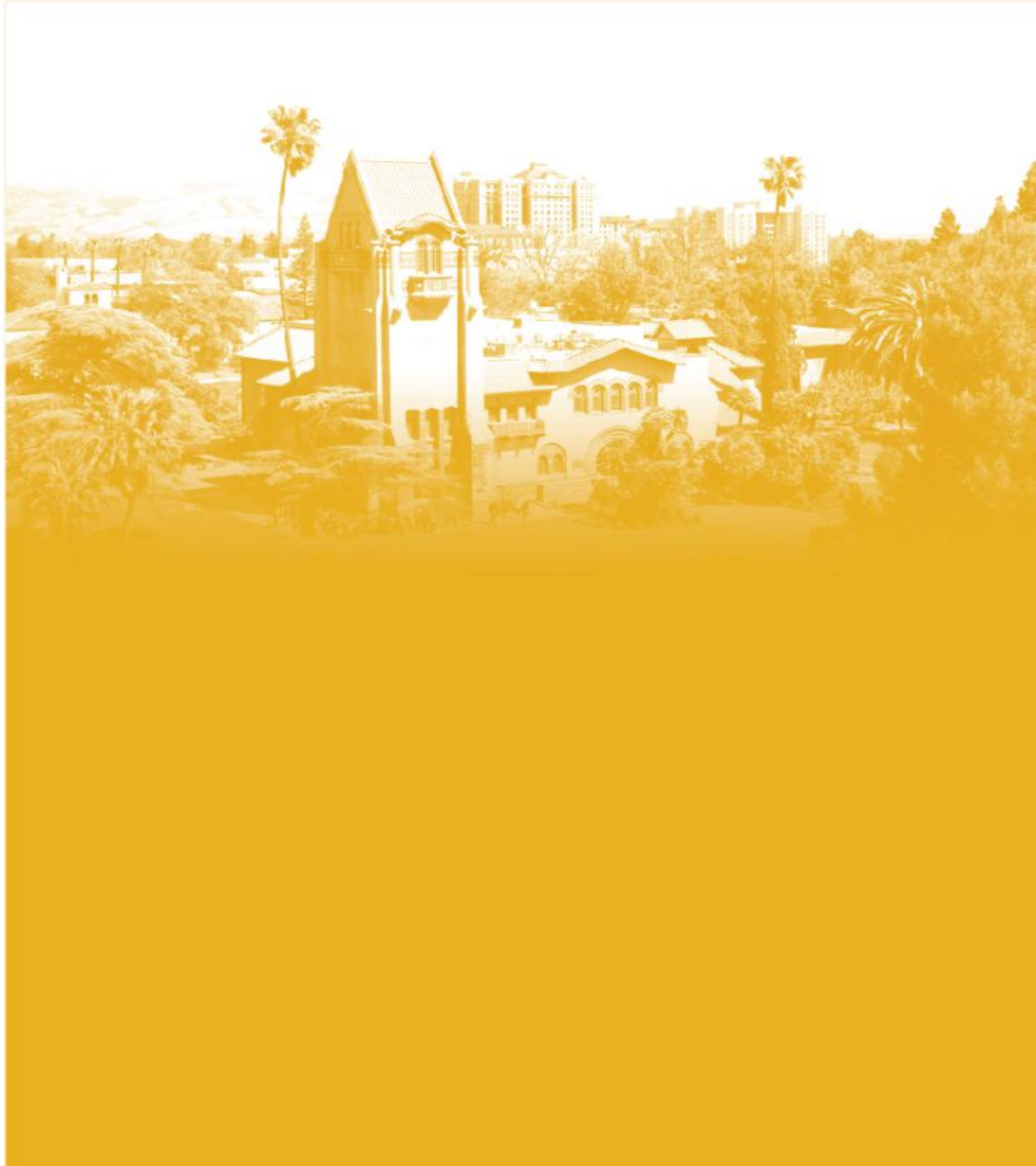




**DATA 255**  
**Deep Learning Technologies**

*Dr. Mohammad Masum*



- Vector databases store and search high-dimensional vector embeddings
  - Scalable semantic search, similarity search, and RAG pipelines.
- Embeddings: Transform text/images/audio into **fixed-size dense vectors** (OpenAI, Sentence Transformers).
- Vector Spaces: Euclidean, Cosine, Inner Product spaces – relevance to similarity.
- Indexing Techniques:
  - Flat Index (Brute Force): Accurate, slow.
  - Approximate Nearest Neighbor (ANN): Fast, scalable. Algorithms: HNSW, FAISS, Annoy.



- LangChain is an open-source framework – developing applications powered by LLMs
- It enables modular development by “chaining” components such as prompts, models, tools, and memory.
- LangChain simplifies the creation of complex LLM workflows—examples include chatbots, question-answering systems, and summarization tools.
- It offers standardized interfaces for key components: LLMs, embedding models, vector databases (e.g., Pinecone), and various utilities.
- It streamline LLM pipelines—fetching contextual data from a vector database and seamlessly passing it to an LLM for processing.
- LangChain eliminates the need to reinvent common LLM integration patterns, allowing developers to focus on higher-level application logic.

The LangChain framework consists of multiple open-source libraries. Read more in the [Architecture](#) page.

- **langchain-core**: Base abstractions for chat models and other components.
- **Integration packages** (e.g. **langchain-openai**, **langchain-anthropic**, etc.): Important integrations have been split into lightweight packages that are co-maintained by the LangChain team and the integration developers.
- **langchain**: Chains, agents, and retrieval strategies that make up an application's cognitive architecture.
- **langchain-community**: Third-party integrations that are community maintained.
- **langgraph**: Orchestration framework for combining LangChain components into production-ready applications with persistence, streaming, and other key features. See [LangGraph documentation](#).

- RAG augments LLM prompts with external data retrieval.
- Fetches relevant, up-to-date, or domain-specific information.
- Grounds LLM responses to reduce hallucination and outdated knowledge.
- Bridges the gap between static LLM knowledge and dynamic information needs.
- Delivers factual, relevant, and context-aware outputs.

## Popular Vector Databases

- FAISS: High-speed ANN, Facebook AI, CPU/GPU optimized.
- Pinecone: Fully managed, scalable, integrates with RAG pipelines.
- Weaviate, Milvus, ChromaDB: Open-source, multimodal, GraphQL/REST APIs.



- A vector database

ONNX convert all-MiniLM-L6-v2

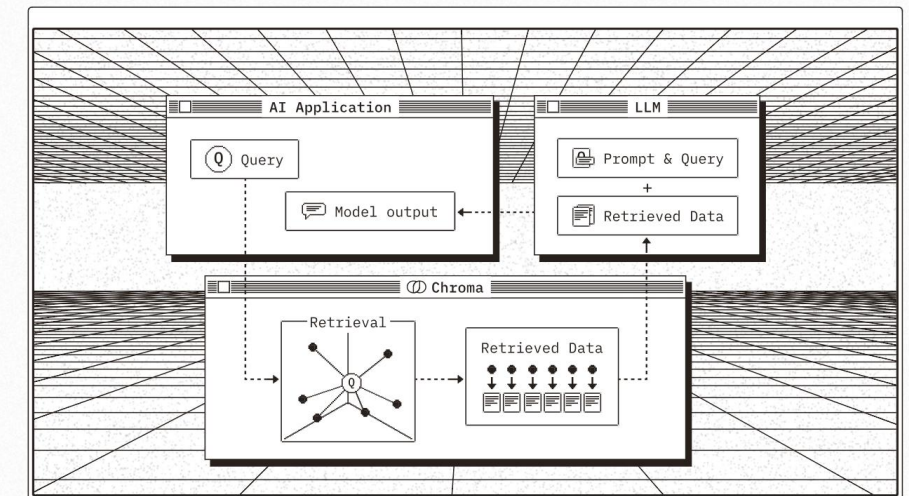
Conversion of sentence-transformers/all-MiniLM-L6-v2

This is a sentence-transformers model: It maps sentences & paragraphs to a 384 dimensional dense vector space and can be used for tasks like clustering or semantic search.

Code Demo

Chroma is the open-source AI application database. Batteries included.

Embeddings, vector search, document storage, full-text search, metadata filtering, and multi-modal. All in one place. Retrieval that just works. As it should be.

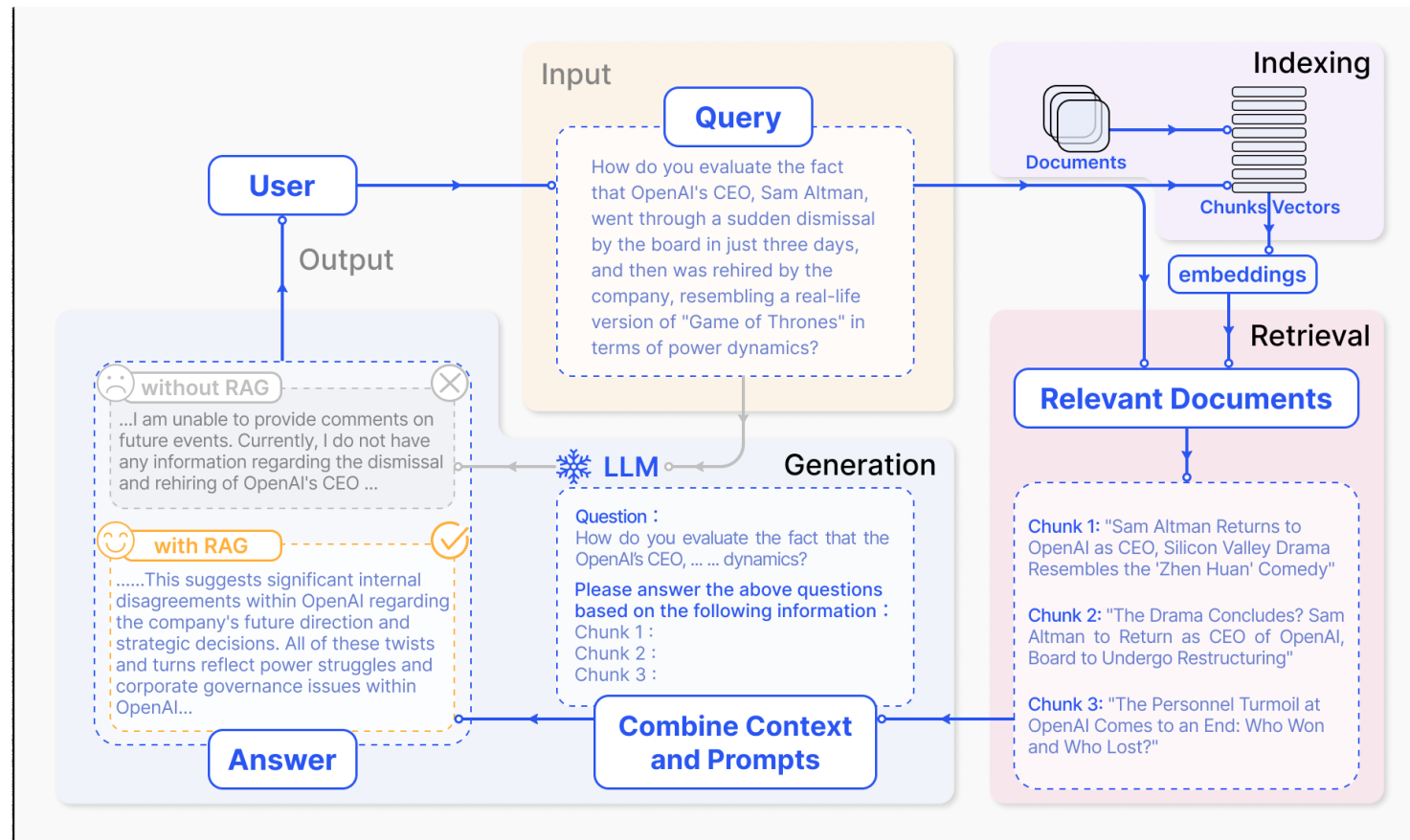




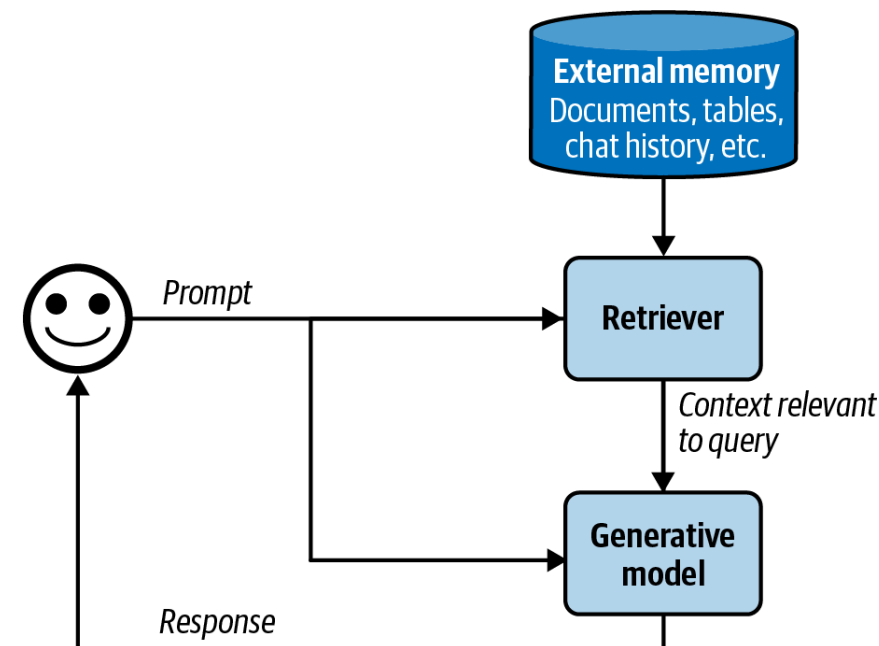
- Data used to train LLMs often lags in timeliness and may not cover all domain knowledge
  - Hallucinations
  - Outdated Knowledge
  - Non-transparent and untraceable reasoning
- Data augmented LLMs can
  - Exhibit domain expert-level capabilities like doctors, lawyers
  - Generate responses based on real data – minimizing the possibility of hallucinations
- Data augmented LLMs application primarily follows:
  - RAG
  - Fine-tuning

# Retrieval Augmented Generation

- RAG enhances accuracy and credibility of the generation – merging LLM’s intrinsic knowledge (training data) with the vast, dynamic repo of external databases leveraging –
  - Vector database, Embeddings, Sentence transformers, Cosine distance



- **Naïve RAG** –follows a "Retrieve-Read" framework involving indexing, retrieval, and generation
  - **Indexing Process:** Raw data from various formats (PDF, HTML, Word) converted into vectors
  - **Retrieval Phase:** Retrieves the most relevant chunks for the expanded prompt context
  - **Generation Phase:** The model generates responses by synthesizing the query and retrieved documents



- **Drawbacks –**
  - If the retriever **fetches low-quality or irrelevant data**, the generator produces **poor responses**
  - **Generation Issues:** Hallucination, irrelevance, bias, or toxic content can arise in the generated responses.
  - **Augmentation Hurdles:** Combining retrieved information can be difficult, leading to redundancy, disjointed responses, and over-reliance on the retrieved content.

- **Retrieval in RAG:** Retrieval is a fundamental concept in search engines, recommender systems, and log analytics
  - It ranks documents based on relevance to a given query.
- **Sparse retrieval (Term-based):** Uses one-hot vectors, where terms are indexed directly (e.g., TF-IDF, BM25).
- **Dense retrieval (Embedding-based):** Uses dense vectors from neural embeddings (e.g., BERT-based retrieval).

- **Term-based Retrieval (Lexical Search):** Finds documents based on keyword matches.
  - **TF-IDF:** Ranks documents based on term frequency and inverse document frequency.
  - **BM25:** Improves TF-IDF by normalizing term frequency and accounting for document length.
  - Uses inverted indexes for fast retrieval.
- **Challenges in Term-based Retrieval:**
  - Does not capture semantic meaning.
  - Struggles with synonymy and polysemy (e.g., "car" vs. "automobile").



- Embedding-based Retrieval (Semantic Search): Maps text into high-dimensional vector space.
  - Uses vector similarity (cosine similarity, Euclidean distance) to find relevant documents.
  - Requires a vector database for efficient retrieval.
- Vector Search: Uses nearest neighbor search (k-NN) to retrieve similar documents.
  - Approximate Nearest Neighbor (ANN) speeds up retrieval for large datasets.
  - Popular libraries: FAISS, Annoy, ScaNN, HNSW.

- **How ANN Works?** – ANN partitions the search space
- It follows a two-step process:
- Indexing Phase: Organizes vectors using clustering, hashing, or graph-based methods.
- Search Phase: Retrieves only the most relevant candidate vectors, skipping unnecessary computations.
- Common ANN techniques include:
  - Graph-based search (HNSW): Constructs a multi-layered graph for efficient traversal.
  - Hash-based search (LSH): Maps similar vectors into the same "buckets."
  - Clustering-based search (IVF-PQ): Groups vectors and searches within relevant clusters.

- Evaluating LLMs in general
  - **Classification:** Measure the correctness of model outputs in tasks like classification, retrieval, or decision-making
  - **Semantic Similarity:** Compares the generated text to the ground truth or reference answer
    - cosine similarity, BLEU, ROUGE, perplexity
  - **Latency (Response Time)** - Measures how quickly the LLM completes a task or generates an answer
  - **Cost** - Evaluates the computational resources and financial expense required to run the model

- **Evaluating Retrieval Quality:**













- **Precision:** Percentage of retrieved documents that are relevant to query.
  - High precision avoid unnecessary noises
- **Recall:** Percentage of relevant documents that were retrieved.
  - High Recall ensures important information is not missed
  - **Recall@K** – how many of the total relevant documents are found in the top k retrieved documents

$$\text{Precision} = \frac{\text{Number of Relevant Documents Retrieved}}{\text{Total Number of Documents Retrieved}}$$

*P@K corresponds to the number of relevant docs among the top K retrieved documents*

$$\text{Recall} = \frac{\text{Number of Relevant Documents Retrieved}}{\text{Total Number of Relevant Documents in the Corpus}}$$

- **Evaluating Retrieval Quality:**
  - **Noise Robustness** – Tests how well the retriever limits the retrieval of irrelevant or misleading documents
    - $NR = \# \text{ irrelevant docs retrieved} / \text{total docs retrieved}$
  - **Mean Reciprocal Rank (MRR):** Evaluates the position of the **first relevant document** in the retrieved list.
    - **Higher MRR → Better chance** the generator gets the **best context** early

Query	Ranked Results (relevant marked in  )	First Relevant Rank	Reciprocal Rank (RR)
Query 1	 Doc3,  Doc5,  Doc10,  Doc7	Rank 1	1/1 = 1.0
Query 2	 Doc8,  Doc1,  Doc9,  Doc4	Rank 2	1/2 = 0.5
Query 3	 Doc6,  Doc11,  Doc2 (no relevant)	No relevant found	0

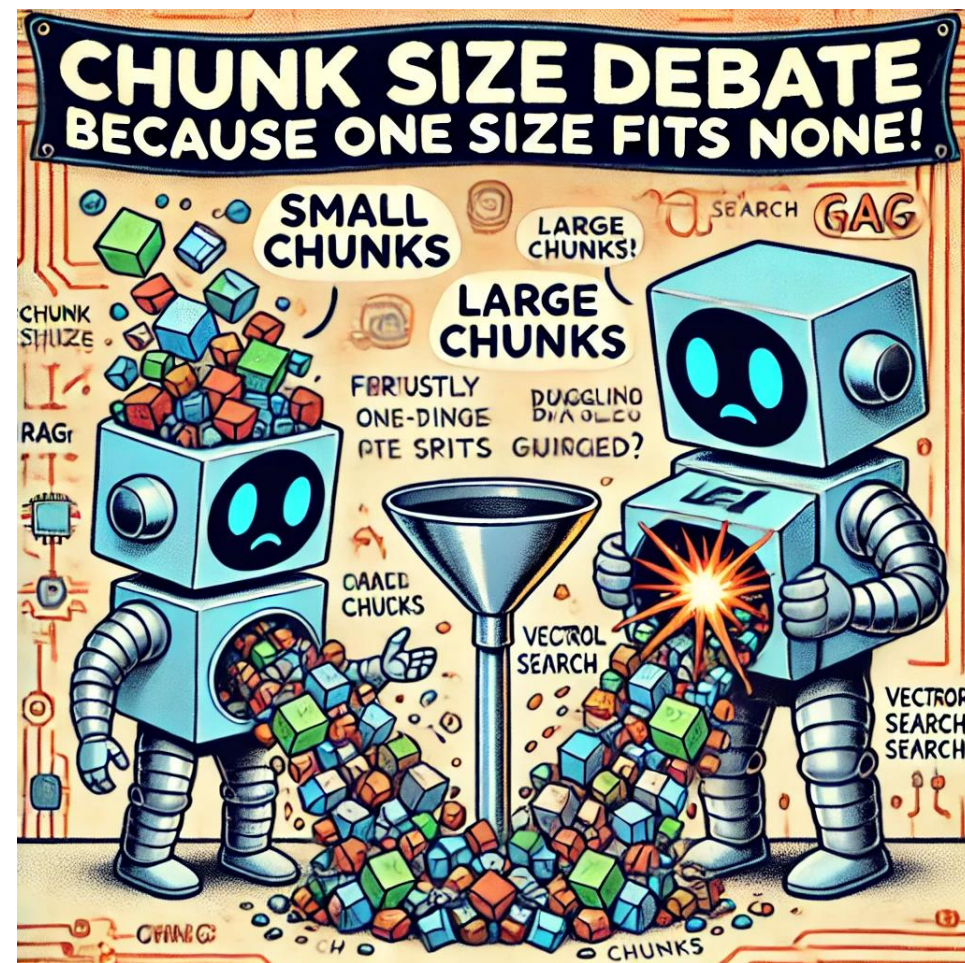
## Evaluating the Generator in RAG

- **Answer Accuracy / Exact Match (EM)** – checks if the generated answer exactly matches the ground truth answer.
- **Semantic Similarity** – Evaluates how semantically close the generated answer - BLEU, ROUGE, BERTScore, or cosine similarity on embeddings
- **Knowledge Gap Detection** – Evaluates the generator's ability to recognize when it doesn't know
- **Groundedness / Faithfulness** – Assesses whether the generated content is grounded in the retrieved documents rather than hallucinating from the model's parametric memory



- Retrieval optimization improves RAG performance through chunking, reranking, query rewriting, and contextual retrieval.
- Chunking strategy directly impacts search relevance, efficiency, and the quality of retrieved context.
- Fixed-length chunking (e.g., 512 words or 2,048 characters) offers simplicity but risks cutting off important context.
- Recursive chunking breaks content into smaller units (sections → paragraphs → sentences) to preserve meaning and continuity.
- Domain-specific chunking ensures accuracy in specialized content like code, Q&A documents, or languages with unique structures.

- **Overlapping Chunks** preserve boundary information
- Token-Based Chunking aligns with the model's tokenizer (e.g., LLaMA 3), ensuring chunks respect token boundaries for better processing efficiency.
  - **Smaller chunks** improve retrieval diversity but increase the number of embeddings
  - **Larger chunks** reduce search complexity but can dilute retrieval precision
- No universal best chunk size exists
  - Smaller chunks enhance flexibility but risk losing context; larger chunks maintain context but reduce retrieval adaptability



- **Reranking** improves retrieval accuracy by refining initial document rankings.
  - **Two-Stage Retrieval:** Uses a fast retriever for candidates, then a precise reranker for ordering.
  - **Time-Based Reranking:** Prioritizes recent data for time-sensitive tasks like news and finance.
  - **Context Positioning:** LLMs focus more on documents at the start and end of the input window.
- **Efficiency Trade-off:** Reranking optimizes token usage while keeping retrieved content relevant.

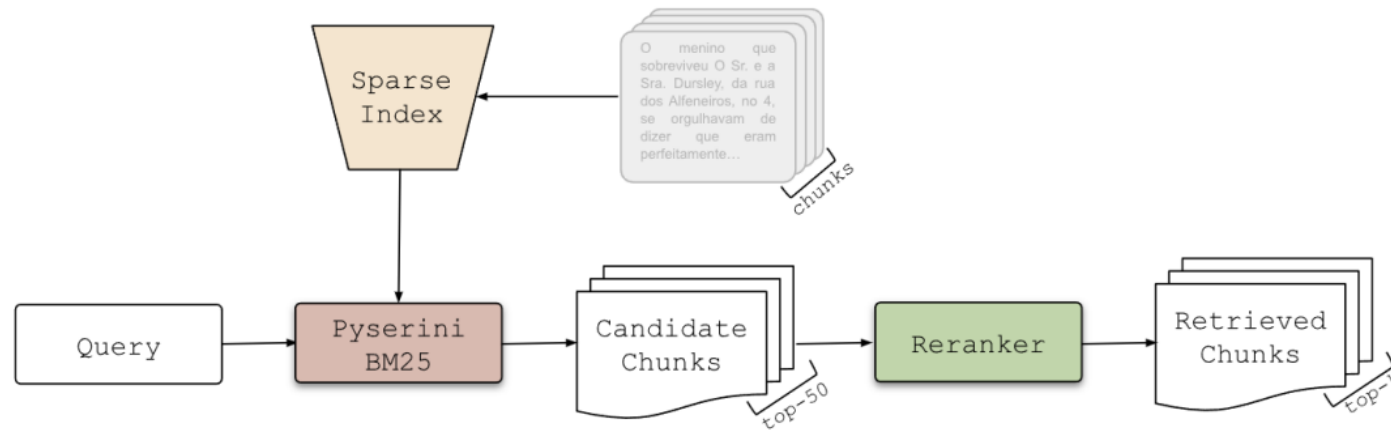


Figure 9: Reranker Pipeline

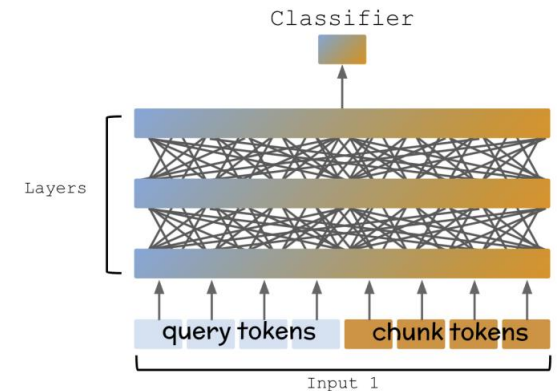


Figure 10: Cross-Encoder

- RAG improves LLM accuracy by retrieving external, task-specific, or private data
- Recent LLMs can process much longer inputs—up to 2 million tokens
  - Do longer context windows make RAG better?
  - Can long context LLMs replace RAG entirely?
- In this study:
  - 20 open-source LLMs
  - Context lengths varied: 2k to 128k

## Long Context RAG Performance of Large Language Models

Quinn Leng\*  
Databricks Mosaic Research

Jacob Portes\*  
Databricks Mosaic Research

- ❖ Results show longer context does not always lead to better RAG performance
- ❖ Many models improved at first, but performance often dropped as context length increased
- ❖ Only a few state-of-the-art LLMs maintained high accuracy beyond 64,000 tokens



For the retrieval stage, we retrieved document chunks using the same embedding model across all settings (OpenAI text-embedding-3-large2 with a chunk size of 512 tokens and a stride of 256 tokens) and used FAISS3 (with IndexFlatL2 index) as the vector store. These chunks were then inserted into the context window of a generative model.

### E.1 Databricks DocsQA

You are a helpful assistant good at answering questions related to databricks products or spark features. You'll be provided with a question and several passages that might be relevant. Your task is to provide an answer based on the question and passages.

Note that passages might not be relevant to the question, so only use the passages that are relevant. If no relevant passage is provided, answer using your knowledge.

The provided passages as context:

{context}

The question to answer:

{question}

Your answer:

DATA 266 | Spring 25

### Long Context RAG Performance of Large Language Models

For generation, we set the temperature to 0.0 and the maximum output sequence length to 1024. We used a simple prompt template to combine the retrieved documents with the user query for each model and dataset

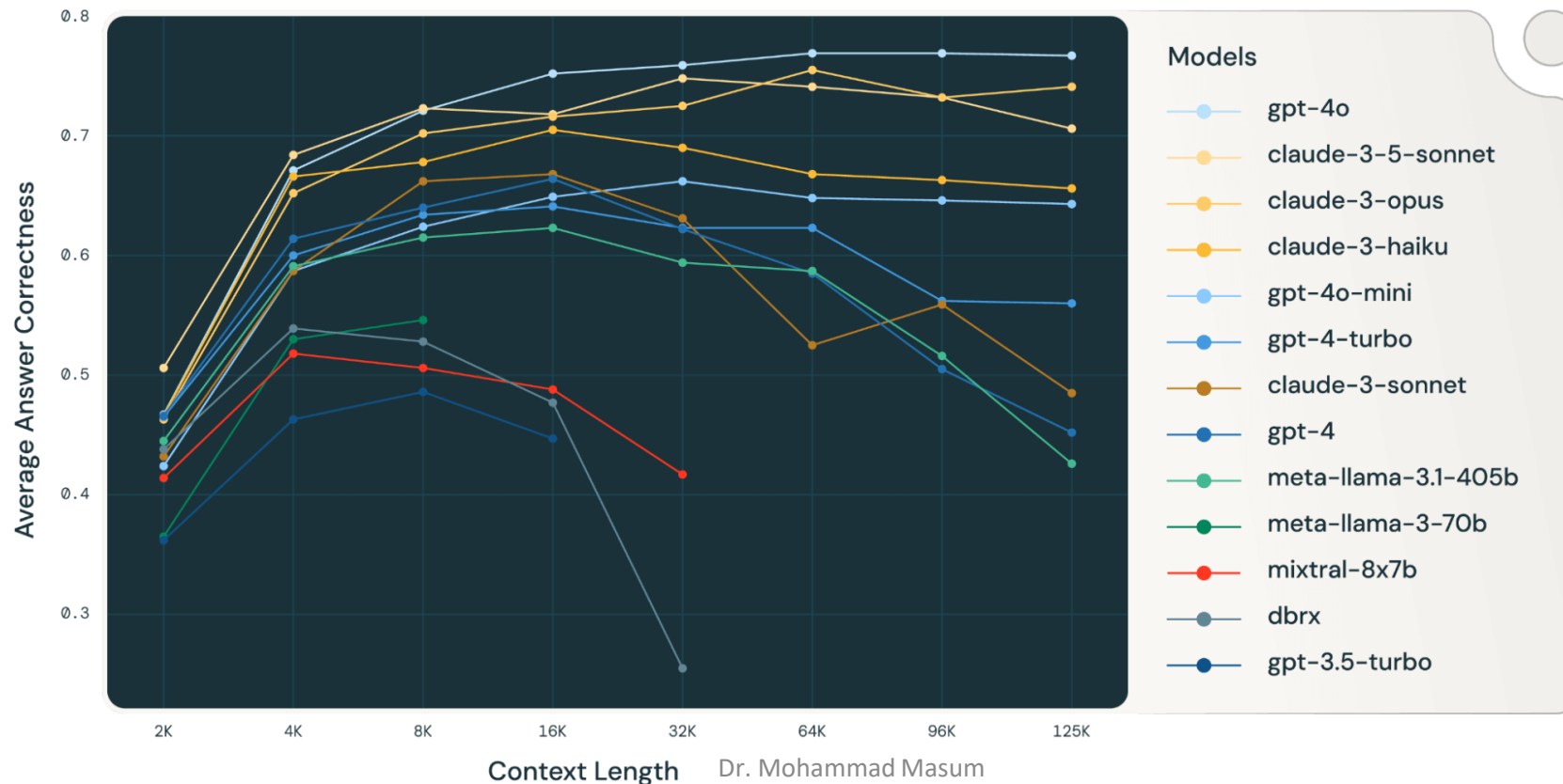
Dataset	Corpus	Queries	Av. doc length (tokens)	Max doc length (tokens)
Databricks DocsQA	7,563	139	2856	225,941
FinanceBench	53,399	150	811	8,633
Natural Questions (dev split)	7,369	534	11,354	13,362

Table S2: Dataset details for the 3 datasets used in our end-to-end RAG benchmark.

## Long Context RAG Performance of Large Language Models

Quinn Leng  
Databricks Mosaic Research

Jacob Portes\*  
Databricks Mosaic Research





- NQ dataset saturates early at 8k tokens
- DocsQA dataset saturates at 96k tokens
- FinanceBench dataset saturates at 128k tokens

## Long Context RAG Performance of Large Language Models

Quinn Leng\*  
Databricks Mosaic Research

Jacob Portes\*  
Databricks Mosaic Research

Num. Retrieved chunks	1	5	13	29	61	125	189	253	317	381
Context Length	2k	4k	8k	16k	32k	64k	96k	128k	160k	192k
Databricks Doc-sQA	0.547	0.856	0.906	0.957	0.978	0.986	0.993	0.993	0.993	0.993
FinanceBench	0.097	0.287	0.493	0.603	0.764	0.856	0.916	0.916	0.916	0.916
NQ	0.845	0.992	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Table S5: Retrieval performance (recall@k) for OpenAI text-embedding-3-large, which was used as the retriever in all of our experiments.

Model	256k	512k	1024k	1500k	2000k
Gemini 1.5 Pro	0.633	0.615	0.627	0.619	0.609
Gemini 1.5 Flash	0.522	0.504	0.514	0.521	0.528

Table S4: Gemini performance above 125k tokens

- **Chunk size** is a critical factor: small chunks risk omitting key info; large chunks risk adding irrelevant content.
- Achieving **optimal chunk size** balances context richness with relevance, boosting factual accuracy and answer quality
  - **Chunk size** and **Top-K retrieval** combinations directly impacted response quality
- Optimizing chunk size lays the foundation for further RAG advancements, including better retrieval and ranking strategies

- **Chunk size and Top-K retrieval** combinations directly impacted response quality
- Best similarity scores were observed with chunk sizes of 512 or 1024 tokens
- Llama3-70B-Instruct performed best with 7-9 chunks, utilizing 40-70% of its context window
  - No improvement in the similarity score generated beyond 10 chunks in almost all combinations

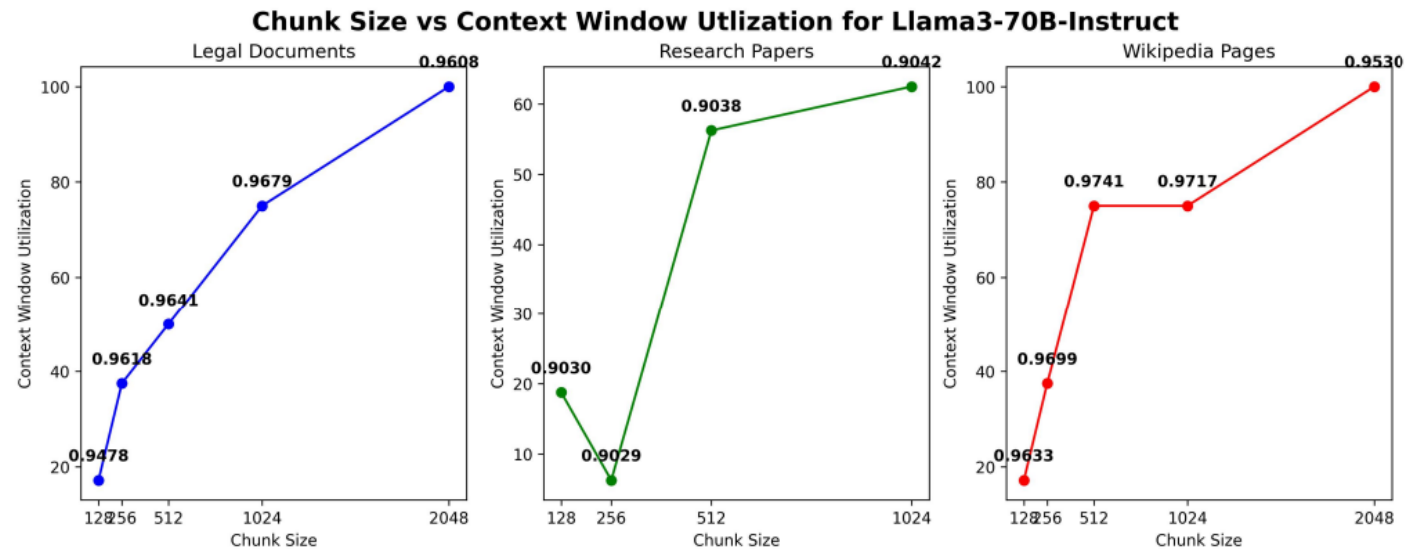


Fig. 3. Chunk Size vs. Context Window Utilization across document types with annotated similarity scores for Llama3-70B-Instruct

- LLMs excel at question-answering but suffer from hallucinations and outdated knowledge.
  - RAG enhances LLM outputs by integrating external knowledge bases.
- Existing RAG systems often fail due to noise in retrieved documents and lack of effective evaluation.
- The paper introduces QA-RAG
  - Focuses on three key RAG abilities: noise robustness, knowledge gap detection, and external truth integration
- Retriever fetches top-k relevant documents from ChromaDB using dense embeddings (MiniLM)
- Generator (LLMs: Llama 2 7b & 13b) processes queries and retrieved docs, producing grounded answers.

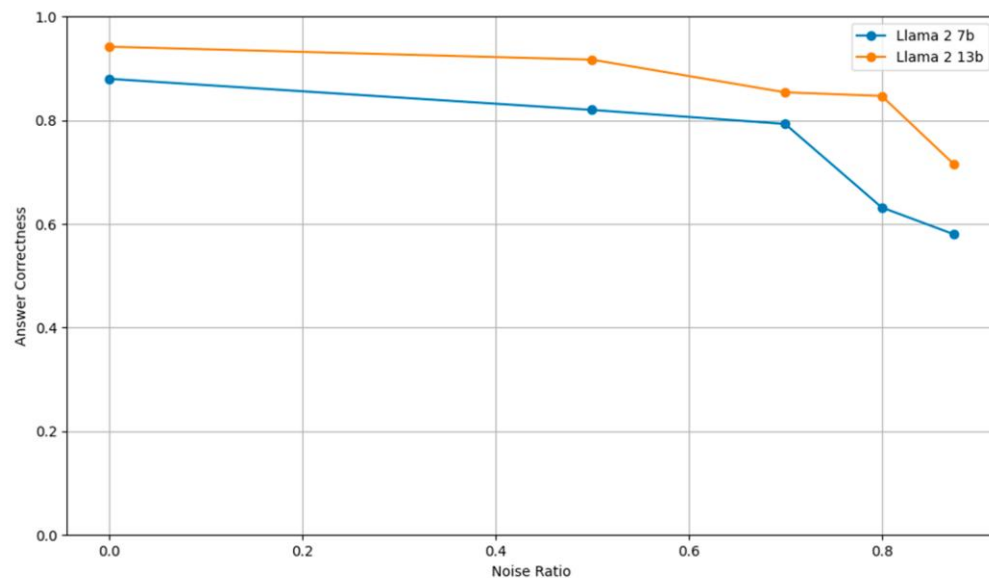
- Dense retriever with top-k=1 delivered best context recall (95%) across datasets.
- Knowledge gap detection: Llama 2-13b identified unanswerable questions with 78% accuracy.
- Noise robustness: Increased top-k introduced more noise, degrading answer quality beyond top-k=3.
- More context or documents doesn't guarantee better answers—optimal top-k and chunk sizing is critical.
- QA-RAG reveals LLMs often favor pre-trained knowledge, even when external facts contradict it.

- LLMs were tested on 500 questions to decide whether to answer or reject based on retrieved information.
- Models were instructed to answer “No” if the required data was missing, simulating knowledge gap detection.
- Results demonstrate that larger models are more effective at recognizing incomplete information and avoiding hallucinated answers.

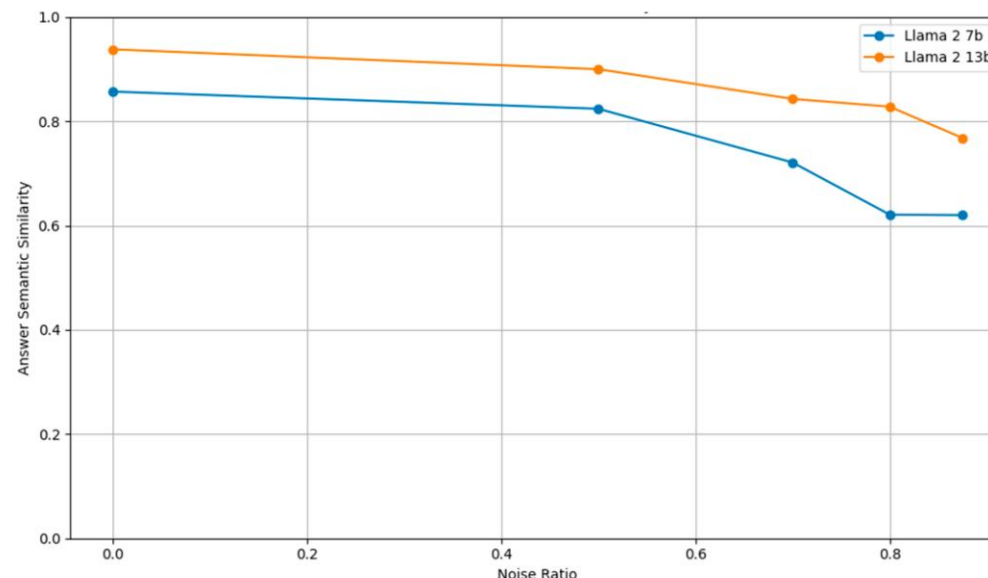
LLM	EM
Llama 2 7b	0.66
Llama 2 13b	0.78



- Evaluated how LLMs handle noisy retrieved contexts with irrelevant or misleading documents.
- Varied top-k retrieval (1, 3, 5, 8) to introduce different noise levels in the input.
- A noise level of 0 provided only the exact relevant context, serving as the baseline.
- Demonstrated LLMs' ability to extract accurate information despite increasing noise, simulating real-world retrieval challenges.

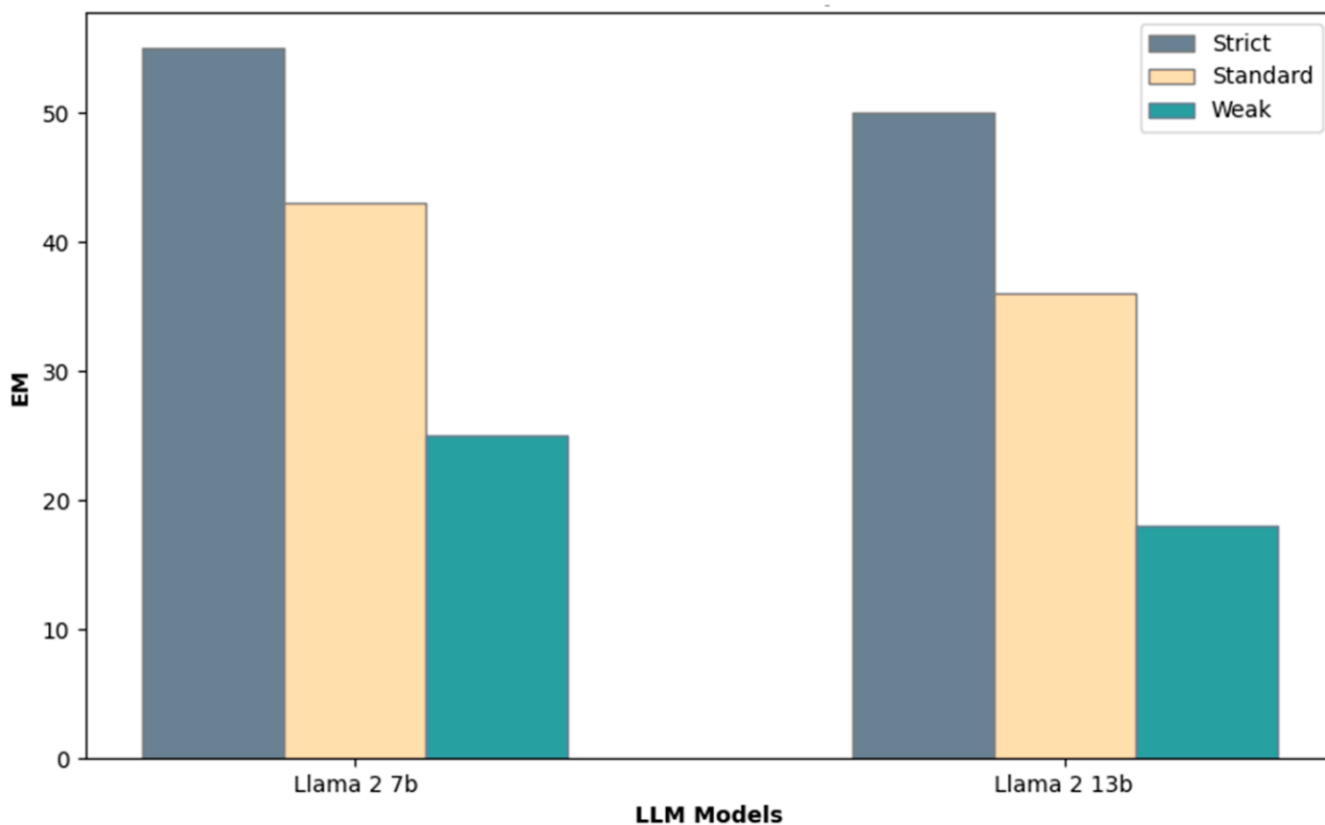


(a)



(b)

### External Truth Integration



#### STRICT PROMPT

You HAVE TO answer ONLY based on the context. Do not introduce any new information. If the context does not contain the information requested, state 'I don't know. Don't make up an answer.

#### STANDARD PROMPT

Use the following pieces of context to answer the question. If you don't know the answer, just say that you don't know. Don't try to make up an answer.

#### WEAK PROMPT

Consider the following context when answering the question. Don't try to make up an answer.

- Directly retrieving by original query does not always achieve correct and relevant documents
- Query rewriting reformulate the query to expand the retrieve docs for a better response
  - Rewritten queries in a way that better preserves the semantics of the original query.

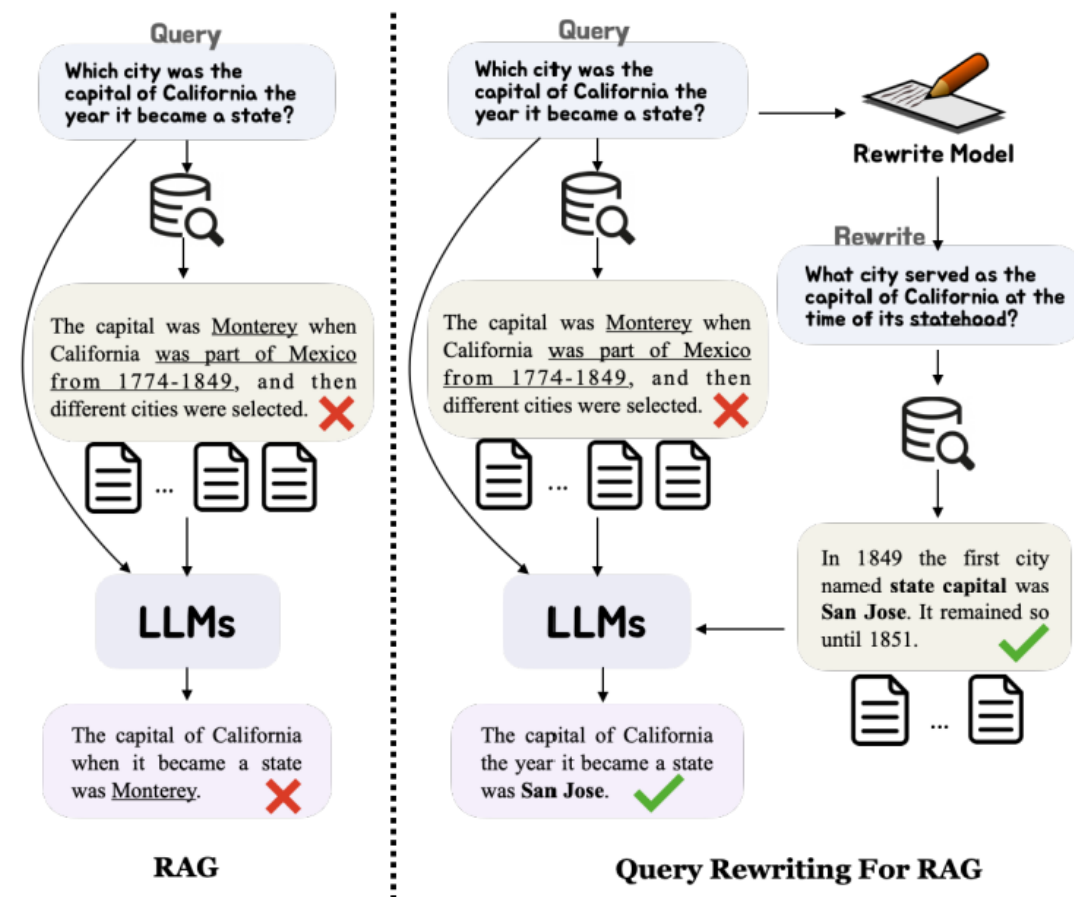


Figure 1: Illustration of query rewriting for RAG. The left part indicates the normal RAG pipeline, while the right part presents the query rewriting to expand more relevant documents for RAG.

<p><b>Original Query</b> <span>Prec@5 4/5 Correct ✓</span></p> <p>What American professional Hawaiian surfer born 18 October 1992 won the Rip Curl Pro Portugal?</p>	<p><b>Original Query</b> <span>Prec@5 0/5 Correct ✗</span></p> <p>What team did the recipient of the 2007 Brownlow Medal play for?</p>	<p><b>Original Query</b> <span>Prec@5 1/5 Correct ✗</span></p> <p>Where is the company that Sachin Warrier worked for as a software engineer headquartered?</p>
<p><b>SFT Rewrite</b> <span>Prec@5 1/5 Correct ✗</span></p> <p>The American Hawaiian surfer with a birthdate of 18 October 1992, <u>which competition</u> did they emerge victorious in, the Rip Curl Pro Portugal?</p>	<p><b>SFT Rewrite</b> <span>Prec@5 1/5 Correct ✗</span></p> <p><u>What squad</u> was the 2007 <b>Brownlow Medal winner</b> a part of?</p>	<p><b>SFT Rewrite</b> <span>Prec@5 1/5 Correct ✗</span></p> <p>Which city is the base of operations for Sachin Warrier's employer as a software engineer?</p>
<p><b>RaFe Rewrite</b> <span>Prec@5 4/5 Correct ✓</span></p> <p><b>Which Hawaiian-American pro surfer</b>, born 18 October 1992, won the Rip Curl Pro Portugal?</p>	<p><b>RaFe Rewrite</b> <span>Prec@5 2/5 Correct ✗</span></p> <p>Which team did the 2007 <b>Brownlow Medal winner</b> play for?</p>	<p><b>RaFe Rewrite</b> <span>Prec@5 3/5 Correct ✓</span></p> <p>Sachin Warrier worked as a software engineer for a company with headquarters where?</p>
(A)	(B)	(C)

Figure 3: Three types of examples, including the original query and rewrites from SFT and RaFe. The Prec@5 results of queries and rewrites are presented, and “Correct” denotes that whether the prediction is correct or not.

Methods	Prec@5	Prec@10	MRR
Original Query	41.41	39.76	54.11
Bad Rewrite	30.74	28.13	43.64
Good Rewrite	<b>46.14</b>	<b>44.34</b>	<b>59.17</b>

Table 5: The comparison of retrieval results between original query and good/bad rewrites.