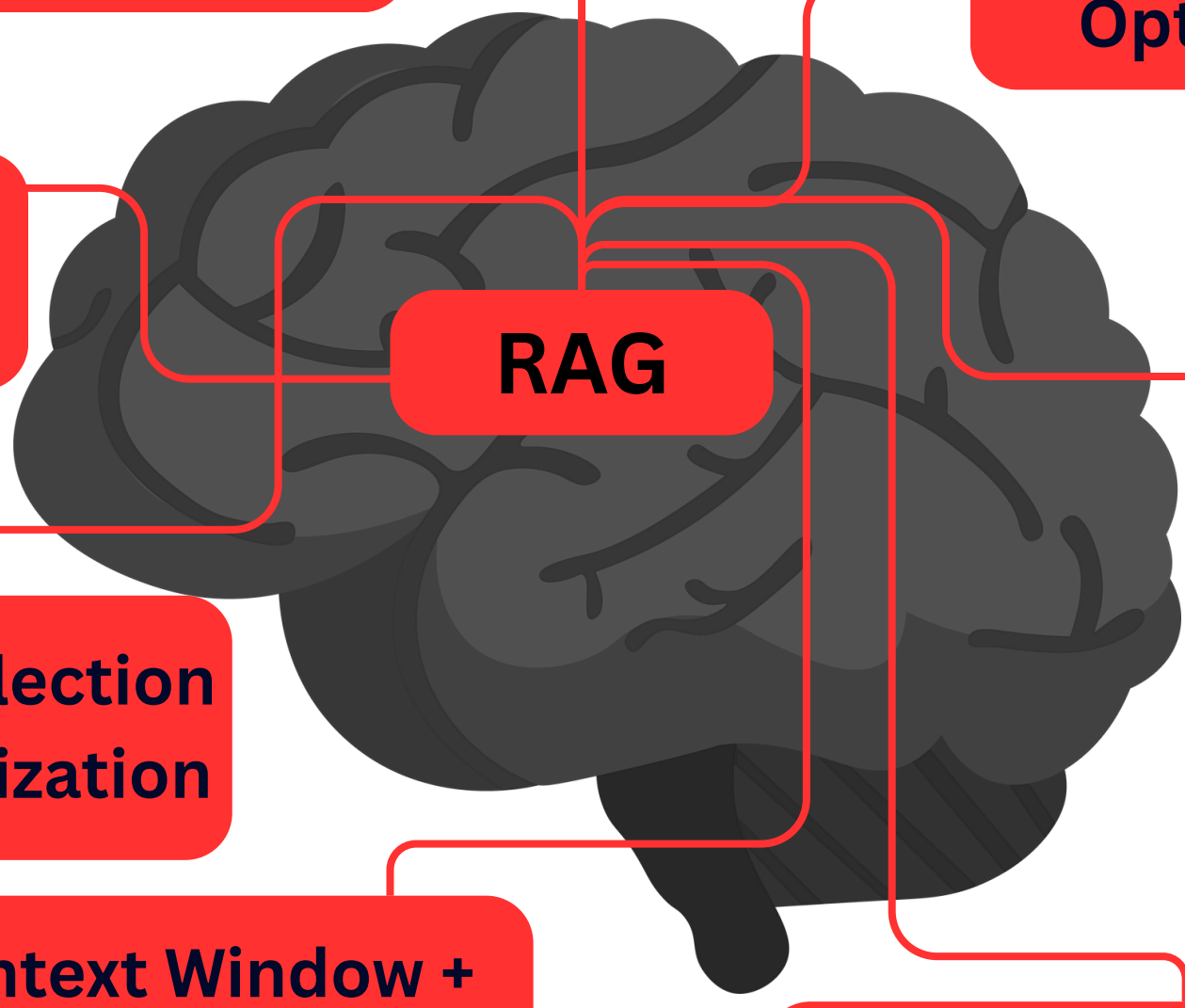
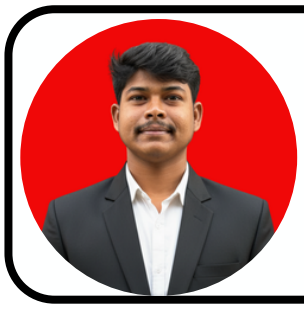
PracticAI

7 Techniques to Reduce Latency in RAG Systems

Build faster and smoother retrieval pipelines



1. Vector Database Optimization
2. Caching Strategies
3. Reranking Optimization
4. Context Window + Prompt Optimization
5. Model Selection and Optimization
6. Parallel Processing
7. Smart Routing and Query Classification



Naresh Edagotti

@PracticAI

What Is Latency?

- Latency is the total time your RAG system takes to answer a query.
- It comes from retrieval, reranking, context processing, and the LLM's generation time.

Lower latency = smoother UX + higher throughput + lower costs.

Example

Query: "Compare ELSS and PPF tax benefits."

Before optimizations: ~1.4 seconds

- Slow search
- Too many chunks
- Long prompt sent to the LLM

After optimizations: ~350–500 ms

- ANN search
- Cache-first retrieval
- Fewer chunks
- Shorter prompt

PracticAI

01 Vector Database Optimization

What It Is

Optimizing how vectors are stored, indexed, and retrieved to minimize search time.

How to Use in RAG

- **Use approximate nearest neighbor (ANN)** algorithms like HNSW (Hierarchical Navigable Small World) or IVF (Inverted File Index) instead of exhaustive search
- **Implement proper indexing strategies** based on your data size and query patterns
- **Optimize embedding dimensions** by using dimensionality reduction techniques like PCA when appropriate
- **Use quantization to reduce vector storage** size and speed up similarity calculations

When to Use

- When retrieval time is the bottleneck (typically over 200ms)
- For large-scale knowledge bases (10K+ documents)
- When you need to support high query throughput
- When memory constraints are a concern

PracticAI

02 Caching Strategies

What It Is

Storing frequently accessed results to avoid redundant computation and retrieval operations.

How to Use in RAG

- **Query caching:** Store results for identical or semantically similar queries
- **Embedding caching:** Cache computed embeddings for common queries
- **Context caching:** Cache retrieved contexts for popular topics
- **Multi-level caching:** Implement L1 (in-memory), L2 (Redis), and L3 (disk) caches

When to Use

- When you have repetitive or similar queries (common in customer support, FAQ systems)
- For queries with predictable patterns
- When serving multiple users with overlapping information needs
- To reduce load on vector databases and LLM APIs

03 Reranking Optimization

What It Is

Efficiently narrowing down retrieved documents to the most relevant ones before sending to the LLM.

How to Use in RAG

- **Two-stage retrieval:** Fast initial retrieval (high k) followed by precise reranking (lower k)
- **Lightweight rerankers:** Use faster models like cross-encoders with smaller architectures
- **Score-based filtering:** Set relevance thresholds to skip reranking when confidence is high
- **Hybrid search:** Combine vector and keyword search for better initial results

When to Use

- When initial retrieval returns too many irrelevant results
- To improve accuracy without significantly increasing latency
- When context window size is limited and you need the best chunks
- For queries requiring semantic understanding beyond simple similarity

PracticAI

04 Context Window + Prompt Optimization

What It Is

Fine-tuning how much context and prompt text you send to the LLM so it processes fewer tokens and responds faster.

How to Use in RAG

- **Dynamic k:** Pick the number of chunks based on how complex the query is.
- **Smaller chunks:** Keep chunk size around 256–512 tokens to speed up retrieval.
- **Context compression:** Summaries, distilled facts or extracted key lines instead of full text.
- **Relevance filtering:** Include only chunks that cross a similarity score threshold.
- **Prompt tightening:** Remove fluff, reduce instructions, and avoid redundant system text.
- **Template optimization:** Use short, reusable prompt templates with fewer tokens.

When to Use

- When LLM generation time is slowing things down.
- When the query doesn’t need a lot of background context.
- When you want to cut API cost while improving speed.
- When retrieved text is long, repetitive or noisy.

05 Model Selection and Optimization

What It Is

Choosing and configuring appropriate models for embedding and generation to balance speed and quality.

How to Use in RAG

- **Smaller embedding models:** Use models like all-MiniLM-L6-v2 (384 dim) instead of larger alternatives
- **Faster LLMs:** Consider using smaller models (GPT-3.5, Claude Haiku) for simpler queries
- **Quantized models:** Use 4-bit or 8-bit quantization for local models
- **Model routing:** Route queries to appropriate models based on complexity

When to Use

- When generation time dominates total latency
- For applications where slight quality trade-offs are acceptable
- When running models locally with limited resources
- To optimize cost alongside latency

06 Parallel Processing

What It Is

Executing multiple operations simultaneously to reduce overall processing time.

How to Use in RAG

- **Parallel retrieval:** Query multiple vector stores or shards simultaneously
- **Concurrent embedding generation:** Process multiple chunks in parallel
- **Asynchronous operations:** Use async/await patterns for I/O operations
- **Batch processing:** Group similar operations together

When to Use

- When you have multi-source retrieval (multiple databases, different document types)
- For large document collections that can be sharded
- When retrieval and reranking can be parallelized
- In systems with sufficient CPU/GPU resources

07 Smart Routing and Query Classification

What It Is

Analyzing queries to route them efficiently through the most appropriate retrieval and generation pipeline.

How to Use in RAG

- **Intent classification:** Identify query type (factual, analytical, conversational)
- **Complexity scoring:** Route simple queries to faster pipelines
- **Domain routing:** Direct queries to specialized sub-indexes
- **Cache-first routing:** Check caches before triggering full retrieval

When to Use

- In multi-domain knowledge bases
- When queries have distinct patterns or categories
- To optimize resource allocation across different query types
- For systems serving diverse use cases

LIKE THIS

CONTENT?

FOLLOW FOR MORE!

NARESH EDAGOTTI

LIKE REPOST SAVE