ADB Project Planning

Project Requirement

● The database must have an indexing system that retrieve the top_k most similar rows to the given input
● The indexing system must be able to handle large datasets (up to 20 million).
● The system must response in a reasonable time

Evaluation Criteria

1. `Accuracy` (Recall): The indexing system must accurately retrieve the top 'k' results for a given query.
2. `Efficiency`: The indexing system must efficiently retrieve the results, with a reasonable query time and memory usage.
3. `Scalability`: The indexing system must be able to handle large datasets and scale appropriately.
4. `Documentation`: The design document must be well-written and provide sufficient detail for someone to understand the indexing system.

Deliverables

- [ ] The index files for four databases
  - [ ] 20M
  - [ ] 15M
  - [ ] 10M
  - [ ] 1M

Timeline

● Week 11: Submit final deliverables

- [ ] final version of the design document
- [ ] Implementation

Resources May Help

- Github Repo CMP 25
- IVF-PQ algorithm in details (very useful)
- IVF_FLAT
- IVF-PQ another article
- IVF_PQ implementation details

Tasks

- [ ] `ProductQuantizer` Class → `2 persons`

    - Implement subvector splitting (divide 70-dim vectors into subvectors)
    - K-means clustering for each subspace to learn codebooks
    - Encoding: map vectors to PQ codes
    - Decoding: reconstruct approximate vectors from codes
- Asymmetric distance computation (query vs PQ codes)

- [ ] `InvertedIndex` Class → `2 persons`
    - K-means clustering for coarse quantization (create partitions)
    - Build inverted lists (map cluster_id → list of vector_ids)
    - Implement partition assignment for new vectors
    - Handle index persistence (save/load inverted lists structure)
- [ ] Index Integration & Search → `2 persons`
    - [ ] Integrate PQ and IVF modules into `_build_index()`
    - [ ] Implement non-exhaustive search (probe n_probe nearest clusters)
    - [ ] Optimize the `retrieve()` method to use the index
    - [ ] Handle edge cases (empty clusters, tie-breaking)
- [ ] Index Persistence & Updates → `1 person`
    - Serialize/deserialize index to `index.dat`
    - Implement index loading in `__init__()`
    - Handle `insert_records()` - decide rebuild vs incremental update strategy
    - Memory-map index structures if needed for large datasets
- [ ] Testing & Benchmarking → `1 person`
    - Create test datasets with known nearest neighbors
    - Measure recall@k compared to brute force
    - Benchmark query latency and index build time
    - Test edge cases (small DB, duplicate vectors, etc.)

Task Distribution

Member			
Samy	ProductQuantizer	build_Index()	Insert Records
Abdullah	ProductQuantizer	retrieve()	Testing
Khiat	InvertedIndex	build_Index()	Testing
Sayed	InvertedIndex	retrieve()	Insert Records

Things we have to decide on

- [ ] IVF n_clusters:
    - **Trade-off**: Fewer clusters = more vectors per cluster = slower search. More clusters = risk of missing true neighbors if query's cluster assignment is wrong.

- **Theoretical optimum**: Balances partition search cost vs number of partitions to probe

- [ ] Number of PQ Subvectors (M)
  - **Constraint**: Must divide dimension evenly (your D=70)
  - **Trade-off**: More subvectors = better approximation BUT more memory and computation
  - **Typical values**: M = 8, 16, 32, 64
  - **Subvector dimension**: d_sub = D/M should be **≥ 4** (otherwise too little information per subvector)
- [ ] PQ Codebook Size (K per subspace)
  Scientific Basis:
  - Standard: **K=256 (8 bits)** per subvector
  - **Why 256?**:
    - 1 byte per subvector = memory efficient
    - 256 centroids provide good granularity for most data
    - Hardware-friendly (byte-aligned)
- [ ] Number of Probes (n_probe)
  Scientific Basis:
  - **Trade-off**: Search speed vs recall
  - More probes = check more clusters = find more true neighbors BUT slower
    Practical Guidelines:
    Based on research (Jégou et al., "Product Quantization for Nearest Neighbor Search"):
- [ ] K-means Parameters