ADROSONIC®

www.adrosonic.com

Vector Database Design for Large-Scale Precision Retrieval in RAG Systems

Designing high-precision retrieval pipelines to eliminate LLM hallucinations in enterprise knowledge systems.

24-Hour Hackathon Challenge · AI / ML · Information Retrieval

24 hrs	2–4	Working demo	RAGAS
CHALLENGE DURATION	TEAM SIZE	REQUIRED	EVALUATION FRAMEWORK

01 PROBLEM BACKGROUND

Business Context

Retrieval-Augmented Generation (RAG) is the standard approach for connecting large language models to private or enterprise data. Before answering a question, the system retrieves the most relevant document chunks from a knowledge base and passes them to the LLM as context.

The problem is that, at scale, standard vector search often returns chunks that are semantically similar but factually incorrect for the specific query. The LLM then hallucinates, generating plausible-sounding but wrong answers. In enterprise settings, this is a trust-breaking failure mode with real operational consequences.

> **CORE CHALLENGE**
> Build a RAG retrieval system that is fast, precise, and demonstrably better than a naive baseline, running entirely on free-tier tools and consumer hardware.

02 OBJECTIVE

Build a vector database and retrieval pipeline that:

- Indexes a meaningful dataset (100,000+ passages from MS MARCO).
- Retrieves relevant context with measurably higher precision than a baseline naive RAG.
- Responds in under 300 milliseconds at query time.
- Includes a simple query interface to demonstrate retrieval quality.

03 TWO-PHASE STRUCTURE

The challenge is split into two phases. Phase 1 is your foundation; build it cleanly and Phase 2 becomes significantly easier.

Phase 1: Baseline RAG

Tasks

- Ingest a portion of the MS MARCO dataset into your chosen vector database (Qdrant, Weaviate, ChromaDB, or pgvector).
- Implement a simple dense vector search pipeline using a HuggingFace embedding model (BGE-small, MiniLM, or similar).
- Build a basic query interface: type a question, retrieve the top-5 passages.
- Record your baseline RAGAS scores (Context Precision and Context Recall).

Deliverable: architecture diagram and a working naive RAG demo with logged RAGAS baseline scores.

Phase 2: Optimised Retrieval

Tasks

- Implement hybrid search: combine dense vector search with sparse keyword search (BM25), the single highest-impact improvement.
- Add metadata filtering: allow queries with constraints such as “documents from category X only”.
- Benchmark: run 100 consecutive queries and log latency. Target: p95 under 300 ms.
- Re-run RAGAS evaluation and show improvement over the Phase 1 baseline.
- Add upsert / delete functionality to show your index supports live updates.

Deliverable: a benchmarking report (PDF or Markdown) showing Phase 1 versus Phase 2 RAGAS scores, latency logs, and a GitHub repo with a README.

04 FUNCTIONAL REQUIREMENTS

FR-1 Data Ingestion and Indexing

- Download and parse the MS MARCO corpus from HuggingFace datasets.
- Index a minimum of 100,000 passages into your chosen vector database.
- Generate embeddings using a HuggingFace sentence-transformer model.
- Store metadata alongside vectors (passage ID, source, optional category tag).

- The indexing pipeline must complete in under 2 hours on consumer hardware (CPU acceptable).

FR-2 Baseline Dense Retrieval (Phase 1)

- Implement cosine similarity or dot-product search over the embedded passage index.
- Return the top-5 passages for any natural language query.
- Log query response time for all evaluation queries.
- Compute and log RAGAS Context Precision and Context Recall on a sample of 20 or more queries.

FR-3 Hybrid Search Implementation (Phase 2)

- Implement BM25 sparse keyword search alongside dense vector retrieval.
- Combine scores using Reciprocal Rank Fusion (RRF) or a weighted linear combination.
- Hybrid search must demonstrably improve RAGAS scores over the dense-only baseline.
- The fusion method and weights must be documented and configurable.

FR-4 Metadata Filtering

- Support at least one filter dimension on query (for example, category, source, or date range).
- Filters must be applied pre-retrieval, at the vector database level, not post-retrieval.
- Demonstrate a filtered query in the live demo.

FR-5 Live Index Updates

- Implement upsert: add or update a passage without reindexing the full corpus.
- Implement delete: remove a passage by ID from the index.
- Show both operations working in the query interface or through an API call.

FR-6 Query Interface

- Build a simple UI (Streamlit, Gradio, or a web frontend) or CLI interface.
- Accept a natural language question and return the top-5 retrieved passages with scores.
- Display the retrieval method used (dense / hybrid) and the per-passage relevance score.
- Allow toggling between Phase 1 (dense) and Phase 2 (hybrid) retrieval modes.

05 NON-FUNCTIONAL REQUIREMENTS

ID	Requirement	Acceptance criterion
NFR-1	RAGAS precision	Context Precision above 0.75 in Phase 2 (Phase 1 baseline recorded for comparison).
NFR-2	RAGAS recall	Context Recall above 0.70 in Phase 2.
NFR-3	Query latency	p95 latency under 300 ms, measured across 100 consecutive queries.
NFR-4	Index scale	Minimum 100,000 passages indexed; 500,000 passages earns bonus points.
NFR-5	Free-tier only	No paid API subscriptions. Groq free tier and HuggingFace are permitted.
NFR-6	Reproducibility	GitHub repo must include a README with setup steps; evaluators must be able to clone and run it.
NFR-7	Benchmarking report	PDF or Markdown file comparing Phase 1 versus Phase 2 RAGAS scores and latency logs.

06 SUCCESS METRICS

Criterion	What judges look for	Weight
Retrieval quality improvement	RAGAS score, Phase 2 above Phase 1 (target: Precision above 0.75, Recall above 0.70)	30%
System performance	p95 latency under 300 ms on 100 consecutive queries	20%
Architecture quality	Modular, clean, well-documented code with clear separation of concerns	20%
Hybrid search implementation	BM25 and dense vector correctly combined with a documented fusion method	20%
Demo and presentation	Clear query demo with a benchmarking report comparing Phase 1 and Phase 2	10%

07 CONSTRAINTS AND RESTRICTIONS

Mandatory Constraints

ID	Constraint	Detail
C-01	Free-tier tools only	No paid GPU clusters, no enterprise API subscriptions. Solutions requiring paid keys will be disqualified.
C-02	RAGAS evaluation mandatory	Both Phase 1 and Phase 2 RAGAS scores must be logged and submitted in the benchmarking report.
C-03	Hybrid search required for Phase 2	Dense-only retrieval in Phase 2 will not score on the Hybrid Search criterion.
C-04	Minimum dataset size	The index must contain at least 100,000 MS MARCO passages. Smaller indexes will not qualify for full scoring.
C-05	Latency benchmark required	100 consecutive queries must be run and p95 latency logged. Self-reported estimates are not accepted.
C-06	GitHub repo required	Evaluators must be able to clone and run your solution. No binary-only or demo-only submissions.
C-07	Phase 1 baseline required	A logged Phase 1 RAGAS baseline is mandatory to demonstrate the Phase 2 improvement.

08 DEMO REQUIREMENTS

Your 10-minute demo must show the following stages clearly. Judges will follow this checklist.

1. **Live query (dense).** type a natural language question. Show the top-5 retrieved passages with relevance scores from Phase 1 (dense-only).
2. **RAGAS baseline.** display Phase 1 RAGAS Context Precision and Context Recall scores from your logged evaluation run.
3. **Hybrid search in action.** run the same query with Phase 2 hybrid retrieval. Show the fused result set and compare passages to Phase 1.
4. **Phase 1 vs Phase 2 comparison.** show your RAGAS Phase 1 versus Phase 2 scores side by side. Highlight the improvement in precision and recall.
5. **Latency benchmark.** display the latency log from 100 consecutive queries. Show the p95 value and confirm it is under 300 ms.
6. **Metadata-filtered query.** demonstrate a query with a metadata filter applied. Show how filtering changes or improves the result set.

7. **GitHub repo and README.** open your GitHub repo. Show the README with setup instructions. Judges should be able to clone and run it without assistance.

09 BONUS OBJECTIVES

These are not required for evaluation but will earn additional points and demonstrate system maturity.

- **Reranker integration:** add a cross-encoder reranker after retrieval to further improve precision.
- **Larger index scale:** index 500,000+ passages (target: 1M).
- **Multi-vector retrieval:** late interaction models (ColBERT) or multi-vector passage representations.
- **Caching layer:** implement query result caching to reduce repeated-query latency.
- **LLM-generated answers:** pass retrieved context to an LLM and display the generated answer alongside passages.
- **Evaluation dashboard:** real-time RAGAS metric display in the query interface.

10 IMPACT

< 0.60 CONTEXT PRECISION, NAIVE RAG (BASELINE)	**> 500 ms** TYPICAL QUERY LATENCY
> 0.75 CONTEXT PRECISION, OPTIMISED RAG (TARGET)	**< 300 ms** QUERY LATENCY TARGET, p95

- Reduces LLM hallucinations in enterprise knowledge systems.
- Enables reliable AI assistants on private company data without expensive retraining.
- Demonstrates production-grade information retrieval engineering within consumer hardware constraints.
- Provides a reusable, open-source retrieval template for enterprise RAG deployments.

> *This challenge goes beyond retrieval accuracy. It explores how well-designed vector architectures can make AI systems trustworthy and production-ready.*

ADROSONIC®

THANK YOU

www.adrosonic.com