DEDRIFT: Robust Similarity Search under Content Drift

Dmitry Baranchuk[*]
Yandex Research
dbaranchuk@yandex-team.ru

Matthijs Douze
Meta AI
matthijs@meta.com

Yash Upadhyay
Meta AI
yashup@meta.com

I. Zeki Yalniz
Meta AI
izy@meta.com

Abstract

The statistical distribution of content uploaded and searched on media sharing sites changes over time due to seasonal, sociological and technical factors. We investigate the impact of this "content drift" for large-scale similarity search tools, based on nearest neighbor search in embedding space. Unless a costly index reconstruction is performed frequently, content drift degrades the search accuracy and efficiency. The degradation is especially severe since, in general, both the query and database distributions change.

We introduce and analyze real-world image and video datasets for which temporal information is available over a long time period. Based on the learnings, we devise DEDRIFT, *a method that updates embedding quantizers to continuously adapt large-scale indexing structures on-the-fly.* DEDRIFT *almost eliminates the accuracy degradation due to the query and database content drift while being up to* 100× *faster than a full index reconstruction.*

1. Introduction

The amount of content available online is growing exponentially over the years. Various online content sharing sites collect billions to trillions of images, videos and posts over time. Efficient Nearest Neighbor Search (NNS) techniques enable searching these vast unstructured databases based on content similarity. NNS is at the core of a plethora of practical machine learning applications including content moderation [17], retrieval augmented modeling [10, 24], keypoint matching for 3D reconstruction [1], image-based geo-localisation [12], content de-duplication [44], k-NN classification [4, 11], defending against adversarial attacks [18], active learning [15] and many others.

NNS techniques extract high dimensional feature vectors (a.k.a. "embeddings") from each item to be indexed. These embeddings may be computed by hand-crafted techniques [35, 28] or, more commonly nowadays, with pre-trained neural networks [7, 37, 36]. Given the database of embedding vectors $\mathcal{D}=\{x_1,\dots,x_N\} \subset \mathbb{R}^d$ and a query embedding $q \in \mathbb{R}^d$, NNS retrieves the closest database example to q from $\mathcal{D}$ according to some similarity measure, typically the L2 distance:

$$\mathrm{argmin}_{x\in\mathcal{D}} \|q-x\|^2 \quad (1)$$

The exact nearest neighbor and its distance can be computed by exhaustively comparing the query embeddings against the entire database. On a single core of a typical server from 2023, this brute-force solution takes a few milliseconds for databases smaller than few tens of thousand examples with embeddings up to a few hundred dimensions.

However, practical use cases require real time search on millions to trillion size databases, where brute-force NNS is too slow [14]. Practitioners resort to approximate NNS (ANNS), trading some accuracy of the results to speed up the search. A common approach is to perform a statistical analysis of $\mathcal{D}$ to build a specialized data structure (an "index" in database terms) that performs the search efficiently. Like any unsupervised machine learning task, the index is trained on representative sample vectors from the data distribution to the index.

A tool commonly used in indexes is vector quantization [22]. It consists in representing each vector by the nearest vector taken in a finite set of centroids $\{c_1,\dots,c_K\} \subset \mathbb{R}^d$. A basic use of quantization is compact storage: the vector can be reduced to the index of the centroid it is assigned to, which occupies just $\lceil \log_2 K \rceil$ bits of storage. The larger K, the better the approximation. As typical for machine learning, the training set $\mathcal{T}$ is distinct from $\mathcal{D}$ (usually $\mathcal{T}$ is a small subset of $\mathcal{D}$). The unsupervised training algorithm of choice for quantization is k-means which guarantees Lloyd's optimality conditions [32].

In an online media sharing site, the index functions as a database system. After the initial training, the index ingests new embeddings by batches as they are uploaded by users and removes content that has been deleted (Figure 1). Depending on the particular indexing structure, addition/deletion may be easy [27], or more difficult [40]. However, a more pernicious problem that appears over the time is content drift. In practice, over months and years, the statistical distribution of the content changes slowly, both for

[*]The work is partially done during the internship at Meta AI

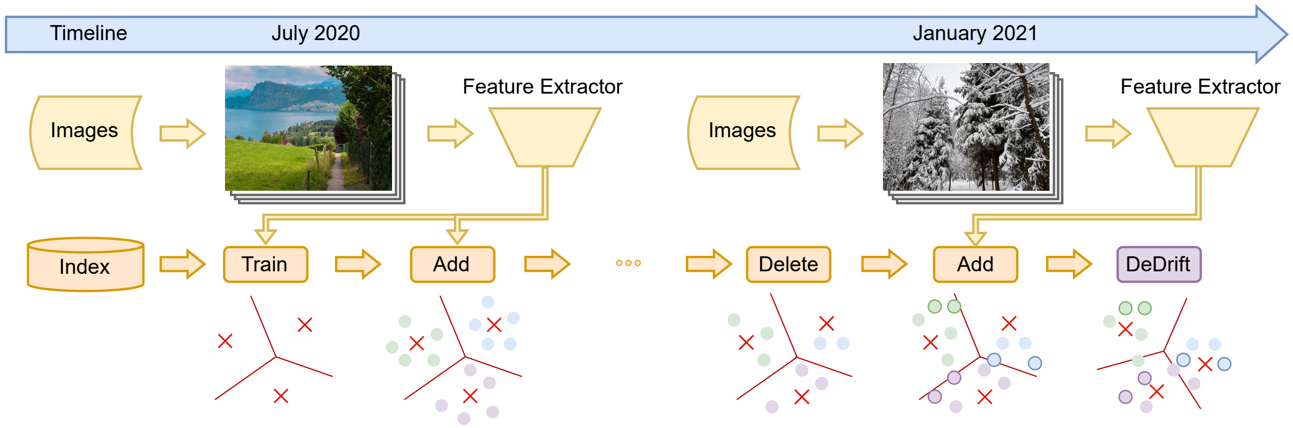

Figure 1. Overview of the dynamic index developed in this work. Images are continuously uploaded to a online sharing platform and their embeddings are added to an index. At any moment, the index can be used to search for similar images. The index quantizes the embeddings into centroids. However, as the content drifts over time, the centroids do not match the data distribution anymore. DEDRIFT introduces a lightweight update procedure to adapt to the new data distribution.

the data inserted in the index and the items that are queried.

The drift on image content may have a technical origin, *e.g.* camera images become sharper and have better contrast; post-processing tools evolve as many platforms offer editing options with image filters that are applied to millions of images. The drift may be seasonal: the type of photos that are taken in summer is not the same as in winter, see Figure 2. Sociological causes could be that people post pictures of leaderboards of a game that became viral, or there is a lockdown period where people share only indoor pictures without big crowds. In rare cases, the distribution may also change suddenly, for example because of an update of the platform that changes how missing images are displayed. The problem posed by this distribution drift is that new incoming vectors are distributed differently from $\mathcal{T}$. Indeed, by design, feature extractors are sensitive to semantic differences in the content. This mismatch between training and indexed vectors impacts the search accuracy negatively. To address this, practitioners monitor the indexing performance and initiate a full index reconstruction once the efficiency degrades noticeably. By definition, this is the optimal update strategy since it adapts exactly to the new data distribution that contains both old and recent vectors. However, at larger scales, this operation becomes a resource bottleneck since all N vectors need to be re-added to the index, and disrupts the services that relies on the index.

Our first contribution is to carefully investigate temporal distribution drift in the context of large-scale nearest neighbor search. For this purpose, we introduce two real-world datasets that exhibit drift. We first measure the drift in an index-agnostic way, on exact search queries (Section 3). Then we measure the impact on various index types that are commonly used for large-scale vector search (Section 4).

Our second contribution is DEDRIFT, a family of adaptation strategies applied to the most vulnerable index types (Section 5). DEDRIFT modifies the index slightly to adapt to the evolution of the vector distribution, without re-indexing all the stored elements, which would incur an $\mathcal{O}(N)$ cost. This adaptation yields search results that are close to the reindexing topline while being orders of magnitude faster. This modification is done while carefully controlling the accuracy degradation. Sections 6 reports and analyzes DEDRIFT's results.

2. Related work

NNS methods. In low-dimensional spaces, NNS can be solved efficiently and exactly with tree structures like the KD-tree [19, 33] and ball-trees [34], that aim at achieving a search time logarithmic in N. However, in higher dimensions, due to the *curse of dimensionality*, the tree structures are ineffective to prune the search space, so there is no efficient exact solution. Therefore, practitioners use approximate NNS (ANNS) methods, trading some accuracy to improve the efficiency. Early ANNS methods rely on data-independent structures, *e.g.* projections on fixed random directions [2, 16]. However, the most efficient methods adapt to the data distribution using vector quantization.

The Inverted File (IVF) based indexing relies on a vector quantizer (the *coarse quantizer*) to partition the database vectors into clusters [41, 27]. At search time, only one or a few of these clusters need to be checked for result vectors. Such pruning approach is required to search in large datasets, so we focus on IVF-based methods in this paper. When scaling up the coarse quantizer can become the computation bottleneck. Several alternatives to plain k-means have been proposed: the inverted multi-index uses a product quantizer [6], graph-based indexes [8] or residual quantizers can also be used [13].

Out-of-distribution (OOD) vectors w.r.t the training distribution are detrimental to the search performance [39]. For

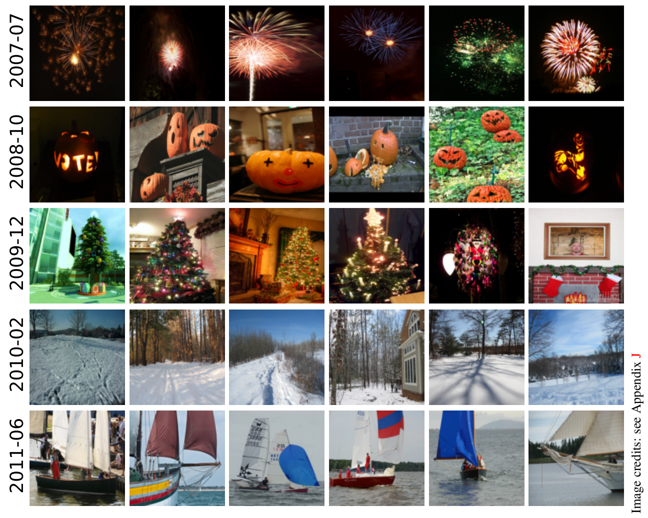

Figure 2. Per row: sample YFCC images that are typical for some months. Refer to 3.3 for how the images were selected.

IVF based methods, they translate to unbalanced cluster sizes, which incurs a slowdown that can be quantified by an *imbalance factor* [26]. In [47], OOD content is addressed before indexing with LSH by adapting a transformation matrix, but LSH's accuracy is limited by its low adaptability to the data distribution. Graph-based pruning methods like HNSW [46] also suffer from OOD queries [25]. In this work, our focus is not on graph-based indexes since they do not scale as well to large datasets.

Database drift. In the literature, ANNS studies are primarily on publicly available offline vector databases where the distribution of query and database examples are fixed and often sampled from the same underlying data distribution [29, 5]. However, these assumptions do not typically hold for real world applications. Not only the query frequency and database size, but also their distributions may drift over time. Recent works [45, 31] simulate this setting and propose adaptive vector encoding methods. In our work, we collect the data with natural content drift and observe that vector codecs are relatively robust to content changes, as opposed to IVF-based indexing structures. Another work [9] adapts the k-means algorithm to take the drift into account. Though this method can improve the performance of IVF indexes, it still requires full index reconstruction.

3. Temporal statistics of user generated content

We present the two representative datasets and report some statistics about them.

3.1. Datasets

We introduce two real-world datasets to serve as a basis for the drift analysis. In principle, most content collected over several years exhibits some form of drift. For the dataset

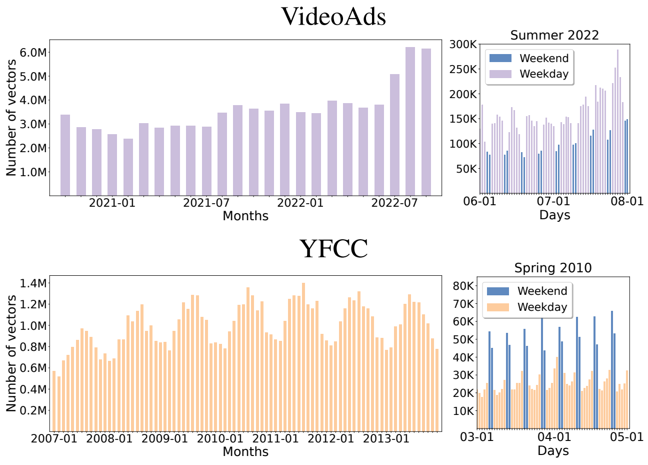

Figure 3. Number of vectors per month (left) and per day on a smaller time span (right).

collection, the constraints are (1) the content should be collected over a long time period (2) the volume should be large enough that statistical trends become apparent (3) the data source should be homogeneous so that other sources of noise are minimized (4) of course, timestamps should be available, either for the data acquisition or the moment the data is uploaded.

The **VideoAds** dataset contains "semantic"[1] embeddings extracted from ads videos published on a large social media website between October 2020 and October 2022. The ads are of professional and non-professional quality, published by individuals, small and large businesses. They were clipped to be of maximum 30 seconds in length. The video encoder for the video ads consisted of a RegNetY [38] frame encoder followed by a Linformer [43] temporal pooling layer which were trained using MoCo [23], resulting in one embedding per video. The dataset contains $86M$ L2-normalized embeddings in 512 dimensions.

The **YFCC** dataset [42] contains 100M images and videos from the Yahoo Flickr photo sharing site. Compared to typical user-generated content sites, Flickr images are of relatively good quality because uploads are limited and users are mostly pro or semi-pro photographers. We filter out videos, broken dates and dummy images of uniform color, leaving $84M$ usable images spanning 7 years, from January 2007 to December 2013. As a feature extractor, we use a pretrained DINO [11] model with ViT-S/16 backbone that outputs unnormalized 384-dimensional vectors. As DINO embeddings have been developed for use as k-NN classifiers, their L2 distances are semantically meaningful.

Before evaluations, we apply a random rotation to all embeddings and uniformly quantize them to 8 bits to make

[1]We call "semantic" embeddings that are intended for input to classifiers. In contrast, copy detection embeddings identify lower-level features useful for image matching [36].

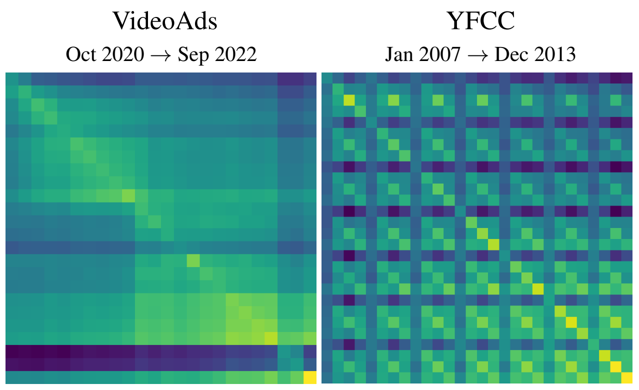

Figure 4. Pairwise similarity matrices between the embeddings over a time period subdivided in steps (1 month for VideoAds and 3 months for YFCC). Blue and yellow correspond to low and high similarities, respectively. Both datasets have noticeable distribution drift over time. In addition, YFCC has clear seasonal correlations.

them less bulky. These operations have almost no impact on neighborhood relations. We do not apply dimensionality reduction methods, e.g., PCA, since it can be considered as a part of the indexing method and explored separately.

3.2. Volume of data

First, we depict daily and monthly traffic for both datasets in Figure 3. For VideoAds, the traffic is lower during weekends and slightly increases over 2 years. For YFCC, the traffic is higher on weekends than on weekdays. This pattern is due to the professional vs. personal nature of the data.

Over the years, the amount of content also grows for both datasets, following the organic growth of online traffic. Interestingly, for YFCC, we observe minimum traffic in January and maximum in July. In the following, we use subsampling to cancel the statistical effect of data volume. Unless stated, we use a monthly granularity because this is the typical scale at which drift occurs.

3.3. Per-month similarities

We start the analysis of temporal distribution drift by visualizing pairwise similarities between months i and j. For each month, we sample a random set $\Phi_i = \{\phi_i^1, .., \phi_i^n\}$ of $|\Phi_i|=10^5$ embeddings. Then, we compute the similarity between Φ_i and Φ_j by averaging nearest-neighbor distances:

$$d(x, \Phi_j) = \frac{1}{L}\sum_{\ell=1}^{L} \|x - \mathrm{NN}_\ell(x, \Phi_j)\|_2 \tag{2}$$

$$S(\Phi_i, \Phi_j) = -\frac{1}{|\Phi_i|}\sum_{\phi_i\in\Phi_i} d(\phi_i, \Phi_j), \tag{3}$$

where L=100 and $\mathrm{NN}_\ell(x, \Phi)$ is the ℓ-th nearest neighbor of x in Φ. Note that the similarity is asymmetric: $S(\Phi_i, \Phi_j) \neq S(\Phi_j, \Phi_i)$ in general. Figure 4 shows the similarity matrices for VideoAds and YFCC datasets. The analysis with daily and weekly granularities is in Appendix A.

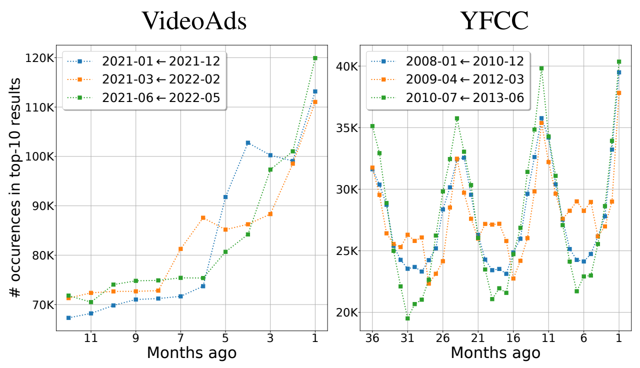

Figure 5. Monthly provenance of the top-10 vectors for queries from a given month.

We observe that the ads data changes over time but does not have noticeable patterns. YFCC content drifts over years and has a strong seasonal dependency. Figure 2 presents images from YFCC for different seasons[2] the seasonal pattern is caused by images of outdoor sceneries (e.g., snow and foliage), seasonal activities (e.g., yachting) or holidays (e.g., Halloween and Christmas). Appendix B studies the impact of these monthly similarity patterns on the distribution of quantization assignments.

3.4. Temporal distribution of nearest neighbors

In this experiment, we investigate the distribution of the nearest neighbors results over predefined time periods. We consider queries Φ_i for month i and search exact top-k (k=10) results in $\Phi_{i-1} \cup ... \cup \Phi_{i-m}$ (m=12 for VideoAds and m=36 for YFCC). Figure 5 shows how many of the $|\Phi_i|\times k$ nearest neighbors come from each of the m previous months for a few settings of the reference month i. We observe that recent content occurs more often in the results for both datasets. Also, for YFCC, there is a clear seasonal trend, *i.e.* content from the same season in years before is more likely to be returned. This hints at a drift because if the vectors were temporally i.i.d. search results would be equally likely to come from all previous m months.

4. Content drift with static indexes

In this section, we assess the effect of content drift for various index types.

4.1. Experimental setting

The evaluation protocol follows a common use case, where the most recent content is used as query data against a backlog of older content.

[2]The images for the month i were selected as follows: a quantizer of size $K = 4096$ is trained on Φ_0. Vectors from Φ_i and Φ_{i-3} are assigned to the K centroids independently. We visualize six random images from one of the top-8 clusters where the size ratio between assignment i over assignment $i-3$ is the largest. Out of these 8, we select one representative cluster for visualization.

Data streaming. The index is trained on months $\{i, ..., i+m-1\}$ and months $\{j, ..., j+m-1\}$ are added to the index, where m is a window size. In the *in-domain* (ID) setting, the training is performed on the same data as the indexed vectors (i=j). In the *out-of-domain* (OOD) setting, the training and indexing time ranges are different (i=0, the first month of the time series and $i \neq j$). If not stated otherwise, we consider m=3 months as it is the maximum retention period of user data for many sharing platforms.

Over time, the index content is updated using a *sliding window* with a time step 1 month: at each month, we remove the data for the oldest month and insert the data for the next month. The queries are a random subset of 10,000 vectors for the month $j+m$. This setting mimics the real-world setting when queries come after the index is constructed.

Metrics. Our ground truth is simply the exact k-NN results from brute force search. As a primary quality measure, we use k−$recall@k$, the fraction of ground-truth k-nearest neighbors found in the top k results retrieved by the search algorithm (k=10) [39]. For IVF-based indexes, the efficiency measure is the number of *distance calculations* (DCS) between the query vector and database entries [6]. At search time, the DCS is given as a fixed "budget": for a query vector q, clusters are visited staring from the nearest centroid to q. The distances between q and all vectors in the clusters are computed until the DCS budget is exhausted (only a fraction of the vectors of the last cluster may be processed). The advantage of this measure over direct timings is that it is independent of the hardware. Appendix G reports search time equivalents for a few settings.

Evaluation. We evaluate the k-recall@k after each content update step over the entire time range. For each index type and DCS setting, we focus on the month where the gap between the ID and OOD settings is the largest.

4.2. Robustness of static indexing methods

In this section, we investigate the robustness of static indexes of the IVF family. Graph-based indexes [46, 20] are not in the scope of this study because they don't scale as well to billion-scale datasets. The components of an IVF index are (1) the codec that determines how the vectors are encoded for in-memory storage (2) the coarse quantizer that determines how the database vectors are clustered.

Vector codecs decrease the size of the vectors for in-memory storage. We consider PCA dimensionality reduction to 128 and 256 dimensions and quantization methods PQ and OPQ [27, 21] to reduce vectors to 16 and 32 bytes. To make results independent of the IVF structure, we measure the 10-recall@10 with an exhaustive comparison between the queries and all vectors in $\mathcal{D}$. In Table 1, we observe a small gap for most settings (<1%). We attribute this to the small codebook sizes of the codecs. In ML terms, they can hardly fit the data and, hence, may be less sensitive to the drift.

Method	VideoAds ID	VideoAds OOD	YFCC ID	YFCC OOD
$PCA128$	0.709	0.698	0.625	0.622
$PCA256$	0.844	0.835	0.867	0.864
$PQ16$	0.237	0.237	0.164	0.160
$PQ32$	0.441	0.441	0.380	0.377
$OPQ16$	0.457	0.446	0.302	0.294
$OPQ32$	0.609	0.601	0.484	0.477

Table 1. Relative 10-recall@10 for in-domain (ID) and out-of-domain (OOD) search, with various vector compression methods. The recall degradation is considered acceptable.

Budget / Method	6000 DCS ID	6000 DCS OOD	12000 DCS ID	12000 DCS OOD	30000 DCS ID	30000 DCS OOD	60000 DCS ID	60000 DCS OOD
IVF16384	0.842	0.732	0.914	0.845	0.966	0.938	0.985	0.973
IVF65536	0.914	0.835	0.956	0.910	0.984	0.967	0.993	0.985
IMI2×8	0.257	0.245	0.391	0.364	0.662	0.592	0.838	0.775
IMI2×10	0.529	0.469	0.732	0.651	0.891	0.841	0.951	0.928
RCQ10_4	0.651	0.531	0.776	0.676	0.899	0.838	0.951	0.916
RCQ12_4	0.809	0.713	0.895	0.826	0.958	0.925	0.981	0.966

Budget / Method	6000 DCS ID	6000 DCS OOD	12000 DCS ID	12000 DCS OOD	30000 DCS ID	30000 DCS OOD	60000 DCS ID	60000 DCS OOD
IVF4096	0.796	0.744	0.892	0.858	0.960	0.945	0.983	0.977
IVF16384	0.894	0.851	0.947	0.922	0.981	0.972	0.993	0.989
IMI2×6	0.245	0.217	0.409	0.360	0.727	0.681	0.872	0.869
IMI2×8	0.449	0.419	0.673	0.627	0.870	0.847	0.948	0.938
RCQ8_4	0.575	0.521	0.730	0.682	0.878	0.850	0.940	0.927
RCQ10_4	0.768	0.718	0.872	0.839	0.949	0.933	0.978	0.971

Table 2. Relative performance of index structures for in-domain (ID) and out-of-domain (OOD) search on VideoAds (top) and YFCC (bottom). The color shade indicates the performance drop between ID and OOD: green is <1%, red is >5%, yellow in-between.

The coarse quantizer partitions the search space. We evaluate the following coarse quantizers with K centroids: IVFK is the plain k-means [27]; IMI2×n a product quantizer with two sub-vectors [6] each quantized to 2^n centroids ($K = 2^{2\times n}$); and RCQn_1_n_2 is a two-level residual quantizer [13], where the quantizers are of sizes 2^{n_1} and 2^{n_2} centroids ($K = 2^{n_1+n_2}$).

Table 2 reports the 10-recall@10 for various DCS budgets representing different operating points. We experiment with different index settings for VideoAds and YFCC, due to their size difference. The content drift has a significant impact in these experiments. We investigate the effect of different window sizes m in Appendix C.

5. Updating the index to accomodate data drift

In this section, we introduce DEDRIFT to reduce the impact of data drift on IVF indexes over time. First, we address the case where vectors are partitioned by the IVF but stored exactly. Then we show how to handle compressed vectors.

Our baseline methods are the lower and upper bounds of the accuracy: **None** keeps the trained part of the quantizer untouched over the entire time span; with **Full**, the index is reconstructed at each update step.

5.1. DeDrift updating strategies

DeDrift-Split addresses imbalance by repartitioning a few clusters. DeDrift-Split collects the vectors from the $k \ll K$ largest clusters into $\mathcal{B}_1$. The objective is to re-assign these vectors into $k_2 > k$ new clusters, where k_2 is chosen so that the average new cluster size is the median cluster size μ of the whole IVF: $k_2 = \lceil |\mathcal{L}_1|/\mu \rceil$. To keep the total number of clusters K constant, vectors from the $k_2 - k$ smallest clusters are collected into $\mathcal{B}_2$. We train k-means with k_2 centroids on $\mathcal{B}_1 \cup \mathcal{B}_2$, and replace the k_2 involved clusters in the index. Other centroids and clusters are left untouched, so the update cost is much lower than N.

DeDrift-Lazy updates all centroids by recomputing the centroid of the vectors assigned to each cluster. In contrast to a full k-means iteration, the vectors are *not* re-assigned after the centroid update. Therefore, **DeDrift-Lazy** smoothly adapts the centroids to the shifting distribution without a costly re-assignment operation. The similar idea was previously considered in the context of VLAD descriptors [3].

DeDrift-Hybrid combines Split and Lazy by updating the centroids first, then splitting k largest clusters.

Discussion. In a non-temporal setting, if the query and database vectors were sampled uniformly from the whole time range, they would be i.i.d. and the optimal quantizer would be k-means (if it could reach the global optimum). The DeDrift approaches are heuristic, they do not offer k-means' optimality guarantees. However, our setting is different: (1) the database is incremental, so we want to avoid doing a costly global k-means and (2) the query vectors are the most recent ones, so the i.i.d. assumption is incorrect. Reason (1) means that we have to fall back to heuristics to "correct" the index on-the-fly and (2) means that DeDrift heuristics may actually *outperform* a full re-indexing.

5.2. DeDrift in the compressed domain

For billion-scale datasets, the vectors stored in the IVF are compressed with codecs, see Section 4.2. However, DeDrift-Split needs to access all original embeddings (*e.g.* from external storage). Besides, DeDrift-Lazy needs to update centroids using the original vectors. Reconstructed vectors could be used but this significantly degrades the performance of the DeDrift variants (see Appendix H). A workaround for DeDrift-Lazy is to store just a subsampled set of training vectors. There is no such workaround for the other methods: they must store the entire database, which is an operational constraint.

Efficient DeDrift-Lazy with PQ compression. We focus on the product quantizer (PQ), which is the most prevalent and difficult to update vector codec. There are two ways to store a database vector x in an IVF index: compress directly (Section 4.2 shows that drift does not affect this much), or store by residual [27], which is more accurate. When storing x by residual, the vector that gets compressed is relative to the centroid c_i that x is assigned to: $r = x - c_i$ is compressed to an approximation $\hat{r}$.

The distance between a query q and the compressed vector $\hat{r}$ is computed in the compressed domain, without decompressing $\hat{r}$. For the L2 distance, this relies on distance look-up tables that are built for every (q, c_i) pair, *i.e.* when starting to process a cluster in the index. The look-up tables are of size $M_{\mathrm{PQ}} \times K_{\mathrm{PQ}}$ for a PQ of M_{PQ} sub-quantizers of K_{PQ} entries. In a static index, one query gets compared to the vectors of L clusters (L a.k.a. nprobe), so look-up tables are computed L times. The runtime cost of computing look-up tables is $L \times d \times K_{\mathrm{PQ}}$ FLOPs.

In DeDrift-Lazy, there are m successive "versions" of c_i. Computing the look-up tables with only the latest version of c_i incurs an unacceptable accuracy impact, so we need to compute look-up tables for each time step. For this, we (1) store the history of c_i's values and (2) partition the vectors within each cluster into m subsets, based on the version c_i that they were encoded with.

The additional index storage required for the historical centroids is in the order of $K \times (m-1) \times d$ floats. For example, for N=10^7, K= 2^{16}, d=384, m=3 and PQ32 encoding, the historical centroids stored in float16 will consume $\sim$16% of the PQ codes. This may be considered significant, especially for large m settings. In the future work, we anticipate addressing this limitation of our approach.

The additional computations required for the look-up tables is $L \times d \times K_{\mathrm{PQ}} \times (m-1)$ FLOPs. For large-scale static indexes, the time to build the look-up tables is small compared to the compressed-domain distance computations. This remains true for small values of m.

Note that the coarse quantization is still performed on K centroids and the residuals w.r.t. historical centroids are computed only for the L nearest centroids.

6. Experimental results

Here, we evaluate DeDrift empirically. All experiments are performed with the FAISS library [30], with the protocol in Section 4.1: ANNS under a fixed budget of distance computations (DCS). Unless specified, we report the results at one time step j chosen to maximize the average recall gap between the None and Full settings over the DCS budgets.

6.1. Uncompressed embeddings

First, we consider DeDrift on IVF methods with uncompressed embeddings. We consider the quantizer is updated

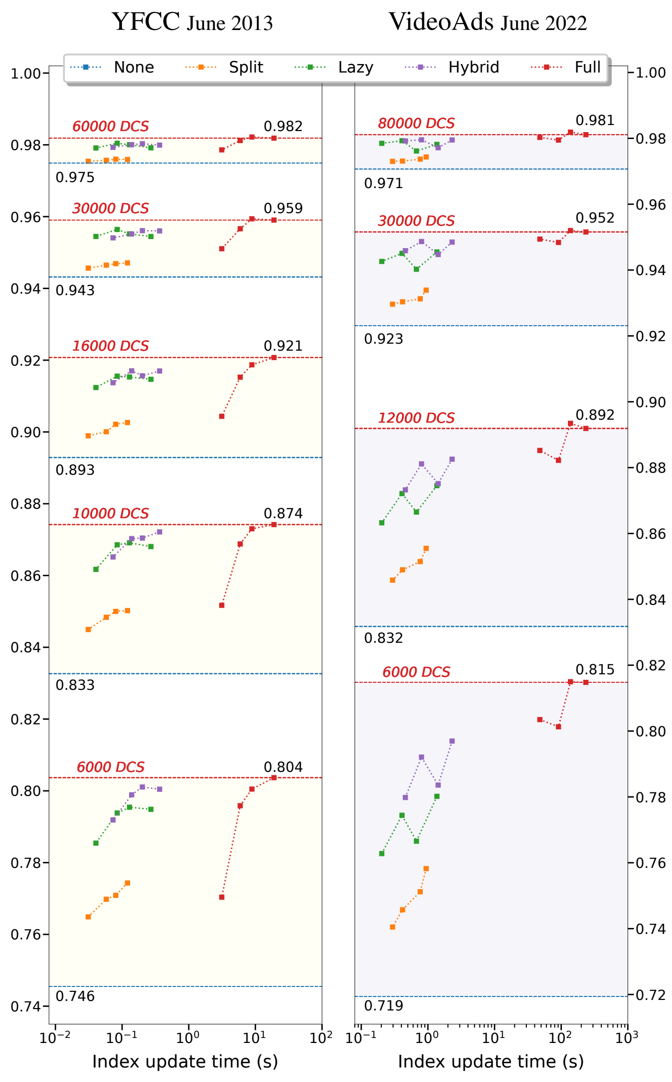

Figure 6. DEDRIFT performance tradeoff: 10-recall@10 as a function of the index update runtime. The upper bounds are full index reconstruction (Full). The lower bound is no reindexing (None). Each point in a sequence represents an update frequency: 6, 3, 2, 1 months (left to right). DEDRIFT variants demonstrate strong robustness to content drift on both datasets, while being two orders of magnitude faster than Full.

every 1, 2, 3 or 6 months to see how long it can be frozen without perceptible loss in performance.

Along with the recall, we measure the average index update time. The IVF contain $K = 16384$ centroids for VideoAds and $K = 4096$ for YFCC (accounting to the different numbers of vectors within m=3 months).

Figure 6 shows that DEDRIFT-Split improves the search performance especially for low DCS budgets, while it is 160× (resp. 250×) more efficient than the index reconstruction (Full) on YFCC (resp. VideoAds). DEDRIFT-Lazy significantly outperforms DEDRIFT-Split on both datasets

Budget (DCS)	6000	12000	20000	30000	60000
None	0.726	0.865	0.917	0.950	0.976
Split	0.796	0.884	0.927	0.951	0.977
Lazy	0.720	0.796	0.832	0.868	0.946
Hybrid	0.824	0.900	0.937	0.959	0.980
Full	0.824	0.904	0.942	0.963	0.984

Table 3. DEDRIFT robustness to outlier content on the YFCC dataset for IVF4096 without compression. DEDRIFT-Lazy noticeably degrades when confronted with a large portion of abnormal images while DEDRIFT-Split and DEDRIFT-Hybrid can successfully avoid the drop in performance. Numbers for j=September 2012.

and all DCS budgets, and is still 70× (YFCC) and 170× (VideoAds) faster than Full. DEDRIFT-Hybrid further improves the DEDRIFT-Lazy performance for low DCS.

Overall, DEDRIFT almost reaches Full on YFCC and significantly reduces the gap between Full and None on VideoAds. *e.g.*, on YFCC, the proposed method provides 5.4%, and 1.3% gains for 6000, and 30000 DCS budgets, respectively. DEDRIFT is about two orders of magnitude cheaper than the full index reconstruction.

In Appendix I, we evaluate Full with *evolving k-means* [9] which adapts the k-means algorithm to the drift. This provides slightly higher recall on both datasets. However, the update costs are similar to the full index reconstruction.

Hyperparameters. We vary crucial parameters for the DEDRIFT variants. For DEDRIFT-Split, we consider k=8 and k=64 clusters for the YFCC and VideoAds datasets, respectively. Higher k values usually lead to slightly better recall rates at the cost of noticeably higher index update costs. DEDRIFT-Lazy performs a single centroid update at a time. Appendix D shows that more training iterations tend to degrade the performance. We hypothesize that further training moves the centroids too far away from the vectors already stored to represent them accurately.

6.2. Robustness to outlier content

We investigate the index robustness to outlier content that occasionally occurs in real-world settings. When starting experiments on the YFCC dataset, we observed bursts of images of uniform color. It turns out these are placeholders for removed images. For the previous experiments, we removed these in the cleanup phase.

To assess DEDRIFT's robustness, we add these images back to the YFCC dataset and repeat the experiments from Section 6.1. Table 3 shows results for September 2012 (month with the largest None-Full gap): DEDRIFT-Lazy significantly degrades compared to all methods, *including no reindexing* (None). In contrast, DEDRIFT-Split and DEDRIFT-Hybrid prevent the performance drop, Hybrid is comparable to the full index update (Full). This shows DEDRIFT-Split makes indexes robust to abnormal traffic.

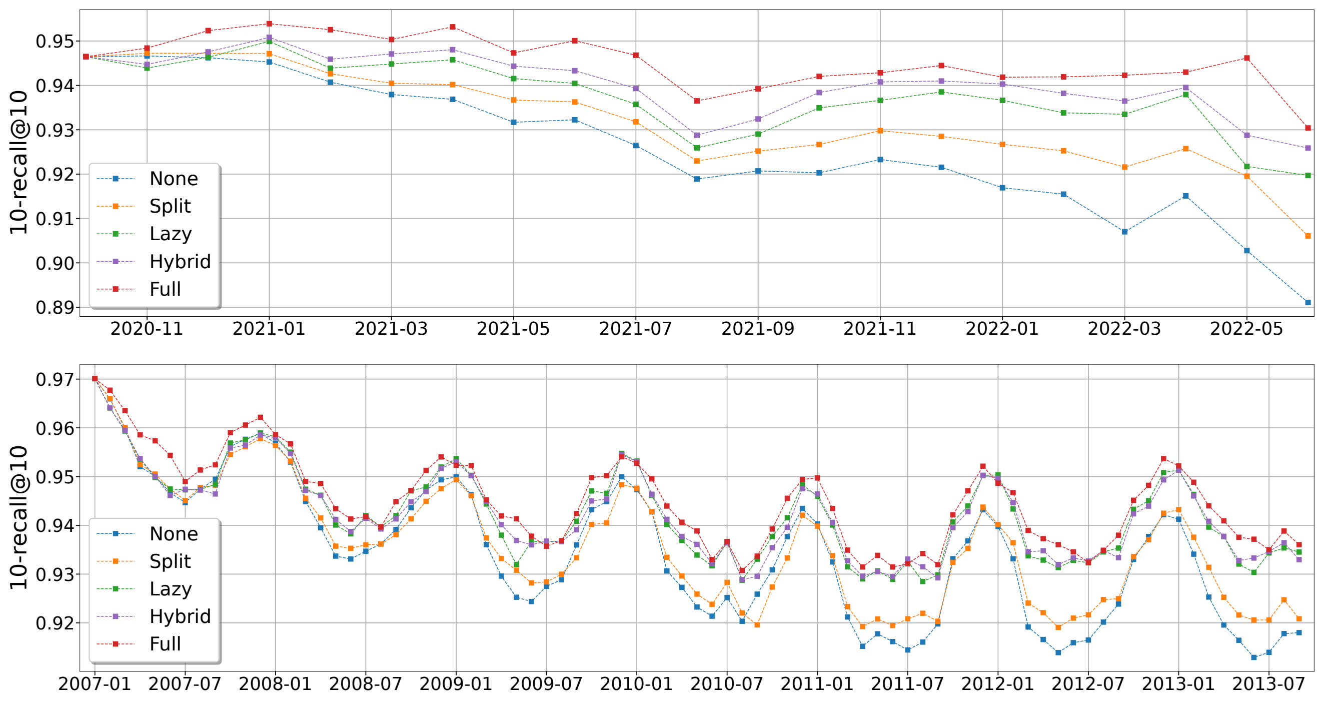

Figure 7. DEDRIFT performance against None and Full over the entire time period for VideoAds (Top) and YFCC (Bottom). For both datasets, we consider 20000 DCS budgets. The x-axis indicates the first month of each time window (j).

	IVF16384,OPQ32, direct encoding				
Budget (DCS)	6000	12000	20000	30000	60000
None	0.501	0.547	0.566	0.577	0.586
Split	0.520	0.556	0.571	0.579	0.587
Lazy	0.530	0.563	0.577	0.583	0.588
Hybrid	0.535	0.564	0.576	0.582	0.589
Full	0.548	0.573	0.583	0.588	0.593
	IVF16384,OPQ32, residual encoding				
Budget (DCS)	6000	12000	20000	30000	60000
None	0.522	0.569	0.589	0.599	0.608
Split	0.544	0.582	0.597	0.605	0.613
Lazy	0.559	0.593	0.607	0.613	0.619
Hybrid	0.561	0.591	0.604	0.610	0.616
Full	0.587	0.615	0.625	0.631	0.635

Table 4. Comparison of the index update methods on the VideoAds dataset for June 2022.

6.3. PQ compressed embeddings

We evaluate indexes with PQ compression. We consider OPQ [21] with 32 bytes per vector. We evaluate two settings: quantize either original embeddings (**direct encoding**) or their residuals w.r.t. the nearest centroid in the coarse quantizer (**residual encoding**). Results for the VideoAds dataset are in Table 4 (see Appendix F for YFCC).

The "residual encoding" is more sensitive to the content drift. Notably, DEDRIFT-Lazy demonstrates significant improvements over no reindexing: $+3.7\%$ and $+1.4\%$ absolute for 6000 and 30000 DCS budgets, respectively. DEDRIFT-Split also outperforms None but the gains are less pronounced compared to DEDRIFT-Lazy. DEDRIFT-Hybrid does not boost DEDRIFT-Lazy further in most cases.

Discussion. DEDRIFT significantly reduces the gap between full index reconstruction and doing nothing. DEDRIFT-Lazy is a key component that brings the most value. We consider it as the primary technique. DEDRIFT-Hybrid demonstrates that DEDRIFT-Split can be complementary to DEDRIFT-Lazy and boost the index performance for low DCS budgets even further. Moreover, the Split variant offers a level of robustness against sudden changes in the dataset distribution.

7. Conclusion

In this paper, we address the robustness of nearest neighbor search to temporal distribution drift. We introduce benchmarks on a few realistic and large-scale datasets, simulate the real-world settings and explore how indexing solutions degrade under drift. We design DEDRIFT, a family of adaptations to the similarity search indexes that mitigate the content drift problem and show their effectiveness.

References

[1] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building rome in a day. *Communications of the ACM*, 54(10):105–112, 2011. 1

[2] Alexandr Andoni, Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab Mirrokni. Locality-sensitive hashing using stable distributions. In Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk, editors, *Nearest-Neighbor Methods in Learning and Vision, theory and practice*. 2005. 2

[3] Relja Arandjelovic and Andrew Zisserman. All about vlad. In *CVPR*, pages 1578–1585. IEEE Computer Society, 2013. 6

[4] Mahmoud Assran, Mathilde Caron, Ishan Misra, Piotr Bojanowski, Armand Joulin, Nicolas Ballas, and Michael Rabbat. Semi-supervised learning of visual features by non-parametrically predicting view assignments with support samples. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8443–8452, 2021. 1

[5] Artem Babenko and Victor Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *Proc. CVPR*, 2016. 3

[6] Artem Babenko and Victor S. Lempitsky. The inverted multi-index. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, 2012. 2, 5

[7] Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. Neural codes for image retrieval. In *Proc. ECCV*, pages 584–599. Springer, 2014. 1

[8] Dmitry Baranchuk, Artem Babenko, and Yury Malkov. Revisiting the inverted indices for billion-scale approximate nearest neighbors. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018. 2

[9] Arkaitz Bidaurrazaga, Aritz Pérez, and Marco Capó. K-means for evolving data streams. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 1006–1011, 2021. 3, 7, 13, 14

[10] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR, 2022. 1

[11] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9650–9660, October 2021. 1, 3

[12] David M Chen, Georges Baatz, Kevin Köser, Sam S Tsai, Ramakrishna Vedantham, Timo Pylvänäinen, Kimmo Roimela, Xin Chen, Jeff Bach, Marc Pollefeys, et al. City-scale landmark identification on mobile devices. In *CVPR 2011*, pages 737–744. IEEE, 2011. 1

[13] Yongjian Chen, Tao Guan, and Cheng Wang. Approximate nearest neighbor search by residual vector quantization. In *Sensors*, 2010. 2, 5

[14] Felix Chern, Blake Hechtman, Andy Davis, Ruiqi Guo, David Majnemer, and Sanjiv Kumar. Tpu-knn: K nearest neighbor search at peak flop/s. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 15489–15501. Curran Associates, Inc., 2022. 1

[15] Cody Coleman, Edward Chou, Julian Katz-Samuels, Sean Culatana, Peter Bailis, Alexander C. Berg, Robert D. Nowak, Roshan Sumbaly, Matei Zaharia, and I. Zeki Yalniz. Similarity search for efficient active learning and search of rare concepts. In *Proceedings of AAAI Conference on Artificial Intelligence*, pages 6402–6410, 2022. 1

[16] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004. 2

[17] Matthijs Douze, Giorgos Tolias, Ed Pizzi, Zoë Papakipos, Lowik Chanussot, Filip Radenovic, Tomas Jenicek, Maxim Maximov, Laura Leal-Taixé, Ismail Elezi, et al. The 2021 image similarity dataset and challenge. *arXiv preprint arXiv:2106.09672*, 2021. 1

[18] Abhimanyu Dubey, Laurens van der Maaten, Zeki Yalniz, Yixuan Li, and Dhruv Mahajan. Defense against adversarial images using web-scale nearest-neighbor search. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pages 8767–8776, 2019. 1

[19] Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977. 2

[20] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143*, 2017. 5

[21] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization for approximate nearest neighbor search. In *CVPR*, 2013. 5, 8

[22] Robert M. Gray and David L. Neuhoff. Quantization. *IEEE transactions on information theory*, 44(6):2325–2383, 1998. 1

[23] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proc. CVPR*, pages 9729–9738, 2020. 3

[24] Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Few-shot learning with retrieval augmented language models. *arXiv preprint arXiv:2208.03299*, 2022. 1

[25] Shikhar Jaiswal, Ravishankar Krishnaswamy, Ankit Garg, Harsha Vardhan Simhadri, and Sheshansh Agrawal. Ood-diskann: Efficient and scalable graph anns for out-of-distribution queries. *arXiv preprint arXiv:2211.12850*, 2022. 3

[26] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Improving bag-of-features for large scale image search. *International journal of computer vision*, 87(3):316–336, 2010. 3

[27] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *TPAMI*, 33(1), 2011. 1, 2, 5, 6

[28] Hervé Jégou, Florent Perronnin, Matthijs Douze, Jorge Sánchez, Patrick Perez, and Cordelia Schmid. Aggregating local image descriptors into compact codes. *PAMI*, 34(9), 2012. 1

[29] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. Searching in one billion vectors: re-rank with source coding. In *Proc. ICCV*, 2011. 3

[30] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *arXiv*, 2017. 6

[31] Qi Liu, Jin Zhang, Defu Lian, Yong Ge, Jianhui Ma, and Enhong Chen. *Online Additive Quantization*, page 1098–1108. Association for Computing Machinery, New York, NY, USA, 2021. 3

[32] Stuart Lloyd. Least squares quantization in PCM. 1982. 1

[33] Marius Muja and David G Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):2227–2240, 2014. 2

[34] Stephen M Omohundro. *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989. 2

[35] F. Perronnin, Y. Liu, J. Sanchez, and H. Poirier. Large-scale image retrieval with compressed Fisher vectors. In *Proc. CVPR*, 2010. 1

[36] Ed Pizzi, Sreya Dutta Roy, Sugosh Nagavara Ravindra, Priya Goyal, and Matthijs Douze. A self-supervised descriptor for image copy detection. In *Proc. CVPR*, 2022. 1, 3

[37] Filip Radenović, Giorgos Tolias, and Ondrej Chum. Fine-tuning CNN image retrieval with no human annotation. *PAMI*, 2018. 1

[38] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces, 2020. 3

[39] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamny, Gopal Srinivasa, Suhas Jayaram Subramanya, and Jingdong Wang. Results of the NeurIPS'21 Challenge on Billion-Scale Approximate Nearest Neighbor Search. In *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track*. 2, 5

[40] Aditi Singh, Suhas Jayaram Subramanya, Ravishankar Krishnaswamy, and Harsha Vardhan Simhadri. Freshdiskann: A fast and accurate graph-based ann index for streaming similarity search, 2021. 1

[41] Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *null*, page 1470. IEEE, 2003. 2

[42] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: the new data in multimedia research. *Commun. ACM*, 59:64–73, 2016. 3

[43] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020. 3

[44] Xiao Wu, Alexander G Hauptmann, and Chong-Wah Ngo. Practical elimination of near-duplicates from web video search. In *Proc. ACM MM*, pages 218–227, 2007. 1

[45] Donna Xu, Ivor W. Tsang, and Ying Zhang. Online product quantization. *IEEE Trans. on Knowl. and Data Eng.*, 30(11):2185–2198, nov 2018. 3

[46] D. A. Yashunin Yu. A. Malkov. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *arXiv preprint arXiv:1603.09320*, 2016. 3, 5

[47] Huayi Zhang, Lei Cao, Yizhou Yan, Samuel Madden, and Elke A Rundensteiner. Continuously adaptive similarity search. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2601–2616, 2020. 3