RAG AND LLM INTERVIEW QUESTIONS

Compiled Reference — Questions 1–45 (Pages 1–15)

Source: Inspireviraltimes.com

1. What is RAG? How does it work? Why is it better than a vanilla LLM?

Answer: RAG (Retrieval Augmented Generation) is a technique that combines external knowledge retrieval with an LLM to generate more accurate, up-to-date and contextual responses. Instead of relying only on the model's internal knowledge, it fetches relevant information from a knowledge base (e.g., vector DB, documents, web) and uses it as context for generation.

How it works (Step by Step)

- User Query → e.g., "What is the latest update on X?"
- Retrieve relevant documents from knowledge base.
- Augment the query with retrieved context.
- LLM generates the final answer using the context.
- Return the response to the user.

Comparison: RAG vs Vanilla LLM

Feature	Vanilla LLM	RAG
Knowledge source	Pre-trained (Static)	External + Internal (Dynamic)
Accuracy	May be outdated or hallucinate	More factual & up-to-date
Context length	Limited	Uses retrieved relevant context
Real-time info	Not possible	Possible (web/docs)
Customization	Hard	Easier (custom KB)
Cost	Lower (single model)	Higher (retrieval + generation)

> ***★ RAG = Retrieval + Augmentation + Generation***

Key Takeaways

- RAG reduces hallucinations.
- It works well for domain-specific questions.
- Needs a good retrieval system (e.g., embeddings).
- Use vector DB for semantic search.

---

2. What are Embeddings? How are they used in RAG?

Answer: Embeddings are dense vector representations of text (or other data) that capture semantic meaning. In RAG, embeddings are used to find semantically similar documents from a vector database.

Why is it important?

- Capture semantic meaning (not just keywords).
- Allow similarity search using vector math.
- Helps in finding relevant context for RAG.

Types of Embeddings

- Text Embeddings (e.g., OpenAI, BGE, Sentence-BERT)
- Image Embeddings (e.g., CLIP)
- Multimodal Embeddings (text + image, etc.)

Popular Embedding Models

Model	Provider	Dimensionality	Best For
text-embedding-3-small	OpenAI	1536	General purpose
text-embedding-3-large	OpenAI	3072	High accuracy
bge-small-en	BGE	384	Fast & efficient
bge-large-en	BGE	1024	Better quality
all-MiniLM-L6-v2	Sentence-BERT	384	Semantic search
CLIP	OpenAI	512	Multimodal (text + image)

Key Takeaways

- Higher dimensions = better accuracy (usually).
- Use cosine similarity for most cases.
- Embeddings can be fine-tuned for your domain.

---

3. What are the different types of RAG? Compare them.

Answer: There are mainly 3 types of RAG: 1. Naive RAG 2. Advanced RAG (e.g., Hybrid, Parent-Child, Graph RAG) 3. Self-RAG

Quick Comparison

Aspect	Naive RAG	Advanced RAG	Self-RAG
Retrieval	Single step	Multi-step / Hybrid	Iterative (LLM decides)
Context source	Fixed docs	Docs + Graph/Hierarchy	Dynamic
Accuracy	Good	Better	Best (adaptive)
Complexity	Low	Medium-High	High
Use case	Simple Q&A	Domain-specific/Complex	Long-form/reasoning

Types in Detail

- Naive RAG: Retrieve top-k relevant docs; pass them to LLM as context; simple but limited.

- Advanced RAG: Hybrid search (keyword + vector); Parent-Child (chunking strategy); Graph RAG (uses knowledge graph).
- Self-RAG: LLM decides what to retrieve; uses reflection/verification; iterative improvement.

★RAG is not just retrieval, it's a smarter way to generate!

Key Takeaways

- Choose type based on use case.
- Naive RAG = simple, fast.
- Advanced RAG = better accuracy.
- Self-RAG = most intelligent & adaptive.

---

4. How do you handle out-of-date information in RAG?

Answer: RAG uses external knowledge sources, which can become outdated. To handle this, we use strategies like time-aware retrieval, source prioritization, refreshing the index and hybrid search (combining fresh and static sources).

Steps to handle out-of-date information

- Use time-aware retrieval — retrieve recent docs (e.g., using timestamps or metadata).
- Prioritize trusted sources — give higher weight to reliable and frequently updated sources.
- Refresh the knowledge base — periodically re-index or update documents.
- Use hybrid search — combine vector search with keyword search for latest results.
- Add fallback — if info is old, show warning or ask user to verify.

Comparison: Different Approaches

Approach	How it handles outdated info	Pros	Cons
Time-aware retrieval	Uses timestamps / metadata	Simple, effective	May miss relevant old info
Source prioritization	Weights trusted sources	High accuracy	Needs manual setup
Index refresh	Re-indexes periodically	Always up-to-date	Expensive, compute heavy
Hybrid search	Combines vector + keyword search	Better recall	More complex to implement

★Keep metadata (date, version) with documents.

Key Takeaways

- Goal: Always provide the most recent and relevant information.

---

5. What is the difference between Vector Database and Traditional Database?

Answer: Traditional databases (like MySQL, PostgreSQL) store structured data and are optimized for exact matches and transactions. Vector databases store embeddings and are optimized for similarity search, which is useful for unstructured data like text, images, etc.

Key Differences

- Data type – Structured vs Unstructured (embeddings)
- Query type – Exact match vs Similarity search
- Use case – Transactions vs Semantic search
- Indexing – B-tree vs Vector index (e.g., HNSW, IVF)
- Output – Exact result vs Top-k similar results

Comparison Table

Feature	Traditional DB	Vector DB
Data type	Structured (tables)	Embeddings (vectors)
Storage model	Rows & columns	Vector index (HNSW, IVF)
Query type	SQL (exact match)	Similarity search (semantic)
Use case	Transactions, analytics	RAG, recommendations, semantic search
Indexing	B-tree, hash index	Vector index (HNSW, IVF)
Output	Exact results	Top-k similar results
Examples	MySQL, PostgreSQL, MongoDB	Pinecone, Weaviate, Faiss, Qdrant

> ★ ***Use traditional DB for structured data and vector DB for semantic search.***

---

6. How do you evaluate the performance of a RAG system?

Answer: RAG performance is evaluated using both retrieval and generation metrics. We check how well the retriever finds relevant documents and how accurate, helpful and grounded the final answer is.

Step-by-Step Evaluation

- Evaluate Retrieval — check relevance of retrieved docs.
- Evaluate Generation — check answer quality and faithfulness.
- Use automated + human evaluation.
- Analyze errors and improve (tune retriever, reranker, prompts).

Example Evaluation Flow

- Retrieve top-5 documents for a query.
- Check if relevant doc is in top-5 → Recall@5.
- Verify if docs are actually relevant → Precision@5.
- Use LLM-as-a-judge to check faithfulness and helpfulness.
- If low, improve retriever, reranker or prompt.

Key Metrics Explained

Metric	What it measures	Ideal value
Recall@k	Relevant docs found in top-k	↑ Higher
Precision@k	Relevant docs in top-k	↑ Higher
Faithfulness	Answer grounded in context	↑ Higher
Answer relevance	Matches user intent	↑ Higher
Helpfulness	Useful and complete answer	↑ Higher

> ***★ Good RAG = Relevant Docs + Accurate & Grounded Answer.***

---

7. What is chunking in RAG? What are the different chunking strategies?

Answer: Chunking is the process of splitting a large document into smaller, manageable pieces (chunks) so that the retriever can find relevant context more accurately. Good chunking improves retrieval quality, reduces noise and helps the LLM generate better responses.

Why is it important?

- Fits within embedding model's token limit.
- Improves semantic search accuracy.
- Reduces irrelevant context.
- Helps in better context utilization.

Chunking Strategies

Strategy	Description	Best For
Fixed Size	Split by fixed number of tokens/characters.	Simple use cases, structured data.
Recursive	Split by separators (e.g., \n, . ;).	General purpose, varied content.
Semantic	Use embeddings to split by meaning/similarity.	Long documents, conceptual text.
Sentence / Paragraph	Split by sentence or paragraph boundaries.	Well-structured text (articles, docs).
Token-based	Split by token count (using tokenizer).	LLM specific, accurate control.

> ***★ Proper chunking = Better retrieval = Better responses***

Key Takeaways

- Choose chunk size based on content type and model context window.
- Semantic chunking often gives better results for unstructured data.

8. What is Hybrid Search in RAG? How does it work and what are its advantages?

Answer: Hybrid search combines multiple search methods (e.g., keyword-based + vector-based) to retrieve more relevant and diverse results. It leverages the strengths of both approaches — exact matching and semantic understanding.

Why use Hybrid Search?

- Keyword search is precise but misses semantic matches.
- Vector search understands meaning but may miss exact terms.
- Hybrid search gives better recall, relevance and handles different query types (factual + semantic).

How Hybrid Search Works

- User Query → Generate Embedding (semantic)
- Vector Search (semantic match) + Keyword Search (exact match)
- Combine Results (merge + rerank)
- Top k Results

Comparison: Search Methods

Feature	Keyword Search	Vector Search	Hybrid Search
Matching	Exact words	Semantic similarity	Both
Recall	Medium	High	Highest
Precision	High	Medium	High
Use case	Factual queries	Semantic queries	General (robust)

> ***★ Combines the best of both worlds!***

Key Takeaways

- Hybrid search = better recall + precision.
- Use RRF or weighted scoring to combine results.
- Works well for diverse and real-world queries.

---

9. What is Reranking? Why is it used and what are popular reranking models?

Answer: Reranking is the process of reordering the retrieved documents based on a more sophisticated model (e.g., cross-encoder) to get the most relevant results at the top. It is used after initial retrieval to improve precision and relevance.

Why is it used?

- Initial retrieval (e.g., BM25, vector search) can be noisy.

- Reranking helps identify truly relevant documents.
- Improves precision and final answer quality.

Reranking Process

- Query → Initial Retrieval (top 100/1000)
- Reranker Model (e.g., Cross-encoder)
- Re-scored Results (top k)

Popular Reranking Models

Model	Type	Key Feature
BERT / RoBERTa	Cross-encoder	High accuracy, slower
Cohere Rerank	Cross-encoder	Optimized for search
ColBERT	Late interaction	Efficient, good balance
MonoT5	Cross-encoder	Generative, flexible
Jina Reranker	Cross-encoder	Fast + high quality

Comparison: Retriever vs Reranker

Aspect	Retriever (e.g., BM25, Vector)	Reranker (e.g., BERT, ColBERT)
Stage	1st (fetch candidates)	2nd (reorder)
Speed	Fast	Slower
Goal	High recall	High precision
Model type	Traditional / Embedding	Transformer-based
Output	Top 100-1000 docs	Top k (final results)

> ***★ Retrieval gives candidates, Reranking gives the best!***

Key Takeaways

- Reranking improves relevance and precision.
- Use cross-encoder models for best results.
- Combines well with hybrid search for maximum performance.

---

10. What are the different types of Vector Databases? Compare them.

Answer: Vector databases store embeddings and provide similarity search (like ANN) to find relevant data. They are optimized for high-dimensional vectors and are used in RAG, semantic search, recommendation systems, etc.

Why do we need them?

- Traditional DBs don't work well with high-dim data.
- Enable fast similarity search (approximate nearest neighbor - ANN).

- Scales for large datasets.

How a Vector DB Works (High Level)

- Documents → Create Embeddings (using LLM / Embedding Model)
- Store in Vector DB (with metadata)
- Query → Convert to Embedding
- Similarity Search (ANN) → Top-k Relevant Results

Types of Vector Databases

Type	Example	Key Features	Best For
Open Source (Self-hosted)	FAISS	High performance, flexible, GPU support	Custom solutions, large scale
Managed Cloud	Pinecone	Fully managed, scales automatically	Production apps, ease of use
Hybrid	Weaviate	Vector + keyword search, filters	RAG, complex search
Graph-based	Neo4j	Graph + vector similarity	Knowledge graphs, relationships
Multi-model	Qdrant	Vector, payload, filtering	Flexible data models

> ***★ Vector DBs are essential for RAG, semantic search and AI applications.***

Key Takeaways

- Fast similarity search.
- Scales to millions/billions of vectors.
- Different DBs for different needs.
- Essential for RAG.

---

11. What is a RAG Pipeline? Explain with a diagram.

Answer: RAG (Retrieval Augmented Generation) is a framework that combines external knowledge retrieval with an LLM to generate accurate, up-to-date and context-aware responses. It helps overcome the limitations of LLMs like outdated information and hallucinations.

Why use RAG?

- Access to real-time / updated information.
- Reduces hallucinations.
- Provides citations / source references.
- Improves accuracy and factuality.

RAG Pipeline (Step by Step)

- User Query

- Retrieve Relevant Documents (Vector DB / Search)
- Augment (Build Prompt with context)
- LLM Generation
- Final Response (with sources)

Types of RAG

Type	Description	Use Case
Naive RAG	Direct retrieval + generation	Simple use cases
Advanced RAG	Re-ranking, filters, etc.	Better accuracy & relevance
Agentic RAG	Uses agents for multi-step reasoning	Complex tasks, tool use
Self-RAG	Self-evaluation and reflection	Higher quality responses

> ***★ RAG = Retrieval (from external data) + Augmentation (add to prompt) + Generation (LLM).***

Key Takeaways

- Key Components: Retriever (Vector DB/Search), Augmenter (Prompt construction), Generator (LLM), Optional: Re-ranker, Guardrails.

---

12. What is the difference between Fine-tuning and RAG? When to use which?

Answer: Fine-tuning and RAG are two different approaches to improve LLMs. Fine-tuning updates the model's weights using task-specific data, while RAG keeps the model unchanged and retrieves relevant information from external sources at inference time.

Key Differences

- Fine-tuning = changes model weights.
- RAG = uses retrieval (no weight change).
- Fine-tuning = better for specialized tasks.
- RAG = better for dynamic / real-time info.

Comparison Table

Aspect	Fine-tuning	RAG
How it works	Updates model weights	Retrieves external info at runtime
Data needed	Labeled task-specific data	External knowledge base
Model change	Yes	No
Best for	Specialized, domain-specific tasks	Dynamic, up-to-date info, general use
Cost	High (compute + data)	Lower (uses existing model)
Latency	Lower (after training)	Higher (retrieval step)

Aspect	Fine-tuning	RAG
Accuracy	Better for specific tasks	Better for factual, real-time info

★ Both can be used together for best results (e.g. RAG + Fine-tuning).

Key Takeaways

- Use Fine-tuning when: task-specific data available, need high accuracy on a specific domain, data is stable/fixed, low latency required (after training).
- Use RAG when: need real-time or updated info, information changes frequently, want to reduce hallucinations, need citations or sources.
- Pro Tip: Use Fine-tuning for domain expertise and RAG for fresh, factual and verifiable information.

---

13. What is a Vector Database? How is it different from a traditional database?

Answer: A Vector Database is a specialized database that stores embeddings (vector representations) of data and allows similarity search, which is useful for AI applications like RAG, semantic search, and recommendation systems.

Why do we need it?

- Traditional DBs use exact match (e.g., SQL).
- Vector DBs use similarity search (e.g., cosine similarity).
- Helps in finding semantically similar data.
- Works well with unstructured data (text, images, etc.).

Key Components

- Embedding model – converts data to vectors.
- Vector index – for fast similarity search.
- Metadata – extra info (e.g., source, id).
- Similarity metric – cosine, dot product, etc.

Comparison: Traditional DB vs Vector DB

Feature	Traditional DB	Vector DB
Data type	Structured (tables)	Embeddings (vectors)
Query type	SQL (exact match)	Similarity search (cosine, dot product)
Indexing	B-tree, hash index	Vector index (HNSW, IVF, etc.)
Use case	Transactions, analytics	RAG, semantic search, recommendations
Output	Exact results	Top-k similar results
Examples	MySQL, PostgreSQL, MongoDB	Pinecone, Weaviate, Faiss, Chroma

> ★ ***Vector DB stores embeddings + metadata, and supports fast similarity search.***

---

14. What is Chunking in RAG? What are the different chunking strategies?

Answer: Chunking is the process of splitting a large document into smaller, manageable pieces (chunks) so that the retriever can find relevant context more accurately. Good chunking improves retrieval quality, reduces noise and helps the LLM generate better responses.

Why is it important?

- Fits within embedding model's token limit.
- Improves semantic search accuracy.
- Reduces irrelevant context.
- Helps in better context utilization.

Chunking Strategies

Strategy	Description	Best For
Fixed Size	Split by fixed number of tokens/characters.	Simple use cases, structured data.
Recursive	Split by separators (e.g., \n, . ;).	General purpose, varied content.
Semantic	Use embeddings to split by meaning/similarity.	Long documents, conceptual text.
Sentence/Paragraph	Split by sentence or paragraph boundaries.	Well-structured text (articles, docs).
Token-based	Split by token count (using tokenizer).	LLM specific, accurate control.

> ★ ***Proper chunking = Better retrieval = Better responses.***

Key Takeaways

- Choose strategy based on data type and use case.
- Keep chunks coherent and semantically complete.
- Test and evaluate (retrieval accuracy, latency, cost).
- Chunking is a crucial step in RAG pipelines.

---

15. What is Hybrid Search in RAG? How does it work and what are its advantages?

Answer: Hybrid search combines multiple search methods (e.g., keyword-based + vector-based) to retrieve more relevant and diverse results. It leverages the strengths of both approaches — exact matching and semantic understanding.

Why use Hybrid Search?

- Keyword search is precise but misses semantic matches.
- Vector search understands meaning but may miss exact terms.
- Hybrid search gives better recall, relevance and handles different query types (factual + semantic).

Techniques for Combining

- Weighted score (e.g., α × vector + β × keyword)
- Reciprocal Rank Fusion (RRF)
- Learning to Rank (LTR)
- Simple union/intersection + reranking

Comparison: Search Methods

Feature	Keyword Search	Vector Search	Hybrid Search
Matching	Exact words	Semantic similarity	Both
Recall	Medium	High	Highest
Precision	High	Medium	High
Use case	Factual queries	Semantic queries	General (robust)

> ***★ Combines the best of both worlds!***

Key Takeaways

- Hybrid search = better recall + precision.
- Use RRF or weighted scoring to combine results.
- Works well for diverse and real-world queries.

---

16. What are the different types of Retrieval methods used in RAG? Compare them.

Answer: Retrieval methods in RAG are used to fetch relevant information from external sources (e.g., vector DB, traditional DB, web) based on the user query.

Why are they important?

- Improve factual accuracy.
- Reduce hallucinations.
- Provide up-to-date information.
- Allow LLMs to access domain-specific data.

Retrieval Flow in RAG

- User Query
- Embed Query (create vector)
- Retrieve Relevant Documents (vector/keyword/hybrid)

- Rerank (optional)
- Top-k Relevant Docs
- Pass to LLM

Types of Retrieval Methods

Method	How it works	Best For	Pros	Cons
Keyword Search	Uses exact keyword matching (e.g., BM25)	Structured text, exact phrases	Simple, fast	Misses semantic meaning
Vector Search	Uses embeddings + similarity (e.g., cosine)	Unstructured text, semantic search	Understands context, higher accuracy	Computationally expensive
Hybrid Search	Combines keyword + vector search	Mixed data (structured + unstructured)	Better relevance, more robust	More complex to implement
Graph-based Search	Uses knowledge graphs + relations	Complex relationships (e.g., facts)	Handles multi-hop queries	Needs knowledge graph setup
Metadata Filtering	Filters using metadata (tags, categories, etc.)	Domain-specific search	More precise results	Requires proper metadata

> ★ ***Vector search is the most widely used in modern RAG systems.***

Key Takeaways

- Use hybrid search for best results.
- Choose method based on your data type.
- Combine with reranking for higher quality.

---

17. What is the difference between RAG and Fine-tuning? When should you use each?

Answer: RAG and Fine-tuning are two different approaches to improve LLMs. RAG retrieves relevant information at inference time, while fine-tuning updates the model weights using task-specific data.

Why it matters?

- RAG is faster, cheaper and more flexible.
- Fine-tuning gives better performance for specific tasks but is expensive and time-consuming.

Comparison Table

Aspect	RAG	Fine-tuning
How it works	Retrieves relevant docs at inference	Updates model weights

Aspect	RAG	Fine-tuning
Data needed	External knowledge base	Labeled task-specific data
Model change	No	Yes
Best for	Dynamic / changing information	Task-specific, fixed domain
Cost	Lower	Higher
Latency	Higher (retrieval step)	Lower (after training)
Accuracy	Better for factual and real-time info	Better for specific tasks

> ***★ RAG = knowledge at inference | Fine-tuning = knowledge in model***

Key Takeaways

- Use RAG when: need real-time or updated info, data is large and changes frequently, want to reduce hallucinations, domain-specific knowledge is needed.
- Use Fine-tuning when: you have high-quality labeled data, task is specific and fixed, need better reasoning or style, latency and cost are not major issues.
- Pro Tip: You can also combine both (RAG + Fine-tuning) for best results.

---

18. What are the challenges in implementing RAG? How can they be solved?

Answer: Implementing RAG comes with several challenges related to retrieval quality, latency, scalability, and data management. These challenges can affect the accuracy, relevance and overall performance of the system.

Why it's important?

- Helps build more reliable and accurate RAG systems.
- Improves user experience and reduces hallucinations.
- Essential for production-level deployment.

Challenges and Solutions

Challenge	Solution
1. Poor retrieval quality	Use better embeddings, hybrid search, reranking, and improve data processing.
2. Latency	Use efficient vector DB, caching, async retrieval, and top-k optimization.
3. Scalability	Choose scalable vector DB (e.g., Pinecone), sharding, and load balancing.
4. Outdated information	Use real-time data sources and periodic updates.
5. Metadata issues	Add proper metadata, filtering and structured data.
6. Hallucinations	Use reliable sources, reranking, faithfulness scores, and grounding.

Challenge	Solution
7. Data privacy & security	Implement access control, encryption, and private vector DBs.

★ Proper design and evaluation are key for successful RAG implementation.

Key Takeaways

- Common Pitfalls: using low-quality data, not using reranking, high latency due to large datasets, ignoring metadata and filtering, over-reliance on single retrieval method.
- Pro Tip: Always evaluate with real user queries and monitor performance (accuracy, latency, relevance).

---

19. What is embedding model? How does it work in RAG?

Answer: An embedding model converts text (or other data like images) into dense vector representations (embeddings) that capture semantic meaning. In RAG, embeddings are used to find relevant documents from a vector database using similarity search.

Why is it important?

- Captures semantic meaning (not just keywords).
- Helps in finding relevant context for the query.
- Improves retrieval accuracy and response quality.

How it works (Step by Step)

- Input text (query / document)
- Tokenization
- Embedding model (e.g., OpenAI, BGE, Sentence-BERT)
- Vector representation (e.g., 768/1024 dimensions)
- Store in Vector DB and use for similarity search

Comparison: Embedding Models

Model	Dimension	Strength
OpenAI (text-embedding-3)	1536	High quality, general use
BGE	1024	Good for multilingual
Sentence-BERT	768	Fast & effective
Cohere Embed	1024	Good for enterprise

★ Embedding model = Converts data → vector (dense representation)

Key Takeaways

- Embeddings capture semantic meaning.
- Similarity search (cosine, dot, etc.) finds relevant context.

- Better embeddings → better RAG results.

---

20. What is chunking in RAG? What are the different chunking strategies?

Answer: Chunking is the process of splitting a large document into smaller, manageable pieces (chunks) so that the retriever can find relevant context more accurately. It helps overcome the limitations of LLMs like context window size and improves retrieval quality.

Why is it important?

- Fits within model's context window.
- Improves retrieval accuracy.
- Reduces hallucinations.
- Better handling of long documents.

Chunking Strategies

Strategy	Description	Best For
Fixed size	Split by fixed number of tokens/characters.	Simple use cases, structured data.
Recursive	Split by separators (e.g., \n, . ;).	General purpose, varied content.
Semantic	Use embeddings to split by meaning/similarity.	Long documents, conceptual text.
Sentence/Paragraph	Split by sentence or paragraph boundaries.	Well-structured text (articles, docs).
Token-based	Split by token count (using tokenizer).	LLM specific, accurate control.
Multi-model	Consider text + images + tables, etc.	Documents with mixed data.

> ***★ Chunking = Breaks large data → smaller chunks (meaningful pieces).***

Key Takeaways

- Choose strategy based on data type & use case.
- Semantic gives better context.
- Combine strategies for best results.

---

21. What is hybrid search in RAG? What are its advantages and when should you use it?

Answer: Hybrid search combines multiple search methods (e.g., keyword-based + vector-based) to retrieve more relevant and diverse results. It leverages the strengths of both approaches — exact matching and semantic understanding.

Why is it important?

- Better accuracy and relevance.
- Handles both exact and semantic queries.
- Reduces missing relevant results.
- Works well for diverse data (text, tables, etc.).

Example Use Case (Query: "python machine learning tutorial")

- Keyword Results: Python tutorial, Machine learning course, ML tutorial (exact match)
- Vector Results: Deep learning guide, Python for data science, AI tutorial
- Hybrid Results (Reranked): 1. Python machine learning tutorial 2. Deep learning guide 3. ML tutorial 4. Python for data science

Comparison: Hybrid vs. Single Method

Aspect	Keyword Only	Vector Only	Hybrid
Accuracy	Medium	High	Highest
Recall	Medium	High	Highest
Handling synonyms	Low	High	High
Handling exact match	High	Low	High
Diversity	Low	High	High

> ***★ Hybrid search = Keyword + Vector → Better results.***

Key Takeaways

- When to use: high accuracy + good recall needed; complex queries or diverse data; when missing relevant results is costly.
- Pro Tip: Use weighted combination (e.g., 0.5 keyword + 0.5 vector) and reranking (e.g., cross-encoder) for best results.

---

22. What is a Vector Store? How does it work in RAG?

Answer: A vector store is a specialized database that stores document embeddings (vectors) and enables fast similarity search. In RAG, it stores embeddings of your knowledge base and helps retrieve relevant documents based on the user query.

Why is it important?

- Enables semantic search (not just keywords).
- Finds most relevant context quickly.
- Improves LLM response quality and accuracy.

How it works (Step by Step)

- Convert documents to embeddings (e.g., OpenAI, BGE).
- Store embeddings in a vector database (e.g., Pinecone, FAISS, Weaviate).

- Receive user query and convert to embedding.
- Find top-k similar vectors using similarity search.
- Retrieve relevant documents and pass to LLM.

Popular Vector Databases

DB	Features	Use Case
Pinecone	Fully managed, scalable, low latency	Production RAG systems
FAISS	Open source, fast, supports large datasets	Research, large scale search
Weaviate	Hybrid search (vector + keyword)	RAG, AI apps
Chroma	Simple, lightweight, Python friendly	Prototyping, small projects

> ***★ Vector store = Embeddings + Index + Similarity Search***

Key Takeaways

- Stores vector representations of data.
- Uses similarity search (cosine, dot, etc.).
- Fast and scalable for large datasets.
- Pro Tip: Choose a vector DB based on your scale, budget and integration needs.

---

23. What is Chunking in RAG? What are the different chunking strategies?

Answer: Chunking is the process of splitting a large document into smaller, manageable pieces (chunks) so that the retriever can find relevant context more accurately. It helps overcome the limitations of LLMs like context window size and improves retrieval quality.

Why is it important?

- Fits within model's context window.
- Improves retrieval accuracy.
- Reduces hallucinations.
- Better handling of long documents.

Comparison: Chunking Strategies

Strategy	Pros	Cons
Fixed Size	Simple, fast	Breaks context
Recursive	Good balance	May split inefficiently
Semantic	Best accuracy	Computationally expensive
Sentence/Paragraph	Natural flow	Sometimes too large
Token-based	Precise control	Needs tokenizer

Strategy	Pros	Cons
Multi-modal	Rich context	Complex to implement

> ***★ Chunking = Breaks large data → smaller chunks (meaningful pieces).***

Key Takeaways

- Choose strategy based on data type and use case.
- Semantic gives better context.
- Combine strategies for best results.

---

24. What is Reranking? Why is it used and what are popular reranking models?

Answer: Reranking is the process of reordering the retrieved documents based on a more sophisticated model (e.g., cross-encoder) to get the most relevant results at the top. It is used after initial retrieval to improve precision and relevance.

Why is it used?

- Better accuracy & relevance.
- Handles query-document interaction.
- Reduces noise from initial retrieval.
- Improves user experience.

How it works

- Query + Retrieved Docs (from vector search)
- Cross-encoder (consider query + doc together)
- Score each document (relevance score)
- Reorder documents (by score)
- Top-k most relevant docs

Popular Reranking Models

Model	Type	Key Feature
BERT / RoBERTa	Cross-encoder	High accuracy, slower
Cohere Rerank	Cross-encoder	Optimized for search
ColBERT	Late interaction	Efficient, good balance
MonoT5	Cross-encoder	Generative, flexible
Jina Reranker	Cross-encoder	Fast + high quality

Comparison: Retriever vs Reranker

Aspect	Retriever (e.g., BM25, Vector)	Reranker (e.g., BERT, CoBERT)
Stage	1st (fetch candidates)	2nd (reorder)
Speed	Fast	Slower
Goal	High recall	High precision
Model type	Traditional / Embedding	Transformer-based
Output	Top 100-1000 docs	Top k (final results)

> ***★ Reranking = Better order + Higher relevance (after initial retrieval).***

Key Takeaways

- Reranking improves relevance and precision.
- Use cross-encoder models for best results.
- Combines well with hybrid search for maximum performance.
- Pro Tip: Always evaluate with real user queries and monitor metrics (accuracy, latency, relevance).

---

25. What is a Vector Index? How is it used in RAG?

Answer: A vector index is a specialized data structure that stores vector embeddings and allows for fast similarity search. In RAG, it helps to quickly retrieve relevant documents from a large corpus based on the user query.

Why is it important?

- Enables fast and scalable similarity search.
- Improves retrieval accuracy.
- Handles large datasets efficiently.

How it works (Step by Step)

- Convert documents to embeddings (e.g., OpenAI, BGE).
- Build vector index (e.g., FAISS, Pinecone, Weaviate).
- Store vectors + metadata (document id, text, etc.).
- For a query, convert to embedding.
- Perform similarity search (top-k).
- Retrieve relevant documents and pass to LLM.

Types of Vector Indexes

Index Type	Example	Use Case
Flat (Exact)	FAISS (IndexFlat)	Small datasets, high accuracy
IVF (Inverted File)	FAISS (IndexIVF)	Large datasets, fast search
HNSW	FAISS (IndexHNSW)	High accuracy, fast search
PQ (Product Quantization)	FAISS (IndexPQ)	Very large data, low memory

Index Type	Example	Use Case
ScaNN	Google ScaNN	Large scale, cloud use

> ***★ Vector index is the backbone of RAG retrieval system.***

Key Takeaways

- Vector index = fast similarity search.
- Used in RAG to fetch relevant docs.
- Examples: FAISS, Pinecone, Weaviate, HNSW, IVF, PQ, ScaNN.
- Note: Vector indexes use approximate nearest neighbor (ANN) algorithms for speed and scalability.

---

26. What is Context Window? How does it affect LLM performance?

Answer: The context window is the maximum amount of text (tokens) an LLM can consider at once while generating a response. It includes the input prompt + retrieved context + generated output.

Why is it important?

- Determines how much context the model can use.
- Affects long-form reasoning and conversation.
- Larger context window = better understanding but higher cost and latency.

How it Affects LLM Performance

- More context → Better understanding → Higher quality responses.
- But: increases computation and memory usage; can lead to attention dilution (less focus); may increase latency and cost.

Comparison of Context Window Sizes

Model	Context Window	Key Advantage	Limitation
GPT-3.5	4K tokens	Fast & cheap	Short context
GPT-4	8K – 32K	Better reasoning	Higher cost
Claude 3	200K	Long context window	Higher latency
Gemini 1.5	1M	Very long context	Expensive
Llama 3	8K – 128K	Open source	Limited reasoning

> ***★ Context window is the memory size of the LLM for processing input + output.***

Key Takeaways

- Context window = max tokens model can see.
- Bigger window = better context handling.
- But higher cost, latency and resource usage.

- Larger context windows allow better retrieval and multi-document understanding in RAG.

---

27. What is Fine-tuning? How is it different from Prompt Engineering?

Answer: Fine-tuning is the process of training a pre-trained model on task-specific data to improve its performance for a specific domain or task. Prompt engineering, on the other hand, uses well-crafted prompts to guide the model's behavior without changing its weights.

Why is it important?

- Fine-tuning improves accuracy and consistency.
- Prompt engineering is faster and cheaper.
- Both are useful in different scenarios.

Example

- Prompt Engineering: "You are a helpful assistant. Answer the question in a concise way: What is photosynthesis?" → Model gives a good answer without training.
- Fine-tuning: Training data (e.g., Q: What is photosynthesis? A: Photosynthesis is the process...) → Model learns from data and improves over time.

Comparison Table

Aspect	**Fine-tuning**	**Prompt Engineering**
What it is	Trains model on task-specific data.	Uses better prompts to guide the model.
Changes weights?	Yes	No
Data needed	Labeled task data	No training data required
Cost	High	Low
Time	Hours – Days	Seconds – Minutes
Performance	Higher (for specific tasks)	Good (for general tasks)
Use cases	Domain adaptation, custom tasks	Quick Q&A, simple tasks, small changes

> ***★ Fine-tuning changes the model. Prompt engineering changes the input.***

Key Takeaways

- Fine-tuning = change weights.
- Prompt engineering = better input.
- Both improve LLM performance in different ways.
- Pro Tip: Use prompt engineering first. If not enough, then go for fine-tuning.

---

28. What is a Vector Database? How is it different from a traditional database?

Answer: A vector database is a specialized database that stores embeddings (vector representations) of data and allows similarity search, which is useful for AI applications like RAG, semantic search, and recommendation systems.

Why is it important?

- Enables semantic search (not just keywords).
- Finds most relevant context for the query.
- Improves retrieval accuracy and response quality.

Key Components

- Embedding model – converts data to vectors.
- Vector index – for fast similarity search.
- Metadata – extra info (e.g., source, id).
- Similarity metric – cosine, dot product, etc.

How is it different from a traditional database?

Feature	Traditional DB	Vector DB
Data type	Structured (tables)	Unstructured (vectors)
Query type	SQL (exact match)	Similarity search (cosine, dot product)
Indexing	B-tree, hash index	Vector index (HNSW, IVF, etc.)
Use case	Transactions, analytics	RAG, semantic search, recommendations
Output	Exact results	Top-k similar results
Examples	MySQL, PostgreSQL, MongoDB	Pinecone, Weaviate, Faiss, Chroma

> ***★ Vector DB stores embeddings + metadata, and supports fast similarity search.***

Key Takeaways

- Note: Vector DB is not a replacement, it complements traditional DBs.

---

29. What is Chunking in RAG? What are the different chunking strategies?

Answer: Chunking is the process of splitting a large document into smaller, manageable pieces (chunks) so that the retriever can find relevant context more accurately. Good chunking improves retrieval quality, reduces noise and helps the LLM generate better responses.

Why is it important?

- Fits within model's context window.
- Improves semantic search accuracy.
- Reduces irrelevant context.
- Helps in better context utilization.

Chunking Strategies

Strategy	Description	Best For
Fixed size	Split by fixed number of tokens/characters.	Simple use cases, structured data.
Recursive	Split by separators (e.g., \n, . ;).	General purpose, varied content.
Semantic	Use embeddings to split by meaning/similarity.	Long documents, conceptual text.
Sentence/Paragraph	Split by sentence or paragraph boundaries.	Well-structured text (articles, docs).
Token-based	Split by token count (using tokenizer).	LLM specific, accurate control.
Multi-modal	Consider text + images + tables, etc.	Documents with mixed data.

> ***★Chunking is a crucial step in RAG pipelines.***

Key Takeaways

- Choose strategy based on data type and use case.
- Keep chunks coherent and semantically complete.
- Test and evaluate (retrieval accuracy, latency, cost).

---

30. What is Hybrid Search in RAG? What are its advantages and when should you use it?

Answer: Hybrid search combines multiple search methods (e.g., keyword-based + vector-based) to retrieve more relevant and diverse results. It leverages the strengths of both approaches — exact matching and semantic understanding.

Why is it important?

- Better accuracy & relevance.
- Handles both exact and semantic queries.
- Reduces missing relevant results.
- Works well for diverse data (text, tables, etc.).

How it works

- Query → Keyword Search (BM25/TF-IDF) + Vector Search (Embeddings)
- Combine Results (merge + re-rank)
- Top k Results

RAG Architecture (with Hybrid Search)

- User Query → Retriever (BM25 + Vector) → Reranker (BERT) → LLM (grounded response)

Comparison: Retriever vs. Reranker

Aspect	Retriever (e.g., BM25, Vector)	Reranker (e.g., BERT, CoBERT)
Stage	1st (fetch candidates)	2nd (reorder)
Speed	Fast	Slower
Goal	High recall	High precision
Model type	Traditional / Embedding	Transformer-based
Output	Top 100-1000 docs	Top k (final results)

★ Hybrid search = Keyword + Vector → Better results.

Key Takeaways

- Common Pitfalls: using low-quality data, not using reranking, high latency due to large datasets, ignoring metadata and filtering, over-reliance on single retrieval method.
- Key Takeaways: Hybrid search = better recall + precision. Use RRF or weighted scoring to combine results. Works well for diverse and real-world queries.
- Pro Tip: Use weighted combination (e.g., 0.5 keyword + 0.5 vector) and reranking (e.g., cross-encoder) for best results.

---

31. What is a Retriever in RAG? What are the different types of retrievers?

Answer: A retriever in RAG is a component that fetches relevant information (documents or passages) from an external knowledge source (e.g., vector DB, search engine, or database) based on the user's query.

Why is it important?

- Brings relevant context to the LLM.
- Improves accuracy and reduces hallucinations.
- Helps the model work with up-to-date information.

Retriever Working Flow

- User Query
- Query Processing (tokenization, embedding, etc.)
- Retriever (vector/keyword/hybrid)
- Relevant Documents (top-k)
- Pass to LLM

Types of Retrievers

Type	How it works	Key Advantage	Use Case
1. Keyword Search	Uses exact keyword matching (e.g., BM25)	Simple, fast	Static docs, FAQs

Type	How it works	Key Advantage	Use Case
2. Vector Search	Uses embeddings + similarity (cosine, dot product)	Semantic search, finds related context	Docs, articles, chat history
3. Hybrid Search	Combines keyword + vector search	Better relevance + accuracy	Enterprise search
4. Graph-based Search	Uses knowledge graphs + relations	Context + relationships	Multi-hop QA
5. Metadata Filtering	Filters using metadata (tags, categories, etc.)	Domain-specific search	E-commerce, personalized search

> ***★Retriever is the bridge between the query and the external knowledge.***

Key Takeaways

- Choose retriever based on data type.
- Use hybrid for better results.
- Tune top-k value for quality vs. speed.
- Combine with reranking for accuracy.
- Popular retrievers: BM25 (keyword), FAISS (vector), Weaviate, Pinecone, ElasticSearch, GraphRAG.

---

32. What is Chunking in RAG? What are the different chunking strategies?

Answer: Chunking is the process of splitting a large document into smaller, manageable pieces (chunks) so that the retriever can find relevant context more accurately. It helps overcome the limitations of LLMs like context window size and improves retrieval quality.

Why is it important?

- Fits within model's context window.
- Improves retrieval accuracy.
- Reduces hallucinations.
- Better handling of long documents.

Chunking Strategies

Strategy	Description	Best For
Fixed size	Split by fixed number of tokens/characters.	Simple use cases, structured data.
Recursive	Split by separators (e.g., \n, . ;).	General purpose, varied content.
Semantic	Use embeddings to split by meaning/similarity.	Long documents, conceptual text.
Sentence/Paragraph	Split by sentence or paragraph boundaries.	Well-structured text (articles, docs).

Strategy	Description	Best For
Token-based	Split by token count (using tokenizer).	LLM specific, accurate control.
Multi-modal	Consider text + images + tables, etc.	Documents with mixed data.

Comparison: Chunking Strategies

Strategy	Pros	Cons
Fixed Size	Simple, fast	Breaks context
Recursive	Good balance	May split inefficiently
Semantic	Better accuracy	Computationally expensive
Sentence/Paragraph	Natural flow	Sometimes too large
Token-based	Precise control	Needs tokenizer
Multi-modal	Rich context	Complex to implement

> ***★ Chunking = Breaks large data → smaller chunks (meaningful pieces).***

Key Takeaways

- Choose strategy based on data type and use case.
- Semantic gives better context.
- Combine strategies for best results.

---

33. What is Reranking? Why is it used and what are popular reranking models?

Answer: Reranking is the process of reordering the retrieved documents based on a more sophisticated model (e.g., cross-encoder) to get the most relevant results at the top. It is used after initial retrieval to improve precision and relevance.

Why is it important?

- Better accuracy & relevance.
- Handles both exact and semantic queries.
- Reduces noise from initial retrieval.
- Improves user experience.

How it works?

- Query → Keyword Search (BM25/TF-IDF) + Vector Search (Embeddings)
- Combine / Rerank (e.g., weighted score)
- Final Results

Popular Reranking Models

Model	Type	Key Feature
BERT / RoBERTa	Cross-encoder	High accuracy, slower
Cohere Rerank	Cross-encoder	Optimized for search
ColBERT	Late interaction	Efficient, good balance
MonoT5	Cross-encoder	Generative, flexible
Jina Reranker	Cross-encoder	Fast + high quality

Comparison: Retriever vs. Reranker

Aspect	Retriever (e.g., BM25, Vector)	Reranker (e.g., BERT, CoBERT)
Stage	1st (fetch candidates)	2nd (reorder)
Speed	Fast	Slower
Goal	High recall	High precision
Model type	Traditional / Embedding	Transformer-based
Output	Top 100-1000 docs	Top k (final results)

★ Reranking = Better order + Higher relevance (after initial retrieval).

Key Takeaways

- Reranking improves relevance and precision.
- Use cross-encoder models for best results.
- Combines well with hybrid search for maximum performance.
- Pro Tip: Use weighted combination (e.g., 0.5 keyword + 0.5 vector) and reranking (e.g., cross-encoder) for best results.

---

34. What is LangChain? What are its main components and advantages?

Answer: LangChain is a framework for building LLM applications. It provides tools, components and abstractions to connect LLMs with external data sources, memory, chains and agents.

Why is it important?

- Simplifies LLM application development.
- Enables integration with external data.
- Provides reusable components and agents.
- Helps build complex workflows.

LangChain Architecture

- User Query → Prompt Template
- Memory ↔ LLM ↔ Tools (External APIs)

- LLM → Chains / Agents
- Chains/Agents → Response

Main Components of LangChain

Component	Description	Example
LLM	Interface to LLMs (e.g., OpenAI, HuggingFace)	ChatOpenAI() (LLM)
PromptTemplate	Formats input prompts with variables.	PromptTemplate("{query}")
Chains	Combines multiple steps (e.g., LLM + retriever).	SequentialChain()
Agents	Decides which tool to use based on the query.	initialize_agent()
Tools	External tools (search, calculator, APIs, etc.)	SerpAPI, PythonREPL
Memory	Maintains conversation history or context.	ConversationBufferMemory()

> ***★ LangChain = LLM + Tools + Memory + Chains + Agents***

Key Takeaways

- LangChain = Framework for LLM apps.
- Main components: LLM, Chains, Agents, Tools, Memory.
- Helps with RAG, multi-step reasoning, and tool use.
- Note: Each component can be used independently or together to build complex applications.

---

35. What is RAG (Retrieval Augmented Generation)? How does it work?

Answer: RAG (Retrieval Augmented Generation) is a technique that enhances LLM responses by retrieving relevant information from an external knowledge source (like a vector database) and using it as context for generation.

Why is it important?

- Reduces hallucinations.
- Provides up-to-date and domain-specific info.
- Improves accuracy and reliability.
- Allows LLMs to use private data.

How RAG works (Step by Step)

- User query (e.g., "What is LangChain?")
- Convert to embeddings (using embedding model)
- Search in vector database (find relevant chunks)
- Augment prompt (query + retrieved context)

- Generate response (using LLM)

RAG vs Fine-tuning

Aspect	RAG	Fine-tuning
Data	External data (at runtime)	Added to model (training)
Update	Easy (change DB)	Requires retraining
Accuracy	Good for factual info	Better for complex tasks
Cost	Lower	Higher
Latency	Higher (retrieval step)	Lower (after training)
Use case	Dynamic knowledge, real-time info	Fixed domain tasks

> ***★ RAG = Retrieve relevant info + Augment prompt + Generate response***

Key Takeaways

- RAG fetches relevant data at runtime.
- Combines retrieval + generation.
- Best for up-to-date and factual information.
- Note: Retrieval can be done using FAISS, Pinecone, Weaviate, etc.

---

36. What is a Vector Database? How is it different from a traditional database?

Answer: A vector database is a specialized database that stores vector embeddings (numerical representations of data) and allows fast similarity search. It is designed to work with AI/ML applications like RAG, semantic search and recommendation systems.

Why is it important?

- Enables semantic search (not just keywords).
- Finds most relevant context for the query.
- Improves retrieval accuracy and response quality.

Vector Index Types

- HNSW (Hierarchical Navigable Small World) – fast, high accuracy
- IVF (Inverted File Index) – good for large datasets
- PQ (Product Quantization) – memory efficient

How it works (Architecture)

- Documents (text, images, etc.) → Embedding Model (e.g., OpenAI, BGE)
- Vectors ([0.12, 0.34, ...]) → Vector Database (stores vectors + metadata)
- Similarity Search (Top-k) → Relevant Documents

Traditional DB vs Vector DB

Feature	Traditional DB	Vector DB
Data type	Structured (tables)	Unstructured (vectors)
Query type	SQL (exact match)	Similarity search (cosine, dot product)
Indexing	B-tree, hash index	Vector index (HNSW, IVF, etc.)
Use case	Transactions, analytics	RAG, semantic search, recommendations
Output	Exact results	Top-k similar results
Examples	MySQL, PostgreSQL, MongoDB	Pinecone, Weaviate, Faiss, Chroma

> ***★ Vector DB = Embeddings + Index + Similarity Search***

Key Takeaways

- Vector DB stores embeddings.
- Uses similarity search.
- Complements, not replaces, traditional databases.
- Pro Tip: Use weighted combination (e.g., 0.5 keyword + 0.5 vector) and reranking (e.g., cross-encoder) for better results.

---

37. What is a Prompt Template? How does it help in RAG?

Answer: A prompt template is a predefined structure or format for the user query that includes placeholders (e.g., {context}, {question}) and guides the LLM to generate better, more consistent and relevant responses.

Why is it important?

- Ensures consistent and structured prompts.
- Helps include retrieved context effectively.
- Improves response quality and accuracy.
- Reduces hallucinations.

How it works (Step by Step)

- Define a template with placeholders (e.g., {context}, {question}, {instructions}).
- Fill the placeholders with retrieved context and user query.
- Pass the final prompt to the LLM.
- LLM generates response based on the structured input.
- (Optional) Add system instructions for better control.

RAG Flow with Prompt Template

- User Query → Retrieve Relevant Context (from vector DB)
- Fill Prompt Template (with context + query)
- LLM → Final Response

★ Prompt Template = Structure + Context + Instructions

Key Takeaways

- Prompt templates = better structure.
- Works hand-in-hand with RAG.
- Improves relevance, accuracy and consistency.
- Note: Helps the model focus on relevant information and follow the desired format.

---

38. What is Multi-Modal RAG? How does it work?

Answer: Multi-Modal RAG is an extension of RAG that retrieves and uses information from multiple data types (text, images, audio, video, etc.) to generate more rich and accurate responses.

Why is it important?

- Many real-world problems are multi-modal.
- Provides richer context and better understanding.
- Improves accuracy and reduces hallucinations.
- Useful for complex queries (e.g., images + text).

How it works (Step by Step)

- Process each modality (text, image, etc.).
- Create embeddings for each modality (e.g., CLIP for images, text embeddings).
- Retrieve relevant data from multi-modal vector store.
- Combine retrieved information (e.g., text + image) into a unified context.
- Pass to LLM with a structured prompt.
- Generate final response (text/image/etc.).

Multi-Modal RAG Architecture

- Text/Image/Audio/Video → Text/Image/Audio/Video Encoder
- → Multi-Modal Embeddings → Vector Database (multi-modal)
- → Relevant Context (text + image + etc.) → LLM → Final Response

★ Multi-Modal RAG = Multiple Data Types + Retrieval + LLM

Key Takeaways

- Supports multiple data types.
- Uses multi-modal embeddings.
- Enables richer and more accurate responses.
- Uses multi-modal embeddings and cross-modal retrieval.

---

39. What is Agentic RAG? How is it different from traditional RAG?

Answer: Agentic RAG uses autonomous agents that can plan, reason, use tools and iteratively retrieve information to solve complex queries, while traditional RAG simply retrieves and uses relevant context once.

Why is it important?

- Handles complex, multi-step queries.
- Uses tools and external data sources.
- More flexible and adaptive.
- Improves accuracy and reliability.

How it works (Step by Step)

- Understand the query and break it into sub-tasks (planning).
- Retrieve relevant information (RAG).
- Use tools (e.g., web search, calculator, APIs) if needed.
- Reason and analyze the results.
- Repeat steps if more information is required (iterative).
- Generate final answer.

Agentic RAG Flow

- User Query → Agent (Planner) → Retrieve (RAG)
- → Use Tools (Web, API, Calculator, etc.) → Reason & Analyze
- → Generate Final Answer

Traditional RAG vs Agentic RAG

Aspect	Traditional RAG	Agentic RAG
Workflow	Single retrieval + generation	Plan → Retrieve → Reason → Use Tools → Generate
Decision Making	No	Yes (agent)
Tool Usage	Limited	Multiple (APIs, search, calculator, etc.)
Iteration	No	Yes
Complex Queries	Struggles	Handles well
Accuracy	Moderate	Higher
Example	FAQ, simple Q&A	Research, analysis, real-world tasks

> ***★Agentic RAG = RAG + Agent + Tools + Reasoning + Iteration***

Key Takeaways

- Agent = decision maker + planner.
- Uses tools and external sources.
- Better for complex, multi-step tasks.
- More accurate but higher cost/latency.
- Note: Agentic RAG is more powerful but also more complex and resource intensive.

40. What is Prompt Injection? How does it work and how can it be prevented?

Answer: Prompt injection is a security vulnerability where a malicious user manipulates the input (e.g., prompt, document, or data) to make the LLM ignore its original instructions and perform unintended actions.

Why is it important?

- Can lead to data leaks, harmful outputs, or unauthorized actions.
- Affects trust and safety of LLM applications.
- Common in RAG systems (injected via data or user input).

How it works? (Step by Step)

- Attacker crafts a malicious input (e.g., "Ignore previous instructions and ...").
- Input is passed to the LLM (directly or via retrieved context).
- LLM treats the injected text as valid instruction.
- LLM generates unintended output (e.g., reveals data, runs actions, etc.).

Common Types

- Direct injection (in user prompt).
- Indirect injection (in retrieved documents).
- Multi-turn injection (in conversation history).

Prevention Techniques

- Input sanitization & validation.
- Use system prompts & guardrails.
- Retrieve only trusted data.
- Apply output filtering.
- Use tools like ReAct / function calling with restrictions.

> ***★ Prompt injection = Manipulated input + LLM follows attacker's intent***

Key Takeaways

- Treat external data as untrusted.
- Use validation and safety checks.
- Combine RAG with guardrails.
- Monitor and log model behaviour.

---

41. What is Function Calling in LLMs? How does it work and what are its advantages?

Answer: Function calling allows an LLM to call external functions or tools (e.g., APIs, calculators, databases) by returning structured outputs (like JSON) instead of plain text.

Why is it important?

- Extends LLMs with real-world capabilities.
- Reduces hallucinations.
- Enables accurate and structured output.
- Useful for agentic systems and tool use.

How it works? (Step by Step)

- User asks a question or gives a task.
- LLM understands the need for a function (based on tool descriptions).
- LLM returns a structured function call (e.g., JSON with function name + params).
- Application executes the function (e.g., API call, calculation, DB query).
- Result is returned to LLM.
- LLM generates final response using the result.

Example: Weather API

- User: "What's the weather in Pune?"
- LLM (function call): { "name": "get_weather", "arguments": { "location": "Pune" } }
- Tool returns: { "temp": 28, "condition": "Sunny" }
- LLM (final): "The weather in Pune is 28°C and sunny."

Advantages

- Access to real-time information.
- Integration with external tools & APIs.
- Reduces hallucination.
- Enables complex, multi-step tasks.

> ***★ Function calling = LLM decides when and which tool to use.***

Key Takeaways

- Use structured output (JSON, schema).
- Validate tool inputs/outputs.
- Provide clear tool descriptions.
- Handle errors gracefully.

---

42. What is Agentic RAG? How is it different from traditional RAG?

Answer: Agentic RAG is an advanced form of RAG where the system uses an autonomous agent to plan, reason, and take actions (e.g., search, use tools, call APIs) to retrieve and use relevant information, instead of just doing a single retrieval step.

Why is it important?

- Handles complex, multi-step queries.
- Improves accuracy and relevance.
- Can use tools, APIs, and external data.
- Better for real-world, dynamic tasks.

Agentic RAG Architecture

- User Query → Agent (Planner) → Tools/APIs (Web, DB, Calculator, etc.)
- Agent ↔ LLM (Reasoning) ↔ Retriever (vector search)
- LLM → Final Answer

Traditional RAG vs Agentic RAG

Aspect	Traditional RAG	Agentic RAG
Workflow	Single retrieval + generate	Plan → Retrieve → Reason → Act → Generate
Tools usage	No	Yes (APIs, web, DB, etc.)
Reasoning	No / Limited	Multi-step reasoning
Handling complex queries	Weak	Strong
Accuracy	Moderate	Higher
Flexibility	Low	High
Use cases	Simple Q&A, search	Complex tasks, agents, real-world problems

> ***★Agentic RAG = RAG + Reasoning + Tools + Planning + Action***

Key Takeaways

- Agent decides what to do and when.
- Uses multiple tools and reasoning steps.
- Works well for complex and dynamic queries.
- Needs better orchestration and error handling.
- Pro Tip: Use structured planning, memory and tool-use frameworks (e.g., LangChain + Agents, AutoGen).
- Note: Agentic RAG is more powerful but also more complex and resource intensive.

---

43. What is Context Window in LLMs? Why is it important? How to handle long context?

Answer: The context window is the maximum amount of text (tokens) an LLM can process at once (including input + output).

Why is it important?

- Limits how much information the model can consider.
- Affects the quality of responses.
- Larger context = better understanding (but higher cost and latency).

How to handle long context?

- Chunking – Split long text into smaller chunks and retrieve relevant ones.

- Summarization – Compress long context into a shorter form.
- Sliding Window – Keep recent context and drop older parts.
- Hierarchical Retrieval – First retrieve relevant docs, then refine.
- RAG – Use external knowledge to reduce the context length.

How RAG Helps with Long Context?

- User Query → Retrieve relevant docs (from vector DB) → Augment context → LLM → Better response

Context Window Comparison

Model	Context Window	Notes
GPT-3.5	4K	Limited long context
GPT-4	8K / 32K	Better long context handling
Claude 3	200K	Very large context.
Gemini 1.5	1M	Supports huge context

> ***★ Context window = Max tokens (input + output) the model can handle.***

Key Takeaways

- Context window limits input size.
- RAG, summarization and chunking help handle long context.
- Larger context = better, but costlier.
- Note: These techniques help manage long context within the model's limit.

---

44. What is Model Quantization? How does it help? What are its types?

Answer: Model quantization is the process of reducing the precision of model weights and activations (e.g., from 32-bit float to 8-bit/4-bit integers) to make the model smaller, faster and more efficient.

Why is it important?

- Reduces model size and memory usage.
- Speeds up inference and training.
- Lowers computation and energy cost.
- Enables deployment on edge devices.

Quantization Workflow

- Pre-trained Model → Calibration / Fine-tuning
- → Quantization (e.g., INT8/INT4) → Optimized Model → Deploy

Example

- Original: 1.2 GB (FP32)
- Quantized: 300 MB (INT8) → 4x smaller!

Types of Quantization

Type	Description	Example / Use case
FP32 (No quant.)	32-bit floating point (highest precision)	Original model (e.g., training)
FP16	16-bit floating point	Faster, less memory (e.g., mixed precision)
INT8	8-bit integers	Good balance of size & accuracy
INT4	4-bit integers	Very small size, lower accuracy

> ***★ Quantization = Lower precision → Smaller model + Faster + Efficient***

Key Takeaways

- Quantization reduces precision (e.g., FP32 → INT8).
- Types: FP16, INT8, INT4.
- Helps in smaller size, faster inference, less cost.
- Slight accuracy loss (can be minimized with techniques like QAT).
- Note: Lower bit-width = Smaller model + Faster inference (with some accuracy trade-off).

---

45. What is In-Context Learning? How does it work? Give an example.

Answer: In-context learning (ICL) is the ability of an LLM to learn and perform a task by simply seeing examples in the prompt, without updating the model weights.

Why is it important?

- No training or fine-tuning required.
- Flexible and quick.
- Works well for few-shot and zero-shot learning.

How it works (Step by Step)

- Provide a task description / instruction.
- Give a few example input-output pairs (shots).
- The model observes the pattern.
- It generates the answer based on the examples.

Example (Sentiment Analysis)

- Prompt: Classify the sentiment (positive/negative).
- Examples: Q: "This movie is great!" → Positive; Q: "It was terrible." → Negative
- Now classify: Q: "It's an okay movie." → ?

ICL vs Fine-tuning vs Zero-shot

Feature	In-Context Learning	Fine-tuning	Zero-shot
Training	No	Yes	No
Examples	In prompt	In training data	None
Model update	No	Yes	No
Flexibility	High	Low	Medium
Performance	Good (few shots)	Best	Lower
Use case	Quick tasks	Production systems	Simple queries

★ICL = Learn from examples in the prompt (no weight update).

Key Takeaways

- ICL uses examples in the prompt.
- No model weight update.
- Works well for few-shot / zero-shot.
- Fine-tuning gives better performance but needs training.
- Real-world Use Case: ICL is used in chatbots, classification, translation, summarization, etc.
