Mastering RAG: Building Production-Ready Retrieval-Augmented Generation with LlamaIndex

Hemamalini Nithyanandam

AGENDA

Mastering RAG:
Building Production-Ready Retrieval-Augmented Generation with LlamaIndex

01 | Building RAG using LlamaIndex

02 | Core LlamaIndex Components

03 | Evaluating RAG effectiveness

04 | Customization Techniques

05 | Advanced Retrievers & Optimized Query Engines

06 | Real time Use cases

07 | Beyond Retrieval: The Evolution Continues

Who Am I ....

Professional

- Software Designer in HP
- 12+ years IT experience
- AWS Solutions Architect
- Data Engineer
  &
- AI Enthusiast

Personal

- Punnagai Foundation
- Certified Yoga Trainer
- Blogger

https://www.linkedin.com/in/hemamalini-nithyanandam/

Lost in Data?

2 AM SRE Nightmare ?

- It's 2 AM, and an SRE receives an alert – a major outage!
- Logs are overwhelming, documentation is scattered, and time is ticking.
- The SRE struggles to find relevant troubleshooting steps.

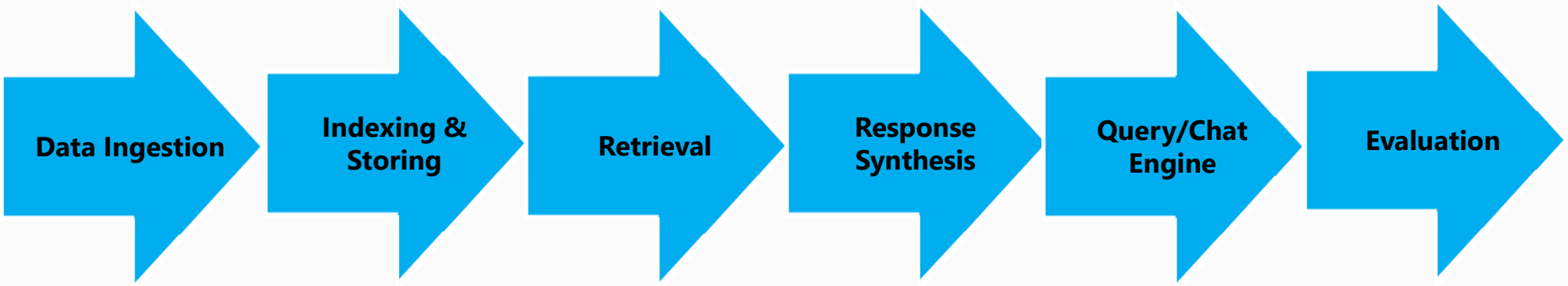
The Hero : RAG to the Rescue!!

Retrieval-Augmented Generation : A Hybrid AI Model

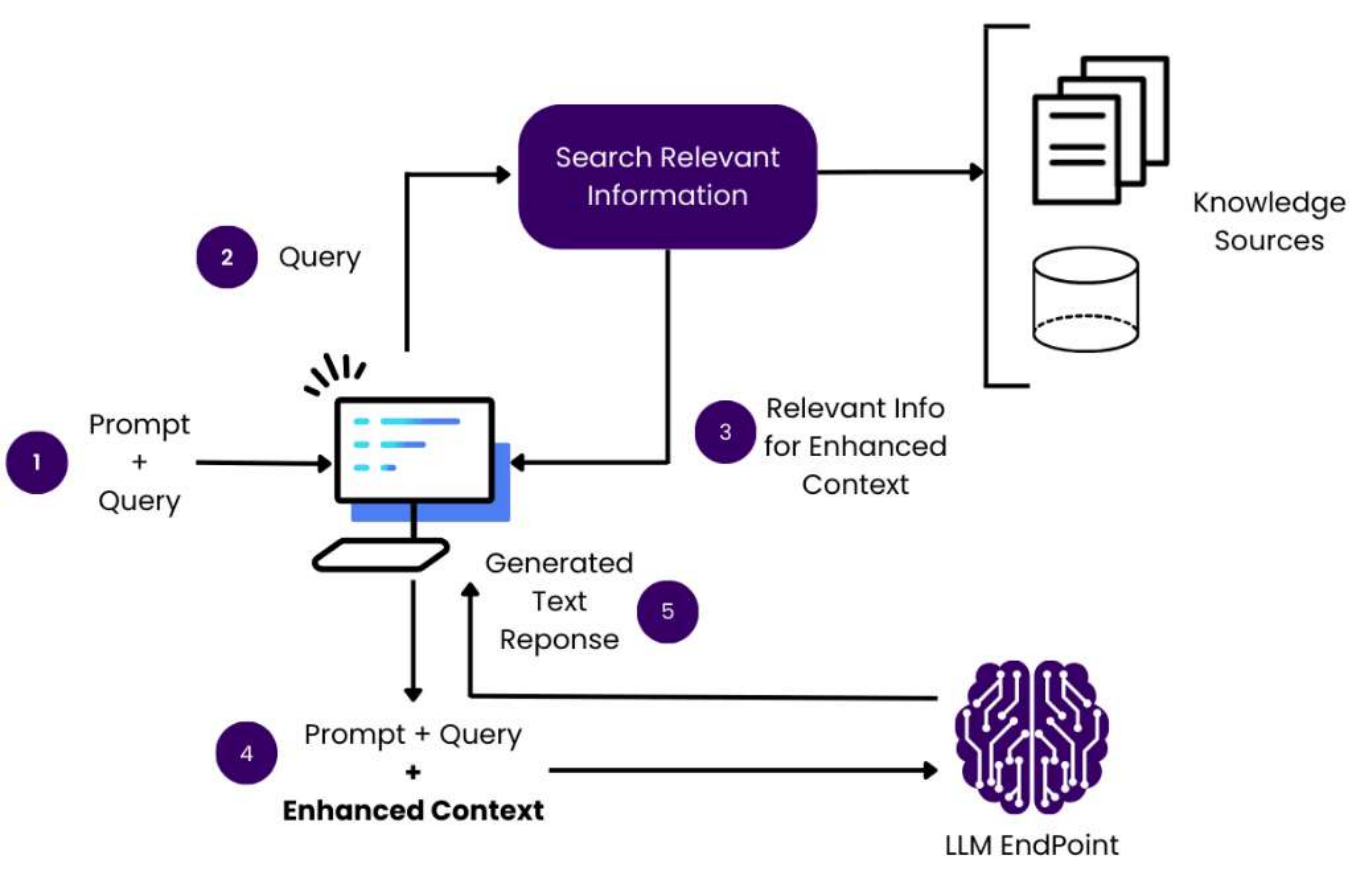
- Combines the power of LLMs with real-time information retrieval.

Two Components:

- **Retrieval**: Dynamically fetches relevant data from external sources (e.g., databases, APIs).
- **Generation**: LLM refines and generates context-aware responses using the retrieved data.



Magic Behind RAG

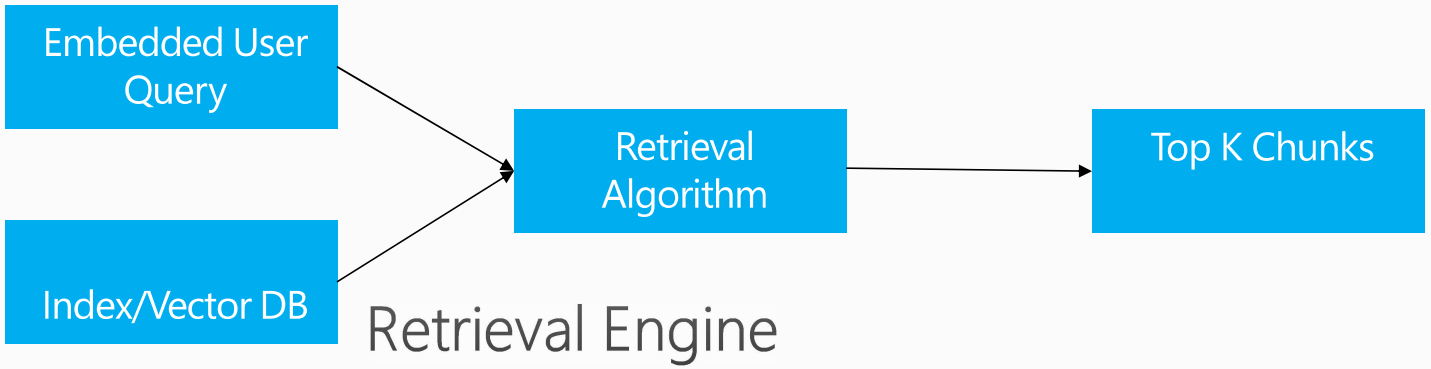


Source: https://www.acorn.io/resources/learning-center/retrieval-augmented-generation/

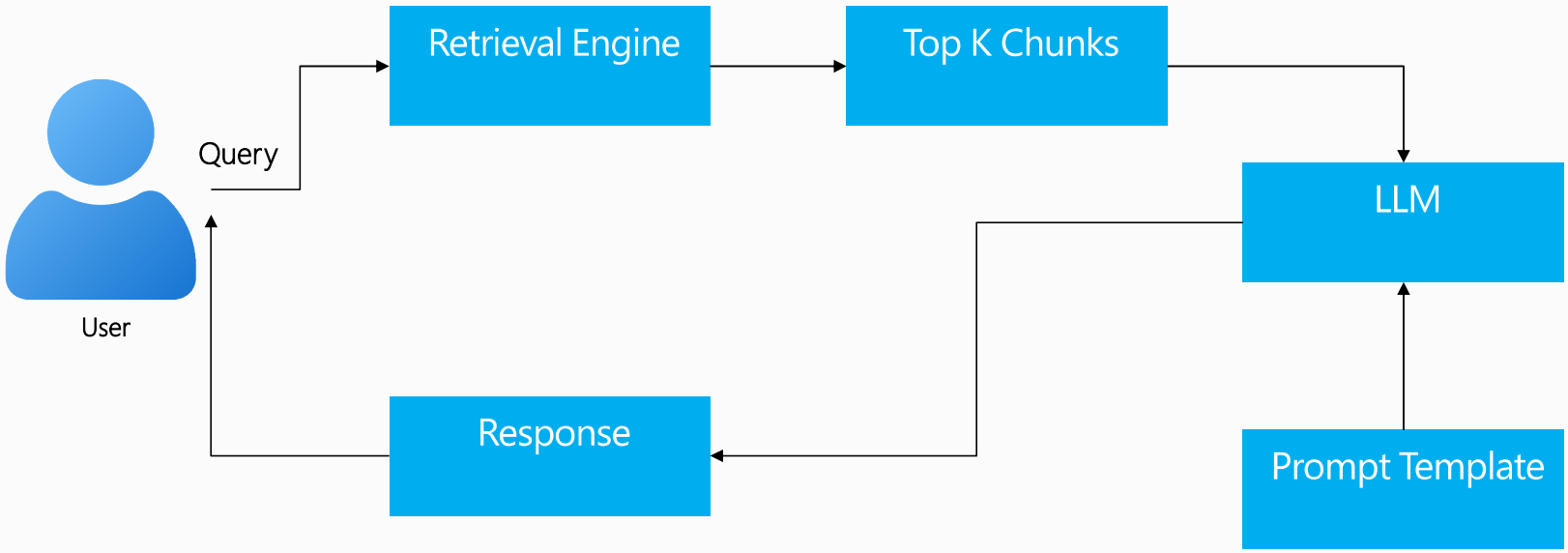
RAG Framework

Embeddings – Numerical vector representations of textual chunks (meaning&context)

- Structured, Unstructured, Programmatic data from different sources like Databases, Document Repo's ,API ingested as chunks converted to embeddings.
- Chunks with embeddings and metadata are indexed for Quick retrieval , to enhance accuracy and build scalable solutions

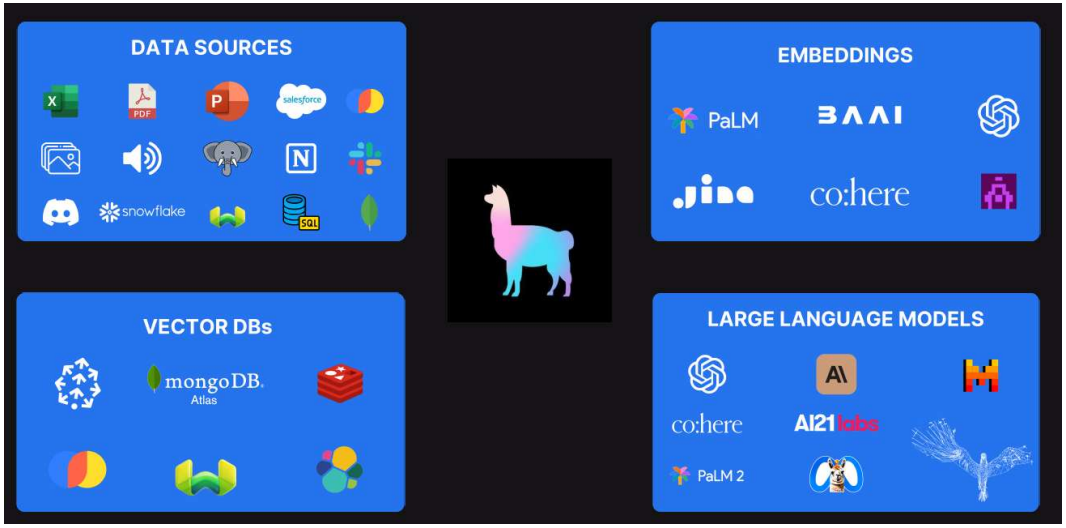


RESPONSE SYNTHESIS



Building RAG using LlamaIndex

- Open source library that facilitates data ingestion, structuring retrieval and integration
- Enables Data connector, supports document operations, utilizes Query routing , enhances doc embeddings



Source: https://www.analyticsvidhya.com/blog/2024/05/build-custom-retriever-using-llamaindex-and-gemini/

LlamaIndex Components

Components of LlamaIndex – Chunking&Tokenization

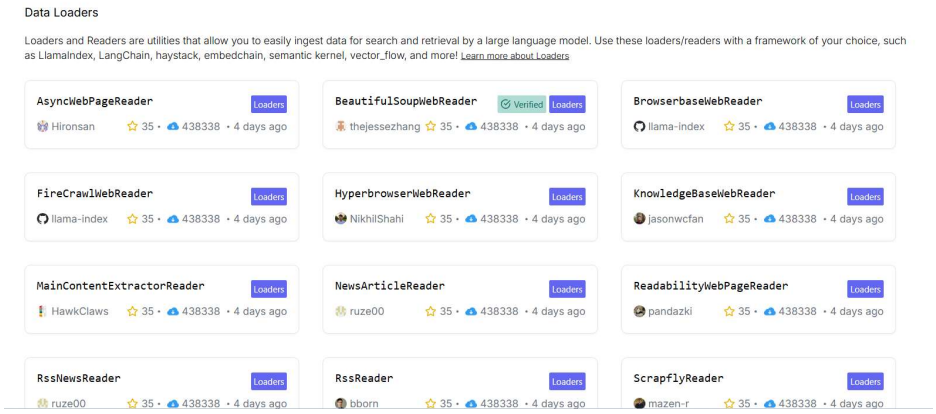
- **Chunking** - breaking text into "Chunks" or smaller segments, which are then vectorized and stored, boosting the efficiency of NLP tasks.
- **Decide Right size for chunk** – Fixed size, Small chunks (more accuracy but high cost), Large chunks (more noise, less accuracy)
- **Tokenization -** dismantling the sentences, paragraphs and articles into smaller chunks
- Sentence Tokenization, Word Phrases (N-Gram Tokenization), Individual Word Tokenization(Uni-Gram Tokenization), Character level tokenization
- **Subword Tokenization in LLM** - breaks the words into known subparts from the dictionary, the model can handle words it hasn't seen before.
- Types – BytePair Encoding (BPE) , WordPiece, SentencePiece

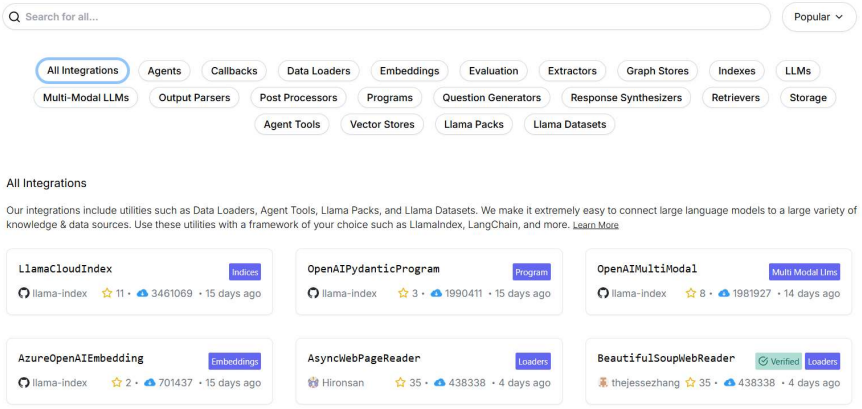
LlamaIndex – Chunking & Tokenization

- **BytePair Encoding (BPE)** - Merges the most frequent pairs of characters or subwords. Widely used in GPT and GPT-2,GPT-3.5 & GPT-4 models
- **WordPiece** - Uses a probabilistic approach to merge subwords. Used in BERT and DistilBERT models
- **SentencePiece -** Uses BPE or unigram models, processing text as Unicodecharacter sequences.
- **https://platform.openai.com/tokenizer**
- **Why is it important to know the total number of tokens in a chunk or document ??**
- **Embedding model -** Important to understand the context limit -Maximum number of tokens that the model can process as input.
- **Text Generation Model -** Models have a token limit for both context window and output responses -(Output) Response Synthesis Limit

LlamaIndex – Data Loaders & Connectors

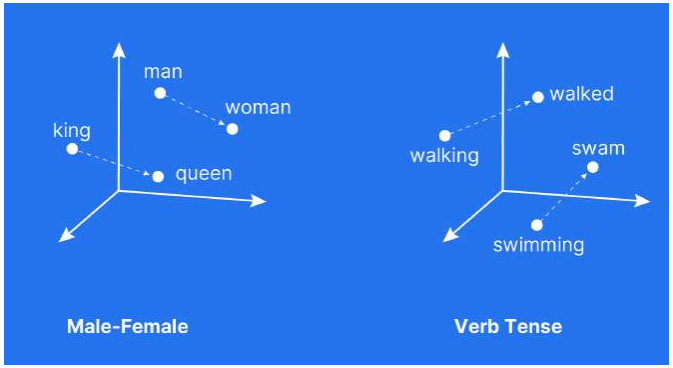
- **Data loaders** – Read and load different types of data sources seamlessly eg SimpleDirectoryReader, CSVReader,etc. **One time Ingestion**
- Data Connectors - Connects to various data sources like SQL databases,NoSQL databases, cloud storage, and APIs. **Realtime , on-demand access**
- https://llamahub.ai





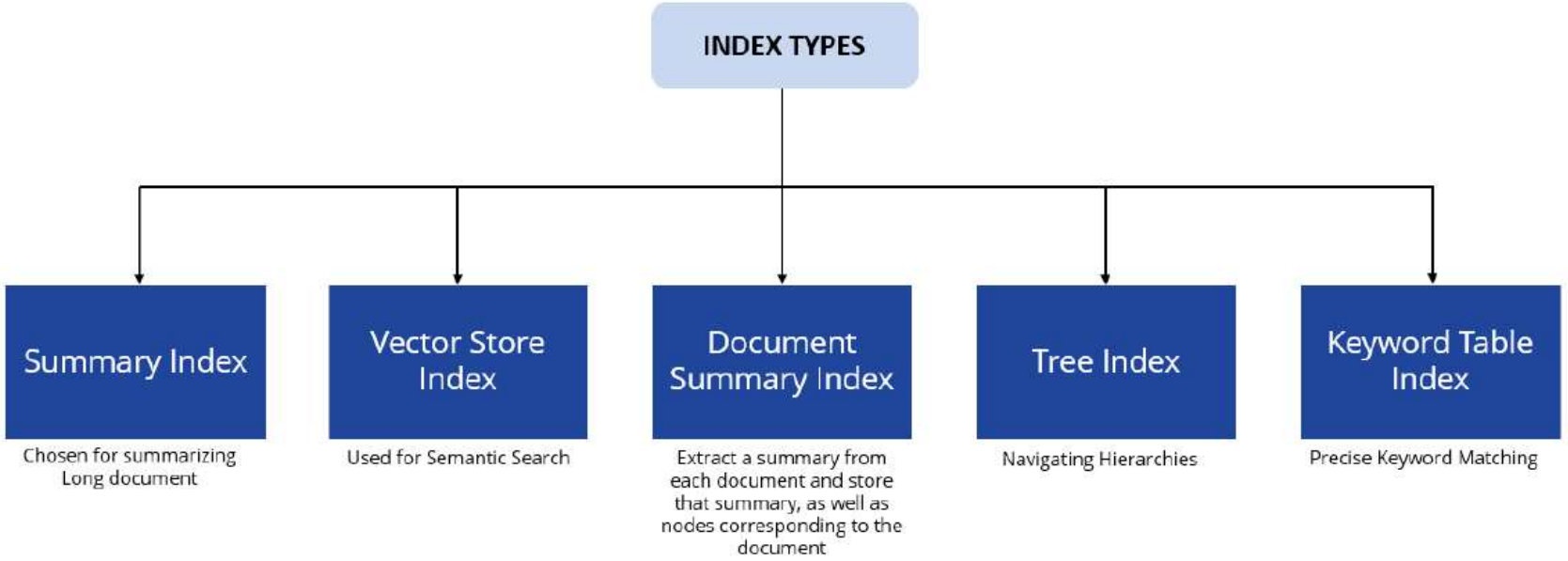
LlamaIndex – Embeddings

- **Numerical vector representations** that captures sematic relationships ,meaning
- **Cosine similarity** is used to determine how similar two vectors are
- **Applications** – Finding most similar word, finding odd one out , Sentence similarity, document clustering
- Look for domain specific embeddings
- **Closed source embeddings** – OpenAI Embeddings, CohereAI, Google Gemini embeddings , JinaAI embeddings,etc
- **Open source embeddings –** BERT, DistilBERT,mpnet,e5,etc



LlamaIndex – Indexing & Retrieval

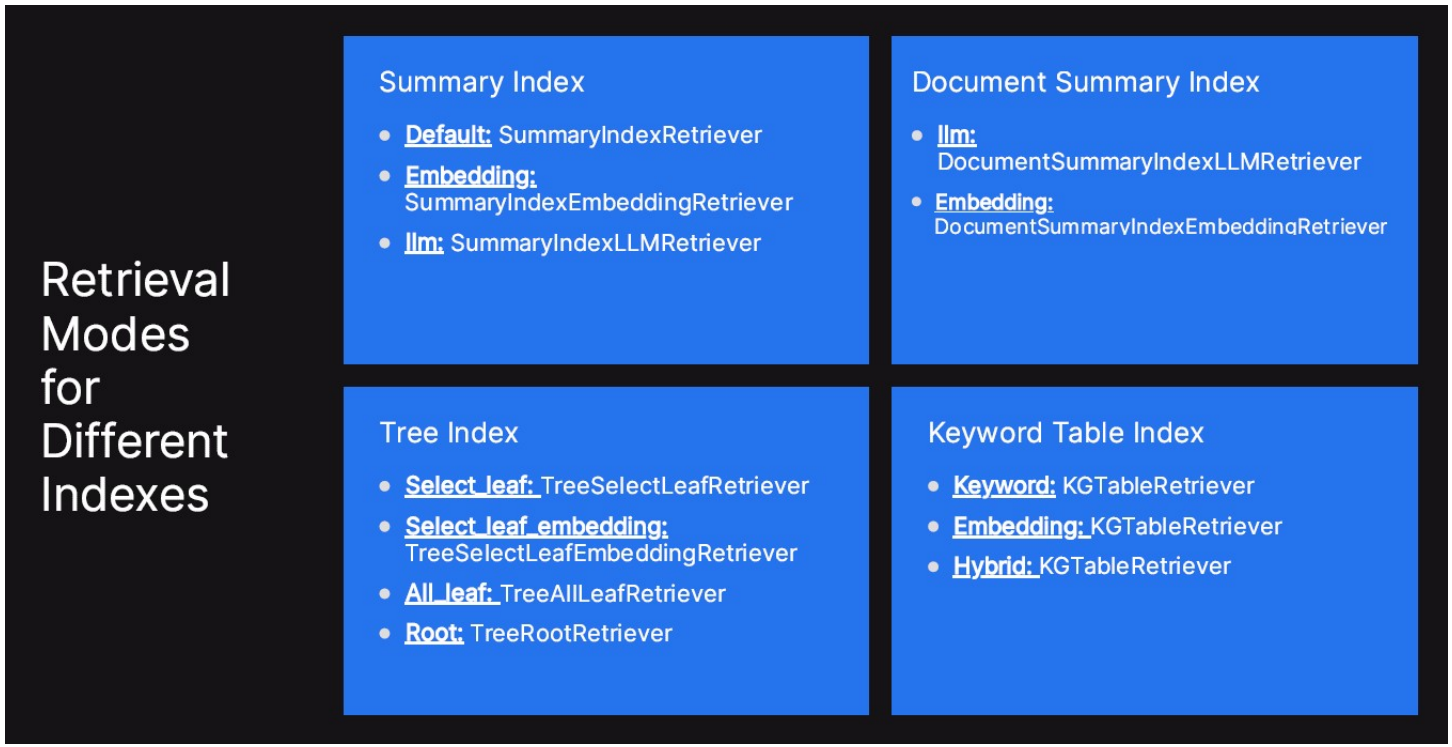
- Structured dataset for quick retrieval, accuracy, scalability, real time performance



Source: https://www.analyticsvidhya.com/blog/2024/05/build-custom-retriever-using-llamaindex-and-gemini/

LlamaIndex – Indexing & Retrieval

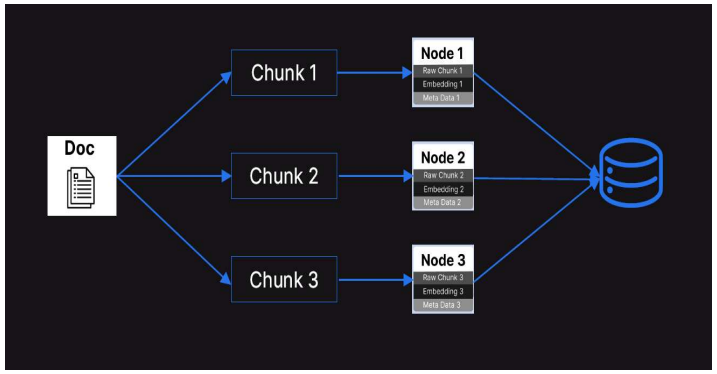
- Retrieval – Top K chunks depends on the specific indexing technique

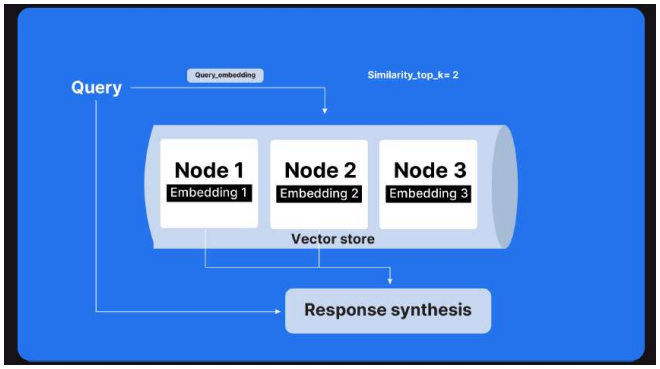


Source: https://www.analyticsvidhya.com/blog/2024/05/build-custom-retriever-using-llamaindex-and-gemini/

LlamaIndex – Indexing & Retrieval

- **1.SummaryIndex** stores all the nodes in the form of the sequence/list in the storage, unlike the vector storage index. Only provides condensed summaries, which might miss detailed information. Less effective for understanding the full context of large documents.
- **2.Vector Store Indexer & Retriever** - Embeddings are created during the querying time rather than during index construction itself. Computationally intensive due to high-dimensional vector operations





Source: https://www.analyticsvidhya.com/blog/2024/05/build-custom-retriever-using-llamaindex-and-gemini/

LlamaIndex – Indexing & Retrieval

- **3.Document Summary Index** – Combines summary efficiency with detailed context. Provides richer insights from large documents. Fast retrieval with comprehensive content.
- Retriever modes – LLM based retrieval ,Embedding based retrieval
- **LLM based retrieval** – Uses LLM to understand summary . Costlier and slower retrieval time. More accurate
- **Embedding Based retrieval** – Uses embeddings to compare summaries and queries. Faster but less accurate
- **4.Keyword Table Index & Retrieval -** Retrieve documents efficiently using keyword-based search instead of embeddings or vector retrieval
- Fast retrieval , storage efficient, not ideal for unstructured data
- 5. **Tree Index -** Organizes documents in a hierarchical tree structure to improve retrieval efficiency. It is especially useful when dealing with long-form documents or structured knowledge bases.

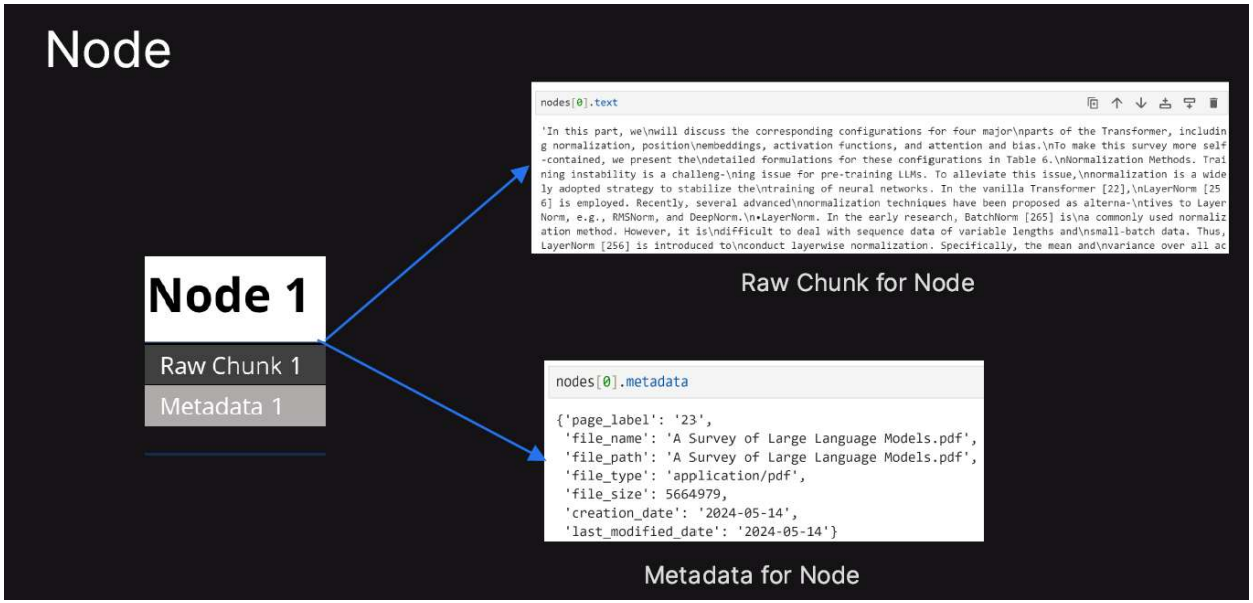
LlamaIndex – LLM

- **Large Language models** – Synthesise the answer based on query and retrieved chunks
- **Closed source LLM's** – GPT's ,Claude ,Gemini models ,etc
- **Open source LLM's** – Llama's , Mistral, Falcon ,Hugging Face
- CustomLLM's https://docs.llamaindex.ai/en/stable/module_guides/models/llms/usage_custom/
- Set context window, number of output tokens

```python
from llama_index.core import KeywordTableIndex, SimpleDirectoryReader
from llama_index.llms.openai import OpenAI
from llama_index.core import Settings

documents = SimpleDirectoryReader("data").load_data()


# set context window
Settings.context_window = 4096
# set number of output tokens
Settings.num_output = 256

# define LLM
Settings.llm = OpenAI(
    temperature=0,
    model="gpt-3.5-turbo",
    max_tokens=num_output,
)
```

LlamaIndex – Node Parsers

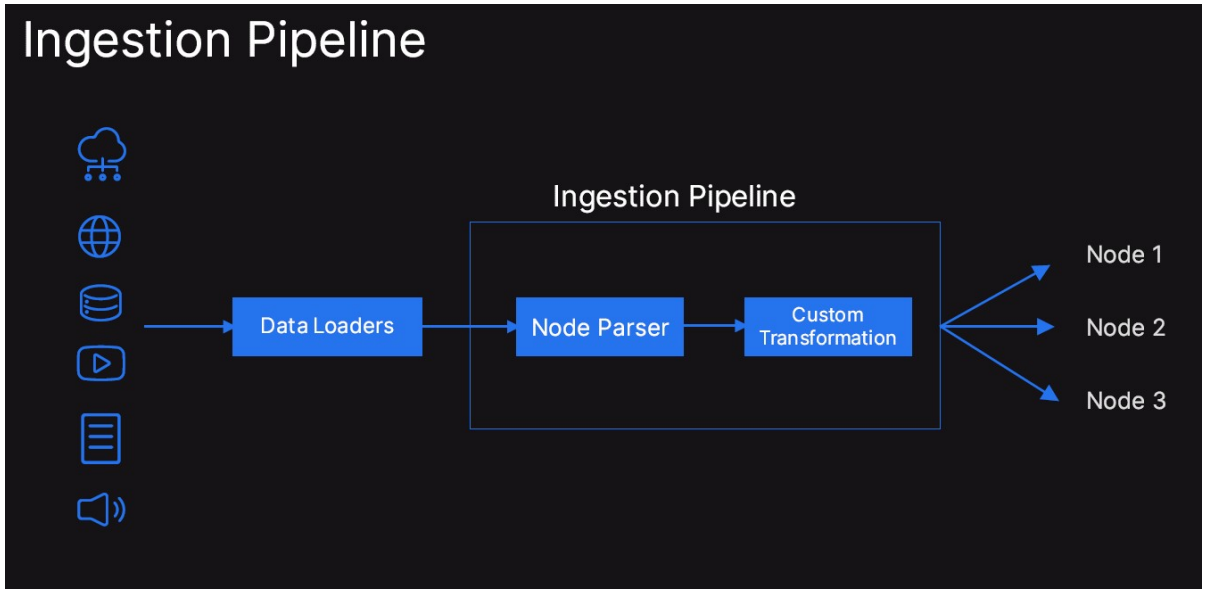
- A small, manageable piece of data that LlamaIndex processes, stores, and retrieves efficiently. These nodes help break down large documents into retrievable chunks, enabling better indexing and search.
- Contains chunks, metadata



Source: https://www.analyticsvidhya.com/blog/2024/05/build-custom-retriever-using-llamaindex-and-gemini/

LlamaIndex – Node Parsers & Transformations

- Break down data into manageable pieces (nodes) based on specific rules or formats, such as text, HTML, JSON, or CSV.
- TextNodeParsers,HTMLNodeparser,JsonNodeParser,CSVNodeParser, HierarchicalNodeParser, MarkdownNodeParser,SimpleNodeParser,SimpleFileNodeParser,etc
- Transformation - https://docs.llamaindex.ai/en/stable/module_guides/loading/ingestion_pipeline/transformations/



LlamaIndex – Query Engine

- Retrieves relavant documents from the index , Process and filters the retrieved data and generate responses
- Generic interface that allows user to ask questions over custom data and returns response
- **Streaming** - LlamaIndex enables response streaming, allowing partial results to be processed or displayed in real time, significantly reducing perceived query latency.
- **Chat Engine –** Enables conversation with your custom data

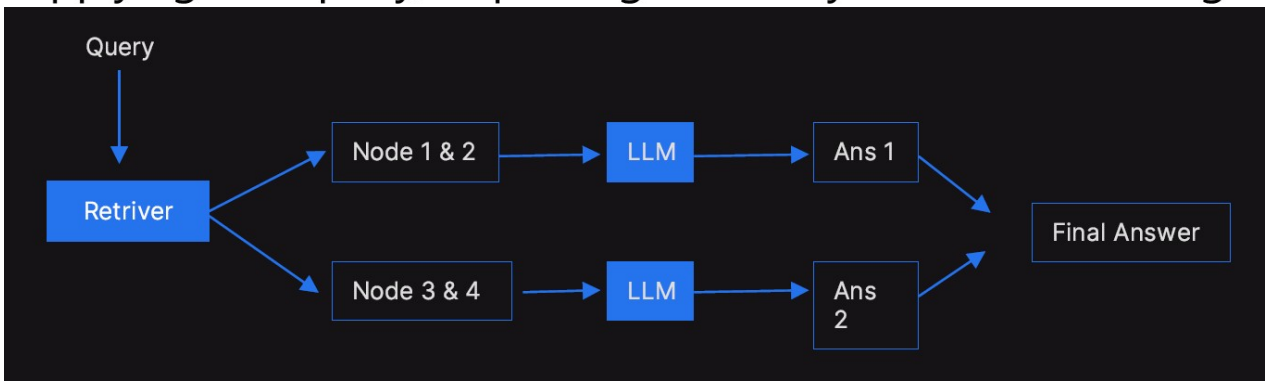
```
chat_engine = index.as_chat_engine()
response = chat_engine.chat("Tell me a joke.")
```

To stream response:

```
chat_engine = index.as_chat_engine()
streaming_response = chat_engine.stream_chat("Tell me a joke.")
for token in streaming_response.response_gen:
    print(token, end="")
```

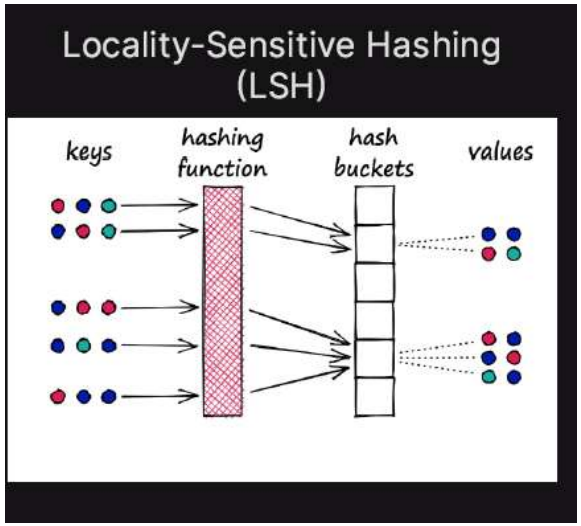
LlamaIndex – Response Synthesizer

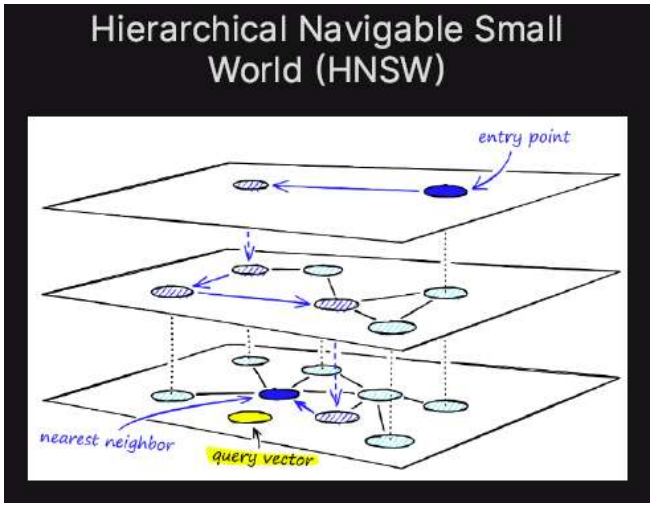
- Generate response from the user query and retrieved chunks. Response mode
- **refine** - create and refine an answer by sequentially going through each retrieved text chunk. This makes a separate LLM call per Node/retrieved chunk.
- **Compact** -concatenate the chunks beforehand, resulting in less LLM calls.
- **tree_summarize**: Query the LLM using the summary_template prompt as many times as needed so that all concatenated chunks have been queried
- **Accumulate**: Applies the query to each text chunk individually, accumulating responses into an array and returning them as a concatenated string. Ideal for querying each chunk separately.
- **Compact Accumulate**: Functions like accumulate but optimizes each LLM prompt by compacting text before applying the query, improving efficiency while maintaining separate chunk processing.



LlamaIndex – Vector Databases

- Specialized storage system designed to handle vector embeddings, allowing for approximate nearest neighbor (ANN) search. Instead of retrieving documents based on exact keyword matches, it finds semantically similar text based on embedding proximity in vector space.
- Vector Stores, Document stores, Index stores, Graph stores, Chat stores
- Pinecone, FAISS, Qdrant,Chroma,MongoDB,Weaviate,etc





LlamaIndex – Vector Databases

Vector Database	Search Techniques
Weaviate	HNSW (Hierarchical Navigable Small World), HNSW-PQ (Product Quantization), DiskANN (Disk-based Approximate Nearest Neighbors)
Vald	NGT (Neighborhood Graph and Tree-based indexing)
LanceDB	IVF (Inverted File Index), PQ (Product Quantization), DiskANN (Disk-based ANN search)
Redis	Flat (Brute Force Search), HNSW (Hierarchical Navigable Small World)
pgvector	IVF (Inverted File Index), IVF-PQ (Inverted File Index with Product Quantization)
Zilliz	Flat (Brute Force Search), Annoy (Approximate Nearest Neighbors Oh Yeah), IVF (Inverted File Index), HNSW/RHNSW (Hierarchical Navigable Small World / Refined HNSW), PQ (Product Quantization), DiskANN (Disk-based ANN search)
Milvus	Flat (Brute Force Search), Annoy (Approximate Nearest Neighbors Oh Yeah), IVF (Inverted File Index), HNSW/RHNSW (Hierarchical Navigable Small World / Refined HNSW), PQ (Product Quantization), DiskANN (Disk-based ANN search)

Category	Vector Databases
Closed-Source	Zilliz, Pinecone
Open Source / Source Available	Milvus, pgvector, Weaviate, Qdrant, Vespa, Vald, LanceDB, Chroma, Redis, Elasticsearch

LlamaIndex – SRE WorkFlow Scenario

- **Dataloaders** – Collect all necessary data sources such as **logs, SRE playbooks, monitoring dashboards, incident reports, and documentation**.
- **Chunking** – Break down large logs, playbooks, and reports into **manageable sections** for efficient processing.
- **Indexing** – Organize and structure the data for **quick and efficient retrieval**.
- **Retrieval** – Identify and extract the **most relevant sections** from logs, playbooks, or documentation to assist in troubleshooting.
- **LLM (Large Language Model)** – Analyzes the retrieved data, **interprets context**, and enhances understanding for more precise recommendations.
- **Response Synthesizer** – Aggregates multiple retrieved sources into a **coherent and actionable response** for SREs.
- **Query Engine** – Understands the engineer's query, selects the most relevant indexed information, and refines the response.
- **Evaluation** – Validates the accuracy, **relevance, and reliability** of the generated response before presenting it to the SRE.

RAG EVALUATION METRICS

EVALUATION METRICS

Evaluating **LlamaIndex-based RAG** involves assessing both the **retrieval** and **generation** components to ensure accuracy, relevance, and efficiency.

Component	Metric	Description
Retriever Evaluation	Hit Rate	Measures the proportion of queries where relevant documents were retrieved.
Retriever Evaluation	Mean Reciprocal Rank (MRR)	Evaluates how well the retrieved documents are ranked based on relevance.
Response Evaluation	Faithfulness	Ensures that the generated answer accurately reflects the provided information and avoids hallucinations.
Response Evaluation	Correctness	Assesses whether the generated answer aligns with a reference answer based on the query. Requires labeled data.
Response Evaluation	Context Relevancy	Checks if the retrieved context is relevant to the query.
Response Evaluation	Answer Relevancy	Determines if the generated answer is relevant to the query.
Response Evaluation	Semantic Similarity	Measures how semantically close the generated answer is to the reference answer. Requires labeled data.
Response Evaluation	Guideline Adherence	Ensures that the response aligns with predefined guidelines and standards.

Retrieval Evaluation Metrics

Metric	Description	SRE Use Case Example
Hit Rate	Measures how often relevant documents are retrieved.	Checking if logs retrieved for an incident investigation contain relevant data.
Mean Reciprocal Rank (MRR)	Evaluates the ranking of relevant documents among retrieved results.	Ensuring critical logs appear at the top when debugging an outage.
Recall@K	Measures the proportion of relevant documents found in the top K retrieved results.	Assessing if relevant playbooks appear in the top 5 recommendations.
Normalized Discounted Cumulative Gain (NDCG)	Weighs retrieval results based on ranking position and relevance.	Prioritizing logs that are most useful in troubleshooting.

✅ **Tools for Evaluation:**

- **LlamaIndex built-in evaluators** (`QueryEngine.evaluate_retrieval()`)
- **BM25/Retrieval Baseline Comparisons**
- **Custom Ranking Tests with Human Feedback**

Response Evaluation Metrics

Metric	Description	SRE Use Case Example
Faithfulness	Checks if the response is factually correct and avoids hallucination.	Ensuring AI-generated incident response steps align with runbooks.
Correctness	Evaluates if the generated answer matches ground-truth responses.	Verifying that AI-assisted troubleshooting steps follow company standards.
Context & Answer Relevancy	Assesses whether the retrieved context and generated answer are relevant to the query.	Making sure AI-generated summaries focus on the root cause of incidents.
Semantic Similarity	Measures how closely the response matches the expected answer.	Comparing AI-generated recommendations with historical incident resolutions.
Guideline Adherence	Ensures that responses comply with internal policies and guidelines.	Validating if AI-generated security recommendations follow compliance rules.

✅ **Tools for Evaluation:**

- **LlamaIndex** `ResponseEvaluator`
- **GPT-based evaluation for faithfulness**
- **BLEU/ROUGE Scores for correctness**
- **Manual Review & Human Feedback**

Latency & Performance Evaluation

3️⃣ Latency & Performance Evaluation

Since RAG systems must be **fast and efficient**, measuring response time and system performance is essential.

Metric	Description	SRE Use Case Example
Query Latency	Measures time taken to retrieve and generate a response.	Ensuring AI-assisted troubleshooting provides near real-time feedback.
Throughput	Evaluates how many queries can be processed per second.	Benchmarking AI response speed during major incidents.
Memory & Compute Efficiency	Tracks resource usage during retrieval and response generation.	Optimizing AI deployments to reduce cloud infrastructure costs.

✅ **Tools for Evaluation:**

- **LlamaIndex profiling tools**
- **Datadog, Prometheus for performance monitoring**
- **Model serving latency benchmarking**

Correctness

Correctness

Uber Technologies annual/quarterly revenue history and growth rate from 2017 to 2023.

- Uber Technologies revenue for the quarter ending September 30, 2023 was $9.292B, a 11.37% increase year-over-year.

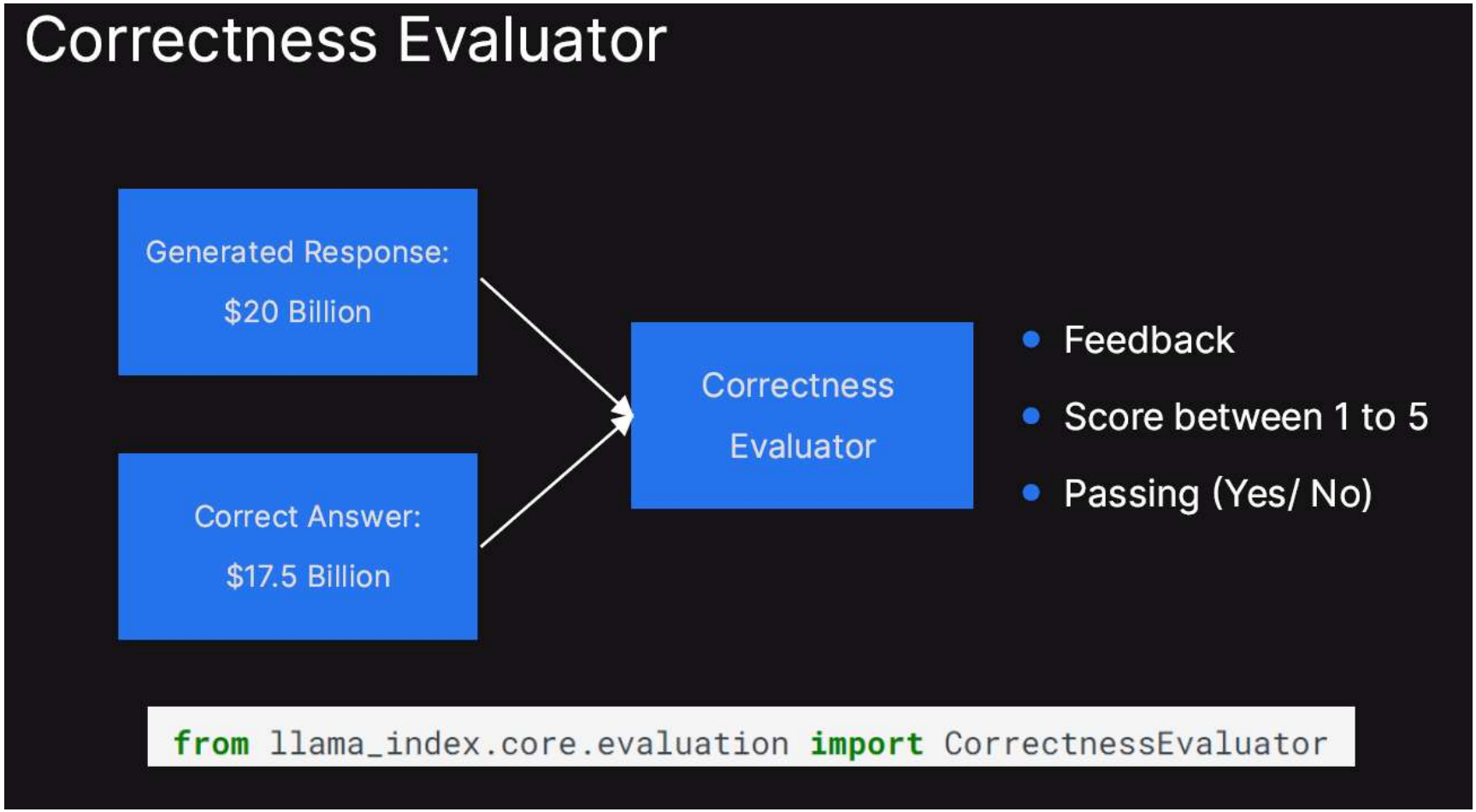
- Uber Technologies revenue for the twelve months ending September 30, 2023 was $35.952B, a 23.77% increase year-over-year.

- Uber Technologies annual revenue for 2022 was $31.877B, a 82.62% increase from 2021.

- Uber Technologies annual revenue for 2021 was $17.455B, a 56.7% increase from 2020.

Query: What is the revenue of Uber in 2021?

Correctness



Correctness

Calculating Factual Correctness

📌 **Formula:**

$$F1\_Score = \frac{|TP|}{(|TP| + 0.5 \times (|FP| + |FN|))}$$

📌 **LlamaIndex Code Snippet:**

```
evaluator.evaluate_factual_correctness(predicted_answer, ground_truth)
```

📌 **SRE Example:**

- **Ground Truth:** "Server CPU usage exceeded 90% due to memory leak."
- **Generated Response 1:** "High CPU usage detected due to memory leak." → **High F1 Score** ✅
- **Generated Response 2:** "CPU overload caused by disk failure." → **Low F1 Score** ❌

Correctness

Calculating Semantic Similarity

📌 **Formula:**

$$Semantic\_Similarity = Cosine\_Similarity(ground\_truth\_vector, generated\_answer\_vector)$$

📌 **LlamaIndex Code Snippet:**

📦 `evaluator.evaluate_similarity(predicted_answer, ground_truth)`

📌 **SRE Example:**

- **Ground Truth:** "Database query timeout occurred at 14:05 UTC."
- **Generated Response 1:** "Query execution delay detected at 14:05 UTC." → **High Similarity Score** ✅
- **Generated Response 2:** "Network latency detected at 14:05 UTC." → **Low Similarity Score** ❌

Correctness

Calculating Answer Correctness Score

📌 Final Score Formula:

$$Answer\_Correctness = (W_1 \times Factual\_Correctness) + (W_2 \times Semantic\_Similarity)$$

📌 LlamaIndex Code Snippet:

📦 `evaluator.evaluate(predicted_answer, ground_truth)`

📌 SRE Example:

- Factual Correctness (F1 Score) = 0.8
- Semantic Similarity = 0.7
- Weights: $W_1 = 0.6, W_2 = 0.4$
- Final Answer Correctness Score = (0.6 × 0.8) + (0.4 × 0.7) = 0.76 ✅

ADVANCED RETRIEVERS FOR RAG

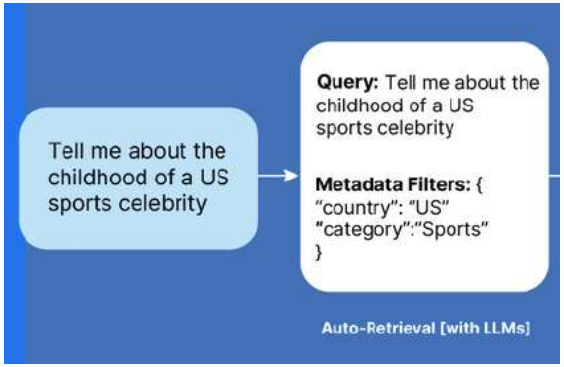
Advanced Retrievers

1. **Auto Retriever** - Use metadata filters and similarity score to retrieve the relevant nodes

🛠 Scenario:
🔹 Query: "Find the root cause of a high CPU spike in Kubernetes."
🔹 Metadata Filters: {Service: Kubernetes, Region: US-East, Severity: Critical}

🔹 Auto-Retriever Search:
- Fetches most relevant logs from the Vector Database.
- Applies semantic ranking and similarity scoring.
- Filters results to top-k logs with the highest relevance.

✅ Final Output:
- Log 1: "Kubernetes node overload detected due to excessive pod scheduling."
- Log 2: "Memory leak identified in microservice X, causing CPU spikes."
- Summary: "High CPU usage in Kubernetes was caused by excessive pod scheduling and a memory leak in microservice X."



Advanced Retrievers

2.Sentence Window Retirever - Enhances retrieval accuracy by selecting relevant text chunks along with adjacent sentences, ensuring better context preservation in query responses.

- **Scenario:**

An SRE wants to **investigate a sudden increase in API latency** across multiple services.

- **Query:**

"Why did the API response time increase drastically at 2:00 AM?"

- **How Sentence Window Retriever Helps:**

Instead of retrieving **isolated log entries**, it fetches **log lines before and after the relevant event**, preserving context.

```
from llama_index.indices.postprocessor import SentenceWindowRetriever

retriever = SentenceWindowRetriever.from_defaults(
    vector_retriever=vector_store.as_retriever(),
    window_size=2  # Fetches two adjacent sentences for context
)

response = retriever.retrieve("Why did the API response time increase at 2:00 AM?")
```

Advanced Retrievers

3.Auto-Merging Retriever: Dynamically combines multiple retrieved chunks into coherent, context-rich responses, ensuring more complete and concise information retrieval.

- **Scenario:**

An SRE is investigating **intermittent service failures** in a **multi-cluster Kubernetes environment.**

- **Query:**

"Why did multiple pods fail across different clusters yesterday?"

- **How Auto-Merging Retriever Helps:**

Instead of returning **disconnected logs from different clusters**, it **merges overlapping insights** from multiple sources into a **single, coherent summary.**

```
from llama_index.indices.postprocessor import AutoMergingRetriever

retriever = AutoMergingRetriever.from_defaults(
    vector_retriever=vector_store.as_retriever(),
    similarity_threshold=0.8  # Automatically merges overlapping results with high relevance
)

response = retriever.retrieve("Why did multiple pods fail across different clusters?")
```

Advanced Retrievers

4. Recursive Retriever: Iteratively refines queries by retrieving additional context from linked documents, enabling deeper and more accurate information retrieval.

🔹 **Scenario:**

An SRE is troubleshooting a **database connection failure** in a **distributed microservices architecture**.

Initial logs indicate a **timeout error**, but the root cause is unclear.

🔹 **Query:**

"Why are database connections failing in the payment service?"

🔹 **How Recursive Retriever Helps:**

- **Step 1:** Retrieves logs from the **payment service** that mention database failures.
- **Step 2:** Identifies that the issue originates from the **database connection pool exhaustion**.
- **Step 3:** Expands the search to fetch logs from the **database service** to pinpoint the exact cause.
- **Step 4:** Uncovers a **network congestion issue** between the API gateway and database, causing slow query responses.

```
from llama_index.indices.query_engine import RecursiveRetriever

retriever = RecursiveRetriever.from_defaults(
    retriever=vector_store.as_retriever(),
    max_depth=3  # Expands search iteratively up to 3 levels deep
)

response = retriever.retrieve("Why are database connections failing in the payment service?")
```

Advanced Retrievers

6. **Hybrid Fusion Retriever**: Combines multiple retrieval methods (e.g., keyword, vector, and metadata search) to enhance accuracy and relevance in query responses.

- Blends multiple retrieval methods (vector and keyword search).
- Uses reciprocal rank fusion to combine results effectively.

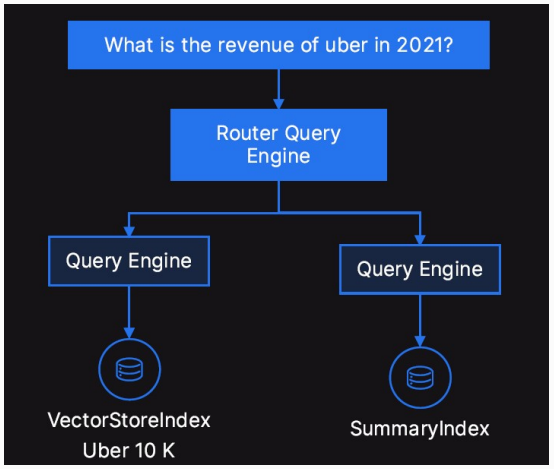
```python
from llama_index.indices.composability import HybridFusionRetriever
from llama_index.indices.vector_store import VectorStoreIndex
from llama_index.indices.keyword_table import KeywordTableIndex

# Create different indexes for hybrid retrieval
vector_index = VectorStoreIndex.from_documents(vector_documents)
keyword_index = KeywordTableIndex.from_documents(keyword_documents)

# Define Hybrid Fusion Retriever
retriever = HybridFusionRetriever(
    retrievers=[
        vector_index.as_retriever(),  # Vector-based retrieval
        keyword_index.as_retriever()  # Keyword-based retrieval
    ],
    fusion_algorithm="reciprocal_rank"  # Combines results using reciprocal rank fusion
)

# Execute query
response = retriever.retrieve("Find the root cause of Kubernetes pod failures")
```

Optimized Query Engines

5.Router Query Engine - Directs queries to the most relevant retriever or index based on query intent, optimizing response accuracy and efficiency.

Sub-Question Query Engine: Breaks down complex queries into smaller, focused sub-questions, retrieves relevant information for each, and synthesizes a comprehensive response.

```python
from llama_index.query_engine import SubQuestionQueryEngine
from llama_index.indices.vector_store import VectorStoreIndex

# Create an index for searching
vector_index = VectorStoreIndex.from_documents(system_logs)

# Define sub-question engine
query_engine = SubQuestionQueryEngine.from_defaults(vector_index.as_query_engine())

# Execute a complex query that requires sub-questions
response = query_engine.query("Why did the API response time increase and how is it related to database errors?")
```




```python
from llama_index.query_engine import RouterQueryEngine
from llama_index.indices.vector_store import VectorStoreIndex
from llama_index.indices.keyword_table import KeywordTableIndex

# Create different indexes for routing
vector_index = VectorStoreIndex.from_documents(vector_documents)
keyword_index = KeywordTableIndex.from_documents(keyword_documents)

# Define routing logic based on query type
query_engine = RouterQueryEngine(
    selector=lambda query: "vector" if "deep analysis" in query else "keyword",
    query_engine_map={
        "vector": vector_index.as_query_engine(),
        "keyword": keyword_index.as_query_engine(),
    }
)
```

Advanced Approaches for RAG

7. Multi-Document Agents

- **What Are Multi-Document Agents?**

Multi-document agents intelligently **retrieve, analyze, and synthesize** information from multiple sources to generate **context-rich responses**.

- **How They Work?**

1. **Query Understanding:** Analyzes the query and determines relevant sources.
2. **Parallel Retrieval:** Fetches documents from **multiple indexes** (vector, keyword, structured).
3. **Fusion & Ranking:** Merges retrieved results using **hybrid fusion** techniques.
4. **Response Generation:** Synthesizes an **accurate and context-aware** answer.

Advanced Approaches for RAG

🔹 **SRE Use Case:**

📌 **Scenario:** An SRE needs to **correlate incidents** across **system logs, monitoring alerts, and service dashboards** to diagnose a **recurring outage**.

✅ **Query:**

"What caused service X downtime over the last 24 hours?"

✅ **Multi-Document Agent Process:**

- Retrieves **incident reports, log files, and monitoring data**.
- Cross-references **alert timestamps with API failures**.
- Generates a **root cause summary** with remediation steps.

```
from llama_index.query_engine import MultiModalQueryEngine

# Define multiple retrievers (logs, incidents, metrics)
query_engine = MultiModalQueryEngine(
    query_engine_map={
        "logs": log_index.as_query_engine(),
        "alerts": alert_index.as_query_engine(),
        "metrics": metric_index.as_query_engine()
    }
)

# Execute cross-source query
response = query_engine.query("Analyze root cause for service X outage in the last 24 hours")
```

ReAct RAG –Reasoning +Action+ knowledge

Feature	Description
Intelligence	Employs a RAG workflow, combining LLMs with external knowledge sources (databases, APIs, documentation) for enhanced context and accuracy.
Behavior	Uses ReAct-style reasoning to break down tasks, dynamically retrieving information as needed. Grounded in real-time or domain-specific knowledge.
Scope	Designed for scenarios requiring high accuracy and relevance, minimizing hallucinations.
Best Use Cases	High-stakes decision-making, domain-specific applications, tasks with dynamic knowledge needs (e.g., real-time updates).
Examples	Legal research tools, medical assistants referencing clinical studies, technical troubleshooting agents.

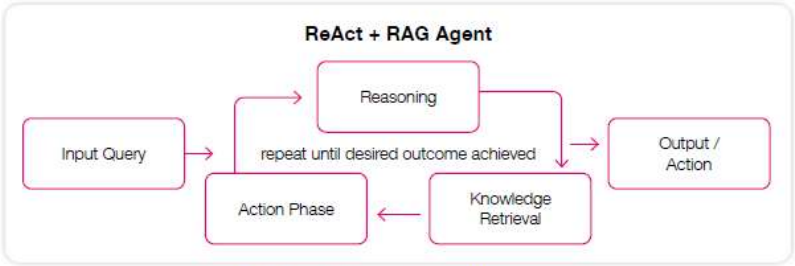
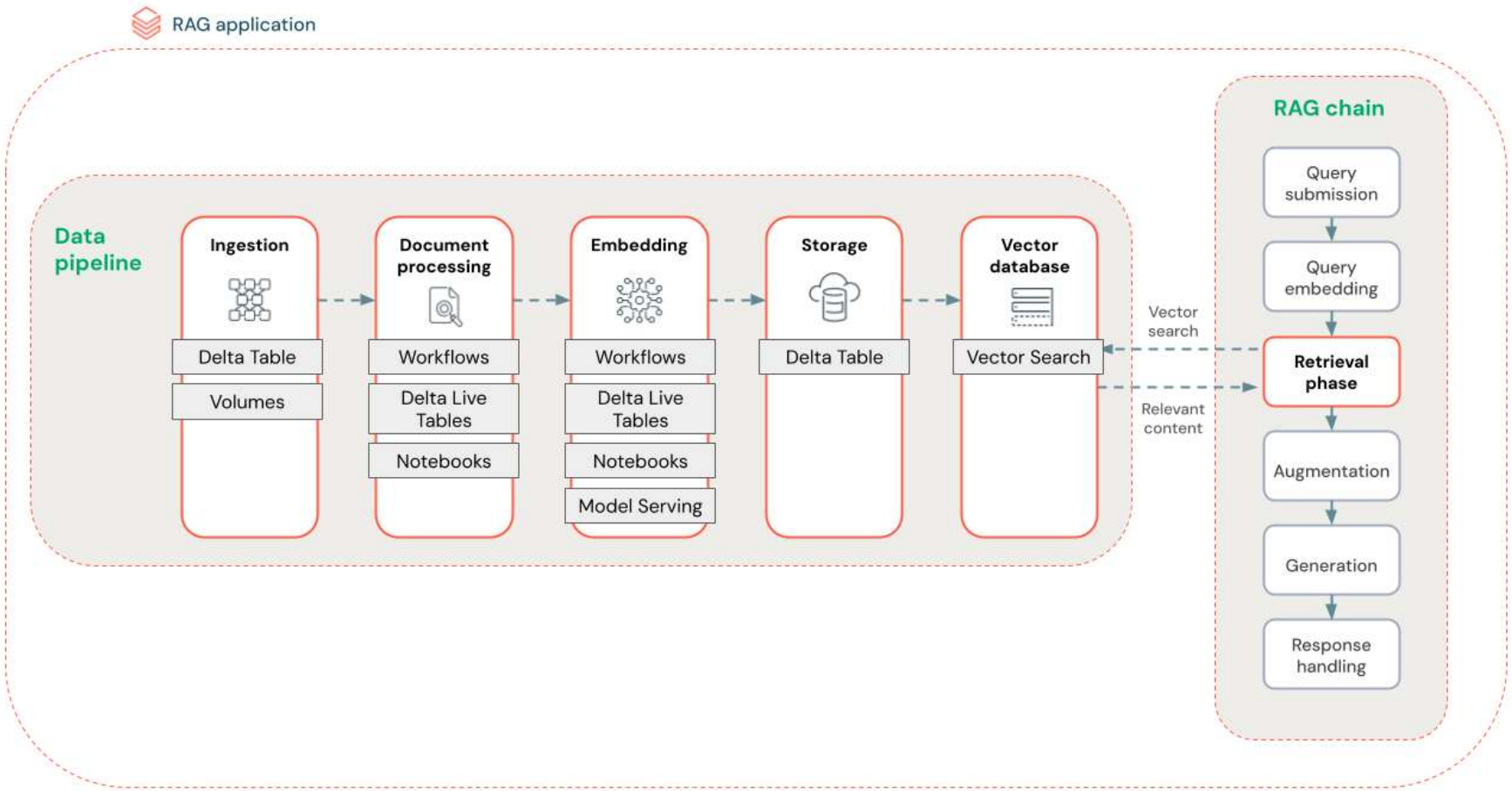


Fig 1.5: Workflow of a ReAct + RAG agent

Source:https://www.galileo.ai/ebook-mastering-agents

Real-time Use Cases

RAG DATA Pipeline



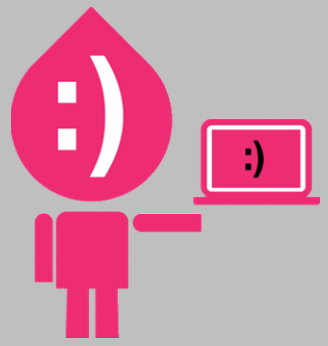
Source: https://www.databricks.com/glossary/retrieval-augmented-generation-rag

Real-time Use cases

- Question- Answering – Incident Responder
- Chatbots –Customer Support Agent
- Document Understanding & Data Extraction – Research Assistants
- Autonomous Agents – HR Assistants
- Multimodal applications – Personalized Shopping

Beyond Retrieval: The Evolution Continues

- LlamaIndex simplifies RAG implementation , powerful tool for data indexing and querying and a great choice for projects that require advanced search.
- LlamaIndex enables the handling of large datasets, resulting in quick and accurate information retrieval.
- **Boosting Performance with Caching** – Speed up retrieval and reduce API costs by leveraging in-memory, disk, or vector caching in LlamaIndex.
- LlamaCloud is a managed platform for data parsing and ingestion, allowing you to get production-quality data for your production LLM application.
- LlamaParse is the world's first genAI-native document parsing platform - built with LLMs and for LLM use cases.
- https://github.com/run-llama/llama_deploy
- https://llamahub.ai/
- https://www.secinsights.ai/
- https://www.npmjs.com/package/create-llama
- https://docs.llamaindex.ai/en/stable/getting_started/discover_llamaindex/



Q&A...

https://github.com/hemsush/ChennAITalk2025