

Advance RAG .....	2
Comparing LLMS for Rag Over Code.....	3
Advanced RAG Taxonomy.....	5
AdvanceRAG Tools Comparison.....	6
Advance RAG : Query Optimization.....	10
Advance RAG : Document Optimization.....	12
Advance RAG : Embedding Models.....	14
Advance RAG : Context Expansion.....	16
Advance RAG : Context Rerankers.....	18
Testing Frameworks.....	21
RAG over different types of Data.....	22
Structured Planning Agent Llama-index.....	24
Explore Evaluation Metrics.....	26
Evaluation Metrics.....	28
Q/A with RetrieverRouter engine from llama-index.....	30
Recursive Retrieval Techniques.....	31
Exploring JSON query engines of llama-index.....	33
Capture Explanations and Compare Our RAG Mindmap.....	38
Exploring text-to-SQL Query engine of llama-index.....	40
Q/A over unstructured vs structured data.....	41
Q/A with ReAct agent for routing.....	42
RetrieverRouterQueryEngine vs StructuredPlanningAgent vs ReAct for query routing.....	46
Mini-Assistant : Combining Q/A and function calling tools with ReAct on top for agentic reasoning.....	47
Compare Table Formats & Different Table Types.....	50
GraphRAG.....	51
Fine Tuning Embedding's.....	53
Embeddings Course 1.....	57
Adapter over embeddings.....	61
AUTO-HYDE.....	63
Flow-AutoHyde.....	65
Fine tuning Rerankers.....	68
Fine tuning LLMs : An estimate.....	73
Fine tuning LLMs.....	74

# Advance RAG

## Rag over pdf

- Number of pdfs - 3 (all from different domains)
- Queries - 50
- Dataset fields - (Queries, Contexts, Ground Truth)
- [GitHub - Rutuja161/Pdf-based-question-answering: This is the research project for my masters in Data analytics. This aims to build a smart question answering system for PDF files using BERT - based transformers.](https://github.com/Rutuja161/Pdf-based-question-answering)
- Evaluators used : 2

## Rag over Code

### .POC

- Number of repos - 1
- Queries - 10
- [GitHub - emumba-com/emumba-nx-boilerplate-plugin](https://github.com/Emumba/nx-boilerplate-plugin)
- <https://github.com/meshalJcheema/Advance-Rag-Mindmap-usecases> Connect your Github account
- Evaluators used : 5

## Evaluators

- Following are the two metrics used from evaluators
- **Response Matching/Answer Relevance** : Measures the percentage of responses that exactly match the expected answer. (uses ground truth)
- **Context Relevance** : measures if the retrieved context has enough information to answer the question being asked. how good is the context retrieved

## Comparing LLMS for Rag Over Code

### Extract content from Github Folder and Split using Langchain ↗

This function does the following:

- Defines a Document class to store file content and metadata.
- Reads files from a specified path and creates Document objects for each file.
- Splits the original documents into smaller chunks using the text splitter, creating a new list of documents.

## PIPELINE - 1 ↗

### GENIE-3 & GENIE-4 & GENIE-9 : Pipeline With Different Bedrock LLMs and Embeddings ↗

Following Pairs were made for analysis:

1. Model Claude-V2, Embeddings - model\_id="amazon.titan-embed-text-v1"
  2. Model llama 2 70b, Embeddings - model\_id="amazon.titan-embed-text-v1"
  3. Model llama 2 70b, Embeddings - model\_id="cohere.embed-english-v3"
  4. Model llama 2 70b, Embeddings - model\_id="text-embedding-ada-002"
- The following pipeline uses FAISS as its vectorDB.

### GENIE-5 : Evaluation of pipeline using LLM ↗

- The llm used is the claude V2 from bedrock service.
- The process automates the evaluation of answer quality, using a language model to assign relevance scores to pairs of original and candidate answers based on their context and query.

## PIPELINE - 2 ↗

### GENIE-6 : RAPTOR OVER RAG ↗

- The following pipeline uses openai embeddings model and bedrock model claude as its llm
- Imports OpenAIEmbeddings and ChatOpenAI from langchain\_openai, and various other classes and functions from langchain for embedding and model interaction purposes.
- Introduces a recursive clustering and summarization function that aims to embed texts, perform clustering at multiple levels, and summarize the texts within each cluster.
- The summarization part is intended to generate a summary for each cluster of texts by utilizing a language model (presumably Claude- v2), following a specified prompt template and output parsing.
- Contains a loop at the end to process a list of queries, generate responses using a chain that involves document retrieval, prompting, and the Claude-v2 model for summarization, and then saves each response to a separate JSON file.

## PIPELINE - 3 ↗

### GENIE-8 : OLLAMA EVALUATION ↗

- The llm is used is llama2 and embeddings used are GPT4allembddings.

## **GENIE-7 : RAG EVALUATION**

- The llm used is Claude - V2 from bedrock service, OpenAI embeddings and Weaviate vectorDB for embeddings manipulation.
- Assembles the queries, answers, contexts, and ground truth answers into a dataset.
- Evaluates the dataset using several metrics (context precision, context recall, faithfulness, and answer relevancy) to assess the quality of the generated answers.
- Converts the evaluation results into a pandas DataFrame and prints it.
- Saves the evaluation results to a CSV file named "ragas\_results.csv".
- Overall, this pipeline demonstrates a workflow for answering queries with context from a vector database, generating answers with a language model, and evaluating the quality of those answers against ground truth data.

# Advanced RAG Taxonomy

The task is designed to develop a comprehensive taxonomy for retrieval augmented generation models. The main goal of the taxonomy is to provide a structured framework by categorising the tools and techniques used in different components of RAG pipeline. ☀

Link to Mind Map: <https://mm.tt/app/map/3231556693?t=U72qr.jV59q>

## Approach: ☀

A comprehensive literature review was conducted to understand the RAGs and its pipeline. This involved studying various survey papers, article and frameworks.

Based on the the gained insight, different components were identified and characterised.

The identified components were carefully fitted into the pipeline of the advanced rag

model. The Rag pipeline is mainly divided into 6 components:

- 1- Pre-Retrieval: It is used before the retrieval process. Main purpose is to pre-process the user query and documents.
- 2- Retrieval: This component is used in the overall retrieval process that includes embedding models and retrieval algorithms.
- 3- Generation: It comprises of the post-retrieval part. That further involves relevance enhancement and LLM- FINE TUNING. Relevance enhancement involves context expansion, response re-ranking and embedding adapter.
- 4- Augmentation: It mainly comprises of 3 different retrievals that helps in the augmentation process.
- 5- RAG Evaluation: Various evaluations methods are categorised along with their implementations.
- 6- Tools: Famous tools that are being used for RAG implementation are classified along with their implementations.

## Information Sources Categorised by Colors: ☀

- Green: Git hub Repository
- Blue: Research Paper
- Yellow: Official Library
- Pink: Blog/ Website/ Colab notebook

# AdvanceRAG Tools Comparison

Here's a comparative tables based on comparison for the following tools

1. LlamaIndex
2. LangChain
3. Haystack
4. Hugging Face

## Definition Analysis between Tools ↗

Category ↗	LlamaIndex ↗	LangChain ↗	Haystack ↗	Hugging Face ↗
Supported Languages	Python TypeScript	Python JavaScript	Python	Python
Main Focus	* Indexing and efficient retrieval	* General-purpose LLM applications * RAG * approach context-aware apps	* Semantic search and Q&A systems	* Pre-trained models, transfer learning * wide range of NLP tasks
Key Features	* Integration with diverse data sources (160+) * list index * customizable workflows	* Modular design * flexible integration * development * deployment * observability * access to various LLMs	* Vendor neutrality * interoperability * extensibility * customization	* Pre-trained models * extensive documentation * large community * easy transfer learning * customization
Pros	* Handles wide range of data formats * Advanced indexing techniques * Customizable RAG process	* Modular architecture Extensive ecosystem	* Designed for search and Q&A * Specialized components * Integrated workflow	* Wide range of state-of-the-art model * User-friendly API * Large community * Customizable

<b>Cons</b>	<ul style="list-style-type: none"> <li>Limited to data retrieval, less suitable for complex NLP tasks</li> </ul>	<ul style="list-style-type: none"> <li>Generalized, not specialized- Performance may vary based on use case</li> </ul>	<ul style="list-style-type: none"> <li>Narrower scope</li> <li>Primarily focused on document understanding</li> </ul>	<ul style="list-style-type: none"> <li>Not specifically focused on search or retrieval systems</li> </ul>
-------------	--	--	---	---

			and retrieval tasks	
<b>Use Cases</b>	<ul style="list-style-type: none"> <li>Q&amp;A chatbots</li> <li>Document understanding and extraction</li> <li>Autonomous agents</li> </ul>	<ul style="list-style-type: none"> <li>Prototyping and experimentation</li> <li>Enterprise-level applications</li> </ul>	<ul style="list-style-type: none"> <li>Enterprise search and QA</li> <li>Specialized knowledge management</li> </ul>	<ul style="list-style-type: none"> <li>Text classification</li> <li>Question answering</li> <li>Language translation</li> </ul>
<b>Vendor Integration</b>	Customizable with different retrieval strategies	Integrates with multiple LLM providers like OpenAI, Cohere, Hugging Face Hub	Integration with various model providers and databases	Provides a wide range of pre-trained models from different providers
<b>Community and Ecosystem</b>	Growing community, lightweight interface	Large and active community, well-documented examples	Well-documented API, growing community	Extensive community and resources, active user base
<b>Performance Optimization</b>	Focused on efficient indexing and retrieval	Flexible but may not match performance of tailored solutions for specific tasks	Optimized for search and Q&A, better performance in semantic search	Optimized for various NLP tasks, transfer learning capabilities
<b>Example Libraries/Tools</b>		LangSmith, LangServe		Transformers

## Advance Rag Use-Cases comparison ↗

Feature	LlamaIndex	LangChain	Haystack	Hugging Face
---------	------------	-----------	----------	--------------

<b>Query Optimisation</b>				
a) Multi-query expansion	✓	✓	✗	✗
b) Sub-query expansion	✓	✓	✗	✗
c) Hyde-adapter for embeddings	✓	✓	✓	✗
d) Rewrite queries	✗	✓	✗	✓
<b>Documents Optimisation</b>				
a) Chunking strategies	✓	✓	✗	✗
b) Metadata attachment	✓	✓	✗	✗

c) Hierarchical index structure	✓	✗	✗	✗
d) Knowledge graph index	✓	✗	✓	✗

<b>Embeddings Model</b>				
a) Ollama	✓	✓	✓	✓
b) Cohere	✓	✓	✓	✓
c) Jina	✓	✓	✓	✓
d) Voyage	✓	✓	✓	✓
e) OpenAI	✓	✓	✓	✓
f) LLM Embedder	✗	✗	✗	✓
g) BGE Embeddings	✗	✗	✗	✓

<b>Context Expansion</b>				

a) Sentence window retrieval	✓	✗	✗	✗
b) Recursive retrieval	✓	✗	✗	✗
c) Auto merging	✓	✗	✗	✗
<b>Context Rerankers</b>				
a) Cohere reranker	✓	✓	✓	✗
b) BGE reranker	✗	✗	✓	✓
c) Rank fusion	✓	✓	✓	
d) ColBERT reranker	✓	✓	✓	✓
e) Long context reorder	✓	✓	✗	✗
f) Jina reranker	✓	✓	✓	✓
g) RankGPT	✓	✓	✓	✓

## Tools and Query Router Engines

### Llmaindex

Llmaindex offers following query routing techniques

1. Pydantic Single Selector
2. LLM Single Selector
3. Pydantic Multi Selector
4. Query engine tools (define description in tools definition)

### Langchain

Langchain offers following query routing techniques

1. Routing with function calling models
2. Routing to multiple indexes
3. Dynamic Routing based on custom logic related to what is your input

### Haystack

Haystack offers following query routing techniques

1. Classifying keywords, questions and statement queries
2. Zero shot classification
3. Routing based on content type and metadata field
4. Conditional routers
5. Filetype routers
6. Text Language router
7. Transformerszeroshottextrouters

### Hugging Face

Hugging face offers following query routing techniques

1. Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing (paper)
2. Decomposing Complex Queries for Tip-of-the-tongue Retrieval (similar to subquery idea)

# Advance RAG : Query Optimization

DATA : PDF documents, link mentioned on main

page Total queries : 50

Evaluation Tools are Uptrain & Tonic Validate

Evaluation Techniques are Baseline Experiment, Multiquery, Subquery, HYDE, Multi step, Step back, Query

Rewriting. Metrics for evaluation are Answer Similarity, Context Precision, Context Similarity.

Answer Similarity	Baseline Experiment	Multiquery	Subquery	HYDE	Multi step	Step back	Query Rewriting
Uptrain	25%	28%	15%	48%	21%	27%	39%
Tonic Validate	44%	44%	30%	64%	48%	44%	50%

Context Precision	Baseline Experiment	Multiquery	Subquery	HYDE	Multi step	Step back	Query Rewriting
Uptrain	45%	51%	22%	100%	44%	43%	49%
Tonic Validate	40%	64%	29%	80%	65%	63%	53%

DATA : Code files from a GitHub repository, link mentioned on main page

Total queries : 10

Context similarity	Multiquery	Subqueries	HYDE	Multi step	Step back prompting
Ragas	31%	82%	91%	91%	90%
Tonic Validate	1.5/5	3.2/5	4.4/5	1.2/5	1/5
DeepEval	10%	40%	85%	10%	45%
UpTrain	14%	34%	57%	23%	21%
Trulens	41%	56%	64%	76%	76%
Falcon-evaluate	25%	50%	96%	40%	33%

## Conclusion

- HYDE performs best! - Expands search space, reduces hallucinations, we perform answer to answer embedding vector stores similarity that is resulting in better results.
- All other techniques,
- They highly depend on type of query user is going for.
- These techniques result in a lot of hallucinations.
- Therefore, on fairly decent queries present in our dataset, their scores lie near our baseline experiment except Hyde and query rewriting.

# Advance RAG : Document Optimization

## Experiment Results ↗

DATA : PDF documents, link mentioned on main

page Total queries : 50

Context Precision	Baseline Experiment	Semantic Chunking	Tik token Chunking	Character text Chunking
Up train	45%	67%	69%	<b>81%</b>
Tonic Validate	40%	74%	70%	<b>74%</b>

Answer Similarity	Baseline Experiment	Metadata Attachment
Up train	25%	<b>52%</b>
Tonic Validate	44%	<b>56%</b>

Context Precision	Baseline Experiment	Metadata Attachment
Up train	45%	<b>73%</b>
Tonic Validate	40%	<b>75%</b>

Answer Similarity	Baseline Experiment	Hierarchical Indexing
Up train	25%	<b>56%</b>
Tonic Validate	44%	<b>68%</b>

Context Precision	Baseline Experiment	Hierarchical Indexing
Up train	45%	<b>80%</b>
Tonic Validate	40%	<b>75%</b>

DATA : Code files from a GitHub repository, link mentioned on main page

Total queries : 10

Context similarity	Recursive Chunking	Meta Data Inclusion	Semantic Chunking	Character Text splitting	Tik Token Based Splitting	Hierarchical Indexing
Ragas	80%	91%	77%	<b>88%</b>	71.08%	<b>82%</b>
Tonic Validate	2/5	1.9/5	1/5	<b>2.1/5</b>	0.9/5	<b>4.4/5</b>
Deep eval	35%	30%	49%	<b>50%</b>	40%	<b>54%</b>
Up train	26%	26%	22%	<b>40.8%</b>	16%	<b>67%</b>
Trulens	70%	72%	74%	<b>75.5%</b>	76%	<b>76%</b>
Falcon-evaluate	50%	46%	45%	<b>59%</b>	59%	<b>96%</b>

## Conclusion ↴

- There are multiple chunking techniques out there, depending upon use cases, code languages, extensions etc, we tried for our experiment the generic ones, all performed better than baseline experiment, but **Character text splitting got maximum scores!**
- Metadata integration** with baseline experiment boosted up the accuracy scores
- Hierarchical indexing** dynamic parsing and creation of multiple layers of indices, not only gives you good results but it also has an optimized Response Time.
- Include metadata, necessary for scenarios where we have domain specific queries. Even include metadata with no domain specific, its good to have it
- The hierarchical and KI graphs can't be compared to chunking, they have different retrievers, nodes are there which are stored processed differently. Their main Plus seems to be their very fast storage and retrieval because of the way their structures are build but at the same time, they are much more complex to be understand-ed and integrated to our RAG pipeline

# Advance RAG : Embedding Models

## Variants and Embedding Size ↗

< TO BE INCLUDED > SOON

## Experiment Results ↗

DATA : PDF documents, link mentioned on main

page Total queries : 50

Answer Similarity	Baseline Experiment (ollama)	OpenAI	JinaAI	BGE3	Cohere	Voyage AI	LLM embedder	Bedrock
Uptrain	25%	36%	38%	50%	37%	34%	28%	24%
Tonic Validate	44%	60%	54%	60%	53%	53%	48%	50%

Context Precision	Baseline Experiment (ollama)	OpenAI	JinaAI	BGE3	Cohere	Voyage AI	LLM embedder	Bedrock
Up train	45%	88%	54%	86%	52%	66%	52%	45%
Tonic Validate	40%	76%	69%	79%	74%	67%	55%	61%

DATA : Code files from a GitHub repository, link mentioned on main page

Total queries : 10

Context Similarity	Openai	Bge - m3	llm-embedder	Cohere	all-minilm	Mxbai - embed large	Voyage	JIna Ai
Ragas	91%	82.5%	80	77%	78%	79%	75%	76%
Tonic Validate	1.9/5	3.4/5	3/5	1.1/5	2.5%	3.1%	2.8/5	2.9/5
Deepeval	40%	33%	38	44%	36%	34%	39%	37%
Uptrain	33%	33%	35	24%	30%	32%	31%	32%

<b>Trulens</b>	75%	<b>76.22%</b>	76.22%	76%	76.22%	76.22%	76.22%	76.22%
----------------	-----	---------------	--------	-----	--------	--------	--------	--------

<b>Falcon-evaluate</b>	55%	<b>46%</b>	48	33%	50%	47%	52%	49%
------------------------	-----	------------	----	-----	-----	-----	-----	-----

## Conclusion ↗

- All performed better than baseline experiment that uses minilm ollama embeddings except bedrock embeddings they were nearly equal to minilm ones.
- BGE-m3 embeddings got maximum scores — (multilingual, trained over short, long texts texts, 568 million parameters!)
- Between all others there does not lie much difference.
- Try them with a reranker for boosted results?
- BGE-m3, LLM embedders can't be integrated into our original pipeline, i made their own RAG pipeline and they need a GPU units, processing on CPU is very slow, in terms of model loading and initialization. Embedding formation time of T4-GPU is given above.

## Advance RAG : Context Expansion

Following techniques are implemented, there is a general overview, how these techniques work.

Technique	Description	RAG use case
<b>Sentence Window Retrieval</b>	<p>Segmenting the corpus into overlapping sentence windows and scoring based on query relevance.</p> <p>Scoring each window based on its relevance to the input query.</p> <p>Providing highest scoring windows as context</p>	<p>Suitable for use cases where detailed context from specific segments of text is necessary,</p> <ul style="list-style-type: none"><li>* question</li><li>* answering</li><li>summarization.</li></ul>
<b>Auto Merging Retrieval</b>	<p>Extracting relevant passages from multiple sources and combining them into a cohesive context.</p> <p>Automatically combining passages into a single context.</p> <p>Feeding the merged context</p>	<p>Ideal for scenarios where information is scattered across different sources, like</p> <ul style="list-style-type: none"><li>* creating comprehensive reports</li><li>* synthesis from diverse documents.</li></ul>
<b>Recursive Retrieval</b>	<p>Iteratively refining the retrieval process to improve relevance and accuracy.</p> <p>Using retrieved information to refine the input query or context.</p> <p>Iterative Retrieval: Repeating the retrieval process with refined queries or contexts.</p>	<p>Best suited for complex queries where initial retrievals may need further refinement, such as</p> <ul style="list-style-type: none"><li>* multi-turn dialogue systems</li><li>* iterative problem-solving tasks</li><li>* Pdfs with tabular data</li></ul>

## Experiment Results ↗

DATA : PDF documents, link mentioned on main

page Total queries : 50

Answer Similarity	Baseline Experiment	Sentence Window	Auto Merging	Recursive
Up train	25%	52%	54%	35%
Tonic Validate	44%	67%	68%	39%



Context Relevance	Baseline Experiment	Sentence Window	Auto Merging	Recursive
Up train	45%	69%	77%	33%
Tonic Validate	40%	74%	73%	55%

DATA : Code files from a GitHub repository, link mentioned on main page

Total queries : 10

Context similarity	Sentence window	Auto Merging	Recursive
Ragas	92%	96%	-
Tonic Validate	4/5	4/5	-
Deep Eval	46%	40%	-
Up Train	71%	65%	-
Tru lens	76.25%	76%	-
Falcon-evaluate	96	100%	-

## Conclusion ↗

- Every technique is better than the baseline experiment, and auto merging and sentence window both gave us quite good results!
- **Sentence Window Retrieval** : Good for extracting specific lines or sections from PR descriptions, comments, or diffs that are relevant to a query
- **Auto Merging Retrieval** : Useful for combining information from multiple PRs, comments, and reviews to provide a comprehensive response
- **Recursive Retrieval** : Highly suitable for iterative queries where initial results may need further refinement, such as detailed PR review or code analysis
- They are provided by framework llama index, and can be integrated with any LLM and embedding model
- For sentence window retrieval, what if we consider base nodes as well? This May increase the performance even more
- We can not test Recursive as the pipeline does not give us the retrieved context, and evaluation matrix required that to generate a score, but in pdf based testing, there are scores defined for it because the pipeline used partially satisfies its concepts, that's why it is highlighted red as well but still its better than baseline experiment.

# Advance RAG : Context Rerankers

## Overview

- The initial list of retrieved documents is reordered based on the new relevance scores generated by the re ranking model.
  - The documents that are deemed more relevant in the given context are moved higher in the ranking.
  - The top n ranked items for our experiments is kept four!
  - The n retrieved items are kept six

## Experiment Results

DATA : PDF documents, link mentioned on main

page Total queries : 50

Answer Similarity	Baseline Experiment	Colbert	Cohere	Jina	BGE	Rag Fusion	Rank Gpt	Long context reorder
Up train	25%	47%	49%	48%	33%	42%	49%	39%
Tonic Validate	44%	63%	54%	61%	51%	56%	60%	49%

Answer Similarity	Baseline Experiment	Colbert	Cohere	JIna	BGE	Rag Fusion	Rank Gpt	Long context reorder
-------------------	---------------------	---------	--------	------	-----	------------	----------	----------------------

<b>Uptrain</b>	45%	<b>76%</b>	67%	<b>82%</b>	43%	82%	78%	63%
----------------	-----	------------	-----	------------	-----	-----	-----	-----

<b>Tonic</b>	40%	<b>83%</b>	77%	<b>88%</b>	66%	75%	87%	67%
--------------	-----	------------	-----	------------	-----	-----	-----	-----

DATA : Code files from a GitHub repository, link mentioned on main page

Total queries : 10

Context Relevance	Colbert reranker	Cohere reranker	Rag-fusion	RankGPT	Bge ranker	Long context reorder	Jina Ai reranker
<b>Ragas</b>	82%	90%	87%	88%	82.5%	93.0%	93%
<b>Tonic Validate</b>	4.5/5	3.2/5	3.6/5	3.8/5	3.4/5	4/5	3.8/5
<b>Deepeval</b>	70%	40%	68%	50%	33%	45.8%	63%
<b>Uptrain</b>	70%	39%	51%	53%	33%	45.2%	48%
<b>Trulens</b>	75.68%	<b>75.68%</b>	<b>75.6%</b>	<b>76.25%</b>	<b>76.22%</b>	<b>76.22%</b>	<b>76.01%</b>
<b>Falcon-evaluate</b>	96%	56%	42%	57%	55%	42%	42%

## Conclusion

- All rerankers performed better than baseline experiment.
- Colbert re ranker provides a comparison between a retrieval score and a ranker score, for all cases ranker score is lower than retrieval score
- Cohere re-rank contradicts its score with relevancy score, as score difference is huge!
- Rag Fusion allows you to re-rank context from different rewritten queries as well but it decreases the scores when you have rewritten queries, mainly cause of the quality of rewritten query's, can change it to sub queries method too, as the quality from it is concluded to be good
- Rankgpt provides you with new ranked order of contexts like 1,2,3,4,..n
- Long context reorder does not provides scores, also gives window issues when pass to llm for QA, typically because of too long contexts, so k is limited to 3 in this case
- BGE reranker pipeline is customized and ran on t4-GPU, also it gives non normalised scores
- Bert base rerankers have a better answer relevance.
- Colbert re ranker seems to perform better than other rerankers, also provides a differences for estimation.
- Colbert, jina re ranker both standout in their scores, and both of them don't rely on llm for re ranking cross encoders based re ranker are yielding good results than the ones which are based on llm - optimized for a pair (query to document) - you get a

better score, and if your evaluator is checking only top few relevant documents, it will rate cross encoders based re ranker a high score

## Comparing Rerankers response time on GPU and CPU [🔗](#)

Re-ranker	CPU Avg Score	GPU Avg Score
Colbert	44s	60s
Cohere	61s	120s
Rag Fusion	80s	60s
Rank GPT	300s	180s
Jina AI	47s	60s
BGE	-	9s
Long Context Reorder	58s	180s

- Bert based Rerankers took less time on CPU, Bert architecture does involve matrix multiplication as well as layered operations and % GPT based Rerankers took less time on GPU
- **BERT-based models:** Their smaller size, bidirectional nature, and extensive CPU optimizations make them efficient on CPU's.
- **GPT-based models:** Their larger size, auto regressive nature, and the need for extensive parallel processing make them better suited for GPU's.

## Testing Frameworks

**TruLens:** TruLens offers complex pipelines that vary for each metric, making integration into RAG pipelines challenging. While you could build a RAG pipeline already integrated with TruLens, this approach limits the ability to incorporate multiple advanced RAG tools. The accuracy score provided by TruLens is inconsistent, typically fluctuating between 74-76%, which undermines reliability.

**Falcon Evaluate:** Falcon Evaluate provides an extensive array of metrics and is unique in utilizing traditional ML models for evaluation. However, its performance on negative testing is subpar, producing inaccurate results. Additionally, its accuracy scores are erratic, often falling between 40-60%, and if a response is exceptionally good, the score jumps above 90%. It rarely outputs scores between these extremes, indicating a lack of nuance in its evaluations.

**Ragas:** Usually becomes inconsistent, higher values of responses are observed when other frameworks are giving low score for similar response

**DeepEval, Tonic Validate, Uptrain:** These frameworks offer essential metrics for evaluation and are easily integrable into custom pipelines. If ranked by performance and consistency, the order would be:

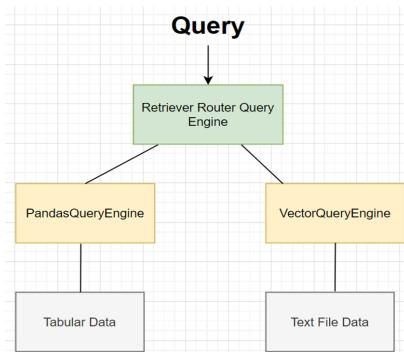
1. Tonic Validate
2. Uptrain
3. Deepeval

The results from these frameworks are not directly comparable due to the use of different models, threshold values, and formulas for calculating metrics.

# RAG over different types of Data

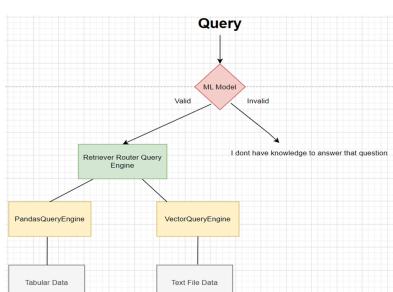
## Key Findings ↗

- Implementing Retrieval Augmented Generation over different types of data, e.g if underlying data contains pdf documents, SQL database, JSON files etc. Exploring how can we create a single RAG over that.
- Llama index provides the concept of multiple query engines, each specifically designed to work with a certain type of data. For example, if you have SQL, JSON, tabular data, then there are different query engines specifically made to handle each kind of data.
- Llama index also provides a Retriever Router Query engine which can be used to route an incoming question to the right query engine.
- The idea was to develop different query engines specific to each data, then plate the RetrieverRouterEngine on top of them and observe whether it routes the question to the correct query engine or not.
- In my example, I built a PandasQueryEngine over a tabular data, a VectorQueryEngine over a pdf data, placed the RetrieverRouterQuery engine on top of them and experimented whether it routes the question to the proper QueryEngine or not.



## Results ↗

- For the most part, the RetrieverRouterQuery engine seemed to be directing the query to the correct QueryEngine.
- An issue that was originating was that the LLM was not restricting its responses to the knowledge base that we had provided. Instead it was generating responses from its own knowledge. To resolve this, we need to adhere to "Strict Prompting"
- Another issue was how to restrict unrelated questions from going down the pipeline, because RetrieverRouterQuery engine tries to fit an incoming question into one of the query engines, so even if a question that is not related to the knowledge about which we have created our RAG system comes in, it will eventually be passed into a query engine which will try to answer it.
- To cater this problem, we can either use an LLM or an ML model to filter out incoming questions, and pass only the relevant one down the pipeline.



## References ↗

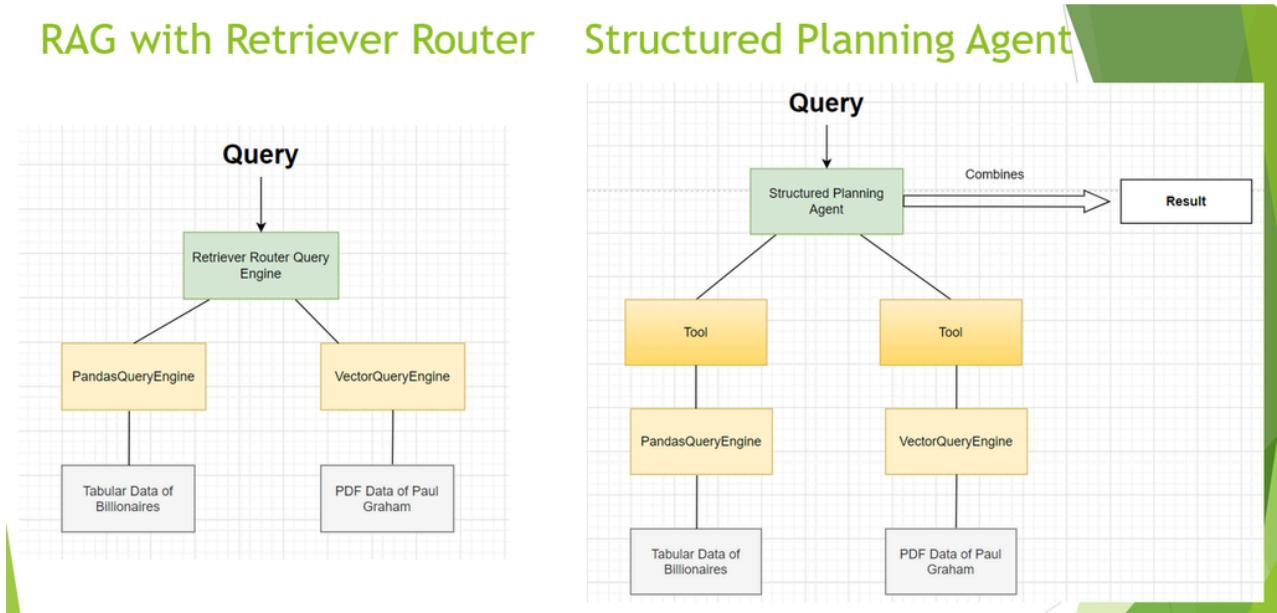
→  [Pandas Query Engine - LlamaIndex](#)

→  [Retriever Router Query Engine - LlamaIndex](#)

## Structured Planning Agent Llama-index

### Structured Planning Agent ↗

- Structured Planning Agent essentially works by resolving the incoming query into a list of sub tasks, executing each sub task through an appropriate tool and then combining the results.



- The above image illustrates the difference between RAG with RetrieverRouterQuery engine and Structured Planning Agent. The Retriever Router analyzes the incoming query and forwards it to **ONE** of the underlying query engines, whereas a Structured Planning Agent break down the incoming query into sub tasks and uses appropriate tools to call relevant underlying query engines to process the sub queries, then combines the results.
- Retriever Router Query Engine is suitable for queries like:
  - “What was the net worth of Bill Gates?”
    - This information is present in the tabular data of billionaires (see image above) and only needs access to that.
  - “Describe the career of Paul Graham.”
    - This information is present in the PDF doc of Paul Graham and only needs access to that.
- On the other hand, Structured Planning Agent is suitable for following queries:
  - “Tell me the net worth of Bill Gates and describe the career of Paul Graham.”
    - One single data source does not contain information to answer the above question. Structured Planning Agent will break down the query into sub tasks:
      - 1) Get net worth of Bill Gates.
      - 2) Get career information regarding Paul Graham.
    - Then use tool1 built on tabular data of billionaires to answer query1 and use tool2 built on PDF data of Paul Graham to answer query2.
    - How does the system know which tool and query engine to use to answer which sub query? We specify that information while building the system. In the image below, while building the tool we associate the relevant query engine with it along with a description about which kind of data it is useful to answer a question from.

```
uber_index = VectorStoreIndex.from_documents(uber_documents)

lyft_tool = QueryEngineTool.from_defaults(
    df_query_engines[0],
    name="billionaires",
    description="Useful for asking questions about billionaires and their net worth.",
)

uber_tool = QueryEngineTool.from_defaults(
    uber_index.as_query_engine(text_qa_template=text_qa_template, refine_template=refine_template),
    name="paul_graham",
    description="Useful for asking questions about paul graham.",
)
```

- After retrieving the results, the Structured Planning Agent combines them and outputs a single answer.

## Key Takeaways

- Structured Planning Agent works well only with a capable LLM. This is because under the hood, a lot of reasoning is happening in order to select the appropriate tools so it is suggested to use the best available llm.
- During experimentation, llama3-70b was not performing well. When GPT4-turbo, which is a better model, was used the results were significantly better.
- Because of agentic reasoning, this approach is slightly more time consuming in yielding answers.

## References

-  Agents - LlamaIndex

# Explore Evaluation Metrics

Giskard:

Automatic evaluation methodology. Covers automatic evaluation as well as manual evaluation by providing human reviewers to evaluate the results provided through automatic evaluation.

Can be used in identifying hallucinations or any kind of harmful responses or legal issues (in case the model generates) Why G-eval and RAGAS produced different results?

G-Eval:

- Coherence: G-Eval is designed to evaluate specific criteria such as coherence through detailed prompts and chain-of-thoughts (CoT). It focuses on how well the generated text maintains a logical and structured flow.
- Prompt-Based Evaluation: The evaluations are driven by a predefined prompt that specifies what aspects of the text to evaluate, such as coherence, conciseness, or grammar. This can make G-Eval sensitive to the specific wording and focus of the prompts.
- Scoring Function: The scoring function in G-Eval uses the probabilities of output tokens from LLMs, aiming to provide fine-grained, continuous scores.

RAGAS:

- Faithfulness: Measures how well the answer is grounded in the given context, which is crucial for ensuring that the information is factually correct and relevant to the retrieved context.
- Answer Relevance: Evaluates if the generated answer addresses the given question appropriately without considering factuality.
- Context Relevance: Assesses the relevance of the retrieved context, focusing on minimizing redundant information and maximizing the usefulness of the context provided to answer the question.

G-Eval: Primarily designed for Natural Language Generation (NLG) tasks, it may weigh criteria like coherence and conciseness more heavily. The auto CoT provides a step-by-step approach that focuses on logical flow and structure.

RAGAS: Designed for Retrieval-Augmented Generation (RAG) systems, it emphasizes the factual consistency of the generated text with the retrieved context (faithfulness), how well the generated answer addresses the question (answer relevance), and the pertinence of the retrieved context (context relevance).

G-Eval: Since it is prompt-based, the quality and detail of the prompt can significantly affect the evaluation

outcome. RAGAS: Utilizes predefined prompts for extracting statements and assessing relevance and faithfulness.

G-Eval: Evaluates context expansion by how well it improves the coherence, structure, and overall quality of the generated text. If context expansion leads to more logically structured and coherent text, G-Eval might score it higher.

RAGAS: Focuses on how context expansion impacts the faithfulness, relevance of the answer, and relevance of the context. If the expanded context improves the factual consistency and relevance without adding redundancy, RAGAS would score it higher.

1. Coherence vs. Faithfulness: G-Eval might score context expansion highly if it improves the logical flow and coherence of the text, even if the expanded context introduces some factual inconsistencies. RAGAS, on the other hand, might penalize the same context expansion if it perceives a decrease in faithfulness or an increase in irrelevant information.
1. Prompt-Based Nuances: The way prompts are constructed in G-Eval can lead to different evaluation outcomes based on the specific criteria emphasized in the prompt. RAGAS's structured approach to extracting and verifying statements might provide a different perspective on the same content.
1. Scoring Granularity: G-Eval's use of token probabilities for scoring might result in more nuanced scores compared to RAGAS's binary verification process, which might lead to discrepancies in the final evaluation.

## TruLens

Use Cases:

- [Detect and Mitigate Hallucination](#)
- [Improve Retrieval Quality](#)
- [Optimize App Configuration](#)
- [Verify the Summarization Quality](#)

Task	Evaluation parameters
Content moderation	Recall and precision on toxicity and bias
Query generation	Correct output syntax and attributes, extracts the right information upon execution
Dialogue (chatbots, summarization, Q&A)	Faithfulness, relevance

Arize AI Phoenix  [Quickstart: Eval](#)  [Phoenix](#)

The standard for evaluating text is human labeling. However, high-quality LLM outputs are becoming cheaper and faster to produce, and human evaluation cannot scale. In this context, evaluating the performance of LLM applications is best tackled by using a separate evaluation LLM. The Phoenix LLM Eval Library is designed for simple, fast, and accurate LLM-based evaluations.

DeepCheck:  [LLM Evaluation Solutions](#)  [Deepchecks](#)

Helps in reducing the risk in LLM pipelines by flagging significant issues and also helps in safeguarding the model from the problem of hallucinations.

It also provides auto-annotation which helps in understanding whether the response is good or bad.

# Evaluation Metrics

Giskard:

Automatic evaluation methodology. Covers automatic evaluation as well as manual evaluation by providing human reviewers to evaluate the results provided through automatic evaluation.

Can be used in identifying hallucinations or any kind of harmful responses or legal issues (in case the model generates) Why G-eval and RAGAS produced different results?

G-Eval:

- Coherence: G-Eval is designed to evaluate specific criteria such as coherence through detailed prompts and chain-of-thoughts (CoT). It focuses on how well the generated text maintains a logical and structured flow.
- Prompt-Based Evaluation: The evaluations are driven by a predefined prompt that specifies what aspects of the text to evaluate, such as coherence, conciseness, or grammar. This can make G-Eval sensitive to the specific wording and focus of the prompts.
- Scoring Function: The scoring function in G-Eval uses the probabilities of output tokens from LLMs, aiming to provide fine-grained, continuous scores.

RAGAS:

- Faithfulness: Measures how well the answer is grounded in the given context, which is crucial for ensuring that the information is factually correct and relevant to the retrieved context.
- Answer Relevance: Evaluates if the generated answer addresses the given question appropriately without considering factuality.
- Context Relevance: Assesses the relevance of the retrieved context, focusing on minimizing redundant information and maximizing the usefulness of the context provided to answer the question.

G-Eval: Primarily designed for Natural Language Generation (NLG) tasks, it may weigh criteria like coherence and conciseness more heavily. The auto CoT provides a step-by-step approach that focuses on logical flow and structure.

RAGAS: Designed for Retrieval-Augmented Generation (RAG) systems, it emphasizes the factual consistency of the generated text with the retrieved context (faithfulness), how well the generated answer addresses the question (answer relevance), and the pertinence of the retrieved context (context relevance).

G-Eval: Since it is prompt-based, the quality and detail of the prompt can significantly affect the evaluation

outcome. RAGAS: Utilizes predefined prompts for extracting statements and assessing relevance and faithfulness.

G-Eval: Evaluates context expansion by how well it improves the coherence, structure, and overall quality of the generated text. If context expansion leads to more logically structured and coherent text, G-Eval might score it higher.

RAGAS: Focuses on how context expansion impacts the faithfulness, relevance of the answer, and relevance of the context. If the expanded context improves the factual consistency and relevance without adding redundancy, RAGAS would score it higher.

1. Coherence vs. Faithfulness: G-Eval might score context expansion highly if it improves the logical flow and coherence of the text, even if the expanded context introduces some factual inconsistencies. RAGAS, on the other hand, might penalize the same context expansion if it perceives a decrease in faithfulness or an increase in irrelevant information.
1. Prompt-Based Nuances: The way prompts are constructed in G-Eval can lead to different evaluation outcomes based on the specific criteria emphasized in the prompt. RAGAS's structured approach to extracting and verifying statements might provide a different perspective on the same content.
1. Scoring Granularity: G-Eval's use of token probabilities for scoring might result in more nuanced scores compared to RAGAS's binary verification process, which might lead to discrepancies in the final evaluation.

## TruLens

Use Cases:

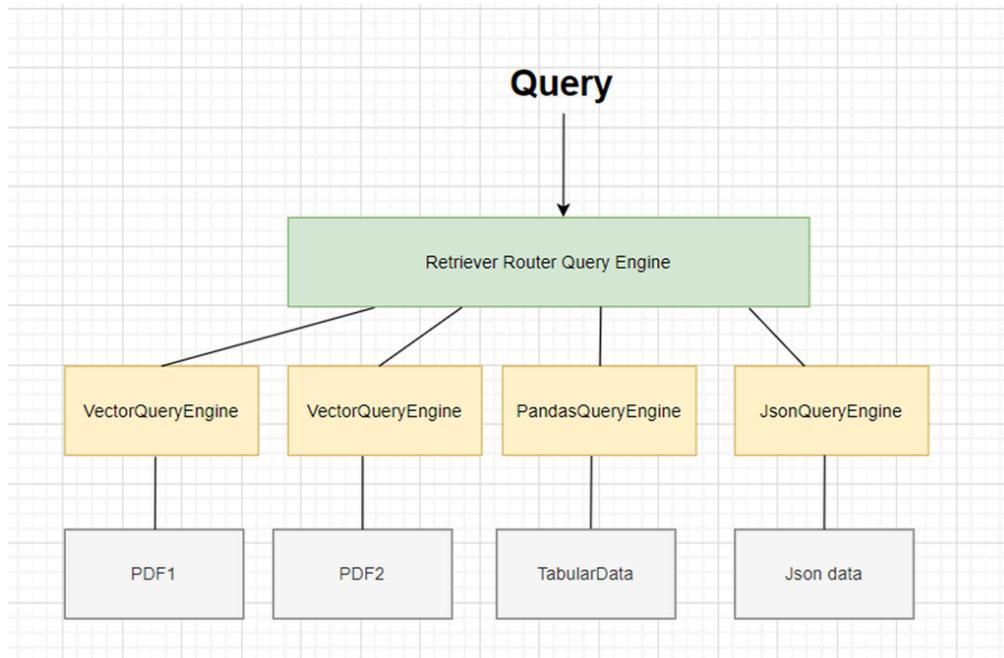
- [Detect and Mitigate Hallucination](#)
- [Improve Retrieval Quality](#)
- [Optimize App Configuration](#)
- [Verify the Summarization Quality](#)

Task	Evaluation parameters
Content moderation	Recall and precision on toxicity and bias
Query generation	Correct output syntax and attributes, extracts the right information upon execution
Dialogue (chatbots, summarization, Q&A)	Faithfulness, relevance

## Q/A with RetrieverRouter engine from llama-index

### Key Points

- Using the llama-index retriever router query engine to observe its efficacy in routing the incoming question to the correct query engine below.



- The image above shows the experimental setup.
- PDF1 contains information regarding Emumba Provident fund policy. PDF2 is related to Titanic. Tabular data consists of a table which has names, net worths and nationalities of top billionaires. Json data contains dummy data related to employees.
- Each kind of data has a respective query engine from llama-index built on top of it.
- Finally, the Retriever Router Query engine is used to route the incoming question to the correct query engine.

### Results

- The results achieved were not very promising.
  - The system was routing the question to the incorrect query engine at times.
  - It was forcing to answer the question using one of the underlying query engines despite the question being out of context.
  - It does not have the ability to break down the question and use multiple query engines to answer parts of it, if that is required.
- Primarily, it is most evident that Retriever Router Query engine is designed for systems built on the same underlying data but different types of indexes, rather than systems which are built on different types of data.

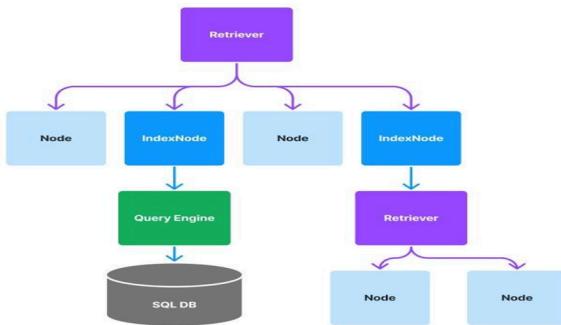
### References

- [Retriever Router Query Engine - LlamaIndex](#)

# Recursive Retrieval Techniques

## Key Takeaways

- Recursive retrieval is beneficial when dealing with hierarchical data or unstructured forms of data, for example data in a tabular form,
- Has the concept of IndexNode which contains NodeData+underlying query engine.
- During retrieval, if an IndexNode is retrieved, the underlying query engine is called which gets executed.



## – QueryEngines:

The concept of query engines is interesting. In general, we've observed that LLMS are not very effective at answering questions from structured forms of data (e.g a table) effectively. Consider an example table containing information regarding billionaires. We want to answer the question "How many people are from India and France" using the data of the following table. (Image1→ Table , Image2→ Query and Response)

No.	Name	Net worth\n(USD)	Age	Nationality	Primary source(s) of wealth
0	Bernard Arnault & Family	\$233 billion	75	France	LVMH
1	Elon Musk	\$195 billion	52	South Africa\\nCanada\\nUnited States	Tesla, SpaceX, Twitter (Currently USA)
2	Jeff Bezos	\$194 billion	60	United States	Amazon
3	Mark Zuckerberg	\$177 billion	39	United States	Meta Platforms
4	Larry Ellison	\$141 billion	79	United States	Oracle Corporation
5	Warren Buffett	\$133 billion	93	United States	Berkshire Hathaway
6	Bill Gates	\$128 billion	68	United States	Microsoft
7	Steve Ballmer	\$121 billion	68	United States	Microsoft
8	Mukesh Ambani	\$116 billion	65	India	Reliance Industries
9	Larry Page	\$114 billion	51	United States	Google

```
response = df.query_engines[0].query(  
    "How many people are from France and India ?"  
)  
print(str(response))  
response  
2  
Response(response='2', source_nodes=[], metadata={'pandas_instruction_str': '(df["Nationality"].str.contains("France|India")).sum()', 'raw_pandas_output': '2'})
```

Now inputting the content of the table and asking the LLM this question directly does not yield good results. LLMS are not great at manoeuvring around this tabular data.

Solution ? : LlamaIndex query engines !!

In this specific case we can use pandasQueryEngine from LlamaIndex. Under the hood, it inputs the dataframe column names and asks it to write python code to extract the information (rather than giving a direct answer). This yields significantly better result than asking the LLM to answer the question directly.

→ Similarly, there are specific query engines available for each specialised task (JSONQueryEngine for JSON related data, SQLEngine for SQL).

References

- [!\[\]\(b7ed91f8ab588d8294f01a601f473704\_img.jpg\) Recursive Retriever + Query Engine Demo - LlamaIndex](#)
- [!\[\]\(73849138de72555a3d50b7b250d294bd\_img.jpg\) Advanced RAG: Recursive Retrieval with LlamaIndex](#)
- [!\[\]\(36986448f272b46b3a5aecdb08ae4383\_img.jpg\) Jerry Liu on Twitter / X](#)

[Jerry Liu on Twitter / X](#)

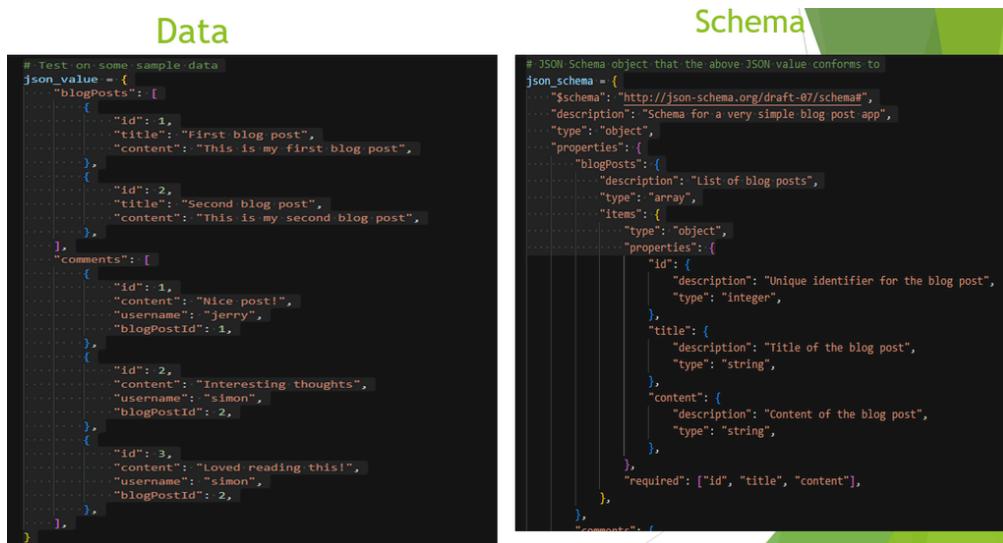
# Exploring JSON query engines of llama-index

## Key Points

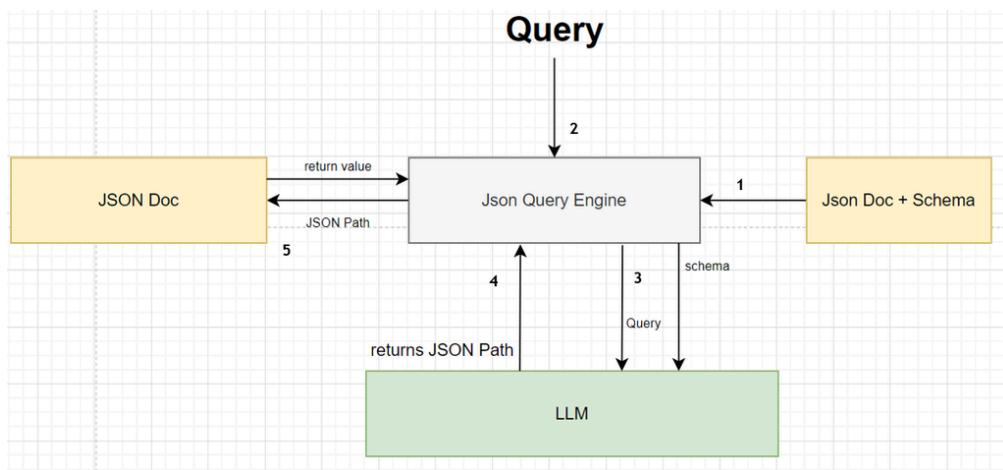
- Llama-index provides two kinds of query engines to query JSON data. ( JSON Query Engine , JSONAnalyze )

## JSON Query Engine

- It is ideally used over a nested JSON data object with a JSON schema as shown in the picture below.



- In the image below, we can understand how does the JSON query engine works under the hood.



- 1) We create the query engine by providing it with the schema and the data
- 2) At runtime, a query is sent to the query engine to be processed. (e.g What did Jerry comment ?)
- 3) The JSON Query engine passes the query and the JSON Schema to the LLM and asks it to generate a valid JSON Path.
- 4) LLM returns a valid JSON Path according to the question provided. (e.g \$.comments[?(a.username=='jerry')])

- \* 5) JSON Query engine executes that JSON Path on the JSON doc to retrieve the answer of the question. (e.g the answer to the above question : "Nice Post!")

**Results:**

- For simple, straightforward questions that involved answering only a single piece of information, the JSON query engine worked fine. (e.g Question: Who commented "Loved reading this!"?)

```

query = "Who commented Loved reading this!?"

nl_response = nl_query_engine.query(
    query,
)
raw_response = raw_query_engine.query(
    query,
)
[4]   ✓ 7.4s

print("Natural Language Response: ", nl_response)
print("Raw Response: ", raw_response)
[5]   ✓ 0.0s
...
Natural Language Response:  The user 'simon' commented 'Loved reading this!'.
Raw Response:  {"username": "simon"}

```

- For slightly more complex questions, where multiple pieces of information need to be fetched from the JSON doc and an 'and' or an 'or' condition is required under the hood, the JSON Query engine doesn't respond well. See image below. "What did jerry and simon comment?" was asked.

```

query = "What did jerry and simon comment?"

nl_response = nl_query_engine.query(
    query,
)
raw_response = raw_query_engine.query(
    query,
)
[6]   ✘ 3.6s

-----
JsonPathParserError                                     Traceback (most recent call last)
File c:\Users\Emumba\AppData\Local\Programs\Python\Python312\Lib\site-packages\llama_index\core\indices\src\indexer\jsonpath_indexer.py:173, in parse(self, path, debug=False)
    172     return ExtentedJsonPathParser(debug=debug).parse(path)
--> 173     def parse(path, debug=False):
    174         lexer = self.lexer_class()
    175         tokens = lexer.tokenize(path)
    176         parser = self.parser_class(tokens=tokens)
    177         return parser.parse()

File c:\Users\Emumba\AppData\Local\Programs\Python\Python312\Lib\site-packages\jsonpath_ng\parser.py:38, in parse(self, path, debug=False)
    37     lexer = lexer or self.lexer_class()
--> 38     return self.parse_token_stream(lexer.tokenize(string))

File c:\Users\Emumba\AppData\Local\Programs\Python\Python312\Lib\site-packages\jsonpath_ng\parser.py:62, in parse_token_stream(self, tokens)
    61     if tokens:
--> 62         new_parser = ply.yacc.yacc(module=self,
    63             debug=self.debug,
    64             tabmodule = parsing_table_module,
    65             start = start_symbol,
    66             errorlog = logger)
--> 67         return new_parser.parse(tokens=tokens)

File c:\Users\Emumba\AppData\Local\Programs\Python\Python312\Lib\site-packages\ply\yacc.py:333, in LRParser.parse(self, tokens)
    332     if self.errorlog:
--> 333         try:
    334             self.yacc_parse(tokens=tokens)
    335         except Exception as exc:
    336             self.errorlog.error(str(exc))
--> 337             raise ValueError(f"Invalid JSON Path: {expression}") from exc
    338     else:
    339         return self.yacc_parse(tokens=tokens)

ValueError: Invalid JSON Path: $.comments[?(@.username=='jerry' || @.username=='simon')].content
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings.

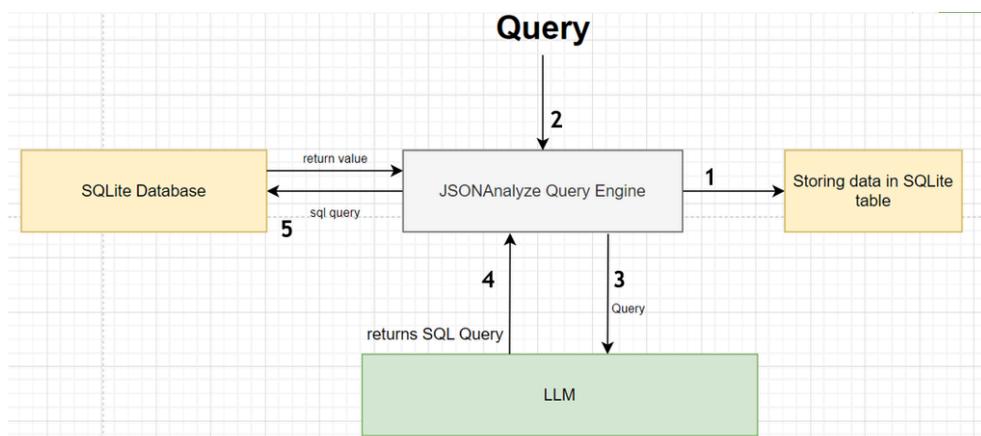
```

## JSONAnalyze Query Engine

- JSONAnalyze Query engine is provided by llama-index to query over typical JSON data that is retrieved from an api(see example in image below). JSONAnalyze is helpful in answering statistical questions over this JSON data.

```
json_list = [
    {
        "name": "John Doe",
        "age": 25,
        "major": "Computer Science",
        "email": "john.doe@example.com",
        "address": "123 Main St",
        "city": "New York",
        "state": "NY",
        "country": "USA",
        "phone": "+1 123-456-7890",
        "occupation": "Software Engineer",
    },
    {
        "name": "Jane Smith",
        "age": 30,
        "major": "Business Administration",
        "email": "jane.smith@example.com",
        "address": "456 Elm St",
        "city": "San Francisco",
        "state": "CA",
        "country": "USA",
        "phone": "+1 234-567-8901",
        "occupation": "Marketing Manager",
    },
    {
        "name": "Michael Johnson",
        "age": 35,
        "major": "Finance",
        "email": "michael.johnson@example.com",
        "address": "789 Oak Ave",
    }
]
```

- In the image below, we can see how JSONAnalyze query engine works under the hood.



- \* 1) We create the query engine by providing it with JSON Data and it stores that data in an SQLite database.
- \* 2) At runtime, a query is sent to the query engine to be processed. (e.g What is the distribution of ages among the individuals ?)
- \* 3) The JSONAnalyze Query engine passes the query and the SQL database schema to the LLM and asks it to generate a valid SQL Query.
- \* 4) LLM returns a SQL Query according to the question provided.
- \* 5) JSONAnalyze Query engine executes that SQL query on the SQLite table to retrieve the answer of the question.

#### Results:

- For most of the questions that conform to the fields and available data in the JSON object (which is converted to an SQLite table under the hood), JSONAnalyze query engine performs well. In the image below, it was able to answer all the these questions without giving an error.

```

questions = [
    "What is the average age of the individuals in the dataset?",
    "What is the maximum age among the individuals?",
    "What is the minimum age among the individuals?",
    "How many individuals have a major in Psychology?",
    "What is the most common major among the individuals?",
    "What is the percentage of individuals residing in California (CA)?",
    "How many individuals have an occupation related to science or engineering?",
    "What is the average length of the email addresses in the dataset?",
    "How many individuals have a phone number starting with '+1 234'?",
    "What is the distribution of ages among the individuals?",
]

# print(json_stats_query_engine.query(questions[-1]))
print(json_stats_query_engine.query(""))

# print(query_engine.query(questions[0]))
✓ 3.5s

The distribution of ages among the individuals is as follows:
- Age 25: 1 individual
- Age 26: 1 individual
- Age 27: 2 individuals
- Age 28: 3 individuals
- Age 29: 2 individuals
- Age 30: 2 individuals
- Age 31: 1 individual
- Age 32: 2 individuals
- Age 33: 1 individual
- Age 35: 1 individual

```

- But for questions that contain any kind of information that is not present in the JSON, JSONAnalyze query engine will through an error.

```
> ^
# print(json_stats_query_engine.query(questions[-1]))
print(json_stats_query_engine.query("How many people are from company 'abcdef'?"))

# print(query_engine.query(questions[0]))

11] ⑧ 3.1s

-- Error executing query: SELECT COUNT(*)
FROM items
WHERE company = 'abcdef';

-----
OperationalError                                     Traceback (most recent call last)
File c:\Users\Emumba\AppData\Local\Programs\Python\Python312\Lib\site-packages\llama_index\co
105 try:
106     # Execute the SQL query
--> 107     results = list(db.query(sql_query))
108 except sqlite_utils.utils.sqlite3.OperationalError as exc:

File c:\Users\Emumba\AppData\Local\Programs\Python\Python312\Lib\site-packages\sqlite_utils\d
496 """
497 Execute ``sql`` and return an iterable of dictionaries representing each row.
498 (
...
501     parameters, or a dictionary for ``where id = :id``
502 """
--> 503 cursor = self.execute(sql, params or tuple())
504 keys = [d[0] for d in cursor.description]

File c:\Users\Emumba\AppData\Local\Programs\Python\Python312\Lib\site-packages\sqlite_utils\d
520 if parameters is not None:
--> 521     return self.conn.execute(sql, parameters)
522 else:

OperationalError: no such column: company
...
109     print_text(f"Error executing query: {sql_query}")
--> 110     raise ValueError("Invalid query") from exc
112 return sql_query, table_schema, results

ValueError: Invalid query
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

In the image above, the question mentions company name, so the LLM generates an SQL query containing a filter on company name, but the original data doesn't contain any field corresponding to a company name, hence when the JSONAnalyze query engine executes this SQL query on the SQLite database, it throws an error.

## References

- \* [JSON Query Engine - LlamaIndex](#)
- \* [JSONAnalyze Query Engine - LlamaIndex](#)

# Capture Explanations and Compare Our RAG Mindmap

Reference: [10 Challenges in Building RAG-Based LLM Applications -](#)

## YouTube Customized Models:

- Techniques: Fine-tuning, modifying, transfer learning, etc.

## **Alternate Approach:**

- Turn inference problem into retrieval plus inference problem aka semantic search.
- Utilize traditional information retrieval to provide context.

## **Stage One (Data Ingestion):**

### 1. Data Ingestion:

- Store data efficiently for retrieval.
- Break down into chunks for manageable processing.
- Convert documents into vector representations using an encoder.
- Store embedding vectors in a vector database.

### 2. Challenges:

- Data quality issues (garbage data, sensitive data, etc.).
- Inaccurate data ingestion (e.g., tables, diagrams, different formats).
- Loss of continuity and high-level context due to document chunking.
- Dependency on embedding model quality.
- Unambiguous or incomplete user queries.
- Incorrect document retrieval and utilization (hallucination).

## **Data Ingestion:**

### 1. Data Preprocessing:

- Data loading and quality checks.

### 2. Chunking:

- Strategies: Document-based, semantic, recursive.
- Selection depends on knowledge base nature, model capabilities, and task.

### 3. Embeddings.

### 4. Scalable Data Ingestion:

- Consider scalability and planning.
- Data cleaning for large datasets.

### 5. Indexing in Vector Database.

## **Pre-Retrieval:**

- Querying Challenges:
  - Handling ambiguous queries.
- Solution:
  - Query optimisation techniques like
    - Query transformations
    - Sub query decomposition
    - Step back prompting

- Multi query retrieval

- Multi step retrieval
- Query rewriting
- Query Routing.

#### **Retrieval:**

- Challenges:
  - Too many or too few chunks.
  - Missing context, irrelevant chunks, query ambiguity.
  - Semantic ambiguity, vector similarity issues.
  - Retrieval optimizations.
- Solutions:
  - Hybrid search
  - Metadata integration
  - Sentence window retrieval
  - Hypothetical document embedding
  - Parent child chunk retrieval
  - Hierarchical index retrieval
  - Multi document agents
  - Chat engine context
  - Chat engine condense plus context

#### **Problem: Retrieval of Irrelevant Chunks.**

- Post-Retrieval:
  - Reranking to address information overload and hallucination.
- Solutions:
  - Utilize re-rankers like cross encoder reranking, cohore, private reranking APIs.

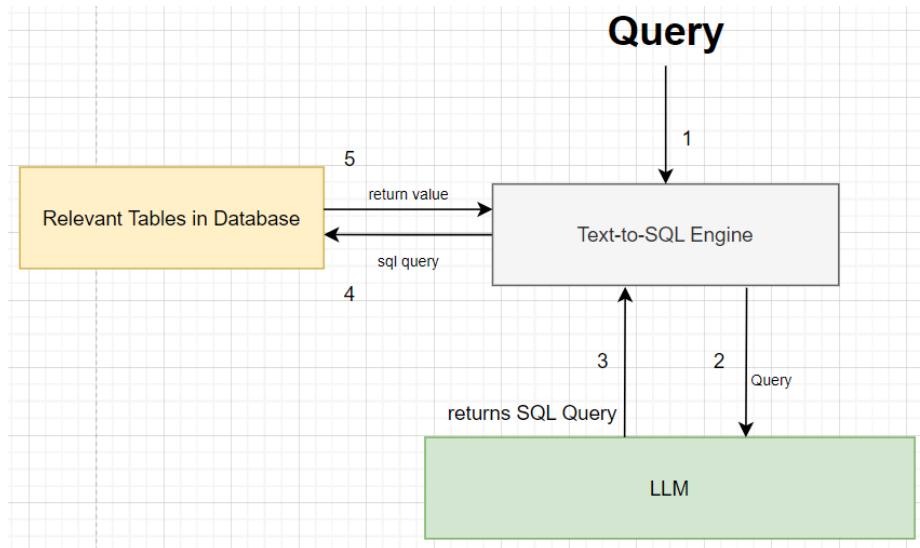
#### **Generation:**

- Problems:
  - Information overload, insufficient context window.
  - Domain adaptation, chaotic contexts.
  - Structuring retrieved information, hallucination, irrelevant responses.
  - Logically incoherent responses.
- Solutions:
  - Information compression
  - Generator fine-tuning
  - Adapter methods
  - Throat of thought
  - Chain of note
  - Chain of verification
  - React (reason + act)
  - Expert prompting.

# Exploring text-to-SQL Query engine of llama-index

## Key Points

- Llama-index text-to-SQL Query engine is designed to answer questions over SQL data effectively.
- It follows the following pipeline (which other techniques of q/a over structured data follow as well)



1. The user's question, and the schema of the table are sent to the LLM.
2. LLM is prompted to generates an SQL query to answer the user's question.
3. LLM generates an SQL query to be run.
4. The generated SQL query is executed on the relevant table.
5. The returned value is formatted into a natural language response.

## References

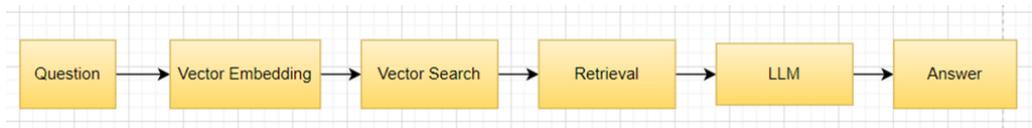
- [!\[\]\(49a5459d3253479473c2fe465ccd81ef\_img.jpg\) Text-to-SQL Guide \(Query Engine + Retriever\) - LlamaIndex](#)

# Q/A over unstructured vs structured data

## Key Points

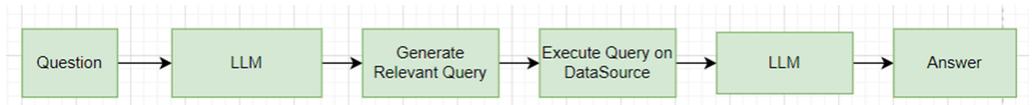
- Lately, two different techniques and pipelines are being used in order to perform Q/A over structured and unstructured data. They utilise the power of LLMs in different ways.

## Unstructured Data



- Chat with unstructured data (e.g PDFs, word files etc) is done through standard Retrieval Augmented Generation. The pipeline is shown in the above image.
- The user's question is converted into a vector embedding (using the same embedding model through which the document chunks were converted into embeddings).
- There's a similarity search (e.g cosine similarity) between the question's vector and vectors of all chunks in the vector database.
- The most similar chunks are retrieved from the vector database
- The embeddings of the question and the chunks are converted back into their textual forms and then fed into an LLM to answer the question.

## Structured Data



- Chat with structured data (Tabular data, SQL data, JSON) does not involve the traditional RAG pipeline because, fundamentally, its underlying workings do not favour Q/A over structured data.
- The idea is that rather than asking the LLM to directly answer a question from structured data, we ask the LLM to write a data-specific query and then execute that query separately on the data.

For example, if we have an SQL table with columns "Countries,Population" and we have a question "What is the total Population of the top three most populous countries ?"

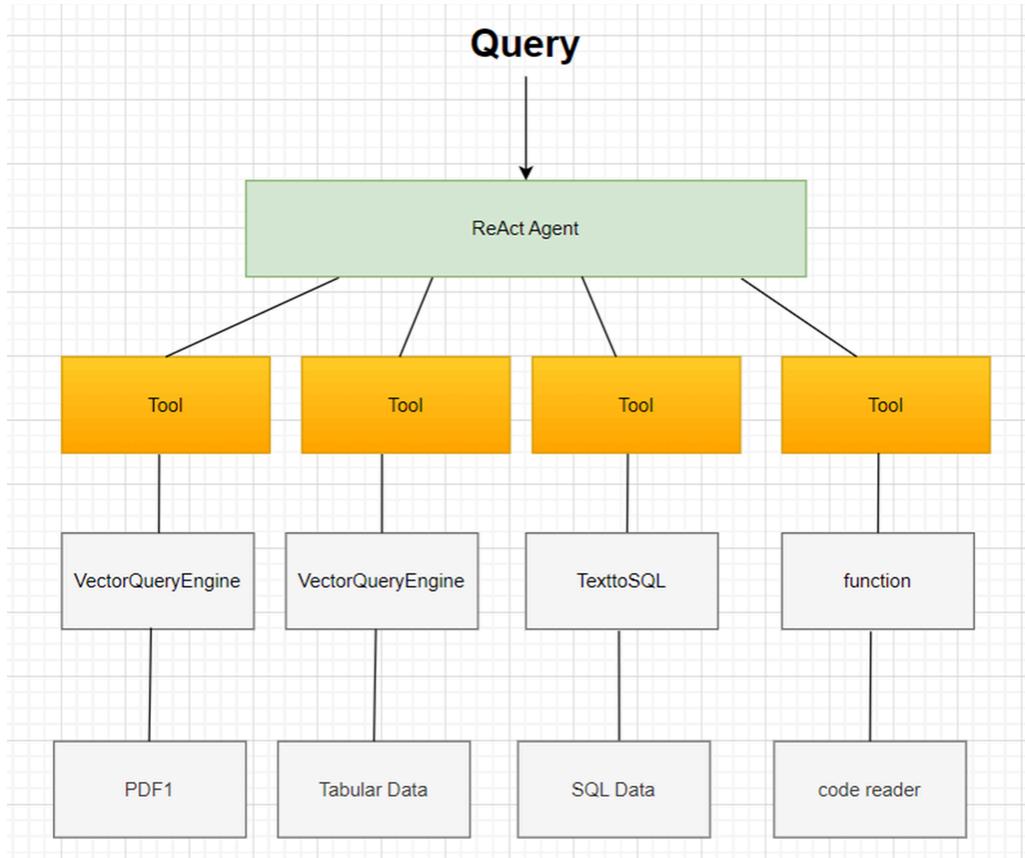
Now rather than asking the LLM this question directly, we ask the LLM to generate an SQL query to answer the question, and we execute that query separately in the table to get the answer.

- The image above illustrates this flow.
- There is
  - no conversion of data into chunks
  - no storage of chunks' embeddings in vector databases
  - no conversion of runtime question into vector embedding
  - no similarity index matching and top-k retrieval from database
- All of the above mentioned steps are part of standard RAG which performs well on unstructured data.

## Q/A with ReAct agent for routing

### Key Points

- Using the ReAct agent as a query routing entity. Its main purpose is to route the incoming question from the user to the correct underlying tool/function.
- Of all the query routing mechanisms tested, ReAct gives the most promising results.



- The image above shows the setup.
- A vector query engine built on PDF. This is a RAG pipeline. (PDF data is about Emumba's policies mentioned on internal site)
- A vector query engine built on Tabular data (Emumba's info regarding OPD centers).
- A text-to-sql query engine built on SQL data. (Dummy data containing a table with countries and their population information and another table containing data about employees).
- A code-reader function which reads code from a file. Is added to the agent to test whether Q/A works alongside agentic functionality.

### Results

- The results were were promising.
- The following images illustrate correct responses from the above system.

(Verbose==True is set so that we can see how the ReAct agent is

reasoning)

Prompt: What is the percentage of provident fund provided by emumba ?  
Thought: The user is asking about the provident fund percentage provided by Emumba. I should use the perks\_and\_benefits tool to get this information.  
Action: perks\_and\_benefits  
Action Input: {'input': 'provident fund'}  
Observation: The Provident Fund is a savings plan for all permanent employees, allowing them to save a portion of their salary each month. Both the employer and employee contribute 7.5% of the gross salary to this fund. Employees become eligible for this benefit upon the confirmation of their probation completion. If an employee resigns within the first year, they only receive their own contribution. After one year, they are eligible for 50% of the employer's contribution, after two years, 75% of the employer's contribution, and after three years, 100% of the employer's contribution. Employees who contribute to a recognized Provident Fund also receive tax exemption benefits. To enroll in the Provident Fund benefit program, employees need to fill out a PF Consent Form and submit it to the Finance Manager. However, due to a government policy, the Provident Fund is currently not being recognized.  
Thought: I can answer without using any more tools. I'll use the user's language to answer.  
Answer: Emumba provides a Provident Fund for all permanent employees, where both the employer and the employee contribute 7.5% of the gross salary each month. However, due to a government policy, the Provident Fund is currently not being recognized.  
Emumba provides a Provident Fund for all permanent employees, where both the employer and the employee contribute 7.5% of the gross salary each month. However, due to a government policy, the Provident Fund is currently not being recognized.  
Emumba provides a Provident Fund where both the employer and the employee contribute 7.5% of the gross salary.  
Prompt: Does emumba provide lunch ?  
Thought: The user is asking about the perks and benefits provided by Emumba. I should use the perks\_and\_benefits tool to answer this question.  
Action: perks\_and\_benefits  
Action Input: {'input': 'Does emumba provide Lunch ?'}  
Observation: Yes, Emumba provides Lunch. They offer a wide variety of tasty, high-quality meals at a subsidized price. The menu changes every three months and includes a variety of Desi food cuisines. If an employee wishes to enjoy lunch, they need to inform HR by sending an email.  
Thought: I can answer without using any more tools. I'll use the user's language to answer.  
Answer: Yes, Emumba provides Lunch. They offer a wide variety of tasty, high-quality meals at a subsidized price. The menu changes every three months and includes a variety of Desi food cuisines. If an employee wishes to enjoy lunch, they need to inform HR by sending an email.  
Yes, Emumba provides Lunch. They offer a wide variety of tasty, high-quality meals at a subsidized price. The menu changes every three months and includes a variety of Desi food cuisines. If an employee wishes to enjoy lunch, they need to inform HR by sending an email.

Prompt: Where is the Hostel of Emumba located ?  
Thought: The user is asking about the location of Emumba's hostel. I should use the "perks\_and\_benefits" tool to get this information.  
Action: perks\_and\_benefits  
Action Input: {'input': 'hostel location'}  
Observation: The context does not provide information about the location of the hostel.  
Thought: The tool did not provide the specific location of the hostel. I cannot answer the question with the provided tools.  
Answer: I'm sorry, but I don't have the specific location of Emumba's hostel. Please contact Emumba directly for this information.  
I'm sorry, but I don't have the specific location of Emumba's hostel. Please contact Emumba directly for this information.  
Prompt: Who should I contact for the information regarding Emumba Hostel ?

Prompt: Who should I contact for the information regarding Emumba Hostel ?  
Thought: The user is asking about the contact for information regarding Emumba's hostel. I should use the perks\_and\_benefits tool to get this information.  
Action: perks\_and\_benefits  
Action Input: {'input': 'Emumba Hostel contact'}  
Observation: The contact for the Emumba hostel is Mr. Kashif Nabeel. You can reach him via email at kashif.nabeel@emumba.com.  
Thought: I can answer without using any more tools. I'll use the user's language to answer.  
Answer: The contact for the Emumba hostel is Mr. Kashif Nabeel. You can reach him via email at kashif.nabeel@emumba.com.  
The contact for the Emumba hostel is Mr. Kashif Nabeel. You can reach him via email at kashif.nabeel@emumba.com.  
Prompt: Where is the host of Emumba ?  
Thought: The user is asking about the host of Emumba. I should use the perks\_and\_benefits tool to get this information.  
Action: perks\_and\_benefits

Prompt: What is the cost of the hostel facility ?  
Thought: The user is asking about the cost of the hostel facility. This information can be obtained from the perks\_and\_benefits tool.  
Action: perks\_and\_benefits  
Action Input: {'input': 'hostel facility cost'}  
Observation: The cost of the hostel facility is a subsidized monthly contribution of PKR 4K per person. This cost is exclusive of utility bills and meals.  
Thought: I can answer without using any more tools. I'll use the user's language to answer.  
Answer: The cost of the hostel facility is a subsidized monthly contribution of PKR 4K per person. Please note that this cost does not include utility bills and meals.  
The cost of the hostel facility is a subsidized monthly contribution of PKR 4K per person. Please note that this cost does not include utility bills and meals.  
Prompt:

Correctly routes all of the above to PDF Query engine

(Verbose==False for the following)

**Prompt: what are the total number of cities in the database ?**

**There are a total of 4 cities in the database.**

Correctly routes to text-to-sql

**Prompt: what are those cities ?**

Some of the cities are Chicago in the United States, Seoul in South Korea, Tokyo in Japan, and Toronto in Canada.

Correctly routes to text-to-sql

**Prompt: what is the population of Tokyo ?**

The population of Tokyo is approximately 13.96 million people.

Correctly routes to text-to-sql

Prompt: What is the population of Karachi ?

I'm sorry, but I don't have the specific population information for Karachi.

routes to text-to-sql and correctly answers the question (there no info regarding Karachi in the database)

Prompt: What are the total number of expenses that can be reimbursed at Emumba and can interns avail the provident fund policy ?  
Thought: The user's question is about the reimbursement policy and provident fund policy at Emumba. I should use the 'policies\_and\_guidelines' tool to get information about the reimbursement policy and the 'perks\_and\_benefits' tool to get information about the provident fund policy.  
Action: policies\_and\_guidelines  
Action Input: {'input': 'reimbursement policy'}  
Observation: The reimbursement policy at Emumba allows employees to claim expenses incurred while conducting business activities. These expenses can be related to client meetings, office supplies runs, or any other costs generated as a representative for Emumba. The reimbursement process involves filling in a Reimbursement Form, attaching the expense receipt along with the bill, signing the form, and submitting it to the Finance and Operations Managers. Reimbursable expenses fall under four categories: meals, travel expenses, training and certifications, and software subscriptions. The reimbursement is paid to employees along with their next salary and is not included in the calculation of their regular pay.  
Thought: I have obtained information about the reimbursement policy at Emumba. Now, I need to use the 'perks\_and\_benefits' tool to get information about the provident fund policy.  
Action: perks\_and\_benefits  
Action Input: {'input': 'provident fund policy'}  
Observation: The Provident Fund is a savings plan for all permanent employees, allowing them to save a portion of their salary each month. Both the employer and the employee contribute 7.5% of the gross salary to the fund. Employees become eligible for this benefit upon confirmation of their probation completion. If an employee resigns within the first year, they only receive their own contribution. After one year, they are eligible for 50% of the employer's contribution, 75% after two years, and 100% after three years. Employees who contribute to a recognized Provident Fund also receive tax exemption benefits. To enroll in the Provident Fund benefit program, employees must fill out a PF Consent Form and submit it to the Finance Manager. However, due to a government policy, the Provident Fund is currently not being recognized. Upon withdrawal, no tax is imposed on the employee's contribution, but the employer's contribution and 50% of the profit are added to the employee's annual taxable income.  
Thought: I have obtained information about the provident fund policy at Emumba. The provident fund policy is applicable to permanent employees and not interns. The reimbursement policy includes four categories: meals, travel expenses, training and certifications, and software subscriptions. Now, I can answer the user's question without using any more tools.  
Answer: At Emumba, there are four categories of expenses that can be reimbursed: meals, travel expenses, training and certifications, and software subscriptions . As for the provident fund policy, it is a benefit for permanent employees and does not apply to interns.  
At Emumba, there are four categories of expenses that can be reimbursed: meals, travel expenses, training and certifications, and software subscriptions. As for the provident fund policy, it is a benefit for permanent employees and does not apply to interns.

This question involves retrieving information from two query engines.

Answer: The Eiffel Tower is located in Paris, France.  
The Eiffel Tower is located in Paris, France.  
Prompt: where is the eiffel tower located ?  
Thought: This question is about a general knowledge fact and doesn't require any of the tools I have. I can answer this directly.  
Answer: The Eiffel Tower is located in Paris, France.  
The Eiffel Tower is located in Paris, France.  
Prompt: Traceback (most recent call last):  
File "d:\Projects\multi-agent RAG\main.py", line 91, in <module>  
    prompt = input('Prompt: ')  
              ~~~~~  
KeyboardInterrupt  
PS D:\Projects\multi-agent RAG> & C:/Users/Emumba/AppData/Local/Programs/Python/Python312/python.exe "d:/Projects/multi-agent RAG/main.py"  
Prompt: where is the eiffel tower located ?  
Thought: This question is not related to the information that the tools provide. I can't use the tools to answer this question.  
Answer: I'm sorry, but I can't assist with that.  
I'm sorry, but I can't assist with that.  
Prompt: ■

Prompt: who is joe biden ?  
Thought: The question is about a general knowledge topic that is not related to the information that the tools provide. I should not use my own general knowledge to answer this question.  
Answer: I'm sorry, but I can only provide information related to the tools I have which are api\_documentation, perks\_and\_benefits, and policies\_and\_guidelines.  
I'm sorry, but I can only provide information related to the tools I have which are api\_documentation, perks\_and\_benefits, and policies\_and\_guidelines.  
Prompt: ■

The above images show how we can stop the system from answering question from its own knowledge by improving system prompts.

Prompt: read the test.py file present in the data directory and generate a summary of what it is about. Also tell me how many male and female employees are there ?  
The 'test.py' file is a simple Flask application that provides a RESTful API for a list of items stored in memory. It supports the basic CRUD operations: Create, Read, Update, and Delete. As for the number of employees, there are 6 female employees and 4 male employees in the company.

This shows how the agent handles a task which requires it to read some file and also involves fetching data from the database.

- All in all, the ReAct agent performs well for routing the incoming question/task to the correct underlying tool.
- It also provides us with the ability to combine function calling with Q/A systems.

# RetrieverRouterQueryEngine vs StructuredPlanningAgent vs ReAct for query routing

## Key Points

- Comparing the performance of different techniques used as routing entities in order to route the incoming query to the correct query engine.

## Retriever Router Query Engine

- Works well only with different indexes (e.g VectorIndex, SummaryIndex) built on top of the same underlying data.
- Doesnt break down a query into sub tasks and doesnt execute part of a query on one query engine and another sub task on another.
- Forces itself to select a query engine to answer a question, even when it is out of context.

## ReAct agent

- Gives good performance among all techniques tested for Query Routing.
- Correctly routes the incoming query to the appropriate tool/query engine.
- If there's no available tool to handle the query, it doesn't forcefully select something and send the question to it.
- Performs well at breaking down complex tasks into subtasks and utilising different tools to solve the problem or answer the question.
- Allows the integration of function calling ability within the Q/A system.
- Easier to control LLMs tendency of answering question from its own inherent knowledge

## Structured Planning Agent

- A kind of a wrapper provided by llama-index to enhance the ability of simple RoutingEngine and add agentic reasoning.
- Allows the user to select the agent of their own choice and embed with the other components of the pipeline. (tools, functions etc).

## Conclusion

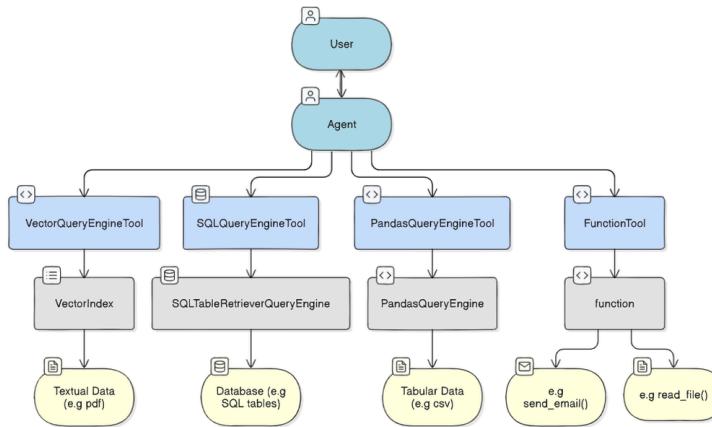
- ReAct agent works the best and allows greater integration of Q/A system with function calling using tools and performing agentic reasoning to breakdown complex tasks and questions into sub tasks and then solving them.

## Mini-Assistant : Combining Q/A and function calling tools with ReAct on top for agentic reasoning

This small project acts like a Mini-Assistant that can chat with databases, PDFs, excel files, and can also perform actions like reading code files and sending emails.

### Setup

Broadly speaking, there are different tools developed to perform specific functionality and a reasoning engine on top that selects which tools to use in order to achieve a task.



### Tools :

- Tool built which chats with PDFs. It contains a RAG pipeline. In this case, the RAG pipeline is built using data present on Emumba's internal site. (CareersAndGrowth, ClubsAndActivities, PerksAndBenefits etc)
- Tool for chatting with SQL database. It is built using llama-index SQLTableRetrieverQueryEngine. In this case, theres a dummy SQL database containing an employees\_info table and city\_population table.
- Tool for chatting with tabular data using PandasQuery engine.
- Tool for reading files.
- Tool for sending an email.

### Agent :

- ReAct agent is used on top which gets the task from user, analyzes which tools would be best utilized for performing which part of the task and then executes the whole task.

### Testing

The System was given different tasks to perform as shown below. The end result as well as the reasoning both are shown.

**Task :** Read the contents of the file named 'tools.py', generate a brief summary of what is about and send that summary as an email to 'shazilahmed445@gmail.com' Tools Utilized → read\_file , send\_email

*Thought: The user wants to read the contents of a file named 'tools.py', generate a summary, and send that summary as an email. I need to use the 'code\_reader' tool to read the file, then use a summarization tool to generate a summary, and finally use the 'send\_email' tool to send the email.*  
Action: code\_reader  
Action Input: {'file\_name': 'tools.py'}

*Observation: 'file\_content': 'from dotenv import load\_dotenv\nfrom llama\_index.llms.openai import OpenAI\nimport os\nfrom llama\_index.core import (\n VectorStoreIndex,\n SimpleDirectoryReader,\n)\nfrom llama\_index.core import Settings, SQLDatabase\nfrom llama\_index.core import QueryEngineTool, ToolMetadata\nfrom llama\_index.experiments.\nsql\_engine import PandasQueryEngine\nfrom llama\_index.readers.file import PyMuPDFReader\nimport camelot\nfrom typing import List\nimport pandas as pd\nimport PyMuPDF\nimport text-d\nSQL\nfrom sqlalchemy import (\n create\_engine,\n MetaData,\n Table,\n Column,\n String,\n Integer,\n VARCHAR,\n Boolean,\n select,\n insert,\n)'*

*Thought: The 'tools.py' file appears to be a Python script that imports various modules and creates several tools for different purposes. These tools seem to be used for reading and querying data from different sources such as PDFs and databases. The tools are created using various libraries such as OpenAI, llama\_index, PyMuPDF, and camelot. The script also sets up a sqlite database and inserts data into it. Now, I will use the 'send\_email' tool to send this summary to the provided email address.*  
Action: send\_email  
Action Input: {'receiver\_email': 'shazilahmed445@gmail.com', 'message': "The 'tools.py' file is a Python script that imports various modules and creates several tools for different purposes. These tools seem to be used for reading and querying data from different sources such as PDFs and databases. The tools are created using various libraries such as OpenAI, llama\_index, PyMuPDF, and camelot. The script also sets up a sqlite database and inserts data into it."}  
Observation: None  
Thought: The email has been sent successfully as there is no error message. I can now confirm this to the user.

The contents of the file 'tools.py' have been summarized and sent to the email address 'shazilahmed445@gmail.com'.

**Task :** How much referral bonus does emumba provide ? Structure this information in a formal email and send it to 'shazilahmed445@gmail.com'. Tools Utilized → ChatPDF , send\_email

*Thought: The current language of the user is English. I need to use the referral\_incentive tool to get the information about the referral bonus provided by Emumba. Then, I will use the email tool to send the information to the provided email address.*

Action: referral\_incentive

Action Input: { 'input': 'referral bonus' }

Observation: The referral bonus at Emumba is awarded when a referred candidate is hired. If an employee's referred candidate is hired for a Mid-Level position, the employee will receive a referral bonus of PKR 50,000. If the referred candidate is hired for a Senior Level position, the referral bonus increases to PKR 100,000. The bonus will be transferred once the referred candidate joins Emumba and it will be paid along with the employee's salary.

*Thought: Now that I have the information about the referral bonus, I can structure this information into a formal email and send it to the provided email address using the send\_email tool.*

Action: send\_email

Action Input: { 'receiver\_email': 'shazilahmed445@gmail.com', 'message': 'Subject: Emumba Referral Bonus Information\nDear Recipient,\n\nI hope this email finds you well. I am writing to inform you about the referral bonus provided by Emumba. The referral bonus is awarded when a referred candidate is hired. If an employee's referred candidate is hired for a mid-level position, the employee will receive a referral bonus of PKR 50,000. If the referred candidate is hired for a senior level position, the referral bonus increases to PKR 100,000. The bonus will be transferred once the referred candidate joins Emumba and it will be paid along with the employee's salary.\n\nBest regards,\n[Your Name]' }

Observation: None

*Thought: I can answer without using any more tools. I'll use the user's language to answer.*

Answer: The email containing the information about the referral bonus provided by Emumba has been sent to "shazilahmed445@gmail.com".

The email containing the information about the referral bonus provided by Emumba has been sent to "shazilahmed445@gmail.com".

**Task :** What is the population of Tokyo ?

Tools Utilized → ChatDB

*Thought: The user is asking for the population of Tokyo. I can use the 'database\_info' tool to get this information.*

Action: database\_info

Action Input: { 'input': 'Tokyo' }

Observation: Tokyo, the capital city of Japan, has a population of approximately 13.96 million people.

*Thought: I can answer without using any more tools. I'll use the user's language to answer.*

Answer: The population of Tokyo is approximately 13.96 million people.

The population of Tokyo is approximately 13.96 million people.

**Task :** How many total employees are there in the employees\_info table ?

Tools Utilized → ChatDB

*Thought: The user is asking about the total number of employees in the employees\_info table. I can use the 'database\_info' tool to get this information.*

Action: database\_info

Action Input: { 'input': 'SELECT COUNT(\*) FROM employees\_info' }

Observation: There are a total of 10 employees in the employees\_info table.

*Thought: I can answer without using any more tools. I'll use the user's language to answer.*

Answer: There are a total of 10 employees in the employees\_info table.

There are a total of 10 employees in the employees\_info table.

**Task :** Briefly describe emumba policy regarding lunch as well as employee referrals

Tools Utilized → ChatPDF

Agent: Briefly describe emumba policy regarding lunch as well as employee referrals

*Thought: The current language of the user is English. I need to use the perks\_and\_benefits tool to get information about the lunch policy and the referral\_incentive tool to get information about the employee referrals policy.*

Action: perks\_and\_benefits

Action Input: { 'input': 'lunch' }

Observation: The office provides a wide variety of tasty, high-quality lunch at a subsidized price. The average cost for a month subscription to the office mess has been approximately PKR 1,200 per person, which is 40% of the total cost of a month's lunch. This amount is deducted from your salary at the end of every month. The menu includes a variety of Desi food cuisines, offering a new flavoursome dish every day.

*Thought: Now that I have the information about the lunch policy, I need to use the referral\_incentive tool to get information about the employee referrals policy.*

Action: referral\_incentive

Action Input: { 'input': 'referral' }

Observation: Referral at Emumba is a process where existing employees recommend potential candidates for open positions in the company. This policy applies to all employees, whether full-time or part-time. The referred candidates should have relevant work experience. The referring employee's role is to identify potential candidates and refer them to the company. Referral bonuses do not apply to individuals who have previously approached the company or are already part of the hiring pipeline. The referral bonus will be transferred once the referred candidate joins Emumba and will be paid along with the employee's salary. The referral bonus for a mid-level position is PKR 50,000/-, while for a senior-level position, it is PKR 100,000/-.

*Thought: I can answer without using any more tools. I'll use the user's language to answer.*

Answer: Emumba provides a wide variety of tasty, high-quality lunch at a subsidized price. The average cost for a month subscription to the office mess is approximately PKR 1,200 per person, which is 40% of the total cost of a month's lunch. This amount is deducted from your salary at the end of every month. The menu includes a variety of best food cuisines, offering a new flavoursome dish every day.

As for employee referrals, Emumba has a policy where existing employees can recommend potential candidates for open positions in the company. This policy applies to all employees, whether full-time or part-time. The referred candidates should have relevant work experience. The referring employee's role is to identify potential candidates and refer them to the company. Referral bonuses do not apply to individuals who have previously approached the company or are already part of the hiring pipeline. The referral bonus will be transferred once the referred candidate joins Emumba and will be paid along with the employee's salary. The referral bonus for a mid-level position is PKR 50,000/-, while for a senior-level position, it is PKR 100,000/-.

Emumba provides a wide variety of tasty, high-quality lunch at a subsidized price. The average cost for a month subscription to the office mess is approximately PKR 1,200 per person, which is 40% of the total cost of a month's lunch. This amount is deducted from your salary at the end of every month. The menu includes a variety of Desi food cuisines, offering a new flavoursome dish every day.

As for employee referrals, Emumba has a policy where existing employees can recommend potential candidates for open positions in the company. This policy applies to all employees, whether full-time or part-time. The referred candidates should have relevant work experience. The referring employee's role is to identify potential candidates and refer them to the company. Referral bonuses do not apply to individuals who have previously approached the company or are already part of the hiring pipeline. The referral bonus will be transferred once the referred candidate joins Emumba and will be paid along with the employee's salary. The referral bonus for a mid-level position is PKR 50,000/-, while for a senior-level position, it is PKR 100,000/-.

**Task :** List down the perks and benefits provided by emumba, structure this output in a formal email and send it to

'shazilahmed445@gmail.com'

Tools Utilized → ChatPDF , send\_email

*Thought: The current language of the user is English. I need to use the perks\_and\_benefits tool to get information about the lunch policy and the referral\_incentive tool to get information about the employee referrals policy.*

Action: perks\_and\_benefits

Action Input: { 'input': 'lunch' }

Observation: The office provides a wide variety of tasty, high-quality lunch at a subsidized price. The average cost for a month subscription to the office mess has been approximately PKR 1,200 per person, which is 40% of the total cost of a month's lunch. This amount is deducted from your salary at the end of every month. The menu includes a variety of Desi food cuisines, offering a new flavoursome dish every day.

*Thought: Now that I have the information about the lunch policy, I need to use the referral\_incentive tool to get information about the employee referrals policy.*

Action: referral\_incentive

Action Input: { 'input': 'referral' }

Observation: A referral at Emumba is a process where existing employees recommend potential candidates for open positions in the company. This policy applies to all employees, whether full-time or part-time. The referred candidates should have relevant work experience. The referring employee's role is to identify potential candidates and refer them to the company. Referral bonuses do not apply to individuals who have previously approached the company or are already part of the hiring pipeline. The referral bonus will be transferred once the referred candidate joins Emumba and will be paid along with the employee's salary. The referral bonus for a mid-level position is PKR 50,000/-, while for a senior-level position, it is PKR 100,000/-.

*Thought: I can answer without using any more tools. I'll use the user's language to answer.*

Answer: Emumba provides a wide variety of tasty, high-quality lunch at a subsidized price. The average cost for a month subscription to the office mess is approximately PKR 1,200 per person, which is 40% of the total cost of a month's lunch. This amount is deducted from your salary at the end of every month. The menu includes a variety of Desi food cuisines, offering a new flavoursome dish every day.

As for employee referrals, Emumba has a policy where existing employees can recommend potential candidates for open positions in the company. This policy applies to all employees, whether full-time or part-time. The referred candidates should have relevant work experience. The referring employee's role is to identify potential candidates and refer them to the company. Referral bonuses do not apply to individuals who have previously approached the company or are already part of the hiring pipeline. The referral bonus will be transferred once the referred candidate joins Emumba and will be paid along with the employee's salary. The referral bonus for a mid-level position is PKR 50,000/-, while for a senior-level position, it is PKR 100,000/-.

*Thought: Now that I have the information about the perks and benefits provided by Emumba, I need to structure this information in a formal email and send it to 'shazilahmed445@gmail.com' using the 'send\_email' tool.*

*Action: send\_email*

*Action Input: ('receiver\_email': 'shazilahmed445@gmail.com', 'message': 'Subject: Perks and Benefits at Emumba\nDear Recipient,\n\nI hope this email finds you well. Here are the perks and benefits provided by Emumba:\n1. Provident Fund: A saving plan for all permanent employees, where both the employer and employee contribute 7.5% of the gross salary. The employee becomes eligible for this benefit upon confirmation of their probation completion. The employee can withdraw their own contribution if they resign within the first year. After one year, they can get 50% of the employer's contribution, 75% after two years, and 100% after three years. Employees who contribute to this fund also receive tax exemption benefits.\n\n2. Health Benefits: These cover room, OT and ICU, physician, surgeon and anesthetist fees, diagnostic tests including MRI and CT scan, medicines required during hospital stay, kidney dialysis and cancer treatment, organ transplant, endoscopy and angioplasty, oxygen and blood supplies, day care procedures, emergency treatment, and pre & post-hospitalization OPD coverage which covers medication & lab tests (up to 30 days).\n\n3. Maternity Benefits: With a limit of PKR 120,000 for normal delivery and PKR 200,000 for complicated delivery. This covers medical expenses related to childbirth, pregnancy and prenatal treatments, gynecologist fees for delivery and consultation, labor room/OT charges, medicines required during hospital stay, diagnostic tests, baby's nursing care while mother is in hospital, circumcision of baby boys, and pre & postnatal expenses.\n\n4. Discounts: Employees can also avail discounts up to 25% on different labs, OPD and dental treatments.\n\n5. Life Insurance: Upon the death of the employee due to natural causes, PKR 500,000 will be payable to their family, and if the death is due to an accidental cause, PKR 1,000,000 will be payable to their family.\n\nBest regards,*

*Observation: None*

*Thought: The email has been sent successfully as there is no error message. I can answer without using any more tools.*

*Answer: The email listing down the perks and benefits provided by Emumba has been sent to 'shazilahmed445@gmail.com'.*

*The email listing down the perks and benefits provided by Emumba has been sent to 'shazilahmed445@gmail.com'.*

**Task :** In the database, how many cities have a population greater than 3 million. Also tell me how often are emumba annual reviews conducted

**Tools Utilized** → ChatDB , ChatPDF

*Thought: The current language of the user is English. I need to use the 'database\_info' tool to find out how many cities have a population greater than 3 million. Then, I will use the 'Performance\_appraisal' tool to find out how often Emumba's annual reviews are conducted.*

*Action: database\_info*

*Action Input: ('input': 'cities with population greater than 3 million')*

*Observation: Two cities with a population greater than 3 million are Tokyo with a population of 13,968,000 and Seoul with a population of 9,776,000.*

*Thought: I have the information about the cities with a population greater than 3 million. Now, I need to use the 'Performance\_appraisal' tool to find out how often Emumba's annual reviews are conducted.*

*Action: Performance\_appraisal*

*Action Input: ('input': 'annual review')*

*Observation: Annual reviews at Emumba are conducted twice every year, hence they are referred to as bi-annual reviews. These reviews are formal meetings between an employee and their manager to discuss the employee's performance over the past six months, identify areas for improvement, and set goals for the next six months. The reviews are part of a 360-degree feedback cycle, meaning feedback is received from managers, co-workers, and peers. This provides a comprehensive view of the employee's performance, skills, and competencies. The reviews also offer a chance to uncover areas that may need development or expose gaps in knowledge. After the review, the employee has the choice to incorporate the feedback or not.*

*Thought: I can answer without using any more tools. I'll use the user's language to answer.*

*Answer: In the database, there are two cities with a population greater than 3 million: Tokyo with a population of 13,968,000 and Seoul with a population of 9,776,000. As for Emumba, annual reviews are conducted twice every year, hence they are referred to as bi-annual reviews. These reviews are part of a 360-degree feedback cycle and offer a chance to uncover areas that may need development or expose gaps in knowledge.*

**Task :** In the database, how many cities have a population greater than 3 million. Also tell me how often are emumba annual reviews conducted ? Structure the information in formal email and send it to 'shazilahmed445@gmail.com'

**Tools Utilized** → ChatDB , ChatPDF , send\_email

*Thought: The current language of the user is English. I need to use the 'database\_info' tool to get the information about the cities with a population greater than 3 million. Then, I need to use the 'Performance\_appraisal' tool to get information about the frequency of Emumba's annual reviews. Finally, I will use the 'send\_email' tool to send the information to the provided email address.*

*Action: database\_info*

*Action Input: ('input': 'cities with population greater than 3 million')*

*Observation: The cities with a population greater than 3 million are Tokyo with a population of 13,968,000 and Seoul with a population of 9,776,000.*

*Thought: I have the information about the cities with a population greater than 3 million. Now, I need to use the 'Performance\_appraisal' tool to get information about the frequency of Emumba's annual reviews.*

*Action: Performance\_appraisal*

*Action Input: ('input': 'frequency of annual review')*

*Observation: The annual reviews at Emumba are conducted twice every year.*

*Thought: I have the information about the frequency of Emumba's annual reviews. Now, I need to structure this information in a formal email and send it to 'shazilahmed445@gmail.com' using the 'send\_email' tool.*

*Action: send\_email*

*Action Input: ('receiver\_email': 'shazilahmed445@gmail.com', 'message': 'Dear Recipient,\n\nI hope this email finds you well. I am writing to provide you with the information you requested.\n\nIn our database, the cities with a population greater than 3 million are Tokyo with a population of 13,968,000 and Seoul with a population of 9,776,000.\n\nAs for the annual reviews at Emumba, they are conducted twice every year.\n\nPlease let me know if you need any further information.\n\nBest regards,\n[Assistant]*

## Compare Table Formats & Different Table Types

# GraphRAG

Paper Link: <https://arxiv.org/pdf/2404.16130.pdf>

Github Repository: <https://github.com/microsoft/graphrag>

GraphRAG Flowchart: <https://tinyurl.com/2s4ytcu>

## Limitations of Basic RAG:

- Baseline RAG struggles to connect the dots.
- Baseline RAG performs poorly when being asked to holistically understand summarized semantic concepts over large data collections or even singular large documents.

## GraphRAG Pipeline:

- **Source documents:** Deciding how finely to split input texts from source documents into chunks for processing, balancing efficiency with the quality of information extraction.
- **Text chunks:** Extracting smaller or larger portions of text impacts the efficiency and accuracy of extracting entity references, balancing between recall and precision in the extraction process.
- **Element instances:** Identifying and extracting instances of graph nodes and edges from text chunks using tailored prompts, including domain-specific examples for enhanced accuracy.
- **Element Summaries:** Using an LLM to generate descriptive summaries of entities, relationships, and claims extracted from texts, ensuring coherence despite variations in entity references.
- **Graph communities:** Creating an undirected graph where nodes represent entities and edges their relationships, employing community detection algorithms like Leiden to reveal hierarchical structures.
- **Community summaries:** Generating summaries of node communities in hierarchical levels, aiding in understanding dataset themes and structures, essential for query-focused summarization.
- **Community answers:** Generating intermediate answers from community summaries for user queries, evaluating and integrating them based on relevance and helpfulness scores.
- **Global answer:** Compiling a final answer to a user query from the integrated community answers, ensuring comprehensive coverage and relevance.

## Datasets:

Podcasts: Compiled transcripts of podcast conversations between Kevin Scott, Microsoft CTO, and other technology leaders (Behind the Tech, Scott, 2024). Size:  $1669 \times 600$ -token text chunks, with 100-token overlaps between chunks (~1 million tokens)

News Articles: Benchmark dataset comprising news articles published from September 2013 to December 2023 in a range of categories, including entertainment, business, sports, technology, health, and science (MultiHop-RAG; Tang and Yang, 2024). Size:  $3197 \times 600$ -token text chunks, with 100-token overlaps between chunks (~1.7 million tokens).

## Community Levels:

Four levels of communities are included.

- CO: Uses root-level community summaries (fewest in number) to answer user queries.

- C1: Uses high-level community summaries to answer queries. These are sub-communities of C0, if present, otherwise C0 communities projected down.
- C2: Uses intermediate-level community summaries to answer queries. These are subcommunities of C1, if present, otherwise C1 communities projected down.
- C3: Uses low-level community summaries (greatest in number) to answer queries. These are sub-communities of C2, if present, otherwise C2 communities projected down.

## **Evaluation Metrics:**

- Comprehensiveness.
- Diversity.
- Empowerment.
- Directness.

## **Use cases of GraphRAG**

1- Critical Information Discovery

2- Analysis

3- Incident Management

4- Health care

Link:<https://www.linkedin.com/pulse/enhancing-intelligent-applications-using-graphrag-suresh-mandava-pgk3c/>

## Fine Tuning Embedding's

Problem Statement : .....

### EXPERIMENT : 1

Initial Dataset : [11 PDF](#)

[documents](#) ↗ Training Dataset : [7](#)

[PDFs](#) ↗

Testing Dataset : [4 PDFs](#) ↗

Queries / Corpus Set for Training : [343](#) ↗

(generated by splitting text/chunks from pdfs, sending them to gpt35 to get Queries for that specific chunk)

Queries / Corpus Set for Testing : [520](#) ↗

(generated by splitting text/chunks from pdfs, sending them to gpt35 to get Queries for that specific chunk)

Number of Top k retrieved context = [5](#) ↗

Comparisons ↗

Embedding ↗	Output Dimension ↗	Training Parameters ↗	Hit Rate ↗
text-embedding-ad-a-002 ( <a href="#">openAI</a> )	1,536 ↗	350 million ↗	92.6 % ↗
BAAI/bge-small-en	384 ↗	20-100 million ↗	88.8% ↗
Fine tuned BAAI/bge-small-en	384 ↗	20-100 million ↗	92.2% ↗
BAAI/bge-m3 ↗	1,024 ↗	up to 500 million ↗	94.2% ↗

Fine tuned BAAI/bge-m3 <small>🔗</small>	<b>1,024</b> <small>🔗</small>	<b>up to 500 million</b> <small>🔗</small>	<b>99.4%</b> <small>🔗</small>
-----------------------------------------------	-------------------------------	------------------------------------------------------	-------------------------------

## EXPERIMENT : 2

Initial Dataset : 132 PDF documents (Extreme AI Expert) dataset

Training Dataset : 90 PDFs

Testing Dataset : 42 PDFs

Queries / Corpus Set for Training :

5522 Queries / Corpus Set for

Testing : 5970 Number of Top k

retrieved context = 10

Comparisons

Embedding	Output Dimension	Training Parameters	Hit Rate
text-embedding-ad-a-002 (openAI)	1,536	350 million	62.3 %
BAAI/bge-small-en	384	20-100 million	44.5%
Fine tuned BAAI/bge-small-en	384	20-100 million	69%
BAAI/bge-m3	1,024	up to 500 million	58%
Fine tuned BAAI/bge-m3	1,024	up to 500 million	-

## EXPERIMENT : 3A

Following Experiment used the data from Genie Research and POCs

wiki. 3A - Dataset has a mix of plain and tabular data targeted

Queries / Corpus Set for Training : **Less than 100**

Queries / Corpus Set for Testing : **Less than**

**100** Number of Top k retrieved context = **10**

Comparisons

Embedding	Output Dimension	Training Parameters	Local Hit Rate	Fine Tuned Hit Rate
ada-002 (openAI)	1,536	350 million	75%	-
bge-small-en	384	33 million	72%	86%
bge-m3	1024	up to 500 million	76%	92%
Stella	8192	435 million	79%	-

## EXPERIMENT : 3B

Following Experiment used the data from Genie Research and POCs

wiki. 3A - Dataset has only tabular data targeted

Queries / Corpus Set for Training : **Less than 100**

Queries / Corpus Set for Testing : **Less than**

**100** Number of Top k retrieved context = **10**

Comparisons

Embedding ↗	Output Dimension ↗	Training Parameters ↗	Local Hit Rate ↗	Fine Tuned Hit Rate ↗
ada-002 (openAI) ↗	1,536 ↗	350 million ↗	66% ↗	- ↗
bge-small-en ↗	384 ↗	33 million ↗	59% ↗	81% ↗
bge-m3 ↗	1024 ↗	up to 500 million ↗	65% ↗	84% ↗
Stella ↗	8192 ↗	435 million ↗	68% ↗	-

- Do a better documentation
- For tabular data, improve chunking, attach proper metadata ( Like topic, titles etc)
- Convert tables into some proper format like json, textual format

## CONCLUSION SO FAR: ↗

1. Fine tuning does increase the retrieval hit rate ↗
2. Models with more training parameters offer better hit rate ↗
3. Dimensions in a vector space shall not be either very high subjected to over fitting nor very low subjected to under fitting ↗
4. Models performances depends upon data semantics ↗
5. Tabular data decreases the Hit Rate ↗
6. We can not rely on LLama index's QA embedding function, It generates queries some times that do not make any sense, plus they hallucinate a lot. ↗

# Embeddings Course 1

COURSE : 1

 Embedding Models: from Architecture to Implementation -  
DeepLearning.AI

## Introduction to Embeddings:

- Embedding models create vectors that enable semantic or meaning-based retrieval systems.

## Key Concepts:

### 1. Embedding Vectors:

- Represent words, phrases, sentences, or other entities as points in a vector space.
- Similar points in vector space have similar semantic meanings.

### 2. Word Embeddings:

- Word2Vec (Google Brain): Predicts a word based on context words.
- GloVe (Stanford University): Simplified mathematical approach, increases context window using recurrent neural networks and transformers.

### 3. Transformers and BERT:

- Transformers : Efficient feed-forward neural networks for sequential data.
- BERT : Deep transformer networks trained on fill-in-the-blank tasks for deep language understanding, enabling advanced NLP applications.

## Applications of Embeddings:

- Text embeddings for words, phrases, sentences, images, videos, and audio clips.
- Used in LLMs, semantic search, product recommendations, anomaly detection, etc.

## Retrieval Systems:

- **Cross Encoder:**
- Concatenates question and answer, computes relevance.
- Accurate but computationally intensive.

- **Sentence Embeddings:**

- Embeds text segments into vectors for fast similarity search.
- Less accurate than cross encoders but faster and scalable.

## **Lesson 1: Word and Sentence Embeddings**

- **Vector Embeddings:**

- Map entities to vectors in space.
- Dense vector representations capture semantic meaning.

- **Applications:**

- Text, images, audio clips using models like CLIP (OpenAI).
- Semantic search, product recommendations, anomaly detection.

- **Sentence Embeddings:**

- Apply word embedding principles to sentences.
- Use in RAG (Retrieval-Augmented Generation) pipelines for query and response embeddings.

## **Lesson 2: Contextualized Token Embeddings**

- **Problem with Word Embeddings:**

- Lack of context sensitivity.

- **Transformers:**

- Introduced in 2017, solve context problems.
- BERT model: Encoder-only transformer for contextualized embeddings.
- Pretrained on Masked Language Modeling (MLM) and Next Sentence Prediction (NSP).
- Effective in tasks like classification, NER, question answering.

## **Lesson 3: Sentence Embeddings Development**

- **Tokenization:**

- Subword tokenizers (BPE, WordPiece, SentencePiece) used in transformers.

- **Initial Attempts:**

- Mean pooling of token embeddings or using CLS token failed.

- **Successful Approach:**

- Dual encoder architecture with contrastive loss.
- Trains separate encoders for questions and answers.
- Use in sentence similarity and RAG

applications. (**Fine tuning**)

 <https://github.com/meshalJcheema/Dual-encoder-Train> Connect your GitHub account

## **Lesson 4: Building and Training Dual Encoders**

- **Architecture:**

- Two independent BERT encoders for questions and answers.
- Uses CLS token embeddings, contrastive loss to optimize similarity.

- **Training Process:**

- Cross entropy loss to match diagonal similarity scores.
- Training loop iterates over batches and epochs to optimize encoders.

- **Practical Example:**

- Small model trained with PyTorch.
- Demonstrates the importance of adequate data and epochs for effective training.

## Lesson 5: Using Embeddings in Production

- **RAG Pipeline:**

- Encode text chunks with answer encoder, store in vector database.
- Query encoder generates query embedding, retrieves matching chunks.
- Use ANN algorithms (HNSW, Annoy, FAZE) for efficient retrieval.

- **ANN Implementation:**

- Approximate nearest neighbor searches for fast and accurate retrieval.

## Conclusions:

### 1. Effectiveness of Embeddings:

- Word embeddings (Word2Vec, GloVe) capture semantic meaning but lack context.
- Contextualized embeddings (BERT) address context but need sophisticated architectures for full sentences.

### 2. Dual Encoder Advantage:

- Separate encoders for queries and responses enhance relevance in RAG systems.
- Contrastive loss ensures semantic matching, making it suitable for real-world applications.

### 3. Practical Implementation:

- ANN algorithms are essential for scaling embedding-based retrieval.
- Two-stage retrieval (retrieve and re-rank) combines speed and accuracy, leveraging both sentence embeddings and cross encoders.

### 4. Future Directions:

- Embedding models are crucial for advanced NLP tasks, including hybrid and multimodal search systems.

**What is a key characteristic of vector embeddings in natural language processing?**

Points in vector space that are similar to each other have similar semantic meaning.

**Which model developed by OpenAI aligns images and text in a shared representation space?**

CLIP

**What is a limitation of using a cross encoder in a RAG**

**pipeline?** It is very slow and does not scale well.

**What is the main problem with traditional word embedding models like Word2vec?**

They do not understand the context of words in a sentence.

**What are the two tasks BERT was pre-trained on?**

Masked Language Modeling and Next Sentence Prediction

**What does a cross-encoder do in the context of a retrieval pipeline?**

Generates new sentences from a given text.

**What is the main issue with using mean pooling for sentence**

**embeddings?** It fails to integrate the context of the entire sentence.

**What is the primary purpose of using a dual encoder architecture in sentence embedding**

**models?** To separate the processing of questions and answers using two independent BERT

encoders.

**What role does contrastive loss play in the dual encoder architecture?**

It ensures that similar pairs of data points are closer in the embedding space and dissimilar pairs are further apart.

**What are the two types of encoders used in the dual encoder architecture for sentence embedding**

**models?** Question Encoder and Answer Encoder

# Adapter over embeddings

## Problem Statement [🔗](#)

Fine-tuning embedding models directly on specific datasets can be resource-intensive and time-consuming, especially when dealing with large document corpus. This is because fine-tuning often requires re-embedding all documents, which can be costly.

Building an adapter over embedding's adapter linear—a simple neural network layer like a 1-layer or 2-layer neural network—addresses this challenge by transforming query embedding's into a new latent space optimized for retrieval over specific data and queries. By applying this linear transformation only to the query embedding's and keeping the document embedding's fixed, we can improve retrieval performance without the overhead of re-embedding the entire document corpus.

## Results [🔗](#)

### Experiment : 1 [🔗](#)

Dataset : Extreme Dataset [🔗](#)

Embedding Model [BAAI/bge-small-en](#) [🔗](#)

Total Queries : 5970 [🔗](#)

K=10 [🔗](#)

Embedding's <a href="#">🔗</a>	Baseline Hit Rate <a href="#">🔗</a>	Fine tuned Hit rate <a href="#">🔗</a>	Liner Adapter <a href="#">🔗</a>	2 layer NN Adapter <a href="#">🔗</a>
BAAI/bge - small-en	44% <a href="#">🔗</a>	62% <a href="#">🔗</a>	52% <a href="#">🔗</a>	50% <a href="#">🔗</a>

### Experiment : 2 [🔗](#)

Dataset : Confluence Dataset [🔗](#)

Embedding Model [BAAI/bge-en](#) [🔗](#)

Total Queries : Less than 100 [🔗](#)

K=10 [🔗](#)

Embedding's	Baseline Hit Rate	Fine tuned Hit rate	Linear Adapter	2 layer NN Adapter
BAAI/bge - m3	76% ↗	92% ↗	79% ↗	75% ↗

## Conclusion: ↗

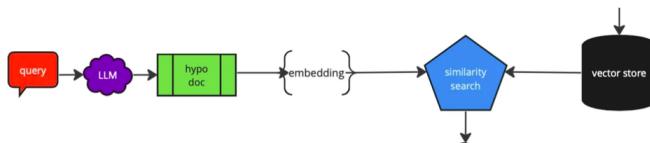
- 1) It is cost wise cheap as no Need to Re-Embed Documents and offers a Lightweight Training ↗
- 2) As noted in experimental results, the performance improvement from using a linear adapter is modest. Fine-tuning the full embedding model generally yields a more significant performance boost because it allows for comprehensive adjustments across both queries and documents. ↗

## AUTO-HYDE

### Limitations of HyDE

HyDE (Hypothetical Document Embeddings) is designed to improve document retrieval but has some notable limitations:

- Model Dependence:** Different language models used in HyDE result in significant variance in retrieval performance. This suggests that choosing the right model is crucial, and there might be further room for optimization in the generative aspect of the process.
- Predefined Prompts:** HyDE typically uses predefined prompts for generating hypothetical documents. This approach is somewhat rigid as it assumes that the query/input fits these prompts, which may not always be the case.
- Limited to Q&A Framework:** HyDE's standard implementation mainly supports question-answering scenarios. Other types of retrieval tasks, where the input isn't a question, might not benefit as much from this method.
- Homogeneity of Document Chunks:** The effectiveness of HyDE can diminish when dealing with non-homogeneous document chunks in terms of style, tone, or structure. A single generalized prompt might not be effective for all types of document chunks.

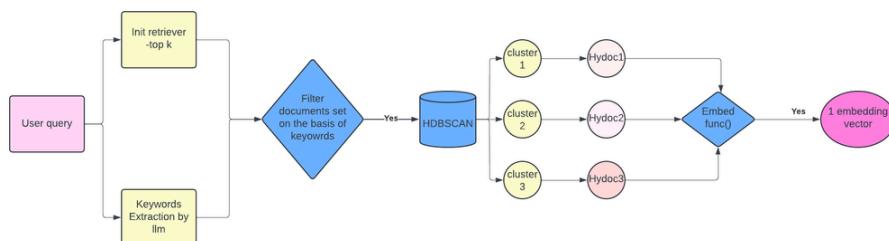


5.

### Working of AutoHyDE

AutoHyDE aims to address the limitations of the standard HyDE approach by automatically discovering relevant patterns in indexed documents and generating more tailored hypothetical documents. Here's how it works:

- Keyword Extraction:** AutoHyDE begins by extracting keywords from the user query to focus the retrieval process on relevant documents.
- Initial Retrieval Expansion:** It retrieves a larger set of documents than typical to explore potentially overlooked relevant documents.
- Keyword-Based Filtering:** Documents containing the extracted keywords are identified from the extended retrieval set, highlighting potentially neglected but relevant documents.
- Clustering of Documents:** The selected documents are clustered based on their content to identify common relevance patterns.
- Generation of Hypothetical Documents:** For each cluster, AutoHyDE generates a hypothetical document that reflects the shared characteristics and content of the documents within that cluster.
- Embedding and Combining:** These hypothetical documents are then embedded and combined into a single vector that represents the query response, enhancing the relevance of the retrieved documents.



## RESULTS

### Experiment : 1

Evaluating HIT RATE	Baseline RAG (K=100)	HYDE (K=100)	AUTO HYDE (K=100)
Extreme Dataset	75%	60%	72%

\*ADD MORE DATASETS\*

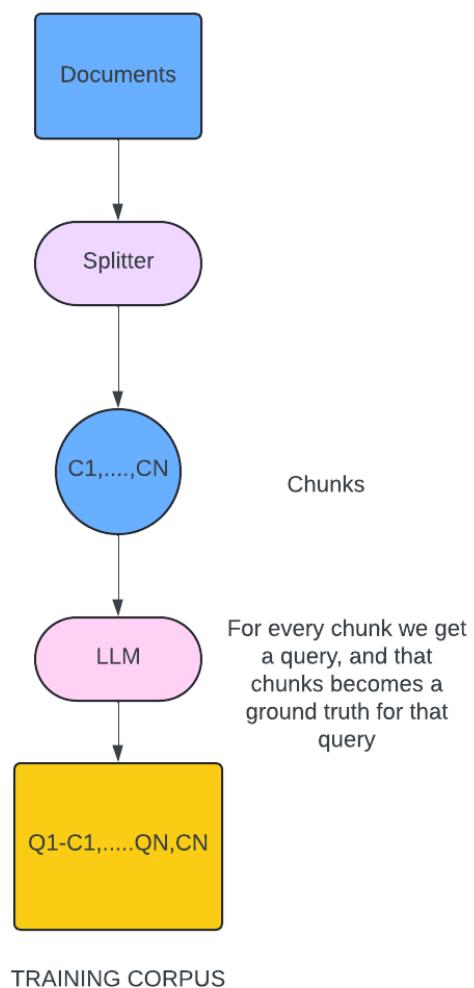
## CONCLUSIONS

- Improved Relevance:** By clustering documents and generating hypothetical documents based on relevance patterns, AutoHyDE captures a broader range of relevant content, improving the accuracy of retrieval.
- Time Complexity:** The process of extracting keywords, performing extra retrieval, clustering, and generating multiple hypothetical documents for each cluster significantly increases the computation time compared to the original HyDE.
- Computational Overhead:** Clustering documents and generating separate hypothetical documents for each cluster is resource-intensive, making it less scalable for large datasets.

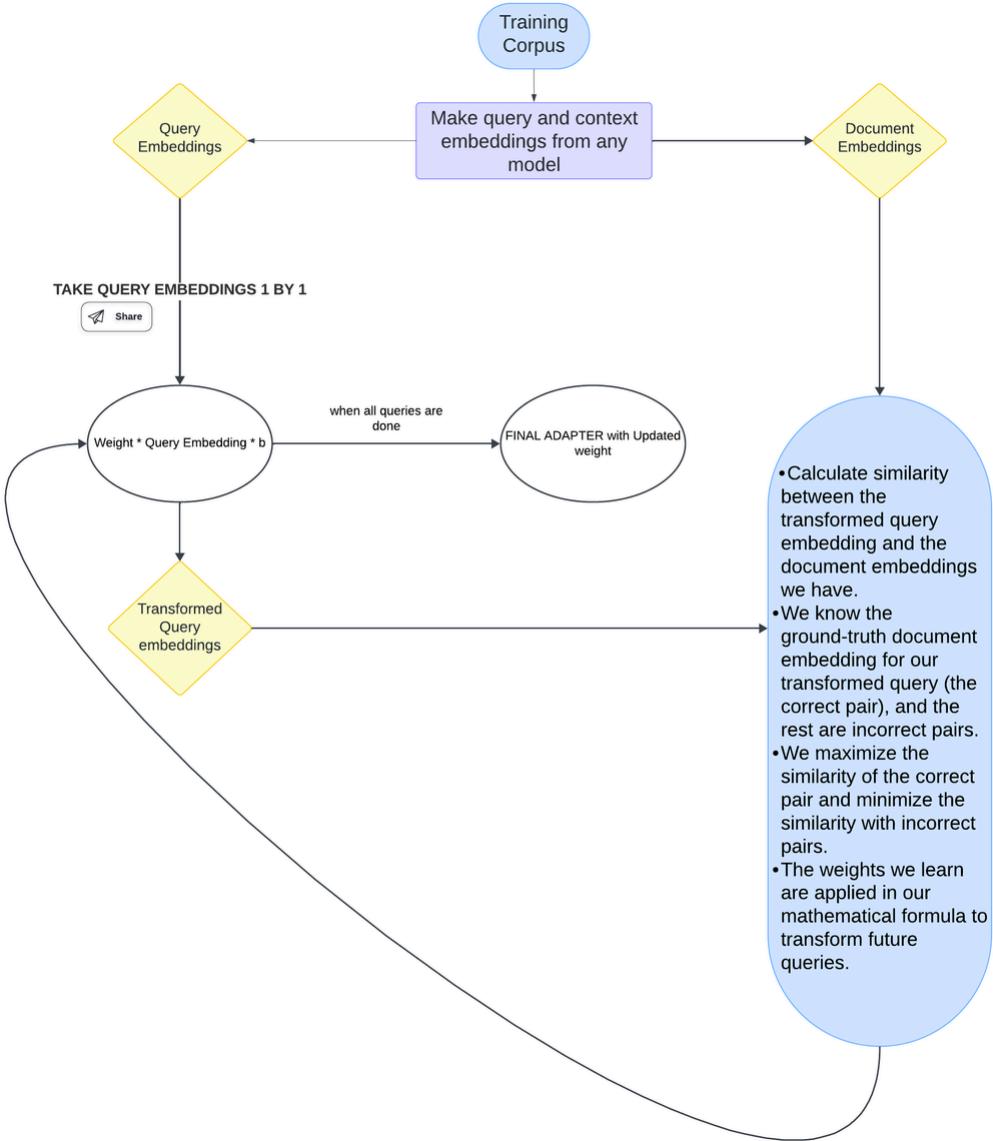


# Flow-AutoHyde

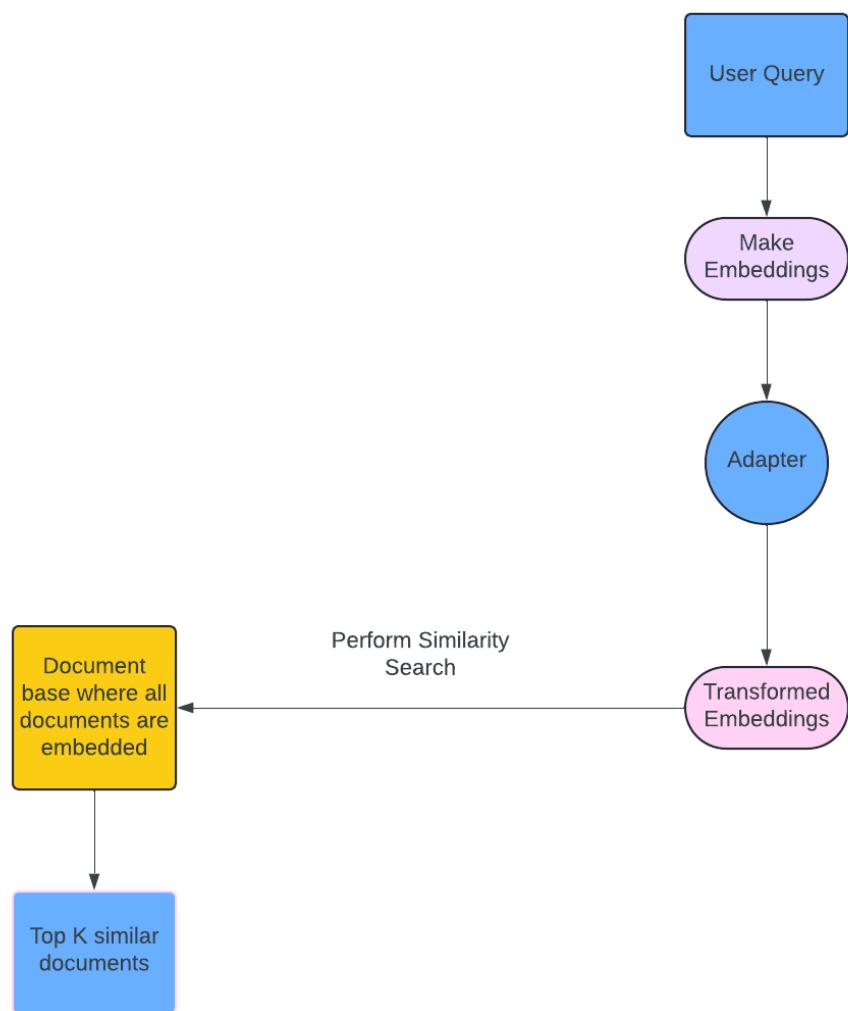
## 1) GENERATION OF TRAINING CORPUS ↗



## 2) ADAPTER TRAINING FLOW ↗



## 3) HOW WE USE FINAL ADAPTER FROM ABOVE IN RAG ↗



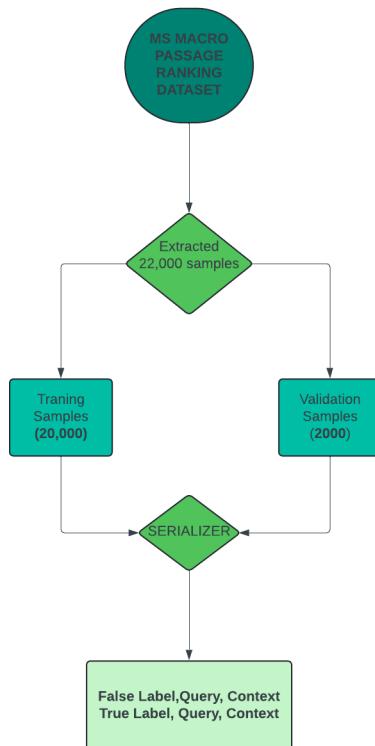
## Fine tuning Rerankers

0.0.Fine-tuning rerankers tailors them to specific tasks for improved ranking accuracy.

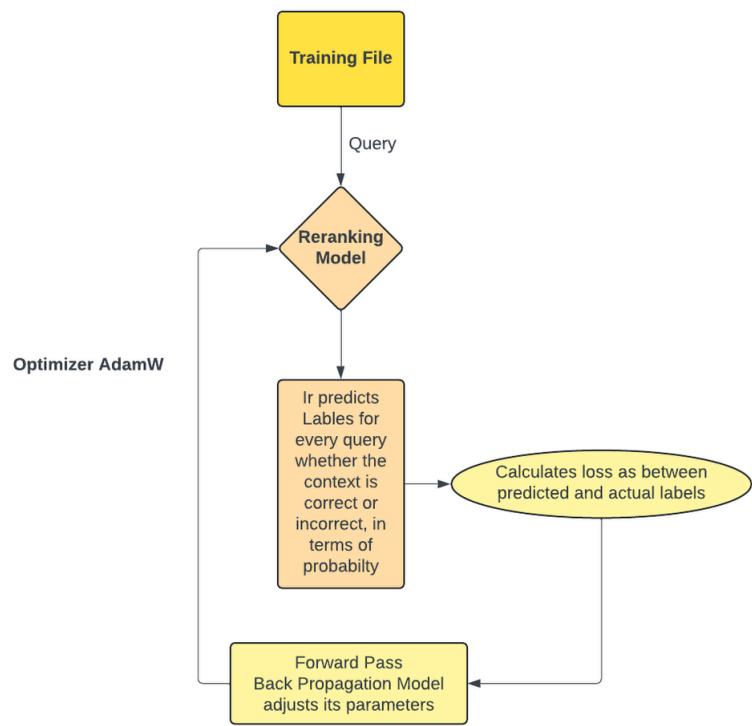
We will fine-tune open-source rerankers to evaluate if they outperform existing models. Our implementation will be divided into three parts: data processing, training loop and evaluation, followed by testing on a dataset or ranked model using performance metrics like nDCG.

The implementation of pipeline goes as follows:

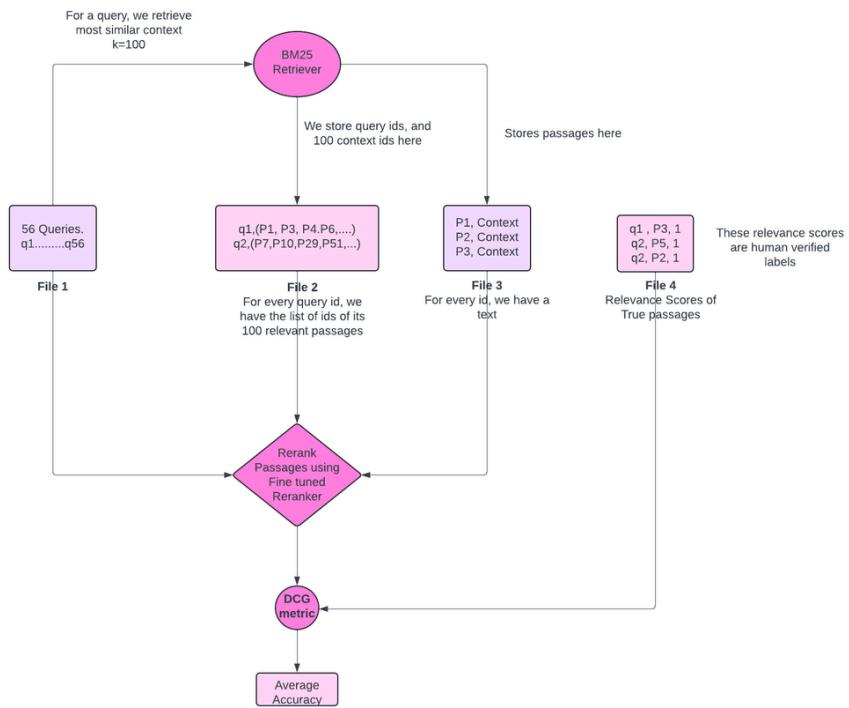
### STEP : 1 - DATA ANALYSIS FOR TRAINING AND VALIDATION ↗



### STEP : 2 - TRAINING LOOP AND EVALUATION ↗



### STEP : 3 - CHECK PERFORMANCE ON TESTING DATASET ↗



## HOW DOES DCG METRIC WORKS? ↴

**DCG = Discounted Cumulated Gain**

Following example is for 1 query, For all queries we simply take an average of final DCG. ↴

Rank	Passage ID	Relevance
1	p1	1
2	p3	0
3	p2	1
4	p4	0
5	p5	0

**Step-by-Step DCG Calculation:**

$$\text{DCG@5} = \frac{1}{\log_2(1+1)} + \frac{0}{\log_2(2+1)} + \frac{1}{\log_2(3+1)} + \frac{0}{\log_2(4+1)} + \frac{0}{\log_2(5+1)} = 1 + 0 + 0.5 + 0 + 0 = 1.5$$

**Step-by-Step Ideal DCG (IDCG@5):**

If the relevant passages were perfectly ranked at the top, the ranking would look like this:

Rank	Passage ID	Relevance
1	p1	1
2	p2	1
3	p3	0
4	p4	0
5	p5	0

$$\text{IDCG@5} = \frac{1}{\log_2(1+1)} + \frac{1}{\log_2(2+1)} + \frac{0}{\log_2(3+1)} = 1 + 0.63 = 1.63$$

**Final nDCG@5 Calculation:**

$$\text{nDCG@5} = \frac{\text{DCG@5}}{\text{IDCG@5}} = \frac{1.5}{1.63} \approx 0.92$$

## RESULTS ↴

### Model : ibm/re2g-reranker-trex (Bi-Encoder) ↴

How to read the nDCG Values?

"The ranking algorithm achieves k% of the ideal ranking quality in the top n results over the 50 passages."

50 passages	nDCG@5	nDCG@10	nDCG@20	nDCG@50	nDCG@100	nDCG@1000
Base version	15 %	18 %	22%	34 %	-	-
Fine tuned version	61 %	60 %	57 %	51 %	-	-

"The ranking algorithm achieves k% of the ideal ranking quality in the top n results over the 100 passages."

100 passages	nDCG@5	nDCG@10	nDCG@20	nDCG@50	nDCG@100	nDCG@1000
Base version	10 %	11 %	13 %	20 %	32 %	-
Fine tuned version	62 %	61 %	59 %	56 %	54 %	-

"The ranking algorithm achieves k% of the ideal ranking quality in the top n results over the 1000 passage"

1000 passages	nDCG@5	nDCG@10	nDCG@20	nDCG@50	nDCG@100	nDCG@1000
Base version	less than 1	less than 1			less than 1	29 %

Fine tuned version	65 %	63 %	61 %	60 %	66 %
--------------------	------	------	------	------	------

## Model : cross-encoder/ms-marco-MiniLM-L-6-v2

How to read the nDCG Values?

"The ranking algorithm achieves k% of the ideal ranking quality in the top n results over the 50 passages."

50 passages	nDCG@5	nDCG@10	nDCG@20	nDCG@50	nDCG@100	nDCG@10 0 0
Base versio n	37 %	38 %	39 %	42 %	-	-
Fine tuned version	57 %	54 %	49 %	53 %	-	-

"The ranking algorithm achieves k% of the ideal ranking quality in the top n results over the 100 passages."

100 passages	nDCG@5	nDCG@10	nDCG@20	nDCG@50	nDCG@100	nDCG@10 0 0
Base versio n	33 %	33 %	33 %	37 %	41 %	-
Fine tuned version	56 %	53 %	51 %	51 %	50 %	-

"The ranking algorithm achieves k% of the ideal ranking quality in the top n results over the 1000 passage"

1000 passages	nDCG@5	nDCG@10	nDCG@20	nDCG@100	nDCG@1000
Base version	16 %	16 %	-	17 %	38 %
Fine tuned version	50 %	47 %	47 %	47 %	57 %

## Model : colbert-v2

How to read the nDCG Values?

"The ranking algorithm achieves k% of the ideal ranking quality in the top n results over the 50 passages."

50 passages	nDCG@5	nDCG@10	nDCG@20	nDCG@50	nDCG@100	nDCG@10 0 0
Base versio n	63 %	58 %	55 %	52 %	-	-
Fine tuned version	62 %	59 %	56 %	51 %	-	-

"The ranking algorithm achieves k% of the ideal ranking quality in the top n results over the 100 passages."

100 passages	nDCG@5	nDCG@10	nDCG@20	nDCG@50	nDCG@100	nDCG@10 0 0
Base versio n	64%	58%	55%	54%	53%	-
Fine tuned version	63%	61%	59%	56%	54%	-

"The ranking algorithm achieves k% of the ideal ranking quality in the top n results over the 1000 passage"

1000 passages	nDCG@5	nDCG@10	nDCG@20	nDCG@100	nDCG@1000
Base version	57%	53%	-	49%	59%

Fine tuned version	66%	63%	-	60%	66%
--------------------	-----	-----	---	-----	-----

## TAKEAWAYS ☕

- **Base version:**
  - The base version achieves quite low nDCG values across different cutoffs. This indicates that the base model struggles with ranking.
- **Fine-tuned version:**
  - The fine-tuned version shows a massive improvement. The fine-tuning clearly improves the ability of the model to rank relevant results in higher positions
- **Exception:**

Colbert reranker only showed massive improvement in fine tuning when we re ranked 1000 passages, given the reason that Colbert is bi-encoder with late interaction which is more efficient for large scale retrieval means on a large dataset.

## Why nDCG Increases for Larger Cutoffs in the Base Model? ☕

- The **base model** shows an increase in nDCG with larger cutoffs because it is performing poorly at ranking the most relevant items in the very top positions (e.g., nDCG@5, nDCG@10). When the cutoff increases (e.g., nDCG@50, nDCG@100), the model has more opportunities to include relevant items, even if they are ranked lower in the list.
- In simpler terms, the base model isn't able to push relevant items to the top but still manages to place them somewhere within the top 100 or 1000 passages, which results in higher nDCG values for larger cutoffs.

## . Why nDCG Decreases with Larger Cutoffs in the Fine-Tuned Model? ☕

- The **fine-tuned model** performs very well at smaller cutoffs like nDCG@5 and nDCG@10, as it ranks the most relevant results at the top of the list. However, as the cutoff increases to nDCG@50, nDCG@100, etc., there is a slight **decrease** in nDCG.
  - This happens because the model has already ranked most of the relevant passages near the top (e.g., within the top 10 or 20), so as the cutoff increases, it's now ranking less relevant or irrelevant passages. This causes the nDCG to drop slightly because the model's ranking quality starts to taper off as it has to fill larger ranks with less relevant items.
  - **nDCG@1000 for 1000 passages** is an exception because the fine-tuned model can place all relevant passages within the entire pool of 1000, leading to a peak score of 66%. This higher score occurs because it's not "cutting off" at a smaller number of passages and thus ranks more relevant items within the full list

## Comparing results before and after fine tuning with MRR: ☕

Model	MRR with base version for Top 50 Passages	MRR with fine tuned version Top 50 Passages
ibm/re2g-reranker-trex (Bi-Encoder)	$0.0928 = 1/0.0928 = 10.77 \text{ rank}$	$0.8884 = 1/0.8884 = 1.13 \text{ rank}$
cross-encoder/ms-marc o- MiniLM-L-6-v2	$0.12 = 1/0.12 = 8.33 \text{ rank}$	$0.9173 = 1/0.9173 = 1.09 \text{ rank}$

## Conclusions ☕

- MRR also shows an improved value after fine tuning. Also, these values are more good than DCG, because in DCG, we got it for more than 1 context. In MRR metric applied here we saw for only first valid context.

## Fine tuning LLMs : An estimate

There are three costs associated with fine-tuning and using the fine-tuned GPT 3.5-turbo model.

1. **Training data preparation.** This involves curating a dataset of text prompts and desired responses tailored to your specific use case. The cost will depend on the time and effort required to source and format the data.
2. **Initial training cost.** This is charged per token of training data. At \$0.008 per 1,000 tokens, a 100,000 token training set would cost  
\$800 for the initial fine-tuning.
3. **Ongoing usage costs.** These are charged per token for both input prompts and model outputs. At \$0.012 per 1,000 input tokens and  
\$0.016 per 1,000 output tokens, costs can add up quickly depending on application usage.

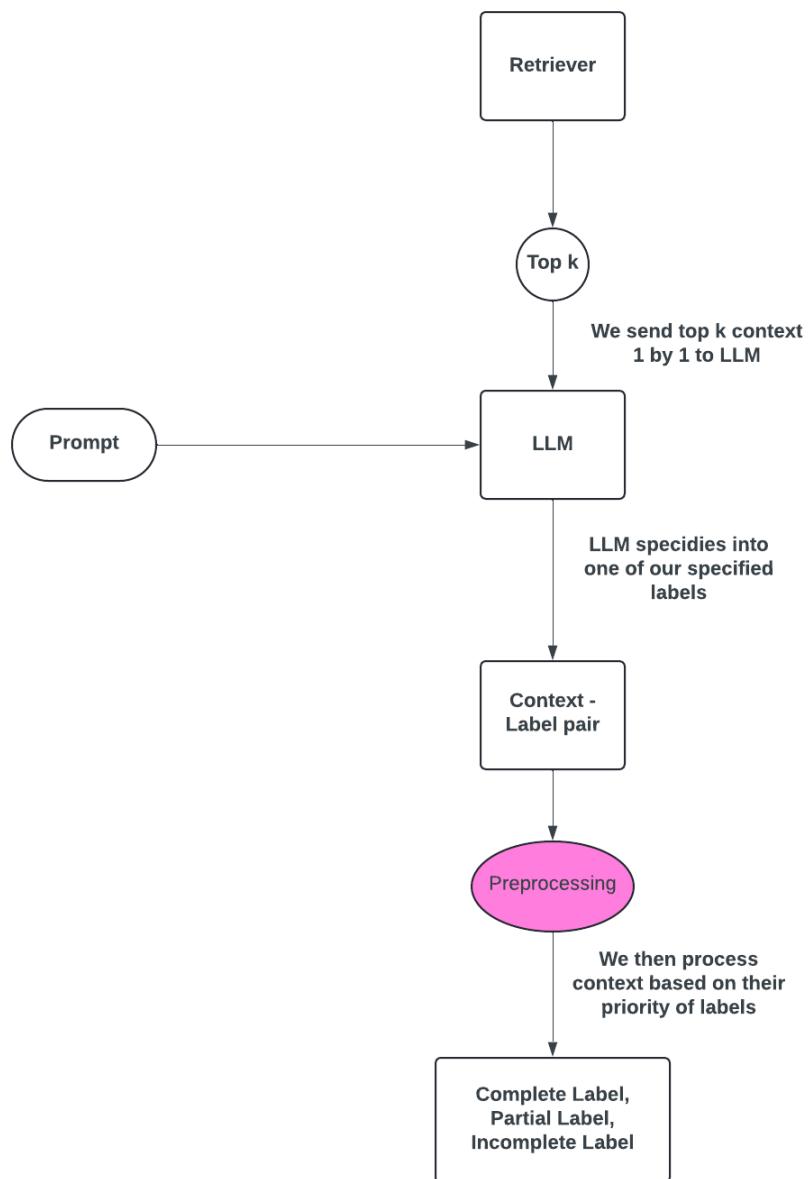
Let's see an example of usage cost scenario:

- Chatbot with 4,000 token prompts/responses, 1,000 interactions per day:  
 $(4,000/1000) \text{ input tokens} \times \$0.012 \times 1,000 \text{ interactions}$   
 $= \$48 \text{ per day}$   
 $(4,000/1000) \text{ output tokens} \times \$0.016 \times 1,000 \text{ interactions}$   
 $= \$64 \text{ per day}$   
Total = \$112 per day or \$3,360 per month
- Text summarization API with 2,000 token inputs, 500 requests per day:  
 $(2,000/1000) \text{ input tokens} \times \$0.012 \times 500 \text{ requests}$   
 $= \$12 \text{ per day}$   
 $(2,000/1000) \text{ output tokens} \times \$0.016 \times 500 \text{ requests}$   
 $= \$16 \text{ per day}$   
Total = \$28 per day or \$840 per month

## Fine tuning LLMs

Use Case : Take a LLM based reranker, Fine tune an opensource llm, then make comparison between fine tuned version and base version of that model

## How Re ranker works? (Customized LLM based reranker) ↗



## Dataset Curation ↗

Our dataset will consist of 3 different kinds of data, meaning query, context pair belonging to three different types of Labels (Complete, Partial, Irrelevant)

- **Complete Label** : We already do have query contexts pair, where contexts is actually the ground truth for that query means we know query answer lies in that

context.

- **Irrelevant Label** : We send our query to a retriever and get top k contexts, as we know ground truth context as mentioned above for queries, we remove that context from retrieved contexts so we have now k-1 contexts, which all come under irrelevant labels
- **Partial Label** : We take 2 queries from our datasets, and combine them into a single query, now, we know that ground truths for the original queries. That both individually are now partial relevant to that combined query.

## Size of dataset

- 2500 Training Dataset in terms of (Question, Context) and Label pairs
- 1000 Validation Dataset in terms of (Question, Context) and Label pairs
- 60 test queries

## Iteration 1

- Went with Mistral-7b, Fine tuned this llm on our dataset, but got cuda out of memory issues.
- Could have reduced data size but i thought to compromise on Model than data.

## Iteration 2

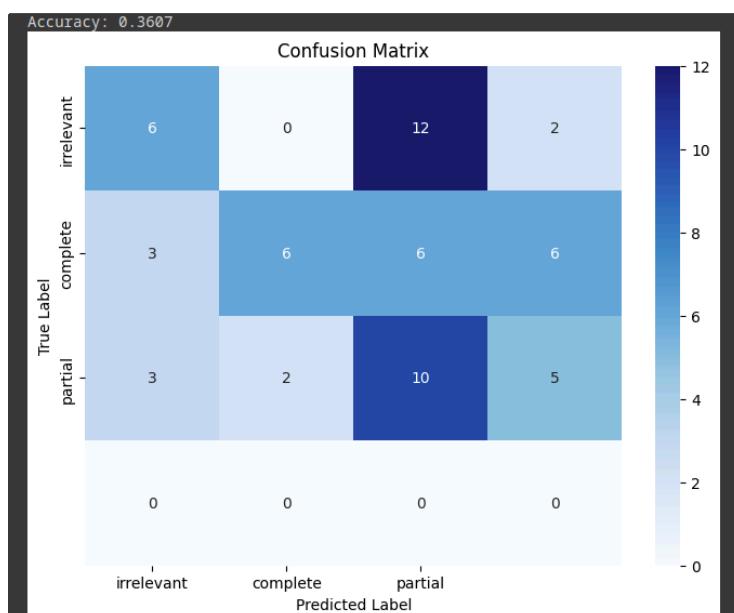
- distilbert/distilgpt2 - 124 MILLION PARAMETERS
- Fine tuned my dataset on this model, it worked
- Results of fine tuning were very very hallucinated
- Why?
- Then i tested the base version of this model first to see, how the base lined version of this model will perform on my dataset.
- Again, the results were bad same as fine tuned. Concluded, that model is not fit for this use case means its not that capable to analyze prompt and perform a classification.

## Iteration 3

- GPT-neo - 1.3 BILLION PARAMETERS
- Again, the results were bad at base version. Model hallucinated a lot. Was not able to understand a prompt.

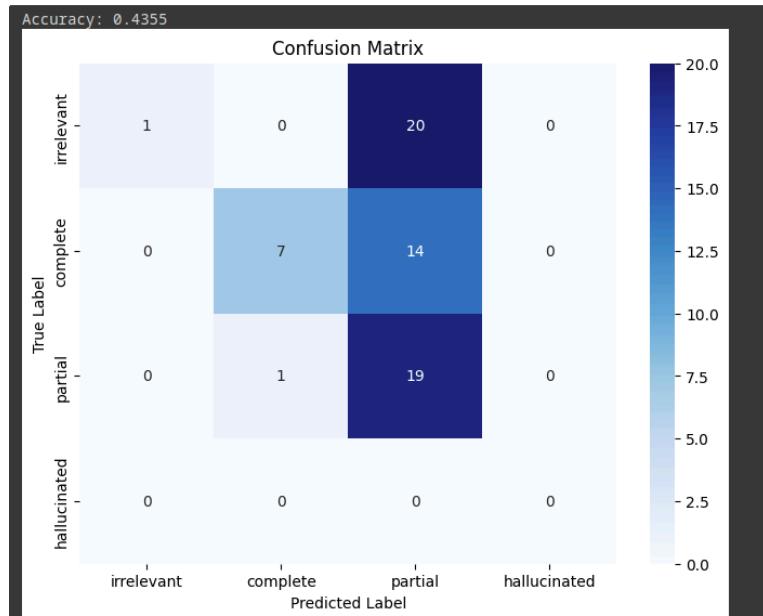
## Iteration 4

- pankajmathur/orca\_mini\_3b - 3 BILLION PARAMETERS
- It did quite well, understood prompt. Hallucinations were very less.
- Accuracy on base line version : **36.07 %**



- Tested test dataset on the our deployed llama3.1 8b to see How its base line version performs, and how our selected model's base version and fine tune version is as compared with it.

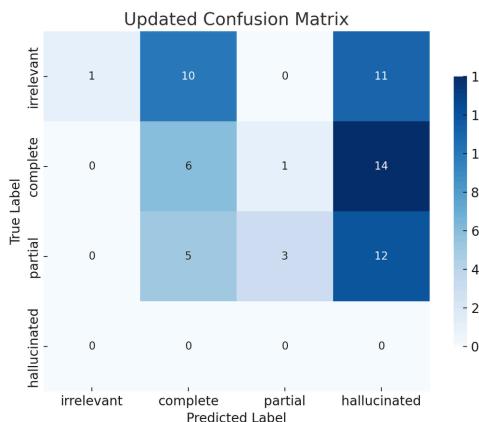
- Accuracy we got : **43.5%**



The analysis from both confusion matrix shows that models are confusing two pairs of labels a lot, (partial and complete ) and ( partial and irrelevant ).

## Iteration 5 - Fine tune Orca mini 3b [🔗](#)

- Tried Fine tuning on original training dataset, got cuda out of memory issues
- Reduced the training datasets, to even half still same error.
- Customized pipeline parameters, batching still same issue. Also batch size 1, 2 they dont cause a much difference in result just an insight. Model never learns complex patters and tends to over fit.
- Customized other few parameters, like reduced sequence length, reduced LoRa rank, Efficient loading of dataset (streaming loading), enabled gradient accumulation and with batch size 4.
- Making above configurations, i was able to fine tuned.



- Accuracy : 17%
- We can clearly see lots of hallucinations

## Iteration 6 - Mistral 7b [🔗](#)

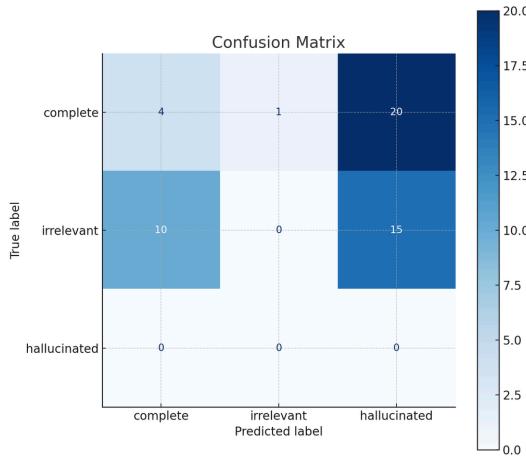
- Fine tuned mistral-7b on same labels and same technique, again model produced hallucinations and a very bad accuracy.

## Conclusion ☺

- Fine tuning rather than improving, shows a decrease in accuracy not a mild one but a drastic one, mainly labels being categorized as hallucinations, which in base version seems to be quite less.

## Iteration 7 - Orca mini 3b only with 2 labels ☺

- This time dataset had about 5000 queries to fine tune on
- Partial label was confused between the complete or irrelevant, so we removed partial
- For base versions we got accuracy score 36%
- For fine tuned version, we got accuracy of 8%



## Conclusion ☺

- Same trend here, a decrease in accuracy not because of miss-classifications but because of fine tuned model hallucinating a lot
- There's overall a decrease in fine tuning in all experiments, not because model is wrongly specifying but model totally fails to understand
- LLM struggled to integrate new factual knowledge through fine-tuning, and as they eventually learn this new information, their tendency to hallucinate increased, and there is a reason to my knowledge
- We retrieved top k contexts, and we labeled the actual context as our complete and other k-1 as irrelevant and technically those are still similar, like top similar that's why they got retrieved
- The technique basically does not make model forget its previous knowledge right, it adds a fine tuned extra knowledge base as a adapter over it.