RAG Reference Topics

Table of Contents

GitHub Repo- Advanced RAG

Part 1: Core RAG

1. Offline / Batch Document Processing & Ingestion

Document ingestion is the foundational step where raw documents are collected, normalized, and prepared for the RAG pipeline. This typically runs as a batch process during off-peak hours.

Key processes include:

Document collection from various sources such as file systems, APIs, databases, and web crawlers. Format detection and conversion handles the diversity of input types including PDFs, Word documents, HTML, markdown, and plain text. Text extraction pulls content from complex formats while preserving structure where possible. Cleaning and normalization removes noise like headers, footers, boilerplate, and encoding issues. Language detection identifies the document language for appropriate downstream processing. Deduplication identifies and handles duplicate or near-duplicate documents to avoid redundant storage and retrieval.

Implementation considerations:

Pipeline orchestration tools like Apache Airflow help manage complex ingestion workflows. Parallel processing accelerates throughput for large document collections. Error handling and retry logic ensure robustness against transient failures. Versioning tracks document changes over time, enabling incremental updates rather than full reprocessing.

---

2. Document Chunking & Segmentation

Chunking divides documents into smaller, semantically coherent units suitable for embedding and retrieval. The chunking strategy significantly impacts retrieval quality.

Common chunking strategies:

Fixed-size chunking splits text into chunks of a predetermined token or character count. This is simple but may break mid-sentence or mid-thought. Sentence-based chunking uses sentence boundaries, preserving grammatical integrity but potentially creating uneven chunk sizes. Paragraph-based chunking respects paragraph boundaries, often aligning with topical shifts. Semantic chunking uses embedding similarity to identify natural breakpoints where the topic shifts. Hierarchical chunking creates multiple granularity levels, enabling both broad and narrow retrieval. Document-structure-aware chunking leverages headings, sections, and other structural cues present in the original document.

Overlap and context windows:

Adding overlap between chunks (typically 10-20% of chunk size) ensures that information spanning chunk boundaries remains retrievable. Some systems also store parent-child relationships between chunks to enable context expansion after retrieval.

Optimal chunk size depends on the embedding model's context window, the nature of queries (specific vs. broad), and the density of information in source documents. Typical sizes range from 256 to 1024 tokens.

---

3. Metadata & Index Design

Metadata enriches chunks with additional context that can improve retrieval precision and enable filtering.

Common metadata fields include:

Source information such as document title, author, publication date, URL, and file path. Structural metadata like section headings, page numbers, and document hierarchy position. Semantic metadata including detected entities, topics, document type, and summary. Operational metadata covering ingestion timestamp, version, access permissions, and confidence scores.

Index design considerations:

Flat indexes store all chunks in a single index, simple but potentially slow for large collections. Hierarchical indexes organize chunks by document, section, or topic for more targeted retrieval. Sharded indexes distribute data across multiple partitions for scalability. Filtered indexes support metadata-based pre-filtering to narrow the search space before semantic matching.

Effective schema design anticipates query patterns. If users frequently filter by date or document type, those fields should be indexed for efficient filtering.

---

4. Embeddings & Vector Search

Embeddings transform text into dense numerical vectors that capture semantic meaning, enabling similarity-based retrieval.

Embedding model selection factors:

Model architecture choices include BERT-based models, sentence transformers, and newer instruction-tuned models like E5, BGE, and GTE or OpenAI & Gemini Embedding Models. Dimensionality affects storage requirements and computational cost, typically ranging from 384 to 1536 dimensions. Domain specificity matters because models fine-tuned on domain-specific data often outperform general-purpose models. Multilingual capabilities are necessary for non-English or mixed-language collections.

Vector search fundamentals:

Distance metrics include cosine similarity (most common), Euclidean distance, and dot product. The choice depends on whether embeddings are normalized. Approximate nearest neighbour (ANN) algorithms like HNSW, IVF, and ScaNN trade exactness for speed, essential for large-scale deployments. Search parameters such as the number of candidates to consider and the number of results to return affect both quality and latency.

Embedding best practices:

Query and document embeddings may benefit from different prompts or prefixes, especially with instruction-tuned models. Batch embedding during ingestion amortizes API costs and latency. Embedding caching avoids redundant computation for unchanged content.

---

5. Sparse Retrieval (BM25 and variants)

Sparse retrieval uses term-frequency-based methods to find documents containing query terms. BM25 remains a strong baseline that excels at exact match and rare term retrieval.

How BM25 works:

BM25 scores documents based on term frequency (how often query terms appear in the document), inverse document frequency (how rare the term is across the corpus), and document length normalization. It assigns higher scores to documents with many occurrences of rare query terms.

Strengths of sparse retrieval:

Excellent for exact keyword matching and rare or domain-specific terminology. No training required and computationally efficient. Interpretable results where you can explain why a document matched. Handles out-of-vocabulary terms that embedding models might misrepresent.

Limitations:

Cannot capture semantic similarity between different words with the same meaning. Struggles with paraphrased queries or documents. Sensitive to vocabulary mismatch between queries and documents.

Implementation options:

Elasticsearch and OpenSearch provide production-grade BM25 implementations. Lucene-based libraries like PyLucene or Whoosh work for smaller deployments. Some vector databases now include built-in BM25 support.

---

6. Hybrid Retrieval (Dense + Sparse)

Hybrid retrieval combines dense vector search with sparse keyword matching to leverage the strengths of both approaches.

Combination strategies:

Score fusion takes weighted combinations of normalized scores from both retrieval methods. Reciprocal rank fusion (RRF) combines rankings based on position rather than raw scores, reducing sensitivity to score scale differences. Cascaded retrieval uses one method for initial candidate retrieval and the other for refinement. Learned combination trains a model to optimally combine signals from both approaches.

Implementation patterns:

Parallel execution runs both retrievers simultaneously and merges results. Sequential execution uses sparse retrieval for initial filtering, then dense retrieval for reranking, or vice versa. Adaptive routing selects the retrieval method based on query characteristics.

Tuning hybrid weights:

The optimal balance depends on query types and corpus characteristics. Keyword-heavy queries benefit from higher sparse weights, while semantic queries favor dense retrieval. Some systems learn adaptive weights per query.

---

7. Reranking (Cross-Encoders, ColBERT)

Reranking applies more computationally expensive models to a smaller set of candidates to improve precision.

Cross-encoder reranking:

Cross-encoders process the query and document together as a single input, enabling full attention between them. This captures fine-grained relevance signals that bi-encoder approaches miss. The tradeoff is computational cost: cross-encoders cannot pre-compute document representations.

Common cross-encoder models include those from the sentence-transformers library fine-tuned for reranking, as well as specialized models like ms-marco-MiniLM.

ColBERT and late interaction:

ColBERT computes token-level embeddings for both query and document, then uses late interaction (MaxSim) to compute relevance. This enables document pre-computation while preserving fine-grained matching. ColBERTv2 and PLAID improve efficiency through compression and optimized retrieval.

Reranking pipeline design:

Retrieve a larger candidate set (typically 50-200 documents) using fast retrieval methods. Apply the reranker to score all candidates. Return the top-k highest-scoring results. Consider computational budget: reranking adds latency proportional to candidate count times model inference time.

---

8. Query Rewriting & Understanding

Query processing transforms user input into forms more likely to retrieve relevant documents.

Query expansion techniques:

Synonym expansion adds related terms to broaden recall. Acronym expansion handles abbreviations common in domain-specific contexts. LLM-based expansion generates semantically related queries or hypothetical answers (HyDE). Pseudo-relevance feedback uses terms from top initial results to expand the query.

Query decomposition:

Complex questions can be broken into sub-questions, each answered independently before synthesis. This is particularly valuable for multi-hop reasoning where the answer requires information from multiple documents.

Intent classification:

Understanding query intent (factual lookup, comparison, procedural guidance, etc.) enables retrieval strategy selection. Different intents may benefit from different retrieval approaches or prompt templates.

Query normalization:

Spelling correction, lowercasing, stopword handling, and stemming/lemmatization can improve matching, particularly for sparse retrieval.

---

9. Context Assembly for Answer Generation

Context assembly selects and arranges retrieved chunks to provide the language model with optimal input for answer generation.

Selection strategies:

Top-k selection takes the highest-scoring chunks up to a token budget. Diversity-aware selection ensures coverage of different aspects or sources. Maximal marginal relevance (MMR) balances relevance with diversity by penalizing redundancy. Hierarchical selection retrieves summary-level content first, then drills into details as needed.

Ordering and formatting:

Position matters because language models often attend more strongly to content at the beginning and end of the context. Recent research suggests "lost in the middle" effects where middle content receives less attention. Source attribution markers help the model distinguish between different sources. Structured formatting with clear separators between chunks aids comprehension.

Context compression:

For long contexts, compression techniques can extract key sentences or generate summaries to fit within token limits. This trades some detail for the ability to include more sources.

---

10. Grounded Answer Generation

Grounded generation produces answers that are faithful to the retrieved context rather than relying on the model's parametric knowledge.

Prompting strategies:

Explicit grounding instructions direct the model to base answers only on provided context. Quote-first approaches ask the model to identify relevant quotes before synthesizing an answer. Structured output formats separate retrieved information from generated synthesis.

System prompt design:

Effective prompts establish clear expectations about using retrieved context, handling information not present in context, and acknowledging uncertainty. They should also specify desired answer format, length, and style.

Faithfulness techniques:

Chain-of-thought prompting that references specific passages increases groundedness. Asking the model to cite sources as it generates encourages attention to context. Post-hoc verification checks generated claims against retrieved content.

---

11. Citation & Attribution

Citation mechanisms link generated statements to their source documents, enabling verification and building trust.

Citation granularity:

Document-level citations reference the source document for each claim. Passage-level citations point to specific retrieved chunks. Sentence-level citations provide the finest granularity, linking individual sentences to supporting evidence.

Implementation approaches:

Inline citations embed references within the generated text using markers like [1] or superscripts. Post-hoc citation matching aligns generated sentences with source passages after generation. Generation-time citation asks the model to produce citations as part of its output.

Citation quality:

Citations should be verifiable, meaning the cited passage actually supports the claim. Completeness ensures all claims have appropriate citations. Relevance means citations point to the most directly supporting evidence, not tangentially related content.

---

12. Hallucination Detection & Reduction

Hallucinations occur when generated content is not supported by the retrieved context or contradicts factual knowledge.

Types of hallucinations:

Intrinsic hallucinations contradict information in the provided context. Extrinsic hallucinations introduce claims not supported by the context, even if potentially true. Fabricated citations reference non-existent sources or misattribute information.

Detection methods:

Entailment-based detection checks whether retrieved context entails generated claims. Self-consistency sampling generates multiple responses and identifies claims that appear inconsistently. Fact verification pipelines decompose claims and verify each against context or external knowledge. Confidence calibration uses model uncertainty signals to flag potentially unreliable content.

Reduction strategies:

Improved retrieval ensures the model has access to relevant information. Stronger grounding prompts reinforce the instruction to use only provided context. Constrained decoding limits generation to content supported by context. Post-generation filtering removes or flags unsupported claims.

---

13. Multi-Modal RAG (Images, Tables, PDFs)

Multi-modal RAG extends retrieval and generation to handle non-textual content.

Image handling:

Image captioning generates textual descriptions for embedding and retrieval. Vision-language models like CLIP enable direct image-text similarity matching. OCR extracts text from images containing text. Visual question answering models can directly answer questions about image content.

Table processing:

Table serialization converts tables to text representations for embedding. Structured extraction preserves row/column relationships for accurate retrieval. Table-specific embedding models capture tabular semantics. SQL generation can enable precise table querying when structure is preserved.

PDF-specific challenges:

Layout analysis identifies text regions, tables, figures, and their relationships. Multi-column detection prevents text flow errors. Header/footer removal eliminates noise. Embedded image and table extraction handles rich PDF content.

Unified retrieval:

Multi-modal indexes store embeddings for different modalities in a shared space. Query routing directs queries to appropriate modality-specific retrievers. Cross-modal retrieval enables text queries to retrieve images and vice versa.

---

14. Knowledge Graphs & Structured Retrieval

Knowledge graphs represent information as entities and relationships, enabling structured queries and reasoning.

Graph construction:

Entity extraction identifies people, places, organizations, concepts, and other entities from text. Relation extraction identifies relationships between entities. Entity linking connects extracted mentions to canonical entities in a knowledge base. Schema design determines the types of entities and relationships to represent.

Graph-based retrieval:

Graph traversal retrieves entities and their neighbors based on query entities. Path finding identifies connections between entities across multiple hops. Subgraph retrieval extracts relevant portions of the graph for context. Hybrid approaches combine graph structure with vector similarity.

Integration with RAG:

Entity-centric retrieval uses identified query entities to retrieve relevant graph neighborhoods. Graph-augmented context adds structured knowledge to retrieved text. Reasoning over graphs enables multi-hop questions requiring relationship traversal. Consistency checking validates generated claims against graph structure.

---

15. RAG Evaluation Metrics

Evaluation quantifies RAG system quality across retrieval, generation, and end-to-end performance.

Retrieval metrics:

Recall@k measures the fraction of relevant documents in the top-k results. Precision@k measures the fraction of top-k results that are relevant. Mean Reciprocal Rank (MRR) considers the position of the first relevant result. Normalized Discounted Cumulative Gain (nDCG) weights relevance by position.

Generation metrics:

Faithfulness measures whether generated content is supported by context. Answer relevance assesses whether the answer addresses the question. Completeness evaluates whether all aspects of the question are addressed. Fluency and coherence measure language quality.

End-to-end metrics:

Accuracy measures correctness for questions with definitive answers. F1 score balances precision and recall for extractive answers. Human evaluation provides nuanced assessment of quality, usefulness, and trustworthiness. Task-specific metrics align with downstream use case requirements.

Evaluation frameworks:

RAGAS provides automated metrics for retrieval and generation quality. TruLens offers tracing and evaluation for LLM applications. Custom evaluation pipelines enable domain-specific assessment.

---

16. Near Real-Time Retrieval & Updates

Real-time RAG handles rapidly changing information and low-latency requirements.

Incremental indexing:

Streaming ingestion processes new documents as they arrive. Partial index updates avoid full reindexing for small changes. Write-ahead logs ensure durability without blocking queries. Eventual consistency models balance freshness with query performance.

Low-latency optimization:

Pre-computation generates embeddings asynchronously before queries arrive. Index sharding distributes load across multiple servers. Caching frequently accessed results and embeddings reduces computation. Query routing directs queries to the most appropriate index partition.

Freshness management:

Time-decay weighting prioritizes recent content in retrieval. Version tracking maintains history for rollback and auditing. Invalidation triggers remove outdated content when source documents change. Freshness indicators signal to users when information may be stale.

Part 2: Agentic RAG

1. Agentic RAG Architecture & Design Patterns

Agentic RAG introduces autonomous decision-making capabilities, enabling systems to dynamically plan and execute retrieval strategies.

Core architectural components:

The planner component analyzes queries and determines retrieval strategies. The executor carries out retrieval actions and tool calls. The evaluator assesses results and decides whether objectives are met. The synthesizer combines retrieved information into coherent responses.

Common design patterns:

The ReAct pattern interleaves reasoning and action, with the agent explicitly thinking about what to do before acting. Plan-and-execute separates high-level planning from step-by-step execution. Tree-of-thoughts explores multiple reasoning paths before selecting the best. Reflection loops enable agents to critique and improve their own outputs.

State management:

Agents maintain state across multiple retrieval steps, including retrieved documents, intermediate conclusions, and remaining objectives. State representation affects the agent's ability to make informed decisions about next steps.

---

2. Pre-Retrieval Agents (Query Optimization, Decomposition)

Pre-retrieval agents process queries before retrieval to improve result quality.

Query analysis:

Intent detection classifies the type of information need. Entity recognition identifies key entities to focus retrieval. Ambiguity detection flags queries requiring clarification. Complexity assessment determines whether decomposition is needed.

Query optimization:

Reformulation rewrites queries for better retrieval performance. Expansion adds synonyms, related terms, or context. Specialization narrows broad queries to specific aspects. Translation converts queries to match document terminology.

Query decomposition:

Sub-question generation breaks complex questions into simpler parts. Dependency identification determines the order of sub-questions when answers inform later queries. Parallel vs. sequential execution balances latency with dependency requirements.

---

3. Post-Retrieval Agents (Re-ranking, De-duplication)

Post-retrieval agents process results to improve quality before answer generation.

Intelligent reranking:

Relevance assessment scores results against query intent, not just surface similarity. Coverage analysis ensures results address all query aspects. Source quality weighting prioritizes authoritative sources. Recency consideration prioritizes fresh information when relevant.

De-duplication strategies:

Semantic de-duplication identifies results with overlapping information content. Source de-duplication limits results from any single source. Contradiction detection flags conflicting information across sources. Complementarity analysis selects results that together provide comprehensive coverage.

Result filtering:

Quality filtering removes low-quality or unreliable sources. Scope filtering ensures results match query constraints (date ranges, domains, etc.). Permission filtering respects access controls and data governance requirements.

---

4. Iterative & Multi-Hop Retrieval

Iterative retrieval executes multiple retrieval cycles, using results from each round to inform subsequent queries.

Multi-hop reasoning:

Some questions require combining information from multiple documents. The answer to one sub-question may contain terms needed to formulate the next query. Iterative retrieval chains connect these information needs.

Retrieval strategies:

Forward chaining starts from known entities and follows relationships to discover new information. Backward chaining starts from the goal and works backward to identify required information. Bidirectional approaches combine both strategies for complex queries.

Termination conditions:

Sufficiency detection determines when enough information has been gathered. Diminishing returns recognition stops when new retrievals add little value. Budget constraints limit the number of retrieval iterations. Cycle detection prevents infinite loops in retrieval chains.

---

5. Tool Use & Function Calling Integration

Agentic RAG systems integrate retrieval with other tools and capabilities.

Tool categories:

Retrieval tools include vector search, keyword search, and knowledge graph queries. Computation tools enable calculations, data analysis, and code execution. External API tools access real-time data, databases, and services. Generation tools create content, summaries, and transformations.

Tool selection:

The agent must decide which tools to use based on query requirements. Tool descriptions help the agent understand capabilities. Few-shot examples demonstrate tool usage patterns. Error handling addresses tool failures gracefully.

Orchestration:

Sequential tool calls execute one tool at a time based on results. Parallel tool calls execute independent tools simultaneously. Conditional branching selects tools based on intermediate results. Tool composition chains tool outputs as inputs to other tools.

---

6. Self-Reflection & Critique Loops

Self-reflection enables agents to evaluate and improve their own outputs.

Critique mechanisms:

Factuality checking verifies generated claims against retrieved evidence. Completeness checking ensures all query aspects are addressed. Consistency checking identifies contradictions within the response. Quality assessment evaluates clarity, coherence, and usefulness.

Improvement strategies:

Targeted retrieval fetches additional information to address identified gaps. Revision rewrites portions of the response to fix issues. Elaboration expands on areas identified as insufficiently detailed. Simplification reduces complexity when the response is unclear.

Iteration control:

Quality thresholds define acceptable output standards. Maximum iteration limits prevent infinite improvement loops. Diminishing returns detection stops when improvements become marginal. User feedback integration allows human guidance of the improvement process.

---

7. Multi-Agent Orchestration & Workflow Control

Complex tasks may require coordination among multiple specialized agents.

Agent specialization:

Researcher agents focus on information retrieval and synthesis. Analyst agents perform reasoning and evaluation. Writer agents generate polished final outputs. Critic agents evaluate and suggest improvements.

Communication patterns:

Hierarchical orchestration uses a supervisor agent directing specialist agents. Peer-to-peer collaboration allows agents to directly coordinate. Message passing enables asynchronous communication between agents. Shared memory provides a common workspace for agent collaboration.

Workflow control:

State machines define explicit transitions between agent activities. Dependency graphs manage task ordering based on information flow. Dynamic routing adapts workflows based on intermediate results. Failure recovery handles agent errors without abandoning the entire workflow.

---

8. Memory Management & Context Caching

Memory systems enable agents to leverage information across multiple interactions.

Short-term memory:

Conversation history maintains context within a session. Working memory holds intermediate results during complex reasoning. Retrieval cache stores recent query results to avoid redundant searches.

Long-term memory:

User preference storage enables personalization over time. Learned patterns capture successful strategies for reuse. Knowledge accumulation builds domain understanding across sessions.

Context management:

Context compression summarizes lengthy histories to fit token limits. Relevance filtering includes only contextually important history. Hierarchical storage maintains detailed recent context with summarized older context. Retrieval-augmented memory uses similarity search to surface relevant past interactions.

---

9. Confidence Scoring & Human-in-the-Loop

Confidence mechanisms enable appropriate human involvement in agent decisions.

Confidence estimation:

Retrieval confidence reflects the quality and relevance of retrieved information. Generation confidence assesses certainty in produced answers. Calibration ensures confidence scores correlate with actual accuracy. Uncertainty quantification distinguishes between different types of uncertainty.

Human intervention triggers:

Low-confidence results flag items for human review. High-stakes decisions require human approval regardless of confidence. Ambiguous queries prompt clarification requests. Novel situations outside training distribution escalate to humans.

Feedback integration:

Corrections update agent behavior based on human feedback. Preferences learn user-specific requirements over time. Demonstrations teach agents through examples of desired behavior. Reinforcement learning from human feedback (RLHF) systematically improves agent policies.

---

10. Failure Handling & Guardrails

Robust systems handle failures gracefully and prevent harmful outcomes.

Failure detection:

Retrieval failures occur when no relevant information is found. Generation failures produce incoherent or off-topic responses. Tool failures result from API errors, timeouts, or invalid inputs. Logical failures occur when reasoning produces inconsistent conclusions.

Recovery strategies:

Retry logic attempts failed operations with modified parameters. Fallback approaches use alternative methods when primary methods fail. Graceful degradation provides partial results when complete answers are impossible. Transparent failure communication explains limitations to users.

Safety guardrails:

Content filtering prevents generation of harmful or inappropriate content. Scope constraints keep agents within defined task boundaries. Resource limits prevent runaway computation or API costs. Output validation checks results against safety criteria before delivery.

Part 3: Infrastructure & Operations

1. Vector Database Selection & Optimization

Vector databases provide specialized storage and retrieval for embedding vectors.

Selection criteria:

Scale requirements determine whether a lightweight library or distributed database is needed. Query patterns influence index type selection. Consistency requirements affect choice between eventual and strong consistency. Operational complexity varies significantly across solutions.

Major options:

Pinecone offers fully managed cloud-native vector search. Weaviate provides open-source vector search with rich features. Milvus is designed for billion-scale vector similarity search. Qdrant offers open-source with advanced filtering capabilities. pgvector adds vector capabilities to PostgreSQL. Chroma is lightweight and developer-friendly for smaller deployments.

Optimization techniques:

Index tuning adjusts parameters for the specific data distribution. Quantization reduces memory usage with acceptable precision loss. Partitioning enables parallel search across data subsets. Replication provides both redundancy and read scaling.

---

2. Latency & Performance Tuning

Low latency is critical for interactive applications.

Latency breakdown:

Embedding computation time depends on model size and hardware. Vector search time scales with index size and search parameters. Reranking time scales with candidate count and model complexity. Generation time depends on model size and response length.

Optimization strategies:

Batch processing amortizes fixed costs across multiple items. Asynchronous processing overlaps independent operations. Hardware acceleration uses GPUs and specialized hardware for embedding and inference. Query optimization reduces search scope through filtering and approximation.

Performance testing:

Load testing identifies throughput limits and breaking points. Latency percentile analysis reveals tail latency issues. Profiling identifies bottlenecks in the processing pipeline. A/B testing measures the impact of optimizations on user experience.

---

3. Caching Strategies

Caching reduces computation by storing and reusing results.

Cache layers:

Embedding cache stores computed vectors to avoid redundant model calls. Retrieval cache stores search results for repeated or similar queries. Response cache stores final answers for exact query matches. Intermediate result cache stores partial computations for reuse.

Cache design:

Key design determines how queries map to cached results. TTL policies balance freshness with cache hit rates. Eviction strategies remove entries when cache capacity is reached. Cache warming pre-populates caches with likely queries.

Semantic caching:

Similar query detection identifies when a new query is similar enough to a cached one. Similarity thresholds balance reuse with relevance. Cache invalidation removes stale entries when source documents change.

---

4. Cost Management & Scaling

RAG systems incur costs across multiple dimensions that must be managed.

Cost components:

Embedding costs include API calls or GPU compute for generating vectors. Storage costs cover vector database and document storage. Inference costs apply to reranking models and LLM generation. Infrastructure costs include compute, networking, and operational overhead.

Cost optimization:

Embedding model selection balances quality with per-token costs. Batch processing reduces per-request overhead. Caching avoids redundant expensive operations. Tiered retrieval uses cheaper methods for initial filtering.

Scaling approaches:

Horizontal scaling adds more instances for throughput. Vertical scaling increases instance resources for per-request performance. Auto-scaling adjusts resources based on demand. Serverless deployment aligns costs with actual usage.

---

5. Monitoring & Observability

Observability enables understanding and improvement of production systems.

Key metrics:

Retrieval metrics include latency percentiles, result quality, and cache hit rates. Generation metrics cover response quality, token usage, and error rates. System metrics encompass throughput, resource utilization, and availability. Business metrics measure task completion rates and user satisfaction.

Logging and tracing:

Request logging captures query, results, and response for analysis. Distributed tracing tracks requests across system components. Error logging enables debugging of failures. Audit logging supports compliance and security requirements.

Alerting:

Threshold alerts trigger on metric values crossing defined bounds. Anomaly detection identifies unusual patterns without explicit thresholds. Escalation procedures ensure appropriate response to critical issues. Runbooks provide guidance for responding to common alerts.

Continuous improvement:

Feedback collection captures user ratings and explicit feedback. Failure analysis investigates errors to identify root causes. A/B testing measures the impact of changes on key metrics. Regular evaluation tracks quality trends over time.