LLM Interview Series Part 1: Retrieval-Augmented Generation (RAG) - System Design and Implementation

Sagar Sudhakara

Feb 2025

1 How would you design a RAG system for handling large-scale document retrieval?

A scalable Retrieval-Augmented Generation (RAG) system consists of several core components:

Document Processing & Indexing

- **Preprocessing:** Tokenization, normalization, and text cleaning.
- **Chunking:** Segmentation into meaningful chunks using fixed-length, semantic segmentation, or recursive chunking.
- **Embedding Generation:** Use transformer-based models (e.g., BERT, OpenAI Embeddings) to generate dense vector representations.
- **Vector Indexing:** Store embeddings in a vector database (FAISS, Weaviate, Pinecone) optimized for Approximate Nearest Neighbor (ANN) search.

Retrieval Module

- **Sparse Retrieval:** Use BM25 for keyword-based retrieval.
- **Dense Retrieval:** Perform vector similarity search using embeddings.
- **Hybrid Retrieval:** Combine BM25 and dense retrieval for improved accuracy.
- **Reranking:** Apply a cross-encoder model (e.g., MonoT5) for relevance ranking.

LLM Integration

- Retrieved passages are formatted into prompts for an LLM.
- The LLM generates a response using both retrieved and internal knowledge.
- Post-processing techniques refine and validate the final output.

2 How does RAG improve LLM performance? Can you describe its architecture?

RAG enhances LLM performance by reducing hallucinations and incorporating external factual information.

RAG Architecture

- **Query Processing:** Tokenization and optional query expansion.
- **Retrieval Module:** Sparse (BM25), dense (FAISS), or hybrid retrieval.
- **Context Injection:** Retrieved documents are concatenated and passed to the LLM.
- **Generation Module:** The LLM generates an answer using the retrieved context.
- **Post-processing:** Filtering and validation for hallucination mitigation.

3 What are the main challenges in implementing RAG at scale?

Scalability Challenges

- Efficient storage and retrieval from billion-scale document indices.
- High memory consumption for dense embeddings.

Retrieval Latency

- Optimizing Approximate Nearest Neighbor (ANN) search for real-time performance.
- Balancing retrieval speed with accuracy.

Model Reliability

- Mismatched or irrelevant retrieval context can degrade response quality.
- Fine-tuning retrieval models to adapt to new document distributions.

4 What techniques would you use to ensure the retrieved documents are relevant?

The following techniques help improve the precision and recall of the retrieval process:

- **Hybrid Retrieval:** This approach combines traditional lexical retrieval methods such as BM25 with dense retrieval models (e.g., embedding-based methods like DPR or ColBERT). BM25 excels at exact keyword matches, while dense retrieval captures semantic similarities. By combining both, the system can improve retrieval accuracy, ensuring that relevant documents that might not contain exact query terms but have similar meanings are still retrieved.
- **Cross-Encoder Reranking:** After retrieving an initial set of top-k documents, a cross-encoder model (e.g., MonoT5 or BERT-based rerankers) can be used to re-rank the documents based on query-document relevance. Unlike bi-encoder models that independently encode queries and documents, cross-encoders jointly process them, allowing for richer contextual understanding and better ranking precision.
- **Query Expansion:** This technique improves recall by reformulating user queries to include semantically similar terms. Query expansion can be done using:
  - Pseudo-relevance feedback (e.g., RM3 model) to extract additional relevant terms.
  - Synonym and paraphrase generation using LLMs.
  - Explicit expansion using knowledge bases like WordNet.

  This helps in cases where users may not phrase their queries in the exact way the relevant documents are indexed.

- **Multi-Hop Retrieval:** For complex queries requiring reasoning over multiple pieces of evidence, multi-hop retrieval retrieves documents iteratively. Instead of fetching all documents in one step, the system retrieves an initial document, extracts key information, and then performs a second retrieval step based on the extracted content. This is particularly useful in answering multi-faceted questions that require integrating information from different sources.
- **Embedding Space Clustering:** By grouping similar documents in the embedding space, retrieval can be made more efficient and context-aware. Clustering techniques such as k-means or HNSW (Hierarchical Navigable Small World graphs) can help retrieve documents that are not just top-ranked individually but belong to semantically related clusters.
- **Filtering and Metadata Constraints:** Applying document filtering based on metadata such as timestamps, document type, or author can refine retrieval results. This ensures that only relevant and up-to-date documents are considered.
- **Context-Aware Retrieval:** Incorporating user-specific context (e.g., domain expertise, past queries, or session history) can personalize retrieval, making it more relevant for specific use cases.

5 How do long-context retrieval techniques (e.g., sliding window, chunking, RePAQ) improve RAG?

Handling long documents in Retrieval-Augmented Generation (RAG) systems presents a challenge because many retrieval models and large language models (LLMs) have token limitations. Long-context retrieval techniques help optimize information retrieval by ensuring that relevant sections of long documents are effectively retrieved while maintaining coherence and context. Below are some key techniques:

- **Sliding Window Retrieval:** This technique segments documents into overlapping chunks of a fixed size. By maintaining overlaps between consecutive chunks, the system preserves contextual continuity, preventing important information from being lost at chunk boundaries. The sliding window approach is particularly useful when dealing with structured documents, where key information spans multiple paragraphs or pages. A typical implementation involves:
  - Setting an optimal window size (e.g., 512 or 1024 tokens).
  - Using a stride (overlap) to maintain context (e.g., 50% overlap).
  - Encoding each chunk separately and retrieving based on embeddings.

  This ensures that relevant sections are not cut off due to arbitrary chunking.
- **Dynamic Chunking (RePAQ - Recursive Passage Aggregation for Querying):** Unlike fixed-size chunking, RePAQ employs an adaptive chunking strategy that recursively refines passage selection based on query relevance. Instead of retrieving static document segments, RePAQ dynamically adjusts passage lengths by:
  - Aggregating smaller, semantically related passages to form contextually rich responses.
  - Dynamically expanding or contracting chunks based on retrieval performance.
  - Reducing redundancy by merging overlapping or highly similar passages.

  This method ensures that retrieved passages contain the most relevant content without unnecessary truncation.
- **Summarization-Based Compression:** When dealing with extensive documents, summarization techniques can be applied to condense information while preserving key details. There are two main approaches:
  - **Extractive Summarization:** Selects key sentences from the document (e.g., using BERTSUM or TextRank) to create a shorter, high-relevance version of the content.
  - **Abstractive Summarization:** Generates a condensed version of the content in a new form, preserving meaning while reducing length (e.g., using T5 or GPT-based models).

This technique is particularly useful for processing lengthy reports, research papers, or legal documents where only the core information is required for retrieval.

- **Hierarchical Retrieval:** Instead of treating all document chunks equally, a hierarchical retrieval strategy first retrieves broad, high-level summaries before diving into more granular content. This can be implemented by:
  - Using metadata or section headings to prioritize document sections.
  - Employing multi-stage retrieval where high-level chunks are retrieved first, followed by more specific, fine-grained passages.
  - Utilizing a two-step reranking approach to refine retrieved results.
- **Multi-Vector Representations:** Instead of embedding entire chunks as a single vector, multi-vector techniques (e.g., ColBERT) allow for finer-grained token-level retrieval. This improves retrieval accuracy by enabling a more nuanced understanding of long texts, especially when dealing with complex or technical documents.
- **Memory-Augmented Retrieval:** Some RAG systems store intermediate retrieval results and dynamically adapt based on prior queries. This ensures that relevant context is not repeatedly lost and allows for efficient cross-referencing of long documents.

6 How do you choose between sparse (BM25) and dense (FAISS, ColBERT) retrieval for a specific use case?

The choice between sparse and dense retrieval depends on the nature of the data, query complexity, and computational constraints.

Sparse Retrieval (BM25, TF-IDF)

- Best suited for **keyword-based search** where exact term matching is critical.
- Works well on **structured text** like FAQs, legal documents, and web search.
- Computationally efficient and interpretable but lacks semantic understanding.

Dense Retrieval (FAISS, ColBERT)

- Uses deep learning models to generate **contextual embeddings**, allowing semantic similarity matching.
- Suitable for **unstructured text, paraphrased content, and multi-lingual retrieval**.
- Computationally expensive, requiring GPU-accelerated vector search.

Feature	**Sparse Retrieval (BM25)**	**Dense Retrieval (FAISS, ColBERT)**
Representation	Term frequency-based scoring	Learned embeddings from neural models
Query Matching	Exact word matches	Semantic similarity
Interpretability	High (TF-IDF weighting)	Low (vector-based similarity)
Computational Cost	Low	High (GPU-intensive)
Performance on Long Queries	Poor	Strong
Multi-lingual Support	Limited	Strong

Table 1: Comparison of Sparse vs. Dense Retrieval

Decision Factors

- If the dataset is **well-structured and query terms are precise** → Use **BM25**.
- If queries require **semantic understanding and paraphrase retrieval** → Use **Dense Retrieval**.
- If computational efficiency is critical → Prefer **sparse retrieval** (BM25).
- If dealing with **long-tail queries and cross-domain data** → Use **hybrid retrieval**.

7 What is Hybrid Retrieval, and when should it be used in RAG?

Hybrid Retrieval combines both **sparse** and **dense** retrieval mechanisms to maximize precision and recall.

How Hybrid Retrieval Works

- **Sparse Retrieval:** Identifies relevant documents based on keyword matching.
- **Dense Retrieval:** Retrieves semantically similar passages based on embeddings.
- **Fusion Strategy:** Combines both results using **weighted scoring** or **reciprocal rank fusion (RRF)**.

When to Use Hybrid Retrieval

- When dealing with **diverse query types** (e.g., factual and semantic queries).
- When high recall is needed in **long-context retrieval**.
- When sparse retrieval alone **fails to capture semantic meaning**.

8 What are contrastive learning approaches (e.g., SimCSE, TSDAE) for improving dense retrieval models?

Contrastive Learning improves retrieval models by training them to **distinguish relevant and irrelevant document pairs**.

Contrastive Learning Techniques

- **SimCSE (Simple Contrastive Sentence Embeddings):** - Uses a **self-supervised** approach where different views of the same sentence are treated as positive pairs. - Trains models to **maximize similarity** between identical sentences while minimizing similarity with random negatives.
- **TSDAE (Text-to-Text Denoising Autoencoder):** - Learns embeddings by reconstructing corrupted text input. - Effective for domain adaptation in dense retrieval.

Benefits of Contrastive Learning in Retrieval

- Helps create more **discriminative embeddings**, reducing false positives.
- Improves retrieval performance on **out-of-domain and low-resource datasets**.
- Enhances robustness against **synonym variations and paraphrasing**.

9 How does Multi-Vector Retrieval (e.g., ColBERT) differ from Single-Vector Retrieval (e.g., BERT, CLIP)?

Retrieval models differ in how they encode and compare documents with queries. Traditional **single-vector retrieval** methods map entire documents into a single dense vector, making them efficient but often imprecise for longer texts. In contrast, **multi-vector retrieval** assigns multiple embeddings per document, enabling more fine-grained matching at the token level. This distinction significantly impacts retrieval accuracy, storage requirements, and computational efficiency.

Single-Vector Retrieval (BERT, CLIP)

- Encodes each document into **a single fixed-size vector**, summarizing the entire content into one numerical representation.
- Well-suited for **short queries and small datasets** due to its lower storage and computational cost.
- Matches documents using **cosine similarity or dot-product** between query and document embeddings.
- Struggles with capturing **fine-grained relevance signals**, particularly in long documents where critical details may be lost in compression.

Multi-Vector Retrieval (ColBERT)

- Represents each document using **multiple token-level embeddings**, preserving granular meaning.
- Implements a **Late Interaction Mechanism**, where token embeddings from queries and documents interact **post-retrieval**, improving precision.
- Enables **soft matching of query terms** by comparing individual token embeddings rather than entire document vectors.
- Provides **higher retrieval accuracy** at the cost of increased storage and inference latency.

Key Differences Between Single-Vector and Multi-Vector Retrieval

Feature	**Single-Vector Retrieval (BERT, CLIP)**	**Multi-Vector Retrieval (ColBERT)**
Encoding Type	One vector per document	Multiple vectors per document (token-wise
Query Matching	Global embedding similarity	Token-level interaction (Late Interaction)
Context Sensitivity	Limited due to compression	High, retains local token relationships
Storage Requirement	Low	High (multiple vectors per document)
Precision	Lower, struggles with long documents	Higher, captures finer word importance
Computational Cost	Low (fast lookup)	Higher due to token-wise comparisons

Table 2: Comparison of Single-Vector vs. Multi-Vector Retrieval

When to Use Each Approach?

- Use **Single-Vector Retrieval** when:
  - Speed and efficiency are priorities.
  - The dataset consists of short documents or well-defined queries.
  - Memory and storage constraints limit embedding sizes.
- Use **Multi-Vector Retrieval** when:
  - Fine-grained token-level retrieval is necessary.
  - The dataset contains long, complex documents where precise matching is required.
  - Latency and storage constraints are less critical than accuracy.

10 What are some efficient vector database architectures used in production-scale RAG systems?

Vector databases are crucial for efficient similarity search in large-scale Retrieval-Augmented Generation (RAG) systems. The most widely used architectures include:

- **FAISS (Facebook AI Similarity Search)** - Uses IVF-PQ (Inverted File Index with Product Quantization) for scalable retrieval. - Efficient for billion-scale datasets using GPU acceleration.
- **ScaNN (Scalable Nearest Neighbors by Google)** - Optimized for low-latency and high-recall ANN searches. - Supports hybrid sparse + dense retrieval models.
- **Weaviate, Pinecone, Milvus** - Cloud-based vector databases optimized for distributed query handling. - Scales efficiently with multi-node clusters and sharded indexes.

11 How do you efficiently index billions of documents for fast retrieval in a RAG system?

Handling billion-scale retrieval efficiently in a Retrieval-Augmented Generation (RAG) system requires a combination of optimized data structures, parallel processing, and memory-efficient indexing techniques. Below are key strategies to achieve fast and scalable document retrieval:

- **Hierarchical Indexing** - Implement multi-level indexing (e.g., IVF-HNSW) to partition the search space hierarchically. - Inverted File Index (IVF) structures help reduce the number of candidate vectors searched. - Hierarchical Navigable Small World (HNSW) graphs allow fast traversal within indexed spaces. - Combine hierarchical tree-based approaches (e.g., KD-Trees, Ball Trees) with ANN search for hybrid retrieval.
- **Approximate Nearest Neighbor (ANN) Search** - Use scalable ANN libraries such as FAISS (Facebook AI Similarity Search), ScaNN (Google's Scalable Nearest Neighbors), or HNSW to efficiently retrieve the closest embeddings. - Implement graph-based retrieval (e.g., HNSW) to optimize for recall and speed. - Optimize query efficiency by tuning parameters such as 'nprobe' in IVF-based FAISS indexes to balance speed vs. accuracy.
- **Vector Compression** - Reduce storage footprint and speed up retrieval using quantization techniques: - Product Quantization (PQ) divides vectors into smaller subspaces, reducing memory usage while preserving search accuracy. - Scalar Quantization (SQ) compresses embeddings by mapping values to a fixed set of centroids. - Binary Quantization (BQ) converts vectors into binary format for ultra-fast Hamming distance calculations. - Adaptive quantization techniques can be applied dynamically to balance precision and compression rate.
- **Distributed Indexing** - Scale indexing horizontally by sharding vector databases across multiple servers (e.g., FAISS on Kubernetes, Milvus, Weaviate). - Employ vector hashing techniques (e.g., Locality-Sensitive Hashing - LSH) to distribute similar embeddings to the same shard. - Use graph partitioning to minimize inter-node search overhead in distributed vector retrieval. - Implement query routing strategies to direct searches to the most relevant partitions based on metadata filtering.

12 What are the scalability challenges of k-NN retrieval in FAISS, Annoy, or ScaNN?

Despite the efficiency gains from Approximate Nearest Neighbor (ANN) search, k-NN retrieval at a large scale introduces several key challenges:

- **Memory Overhead** - Storing high-dimensional embeddings for billions of documents requires terabytes of RAM or GPU memory, making cost-effective scaling difficult. - FAISS provides quantization techniques (IVFPQ, OPQ), but trade-offs between speed, precision, and memory footprint must be carefully tuned. - Disk-based retrieval (e.g., DiskANN) can be leveraged when RAM is a bottleneck.

- **Indexing Latency** - Constructing k-NN graphs (e.g., HNSW, VP-Trees) is computationally expensive, especially when inserting new embeddings in real-time streaming pipelines. - Dynamic reindexing is challenging due to the need for balancing search efficiency with incremental updates. - Graph rewiring (in HNSW) adds extra computation overhead as edges must be optimized for fast traversal.
- **Load Balancing in Distributed Retrieval** - Multi-node retrieval setups introduce query routing complexity, where queries must be directed efficiently to relevant index shards. - Embedding skew can occur if certain clusters contain disproportionately large amounts of data, leading to uneven workload distribution. - Deploying vector databases like Milvus, Weaviate, or Vespa with dynamic shard balancing mitigates these issues but requires additional orchestration.
- **Cold Start and Index Refreshing** - When new documents arrive, re-indexing embeddings can lead to downtime or temporary inconsistency. - Techniques such as background index updating and delayed ANN refreshing (e.g., in FAISS) help mitigate performance drops. - Real-time document updates require hybrid approaches, where fast metadata filtering is used alongside ANN-based retrieval.

13 How would you handle retrieval latency issues when integrating RAG with an LLM?

To reduce retrieval latency in a RAG system:

- **Pre-computed Caches:** Store frequent query embeddings to avoid redundant computation.
- **Optimized ANN Search:** Use hierarchical or graph-based ANN structures.
- **Batch Query Execution:** Retrieve embeddings for multiple queries in parallel.

14 What techniques would you use to make a low-latency, real-time RAG system?

- **Asynchronous Processing:** - Decouple retrieval and generation steps using non-blocking queries.
- **Sparse-Dense Hybrid Models:** - Combine BM25 for fast initial search, followed by dense retrieval reranking.
- **Memory-Efficient Embeddings:** - Use knowledge distillation or quantized embeddings to reduce footprint.

15 How do you evaluate the relevance of retrieved passages before passing them to an LLM?

Relevance evaluation techniques:

- **Embedding Similarity:** - Compute cosine similarity between query and retrieved embeddings.
- **Cross-Encoder Reranking:** - Use fine-tuned cross-encoder models (e.g., MonoT5) for ranking quality.
- **Human Evaluation:** - Manually score retrieved passages for precision and recall.

16 Can you explain query expansion and its impact on retrieval in RAG systems?

Query expansion enhances retrieval effectiveness by reformulating the user query to include additional terms or representations that improve document matching. This technique is crucial in Retrieval-Augmented Generation (RAG) systems, where recall and relevance play a key role in downstream answer generation.

Types of Query Expansion

- **Lexical Expansion (BM25 and Traditional IR)** - Expands queries using synonyms, stemming, lemmatization, and term variants. - Example: Expanding "AI regulation" to include "artificial intelligence laws" and "machine learning policies". - Techniques include thesaurus-based expansion (e.g., WordNet) and pseudo-relevance feedback (e.g., Rocchio algorithm). - Works well in keyword-based retrieval but may not capture deep semantic meaning.
- **Semantic Expansion (Dense Retrieval and LLM-based)** - Leverages embeddings and contextual similarity to expand queries with paraphrases or related concepts. - Example: Using a Transformer model (e.g., BERT, T5, GPT-4) to rephrase "latest research on deep learning" into "recent advances in neural networks". - Uses vector space rather than simple term-matching, making it more effective for capturing conceptual meaning.

Impact of Query Expansion on RAG Systems

- **Improves Recall** - Helps retrieve more relevant documents, reducing false negatives. - Particularly useful in zero-shot retrieval, where exact term matches may be missing.
- **May Introduce Noise** - Expanding queries too aggressively may reduce precision, retrieving irrelevant results. - Example: Expanding "COVID-19 vaccine policies" to include "flu vaccine guidelines" might dilute the relevance.
- **Enhances Robustness for Long Queries** - Expansion is useful when handling long-tail queries where vocabulary mismatch is common. - Particularly beneficial in legal, medical, and technical domains where terminology variations exist.
- **Balances Exploration vs. Specificity** - A well-designed expansion strategy optimizes for coverage without excessive generalization. - Hybrid methods (e.g., combining lexical and semantic expansion) yield better results.

17 What are the downsides of naïve text chunking in retrieval, and how would you improve chunk granularity?

Problems:

- **Context Loss:** Fixed-length chunks may cut off meaningful context.
- **Overlapping Chunks:** Excessive overlap increases redundancy.

Solutions:

- **Adaptive Chunking:** Use text segmentation based on topic shifts.
- **Hierarchical Retrieval:** Retrieve entire sections before selecting sentences.

18 How does adaptive retrieval improve the relevance of search results?

Adaptive retrieval enhances search relevance by dynamically modifying retrieval strategies based on query characteristics, user intent, and feedback mechanisms. Unlike static retrieval systems, adaptive retrieval continuously refines search results through context-aware adjustments and learning-based optimization.

Key Adaptive Retrieval Techniques

- **Context-Aware Retrieval** - Adjusts retrieval strategies dynamically based on the query type, document corpus, and historical user interactions. - Uses query embeddings, session history, or metadata (e.g., user location, domain knowledge) to refine document ranking. - Example: In a customer support chatbot, retrieving different knowledge base articles depending on whether a user is troubleshooting or making a purchase inquiry.

- **Reinforcement Learning-Based Retrieval** - Employs reinforcement learning (RL) to optimize retrieval strategies over time by continuously learning from user interactions and feedback. - Uses reward-based learning to adjust ranking policies, promoting high-utility results and demoting irrelevant ones. - Example: A search engine optimizing results by tracking which documents users click on and adjusting retrieval weights accordingly.
- **Multi-Stage Retrieval Adaptation** - Dynamically switches between retrieval models based on query complexity. - For ambiguous queries, expands retrieval scope; for specific queries, narrows the search to highly relevant results. - Example: A legal research system retrieving broad case law references for general queries but focusing on jurisdiction-specific documents for legal professionals.

19 How do you decide when to use reranking (e.g., Cross-Encoders, MonoT5) in a RAG pipeline?

Reranking is a crucial step in Retrieval-Augmented Generation (RAG) pipelines when the initial retrieval results contain noise or when higher precision is required. Reranking models such as Cross-Encoders and MonoT5 improve ranking quality by incorporating deeper semantic understanding.

When to Use Reranking?

- **High-Precision Retrieval Scenarios** - When query-document matching requires deep contextual reasoning. - Useful in legal, medical, and technical search applications where minor ranking differences impact the final response.
- **Multi-Step Retrieval Pipelines** - When the first-stage retriever (e.g., BM25, dense embeddings) produces noisy or broad results, reranking helps refine them. - Essential in hybrid retrieval setups where a combination of keyword-based and semantic search is used.

Trade-offs of Reranking

- **Increased Computation Cost** - Cross-Encoders require pairwise comparisons, leading to higher inference latency compared to lightweight bi-encoder models. - Trade-off between reranking depth (top-10 vs. top-100 results) and retrieval speed.
- **Impact on End-to-End Performance** - While reranking improves precision, excessive use may slow down real-time systems. - A balance between retrieval recall and reranking efficiency is necessary to avoid bottlenecks.

20 What are the main failure points in a RAG pipeline, and how would you mitigate them?

Common Failure Points:

- **Irrelevant Retrieval:** The retrieval system returns passages that are not relevant to the query.
- **Incomplete Context:** Retrieved documents contain partial or insufficient information.
- **Hallucinations:** The LLM generates factually incorrect content when retrieval fails.
- **Latency Bottlenecks:** Large-scale retrieval can slow down response time.

Mitigation Strategies:

- **Hybrid Retrieval:** Combine BM25 with dense retrieval for better coverage.
- **Cross-Encoder Reranking:** Improve passage ranking using models like MonoT5.
- **Fallback Mechanisms:** When retrieval confidence is low, use a static knowledge base.
- **Adaptive Query Reformulation:** Automatically rephrase ambiguous queries.

21 How would you handle hallucinations in a retrieval-augmented system?

Techniques to Reduce Hallucinations:

- **Fact Verification Models:** Use post-processing models to fact-check generated responses.
- **Confidence Scoring:** Assign confidence scores to generated text based on retrieval strength.
- **Strict Context Binding:** Ensure the LLM does not generate responses beyond retrieved context.
- **Re-Retrieval Mechanism:** If confidence is low, trigger another retrieval iteration.

22 What are the main challenges in retrieval grounding for hallucination reduction?

Retrieval grounding plays a critical role in mitigating hallucinations in Retrieval-Augmented Generation (RAG) systems by ensuring that responses are anchored in factual, high-quality evidence. However, effective grounding faces several challenges related to retrieval quality, source reliability, and contextual relevance.

Key Challenges

- **Noisy or Conflicting Sources** - Retrieved documents may contradict each other, leading to inconsistencies in generated responses. - Misinformation and low-quality sources can introduce biases or reinforce incorrect claims. - Example: A medical RAG system retrieving both peer-reviewed studies and unverified blog posts on a health topic.
- **Context Dilution** - Large retrieval chunks may contain a mix of relevant and irrelevant information, making it harder for the model to focus on the most pertinent facts. - This issue is exacerbated when dense retrieval methods fetch long passages rather than targeted snippets. - Example: Retrieving a full research paper when only the conclusion section is relevant.
- **Sparse Domain-Specific Knowledge** - In highly specialized fields (e.g., law, finance, scientific research), the available retrieval corpus may be incomplete or outdated. - Models may struggle to find supporting evidence for niche queries, increasing the likelihood of hallucination. - Example: A legal AI assistant failing to retrieve recent case law updates due to an outdated database.

Strategies for Improving Retrieval Grounding

- **Knowledge Graph Integration** - Enhances retrieval accuracy by linking structured, verified information with retrieved unstructured text. - Example: Augmenting biomedical search results with ontology-based knowledge graphs like UMLS. - Reduces hallucination by reinforcing claims with well-defined entity relationships.
- **Dynamic Chunk Selection** - Instead of retrieving fixed-length chunks, rank and filter passages based on information density and semantic relevance. - Uses query-dependent scoring to emphasize sections containing the highest factual content. - Example: Extracting only regulatory clauses from a 100-page legal document rather than returning the entire text.
- **Multi-Hop Retrieval** - Retrieves supporting evidence iteratively by linking contextually relevant documents across multiple search steps. - Improves response consistency by constructing a chain of supporting facts rather than relying on a single document. - Example: Answering a question about climate policy by retrieving government reports first, then cross-referencing with scientific studies.
- **Adaptive Reranking Mechanisms** - Uses cross-encoder rerankers (e.g., MonoT5, ColBERT) to prioritize highly informative documents while demoting noisy sources. - Helps resolve conflicts in retrieval by ranking sources based on credibility scores.

- **Fact Verification Pipelines** - Implements post-retrieval validation using fact-checking models that compare generated responses against retrieved evidence. - Example: Applying Natural Language Inference (NLI) models to assess whether retrieved content supports or contradicts a claim.

23 How do you implement self-correcting mechanisms in a RAG-based chatbot?

Self-Correction Techniques:

- **User Feedback Loop:** Allow users to rate or correct model responses.
- **Iterative Prompting:** Re-run retrieval and generation if the initial output is flagged as unreliable.
- **Self-Consistency Decoding:** Generate multiple responses and select the most consistent one.
- **Contrastive Reranking:** Use a secondary model to compare multiple answers.

24 Explain context window spillage and how you mitigate it in RAG pipelines.

Definition: Context window spillage occurs when retrieved passages exceed the LLM's maximum token limit, leading to truncated inputs.

Mitigation Strategies:

- **Dynamic Chunking:** Select only the most relevant sections.
- **Summarization-Based Compression:** Condense retrieved documents before inputting to the LLM.
- **Hierarchical Retrieval:** Retrieve at different levels (document $\rightarrow$ section $\rightarrow$ sentence).

25 How do you decide the embedding model for RAG? What factors influence the choice?

Factors Influencing Embedding Model Selection:

- **Domain-Specific vs. General Models:** Use BioBERT for medical text, LegalBERT for legal queries.
- **Scalability Considerations:** Small models (MiniLM) for speed, large models (E5-large) for accuracy.
- **Multilingual and Cross-Lingual Retrieval:** If retrieval spans multiple languages, specialized multilingual models are essential:
  - **Multilingual Support:** mBERT, LaBSE, and XLM-R are effective for handling multilingual queries.
  - **Cross-Lingual Retrieval:** Models like LASER and MuRIL help retrieve semantically relevant information across languages.

26 Explain the role of knowledge distillation in improving retrieval models.

Purpose: Knowledge distillation transfers knowledge from a large, powerful model (teacher) to a smaller, efficient model (student).

Benefits in RAG:

- **Faster Retrieval:** Student models are lightweight, reducing inference time.
- **Lower Compute Cost:** Enables efficient deployment on edge devices.
- **Maintains Performance:** Retains most of the teacher model's retrieval accuracy.

27 How does contextual compression (e.g., Long-context RAG, memory-efficient retrieval) help in LLM inference?

Techniques for Contextual Compression:

- **Sentence Scoring:** Prioritizes high-information-density sentences, ensuring only the most relevant content is retained.
- **Extractive Summarization:** Eliminates redundancy by distilling key points from lengthy passages while maintaining factual integrity.
- **Adaptive Chunk Merging:** Dynamically adjusts the granularity of retrieved chunks, balancing recall and precision to improve retrieval effectiveness.

28 Explain the concept of "retrieval collapse" in dense retrieval systems.

Definition: Retrieval collapse occurs when a dense retrieval system repeatedly retrieves the same irrelevant documents, reducing search diversity.

Causes:

- **Overfitting to Training Queries:** The retrieval model learns narrow distribution patterns.
- **Embedding Degradation:** Poorly trained embeddings collapse into uniform clusters.

Mitigation Strategies:

- **Diversity-Enhancing Negative Sampling:** Train with varied negative samples.
- **Hybrid Sparse-Dense Models:** Introduce lexical-based re-ranking.
- **Re-ranking with Contrastive Learning:** Force dissimilar passages to be distinguishable.

29 What are the trade-offs between static and dynamic retrieval augmentation in generative models?

Feature	**Static Retrieval**	**Dynamic Retrieval**
Retrieval Type	Precomputed indexes	Real-time retrieval
Adaptability	Limited	High (query-dependent)
Compute Cost	Low	High (due to re-querying)
Use Case	FAQ-based, static knowledge	Open-domain, evolving topics

Table 3: Comparison of Static vs. Dynamic Retrieval