RAG Precision Improvement Guide · For AI-Powered Observability Systems

Part 1: Approaches Checklist

Already Applied in This Project

#	Approach	Implementation
1	Metadata filtering (dynamic per query)	`RagConfig.dynamicFilter()` routes ERROR queries to `hasErrors=true` chunks
2	Query classification	`QueryClassifier.java` · keyword-based intent detection, no LLM call
3	Trace deduplication	In-memory `ConcurrentHashMap` + `embeddingStore.removeAll(traceId filter)`
4	Document type tagging	`type=documentation` and `type=telemetry` metadata on all chunks
5	Metadata-annotated context	LLM sees `[TRACE | service=X | traceId=Y]` prefix per chunk
6	Structured prompt	Role, query type, instructions, output format in prompt template
7	Lower temperature	0.7 to 0.3 for factual precision
8	Dynamic maxResults/minScore	Error queries get 7 chunks at 0.4 threshold; docs get 5 at 0.5

Not Yet Applied · Next Steps

#	Approach	What It Does	How to Apply
9	Evaluation framework	Measure if answers are improving	Create 20 Q&A pairs with expected answers, score each response
10	Hybrid search (keyword + vector)	Vector search misses exact matches like trace IDs	Use ChromaDB `where_document` for keyword matching alongside similarity
11	Chunk size tuning	Balance between context and noise	Experiment: 1000/100 vs 1500/300 vs 2000/400, measure answer quality
12	Query expansion	Broaden retrieval for ambiguous questions	"Why slow?" also searches "latency", "duration", "bottleneck"
13	Contextual compression	Remove irrelevant parts from chunks	Post-retrieval: strip non-relevant spans from trace narratives
14	Multi-query retrieval	Complex questions need multiple searches	"Compare order vs payment perf " becomes 2 separate retrievals
15	Embedding model eval	nomic-embed-text may not be optimal	Try mxbai-embed-large, all-minilm · compare retrieval recall
16	Few-shot prompting	Show the LLM what a good answer looks like	Add 1-2 example Q&A pairs in the system prompt
17	Negative prompting	Tell the LLM what NOT to do	"Do NOT summarize. Extract specific facts with trace IDs."
18	Source attribution	Track which chunks contributed to answer	Log chunk usage, compare against answer content
19	Time-decay weighting	Recent traces rank higher	Add `ingestedAt` metadata, boost recent chunks
20	Guardrails	Prevent hallucination	Post-process: flag answers that don't reference any trace ID or source

Part 2: The RAG Precision Stack

Layer 1: Data Quality (Foundation)

- Clean, well-structured documents
- Rich metadata on every chunk (type, source, traceId, service, errors)
- Deduplication at ingestion time
- Appropriate chunk sizes per data type

Layer 2: Retrieval Engineering (Highest Impact)

- Metadata filtering narrows search space before similarity matching
- Dynamic retrieval parameters per query type
- Hybrid search (vector + keyword) for exact matches

- Query expansion for better recall

Layer 3: Context Engineering (Medium Impact)

- Annotate chunks with metadata labels for LLM consumption
- Contextual compression removes noise
- Multi-query retrieval for complex questions
- Re-ranking retrieved chunks by relevance (requires scoring model)

Layer 4: Prompt Engineering (Fine-Tuning the Output)

- Structured role and instructions
- Query type awareness in prompt
- Few-shot examples of ideal answers
- Negative examples of what to avoid
- Lower temperature for factual precision

Layer 5: Evaluation (Continuous Improvement)

- Ground truth Q&A test set
- Automated scoring (exact match, semantic similarity, human eval)
- A/B testing different retrieval strategies
- Logging and monitoring retrieval quality in production

Part 3: Books and Resources

Practical RAG and Prompt Engineering

- "Building LLM Apps" by Valentina Alto Hands-on RAG patterns with LangChain. Covers chunking, retrieval, prompt design.
- "Prompt Engineering for Developers" (DeepLearning.AI, free course) Andrew Ng + OpenAI. Short, practical, directly applicable.
- LlamaIndex Documentation (llamaindex.ai) Best free reference for RAG architecture patterns, even if you use LangChain4j.

Understanding LLMs (Application-Level)

- "Build a Large Language Model from Scratch" by Sebastian Raschka Builds intuition for how LLMs work without requiring PhD-level math.
- "Let's Build GPT" by Andrej Karpathy (YouTube, 2 hours) Best ROI for understanding transformer architecture. Watch once.
- "AI Engineering" by Chip Huyen Production AI systems · evaluation, deployment, monitoring. Directly relevant to what you're building.

Observability (Your Domain)

- "Observability Engineering" by Charity Majors, Liz Fong-Jones, George Miranda The observability bible. Traces, metrics, logs, SLOs.
- "Distributed Tracing in Practice" by Austin Parker et al. Directly relevant to your project's tracing

architecture.
- OpenTelemetry Official Documentation (opentelemetry.io) The spec you're instrumenting against. Essential reference.

Skip These (Overkill for Application Engineers)

- Stanford CS336 · model training from scratch
- "Attention Is All You Need" paper deep-dive · skim it, don't study it
- GPU/CUDA programming material
- Any foundation model training curriculum

Part 4: Learning Path Recommendation

Month 1: Retrieval Engineering (You Are Here)

- Applied metadata filtering, dedup, dynamic retrieval
- Next: Add evaluation framework (approach #9)
- Next: Try hybrid search (approach #10)
- Read: LlamaIndex RAG patterns docs

Month 2: Production Patterns

- OTel Collector integration (fan-out for real-time embedding)
- Error handling + structured logging across services
- Resilience patterns (circuit breaker, retry)
- Read: "Observability Engineering" by Charity Majors

Month 3: Advanced RAG

- Multi-query retrieval for complex questions
- Contextual compression
- Embedding model comparison
- Few-shot prompting with domain examples
- Read: "AI Engineering" by Chip Huyen

Month 4: Evaluation and Optimization

- Build automated evaluation pipeline
- A/B test retrieval strategies
- Time-decay weighting for traces
- Guardrails and hallucination detection
- Watch: Karpathy's "Let's Build GPT"

Part 5: Key Insight

The single most important thing to understand:

RAG precision is 80% retrieval, 15% prompt, 5% model.

If the right chunks don't reach the LLM, no amount of prompt engineering or fine-tuning will fix the answer. Fix retrieval first. Always.

Project: Incident Triage System Author: Revanth Katanguri Generated: February 2026