

# Technical Assignment

## Objective:

- Evaluate a candidate's depth and practical skills across:
- Retrieval-Augmented Generation (RAG) system design and robustness (conflict detection, hallucination control).
- Retrieval quality (vector-only vs. hybrid search) and LLM control (prompting, chain-of-thought control, response safety).
- Automation thinking and backend orchestration (no-code/low-code and workflow tools such as n8n, Airbyte, n8n-style agents).
- Scalable system & storage design (vector DB choices and trade-offs: Pinecone, Weaviate, FAISS, Chroma, etc.).
- Backend logic and workflow integration (end-to-end ingestion, embedding pipelines, APIs, rate limiting, monitoring).

## Important — Selection & Deadline:

- Choose exactly ONE assignment from Options A / B / C below.
- Deadline: Complete and submit within 48 hours from receipt.
- Late submissions will not be accepted unless prior written exception is granted by the hiring team.
- Follow the Mandatory Output Format exactly — deviations will be penalized.

## Dataset Flexibility (Applies to All Options)

You may choose or create your own dataset relevant to the selected option.

### Requirements:

- The dataset must be realistic and domain-relevant to the chosen scenario.

#### Minimum size:

- Option A: At least 10 heterogeneous documents with intentional conflicts.
- Option B: At least 50 structured/unstructured documents suitable for retrieval benchmarking.
- Option C: At least 100 documents for ingestion and indexing demonstration.

#### Clearly describe:

- Data source (public dataset, synthetic, or self-created).
- Document types (PDF, DOCX, TXT, logs, etc.).
- How conflicts, compliance gaps, or metadata structures were introduced (if synthetic).
- Include sample documents in the GitHub repository.
- Explain any preprocessing assumptions (cleaning, chunking strategy, metadata tagging).

## Deliverables (All Required)

- GitHub Repository — code, README, run instructions, and test data.

# Technical Assignment

- README — architecture diagram, technology choices, assumptions, and reproduction steps.
- Short Demo Video ( $\leq 5$  mins) — show working demo and design decisions.
- Single-Page Design Doc (PDF or MD) — scalability plan, bottlenecks, monitoring, and cost trade-offs.

## Option A — RAG Robustness & Conflict Detection (Document QA)

### Scenario

You are given a set of heterogeneous documents (reports, meeting notes, logs) about a hospital's performance. Some of them contain conflicting claims — e.g. one says "Patient satisfaction improved by 20%," while another says "Complaints increased by 15%."

### Tasks

- Build a retrieval + LLM pipeline to answer free-text queries (e.g., "Has patient satisfaction improved in Q1?").
- Detect and mark conflicting facts from retrieved documents.
- Provide source provenance (document ID, snippet, similarity score).
- Include confidence calibration and reasoning.
- Use a vector DB (Pinecone, Weaviate, FAISS, or Chroma) and explain why you chose it.
- Demonstrate with at least two queries showing conflicts and your system's detection mechanism.

### Example

Query: "How has patient satisfaction changed over Q1?" Retrieved Context:

Report\_A.pdf: "Overall patient satisfaction improved by 20% compared to last quarter."

Report\_B.pdf: "Emergency department complaints rose by 25% in Q1."

### Your Output Example:

#### Answer:

Mixed signals detected. While survey data suggests overall improvement, complaints in emergency care increased notably.

Conflicting Evidence:

- Report\_A.pdf → +20% satisfaction (Survey)
- Report\_B.pdf → +25% complaints (Emergency)

Confidence Level: Medium

#### Reasoning:

Conflicting reports from two different departments; overall improvement plausible but localized dissatisfaction observed.

## Option B — Hybrid vs Vector-only Retrieval Benchmark

### Scenario

## Technical Assignment

You are given a corpus of policy and operations manuals and must answer compliance-related questions (e.g., “What are the infection control compliance gaps?”).

### Tasks

- Implement two retrieval systems:
- Vector-only (semantic search)
- Hybrid (BM25/keyword + vector rerank)
- Use 8–10 benchmark queries and measure retrieval precision, recall@k, and latency.
- Create a comparison report showing retrieved results and downstream LLM responses.
- Explain trade-offs: index size, update latency, embedding refresh cost, etc.
- Describe your embedding pipeline and versioning strategy.

### Example

**Query:** “What are the infection control compliance gaps?” Vector-only Result:

Retrieved general compliance policy section (less specific). Hybrid Result:

Retrieved infection control section with “hand hygiene” and “equipment sterilization” clauses (more relevant).

### Your Output Example:

#### Answer:

Hybrid search retrieves more targeted compliance clauses on infection control compared to vector-only.

Precision: Hybrid (0.85) vs Vector (0.60)

Latency: Hybrid (190ms) vs Vector (120ms)

#### Conclusion:

Hybrid retrieval improves relevance slightly at the cost of higher latency.

Confidence Level: High

## Option C — Scalable Knowledge System + Backend Orchestration

### Scenario

Your current RAG prototype works with 5 documents. Now, design how to scale to 10,000+ documents and integrate backend workflows using automation tools.

### Tasks

- Design a scalable ingestion pipeline: File ingestion → parsing (PDF/DOCX/PPTX) → metadata → embeddings → vector DB ingestion.
- Propose a production-ready architecture including vector store, metadata DB, cache, LLM gateway, and monitoring.
- Build a small prototype (100 docs) to demonstrate ingestion and indexing.
- Design an automation workflow using n8n or Airbyte (e.g., Watch S3 → Parse → Embed → Store → Notify).
- Define monitoring (latency, error rate, embedding drift), cost optimization, and backup strategy.
- Present a scaling roadmap and explain bottlenecks.

### Example

# Technical Assignment

## Workflow Example (n8n):

[S3 Trigger] → [File Parser Node] → [Embedding Generator Node] → [Vector DB Ingestion Node] → [Slack Notification]

## Output Example:

### Answer:

Proposed architecture scales to 10K+ docs using asynchronous ingestion and distributed vector DB (Weaviate).

## Key Components:

- S3 for storage
- Lambda/n8n flow for embedding pipeline
- Weaviate cluster for retrieval
- Prometheus for monitoring latency/errors

Confidence Level: High

## Reasoning:

Design supports concurrent ingestion, fault tolerance, and observability.

## **Storage & Backend Requirements (Applies to All Options)**

- Must use an embeddings-based vector DB for semantic search; justify trade-offs.
- Show embedding versioning and schema migration handling.
- Describe how backend handles concurrency, rate limiting, and secret management.
- Provide a deployment YAML or pseudo-spec for vector DB + API + worker.
- Demonstrate one automation workflow (n8n / Airbyte / equivalent).

## **Prompting & LLM Control (Applies to All Options)**

- Include system and user prompts used for final answers.
- Implement guardrails (citations, output length, refusal cases).
- Demonstrate one iterative prompting or chain-of-thought refinement example.
- Discuss hallucination reduction (context window tuning, evidence filtering, temperature control).

## **Submission Checklist**

- GitHub repo (code + tests + Sample dataset)
- README (how to run + architecture + assumptions)
- Demo video ( $\leq 5$  minutes)
- Design doc (PDF or MD)
- Results file in Structured Output Format