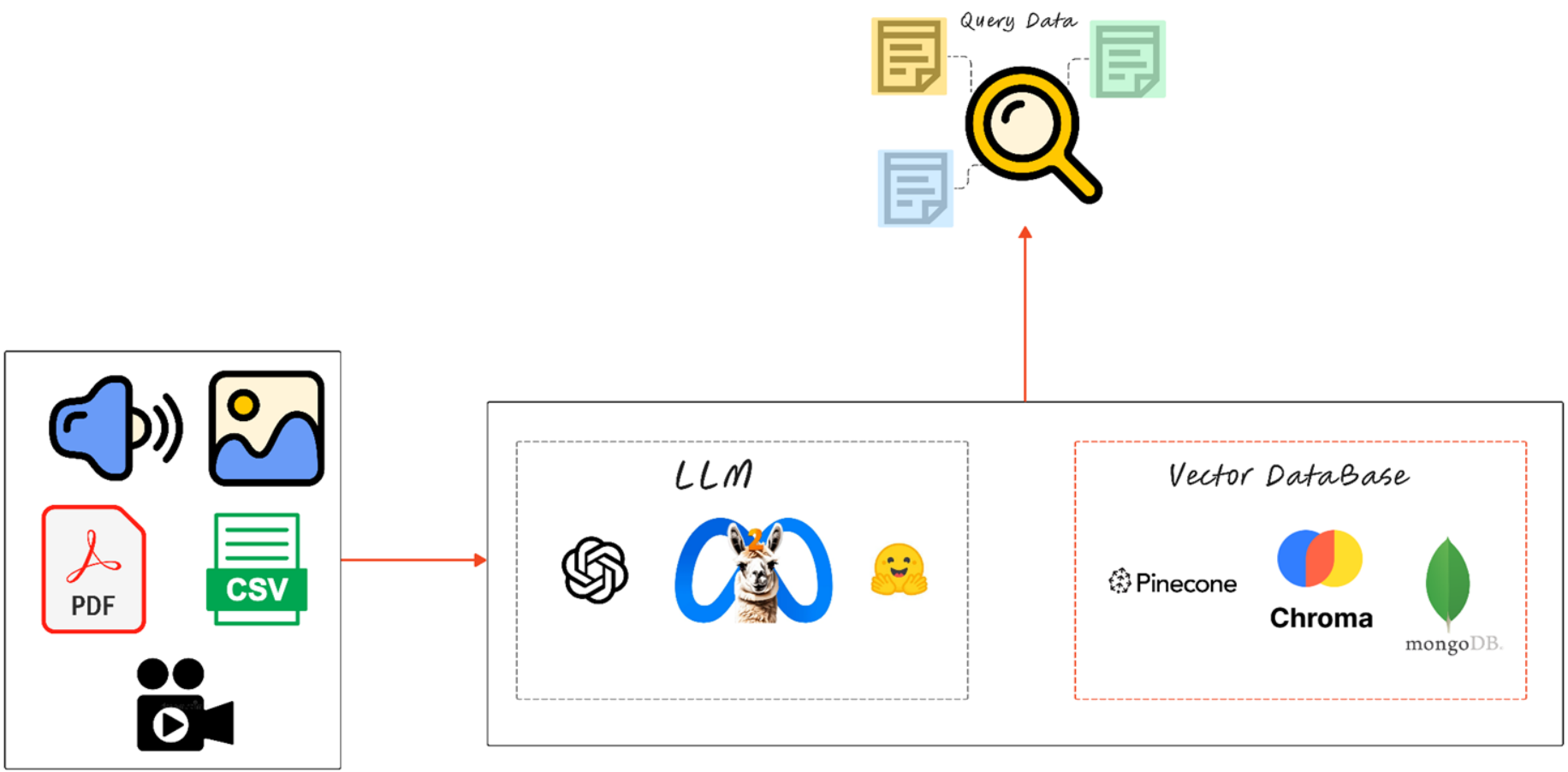
1. Limited Token Limit
2. Less Fine-Grained Control
3. Overhead for Users:
    a. Users might need to experiment with various prompts to get the desired output, which can be time-consuming and require expertise in framing questions.



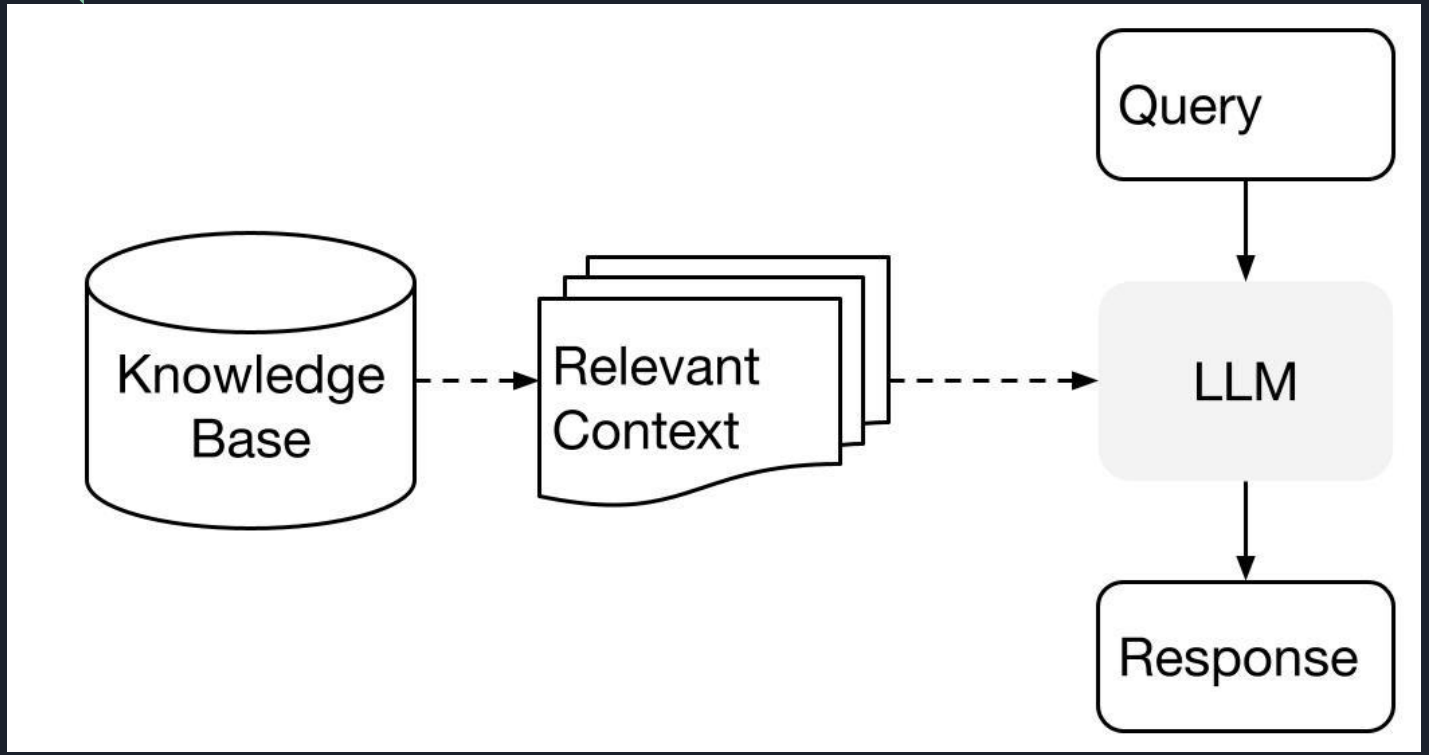
Data can come in different forms!

- MP4
- Audio
- Images
- CSV / tabular
- PDF


PDF
SQL
salesforce


Query Data
PDF
CSV
LLM
Vector DataBase
Pinecone
Chroma
mongoDB.

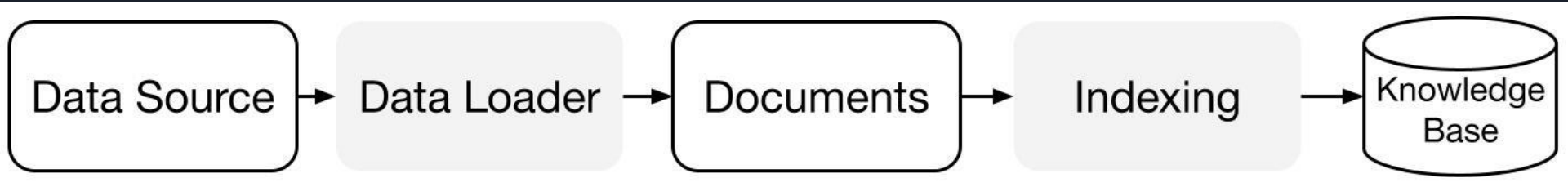
Idea:



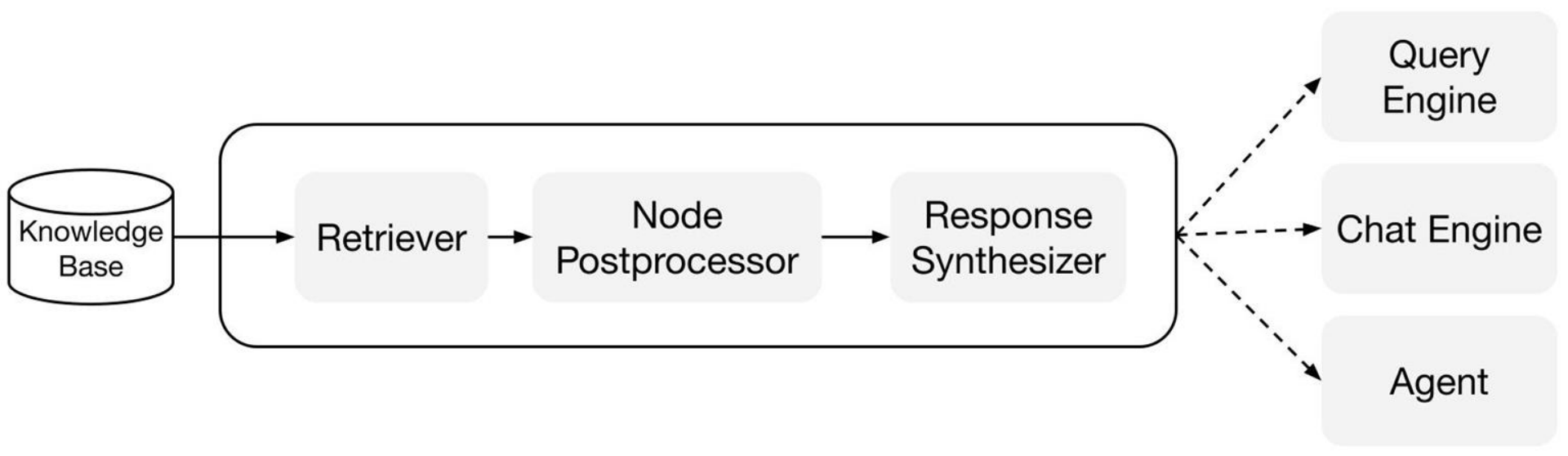
Can be divided into 2 stages:

1. Indexing stage
    a. Indexes your data into the knowledge base
2. Querying stage
    a. Retrieves the data you ask from / wish to obtain from the knowledge base

Indexing Stage



Querying Stage

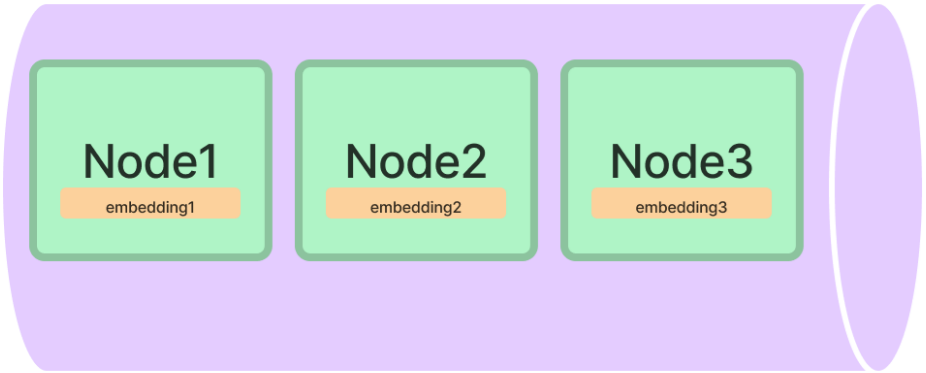


LlamaIndex is NOT:	**LlamaIndex is:**
A new pre-trained Large-language-model	A bridge between your data with any LLM, and/or vector storage
A new vector database storage system	A platform for you to perform queries from your own data that has been ‘ingested’ into the LLM

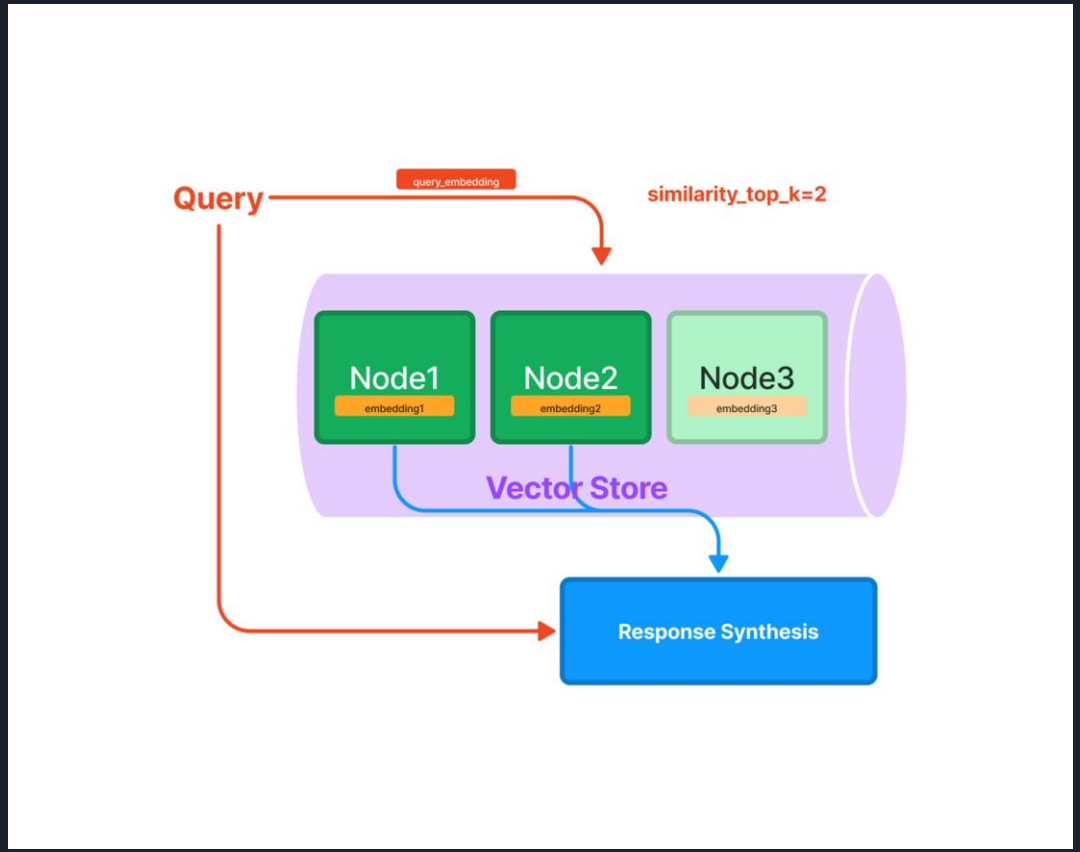
How do they index your data

1. Vector Storage Index
2. List Index
3. Tree Index
4. Keyword Table Index

Vector Store Index



Vector Store Index (during querying)

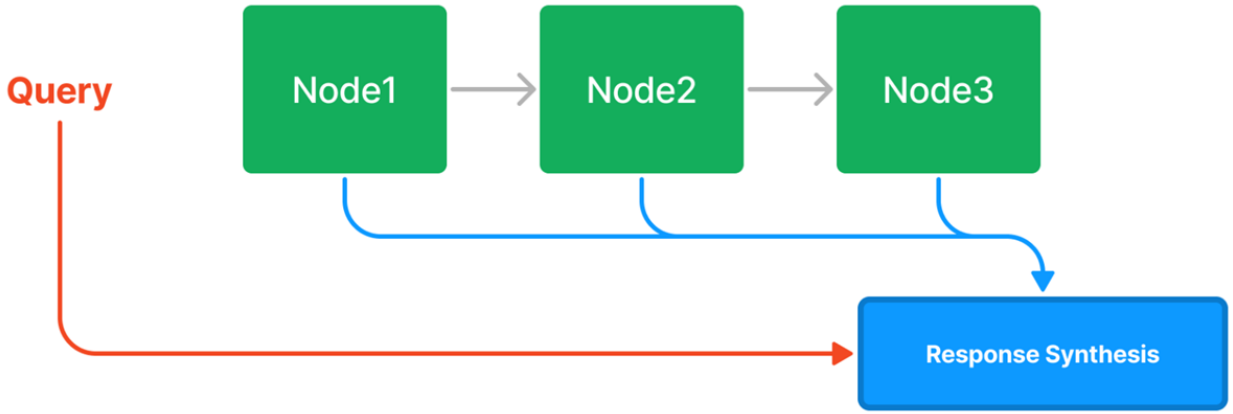


Vector Store Index

Good for	**Bad For**
Your query only requires specific portion of your data	The importance of your data is uniformly distributed
You want fast retrievals	

List Index

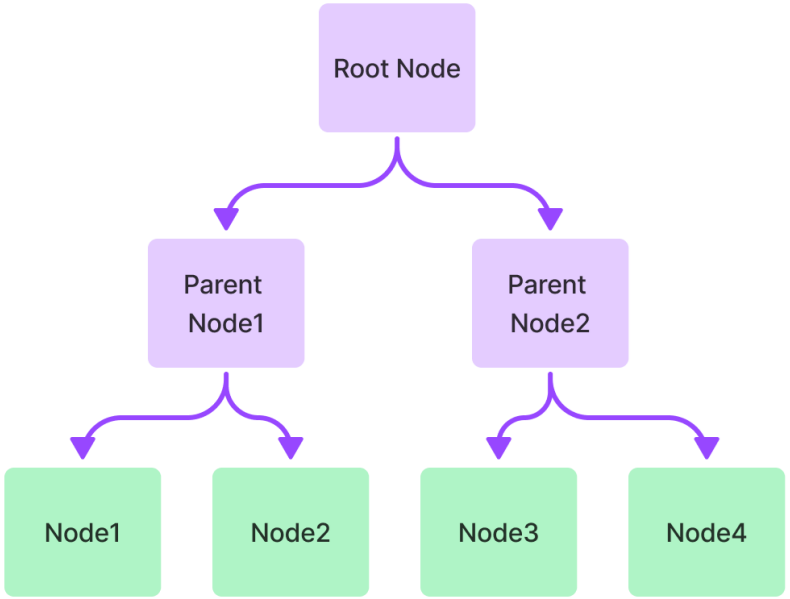
List Index (Querying)



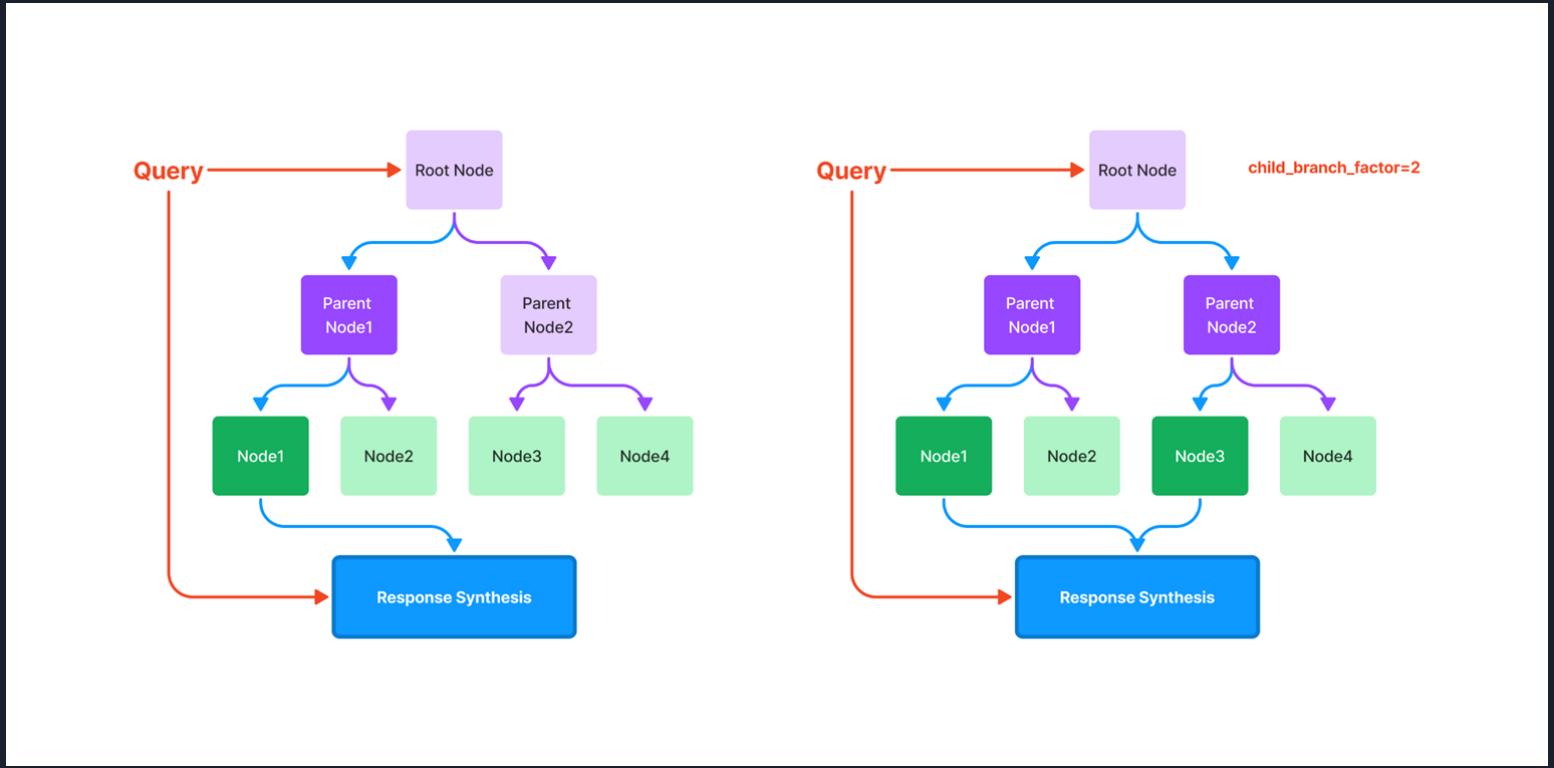
List Index

Good for	**Bad For**
The importance of your data is uniformly distributed	Speed: Slower than vector store index
Considers all nodes	

Tree Index


Root Node
Parent Node1
Parent Node2
Node1
Node2
Node3
Node4

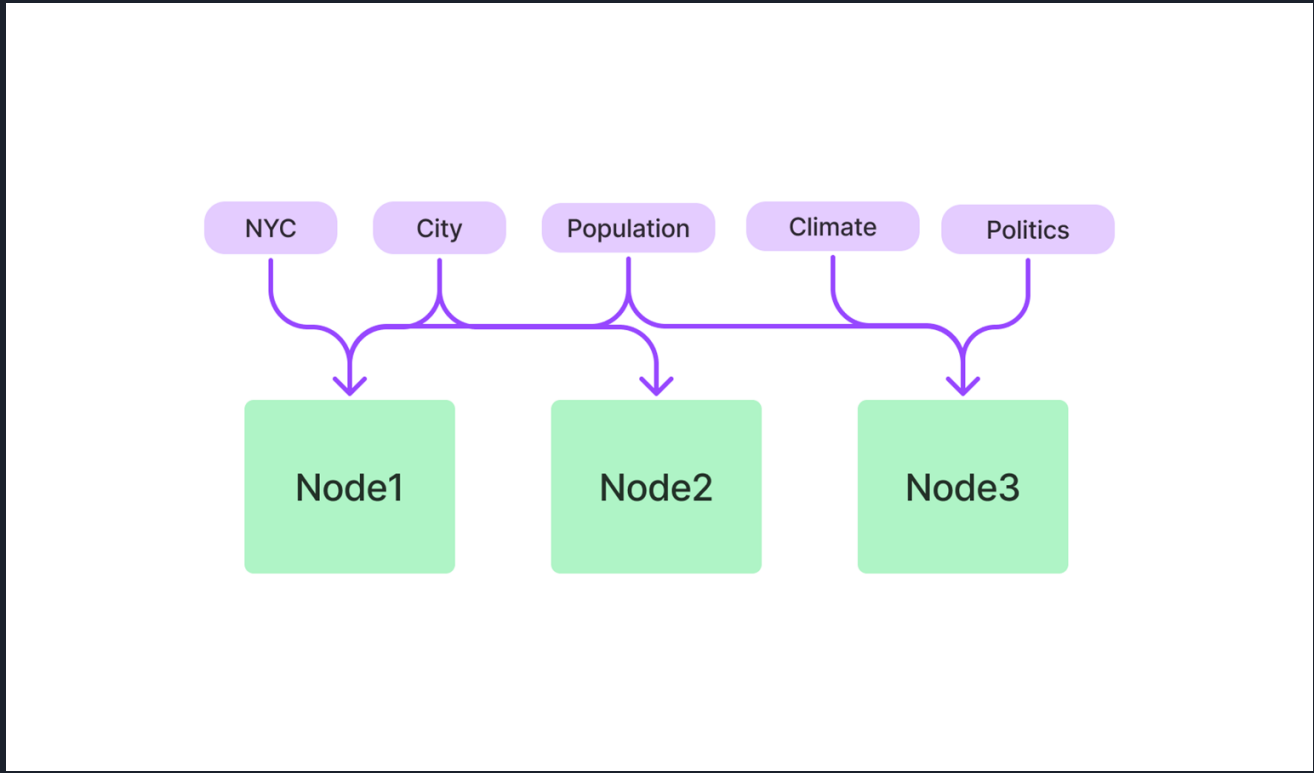
Tree Index (Querying)



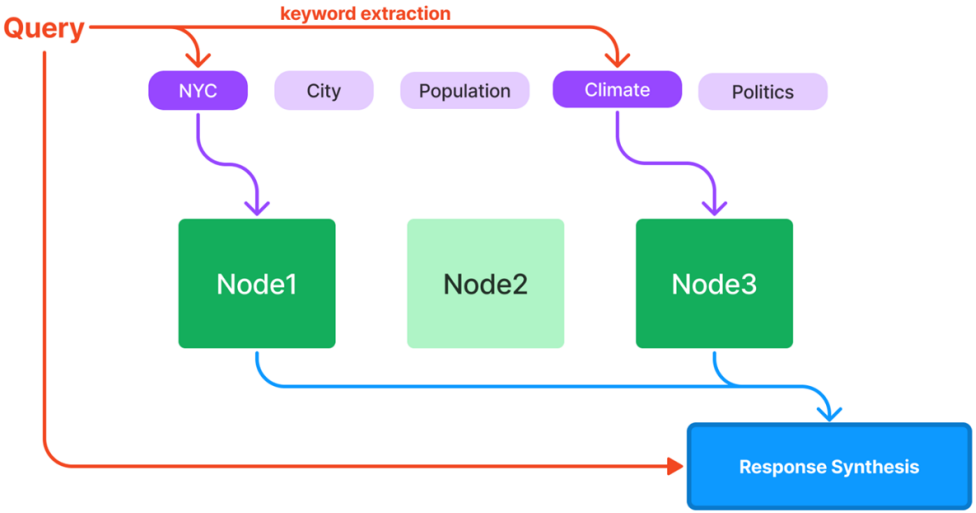
Tree Index

Good for	**Bad For**
Your text is hierarchical in nature	Speed: Slower than vector store index

Keyword Table Index



Keyword Table Index



Keyword Table Index

Good for	**Bad For**
Your data and queries are heavily keyword dependent (eg scientific literature)	Data in which keywords can be used across different contexts
Speed	If the importance of your information is uniformly distributed across your data

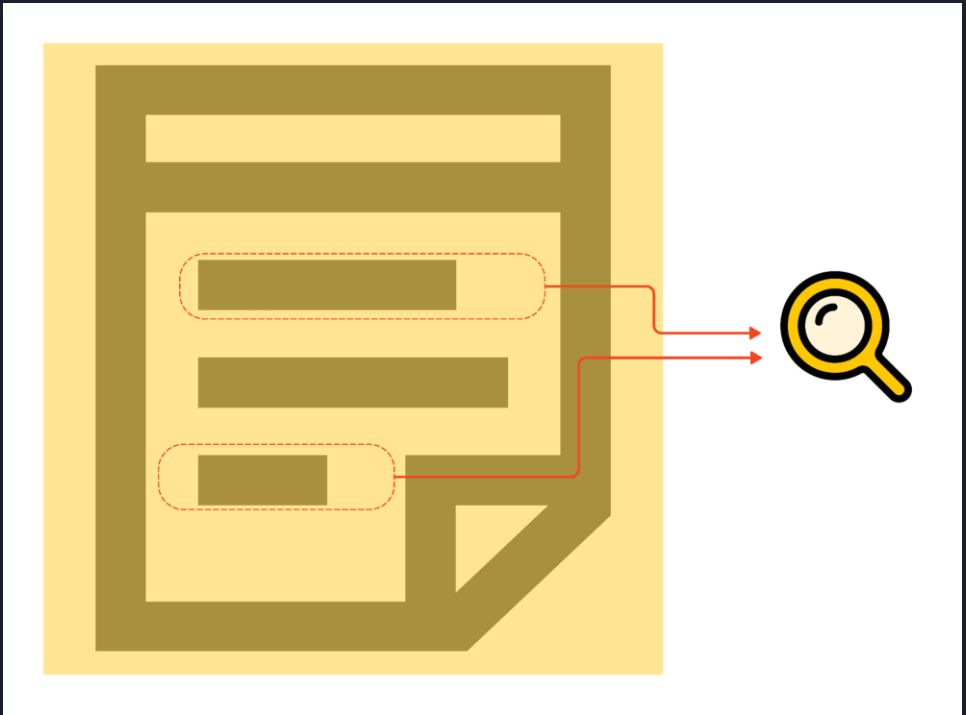
Use Examples + demos

1. Semantic Search
2. Summarization
3. Synthesis over heterogeneous data
4. Sub-question Query Engine
5. Join Q&A Summary Query Engine
6. Multi-step Query engine
7. Compare & Contrast
8. Querying over structured data

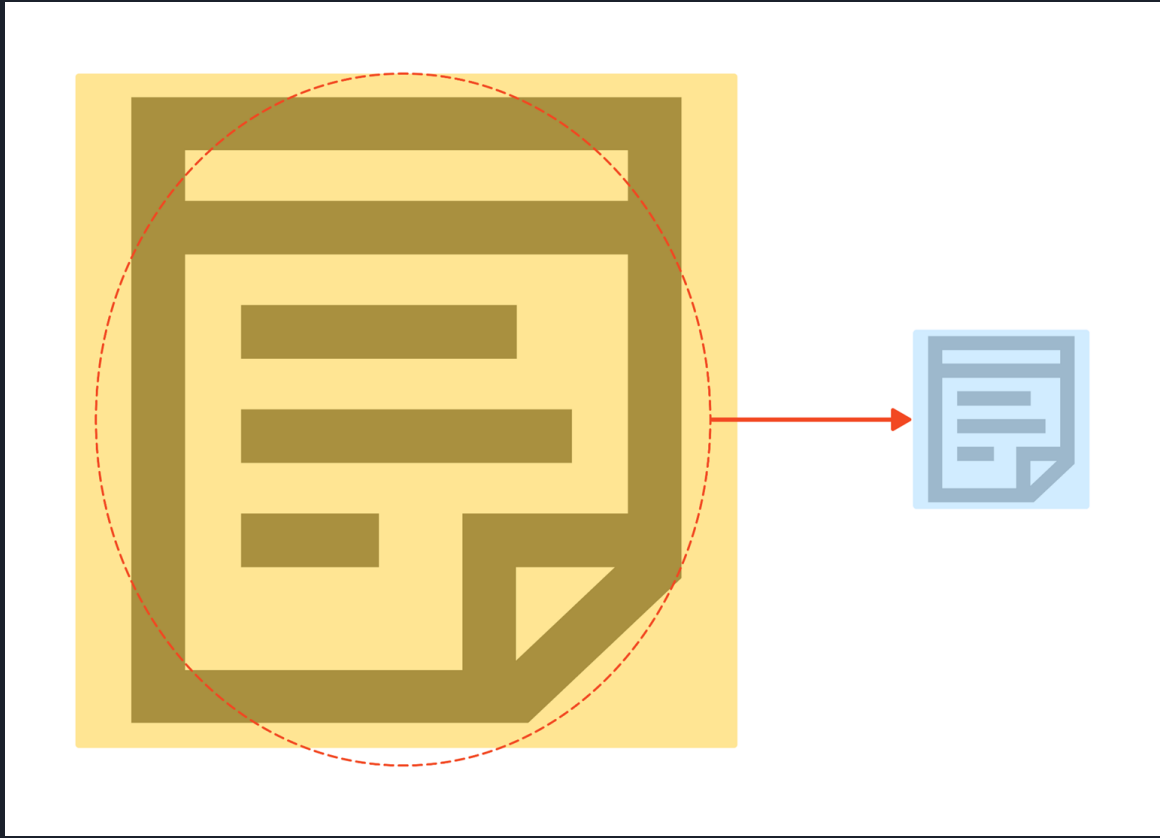
If you are lost, you can always reference the code at my github

https://github.com/cja5553/llama_index_demo

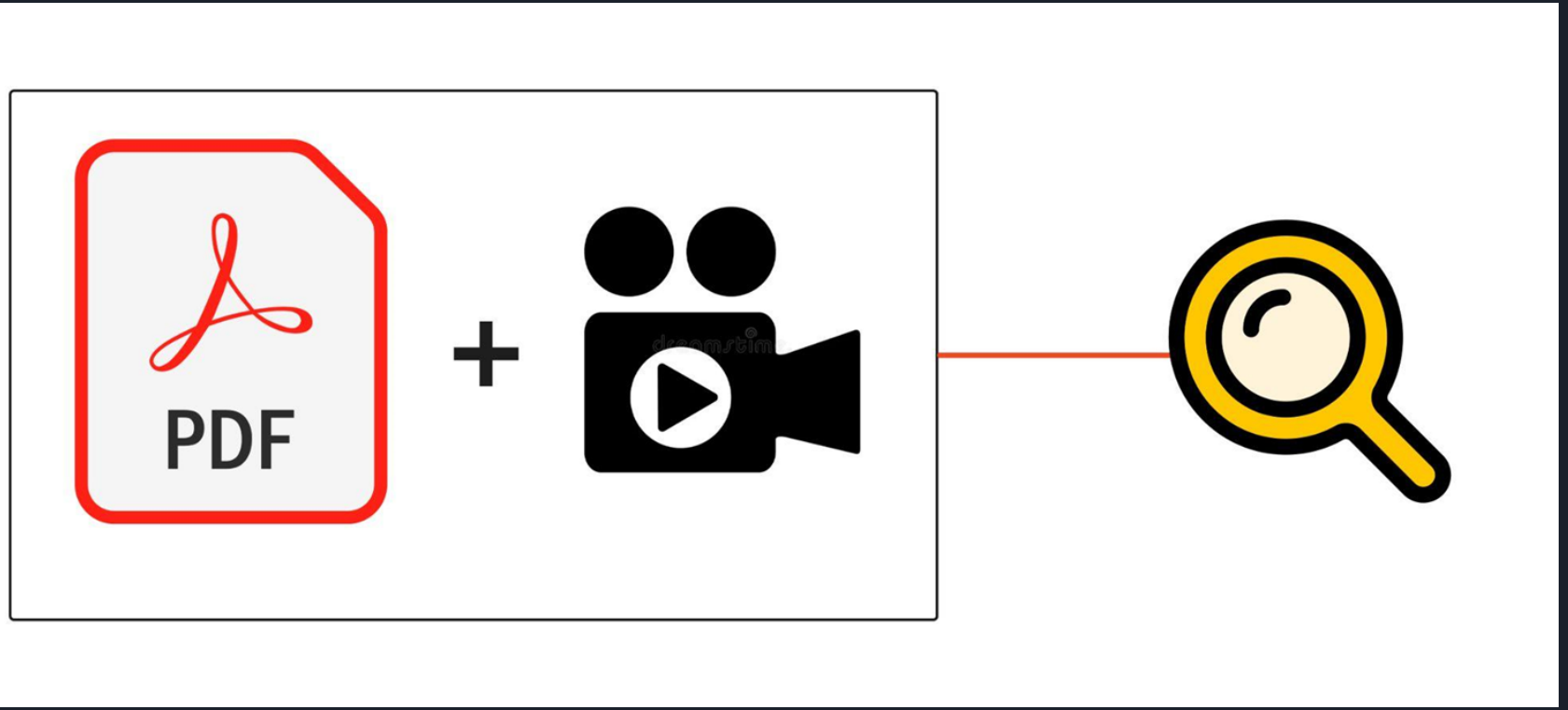
Semantic Search



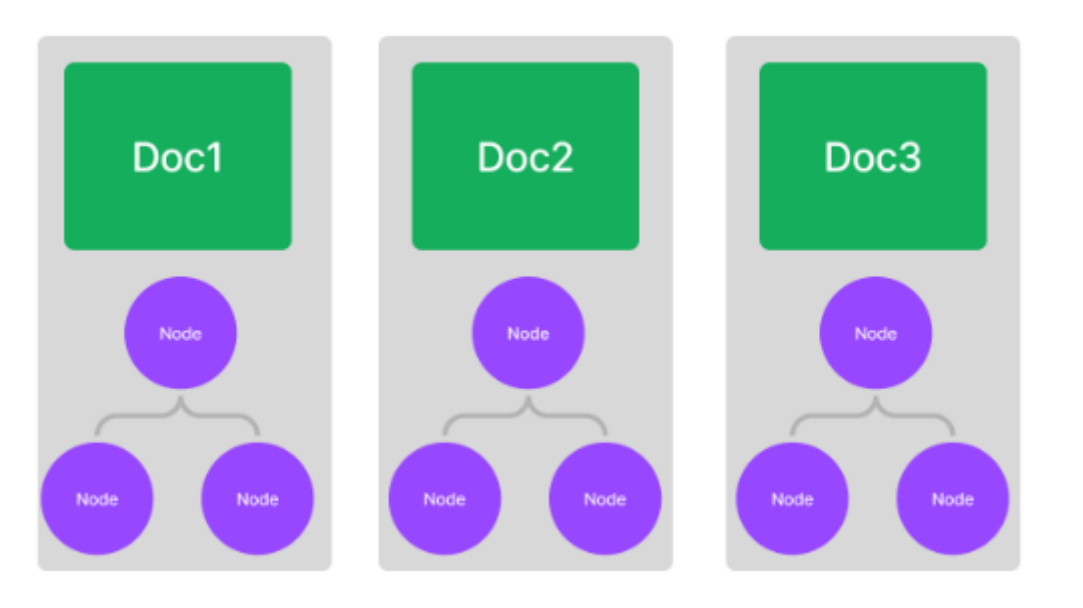
Summary



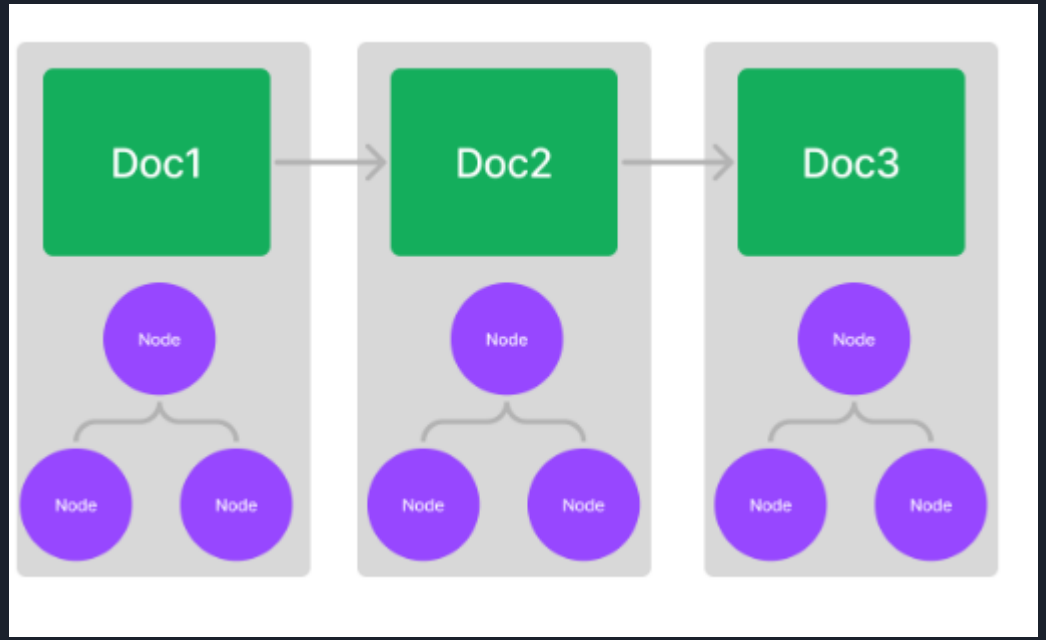
Synthesis over heterogeneous data



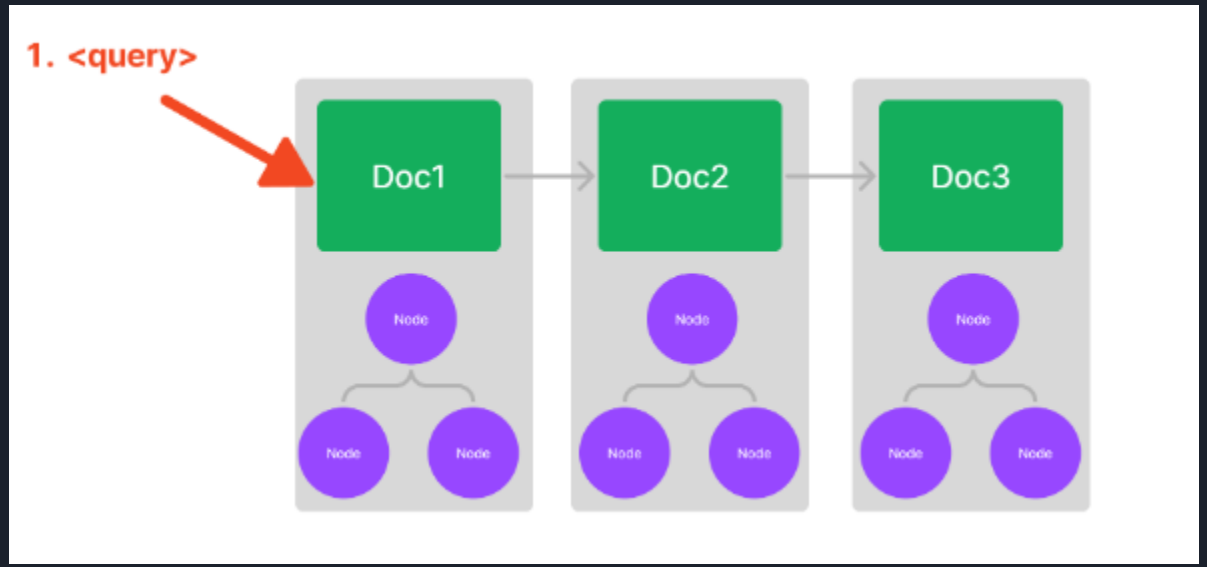
Synthesis over heterogeneous data - composability



Synthesis over heterogeneous data - composability

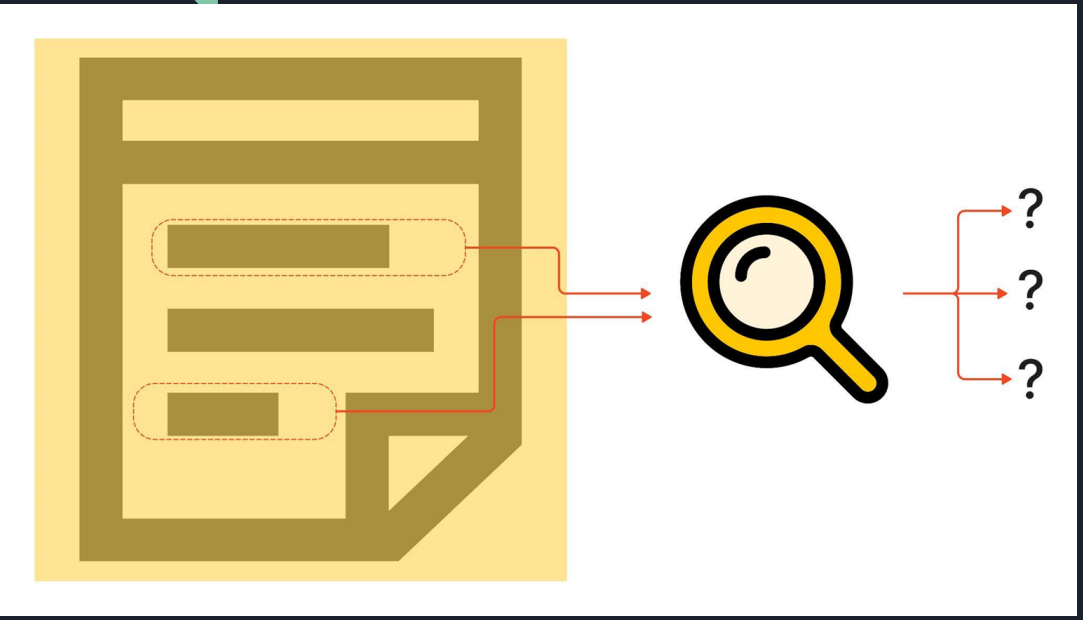


Synthesis over heterogeneous data - composability



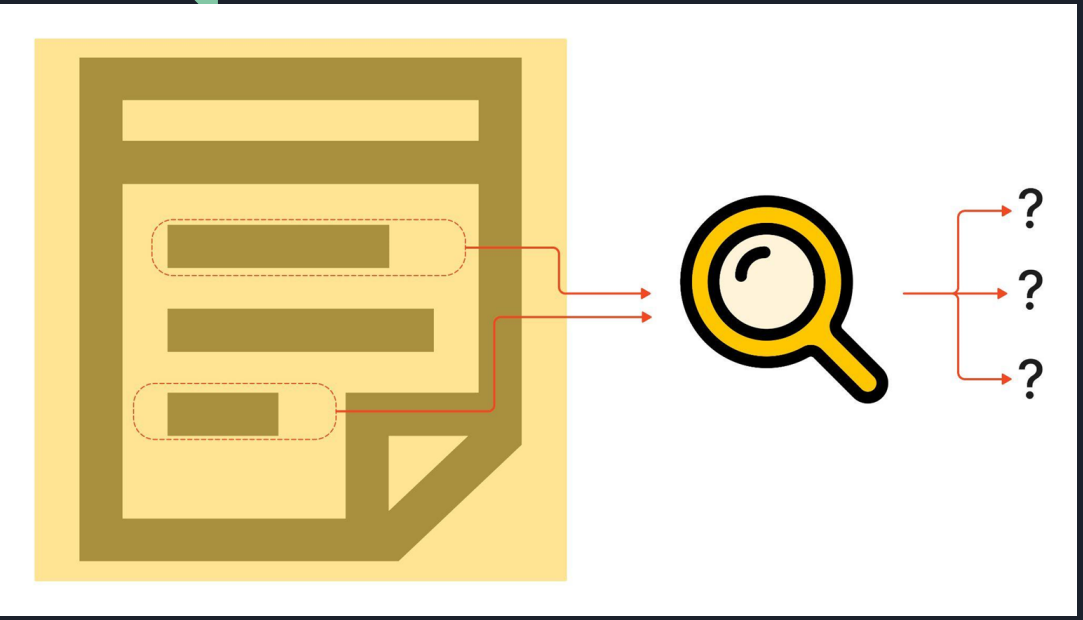
Demo

Sub-question Query Engine



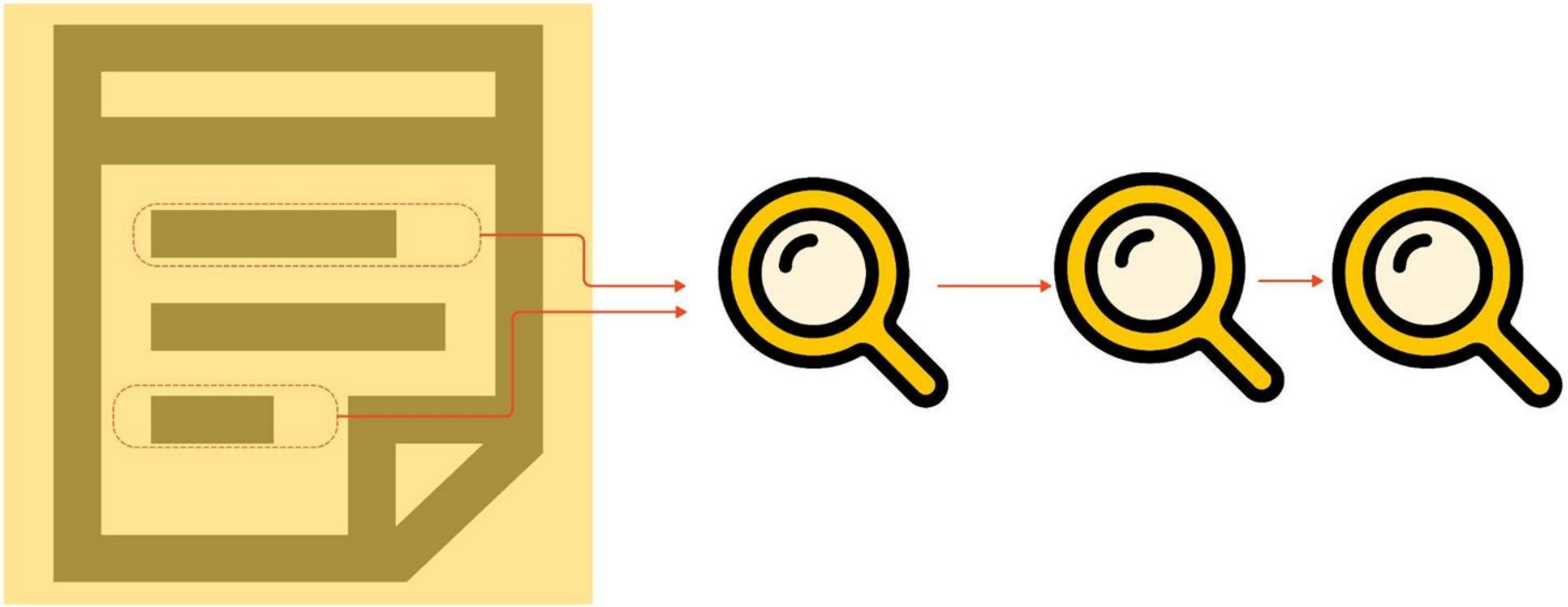
- Breaks down answering complex questions
- including from multiple data sources

Sub-question Query Engine

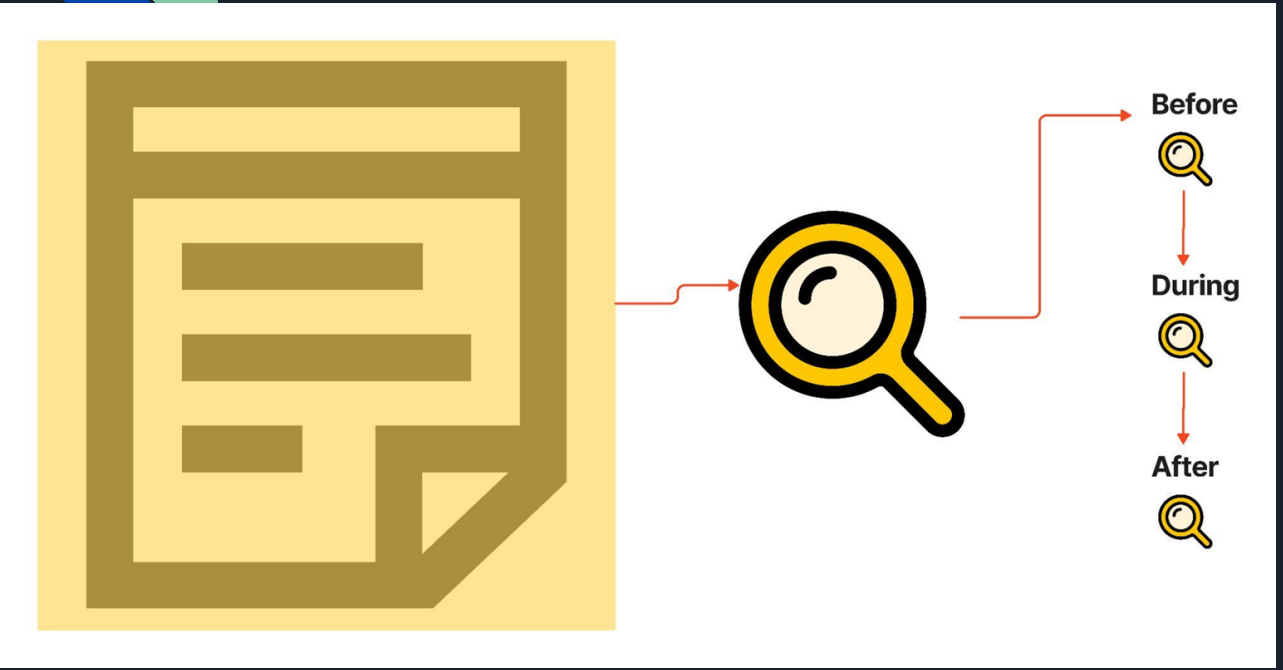


1. first breaks down the complex query into sub questions for each relevant data source
2. gather all the intermediate responses and synthesizes a final response.

Join Q&A Summary Query Engine



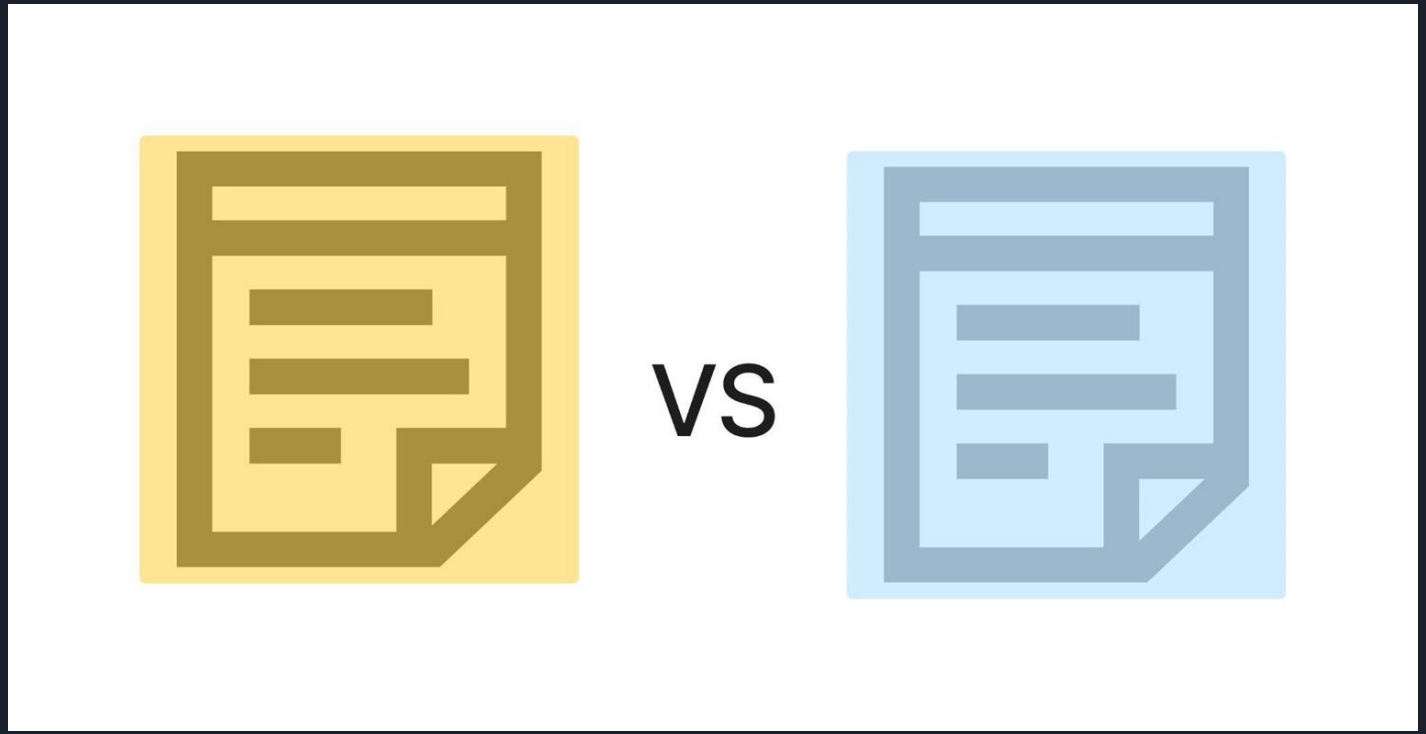
Multi-step Query engine



decompose a complex query into **sequential** subquestions

Demo

Compare & Contrast



Demo

Querying over structured data

Demo