🟢 1. Basics of RAG

- Problem with LLMs alone: They rely only on pretraining data → outdated, hallucinations, no domain grounding.
- Solution (RAG): Combine retrieval (fetching relevant documents) with generation (LLM answers).
- Core idea:
    1. Retrieve → Find relevant knowledge from external sources (vector DB, search engine).
    2. Augment → Inject retrieved context into the LLM prompt.
    3. Generate → LLM produces grounded, accurate output.

🔧 2. RAG Pipeline Components

(a) Document Ingestion

- Collect domain-specific data (PDFs, policies, FAQs, tickets).
- Preprocess: clean text, chunk into passages (e.g., 512–1024 tokens).
- Embed chunks into vectors using models like sentence-transformers, OpenAI embeddings, or Hugging Face.
- Store embeddings in a vector database (Pinecone, FAISS, Weaviate, Milvus).

(b) Retrieval

- Query → Convert user input into embedding.
- Search → Find nearest neighbors in vector DB (semantic similarity).
- Hybrid retrieval → Combine semantic + keyword search (BM25 + embeddings).
- Return top-k relevant chunks.

(c) Augmentation

- Build a context window: retrieved chunks + user query.
- Format into a structured prompt:
- Answer the question using only the context below:
- Context: [retrieved docs]
- Question: [user query]

(d) Generation

- LLM (GPT, LLaMA, Falcon, etc.) generates response using augmented context.
- Output is grounded in retrieved docs → reduces hallucination.

📊 3. Advanced RAG Techniques

🔍 Retrieval Enhancements

- Chunking strategies: Overlapping windows, semantic chunking (split by meaning not length).
- Metadata filtering: Restrict retrieval by tags (date, author, department).
- Re-ranking: Use cross-encoder models to refine top-k results.

🧠 Context Management

- Map-Reduce Summarization: Summarize large sets of retrieved docs before feeding to LLM.
- Context compression: Extract only the most relevant sentences.
- Long-context models: Use LLMs with 100k+ token windows (Claude, Gemini, GPT-4 Turbo).

🏢 Enterprise Features

- Policy grounding: Ensure answers cite compliance documents.
- Audit trails: Log retrieved docs + generated answers for transparency.
- Multi-source retrieval: Combine vector DB + SQL + APIs.
- Access control: Restrict retrieval based on user roles.

⚡ Performance Optimizations

- Caching: Store embeddings for frequent queries.
- Approximate nearest neighbor (ANN): Speed up retrieval in large DBs.
- Streaming generation: Show partial answers while LLM continues.

🚀 4. RAG Variants

- Vanilla RAG: Simple retrieval + generation.
- Iterative RAG: Multi-step retrieval → refine query after initial answer.
- Agentic RAG: LLM acts as an agent, deciding when/how to retrieve.
- Multi-modal RAG: Retrieve across text, images, audio, video.
- Knowledge Graph + RAG: Combine structured graph queries with semantic retrieval.

🏢 5. Enterprise AI Context (Your Prep)

For Salesforce Agentforce / enterprise copilots, the RAG pipeline typically looks like this:

1. Data ingestion: Policies, CRM records, tickets → chunk + embed.
2. Vector DB: Pinecone/Weaviate with UUIDs for version control.
3. Retrieval: Hybrid search (semantic + keyword).
4. Augmentation: Context window with citations.
5. Generation: LLM answers with compliance-safe formatting.
6. Guardrails: Filters for hallucination, bias, unsafe content.

7. Evaluation: Metrics like grounded accuracy, citation coverage, latency.

🧩 Visual Flow (Scratch → Advanced)

User Query

↓

Embed Query → Vector DB (semantic search)

↓

Retrieve Top-k Chunks (with metadata filters)

↓

Augment Prompt (context + query)

↓

LLM Generation (grounded answer)

↓

Post-processing (citations, safety filters, formatting)

↓

Final Answer

✅ In short: RAG is the bridge between LLM intelligence and real-world knowledge.

- Scratch → simple retrieval + generation.
- Advanced → hybrid search, compression, multi-source, compliance guardrails.