**Assignment Instructions**

For every assignment:

1. **Generate Data** – Create your own dataset (≥300 records, mix of short/long texts, include Hindi/Hinglish).
2. **Save in Formats** – Store the same dataset in **CSV, TXT, DOCX, PDF** (optionally JSON).
3. **Read & Clean** – Build loaders for each format, clean text (dedupe, normalize, detect language, chunk into 250–400 tokens).
4. **Pipeline** – Write one script that: loads raw → cleans → chunks → saves unified clean.jsonl in data/processed/.
5. **Embed & Index** – Convert chunks into vectors with a multilingual model, store in **FAISS or Qdrant** with metadata.
6. **Evaluate** – Prepare 30+ queries with expected results, test retrieval, and report Recall@10/MRR@10.
7. **Deliverables** –Build your github repo and maintain your readme

**Each assignment must begin with a sample of processed data (3 records), vector count, and pipeline run confirmation.**

**Section A – Fundamentals (Q1–Q5)**

1. Install FAISS and Qdrant, create a vector DB, and store 100 text embeddings (using EURI API / OpenAI embeddings). Verify storage and retrieval.
2. Implement a similarity search in FAISS for a small dataset of 20 product descriptions. Show top-5 most similar products for a query.
3. Compare cosine similarity vs inner product in FAISS on the same dataset — which one retrieves more relevant results?
4. Insert the same dataset into Qdrant. Write queries to retrieve the nearest neighbors. Compare retrieval speed vs FAISS.
5. Perform a dimensionality reduction (PCA/TSNE) of your stored vectors and visualize clusters. Explain your observation.

**Section B – Applications (Q6–Q10)**

1. Build a small FAQ chatbot: store FAQs in Qdrant, embed queries, and return the closest FAQ answer.
2. Create a plagiarism checker: store 10 student essays in FAISS. For a new essay, check if similarity > threshold with existing ones.
3. Implement a news recommendation prototype: store 50 news articles; query for “AI breakthroughs” and compare nearest results.
4. Store resume embeddings and job descriptions. Write a function that returns the top-3 matching resumes for each job description.
5. Use a multilingual embedding model: store English documents, then query in Hindi. Compare recall vs query-translation approach.

**Section C – Advanced Challenges (Q11–Q15)**

1. Store embeddings of Python code snippets and retrieve code by natural language queries like “binary search function”.
2. Build a cross-modal search: store both image embeddings and text captions, then allow search by either text or image.
3. Integrate FAISS or Qdrant with LangChain: perform retrieval-augmented generation (RAG) from your stored documents.
4. Benchmark retrieval time for 10k vectors in FAISS vs Qdrant. Which scales better, and why?
5. Add metadata filters in Qdrant (e.g., category = “electronics”). Query with both semantic similarity + metadata filter.