RAG-LLM Efficiency Engine: A Generic Approach

By, Sweta Shukla

Date: 22/04/2025

Indian Institute of Information Technology, Raichur

Guided by,

Dr. Priodyuti Pradhan

Problem

Large Language Models (LLMs) are powerful but can "hallucinate" or generate plausible but incorrect information, especially on specialized topics. They lack access to real-time or domain-specific knowledge beyond their training data.

Solution??

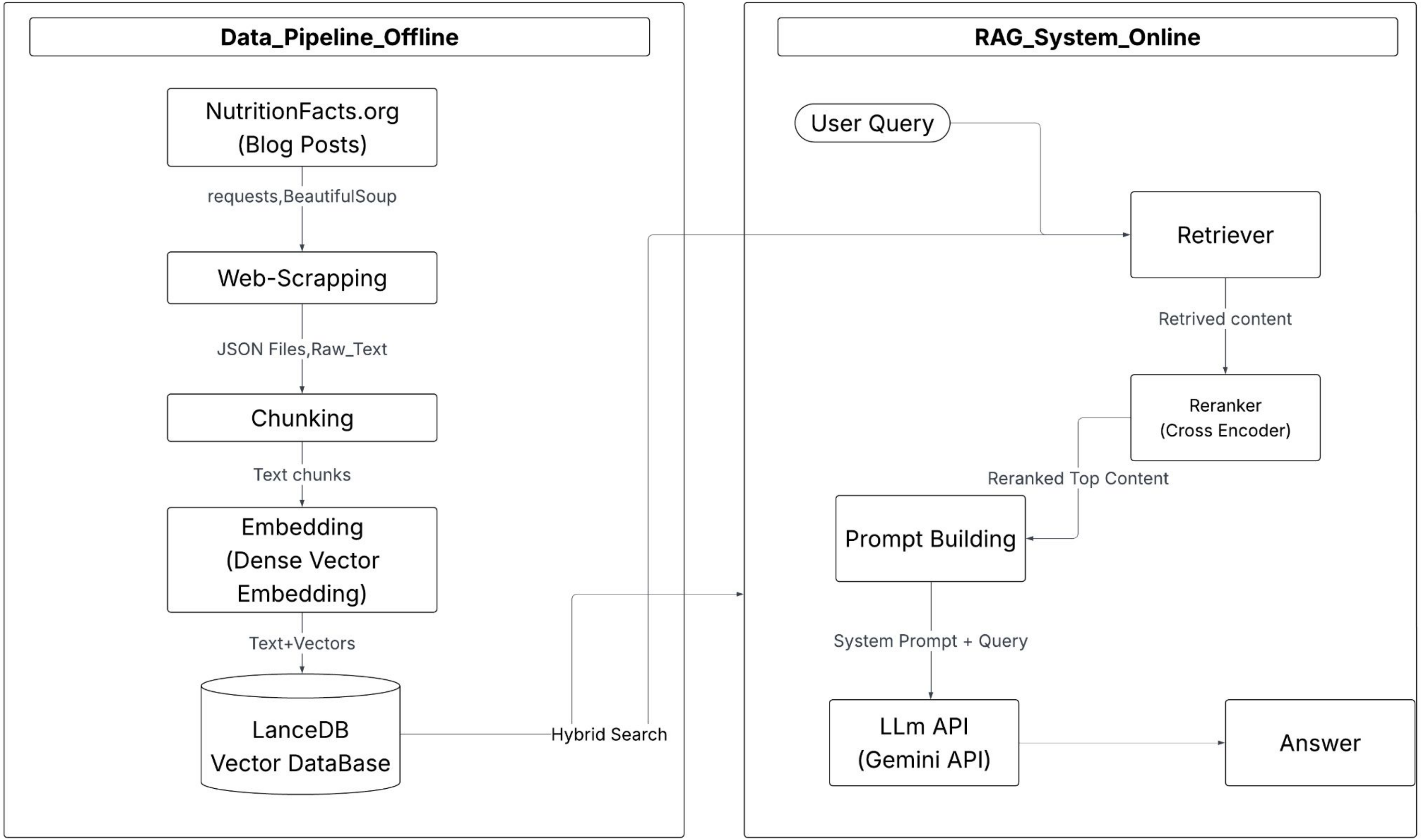
Retrieval-Augmented Generation (RAG) - A powerful architecture combining information retrieval with LLM generation.

What is RAG?

- **Retrieval:** It retrieves relevant snippets from a large knowledge base based on the user's query.
- **Augmentation:** It enriches the user's query or LLM prompt with retrieved context for better, accurate responses.
- **Generation:** An LLM generates an answer using both the original query and retrieved context, ensuring factual grounding.
- **Benefit:** Reduces hallucination, improves factual accuracy, allows grounding in specific/updated knowledge.

System Architecture: A Modular RAG Pipeline

1. **Data Preparation Pipeline:** Ingesting and preparing any text corpus for retrieval (Scraping/Loading ⟶ Chunking ⟶ Embedding ⟶ Vector Storage).
2. **RAG Core:** The real-time query processing flow (Query ⟶ Retrieve ⟶ Rerank ⟶ Augment Prompt ⟶ Generate).


Data_Pipeline_Offline
NutritionFacts.org
(Blog Posts)
requests,BeautifulSoup
Web-Scrapping
JSON Files,Raw_Text
Chunking
Text chunks
Embedding
(Dense Vector
Embedding)
Text+Vectors
LanceDB
Vector DataBase
Hybrid Search
RAG_System_Online
User Query
Retriever
Retrived content
Reranker
(Cross Encoder)
Reranked Top Content
Prompt Building
System Prompt + Query
LLm API
(Gemini API)
Answer

```
100%|██████████| 1248/1248 [01:31<00:00, 13.58it/s]
fts 1.0 2.0238782051282014
100%|██████████| 1248/1248 [02:51<00:00,  7.29it/s]
vector 0.8044871794871795 1.058506944444448
100%|██████████| 1248/1248 [03:15<00:00,  6.37it/s]
hybrid 0.9735576923076923 1.5301014957264971
```

Linear Combination

```
10 results for query
7 unique URL(s)
7 unique Title(s)
Paragraphs:
  1. (0.100) 'How to Eliminate 90 P
  2. (0.100) 'Why Don’t More Doctor
  3. (0.097) 'How to Eliminate 90 P
  4. (0.066) 'Magnesium-Rich Foods
  5. (0.065) 'How to Eliminate 90 P
  6. (0.063) 'How to Eliminate 90 P
  7. (0.036) 'The Most Anti-Inflamm
  8. (0.022) 'How to Eat to Reduce
  9. (0.021) 'Breast Cancer and Alc
 10. (0.019) 'How to Prevent Heart
```

Cross-Encoder Rearanker

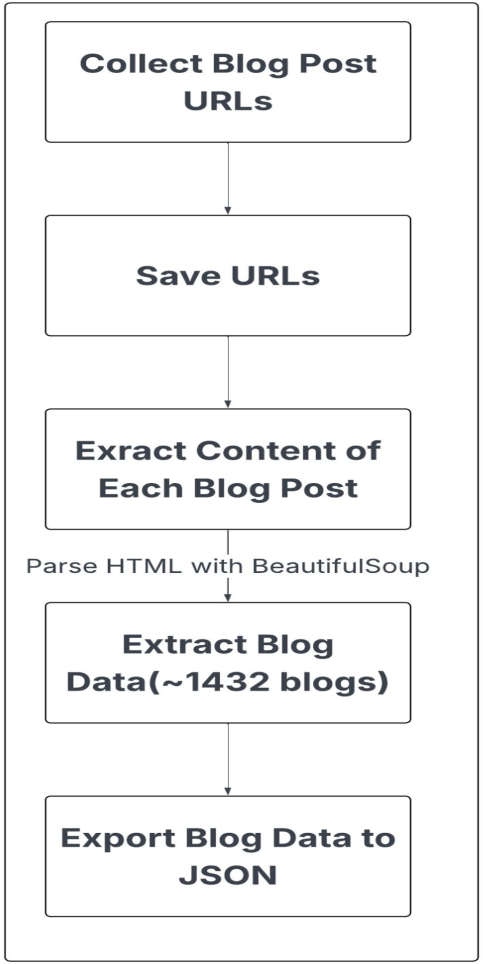
```
10 results for query
8 unique URL(s)
8 unique Title(s)
Paragraphs:
  1. (0.591) 'How to Prevent Heart Disea
  2. (0.591) 'How to Prevent Heart Disea
  3. (0.514) 'Breast Cancer and Alcohol:
  4. (0.254) 'The Most Anti-Inflammatory
  5. (0.084) 'How to Prevent Heart Disea
  6. (0.083) 'How to Eliminate 90 Percen
  7. (0.038) 'How Much Nutrition Educati
  8. (0.026) 'How to Treat High Lp(a), a
  9. (0.017) 'Moderation Kills' : 'What
 10. (0.013) 'How to Eliminate 90 Percen
```

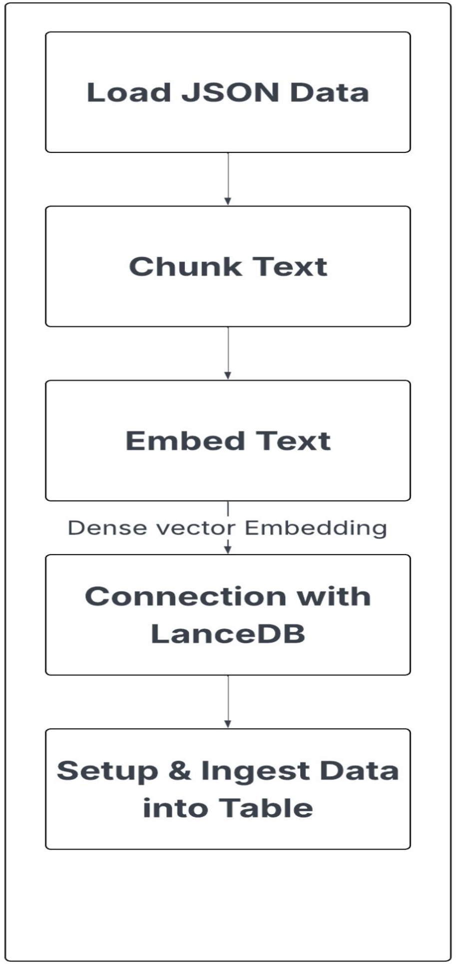
Results

Implementation - Data Preparation Pipeline

- **Objective:** Convert a raw text corpus into a searchable vector index.
- **Used example Corpus:** NutritionFacts.org blog posts.
- **Steps:**
  - **Ingestion:** Acquiring documents (e.g., via web scraping).
  - **Chunking:** Breaking documents into smaller, semantically meaningful units.
  - **Embedding:** Transforming text chunks into dense vector representations using Sentence Transformer.
  - **Vector Storage:** Loading vectors and associated metadata into LanceDB.

WEB SCRAPING

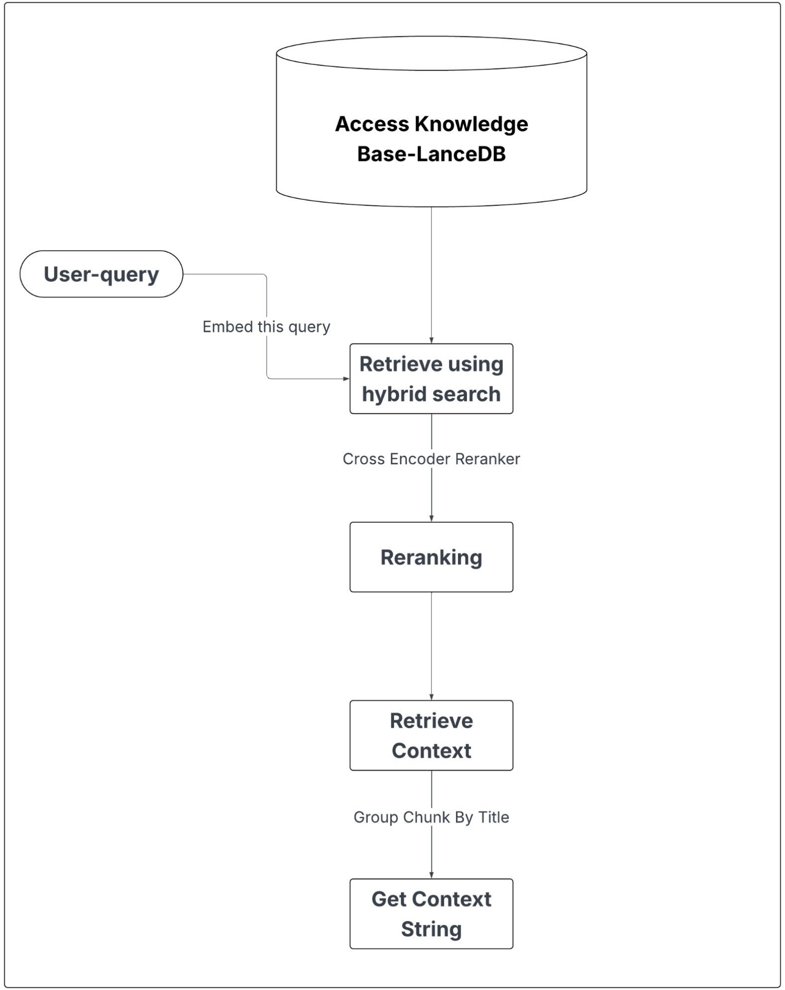


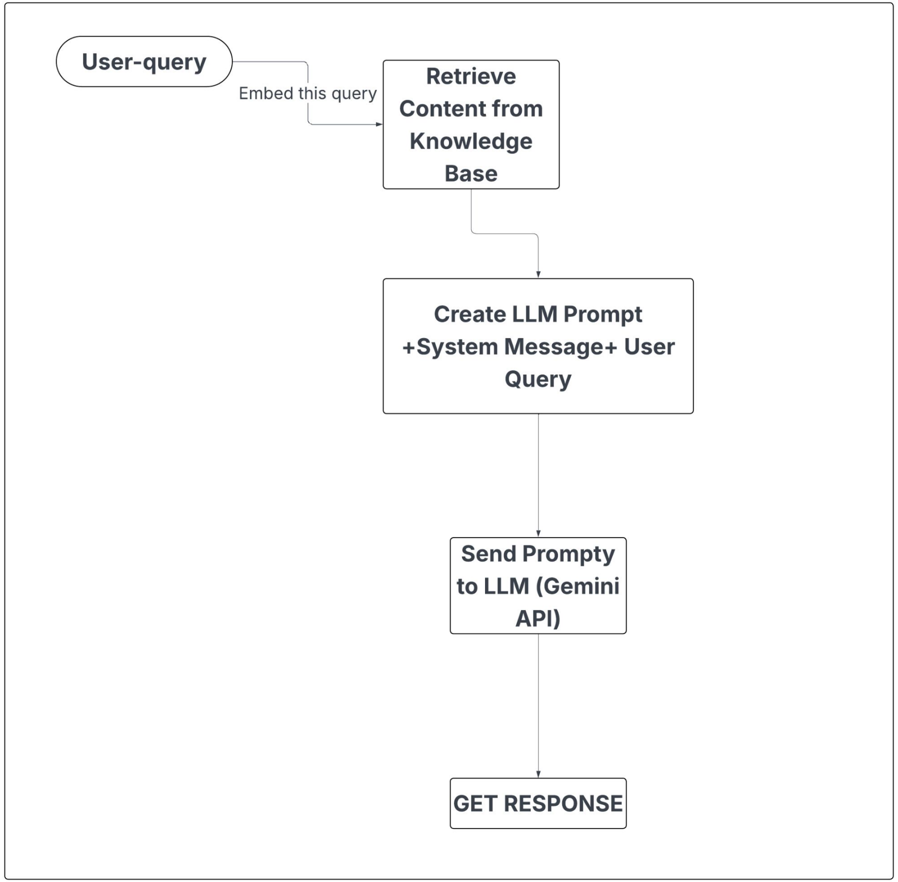


DATA INGESTION

Implementation - RAG Core: Retrieval & Reranking

- **Goal:** Identify the most relevant text chunks for the input query from the Vector DB.
- **Initial Retrieval**
  - Using Hybrid Search in LanceDB (combining keyword and vector similarity) for broad recall.
- **Reranking**
  - *Why Rerank?* Initial retrieval prioritizes speed/recall; reranking refines precision using more computationally intensive models.
  - Method: Employing a CrossEncoderReranker to re-score the top candidates based on deeper semantic relevance to the query.

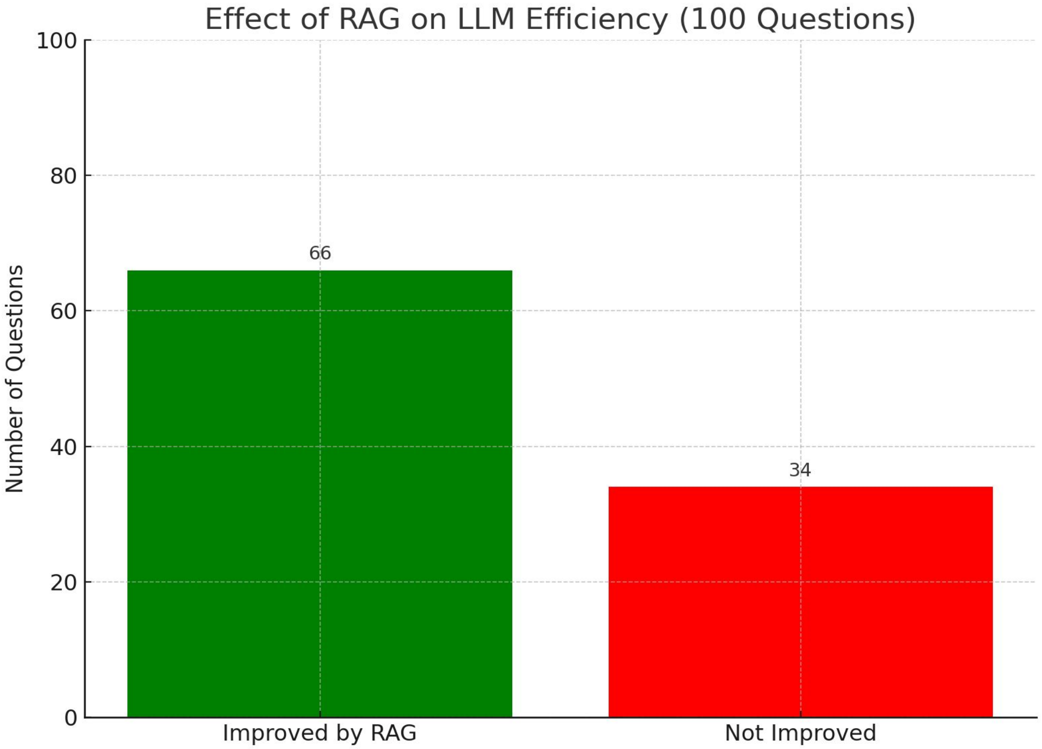

Access Knowledge Base-LanceDB
User-query
Embed this query
Retrieve using hybrid search
Cross Encoder Reranker
Reranking
Retrieve Context
Group Chunk By Title
Get Context String


User-query
Embed this query
Retrieve Content from Knowledge Base
Create LLM Prompt +System Message+ User Query
Send Prompty to LLM (Gemini API)
GET RESPONSE

Implementation - RAG Core: Generation

- **Goal:** Generate a final answer grounded in the retrieved and reranked context.
- **Prompt Augmentation**
  - Constructing a specific prompt for the LLM, incorporating the top N retrieved text chunks as context.
  - Instructing the LLM to base its answer solely on the provided context.
- **LLM Integration**
  - Utilizing the LLM API for efficient generation with models.
- **Demonstration**

Results



Effect of RAG on LLM Efficiency (100 Questions)
100
80
60
40
20
0
Number of Questions
66
34
Improved by RAG
Not Improved

Challenges & Learnings (Focus on RAG)

- **RAG-Specific Challenges:**
  - Optimizing chunk size (too small = loss of context, too large = noise).
  - Balancing retrieval and the role of reranking.
  - Prompt engineering to effectively instruct the LLM to use context.
- **Key Learnings:** RAG is a powerful but modular architecture requiring careful tuning of each component (embedding, retrieval, reranking, prompting) for optimal performance.

Thank You & Questions