



ADB Phase 1 Report

Seifeldin Sameh Dakroury
Amr Ashraf Ali
Abdallah Ahmed Mohamed
ElHussein Awad

1220246
1220309
1220251
1220226

1. Introduction

After evaluating several indexing strategies for large vector collections, our team decided to adopt **IVF-kMeans** as our indexing strategy. We selected this method because it offers an effective balance between retrieval accuracy, computational efficiency, and scalability. In particular, it enables us to search only a small portion of the dataset for each query, which is essential when working with millions of 64-dimensional embeddings.

2. System Structure

Our system consists of three main components:

1. **A binary file that stores all vectors**
Each vector occupies a fixed-size block, which allows direct access to any row without loading the full dataset.
2. **An IVF-kMeans index stored completely on disk**
This index includes the centroids produced during clustering and the inverted lists mapping each centroid to the vector identifiers assigned to it.
3. **A retrieval mechanism**
Given a query vector, the system determines the most relevant clusters and compares the query only with vectors in those clusters.

All components interact in a way that minimizes memory usage during queries and supports scaling up to the largest dataset sizes.

3. IVF-kMeans Index Design

3.1 Role of Clustering

IVF-kMeans begins by partitioning the vector space into a fixed number of regions, each represented by a centroid. During indexing, every vector is associated with its nearest centroid, which creates an inverted list for that cluster. This divides the full dataset into smaller groups that can be searched independently.

We use this structure because it allows retrieval to focus only on the clusters located near the query vector rather than processing the entire dataset.

3.2 Components of the Index

The index is composed of the following files:

- **Centroids file:**
Contains all cluster centers generated by k-means.
- **Inverted list files:**
One file per cluster storing the vector IDs assigned to that cluster.

This organization ensures that the system reads only the parts of the index needed for a given query.

4. Index Construction

4.1 Sampling Strategy

Since running k-means on millions of vectors is computationally heavy, we train the clustering model on a representative sample. We select a sufficiently large but manageable subset of vectors so that the centroids reflect the overall shape of the embedding distribution.

4.2 Training the k-Means Model

We train the model using mini-batch k-means, which efficiently handles large sample sizes. The number of clusters is chosen to balance inverted list size with the complexity of centroid comparisons. In practice, a few thousand clusters provide good performance for datasets in the 10–20 million range.

4.3 Assigning Vectors to Clusters

After computing the centroids, the full dataset is processed. For each vector:

1. It is accessed directly from the data file.
2. Its nearest centroid is identified.
3. Its identifier is appended to the appropriate cluster's inverted list file.

We store only the vector IDs in the inverted lists, as this keeps the index size manageable and avoids duplicating the embeddings.

4.4 Writing the Index to Disk

All centroids and inverted lists are saved in a directory dedicated to the index. This structure enables targeted reading during retrieval while remaining compact enough to stay within the allowed index size limits.

5. Retrieval Method

5.1 Query Preprocessing

When a query vector arrives, we normalize it and load the centroid matrix to identify which clusters are most relevant to search. The centroid file is sufficiently small that loading it introduces negligible overhead.

5.2 Selecting Clusters to Search

We compute the distances between the query and all centroids. Only a small number of the closest clusters are selected. Searching multiple clusters improves accuracy, while limiting the number keeps the time per query low.

5.3 Gathering Candidate Vectors

For each selected cluster, we read the corresponding inverted list file and fetch the actual vectors referenced by those IDs. Only these vectors which are typically a small fraction of the full dataset are evaluated for similarity to the query.

This drastically reduces the amount of work needed compared to checking all vectors in the dataset.

5.4 Ranking and Returning Results

We compute similarity scores between the query and all candidate vectors, rank them, and output the top-K. Although IVF-kMeans produces approximate results, probing multiple clusters and using moderate cluster granularity gives strong retrieval accuracy.

6. Reasons for Choosing IVF-kMeans

Our team selected IVF-kMeans specifically for the following reasons:

- 1. High practical efficiency in high-dimensional spaces**
Linear-space tree indexes deteriorate significantly as dimensionality increases. IVF-kMeans avoids this issue.
 - 2. Scalable to tens of millions of vectors**
The inverted lists keep retrieval time manageable.
 - 3. Compact index structure**
Storing only IDs, rather than full vectors, helps meet the index size constraints.
 - 4. Suitable for disk-based retrieval**
The index can be organized so that only small portions must be read for each query.
 - 5. Straightforward implementation in Python**
The approach works well with the allowed libraries and does not require specialized packages.
-

7. Design Trade-Offs and Parameter Decisions

Throughout our design, we made several parameter choices that present trade-offs:

- **Number of clusters:**
More clusters produce shorter inverted lists, improving query time but increasing index size and centroid comparison cost.
Fewer clusters reduce the index footprint but increase retrieval latency.
Value to be chosen has to balance these effects.
 - **Number of clusters probed per query:**
Probing more clusters improves recall but increases the number of candidate vectors.
We will select a moderate probing count to maintain reliability while keeping query times within limits.
 - **Sampling size for k-means:**
Larger samples yield better cluster quality but prolong index construction.
We chose a sample large enough to represent the global distribution without making clustering prohibitively slow.
 - **File organization:**
Using one file per inverted list simplifies selective loading but increases the total number of files to manage.
-

8. Conclusion

Our design is built around IVF-kMeans because it offers a practical and effective method for handling large-scale vector retrieval under limited memory and time budgets. By organizing vectors into clustered regions and restricting the search to a small set of inverted lists, the system supports fast approximate nearest-neighbor queries while maintaining strong retrieval quality. The structure is simple to implement, storage-efficient, and well-suited to our project's operational constraints.