ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Systems@ETH zürich

Master's Thesis Nr. 434

Systems Group, Department of Computer Science, ETH Zurich

Recommendations for Building IVF Indexes in Billion-scale ANNS Systems

by

Antonio Lopardo

Supervised by

Prof. Gustavo Alonso

October 2022–April 2023

D INFK

Contents

Abstract

Vector databases are designed to store unstructured data in semantically meaningful representations. Approximate nearest neighbor search (ANNS) is the primary query method for vector DBs, and the inverted file index (IVF) is among the most popular ANNS techniques. IVF indexes use clustering to dynamically reduce the search space based on the query. A high-quality IVF index minimizes the number of vectors that need to be scanned to reach high recall. The objective we set out to achieve with this thesis was to understand and characterize the effectiveness of different techniques and training procedures used to build performant IVF indexes that can hold billions of vectors. To achieve this, we compared a form of hierarchical clustering (HC) recently applied to building these indexes with more computationally intensive flat clustering (FC) techniques widely implemented in ANNS libraries. We experimented with training with cluster size penalties, with gradient descent, or using different centroids update frequencies. We also tested the importance of replication to avoid boundary issues in IVF indexes. Our results show that HC has competitive performance compared to flat clustering, despite its lower asymptotic complexity, and that it is also beneficial as an initialization procedure to seed flat clustering. Moreover, HC initialization speeds up convergence of flat clustering with k-means, and on some datasets, it improves recall@10 performance at convergence when compared to a clustering initialized with uniformly sampled points from the dataset. Ultimately, using the results of our experiments, we created a comprehensive set of recommendations that helps identify the most appropriate techniques to be applied when building IVF indexes by users with varying computational budgets.

Chapter 1

Introduction

Information retrieval has been a fundamental aspect of database systems since their inception [33]. While record keeping remains valuable, the growing volume and diversity of data stored in databases have increased the complexity of retrieving pertinent information, thus garnering more attention in research [25]. The transition from structured tabular data to unstructured data has been especially significant, as it has spurred the evolution and implementation of vector stores and databases (VectorDBs).

VectorDBs are designed to store and retrieve unstructured data, like images, text, or more complex entities, in semantically meaningful, multidimensional vector formats. Producing these vectors involves processing the unstructured data with models or functions that have been designed or trained for this purpose [37] [21] [1] [26] [6] [19]. Although a diverse range of methods exists, they tend to share two primary objectives. First, generating compressed representations of the unstructured data, although this is not always the case when dealing with text. Second, producing closer representations of data points that are more semantically similar to each other and more distant representations of data points that are less similar. In other words, these procedures aim to keep semantically similar data points close together and dissimilar data points far from each other. Due to this property, we can reduce information retrieval in VectorDBs to vector similarity search, which, in this context, simply entails calculating the distances between the query vector and all the vectors already in the DB, to find the query's nearest neighbors.

This formulation greatly simplifies information retrieval in applications that involve, for instance, natural language processing, where systems often use different variations of exact match search [8]. However, since the scale of the problem increases linearly with the number of vectors stored in the DB, forms of approximate nearest neighbor search (ANNS) are needed to query vector stores with billions of vectors.

All variations of ANNS share the same objective: to provide significant speedups over exact search methods while still delivering acceptable recall performance for the given application.

Our focus for this work was primarily on the Inverted File (IVF) index [3], a data structure that uses clustering in its initialization procedure and that helps to perform ANNS by reducing the number of unnecessary calculations needed to retrieve a query's approximate nearest neighbors.

The most widely used implementations of IVF indexes [17] use the k-means clustering objective [13] and Lloyd's algorithm [23], since it has linear complexity in the number of data points to cluster and can be easily parallelized. This strategy works well when all data fits in memory and the number of centroids is many orders of magnitude smaller than the number of data points. Yet, it can prove challenging to run this clustering procedure at the billion-scale if the number of centroids is only a few orders of magnitude smaller than the number of data points. This is partly due to the number of redundant distance calculations performed between points and centroids, that are too far away from each other to influence the centroids' location.

One way researchers have recently tried to avoid these unnecessary computations and limit the complexity of clustering in the IVF index-building process is by using a form of hierarchical clustering with k-means (HC) [5]. HC performs k-means clustering recursively on increasingly smaller partitions of the dataset, each time with a small k. The authors in [5] detail the architecture of an end-to-end system that achieves state-of-the-art performance in billion-scale approximate nearest neighbor search. Yet, they did not provide a detailed comparison of HC and standard (flat) k-means clustering (FC) to demonstrate the effectiveness of each method in various settings. A thorough evaluation of the differences between HC and flat k-means clustering would provide valuable insights for researchers and practitioners aiming to optimize approximate nearest-neighbor search algorithms. By understanding the strengths and limits of HC, it becomes easier to determine the most suitable clustering approach for any specific dataset and improve the overall efficiency of the IVF index-building process.

This work aims to provide a comprehensive study on how to construct high-quality IVF indexes to minimize I/O cost in large-scale ANNS.

To do so, first, we provide a detailed comparison between HC and standard (flat) k-means clustering.

Second, we evaluate using cluster size penalties while training to penalize clusters that become too big.

Third, we test the effectiveness of recently introduced replication techniques used to decide which vectors if any need to be stored in the multiple parts of the IVF index.

Fourth, we assess whether training a clustering with gradient descent is feasible or recommended at this scale.

Finally, we examine the advantages and disadvantages of more frequent centroids updates when training a (flat) k-means clustering.

All in an effort to produce a set of recommendations that users with different computational budgets can follow to maximize the effectiveness of their own IVF index-building process.

The experiments we carried out to directly compare HC and FC led us to conclude that:

- HC performs much better than other simple baseline clustering techniques and competitively but worse than FC.
- The cluster size distributions produced by hierarchical clustering were more balanced than those produced when training FC with Lloyd's algorithm.

Because of the more balanced cluster size distributions produced by HC, we also experimented with it as an initialization technique for FC's centroids. Our findings from these experiments were:

- HC initialization for FC is effective at producing cluster size distributions that are more balanced than other initialization techniques
- HC initialization leads to better optima at convergence for some of our datasets, but it is less effective on others

In trying to understand the extent to which HC initialization's benefits were due to its effects on the distribution of cluster sizes, we also experimented with training with cluster size penalties, concluding that:

- Cluster size penalties need to be tweaked to be effective, and using them increases the complexity of clustering
- HC initialization, when effective, is preferable to using cluster size penalties because it requires less configuration search

After testing gradient descent training for clustering along with more frequent updates to the centroids, we concluded that:

- GD training requires too much hyperparameter search to get competitive performance with FC
- In our setting, updating centroids more frequently than once per epoch leads to underutilized clusters

Finally, our replication experiments show that the replication procedure used by [5], can lead to redundant replication with limited effects on recall. Thus, the parameters of the procedure need to be tweaked and so, we recommend it only for users with the budget to test multiple configurations of the parameters.

Our final recommendation set for users with different computational and storage budgets is summarized in table 1.1, we will expand on each option in Chapter 5 after giving more details on the experiments we introduced above.

This recommendation set can be useful on its own. Still, it also features as part of a larger project that aims to design a state-of-the-art, distributed ANNS system with a focus on minimizing I/Os, effective load balancing, and adequate handling of hot queries.

Comp. Budget	Preliminaries	Clustering Method
Very Low	—	Hierarchical Clustering
Low	—	Flat Clustering with HC init
Medium	1M scale HC init	FC with HC init or random init
High	penalty value search replication config search	FC with HC init or with penalty and RNG replication

Table 1.1: Our experiments indicate that relevant budget tiers are
Very Low budgets that allow running one iteration of FC or less
Low budgets that have resources to run between 1 and 3 iterations of FC
Medium budgets that allow running more than 3 full iterations of FC
High budgets that allow running hyperparameter search on the exact cluster size penalty to use at the 1B scale

Chapter 2

Background & Related Work

2.1 Clustering Algorithms

We mentioned in Chapter 1 , that the main focus of this work is analyzing different techniques to build IVF indexes. IVF indexes are closely tied to clustering, and in this section we'll detail the most relevant clustering techniques for our use case, along with some background and related work on the topic.

2.1.1 Flat K-means Clustering

K-means clustering [13] is a clustering method designed to partition a dataset into k distinct groups or partitions. The objective is to create these groups in a way that minimizes the sum of squared distances between each data point and its closest centroid. The centroids are each partition's representative vector, the mean of the vectors in the partition. Although finding the optimal solution to this problem is NP-hard, approximate solutions that reach local optima have been developed, with Lloyd's algorithm [23] being one of the most widely used. Lloyd's algorithm is a form of iterative refinement where each iteration can be divided into two steps. The first step involves assigning data points to their nearest centroids, and the second step involves updating the centroids using the mean of the data points assigned to them. For a more detailed look at LLoyd's algorithm, see Algorithm 1.

2.1.2 Initializations

Lloyd's algorithm does not guarantee convergence to the global optimum. Its output is highly sentive to the initial configuration of the centroids. Consequently, many initialization strategies have been developed to improve the quality of the final clustering or speed up convergence. Among the simplest tech-

Algorithm 1 Lloyd's Algorithm (k-means)

Input: dataset to cluster $\mathbf{X}$, number of clusters $\mathbf{k}$, number of iterations $\mathbf{it}$
Output: centroids C, dataset partitions **partitions**

Initialization $C = \text{sample}(\mathbf{X}, \mathbf{k})$
for $i \in \{1, 2, \ldots, \mathbf{it}\}$ **do**
  partitions $= \text{assign}(\mathbf{X}, C)$ {Assign each point to the nearest centroid}
  for $j \in \{1, 2, \ldots, \mathbf{k}\}$ **do**
    $C_j = \text{mean}(\mathbf{partitions}[j])$ {Update centroids as the mean of assigned points}
  end for
end for

niques is the Forgy method [9] that selects k random data points from the dataset and utilizes them as the initial centroids for the clustering. The complexity of this procedure is no higher than the cost of random uniform sampling from an array. More complex techniques like the maximin method [18] and kmeans++ [2] exist. However, since they rely on iterative processes with extensive distance calculations for each iteration, they are too complex to be applied to clustering at the billion-scale.

2.1.3 Hierarchical Clustering with K-means

One of the significant drawbacks of flat k-means clustering, especially when the number of centroids is very high compared to the number of points to cluster, is the proportion of redundant distance calculations performed between points and centroids too far from each other ever to influence the positions of the centroids.

The principal, more efficient clustering technique we experimented with is a form of hierarchical clustering with k-means (HC, or simply hierarchical clustering in later sections), a variation of bisecting k-means. In this variation, the initial number of clusters is greater than two, and all existing clusters undergo further partitioning unless their size falls below a predefined threshold, see Figure 2.1 for a more in-depth explanation with a practical example and Algorithm 2 for the full algorithm. The authors of SPANN [5] used a variation of this clustering technique in their system. They also mention that it is significantly less computationally expensive than flat clustering with k-means. The complexity of k-means is $O(|\mathbf{X}| * N)$ where $|\mathbf{X}|$ is the total size of the dataset, and N is the number of clusters. Meanwhile, the complexity of the hierarchical clustering with k-means procedure we described above has a time complexity of $O\left(|\mathbf{X}| * k * \log_k(N)\right)$ where k is the number of clusters used at every level of the hierarchical clustering, and N is the total number of final clusters identified by the procedure.

Algorithm 2 Hierarchical K-means with Dynamic K

Input: dataset to cluster **X**, number of clusters **k**, number of iterations **it**, threshold **t**
Output: centroids C, dataset partitions **partitions**

Initialization: **partitions_queue**.add((mean(**X**), **X**))
$centroids\_cnt = 0$
while not **partitions_queue**.empty() **do**
  centroid, **partition** = **partitions_queue**.pop()
  if len(**partition**) > **t** **then**
    $\tilde{k} = \min(\mathbf{k}, \lceil \text{len}(\mathbf{partition})/\mathbf{t} \rceil)$
    C_{nested}, **nested_partitions** = Lloyd's Algorithm(**partition**, $\tilde{k}$, **it**)
    for $\tilde{c}$, $\tilde{p}$ in zip(C_{nested}, **nested_partitions**) **do**
      partitions_queue.add($\tilde{c}$, $\tilde{p}$)
    end for
  else
    $C[centroids\_cnt]$ = **centroid**
    centroids_cnt + +
  end if
end while
partitions = assign(**X**, C) {Assign each point to the nearest centroid}

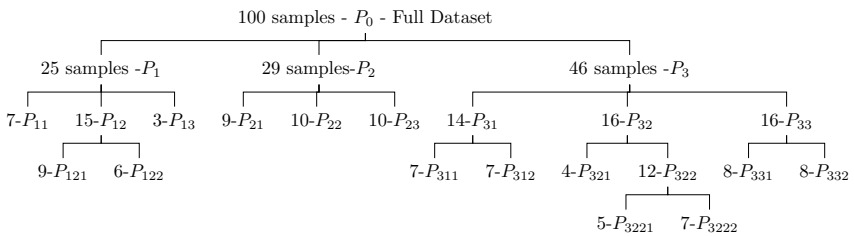


Figure 2.1: HC on a dataset with 100 samples, $k = 3$ and threshold set to 10. We can see how the first clustering procedure partitions the dataset in 3 partitions P_1, P_2 and P_3. Focusing on $P1$, the partition has 25 samples, more than the threshold, so it needs to be partitioned further. P_{11} and P_{13} are smaller than the threshold, they will not be partitioned again, but P_{12} will. However, since its number of samples is not higher than double the threshold, we will use $k = 2$ in this clustering procedure in order to avoid assigning too many centroids to a small partition. The resulting partitions are both smaller than the threshold size, and they become leaf nodes in our tree. The final results of the HC procedure is the set of centroids linked to each of the partitions at the leaf nodes, after running HC we only make use of the these centroids and discard the rest of the hierarchy.

2.1.4 Gradient Descent Training & Deep Clustering

Lloyd's algorithm is one of many ways to train a clustering that minimizes the square distance between points and their closest centroids. More general optimization techniques like stochastic gradient descent (SGD) can be used alongside an appropriate reconstruction loss for the same purpose. The equivalence of the k-means objective to forms of matrix factorization [4] opens even more optimization approaches. Consequently, some researchers have tried to apply SGD training directly to the problem with more [40] or less [31] sophisticated variations of the procedure.

Yet, much of the literature around SGD-trained clustering is related to deep clustering [11][32][10][27], a technique that involves using a single training procedure to learn both a meaningful semantic representation of the data and its k-means clustering jointly. This approach makes full use of SGD's increased flexibility. However, because of its complexity, it would be too computationally expensive if applied to the number of vectors in our scenario.

Thus, in our tests, we focused on the matrix factorization and reconstruction loss minimization formulations without delving into deep clustering to keep complexity down.

One notable, non-ANNS-related application of clustering in an SGD-trained neural net that stands out for its scale is ELIAS [12]. In it, the model learns a soft clustering of different labels, that is trained jointly with the rest of the architecture. The authors make significant use of sparsity to avoid unnecessary computations, and they report that their approach scales even to 3 million labels. Their Extreme Multi-label Classification task (XMC) is significantly more complex than ours. Still, our scale is also considerably bigger, so it is hard to apply their findings directly to billion-scale ANNS, but they are still worth mentioning.

2.2 Approximate Nearest Neighbors Search

As we mentioned in Chapter 1, numerous ANNS techniques have been developed, some based on hashing, others on clustering or building nearest neighbors graphs using the vectors in the DB. Yet, they all have the same aim, avoiding unnecessary distance calculations between any query vector and the vectors in the DB that are too far from it to be among its nearest neighbors. In this chapter, we will detail some of these techniques and provide more background on the design of billion-scale ANNS systems, outlining their goals and requirements and the solutions that have been developed to meet them.

2.2.1 Clustering-Based techniques

IVF The Inverted Files index (IVF index) [3] is a data structure that uses clustering for ANNS. More specifically, when the index is built, a clustering procedure is used to partition the vector space of the DB in disjointed regions or Voronoi cells. Each region has a representative centroid, typically the mean

of the points in each cell, that is used for ANNS. When querying the vector DB with the IVF index, we can use these representative centroids to restrict our search to the Voronoi cells whose centroids are closest to the query point. Looking at Figure 2.2 we can see how to cut down on unnecessary operations by not searching through regions with centroids far from the query. The number of regions in which to perform exact nearest neighbors search to achieve high recall depends on many factors. Such as the total number of cells or the number of features of the vectors in the index, that due to the curse of dimensionality, has substantial effects on the performance of IVF indexes. Regardless, the number of regions considered salient given a query is not a set construction parameter for the index and can be dynamically changed after building the index.

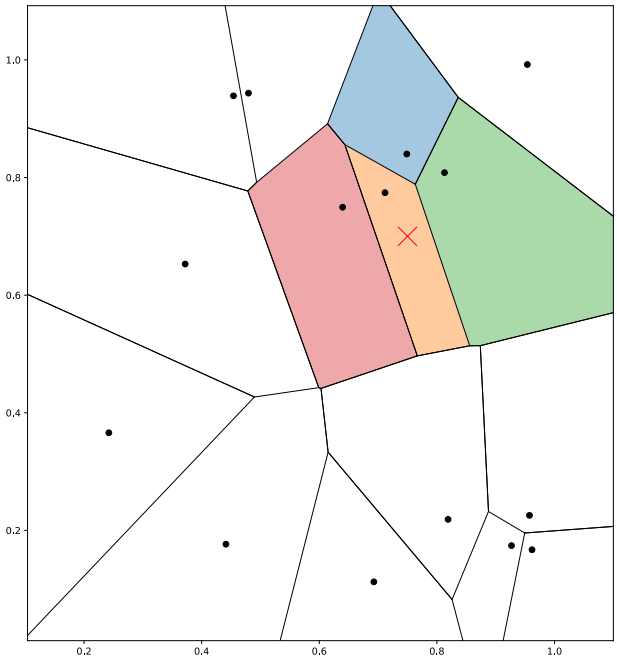


Figure 2.2: Plot of 2D space partitioned in Voronoi cells. When we receive a query, in the plot marked by a red x, we can limit ourselves to searching only through the cells where the centroids, black dots in the plot, are closest to the query.

The most widely used implementations of the IVF index-building process use the k-means objective for clustering trained with Lloyd's algorithm since

it has linear complexity in the number of data points to cluster and can be easily parallelized. This strategy works well in practice when all data fits in memory, and the number of IVF cells is many orders of magnitude smaller than the number of data points. However, as we mentioned in chapter 1, other forms of clustering have been applied to this problem at the billion-scale, for configurations where the number of clusters is only one or two orders of magnitude smaller than the number of data points, more on this in Section 2.3.1.

2.2.2 Other techniques

Graph-based ANNS The most significant Graph-based ANNS technique at the billion-scale is the HNSW graph index [24]. The main idea behind this index is to create a multi-layered graph structure that enables fast navigation to the nearest neighbors of a query point. Each graph layer is a subgraph containing a subset of the data points in the vector DB, with the top layer containing the fewest points. The layers are created in such a way as to ensure that each layer above has fewer points than the layer below it.

During the search process, the algorithm starts at the top layer and navigates toward the query point's nearest neighbor using greedy search. Once it reaches the closest point in the graph of that layer, it moves to a lower layer. At the lowest layer, it performs other forms of approximate nearest neighbor search on the local neighborhood of the query point.

The main reason why we focused on the IVF index over the HNSW's index is that the iterative building procedure of the graph is hard to parallelize and has complexity of $O(|\mathbf{X}| \log(|\mathbf{X}|))$. Furthermore, once the graph index is built, it is hard to distribute across machines in a distributed setting.

Locality Sensitive Hashing Locality Sensitive Hashing (LSH) [36] is a family of approximate nearest neighbor search techniques for high-dimensional spaces. The main idea behind LSH is to map similar data points to the same or nearby "buckets" in a lower-dimensional space using a family of hash functions, making it easier to search for nearest neighbors. LSH can, in short, be used as a form of dimensionality reduction where, in the lower dimensional representation of a point, each dimension corresponds to the bucket assignment of the point using a hashing function that hashes similar inputs to the same bucket.

Despite the fact that LSH is fast and efficient, its recall performance is not competitive with other ANNS techniques at the billion-scale. That's why we focused on IVF indexes over indexes that use LSH in this work.

Product Quantization Product quantization (PQ) [15] is a clustering-based dimensionality reduction technique. When using product quantization, instead of running a clustering procedure on the full high dimensional space of the vectors, we split the space into lower dimensional subspaces and run clustering in those subspaces. Typically, the clustering procedures are performed on subsets of vectors' dimensions, resulting in lower dimensional representations of the

subsets that, when concatenated, form the lower dimensional representations of the initial vectors. For example, a 128-sized vector space could be split into 8 subspaces with 16 dimensions. We can perform k-means clustering on each subspaces with k set to 256, resulting in an 8 bit or 1Byte representation of the 16 dimensions in each subspace, leading to an 8 byte representation for the initial vectors. Like other forms of quantization, it is necessary to keep track of the codebook that allows us to go from the quantized representation to the original one. In this case, the codebook contains the centroids from the clustering procedures run on each subspace.

PQ can be very useful since it reduces both the computational complexity of search and the storage requirements of ANNS systems. It is often used in conjunction with IVF indexes to increase efficiency further.

2.3 Related billion-scale ANNS systems

Many of the techniques we just described have been designed to run fully in memory. This limits their direct application to billion-scale datasets, where more complex hybrid ANNS systems that use disks to store data are needed. Multiple designs have been developed for these systems, using graph-based and clustering-based techniques.

2.3.1 SPANN

The IVF-based approach introduced in [5] is highly suited to ANNS at the billion scale. It uses hierarchical clustering with k-means as an efficient alternative to flat k-means clustering in order to reduce the complexity of the index-building procedure. After the index is built, all posting lists are stored on disk, and all centroids are kept in memory. With the centroids in memory, SPANN also uses a secondary index to perform ANNS over them too, speeding up the search procedure further.

SPANN's designers attempted to address multiple challenges with their new architecture, such as reducing slow I/O operations on vectors stored on SSDs and limiting boundary issues for points with similar distances to multiple centroids.

They minimize the number of read operations from disk by building an IVF index with a very high number of Voronoi regions. In their 1M-scale experiments, they use as many as 160 thousand centroids. The posting lists for such small clusters occupy only 1-2 SSD pages, limiting the number of unnecessary vectors retrieved during search. Running flat k-means clustering at the billion-scale with this many clusters is very expensive and involves many redundant distance calculations. To avoid them, SPANN uses a variation of HC. Nonetheless, the size of the clusters leads many points to be close to multiple centroids. To deal with these vectors at the boundaries of different regions, SPANN's authors use a specific replication strategy that allows the same vector to appear in multiple posting lists. The strategy is based on building a relative neighborhood graph [14] around each point and considers not only the relative positions

of data points to centroids but also the centroid-to-centroid distances to avoid unnecessary replications.

2.3.2 Other Approaches

Another billion-scale ANNS system is HM-ANN [30], which uses a variation of HNSW leveraging Intel's Optane persistent memory [39]. The authors do away with the previous state-of-the-art quantization techniques and instead optimize their HNSW implementation to utilize their two-tier Heterogeneous Memory efficiently. In designing our system, however, we wanted to avoid making use of features of specific proprietary hardware, so we focused on more general architectures.

Disk-ANN [34] proposes an alternative design that instead uses only DRAM and cheaper SSDs. Its architecture is still based on HNSW, but it manages to keep in check the size of the in-memory graph index by using product quantization. Full-size versions of the vectors are kept on disk and used to refine the approximate nearest neighbor search at the lowest layer of the graph index. But as we mentioned when introducing HNSW indexes, they tend to be difficult to distribute over multiple machines and build efficiently at these scales. Moreover, despite the compression provided by PQ, keeping some form of all the data in memory still limits the scalability of this approach.

Chapter 3

Retrieval System Proposed in the Wider Project

3.1 Requirements and Goals

In designing our billion-scale, distributed ANNS system, we embraced SPANN's IVF approach to minimizing I/Os but expanded the architecture to a distributed scenario with a focus on load balancing and effective handling of hot queries.

3.1.1 Minimizing I/Os

For the scale of our system, we target 10 billion to 100 billion vectors. Each vector with 128∼1024 dimensions, stored as float32, thus the size per vector is 0.5∼4 KB. The size of the entire dataset will be 5∼400 TBs. As we mentioned before, scaling the number of vectors in the system beyond 10^{10} makes fully in-memory implementations infeasible and distributed solutions, despite the communication and coordination overhead, more viable. At this size, the vast majority of the dataset will be only on disks (SSDs) at any one time. In this scenario, minimizing the I/Os of the system is one of the keys to achieving optimal performance.

Build a Fine-grained Vector Index To minimize I/Os, our system should do exact nearest neighbors search among as few vectors as possible due to the high costs of reading from SSDs. To this end, similarly to SPANN, we need:

- A clustering algorithm that can partition the vector space into many fine-grained blocks. The degree of granularity is still a research question, but they can be as small as an SSD page size (4∼16 in practice).
- A balanced size distribution across the clusters, such that the query latency will not be dominated by scanning several huge blocks.

Global Clustering rather than Per-partition Clustering To improve the quality of the index, we use the entire dataset for clustering, rather than

partition the dataset and constructing independent indexes per partition like SPANN does.

Fast Vector Index Traversal A fine-grained vector index minimizes the amount of disk I/Os involved in a search. However, such fine granularity results in many centroid vectors being indexed, thus the index traversal can be a potential bottleneck. It is necessary then to speed up index traversal with other fully in memory ANNS techniques, such as HNSW or variations of it.

3.1.2 Balancing Workloads

Building an ANN system on a single machine is different from building a distributed ANN system, which typically involves dozens of machines or more. In the distributed case, the slowest machines in the entire system can disproportionately affect the system's overall latency. Thus, balancing workloads between machines and minimizing the run time variance between them is crucial for performance.

We thus aim to balance the workload in our system by adequately distributing the posting lists from our fine-grained index to all of our machines. One approach is to make use of a sample query set as a guide. Given the sample set, we can score each cluster by its access frequency, and distribute clusters of similar scores to different machines. This minimizes the chance that one machine holds many hot data points and becomes the bottleneck of the entire system. Another approach could entail distributing clusters according to their spatial locality. If two clusters are next to each other, then they should be dispatched to different machines. This approach can reduce the chances that a single machine will become a bottleneck, because any query will only visit nearby clusters, and if they are distributed over multiple machines we can load balance effectively regardless of the query distribution.

3.1.3 Handling Hot Queries

Some queries are more frequent than others, thus the related database vectors are retrieved more often than others. As a result, some special treatment to accelerate those hot queries should improve overall system performance.

We can build an in-memory buffer manager for each storage node, such that the more frequently-accessed clusters can be retrieved from memory rather than from disks. The policy of the buffer manager can be flexible. One approach is to rely on a statistical analysis of the query set, and determine a list of frequently-accessed clusters to be held in memory. Another approach is to implement a dynamic buffer manager, and adopt policies such as LRU which can handle query distribution shifts at run time. An interesting adaptation to LRU is to consider not only each posting list's popularity, but also its size.

3.2 Rationale for the Thesis

The objective we set out to achieve with this thesis was to understand and characterize the effectiveness of different training procedures to build performant inverted file indexes that hold billions of points. As we detailed in subsection 2.2.1, clustering is at the core of IVF indexes, it thus became our primary focus for this work. The result is a set of recommendations of different techniques and clustering procedures aimed at users with varied computational budgets.

Chapter 4

Clustering Analysis

4.1 Settings, datasets and methodology

In this section, we will outline the general design decisions behind our experiments, including the datasets we used to evaluate the indexes built using different clustering techniques and the methodology for evaluation.

4.1.1 Datasets

Datasets	Data point type	Dim	Number of points
Deep1B	image	96	10^9
Sift1B	image	128	10^9
Sbert10M	sentence	384	10^7
Glove1M	word	100	1183514

Table 4.1: Summary table for our datasets.

Deep1B Deep1B [38] consists of image embeddings extracted from the last fully-connected layer of the GoogLeNet model [35]. This dataset's vectors have 96 features and the full dataset contains one billion vectors

Sift1B Sift1B [16] is a large dataset of scale and rotation invariant representations of images. This dataset's vectors have 128 features, and the full dataset contains one billion vectors.

GloVe-100-angular Glove-100-angular is a collection of words encoded with a GloVe model [28] designed to produce word embeddings. This dataset's vectors

have 100 features and the full dataset contains 1,183,514 vectors. We will refer to this dataset as **Glove1M**.

Sbert10M Sbert10M is a collection of sentences encoded with a small Sentence-Bert model [29]. This dataset's vectors have 384 features, and we tested indexes trained with up to 10 million points for this dataset.

We compiled this dataset ourselves, using sentences from Common Crawl [7] and a widely available Sbert model. The dataset contains significantly more duplicates than the other datasets, but it provides a real world use-case for ANNS systems that goes beyond benchmarks, so it's worth including.

4.1.2 Methodology

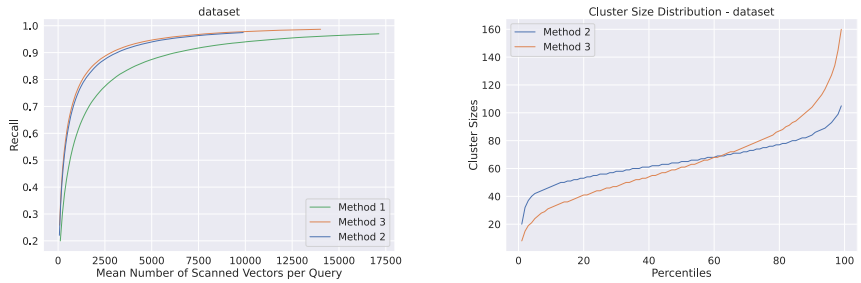


Figure 4.1: Prototypes of the two main plots we will use throughout our experiments

When evaluating an end-to-end ANNS system, such as the one described in Chapter 3, the primary performance metric is the average latency needed to achieve a certain level of recall. This average latency, however, is a measurement intrinsically tied to specific hardware and unlikely to be of much use when running on entirely different machines. So, as a proxy for latency, we will use the average number of vectors scanned to achieve a certain recall over the query set. Moreover, since our focus for this work was primarily on IVF indexes, and their performance is highly sensitive to the number of posting lists, we have decided to compare only indexes with the same number of them. Because of this, our latency measure will not take into account the cost of identifying the relevant Voronoi regions in the search procedure but only count the number of vectors retrieved from the selected posting lists.

Therefore, one of the two main plots we will use throughout this work to compare different index-building procedures is a recall to average number of vectors scanned plot, as in the first panel of Figure 4.1. When looking at this plot, we will highlight the recall@10 performance of indexes trained with different clustering procedures when scanning the same average number of vectors. Additionally, since one of the design priorities we would like to achieve is a

balanced distribution of posting lists sizes, in order to avoid big clusters that might prove to be significant bottlenecks, we will also include plots to show the distributions of posting lists sizes for each index building procedure, as in the second panel of Figure 4.1.

4.2 Flat vs Hierarchical Clustering with K-means Objective

4.2.1 Motivations and Setup

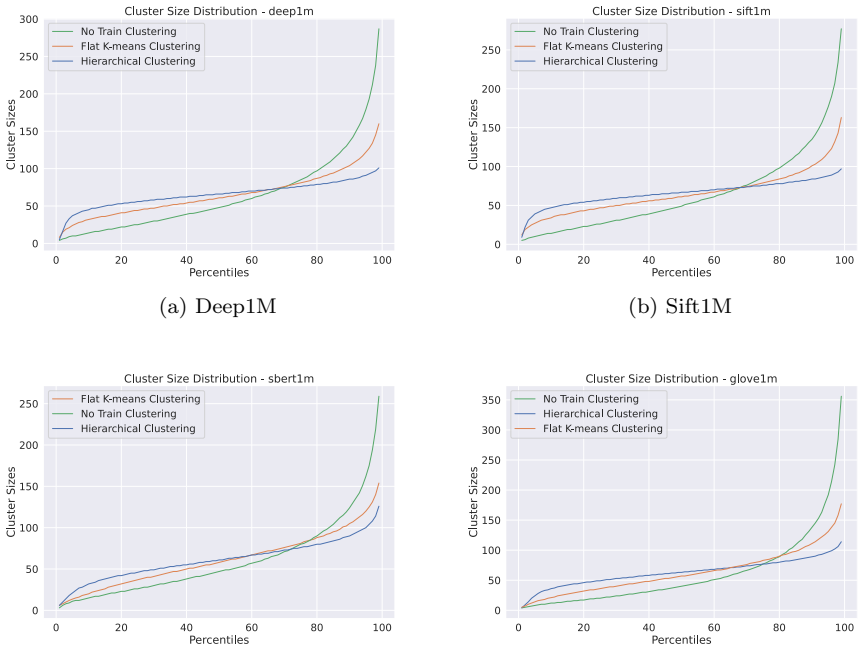
The hierarchical clustering procedure described in subsection 2.1.3 has only recently been applied to building IVF indexes. Consequently, we were curious to compare it against k-means clustering, the standard procedure currently employed to build IVF indexes in many widely used implementations. To give a more complete picture of HC's recall@10 performance, we also compared it to a clustering where the centroids are simply points sampled uniformly from the dataset we want to cluster without any training. We will refer to this clustering as "No Train Clustering"(NTC) since, after initialization, the centroids are never updated. The computational complexities of the clustering procedures are summarized in Table 4.2. We consider the No Train Clustering procedure to be an efficient baseline, while flat k-means clustering trained with Lloyd's algorithm should, given an adequate initialization, be the performance upper bound in this comparison due to the design of IVF indexes. To compare the three clustering procedures fairly, we first ran HC, noted the number of clusters it produced, and used the same number for the other two procedures. This approach aligns the cost of the initial distance calculations needed to select the relevant Voronoi regions. Throughout our experiments at all scales, we set HC's threshold size to 100. Settings the threshold to this order of magnitude allowed us to run the 1B-scale experiments with FC that we will detail in Section 4.3.

Complexity of Clustering Methods		
No Train Clustering	Hierarchical Clustering	Flat Clustering
$O(\|\mathbf{X}\|)$	$O\left(\|\mathbf{X}\| * k * \log_k(N)\right)$	$O(\|\mathbf{X}\| * N)$

Table 4.2: Computational complexities of the clustering procedures we compare in this section. $|\mathbf{X}|$ is the size of the dataset to cluster, k the maximum number of clusters used at each level of HC and N is the number of clusters used in flat k-means clustering, or the total clusters at the end of the HC procedure.

4.2.2 Results & Discussion

Hierarchical clustering performs much better than "no train clustering" and competitively with flat k-means clustering. Table 4.3 shows



(a) Deep1M (b) Sift1M

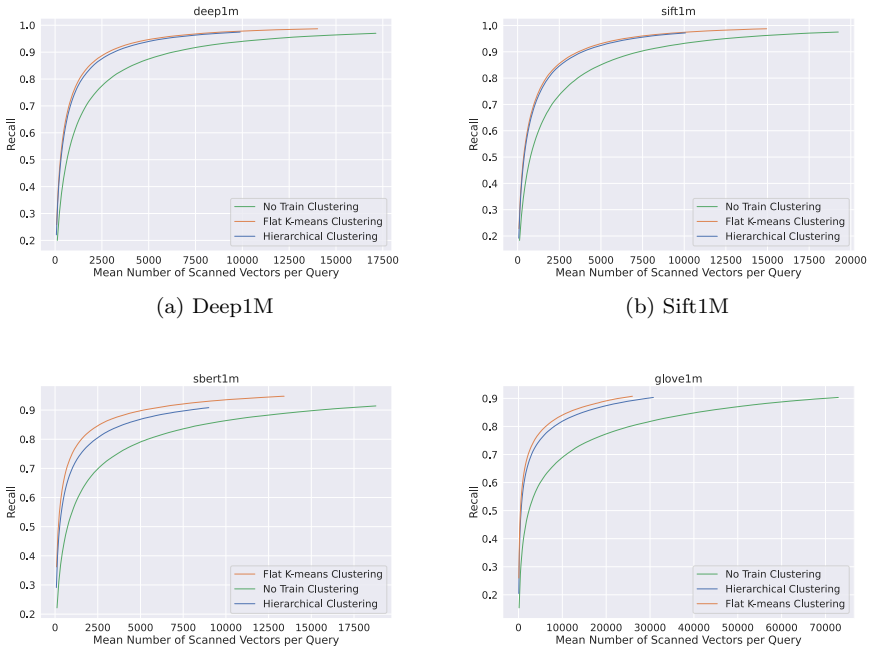
(c) Sbert1M (d) Glove1M

Figure 4.2: Plots of the distribution of cluster sizes produced for each dataset by each training procedure

that hierarchical clustering performs much better than "No Train Clustering," cutting down the cost of achieving 90% recall@10 by almost half in all of our datasets. As we expected, flat clustering trained with Lloyd's algorithm was the best-performing clustering procedure. Still, the gap between hierarchical and flat clustering was much smaller than between NTC and HC.

The plots in Fig 4.3 show the relative performance of the three procedures over a wider range of scanned vectors, highlighting how much closer HC is to flat clustering compared to NTC. In Table 4.4, we quantify the gaps by looking at how much lower the recall@10 performance of HC and NTC is when scanning the same amount of vectors needed to reach 90% recall@10 for the best-performing procedure, flat clustering. The gap between HC and flat clustering is between 1% and 3%, while the gap between NTC and flat clustering is much wider at around 10-11% for all of our datasets.

The cluster size distributions produced by hierarchical clustering were more balanced than those produced when training FC with Lloyd's algorithm. Regardless of the recall@10 performance of the three clustering procedures, we see in Fig 4.2 that they also differ in the distribution of the sizes



(a) Deep1M (b) Sift1M

(c) Sbert1M (d) Glove1M

Figure 4.3: Plots of the experiments comparing the three clustering procedures' recall@10 performance when scanning different numbers of vectors. To look at the specific number of vectors scanned to achieve 90% recall@10 see table 4.3.

of clusters they produce. NTC produces the most unbalanced distribution. The distribution of cluster sizes produced by HC is significantly more balanced since HC avoids very large clusters by design, and only a small proportion of its clusters are small. Flat clustering following the k-means objective results in a distribution of cluster sizes that is less balanced than those produced by HC. The differences in the cluster size distribution of the three procedures are significant since large clusters can be bottlenecks that force us to scan vectors that are not always relevant. Having similarly sized postings lists also facilitates load balancing in more complex architectures.

Datasets	Clustering Methods		
	No Train Clustering	Hierarchical Clustering	Flat Clustering
Deep1M	6284 (100%)	3142 (50.0%)	**2831** (45.1%)
Sift1M	7278 (100%)	3894 (53.5%)	**3643** (50.1%)
Sbert1M	15434 (100%)	7874 (51.0%)	**5175** (33.5%)
Glove1M	69900 (100%)	29238 (41.8%)	**22844** (32.7%)

Table 4.3: The table shows the number of vectors that need to be scanned to achieve 90% recall@10 on the datasets' query sets, lower is better. To compare the indexes fairly, we first run hierarchical clustering, and used for flat clustering the same number of clusters produced by HC.
When running HCFor all datasets but Sbert1M we set $k = 32$, for Sbert1M is set to 256.
When training 12 different runs with different seeding of FC on Deep1M the empirical standard deviation for the metric shown here is 34.19.
When doing the same on Sift1M the empirical standard deviation is 29.73.
When doing the same on Sbert1M the empirical standard deviation is 134.30.
When doing the same on Glove1M the empirical standard deviation is 246.37.

Datasets	Clustering Methods		
	No Train Clustering	Hierarchical Clustering	Flat Clustering
Deep1M	79.55%	88.92%	90%
Sift1M	80.21%	89.28%	90%
Sbert1M	79.45%	87.11%	90%
Glove1M	78.83%	88.33%	90%

Table 4.4: The table shows the recall@10 performance reached by each procedure when scanning the amount of vectors needed to reach 90% recall@10 for the best performing procedure.

4.3 HC Initialization and Speed of Convergence

4.3.1 Motivations and Setting

In section 4.2, we showed hierarchical clustering's significantly higher recall@10 performance when compared to a simpler baseline technique like NTC. Yet, the indexes built using flat k-means clustering still performed best. Given HC's significantly lower computational complexity however, in this section we also tested whether it would make for an effective centroids-initialization procedure to apply to FC. Our objective in this set of experiments was to see if initializing

the centroids of a flat k-means clustering procedure with the centroids derived from HC would lead to faster convergence for FC. The settings we used for these experiments don't deviate significantly from those used in section 4.2. We built indexes using flat k-means clustering trained for 5, 10 or 50 iterations and with centroids initialized either with points sampled randomly from the dataset or with centroids produced with HC.

4.3.2 Results & Discussion

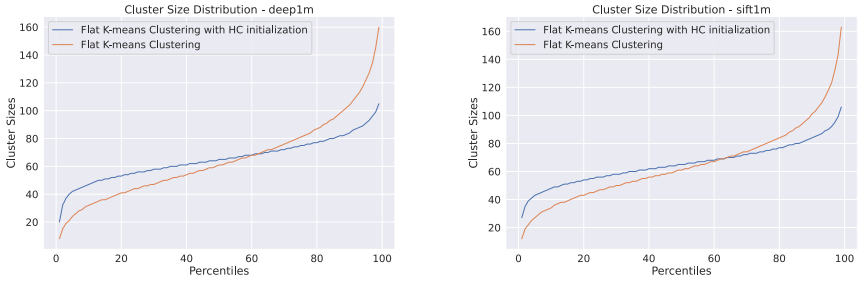
Datasets	**Clustering Methods**		
		Flat Clustering	
	No Train Clustering	HC Initialization	Random Initialization
		After 5 iterations	
Deep1M	6284 (100%)	2650 (42.2%)	3111 (49.5%)
Sift1M	7278 (100%)	3282 (45.1%)	3941 (54.1%)
Sbert1M	15434 (100%)	5949 (38.5%)	5669 (36.7%)
Glove1M	69900 (100%)	23256 (33.3%)	24187 (34.6%)
		After 10 iterations	
Deep1M	6284 (100%)	2610 (41.5%)	2934 (46.7%)
Sift1M	7278 (100%)	3208 (44.01%)	3760 (51.7%)
Sbert1M	15434 (100%)	5719 (37.1%)	5345 (34.6%)
Glove1M	69900 (100%)	22578 (32.3%)	23724 (33.9%)
		After 50 iterations	
Deep1M	6284 (100%)	**2568** (40.9%)	2831 (45.1%)
Sift1M	7278 (100%)	**3174** (43.6%)	3643 (50.1%)
Sbert1M	15434 (100%)	5638 (36.5%)	**5175** (33.5%)
Glove1M	69900 (100%)	**22196** (31.8%)	22844 (32.7%)

Table 4.5: The table shows the number of vectors that need to be scanned to achieve 90% recall@10 on the datasets' query sets, lower is better. To compare the indexes fairly, we first run hierarchical clustering, and used for flat clustering the same number of clusters produced by HC.

HC initialization leads to better optima at convergence for Deep1M and Sift1M, but it is less effective on other datasets. Table 4.5 shows that running more than 5 iterations of Lloyd's algorithm provides some gains in recall@10 performance, but that at least for Deep1M and Sift1M HC initialization can lead to faster convergence and better performance at convergence. The results for Glove1M and Sbert1M however don't show a significant effect and in

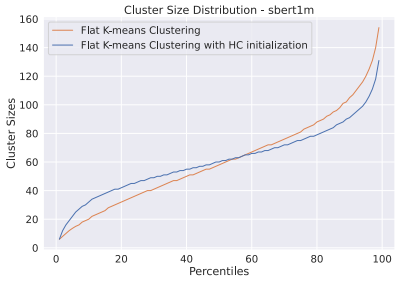
Nr. Iters	Methods	
	Flat K-means Clustering with HC init	Flat K-means Clustering
1	**6683**	7918
2	**6284**	6476
3	6116	**5988**
4	6026	**5759**

Table 4.6: The table shows the number of vectors that need to be scanned to achieve 90% recall@10 on the Sbert query sets, lower is better. We trained for different amounts of iterations

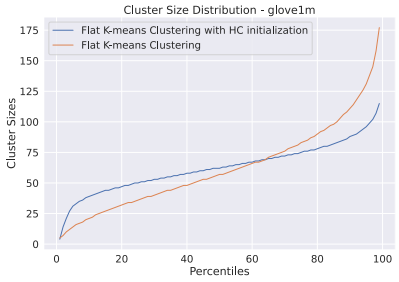


(a) Deep1M

(b) Sift1M



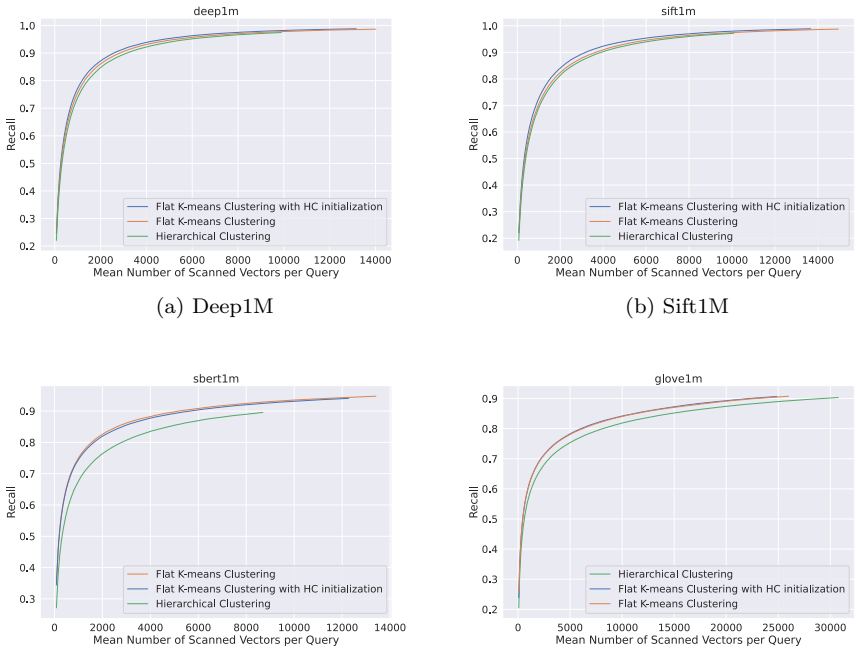
(c) Sbert1M



(d) Glove1M

Figure 4.4: Plots of the distribution of cluster sizes produced for each dataset by FC training with each initialization procedure

the case of Sbert HC initialization seems to slightly hurt recall@10 performance at convergence. For Sbert, HC initialization is better than random initialization only for a few iterations, as is visible in Table 4.6. The gap between random initialization and HC initialization is comparable to the gaps shown in Table 4.3 between HC and FC, for the datasets where HC initialization is effective.



(a) Deep1M

(b) Sift1M

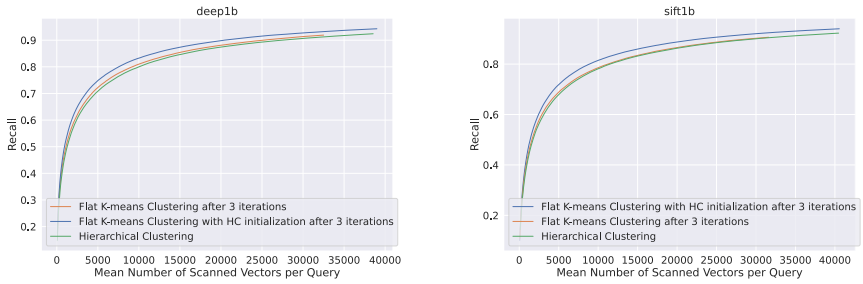
(c) Sbert1M

(d) Glove1M

Figure 4.5: Plots of the experiments comparing HC with flat k-means clustering initialized with randomly sampled points from the dataset or with the centroids produced by HC. To look at the specific number of vectors scanned to achieve 90% recall@10 see table 4.5.

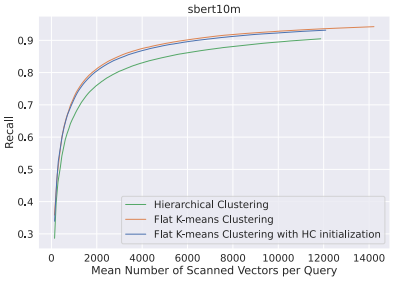
HC initialization effectiveness can be tested for using smaller scale experiments. We also see the same effects of HC initialization at different scales. For Deep1B and Sift1B, HC initialization of the flat clustering procedure results in better performing indexes while our tests on Sbert10M show that HC initialization is not effective on that dataset, even at the higher scale, see Figure 4.6.

HC initialization for FC is effective at producing cluster size distributions that are more balanced than the alternative initialization technique. The selective effectiveness of HC initialization seems to suggest that there are datasets for which it is suited and others for which it is not helpful, at least as tested here. It's also worth pointing out the fact that the datasets that benefitted most from HC initialization were also the ones where HC by itself performed very nearly as well as flat clustering already, as shown in Table 4.3. Sbert1M, instead, was the dataset where HC performed the worst relative to flat clustering. Part of the issue might be related to HC's bias towards balanced



(a) Deep1B

(b) Sift1B



(c) Sbert10M

Figure 4.6: Plots of the bigger scale experiments comparing HC with flat k-means clustering initialized with randomly sampled points from the dataset or with the centroids produced by HC. We can see from these plots that HC initialization is effective on Deep1B and Sift1B but not on Sbert, consistently with our smaller scale experiments.

clusters. This tendency is carried over to the flat clustering initialized with centroids from HC, as shown in Figure 4.4. The bias carried over from HC appears to help in Deep1B and Sift1B, but it has a slightly adverse effect on Sbert and only a small effect on Glove1M. Balancing the cluster size distribution is still a positive, as we discussed in Section 3.

The quick convergence speed of the clustering procedure is an incentive to find different ways to achieve better optima. Another interesting aspect to note is just how quickly the training converges, when initializing with sampled vectors, but especially with HC initialization. The improvements gained by training for longer than 5 iterations are marginal compared to ones provided by training with hierarchical clustering initialization. This highlights the value of alternative methods that go beyond Lloyd's algorithm and simpler initializations to achieve better performance with less computational cost. In table 4.5, for both Sift1M and Deep1M the performance of the indexes initialized with HC and trained with Lloyd's algorithm for only 5 iterations exceeds

that of the randomly initialized indexes trained for 50 iterations. It goes to show that, in this setting, exploring different training techniques often is preferable to simply running more and more iterations of Lloyd's algorithm. Especially since, at the billion-scale, for vectors with around 100 dimensions, one iteration can take close to a day, even when training on a multi-GPU setup with 4 NVIDIA RTX3090s.

4.4 Cluster Size Penalties

4.4.1 Motivation and Setting

To better understand the mechanisms behind why HC initialization is effective on at least some of our datasets, we focused on its effects on the distribution of posting list sizes. More specifically, we tried to replicate those effects by penalizing during training clusters that are big. As shown in [22], one straightforward way to involve cluster sizes in the point-to-centroid distance calculations is by adding a term to the distance metric that is proportional to the cluster sizes, see Equation 4.7 for an example with the L^2 distance. Using cluster sizes as part of the distance metric changes Lloyd's algorithm by adding another assignment procedure dedicated to measuring the cluster sizes, see Algorithm3 for more details. Making centroids from big clusters effectively more distant to any point is effective at producing more balanced cluster size distributions, but only with the right value of the λ. Therefore, for each dataset, we run multiple experiments with different values of λ, and we will report later the results for the best-performing penalties.

4.4.2 Results & Discussion

Cluster size penalties need to be tweaked to be effective, and it increases the complexity of clustering. As we expected when performing extensive hyperparameter search on the penalty value, it is possible to find a λ that is effective at producing more balanced cluster size distributions compared to not using any penalties at all, see 4.8. Table 4.7 shows that these more balanced distributions seem to correlate with improved performance for the indexes built on Sift1M and Deep1M. Yet, the effects on Glove1M and Sbert1M are more limited, just like what we saw for HC initialization. On Sbert1M specifically, however, the best performing λ was able to produce a more balanced cluster size distribution and not negatively affect recall. This result leads us to con-

$$d_\lambda(x, c_i) = \|x - c_i\|_2^2 + \lambda s_i$$

Figure 4.7: Modified L^2 distance metric when training K-means clustering objective with cluster size penalties. With s_i being the cluster size of the cluster with centroids c_i

Algorithm 3 Lloyd's Algorithm with penalties (k-means)

Input: dataset to cluster $\mathbf{X}$, number of clusters $\mathbf{k}$, number of iterations $\mathbf{it}$
Output: centroids C, dataset partitions **partitions**

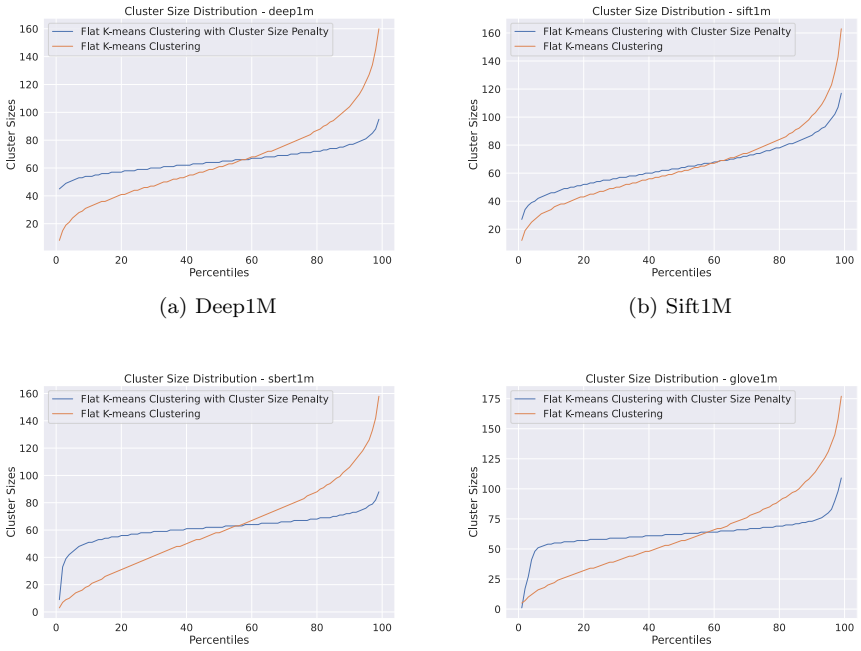
Initialization $C = \text{sample}(\mathbf{X}, \mathbf{k})$
for $i \in \{1, 2, \ldots, \mathbf{it}\}$ **do**
  partitions $= \text{assign}(\mathbf{X}, C)$ {Assign each point to the nearest centroid}
  Initialize a zero vector $\mathbf{s}$, of size $\mathbf{k}$
  for $j \in \{1, 2, \ldots, \mathbf{k}\}$ **do**
    $s_j = \text{len}(\mathbf{partitions}[j])$
  end for
  partitions $= \text{assign\_with\_penalty}(\mathbf{X}, C, \lambda, \mathbf{s})$
  for $j \in \{1, 2, \ldots, \mathbf{k}\}$ **do**
    $C_j = \text{mean}(\mathbf{partitions}[j])$ {Update centroids as the mean of assigned points}
  end for
end for

clude that, although HC initialization effect on the distribution of cluster sizes is part of why it is effective on some datasets and less effective on others, we should look elsewhere to understand why HC initialization affects performance negatively on Sbert1M.

HC initialization, when effective, is preferable to using cluster size penalties. Nevertheless, despite the results of these experiments, recommending the use of cluster size penalties is difficult, especially at the billion-scale where performing hyperparameter search on the λ might mean running tens of iterations of flat clustering that don't add in any way to the performance of the final index. So, since the positive effects of HC initialization can be tested at a smaller scale, when it is effective, using it as a method to reduce the likelihood of producing big clusters is preferable to using cluster size penalties.

Datasets	**Clustering Procedure**	
	Flat K-means Clustering with Penalty	Flat K-means Clustering
Deep1M	**2521**	2831
Sift1M	**3390**	3643
Sbert1M	**5174**	5175
Glove1M	**22368**	22844

Table 4.7: The table shows the number of vectors that need to be scanned to achieve 90% recall@10 on the datasets' query sets, lower is better. The best values for the penalty differ between datasets.



(a) Deep1M (b) Sift1M

(c) Sbert1M (d) Glove1M

Figure 4.8: Plots of the distribution of cluster sizes produced for each dataset by FC training with or without penalties

4.5 Replication

4.5.1 Motivation and setup

As we mentioned in Chapter 2, one of the integral elements of SPANN's recipe to minimize I/Os and reduce latency is replication, a technique that involves inserting the same vector in multiple posting lists, not only the one belonging to the centroid closest to the vector. Replication can improve recall by including points at the boundaries of Voronoi cells in more than one posting list. Yet, excessive replication can lead to increased storage requirements and worse latency if copies of the same vector are in too many posting lists. So the exact replication strategy to employ is crucial to the effectiveness of this technique. SPANN's replication strategy relies not only on the point-to-centroids distances to decide whether a data point should be replicated or not, but also the relative positions and distances of the centroids nearest to the point. More specifically, a point gets replicated in the posting list of a nearby "candidate" centroid if no other centroid, in the posting list of which, the point has already been replicated, is closer to the "candidate" centroid than the "candidate" centroid is to the point. This criterion is related to the notion of the "Relative Neighborhood Graph"

[14] and results in the replication procedure detailed in 4. Following SPANN's configurations, we set the number of max replicas per point to 8. After some tests with different values of the max number of candidate centroids parameter, we picked 64 for our experiments.

Algorithm 4 Relative Neighborhood Graph Procedure

Input: dataset $\mathbf{X}$, centroids $\mathbf{C}$, max number replicas per point ρ, max number of candidate centroids per point γ
Output: the posting lists, **posting_lists**
Initialization: initialize all posting lists to empty lists
for $x_i \in \mathbf{X}$ **do**
  Compute distances between x_i and centroids in $\mathbf{C}$, store in vector $\mathbf{d}$
  Store the ids of the closest γ centroids in order in **top_candidate_ids**
  Initialize an empty list, **rng**, to store the ids of the posting lists in which x_i has already been replicated
  for candidate_id in **top_candidate_ids do**
    Compute distances between the centroid with candidate_id and the centroids already in **rng**, store in vector **rng_distances**, set **rng_distances** $\leftarrow$ [0] if **rng** is empty
    if min(**rng_distances**) $>$ **d**[candidate_id] **then**
      posting_lists[candidate_id].append(x_i)
      rng.append(candidate_id)
    end if
    if len(**rng**) $= \rho$ **then**
      break from the inner for loop
    end if
  end for
end for

4.5.2 Results & Discussion

Replication can be helpful, but it can also lead to redundant replication with limited effects on recall. The numbers of replicated points in table 4.8 show that replication had limited effects on our two text datasets but significantly more on Deep1M and Sift1M. Both datasets saw significant amounts of replication, but they did not show both significant improvements. In table 4.9 , for Deep1M we have to scan almost as many vectors to reach 90% recall@10 with or without replication when using either HC or flat k-means clustering. Because of the risk of significantly increasing the storage requirements on our system while providing limited benefits in recall, we don't recommend using replication in general. Different values of the parameters might reduce unnecessary replication, but since the replication procedure is as expensive as the assignment step of Lloyd's algorithm, it's hard to justify this configuration for lower computational budgets. In SPANN's settings, with more clusters than ours, the procedure was likely to help more because, proportionally, more points

are at or near boundaries. We plan to experiment with smaller clusters as part of our next steps. For more info, see chapter 6.

Datasets	**Total number of datapoints indexed**	
	Flat K-means Clustering	Flat K-means Clustering with Replication
Deep1M	10^6	1541763
Sift1M	10^6	1836029
Sbert1M	10^6	1065747
Glove1M	1183514	1204969
	Hierarchical Clustering	Hierarchical Clustering with Replication
Deep1M	10^6	1361503
Sift1M	10^6	1536541
Sbert1M	10^6	1050924
Glove1M	1183514	1197793

Table 4.8: The table shows the number of vectors indexed for each method and dataset with or without replication.

Datasets	**Methods**	
	Flat K-means Clustering with Replication	Flat K-means Clustering
Deep1M	**2807**	2831
Sift1M	**3196**	3643
Sbert1M	**4787**	5175
Glove1M	**22537**	22844
	Hierarchical Clustering with Replication	Hierarchical Clustering
Deep1M	**3071**	3142
Sift1M	**3444**	3894
Sbert1M	**7538**	7874
Glove1M	**28854**	29238

Table 4.9: The table shows the number of vectors that need to be scanned to achieve 90% recall@10 on the datasets' query sets, lower is better. We show the results for indexes built using either FC or HC and with or without replication.

4.6 Training with Gradient Descent and Adam

4.6.1 Motivations and Setting

The related works we described in Section 2.1.4 made extensive use of SGD-training's flexibility in defining the objective to optimize and the recently developed training tools and methods that facilitate experimentation at bigger scales. Nevertheless, because of their computational complexity, we found little use for them at th billion vectors scale. Therefore, in our experiments, we initially just focused on trying to carry out clustering with gradient descent, using it to build IVF indexes that perform at least as well as indexes that use clustering trained with Lloyd's algorithm. We also tested, just like we did for Lloyd's algorithm, SGD-trained clustering using cluster size penalties. Yet, throughout our testing, hyperparameter search emerged as an inescapable necessity at every step, to choose the optimizer, the gradients accumulation strategy, and to find parameters like learning rate, batch size, and those of the optimizer. As we will detail in the discussion section, this added complexity, especially when applied at the billion-scale makes SGD training impractical, regardless of performance. Ultimately, we decided to run clustering procedures using gradient descent training using a mean squared error reconstruction loss, the ADAM [20] optimizer, batch sizes between 8 thousand and 32 thousand, and the best learning rate value we could find after multiple runs of hyperparameter search, for each dataset.

4.6.2 Results & Discussion

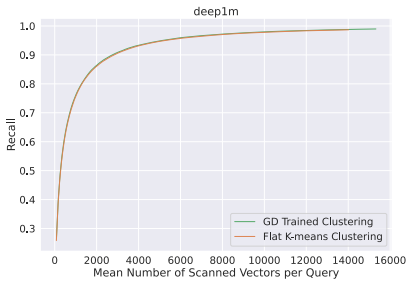
SGD training for clustering, even with extensive hyperparameter search can struggle to reach the performance achieved when training with Lloyd's algorithm. Table 4.10 shows that we could find configurations of the SGD training that performed competitively with Lloyd's algorithm for Deep1M and Sift1M. Yet, the best parameters we could find to train clustering for Sbert1M and Glove1M, however, did not perform nearly as well as the ones trained with Lloyd's algorithm. Trying to understand why we took a look at the cluster size distributions, see Figure 4.11 , and it was clear that this training procedure, when applied to these datasets, was resulting in a significant proportion of empty clusters likely causing the gap in performance with FC, more on this in section 4.7.1.

SGD training requires too much hyperparameter search to get competitive performance with FC as the scale increases. Regardless of the performance of this training procedure at the 1 million-scale, we found it even more difficult to train at the 100 million-scale. Using a multi-GPUs set up with TitanXps we still needed to accumulate gradients over multiple batches to generate meaningful updates. We also found it necessary to use a specific learning rate schedule because even when training with ADAM after of a few full iterations through the Deep100M dataset, the training loss stopped improving. Even with these added tweaks, at this scale, we achieved better performance

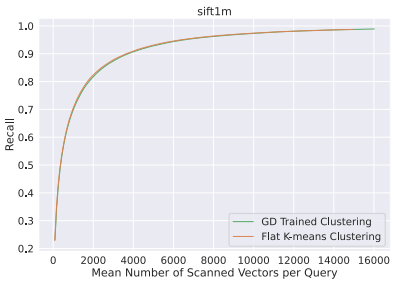
than hierarchical clustering with a SGD trained clustering, but not the same performance of an index built with FC, see Figure 4.10.

Datasets	**Clustering Procedure**	
	SGD trained clustering	Flat K-means Clustering
Deep1M	**2747**	2831
Sift1M	3736	**3643**
Sbert1M	6743	**5175**
Glove1M	26200	**22844**

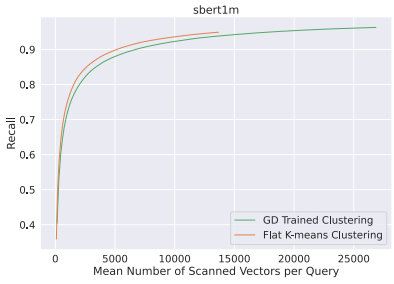
Table 4.10: The table shows the number of vectors that need to be scanned to achieve 90% recall@10 on the datasets' query sets, lower is better.



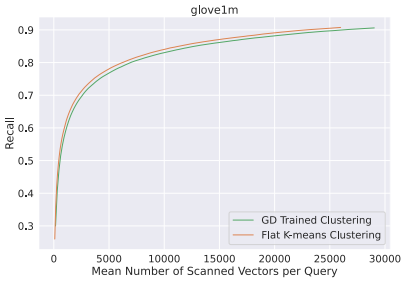
(a) Deep1M



(b) Sift1M



(c) Sbert1M



(d) Glove1M

Figure 4.9: Plots of the experiments comparing flat k-means clustering with SGD trained clustering. To look at the specific number of vectors scanned to achieve 90% recall@10 see table 4.10

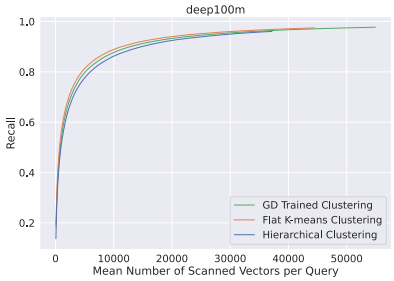
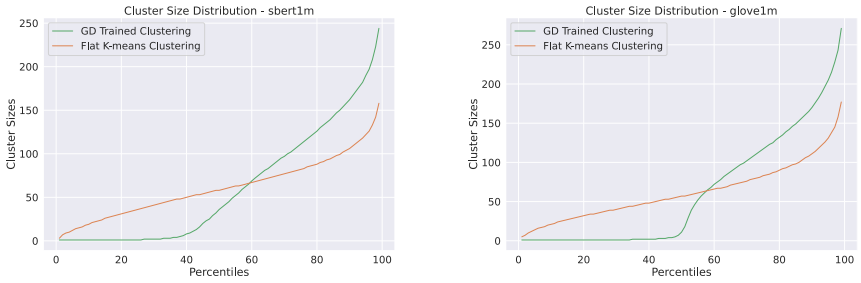


Figure 4.10: Plots of the experiments comparing flat k-means clustering with SGD trained clustering for Deep100M



(a) Sbert1M (b) Glove1M

Figure 4.11: Plots of the distribution of cluster sizes produced for the Sbert1M and Glove1M datasets by FC trained with SGD or Lloyd's algorithm.

4.7 Centroids Update Frequency

4.7.1 Motivations and Setting

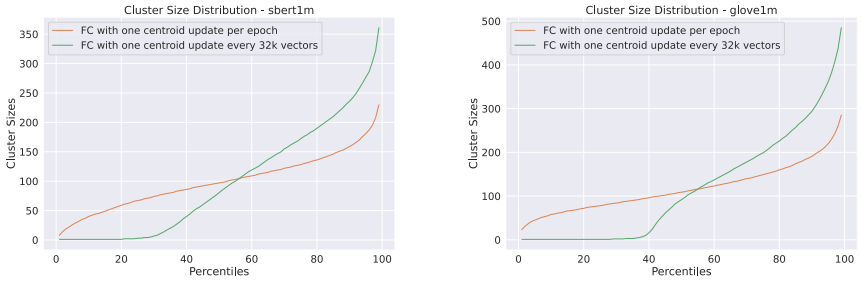
As we mentioned in Subsection 2.1.4 we were interested in exploring the effectiveness of training clustering with stochastic gradient descent (SGD) techniques. Unlike Lloyd's algorithm, as detailed in Algorithm 1, SGD relies on making frequent updates to the centroids. Still, these updates are calculated using only subsets of the training dataset. So to better understand the results of our experiments with SGD training, we looked into mini-batch-based versions of k-means that have been shown to be effective and help speed up convergence of clustering where the number of points to cluster exceeds the number of centroids by many orders of magnitude [31]. Nevertheless, we wanted to compare the performance at convergence of the Mini-batch-based training procedure and standard Lloyd's algorithm in our setting where the number of points to cluster is only one or two orders of magnitude bigger than the number of clusters. We focused on the one million-scale and trained with 10 thousand centroids for 50

iterations, 50 full runs through each dataset. For each dataset, the different runs had the same initialization, but the centroids were updated at different frequencies. We chose to test updates calculated based on 32k vectors, 500k vectors, or twice per iteration and once per iteration, as in Lloyd's algorithm. The setting with 32k vectors per batch is a stand-in for mini-batch in GPU training, a situation similar to the experiments we ran for SGD-trained clustering in Section 4.6.

4.7.2 Results & Discussion

Datasets	**Update frequency**		
	every 32k samples	twice per epoch	once per epoch
Deep1M	3909	3697	**3317**
Sift1M	4989	4704	**4337**
Sbert1M	8921	7122	**6633**
Glove1M	35328	30544	**29138**

Table 4.11: The table shows the number of vectors that need to be scanned to achieve 90% recall@10 on the datasets' query sets, lower is better.



(a) Sbert1M (b) Glove1M

Figure 4.12: Plots of the distribution of cluster sizes produced for the Sbert1M and Glove1M datasets by FC trained with different update frequencies.

In our setting, updating centroids more often than once per epochs leads to worse performance at convergence. The results from 4.11 show that at convergence, mini-batch training, in our setting, incurs significant penalties, and so that updates only once per epoch are preferable. We conducted the rest of our experiments with Lloyd's algorithm, as presented in 1

In our setting, updating centroids more often than once per epochs leads to underutilized centroids. As visible in the cluster size distributions of the different clustering procedures in 4.12 it seems that a significant percentage of the clusters are empty when training with smaller batches leading to worse performance.

Chapter 5

A Systematic Approach to Clustering for ANNS

From our experiments, we could extract multiple valuable conclusions.

Hierarchical clustering performs much better than "no train clustering" and competitively with flat k-means clustering. It is clear from the results of our first experiment from subsection 4.2 that hierarchical clustering, despite its significantly lower computational complexity, performs competitively with flat k-means clustering. Moreover, compared to the performance of the "no train clustering" technique, HC can help us cut down on our latency measure by around half.

Hierarchical clustering Initialization, when effective, is preferable to the use of cluster size penalties to produce more balanced cluster size distributions and, in some cases, improves performance. The results in sections 4.2 and 4.4 show that HC initialization can have a similar positive effect as the use of cluster size penalties, but it does not require significant hyperparameter search. Unlike using cluster size penalties, HC initialization does seem to have negative effects when training a flat clustering on sentences encoded with Sbert, while the effects are positive for the rest of our datasets. We have verified that this discrepancy also carries over to experiments at different scales, so preliminary million-scale experiments should be enough to identify whether HC init is effective on a specific billion-scale dataset.

GD adds too much complexity to the training procedure and requires too much hyperparameter search to be included in the recommendation framework. As we detailed in Section 4.6, in some of our datasets in our setting, using GD training for clustering leads to indexes that struggle to achieve competitive performance compared to indexes where the clustering procedure used LLoyd's algorithm. Moreover, the need for extensive hyperparameters

search and the increased complexity of the procedure make GD training for clustering a poor alternative to Lloyd's algorithm.

Comp. Budget	Preliminaries	Clustering Method
Very Low	—	Hierarchical Clustering
Low	—	Flat Clustering with HC init
Medium	1M scale HC init	FC with HC init or random init
High	penalty value search replication config search	FC with HC init or with penalty and RNG replication

Table 5.1: Our experiments indicate that relevant budget tiers are
Very Low budgets that allow running one iteration of FC or less
Low budgets that have resources to run between 1 and 3 iterations of FC
Medium budgets that allow running more than 3 full iterations of FC
High budgets that allow running hyperparameter search on the exact cluster size penalty to use at the 1B scale

From our experiments, we formulated the recommendation framework that users with different computational budgets can follow to decide on the most adequate clustering procedure for their dataset and budget For users with very low computational budgets to build their index, we suggest simply using HC. We have found HC to be much more effective than other baseline efficient clustering techniques at this scale, as we showed in Section 4.2. Those experiments also lead us to believe this recommendation is valid for budgets that allow to run up to one full iteration of FC. We suggest running FC with HC initialization for users with low computational budgets. Despite the discrepancies we highlighted in Section 4.3, for all our datasets, HC initialization improved performance for the first few FC iterations. Additionally, it also helps to flatten the distribution of cluster sizes. From our result on the Sbert10M dataset in 4.3, we believe this recommendation is valid for users with a budget that allows them to run fewer than 3 full FC iterations. Once users can run more than a few FC iterations, it makes sense to check with smaller-scale experiments if HC initialization is effective for their dataset and application. In Section 4.3, we showed that HC initialization's effectiveness does not seem to be tied to the problem's scale but more to the dataset and query set, so running smaller scale experiments before the billion-scale training runs can help users pick the most effective initialization method. Section 4.3 we also showed the limited value of running FC for tens of iterations. So, for users with high computational budgets that could run that many iterations of FC, we recommend testing the effectiveness of HC initialization just like users with medium budgets. However, in case HC initialization proves ineffective, we also

recommend running hyperparameter search to find the best-performing values of the cluster size penalty as shown in Section 4.4. The additional computational budget could also be used to fine-tune the RNG replication procedure, with the caveat that, unlike the other techniques we discussed above, replication can increase the storage requirements for our system, and it does not just add a one-time cost when building the index.

Chapter 6

Conclusion

6.1 Next Steps

There are things we would like to explore further before submitting our work for publication. Undoubtedly we want to add more datasets to understand better the discrepancy between Deep1B, Sift1B, and Sbert in particular. We have some suspicions related to Sbert's higher dimensionality or its higher number of duplicates. Still, we have yet to verify them, and testing on a more heterogeneous dataset group should help. Especially since from the experiments we carried out in section 4.3, the effects of the Hierarchical Clustering initialization can be assessed even at smaller scales, making it easier to test the technique on more datasets. Nevertheless, we are also interested in carrying out longer billion-scale experiments on more than two to strengthen our analysis further. Testing more configurations of hierarchical clustering might also prove fruitful, despite the difficulties of comparing indexes built with different numbers of Voronoi regions. Expanding testing of HC to lower thresholds closer to SPANN's setting to see if it is any closer to flat k-means clustering when there are fewer points per cluster is also something we would like to explore. Additionally, we plan to test the effectiveness of training FC with only subsets of the data that needs to be indexed to compare it against HC and verify whether that is a valid alternative at a fraction of the complexity of FC.

6.2 Limitations

Even if we carried out all the tests we just mentioned, this analysis of IVF index-building methods is still limited by the lack of a real-world latency to recall analysis, as well as the fact that we did not consider any way of populating the posting lists that does not involve all points to centroids distances to be calculated. This process is as expensive as one full iteration of flat k-means clustering. Still, adding another approximate technique for this step would have likely made results lose generality.

Bibliography

[1] Jonah Anton et al. *Audio Barlow Twins: Self-Supervised Audio Representation Learning*. 2022. arXiv: 2209.14345 [cs.SD].

[2] David Arthur and Sergei Vassilvitskii. "K-Means++: The Advantages of Careful Seeding". In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '07. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035. ISBN: 9780898716245.

[3] Artem Babenko and Victor Lempitsky. "The inverted multi-index". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3069–3076. DOI: 10.1109/CVPR.2012.6248038.

[4] Christian Bauckhage. *k-Means Clustering Is Matrix Factorization*. 2015. arXiv: 1512.07548 [stat.ML].

[5] Qi Chen et al. *SPANN: Highly-efficient Billion-scale Approximate Nearest Neighbor Search*. 2021. arXiv: 2111.08566 [cs.DB].

[6] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[7] Jesse Dodge et al. *Documenting Large Webtext Corpora: A Case Study on the Colossal Clean Crawled Corpus*. 2021. arXiv: 2104.08758 [cs.CL].

[8] Simone Faro and Thierry Lecroq. *The Exact String Matching Problem: a Comprehensive Experimental Evaluation*. 2010. arXiv: 1012.2547 [cs.DS].

[9] E. W. Forgy. "Cluster analysis of multivariate data : efficiency versus interpretability of classifications". In: *Biometrics* 21 (1965), pp. 768–769.

[10] Boyan Gao et al. *Deep clustering with concrete k-means*. 2019. arXiv: 1910.08031 [cs.LG].

[11] Aude Genevay, Gabriel Dulac-Arnold, and Jean-Philippe Vert. *Differentiable Deep Clustering with Cluster Size Constraints*. 2019. arXiv: 1910.09036 [cs.LG].

[12] Nilesh Gupta et al. *ELIAS: End-to-End Learning to Index and Search in Large Output Spaces*. 2023. arXiv: 2210.08410 [cs.LG].

[13] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*. en. 2nd ed. Springer series in statistics. New York, NY: Springer, Feb. 2009.

[14] J.W. Jaromczyk and G.T. Toussaint. "Relative neighborhood graphs and their relatives". In: *Proceedings of the IEEE* 80.9 (1992), pp. 1502–1517. DOI: 10.1109/5.163414.

[15] Herve Jégou, Matthijs Douze, and Cordelia Schmid. "Product Quantization for Nearest Neighbor Search". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.1 (2011), pp. 117–128. DOI: 10.1109/TPAMI.2010.57.

[16] Hervé Jégou et al. *Searching in one billion vectors: re-rank with source coding*. 2011. arXiv: 1102.3828 [cs.IR].

[17] Jeff Johnson, Matthijs Douze, and Hervé Jégou. *Billion-scale similarity search with GPUs*. 2017. arXiv: 1702.08734 [cs.CV].

[18] I. Katsavounidis, C.-C. Jay Kuo, and Zhen Zhang. "A new initialization technique for generalized Lloyd iteration". In: *IEEE Signal Processing Letters* 1.10 (1994), pp. 144–146. DOI: 10.1109/97.329844.

[19] Shima Khoshraftar and Aijun An. *A Survey on Graph Representation Learning Methods*. 2022. arXiv: 2204.01855 [cs.LG].

[20] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[21] Yuanze Lin, Xun Guo, and Yan Lu. *Self-Supervised Video Representation Learning with Meta-Contrastive Network*. 2021. arXiv: 2108.08426 [cs.CV].

[22] Hongfu Liu et al. "Fast Clustering with Flexible Balance Constraints". In: *2018 IEEE International Conference on Big Data (Big Data)*. 2018, pp. 743–750. DOI: 10.1109/BigData.2018.8621917.

[23] S. Lloyd. "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137. DOI: 10.1109/TIT.1982.1056489.

[24] Yury Malkov et al. "Approximate nearest neighbor algorithm based on navigable small world graphs". In: *Information Systems* 45 (2014), pp. 61–68. ISSN: 0306-4379. DOI: https://doi.org/10.1016/j.is.2013.10.006. URL: https://www.sciencedirect.com/science/article/pii/S0306437913001300.

[25] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval*. en. Cambridge, England: Cambridge University Press, July 2008.

[26] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[27] Ahmad Mustapha, Wael Khreich, and Wasim Masr. *A Deep Dive into Deep Cluster*. 2022. arXiv: 2207.11839 [cs.CV].

[28] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162.

[29] Nils Reimers and Iryna Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. 2019. arXiv: 1908.10084 [cs.CL].

[30] Jie Ren, Minjia Zhang, and Dong Li. "HM-ANN: Efficient Billion-Point Nearest Neighbor Search on Heterogeneous Memory". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 10672–10684. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/788d986905533aba051261497ecffcbb-Paper.pdf.

[31] D. Sculley. "Web-Scale k-Means Clustering". In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: Association for Computing Machinery, 2010, pp. 1177–1178. ISBN: 9781605587998. DOI: 10.1145/1772690.1772862. URL: https://doi.org/10.1145/1772690.1772862.

[32] Sohil Atul Shah and Vladlen Koltun. *Deep Continuous Clustering*. 2018. arXiv: 1803.01449 [cs.LG].

[33] Abraham Silberschatz, Henry Korth, and S Sudarshan. *Database System Concepts*. 7th ed. New York, NY: McGraw-Hill, Apr. 2019.

[34] Suhas Jayaram Subramanya et al. "DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node". In: *NeurIPS 2019*. Nov. 2019. URL: https://www.microsoft.com/en-us/research/publication/diskann-fast-accurate-billion-point-nearest-neighbor-search-on-a-single-node/.

[35] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[36] Jingdong Wang et al. *Hashing for Similarity Search: A Survey*. 2014. arXiv: 1408.2927 [cs.DS].

[37] Jiahao Xie et al. "Unsupervised Object-Level Representation Learning from Scene Images". In: *CoRR* abs/2106.11952 (2021). arXiv: 2106.11952. URL: https://arxiv.org/abs/2106.11952.

[38] Artem Babenko Yandex and Victor Lempitsky. "Efficient Indexing of Billion-Scale Datasets of Deep Descriptors". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2055–2063. DOI: 10.1109/CVPR.2016.226.

[39] Jian Yang et al. "An Empirical Guide to the Behavior and Use of Scalable Persistent Memory". In: *18th USENIX Conference on File and Storage Technologies (FAST 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 169–182. ISBN: 978-1-939133-12-0. URL: https://www.usenix.org/conference/fast20/presentation/yang.

[40] Penghang Yin et al. *Stochastic Backward Euler: An Implicit Gradient Descent Algorithm for k-means Clustering*. 2018. arXiv: 1710.07746 [math.OC].

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

RECOMMENDATIONS FOR BUILDING IVF INDEXES IN BILLION-SCALE ANNS SYSTEMS

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
LOPARDO	ANTONIO

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
ZÜRICH, 17/04/23	Antonio Lopardo

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.