==
======

GENAI, RAG, & AGENTIC SYSTEMS: THE COMPLETE 76-QUESTION TECHNICAL GUIDE (2026)

==
======

--- 🔷 RAG: CORE & FAILURE SCENARIOS ---

1. Why not fine-tune? Fine-tuning is for style/skills; RAG is for facts. RAG is cheaper, allows real-time updates, and provides citations.
2. Wrong answers (data exists): Likely retrieval failure (Top-K too low), semantic gap in embeddings, or "Lost in the Middle" context window issues.
3. Still Hallucinates: Context might be irrelevant or contradicting. Solve with "NLI" (Natural Language Inference) or grounding prompts.
4. Retrieval Health: Measure using Recall@K (is the ground truth in the results?) and MRR (Mean Reciprocal Rank).
5. Measuring Accuracy: Use the RAGAS framework (Faithfulness, Answer Relevance, Context Precision).
6. Multi-chunk answers: Use "Map-Reduce" chains or "Long Context" models with "Recursive Retrieval" to synthesize data.
7. Follow-up context: Use "Query Condensation"—re-writing a follow-up ("Tell me more") into a standalone query using chat history.
8. When NOT to use history: For high-security tasks, independent per-query isolation, or when token limits are critical.
9. Vague queries: Implement a "Clarification Agent" to ask the user for missing parameters before searching.
10. Contradictory context: Use LLM-as-a-judge to prioritize newer dates or higher-authority source metadata.
11. Prevent conversational hallucinations: Use "Chain of Verification" (CoVe) and strict "Answer only from context" system prompts.
12. DB Down: Fallback to keyword search (Elasticsearch), a static FAQ cache, or a graceful "Service Unavailable" message.

--- 🔷 RETRIEVAL, CHUNKING & SEARCH ---

13. Chunk size/overlap: Chunk size depends on "Average Information Unit" (usually 512 tokens). Overlap (10-20%) prevents loss of context at splits.
14. Hybrid Search: Vectors handle semantic meaning; BM25 (Keyword) handles specific IDs, acronyms, and rare product names.
15. Pure Keyword: Better for SKU lookups, part numbers, or when users search for exact technical strings.
16. Re-ranking: Use a Cross-Encoder (Cohere/BGE) on the top 20-50 results to prioritize the most relevant few.
17. Noisy docs: Clean HTML/OCR text pre-embedding; use a "Summary Index" for high-level queries.
18. Updating docs: Use document hashing (MD5). Only re-embed and upsert if the content hash has changed.
19. Scaling to millions: Use HNSW indexing, sharding/partitioning by metadata (e.g., Tenant ID), and quantized embeddings.

--- 🔷 LANGCHAIN: ARCHITECTURE & DESIGN ---

20. When NOT to use LangChain: For ultra-low latency requirements or simple, single-prompt apps where abstraction adds bloat.
21. Production-Grade: Use LangSmith for tracing, Pydantic for strict output parsing, and explicit error handling.
22. Stateless Memory: Use external stores (Redis/DynamoDB) to persist chat history keyed by SessionID.
23. Versioning Prompts: Use a Prompt Registry (LangSmith/Git) to decouple prompt logic from application code.
24. Observability: Integrate OpenTelemetry or LangChain Tracing to identify bottlenecks in the chain.
25. Retries/Fallbacks: Use exponential backoff for 429 errors; fallback to a cheaper model (e.g., Claude 3 Haiku) if the primary fails.

--- 💠 LANGGRAPH: AGENTIC SYSTEMS ---

26. LangGraph over LangChain: LangGraph supports "Cyclic" workflows (loops) and fine-grained state management.
27. Deterministic Workflows: Define a State Graph where transitions are governed by code logic rather than LLM whims.
28. Loop Prevention: Set a 'recursion_limit' in the graph and implement a "Max Retries" node.
29. State Management: Use a 'TypedDict' or Pydantic class to pass variables between nodes in the graph.
30. Human-in-the-loop: Use 'interrupt_before' to pause the graph for human approval on sensitive actions (e.g., deleting data).
31. Debugging: Use "State Snapshots" to inspect the exact values in the graph at any point of failure.
32. LangGraph + RAG: Allows for "Self-RAG" where the agent critiques its own retrieval and re-tries if the context is poor.

--- 💠 MULTI-AGENT DESIGN ---

33. Hiring/Support System: Design a "Supervisor" agent to route tasks to "Resume Screener," "Scheduler," or "QA" agents.
34. Agent Creation: Create specialized agents (Researcher vs. Writer) to reduce the "Cognitive Load" on a single LLM call.
35. Communication: Agents communicate via a "Shared State" (LangGraph) or "Message Passing" (AutoGen style).
36. Agent Failure: Implement "Graceful Degradation"—if the 'Researcher' fails, the 'Writer' notifies the user.
37. Output Validation: Use Pydantic to ensure an agent's output matches the expected schema before the next agent starts.

--- 💠 TOOL CALLING & INTEGRATIONS ---

38. Preventing Wrong Tools: Use strict Pydantic schemas and Few-Shot examples in the tool description.
39. Validating Tools: Use "Guard" functions to check tool outputs for PII or nonsensical data before returning to the LLM.
40. Partial Failures: Catch tool exceptions and pass the error back to the LLM so it can attempt a "Self-Correction."
41. Idempotency: Ensure tool calls (like "Charge User") won't execute twice if a retry occurs (use unique RequestIDs).

42. Security: Never give an LLM raw SQL access. Use a middleware API with restricted scopes (Least Privilege).

--- 💠 AWS GENAI & ARCHITECTURE ---

43. Bedrock vs. OpenAI: Bedrock offers VPC security, SOC/HIPAA compliance, and access to multiple model providers.
44. SageMaker vs. Bedrock: SageMaker is for custom fine-tuning or hosting proprietary model weights; Bedrock is for serverless API use.
45. Enterprise RAG on AWS: Ingest via S3 -> Lambda -> OpenSearch (Vector). Query via Bedrock.
46. Multi-tenant AWS: Use OpenSearch "Fine-Grained Access Control" and Metadata filtering to isolate data.
47. RAG Pipeline Services: S3 (Storage), Bedrock (LLM), OpenSearch (Vector DB), Lambda (Orchestration).
48. Scaling on AWS: Use Bedrock "Provisioned Throughput" for guaranteed latency during peaks.
49. Secrets: Store API keys and Database credentials in AWS Secrets Manager, never in environment variables.
50. OpenSearch Hybrid: Use "Search Pipelines" in OpenSearch to combine k-NN scores with BM25 scores.

--- 💠 COST, PERFORMANCE & SCALING ---

51. Cost Spike (5x): Check for infinite loops in agents or a "Prompt Injection" attack. Check token usage per user.
52. Monitoring: Use CloudWatch metrics or LangSmith to track token consumption per "Provider/Model."
53. Reduce Cost: Use "Semantic Caching" (Redis) for common queries and "Prompt Compression."
54. Caching: Cache the (Query, Response) pair using a vector similarity check to serve "close enough" answers.
55. Traffic Spikes: Use a "Queue-based" architecture (SQS) to throttle requests to the LLM.

--- 💠 SECURITY, COMPLIANCE & SAFETY ---

56. Private Data Leakage: Use PII Redaction (Comprehend) on the input before it hits the embedding or LLM.
57. Tenant Isolation: Use "Metadata Filtering" (e.g., `where tenant_id = 'A'`) in every vector search.
58. Prompt Injection: Use "Delimiters" and a "System Guard" model to scan user input for malicious instructions.
59. Embedded Malice: Scan retrieved documents for "Ignore previous instructions" strings before passing to the LLM.
60. PII in Responses: Use an "Output Guardrail" to mask emails/phone numbers in the final response.
61. Guardrails: Use Amazon Bedrock Guardrails to filter toxic content and PII automatically.

--- 💠 EVALUATION & RELIABILITY ---

62. Production Eval: Use "LLM-as-a-judge" (G-Eval) to score production traces on a scale of 1-5.
63. Metrics: Faithfulness (is it in the doc?), Answer Relevance, and Context Recall.
64. Drift Detection: Monitor "Embedding Drift" (average distance of queries over time) or "Output Length" shifts.
65. Model Upgrade: Always run a "Golden Dataset" (test set) against the new model version before switching.
66. A/B Testing: Route 10% of traffic to a new prompt/model and compare "User Thumbs Up/Down" rates.
67. Rollback: Maintain a "Blue/Green" deployment for LLM chains to switch back to the old model instantly.

--- 🔷 SYSTEM DESIGN ---

68. Enterprise Doc AI: S3 -> Textract -> Hybrid OpenSearch -> Bedrock (Claude 3.5 Sonnet) -> Citations.
69. HR AI: Focus on "Strict PII masking" and "Policy Routing" to ensure the AI doesn't give legal advice.
70. Support AI: Multi-agent (Router -> Docs Agent -> CRM Agent) with Human-in-the-loop for refunds.
71. Explaining to Leadership: "It's an 'Open Book' system that reduces cost by automating 80% of routine lookups."
72. Auditability: Log every Trace ID, retrieved Chunk IDs, and the Model Version to a centralized database.

--- 🔷 LEADERSHIP & COMMUNICATION ---

73. Explain RAG: "It's like giving the AI a search engine so it doesn't have to rely on its memory."
74. Justifying Cost: Compare LLM cost per ticket vs. Human cost per ticket. Highlight 24/7 availability.
75. Build vs. Buy: Buy for generic tasks (summarizing emails); Build for proprietary data (your company’s IP).
76. Client Failures: Acknowledge the error, explain the "Guardrail" being added, and emphasize the "Human-in-the-loop" safety net.

===