RETRIEVAL-AUGMENTED GENERATION

# RAG Systems & Architecture

A builder's primer on how retrieval grounds language models — the pipeline, the components, the design decisions, and a complete curated study library.

RAG · VECTOR SEARCH · EMBEDDINGS · LLM GROUNDING · RERANKING · AGENTIC RAG · ENTERPRISE ARCHITECTURE · EVALUATION

ABSTRACT

Retrieval-Augmented Generation couples a generative language model with an external, searchable knowledge store so that answers are grounded in retrieved evidence rather than parametric memory alone. This primer walks the full RAG pipeline — ingestion, chunking, embedding, indexing, retrieval, reranking, and grounded generation — explains the core architectural patterns (naive, advanced, modular, and agentic RAG), sets out a layered enterprise reference architecture with worked cloud stacks (AWS, Azure, open-source) and domain blueprints, surveys the design trade-offs that decide production quality, and closes with a structured, link-rich study library spanning foundational papers, courses, frameworks, evaluation tooling, and enterprise architecture references.



## 01 Why RAG exists

Large pre-trained language models store a great deal of factual knowledge in their weights, but that knowledge is frozen at training time, opaque about its sources, and prone to confident fabrication. RAG was introduced by Lewis et al. (2020) to address exactly this gap: combine the model's **parametric memory** (what it learned during training) with a **non-parametric memory** (an external index of documents queried at inference time).

The result is a system that can cite its sources, stay current as the knowledge store is updated, and answer domain-specific questions without retraining the underlying model. These three properties — provenance, freshness, and adaptability without retraining — are why RAG became the default architecture for grounded question-answering, enterprise assistants, and document chat.

PROBLEM

**Stale knowledge**

Weights freeze at training cutoff; the world keeps moving.

PROBLEM

**No provenance**

Models can't point to where an answer came from.

PROBLEM

**Hallucination**

Fluent but unfounded output on knowledge-intensive tasks.

RAG FIX

**Ground + cite**

Retrieve evidence first, then generate conditioned on it.

## 02 The RAG pipeline, end to end

Every RAG system, however sophisticated, is built from two phases: an offline **indexing** phase that prepares the knowledge store, and an online **query** phase that runs at request time. Understanding the seven stages below is the core mental model.

### Indexing phase (offline)

| STAGE | WHAT HAPPENS | KEY DECISIONS |
|---|---|---|
| **1. Ingestion** | Load source documents — PDFs, HTML, databases, wikis — and extract clean text plus metadata. | Parsers, OCR for scans, metadata schema (source, date, author, section). |
| **2. Chunking** | Split documents into retrievable units small enough to embed and fit the context window. | Chunk size, overlap, fixed vs. semantic vs. recursive splitting. |
| **3. Embedding** | Convert each chunk into a dense vector using an embedding model. | Embedding model choice, dimensionality, domain fit, cost. |
| **4. Indexing** | Store vectors (and often the original text) in a vector database with an ANN index. | Vector DB, index type (HNSW, IVF), hybrid sparse+dense. |

### Query phase (online)

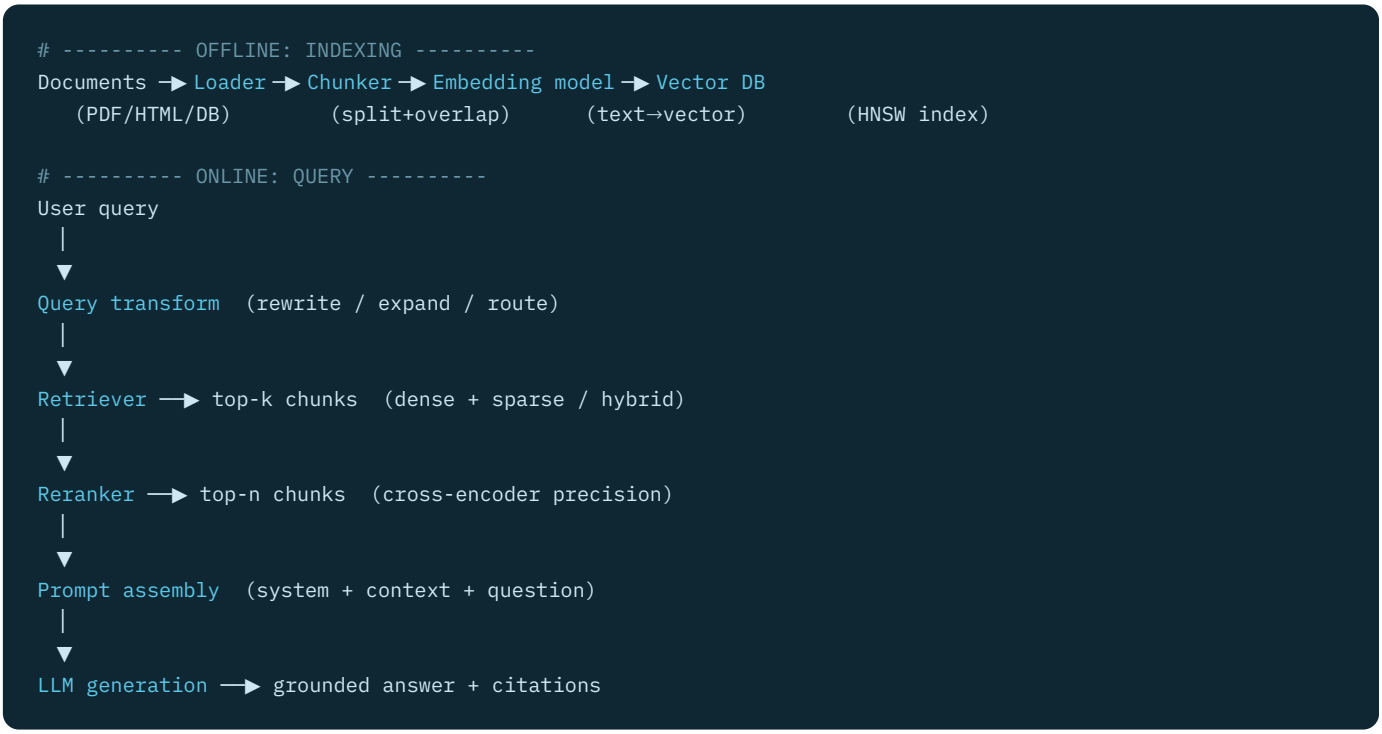
| STAGE | WHAT HAPPENS | KEY DECISIONS |
|---|---|---|
| **5. Retrieval** | Embed the user query and fetch the top-k most similar chunks from the index. | k value, similarity metric, query transformation, hybrid search. |
| **6. Reranking** | Re-score the retrieved candidates with a more precise cross-encoder and keep the best. | Reranker model, candidate pool size, latency budget. |
| **7. Generation** | Inject the top chunks into the prompt as context; the LLM answers grounded in them, ideally with citations. | Prompt template, context ordering, citation format, refusal on weak evidence. |

DESIGN NOTE

Retrieval quality caps generation quality. An LLM cannot answer from evidence it never received — so most production effort goes into stages 2–6, not the prompt. "Garbage retrieved in, garbage generated out."

## 03 Reference architecture

A minimal but production-shaped data flow. The dashed boundary separates the one-time offline build from the per-request online path.




## 04 Architectural patterns

RAG implementations fall on a maturity spectrum. Each tier adds components that fix specific failure modes of the tier below.

| PATTERN | SHAPE | SOLVES |
|---|---|---|
| **Naive RAG** | Embed → retrieve top-k → stuff into prompt → generate. The "hello world" of RAG. | Baseline grounding; fast to build. |
| **Advanced RAG** | Adds pre-retrieval (query rewriting, expansion) and post-retrieval (reranking, compression) steps. | Low precision, irrelevant context, lost-in-the-middle. |
| **Modular RAG** | Swappable modules — routing, hybrid search, iterative retrieval, memory — composed into a graph. | Diverse query types, complex pipelines, reuse. |
| **Agentic RAG** | An LLM agent decides when to retrieve, what to query, and whether to retrieve again, looping until satisfied. | Multi-hop reasoning, tool use, self-correction. |
| **GraphRAG** | Builds a knowledge graph from the corpus; retrieval traverses entities and relationships, not just chunks. | Global "connect-the-dots" questions across a whole corpus. |

## 05 Design decisions that decide quality

### Chunking strategy

Too small and chunks lose context; too large and they dilute relevance and waste the context window. Start with recursive splitting around 256–512 tokens with ~10–20% overlap, then tune. Semantic chunking (splitting on meaning boundaries) often beats fixed-size for heterogeneous documents.

### Embedding model

The embedding model defines what "similar" means for your retriever. Match it to your domain and language, check it on a benchmark like MTEB, and weigh dimensionality and cost against retrieval gains. The query and document embeddings must come from the same model family.

### Hybrid search & reranking

Dense vector search captures semantic similarity but misses exact terms (names, codes, IDs). Combining it with sparse keyword search (BM25) and fusing the results — then reranking the union with a cross-encoder — is the single most reliable quality lever in production RAG.

### Grounding & refusal

Instruct the model to answer only from the supplied context and to say it doesn't know when evidence is weak. Pair this with citation rendering so users can verify claims against sources — the original promise of RAG.

**BUILDER CHECKLIST**

Evaluate retrieval (recall@k, MRR) *before* generation · use hybrid search + a reranker · cite sources in output · cap context to the strongest n chunks · add a "no answer" path · measure faithfulness and answer relevance continuously.

## 06 Evaluating a RAG system

RAG failures hide in two places — retrieval and generation — so evaluation must cover both, separately.

| METRIC | MEASURES | QUESTION IT ANSWERS |
|---|---|---|
| **Context recall / recall@k** | Retrieval | Did we fetch the chunks that contain the answer? |
| **Context precision / MRR** | Retrieval | Are the relevant chunks ranked near the top? |
| **Faithfulness** | Generation | Is the answer supported by the retrieved context, or hallucinated? |
| **Answer relevance** | Generation | Does the answer actually address the user's question? |

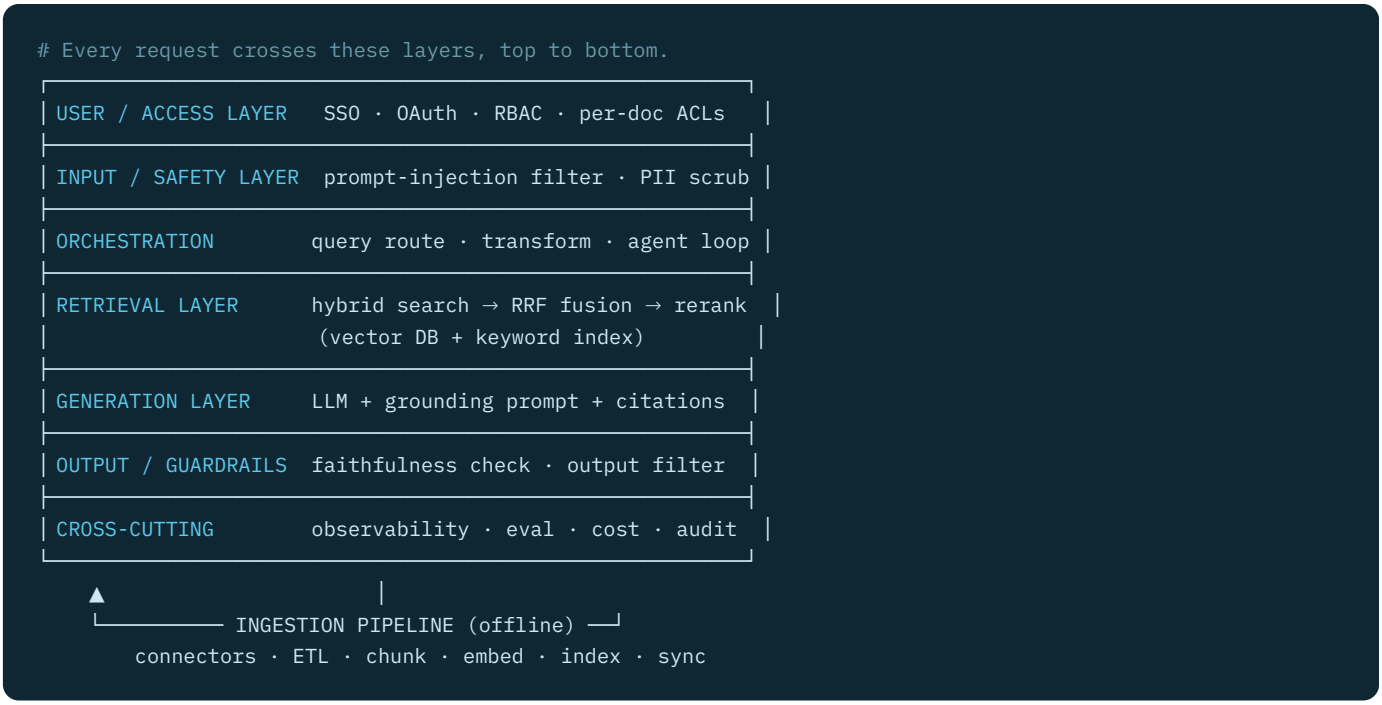
Frameworks like RAGAS, TruLens, and DeepEval automate these metrics; the library below links each one.

## 07 Enterprise reference architecture

Moving RAG from a notebook to production means wrapping the seven-stage pipeline in the layers an enterprise actually requires — access control, governance, observability, and cost management. Naive RAG

succeeds only 10–40% of the time in enterprise settings, which is why production systems are layered, not linear.

### The layered enterprise stack




**WHY THE LAYERS MATTER**

The retrieval and generation core is identical to Section 03. What makes it *enterprise* is everything around it: identity-aware retrieval (a user must only retrieve chunks they're authorized to see), input/output guardrails, and the cross-cutting observability and audit trail that compliance demands. 73% of enterprises cite data security as the primary barrier to AI adoption — so security is a prerequisite, not a feature.

### Production design imperatives

| CONCERN | ENTERPRISE PATTERN |
|---|---|
| **Access control** | Metadata-filtered retrieval enforcing per-document ACLs; the retriever never returns chunks the caller can't see (identity-aware RAG). |
| **Retrieval quality** | Hybrid search (dense + BM25/SPLADE) fused with Reciprocal Rank Fusion, then cross-encoder or ColBERT reranking — now the production baseline. |
| **Governance** | Data lineage, source freshness/sync, retention policy, and a context/governance layer upstream of the index. |
| **Observability** | Per-stage tracing (retrieval hits, latency, token cost), online eval, and human feedback capture. |
| **Multi-tenancy** | Namespace or per-tenant index isolation so one customer's data never leaks into another's retrieval. |
| **Cost control** | |

Serverless where possible, caching, embedding reuse, and right-sizing k and context length per query class.

## 08 Reference stacks & worked examples

The same architecture, realized on three common stacks. Choose by where your data and compliance posture already live.

### Cloud reference stacks

| LAYER | AWS | AZURE | OPEN / PORTABLE |
|---|---|---|---|
| **Ingestion** | Bedrock Knowledge Bases, S3, Glue | AI Search indexers, Blob, Doc Intelligence | Unstructured.io, Airbyte |
| **Embeddings** | Bedrock (Titan / Cohere) | Azure OpenAI embeddings | BGE, E5, Nomic (self-host) |
| **Vector store** | OpenSearch, Aurora pgvector, Neptune (GraphRAG) | Azure AI Search (hybrid) | Qdrant, Weaviate, Milvus, pgvector |
| **Orchestration** | Bedrock Agents | Azure AI Foundry / Prompt Flow | LangChain, LlamaIndex, Haystack |
| **Generation** | Bedrock (Claude, etc.) | Azure OpenAI / model catalog | vLLM, Ollama (self-host) |
| **Eval & obs** | Bedrock Eval, CloudWatch | AI Foundry evaluations | RAGAS, TruLens, Langfuse |

### Worked example — internal knowledge assistant

An employee asks an internal assistant a policy question. SSO identifies the user; the orchestrator rewrites the query and routes it; hybrid retrieval pulls candidate chunks *filtered to the documents that user may access*; a reranker keeps the top 4; the LLM answers grounded in them with clickable citations; a faithfulness check runs before the answer is returned and the whole trace is logged for audit. This enterprise-search pattern is the largest RAG application segment, turning keyword search into cited, synthesized answers across the org's knowledge base.

### Domain blueprints

| DOMAIN | RAG APPLICATION | ARCHITECTURAL EMPHASIS |
|---|---|---|
| **Financial services** | Compliance Q&A over regulations, policies, and market feeds with full source citation. | Freshness, provenance, audit trail. Largest RAG segment by end user. |
| **Healthcare** | Clinical decision support grounded in peer-reviewed literature and institutional protocols. | HIPAA posture, faithfulness, refusal on weak evidence. |
| **Legal** | Natural-language search across case law and contracts, cited to exact clauses. | Precision retrieval, clause-level citation, multi-hop. |
| **Customer support** | Grounded answers from product docs and ticket history. | Latency, caching, deflection metrics. |

ENTERPRISE READINESS CHECKLIST

Identity-aware retrieval (per-doc ACLs) · hybrid search + RRF + reranking · prompt-injection & PII guardrails on input and output · per-stage observability and online eval · tenant isolation · data lineage & retention policy · cost monitoring · a documented "no answer" path.

## 09 Study library

A curated, sequenced path from first principles to production. Work top to bottom within each section; sections A–G move from theory to tooling to enterprise architecture.

### A Foundational papers

| RESOURCE | SOURCE | WHY IT MATTERS |
|---|---|---|
| Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks (Lewis et al., 2020) | arxiv.org/abs/2005.11401 | The original RAG paper. Read this first. |
| Dense Passage Retrieval (Karpukhin et al., 2020) | arxiv.org/abs/2004.04906 | How dense retrievers beat BM25 — the retrieval backbone. |
| REALM: Retrieval-Augmented LM Pre-Training (Guu et al., 2020) | arxiv.org/abs/2002.08909 | Retrieval baked into pre-training; conceptual sibling of RAG. |
| Retrieval-Augmented Generation for LLMs: A Survey (Gao et al., 2023) | arxiv.org/abs/2312.10997 | The definitive survey — naive vs. advanced vs. modular RAG. |
| From Local to Global: GraphRAG (Microsoft, 2024) | arxiv.org/abs/2404.16130 | Graph-structured retrieval for corpus-wide questions. |

### B Courses & structured learning

| RESOURCE | SOURCE | FORMAT |
|---|---|---|
| DeepLearning.AI — Building & Evaluating Advanced RAG | deeplearning.ai/short-courses | Short course; sentence-window & auto-merging retrieval. |
| DeepLearning.AI — Preprocessing Unstructured Data for RAG | deeplearning.ai/short-courses | Ingestion and chunking deep-dive. |
| Hugging Face — RAG & Advanced RAG cookbook | huggingface.co/learn | Hands-on notebooks, open models. |
| LangChain RAG tutorials | python.langchain.com/docs/tutorials | Build a working pipeline step by step. |
| LlamaIndex RAG learning guide | docs.llamaindex.ai | Indexing-first framework with strong RAG primitives. |

### C Frameworks & orchestration

| RESOURCE | SOURCE | ROLE |
|---|---|---|

| | | |
|---|---|---|
| LangChain | python.langchain.com | General LLM orchestration; widest integrations. |
| LlamaIndex | docs.llamaindex.ai | Data framework purpose-built for RAG. |
| Haystack (deepset) | haystack.deepset.ai | Production-oriented pipelines and components. |
| DSPy | dspy.ai | Programmatic prompt & pipeline optimization. |

## D Vector databases & retrieval

| RESOURCE | SOURCE | NOTES |
|---|---|---|
| pgvector | github.com/pgvector/pgvector | Vectors inside Postgres — pragmatic default. |
| Qdrant | qdrant.tech | Open-source, fast, strong hybrid search. |
| Weaviate | weaviate.io | Built-in modules and hybrid retrieval. |
| FAISS (Meta) | github.com/facebookresearch/faiss | The ANN library under many systems. |
| MTEB embedding leaderboard | huggingface.co/spaces/mteb/leaderboard | Compare embedding models before you commit. |

## E Evaluation tooling

| RESOURCE | SOURCE | MEASURES |
|---|---|---|
| RAGAS | docs.ragas.io | Faithfulness, context precision/recall, answer relevance. |
| TruLens | trulens.org | RAG triad: context relevance, groundedness, answer relevance. |
| DeepEval | github.com/confident-ai/deepeval | Unit-test-style LLM & RAG evaluation. |

## F Engineering reading

| RESOURCE | SOURCE | NOTES |
|---|---|---|
| Anthropic — Contextual Retrieval | anthropic.com/news/contextual-retrieval | Reduces failed retrievals by prepending chunk context. |
| Pinecone Learning Center | pinecone.io/learn | Clear explainers on embeddings & retrieval. |
| Wikipedia — Retrieval-augmented generation | en.wikipedia.org/wiki/Retrieval-augmented_generation | Solid, well-sourced overview for orientation. |

## G Enterprise architecture references

| RESOURCE | SOURCE | WHY IT MATTERS |
|---|---|---|

| | | |
|---|---|---|
| AWS — RAG on Amazon Bedrock Knowledge Bases | docs.aws.amazon.com/bedrock | Managed end-to-end RAG reference on AWS. |
| Azure — RAG solution architecture (Architecture Center) | learn.microsoft.com/azure/architecture | Production blueprint with security & governance. |
| Azure AI Search — Retrieval-augmented generation overview | learn.microsoft.com/azure/search | Hybrid search + grounding building blocks. |
| GCP — Vertex AI RAG Engine | cloud.google.com/vertex-ai | Managed RAG and grounding on Google Cloud. |
| Applied AI — Enterprise RAG Architecture: A Practitioner's Guide | applied-ai.com/briefings/enterprise-rag-architecture | Production failure modes and the patterns that fix them. |
| Databricks — Generative AI / RAG reference patterns | databricks.com/glossary/retrieval-augmented-generation | Data-platform-centric enterprise patterns. |

KEEP BUILDING WITH INTELLIFORGE

## From reading about RAG to shipping it

This primer is part of the IntelliForge learning ecosystem. Start free, go deep, then build production agents with us.

FREE · 12-WEEK BOOTCAMP

### IntelliForge Learning

Hands-on AI engineering from fundamentals to RAG & agents.

learning.intelliforge.tech

▲ Intensive track · upskill.intelliforge.tech

OPEN-SOURCE TOOL

### Chronicle

Research copilot — synthesize any source in this library.

deep-research.intelliforge.tech

PAID INTENSIVE

### Upskill Programme

Build & deploy real AI products with mentorship.

upskill.intelliforge.tech

**IntelliForge Research** · IF-RES-2026-010 · "RAG Systems & Architecture" · Version 1.0, June 2026
Published by **Intelliforge Digital Services**, Kondapur, Hyderabad · PAN ADQPH9644E