

“Taking a RAG POC to production is mainly about making retrieval reliable, securing the system, controlling hallucinations, and making it observable and scalable.”

1. Improve the ingestion pipeline

POC usually does:

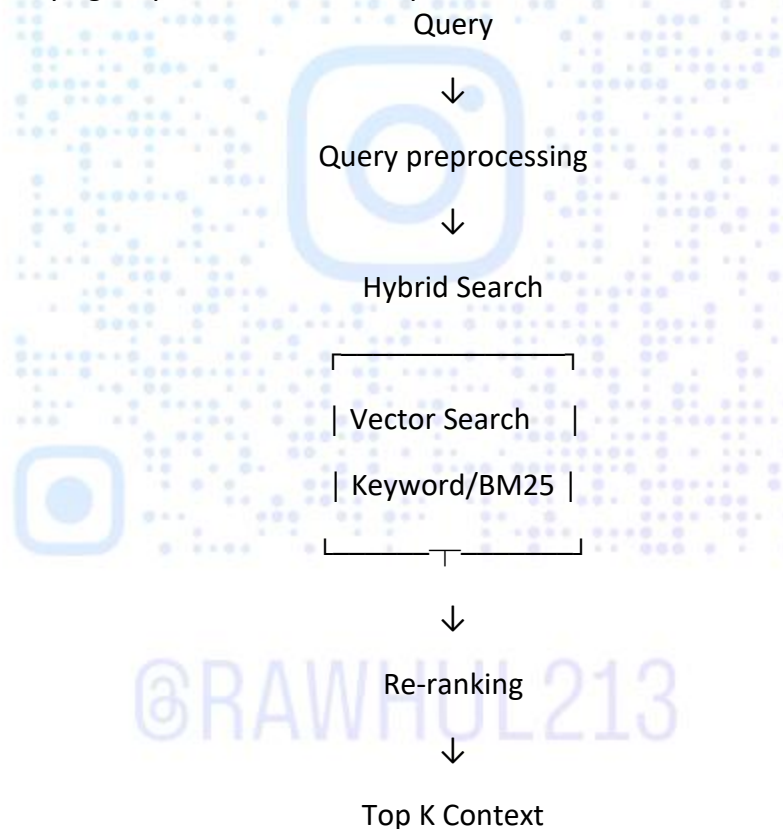
PDF → chunks → embeddings → vector DB

Production needs:

Multiple file formats-Document parsing and OCR, Metadata extraction, Better chunking strategy, Deduplication, Incremental document updates/deletions, Versioning
Async/background ingestion, Failure handling and retry

2. Production-grade retrieval

Instead of relying only on vector similarity:



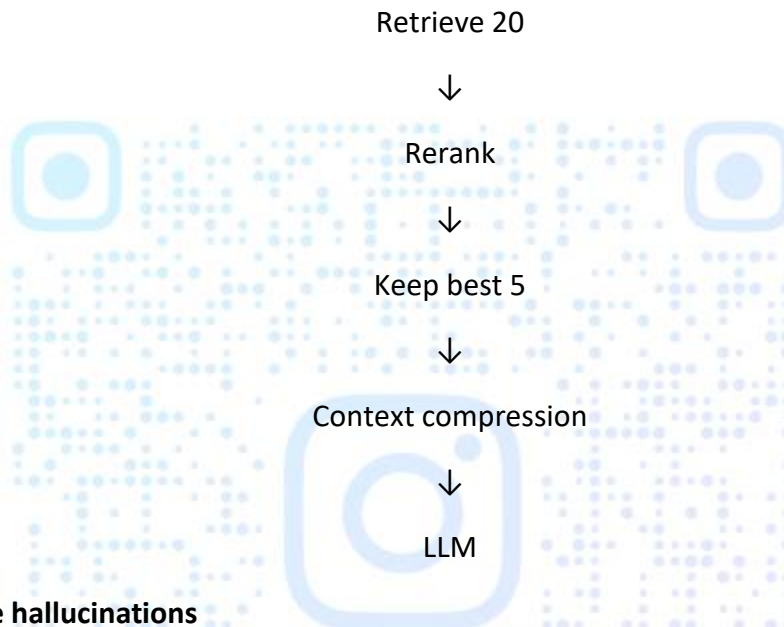
I'd typically use **hybrid retrieval + reranking** because pure semantic search can miss exact terms such as IDs, product names, error codes, or technical terminology.

3. Control the context

Don't blindly send Top-K chunks to the LLM.

- Relevance threshold
- Reranking
- Deduplication
- Context compression
- Token-budget management
- Metadata filtering

For example:



4. Reduce hallucinations

The production prompt should enforce:
Answer only from retrieved context.

If the context doesn't contain the answer,
say "I don't have enough information."

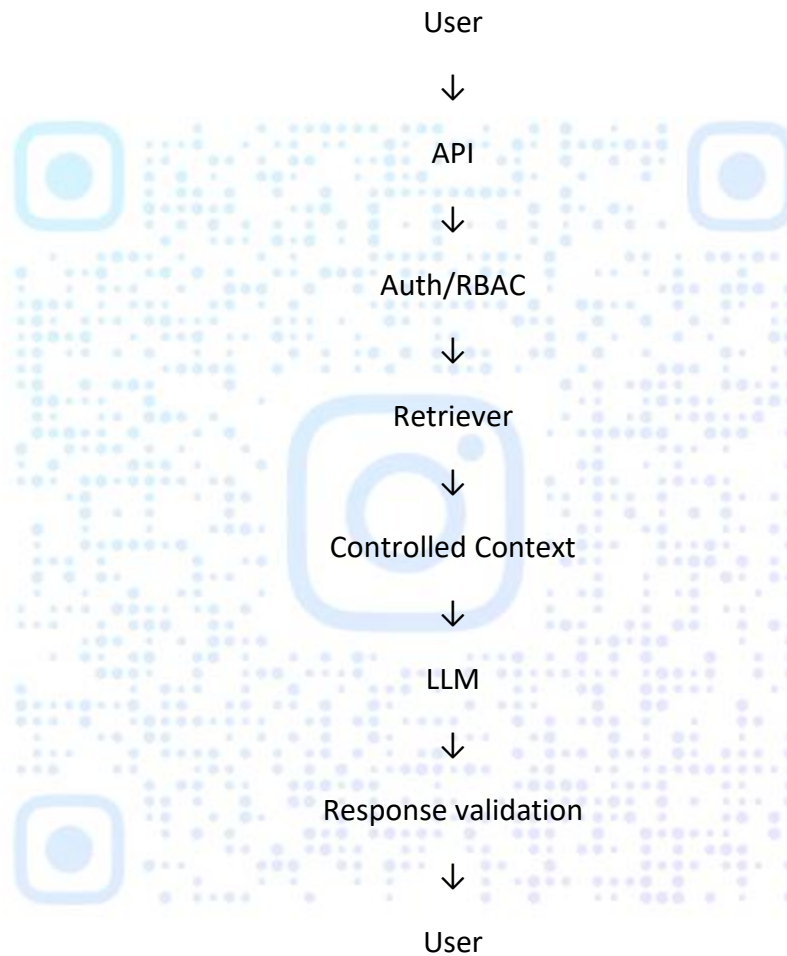
Also return **citations/source references** so users can verify the answer.
I'd additionally evaluate:

- Faithfulness
- Context relevance
- Answer relevance
- Citation correctness
- Retrieval recall

5. Security

- This becomes very important in production. I would implement:
- Authentication
- Authorization/RBAC
- Tenant isolation
- Encryption
- Secrets management
- PII handling
- Prompt-injection protection
- Input/output validation
- Rate limiting

For example, I wouldn't allow the LLM to directly access the production database.
Instead:



6. Observability

A POC usually logs:
question → answer

Production should trace the entire pipeline:

Request

- └─ latency
- └─ retrieved documents
- └─ similarity scores
- └─ reranker scores
- └─ prompt tokens
- └─ completion tokens
- └─ model
- └─ latency per component
- └─ errors
- └─ final response

This allows you to answer **why the RAG system gave a bad answer**.

7. Evaluation before deployment

I'd create a golden dataset:

Question, Expected Answer, Relevant Documents

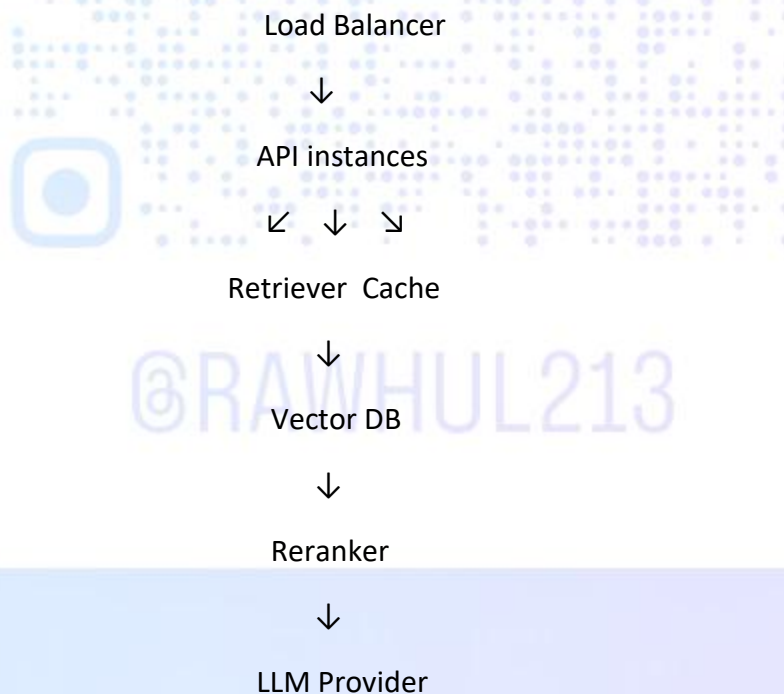
Then run automated evaluation whenever retrieval, prompts, embeddings, or models change.

I'd measure things such as:

- Retrieval Recall@K
- MRR / NDCG
- Context Precision
- Context Recall
- Faithfulness
- Answer Relevance
- Latency
- Cost

8. Scalability

For production traffic:



Go with-

- Async APIs
- Connection pooling
- Caching
- Horizontal scaling
- Queue-based ingestion

- Batch embedding
- Streaming responses
- Timeouts/retries/circuit breakers

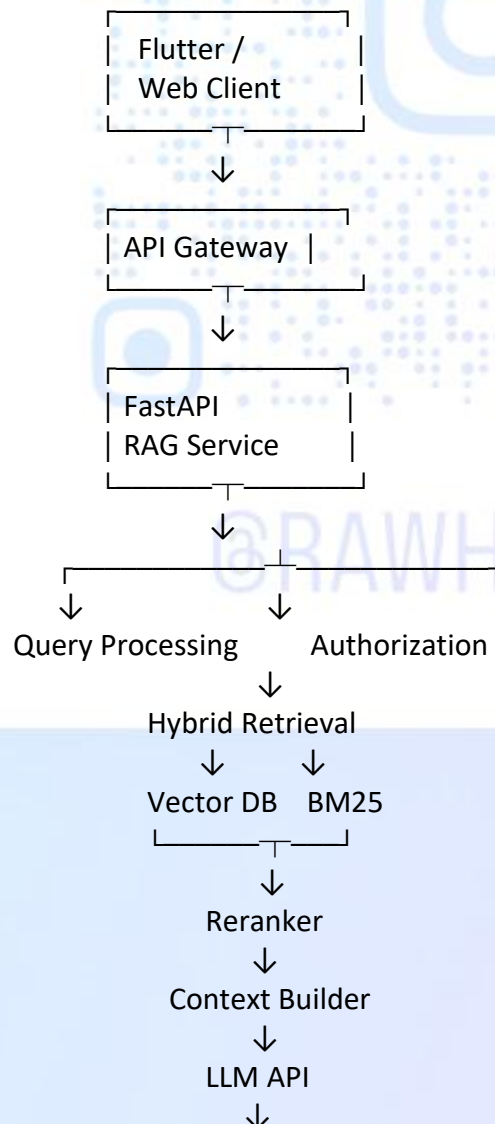
9. Cost optimization

LLM cost can become significant. I will optimize:

- Smaller model for query rewriting/classification
- Larger model only for final generation
- Embedding caching
- Response caching where appropriate
- Reduce unnecessary retrieved context
- Batch embedding
- Token limits

10. Deployment architecture

A practical production architecture could be:



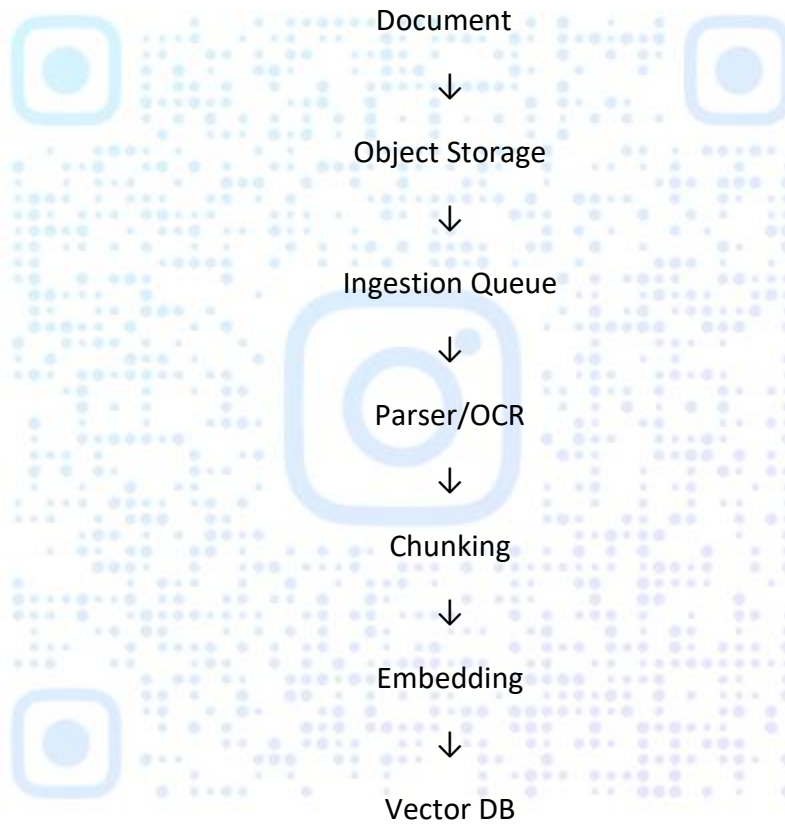
<https://www.instagram.com/rawhul213>

Guardrails/Validation



Response

Documents Ingestion:-



@RAWHUL213