OOD-DiskANN: Efficient and Scalable Graph ANNS for Out-of-Distribution Queries

Shikhar Jaiswal*
t-sjaiswal@microsoft.com
Microsoft Research India
India

Ravishankar Krishnaswamy
rakri@microsoft.com
Microsoft Research India
India

Ankit Garg
garga@microsoft.com
Microsoft Research India
India

Harsha Vardhan Simhadri
harshasi@microsoft.com
Microsoft Research India
India

Sheshansh Agrawal
sheshansh.agrawal@microsoft.com
Microsoft
USA

ABSTRACT

State-of-the-art algorithms for Approximate Nearest Neighbor Search (ANNS) such as DiskANN, FAISS-IVF, and HNSW build data dependent indices that offer substantially better accuracy and search efficiency over data-agnostic indices by overfitting to the index data distribution. When the query data is drawn from a different distribution – e.g., when index represents image embeddings and query represents textual embeddings – such algorithms lose much of this performance advantage. On a variety of datasets, for a fixed recall target, latency is worse by an order of magnitude or more for Out-Of-Distribution (OOD) queries as compared to In-Distribution (ID) queries. The question we address in this work is whether ANNS algorithms can be made efficient for OOD queries if the index construction is given access to a small sample set of these queries. We answer positively by presenting OOD-DiskANN, which uses a sparing sample (1% of index set size) of OOD queries, and provides up to 40% improvement in mean query latency over SoTA algorithms of a similar memory footprint. OOD-DiskANN is scalable and has the efficiency of graph-based ANNS indices. Some of our contributions can improve query efficiency for ID queries as well.

CCS CONCEPTS

• **Theory of computation** → **Data compression**; **Nearest neighbor algorithms**; • **Information systems** → *Retrieval models and ranking*.

KEYWORDS

Approximate Nearest Neighbor Search (ANNS), Query-Aware Product Quantization (Query-Aware PQ), Graph Algorithms

1 INTRODUCTION

Embedding-based retrieval (aka semantic- or dense-retrieval) is increasingly the paradigm of choice for search and recommendation systems across various domains such as document retrieval [28], web relevance [14, 19, 48], advertisement [12] and content-based image retrieval [29, 44] to name a few. These systems rely on searching an index built over the embeddings corresponding to the objects of interest, to retrieve the nearest embeddings to a query's embedding, based on some geometric distance (such as ℓ_2). Since solving the problem exactly requires an expensive exhaustive scan of the database – which would be impractical for real-world indices that span billions of objects – practical interactive search systems use Approximate Nearest Neighbor Search (ANNS) algorithms with highly sub-linear query complexity [10, 18, 24, 30] to answer such queries. The quality of such ANN indices is often measured by k-recall@k which is the overlap between the top-k results of the index search with the ground truth k-nearest neighbors (k-NNs) in the corpus for the query, averaged over a representative query set.

State-of-the-art algorithms for ANNS, such as graph-based indices [16, 24, 30] which use data-dependent index construction, achieve better query efficiency over prior data-agnostic methods like LSH [6, 18] (see **Section A.1** for more details). Such efficiency enables these indices to serve queries with $> 90\%$ recall with a latency of a few milliseconds, required in interactive web scenarios. They do so by building an index that makes it easy to query a point from the indexed (also called base) dataset itself, thereby enabling the index to answer queries that are drawn roughly from the same distribution. Embedding-based retrieval, via such ANNS indices, is widely deployed for web services in the industry [2].

With the advancement of multi-modal embeddings [37, 39, 43], there is an increasing need for accurate ANNS algorithms that work well with "cross-modal queries". As a motivating example, consider a situation where a user searches through an image index with only a textual description as input. Jointly learnt image-text models can preserve query semantics well, but the text embeddings they generate often lie in a different distribution than the image embeddings, even if both the embeddings share the same representation space.

Therefore, it is not surprising that when queries are drawn from a different distribution – such as in the cross-modal scenario – indices that optimize for base data distribution exhibit a large drop in accuracy and retrieval efficiency for these "out-of-distribution" (OOD) queries. **Figure 1** compares the recall-vs-latency curves of SoTA [7] data-dependent algorithms – HNSW [30], FAISS-IVF [10, 25] and DiskANN [24]– on three datasets. Two of the data sets are text-to-image while the other corresponds to web advertisements and short query embeddings. There is an order of magnitude gap in the latency required to compute NNs of in-distribution or ID queries (these queries are sampled from the distribution generating the base data set, without replacement) and the OOD queries in

*Work done during Research Fellowship at MSR India.

Under Review, April 30–May 04, 2023, Austin, TX
2023. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

both the text-to-image datasets. Such a large drop in latency can make it infeasible to serve accurate results within strict latency budgets needed in web scenarios. To serve multi-modal embeddings and other OOD query sets, there is a need for robust ANNS indices which can adapt to queries drawn from a distribution other than the base data distribution. Hence, this paper asks: **Given a small sample set drawn from the query distribution apriori, is it possible to use such a set to build a better ANNS index that works well for OOD queries?**

1.1 What Qualifies A Query As OOD?

As it is difficult to provide an all-encompassing definition of OOD for all retrieval algorithms, we adopt a simple and natural proxy that seems to correlate highly with the gap in query efficiency between ID and OOD queries for various data-dependent graph and clustering-based ANNS algorithms. *We say that a set of queries are OOD with respect to the points in the base set if the histogram of Mahalanobis distances between queries and base points is significantly different from the histogram for base point-to-base point distances.* This formulation allows one to categorise query sets as weakly or strongly OOD, through the significance of differences between the histograms. Notice that the gap between ID and OOD histograms in **Figure 2** directly correlates with the gap between ID-OOD query efficiency for all three indices in **Figure 1**.[1] While this definition is coarse-grained, our empirical measurements on graph algorithms justifies its use, since the edges in graph-based ANN indices tend to "locally" approximate the base point set (see **Section 2** and **Section A.1**). Another criteria that correlates with gap between ID and OOD query performance is the cluster radii of the top-10 NNs for a given query (see **Figure 3**), which is the minimum radius of a ball enclosing the 10^{th} closest NN.

1.2 Why Is It Hard To Serve OOD Queries?

Compared to the ID queries, OOD queries pose two major problems:

Poor "Clusterability" of the k-NNs of a Query: The large gap in latency for OOD queries can be explained by the fact that the search algorithm needs to look at a lot more points in the index to hit the same recall target as ID queries. This discrepancy can be explained by looking at the "clusterability" of the k-NN nodes pertaining to an ID query and an OOD query. As evident from **Figure 3**, OOD queries have their corresponding k-NNs spread over a larger volumetric space than their ID counterparts. This has different bearings for different types of ANNS algorithms.

For clustering-based solutions, such as FAISS-IVF, this means that the k-NNs of an OOD query do not lie entirely in the clusters deemed closest to the query. Thus, a larger number of clusters need to be probed for finding all of the k-NNs, leading to a higher latency.

For graph-structured ANNS indices, this translates to poor connectivity among the k-NNs of a given query – **Figure 4** demonstrates this phenomenon on a 200 point subgraph of the Yandex T2I dataset, where the k-NNs of an ID query are locally well connected whereas that of an OOD query are much less connected for the graph constructed by the Vamana [24] indexing algorithm. Poor edge connectivity among the k-NNs of a query directly translates into poor recall scores, since the greedy search is prone to running into a "local minima" by visiting only a fraction of the true k-NNs.

Misaligned Loss of Quantization Schemes: Quantization schemes such as Product Quantization (PQ [26]) and Optimized Product Quantization (OPQ [17]) – that are critical to large billion-scale indices [42] like FAISS-IVFPQ [10, 25] and DiskANN [24] – are extremely inaccurate for OOD queries. These quantization methods allow the indexing of billion+ high-dimensional vectors that do not fit in DRAM natively (100M points in 1024 dimensions needs 384GBs of memory) on commodity machines with little DRAM (e.g. 64GB). They also allow for faster, albeit approximate, distance comparisons owing to their smaller CPU and memory bandwidth requirements leading to algorithms with low latency. However, classical formulations of these schemes *attempt to minimize ℓ_2 distance distortions between points in the base data distribution and their quantized vectors.* When used to compute distance estimates between queries drawn from a different distribution and the base dataset, this leads to extreme distortions in relative ranking of distances from the query to its nearest point – *in fact, the actual nearest neighbor may not be in the top 10 candidates in terms of quantized distances to the query!* Therefore, even if we attempt to correct this by re-ranking the algorithm's limited candidate pool with higher precision vectors, the recall may not improve since the top neighbors may not even show in the candidate pool.

1.3 Our Contributions

This paper develops the following techniques towards addressing these challenges:

(1) We present RobustVamana, which improves upon the Vamana graph construction algorithm of DiskANN, and provides superior search latencies for OOD queries (**Section 2**).
(2) We argue, theoretically and empirically, how the existing PQ compression schemes can cause large distortions in distance estimates for OOD queries. We therefore introduce a new formulation, called Accurate PQ (or APQ), that improves performance for OOD queries (**Section 3**).
(3) We present ParallelGorder, a parallel and scalable graph reordering algorithm based on a previous work [46], which reduces I/O requests to SSDs, the primary performance bottleneck for large-scale external memory indices. This provides improved latency for SSD-based indices (**Section 4**).
(4) We put them all together to build billion-scale indices, and demonstrate efficiency gains for OOD queries by upto 40% in terms of mean latency or, alternatively, upto 15% in terms of recall over prior SoTA [42].

1.4 Notation

We denote the dataset of base points as $\mathcal{X}$, where i^{th} point has coordinates $x_i \in \mathbb{R}^D$. We consider directed graphs with vertices corresponding to points in $\mathcal{X}$, and edges $\mathcal{E}$ between them. We refer to such graphs as $\mathcal{G} = (\mathcal{X}, \mathcal{E})$ with slight notational overhead. We denote the sample set of query points as $\mathcal{Q}$, where the i^{th} query point has coordinates $q_i \in \mathbb{R}^D$.

[1] A lot of work has previously been undertaken for OOD detection using Mahalanobis distance, among other approaches, especially in the machine learning [33, 34, 47], computer vision [11, 40] and natural language processing [38] space. The terminology developed for the purpose of OOD categorisation conforms heavily to to these areas, whereas no concrete definitions exist for cross-modal information retrieval to the best of our knowledge.

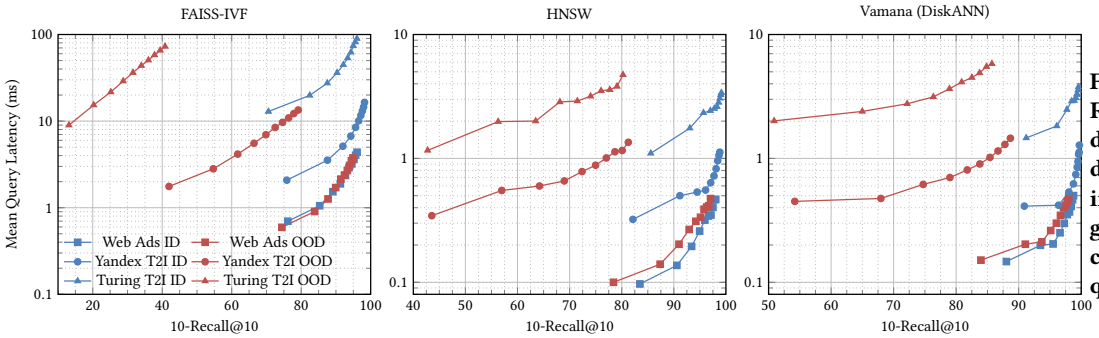


Figure 1: Latency vs. Recall for 10M scale datasets (Web Ads, Yandex T2I [39, 42], Turing T2I [43]) show the gap between query efficiency for ID and OOD queries.

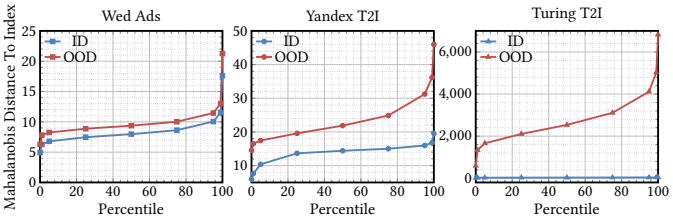


Figure 2: Histogram of Mahalanobis distances for base set-base set (ID) and query set-base set (OOD) for three datasets with minimal, weak and strong OOD properties (left to right).

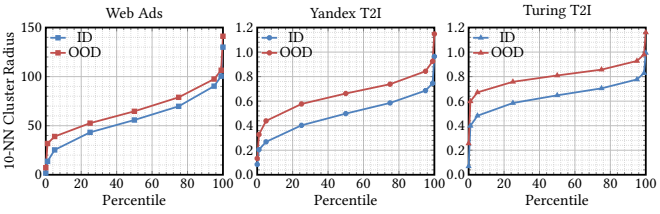


Figure 3: Cluster radii of top-10 NNs of index point (ID) and query (OOD) sample sets.

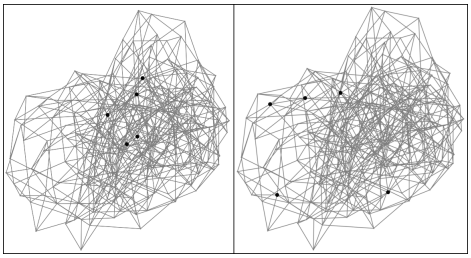


Figure 4: (Left) Top-5 NNs (in black) of an ID query over a random subgraph of the Yandex Text-to-Image-1B dataset. (Right) Top-5 NNs of an OOD query over the same subgraph. Interconnected vertices appear closer to each other.

1.5 Related Work

PQ and OPQ have been the seminal works in the area of vector compression for fast distance estimates and compact memory footprint. They have also inspired algorithms such as LOPQ [27], LSQ [31], LSQ++ [32], ERVQ [4], CQ [50] and AQ [8], and learnt decoder-based techniques such as UNQ [35] and Learnt AQ [5], all of which have pushed the envelope on the accuracy of the distance estimates.

2 ROBUSTVAMANA GRAPH CONSTRUCTION

In this section, we focus on graph-based ANNS indices and present techniques which substantially improve over existing methods for the OOD setting. We focus on the Vamana graph construction heuristic, which is an integral part of the DRAM/SSD hybrid DiskANN system, as our goal is to improve the SoTA for very large scale ANNS scenarios, thereby necessitating SSD-resident indices. An interested reader may refer to **Section A.1** for preliminaries.

We first explain the drawbacks existing algorithms face with OOD queries by briefly describing how such algorithms build and search graph indices. Many popular graph-based indices such as Vamana, HNSW, NSG [16], etc. build so-called *navigable graphs* which employ a simple *greedy heuristic* at search time – the search starts with a *candidate list* $\mathcal{L}$ initialized with a starting vertex s. At any intermediate step, we choose a candidate $p \in \mathcal{L}$, and add its graph out-neighbors $N_{\text{out}}(p)$ to $\mathcal{L}$. To prevent search complexity from exploding, we truncate the list $\mathcal{L}$ to a size of L by retaining the L closest candidates to the query. The search stops once the list does not change, i.e., we arrive at a locally optimal set of candidates.

At a high level, the graphs are then typically constructed in a manner so that every base point can be reached starting from s using the greedy search heuristic. Of course there are nuances in the manner in which the algorithms ensure the degree of the graph is bounded, and the specifics of how to add links to ensure good search performance. However, to our knowledge, all algorithms construct the graph index by only considering the base dataset, completely oblivious of the query distribution. Indeed, this can pose major challenges in the search navigability for OOD queries, as highlighted in **Section 1.2**.

We address this problem by giving access to a small sample, say $\sim$ 1-2% of the base dataset size, of OOD queries to the index construction algorithm. Like Vamana, our RobustVamana algorithm incrementally and iteratively builds the navigable graph, with the i^{th} iteration ensuring that the graph can connect point x_i. A crucial difference is that we include the query sample points in this process. However, a query point will only explore candidates from the base dataset and add links to the closest R base points, whereas a base point will add links to other base points as well as the query points.

Then using these base / query edges we have identified in the above process, we add cross-links between base points which co-occur as the out-neighbors for different query points, subject to the degree threshold of the graph. This has the intended effect of interconnecting all of the NNs of a sample query. Keeping the query points embedded in the index is a no go, since they can saturate the candidate list during search, by virtue of being closer to the evaluation queries. As such, they can kick the true NNs from the base set out of the candidate list, leading to poor recall. Hence, once the cross-links are added using RobustStitch (**Algorithm 3**), we remove these sample query points from the index. The complete specification of RobustVamana is provided in **Algorithm 4**.

2.1 Improvement Over Vamana

Figure 5 summarises our results, where we observe an improvement of 4-10 recall points, for the same latency target across the text-to-image datasets. For the Web Ads dataset, we observe a negligible change in latency vs. recall. This suggests that RobustVamana improves OOD query search performance, without interfering with the ID query search performance. Since the query sample size is merely 1-2% of the base dataset, the overall build-phase time cost and peak RAM footprints increase by the same amount, while the search-phase peak RAM and disk footprints remain unchanged.

Algorithm 1: GreedySearch (s, x_q, τ, $\mathcal{L}$)

Data: Start node s, query vector x_q, boolean τ, search list size $\mathcal{L}$.
Result: Result set $\mathcal{L}$ containing near neighbors, and a set $\mathcal{V}$ containing all visited nodes.

begin
  Initialize sets $\mathcal{L} \leftarrow \{s\}$ and $\mathcal{V} \leftarrow \emptyset$.
  while $\mathcal{L} \setminus \mathcal{V} \neq \emptyset$ **do**
    Let $p^* \leftarrow \arg\min_{p \in \mathcal{L} \setminus \mathcal{V}} \|x_p - x_q\|$
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{p^*\}$
    if τ **then**
      $\mathcal{L} \leftarrow \mathcal{L} \cup \{v \mid v \in N_{\text{out}}(p^*), v \in \mathcal{X}\}$
    else
      $\mathcal{L} \leftarrow \mathcal{L} \cup N_{\text{out}}(p^*)$
    if $|\mathcal{L}| > L$ **then**
      Update $\mathcal{L}$ with closest L nodes to x_q.
  return $[\mathcal{L}; \mathcal{V}]$

3 QUERY-AWARE PRODUCT QUANTIZATION

We now turn our attention to devising improved *quantization schemes* by making them query aware. Product Quantization (PQ) and its variants like OPQ [17] are a very popular class of quantization methods that enable large-scale ANNS on a modest memory footprint. At a high level, these methods compress D-dimensional vector data into a certain number of bytes per vector as follows: the D dimensions are divided into M *chunks* of D/M dimensions each. Within each chunk in D/M-dimensional subspace, an algorithm *learns* K (usually taken to be $K = 256$) representative pivots/centers that best *approximate* the original coordinates in the chunk across the dataset. Typically, this step employs the K-means heuristic over the base vectors restricted to that subspace. Then, each vector

Algorithm 2: RobustPrune (p, $\mathcal{V}$, α, R)

Data: Point $p \in \mathcal{X}$, candidate set $\mathcal{V}$, parameter $\alpha \geq 1$, outdegree R.
Result: $\mathcal{G}$ is modified by setting at most R out-neighbors for p.

begin
  $\mathcal{V} \leftarrow \mathcal{V} \cup N_{\text{out}}(p) \setminus \{p\}$
  $N_{\text{out}}(p) \leftarrow \emptyset$
  while $\mathcal{V} \neq \emptyset$ **do**
    $p^* \leftarrow \arg\min_{p' \in \mathcal{V}} \|x_p - x_{p'}\|$
    $N_{\text{out}}(p) \leftarrow N_{\text{out}}(p) \cup \{p^*\}$
    if $|N_{out}(p)| = R$ **then**
      break
    for $p' \in \mathcal{V}$ **do**
      if $\alpha \cdot \|x_{p^*} - x_{p'}\| \leq \|x_p - x_{p'}\|$ **then**
        Remove p' from $\mathcal{V}$.

Algorithm 3: RobustStitch (p, $\mathcal{W}$, $\mathcal{S}$)

Data: Point $p \in Q$, its in-neighbors $\mathcal{W}$ and spare space counts $\mathcal{S}$.
Result: $\mathcal{G}$ is modified by interconnecting out-neighbors of p.

begin
  for $v \in \mathcal{W}$ **do**
    Remove p from $N_{\text{out}}(v)$.
    $N_{\text{out}}(v) \leftarrow$
     $N_{\text{out}}(v) \cup \{\text{closest } \mathcal{S}[v] \text{ elements from } N_{\text{out}}(p) \setminus \{v\}\}$

Algorithm 4: RobustVamana Indexing Algorithm

Data: Parameters α_1, α_2, L and R.
Result: Directed graph $\mathcal{G}$ over $\mathcal{X}$ with out-degree $\leq R$.

begin
  Let s denote the point closest to the centroid of the dataset $\mathcal{X}$.
  for $\alpha \in \{\alpha_1, \alpha_2\}$ **do**
    for $i \in (\mathcal{X}, Q)$ **do**
      if $i \in \mathcal{X}$ **then**
        Let $[\mathcal{L}; \mathcal{V}] \leftarrow$ `GreedySearch`$(s, x_i, \mathit{false}, L)$
        Update $N_{\text{out}}(i)$ using `RobustPrune`$(i, \mathcal{V}, \alpha, R)$.
      else
        Let $[\mathcal{L}; \mathcal{V}] \leftarrow$ `GreedySearch`$(s, q_i, \mathit{true}, L)$
        Update $N_{\text{out}}(i)$ using closest R nodes from $\mathcal{L}$.
      for $j \in N_{out}(i) \wedge j \in \mathcal{X}$ **do**
        Update $N_{\text{out}}(j) \leftarrow N_{\text{out}}(j) \cup i$
        if $|N_{out}(j)| > R$ **then**
          Update $N_{\text{out}}(j)$ using
           `RobustPrune`$(j, N_{out}(j), \alpha, R)$.
  Let $\mathcal{S} \leftarrow \{s_i \mid s_i \leftarrow (R - |N_{\text{out}}(i)|),\ i \in \mathcal{X}\}$
  for $p \in \mathcal{X}$ **do**
    $k \leftarrow \{v \mid v \in N_{\text{out}}(p),\ v \in Q\}$
    $\mathcal{S}[p] \leftarrow \lfloor \frac{\mathcal{S}[p]}{|k|} \rfloor + 1$
  for $p \in Q$ **do**
    Let $\mathcal{W} \leftarrow \{v \mid p \in N_{\text{out}}(v),\ v \in \mathcal{X}\}$
    Run `RobustStitch`$(p, \mathcal{W}, \mathcal{S})$ to update the graph.

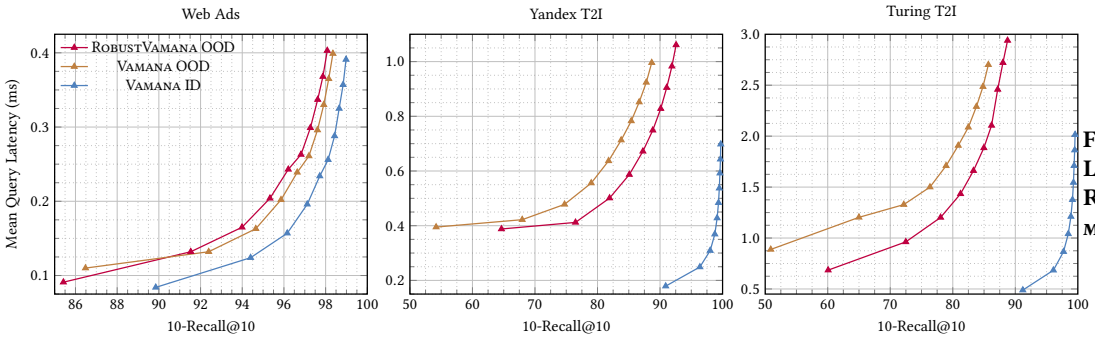


Figure 5: (Left to right) Latency vs. Recall for RobustVamana and Vamana at 10M scale.

can be represented by the closest pivot/center within the chunk, thereby requiring only $\log_2 K$ bits to encode, which is 1 byte for $K = 256$. Following this process independently for each chunk, we can approximate any vector using a cross product of pivots in the chunk-wise sub-spaces using a total of M bytes. Schemes like OPQ, LOPQ and others build on top of this basic primitive.

Our main contribution in this section is to essentially redesign the quantization process within each chunk by making it aware of the target query distribution. An interested reader may further refer to **Section A.2**, **Section A.3** and **Section A.4**.

3.1 Problem with Existing Methods

Consider an existing quantization scheme and suppose that for some base vector x_i, it's restriction x_i^j within a chunk j is assigned to a representative pivot μ. Also let q^j denote the restriction of a query q to the this chunk. Then we have:

$$\|q^j - x_i^j\|_2^2 = \|q^j - \mu + \mu - x_i^j\|_2^2 = \|q^j - \mu\|_2^2 + \|x_i^j - \mu\|_2^2 - 2\langle q^j - \mu, x_i^j - \mu\rangle \tag{1}$$

Thus, chunk-wise error between actual and estimated distance is:

$$\underbrace{\|q^j - x_i^j\|_2^2}_{\text{Actual}} - \underbrace{\|q^j - \mu\|_2^2}_{\text{Estimated}} = \langle x_i^j + \mu - 2q^j, x_i^j - \mu\rangle \tag{2}$$

A positive value of the above term signifies that the base point appears closer to the query than it actually is, and vice versa. To provide intuition on why existing methods are not very performant on OOD datasets, we re-organize this distortion as:

$$\langle x_i^j + \mu - 2q^j, x_i^j - \mu\rangle = 2\|\frac{x_i^j + \mu}{2} - q^j\|_2 \|x_i^j - \mu\|_2 \cos\theta \tag{3}$$

where θ is the angle between the above two inner product vectors. The crucial issue is that *existing PQ schemes only aim to minimize the squared euclidean norm of the second multiplicative term* $\|x_i^j - \mu\|_2$, which is optimized by finding by the K pivots through classical K-means method and assigning each point to the nearest pivot.

In order to remedy this issue, we propose Accurate Product Quantization (APQ) which *chooses to directly optimize for the true distortion* in Equation 2. More formally, for each sample query $q \in \mathcal{Q}$, let $H_T(q)$ denote the set of closest T vectors to q for a parameter T. Then, for each base point $x_i \in \mathcal{X}$, we define a set of relevant queries as follows: $\mathcal{Q}_{x_i} = \{q : x_i \in H_T(q)\}$, and if $|\mathcal{Q}_{x_i}| > \phi$, we refine $\mathcal{Q}_{x_i}$ to contain the closest ϕ queries.

For each chunk, we are now ready to define our loss function $\mathcal{L}_{\text{APQ-L2}}$ for assigning the restriction x_i^j, of x_i in the j^{th} chunk, to a candidate pivot vector μ:

$$\mathcal{L}_{\text{APQ-L2}}(x_i^j, \mu, \mathcal{Q}_{x_i}) = \frac{1}{|\mathcal{Q}_{x_i}|}\sum_{q \in \mathcal{Q}_{x_i}} \begin{cases} -\langle x_i^j + \mu - 2q^j, x_i^j - \mu\rangle, & \text{if } x_i \in \text{H}_{T'}(q) \\ |\langle x_i^j + \mu - 2q^j, x_i^j - \mu\rangle|, & \text{otherwise} \end{cases} \tag{4}$$

where, $\mu \in \mathbb{R}^{\frac{D}{M}}$ is the representative pivot (to be learnt) that x_i^j will be assigned to in the quantization. This formulation is inspired from prior work employing contrastive losses [9, 15, 21, 36]. In the first case, the optimizer pushes both the negative and positive distortions in the positive direction for a select subset of relevant queries which are known to be really close (as $T' << T$). In the second case, the optimizer minimizes the magnitude of the distortions themselves.

The overall optimization for the given chunk j is to choose K pivots $\{\mu_1, \mu_2, \ldots, \mu_K\}$ and assign each base vector x_i^j to one of these pivots $\mu_{\sigma(i)}$ to minimize the total loss $\sum_i \mathcal{L}_{\text{APQ-L2}}(x_i^j, \mu_{\sigma(i)}, \mathcal{Q}_{x_i})$. We optimize this loss objective in Equation 4 through an alternating minimization approach. **Algorithm 5** and **Algorithm 6** summarise our approach where we partition our base dataset into M chunks, and learn a set of K independent pivots for each chunk. We initiate the learning process for a chunk by choosing a random set of K pivot points, and follow up with a membership assignment and pivot update steps in an alternating fashion. The learnt set of pivots pertaining to the j^{th} chunk form a dictionary C_j, and the final set of M dictionaries form our learnt codebook C. This codebook is then used to encode points in the index.

To illustrate the power and generality of this approach, we also deploy this approach within the OPQ framework which learns a product quantization of the vectors after a suitable rotation of the data and queries. We don't modify the steps which learn the rotation, but employ the APQ algorithm in lieu of the steps which learn the PQ codebook. We refer to the resulting scheme as AOPQ.

3.2 Improvement Over PQ and OPQ

As evident from **Figure 6**, we observe a 3-4% improvement in recall, for the same latency target, in the case of text-to-image datasets. We also observe a negligible change in performance for the minimally OOD Web Ads queries. Since we need to construct query sets for each base point, we have a disk space overhead of roughly 400 bytes per base point (for storing roughly 100 query ids). Additionally, we train Accurate PQ through gradient descent, which takes roughly 1 minute per chunk using Tensorflow [1] on a 32-core CPU.

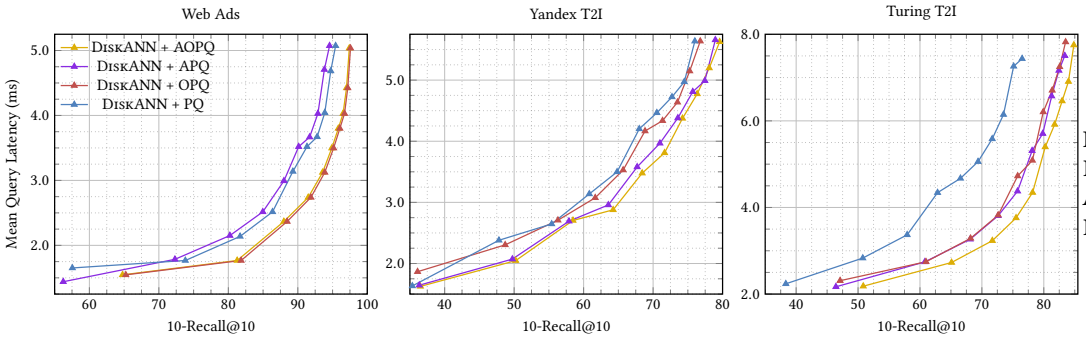


Figure 6: (Left to right) Latency vs. Recall for AOPQ, APQ, OPQ and PQ at 10M scale.

Algorithm 5: Chunk-Wise Accurate PQ Algorithm

Data: The restriction $\mathcal{X}^j$ of the given data $\mathcal{X}$ to a chunk of D/M dimensions, and number of pivots K.
Result: Learnt pivots pertaining to the chunk.
Initialization Step: Sample K random points from $\mathcal{X}^j$, and assign them as initial pivots $\mu_1, \ldots, \mu_K$.
begin

Membership Assignment Step: Find the most suitable pivot $\tilde{\mu}$ for each base point chunk x_i^j, by looking at all possible choices for μ:

$$\tilde{\mu} = \underset{\mu \in \{\mu_1, \ldots, \mu_K\}}{\operatorname{argmin}} \mathcal{L}_{\text{APQ}}(x_i^j, \mu, Q_{x_i})$$

Pivot Update Step: For every pivot μ_i, let X_{μ_i} be the set of base points assigned to it. Then update μ_i by running gradient descent to optimize the loss:

$$\mu_i = \underset{\mu \in \mathbb{R}^{D/M}}{\operatorname{argmin}} \sum_{x_i^j \in X_{\mu_i}} \mathcal{L}_{\text{APQ}}(x_i^j, \mu, Q_{x_i})$$

Repeat **Membership Assignment Step** and **Pivot Update Step** until either convergence or maximum number of iteration is reached.

Algorithm 6: Accurate PQ Codebook Learning Algorithm

Data: Number of pivots per chunk K and number of chunks M.
Result: Learnt codebook $C = \{C_1, C_2, \ldots, C_M\}$ of pivots pertaining to all chunks.
begin
parallel for $j \in \{0, \ldots, M-1\}$ **do**

Initialization Step: Create a chunk dataset $\mathcal{X}^j$ by splitting each data point x_i into M chunks, each of dimension $\frac{D}{M}$, such that for a chunk j:

$$x_i^j = (x_{i,1+j*\frac{D}{M}}, x_{i,2+j*\frac{D}{M}}, \ldots, x_{i,(j+1)*\frac{D}{M}})$$

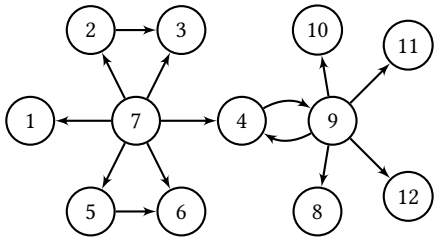
Chunk-Wise Learning Step: Create a dictionary C_j with K pivots using **Algorithm 5**.

Encoding Step: Create a codebook $C = \{C_1, C_2, \ldots, C_M\}$. For each chunk vector x_i^j, pick a pivot $\mu_j \in C_j$ according to the **Membership Assignment Step** of **Algorithm 5**. Hence, each base point x_i can be encoded as a concatenation of $\{\mu_1, \mu_2, \ldots, \mu_M\}$.

4 PARALLELGORDER GRAPH REORDERING

Next, we show how we can *re-order* the graph nodes to improve locality of reference during graph traversal for index search. While there has been some recent work on this topic for million-scale and memory-resident indices such as HNSW [13], we present ParallelGorder with two notable differences: (a) it is highly parallelizable, enabling it to re-order billion-scale graphs, and (b) it optimizes for the number of SSD I/Os during search, for disk-resident graphs, as opposed to the cache miss rate considered in prior work.

Graph-based ANNS algorithms, at their core, traverse nodes in the graph in a greedy manner by walking over to the closest, unexplored node to the query in the candidate list. After arriving at a particular node, we need to access the data vectors of its out-neighbors. These accesses are essentially *random accesses*, leading to large cache miss rates. Moreover a node, once explored, is never referenced again for that query. This essentially means that greedy graph traversal has poor inherent locality of reference.



2	3	7	1	4	5	6	9	8	10	11	12

Gorder Ordering for the above graph.

2	3	7	1

9	4	8	10

5	6	11	12

A possible ParallelGorder Ordering.

Figure 7: Graph ordering example for a window of size 4. Seed nodes are highlighted in blue. Each window signifies a packed disk sector.

Recently, there has been some work on *graph reordering methods* to improve search latency by placing information regarding nodes

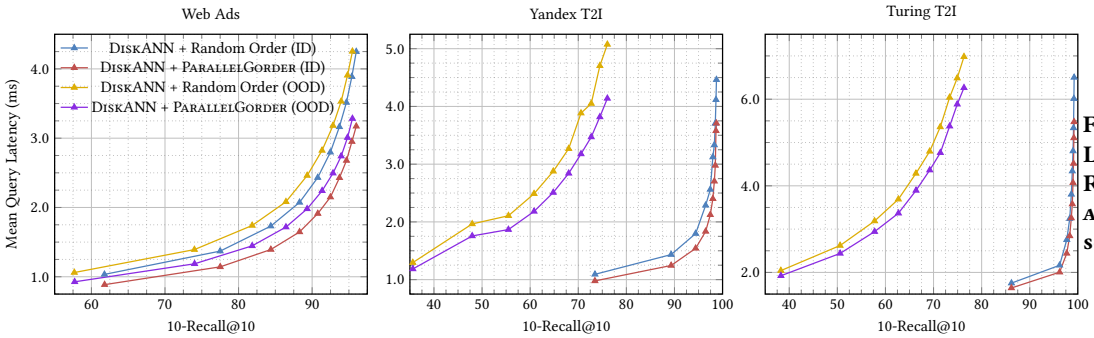


Figure 8: (Left to right) Latency vs. Recall for Random Order and ParallelGorder at 10M scale.

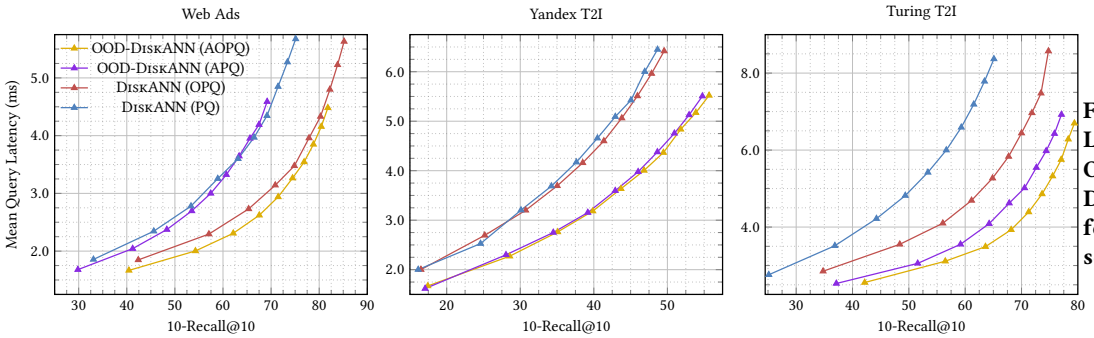


Figure 9: (Left to right) Latency vs. Recall for OOD-DiskANN and DiskANN with different quantization schemes at full scale.

Algorithm 7: ParallelGorder (S_{sector}, S_{node})

Data: Sector size S_{sector} and max node size S_{node} (both in bytes).
Result: Array P holding the new order of nodes.
begin
  Initialize empty array P of size $|\mathcal{X}|$.
  $w \leftarrow \lfloor \frac{S_{\text{sector}}}{S_{\text{node}}} \rfloor$; $\mathcal{D} \leftarrow \{v \mapsto \text{false}\}\ \forall\ v \in \mathcal{X}$
  parallel for $i \in [0, 1, \ldots, \lfloor \frac{|\mathcal{X}|}{w} \rfloor - 1]$ **do**
    Pick a random, unpacked seed node s.
    SectorPack($P[i * w]$, $\mathcal{D}$, s, w,)
  Pick a random, unpacked seed node s.
  SectorPack($P[\lfloor \frac{|\mathcal{X}|}{w} \rfloor * w]$, $\mathcal{D}$, s, w,)

Algorithm 8: SectorPack (P, $\mathcal{D}$, s, w)

Data: Sub-array P, bitmap $\mathcal{D}$ of already packed nodes, seed node s and the number of elements to pack w.
Result: Sub-array P with nodes to pack in the disk sector.
Member Function $\mathcal{H}$ :: IncrementKey(v)
  if $v \in \mathcal{H}$.keys() **then**
    $v_{\text{count}} \leftarrow \mathcal{H}$.get_count($v$)
    $\mathcal{H}$.delete(v) ; $\mathcal{H}$.insert($v, v_{count} + 1$)
  else
    $\mathcal{H}$.insert(v, 1)
begin
  Initialize empty max heap $\mathcal{H}$.
  $i \leftarrow 0; P[i] \leftarrow s$
  while $i < w$ **do**
    $v_e \leftarrow P[i]; i \leftarrow i + 1$
    for $u \in N_{out}(v_e)$ **do**
      $\mathcal{H}$.IncrementKey(u)
    for $u \in N_{in}(v_e)$ **do**
      $\mathcal{H}$.IncrementKey(u)
      for $t \in N_{out}(u)$ **do**
        $\mathcal{H}$.IncrementKey(t)
    while true **do**
      if $\mathcal{H}$.empty() **then**
        Pick a random unpacked seed node v_{max}.
        break
      $v_{\text{max}} \leftarrow \mathcal{H}$.top() ; $\mathcal{H}$.pop()
      if not $\mathcal{D}[v_{max}]$ **then**
        break
    $\mathcal{D}[v_{\text{max}}] \leftarrow$ true; $P[i] \leftarrow v_{\text{max}}$

likely to be referenced together on the same or adjacent cache lines. This has the effect of reducing the number of cache misses incurred during graph traversal, as one needs fewer cache line transfers to access the set of referenced nodes. Wei et al. [46] introduces a greedy heuristic called Gorder to compute such a re-ordering, and apply it to the HNSW algorithm for small, million-scale indices. An interested reader may also refer to Coleman et al. [13], which has surveyed several other reordering techniques applied to the HNSW algorithm, and consistently observed the best speedups with Gorder algorithm for in-memory graph ANNS.

4.1 Locality for External Memory

While previous work on re-ordering has focused on small-scale RAM-resident graph indices, these in-memory indices become prohibitively expensive for large-scale datasets.

We argue that there is an equally strong motivation for re-ordering on billion-scale indices as well. As the DiskANN greedy

search traverses a graph, the out-neighborhood of the referenced node is fetched by making a random SSD read request. One of the major bottlenecks for search performance on such disk-based indices is the maximum number of random I/O Operations Per Second (IOPS) supported by the SSD. Waiting on such I/O typically accounts for more than 50% of the overall latency for DiskANN even when used with the latest NVMe (Non-Volatile Memory Express) SSDs. Our main idea is that, since random reads on SSDs are typically done in sectors of 4KBs or larger, there is scope for reducing the number of SSD reads if we optimize the index layout by *placing nodes which are likely to be referenced in the same search path, in the same SSD sector.*

While Gorder has shown reasonable success for small datasets, there are significant scalability issues for larger datasets (> 100 million). Vanilla Gorder is ill-suited for this task, since it has an $O(NR^2)$ time complexity of computing the re-ordering, where N denotes the number of graph nodes, and R denotes the maximum out-degree. Moreover, the algorithm is based on a sliding window-based greedy implementation, and is inherently non-parallelizable. As a result, the total build-time cost from our experiments ends up being around 2× as much as the index construction.

To this end, we make use of the fact that *one only needs to maximise the locality between all the graph nodes stored in a particular disk sector*, without worrying about the locality of nodes spread across two different sectors. This then opens up the possibility to "pack" each sector with nodes individually (see **Figure 7**), *making the entire process parallelizable.* In case of a race condition, where a node has multiple different sectors where it can be packed into, we simply choose a sector at random for the node, and re-initiate the packing process on the rest of the sectors.

Algorithm 7 and **Algorithm 8** describe our approach, where we make use of a standard max heap for tracking the most suitable node to add to a partially-packed sector, starting with a randomly-chosen seed node. Note that using a max heap pushes the runtime of our algorithm to $O(NR^2 \log R)$, which is only slightly worse than the original, non-parallel algorithm since R is typically < 150.

4.2 Observed Improvement

We observe anywhere from 15-40% reduction in I/O requests in our experiments. As evident from **Figure 8**, this leads to 10-25% improvement in mean latency, for the same recall target. These gains are obtained with roughly a 7-12% increase in build-time, keeping the peak RAM and disk usage unchanged during both the build and search, across all datasets.

5 EVALUATION

Datasets: We evaluate our work on three large-scale datasets. Two of these are Text-to-Image (T2I) datasets, namely Yandex Text-to-Image-1B (200 dimensions) and Turing Text-to-Image (1024 dimensions). In Yandex Text-to-Image-1B, the base dataset consists of 1 billion image embeddings produced by the Se-ResNext-101 [22] model, and queries are textual embeddings produced by a variant of the DSSM [23] model. In Turing Text-to-Image, the base dataset consists of 87 million image embeddings, and the queries are textual embeddings, both generated by the Turing Bletchley [43] model. The third, Web Ads, is an web query-to-advertisements dataset in 64 dimensions. The base embeddings encode both the text and images in 2.4 billion product advertisements scraped from a well-known web index, and encoded by a variant of the CLSM [41] model. For Yandex Text-to-Image-1B and Web Ads datasets, we evaluate our results on a set of 100K test queries and for Turing Text-to-Image, we evaluate our results on a set of 30K test queries. All queries used during build or evaluation, whether ID or OOD, are sampled randomly and without replacement from these datasets.

Hardware: All experiments were conducted on a bare-metal server with 2× Xeon Gold 6140 CPUs (36 cores, 72 threads), with 500 GBs of DDR4 RAM and a 3.2TB Samsung PM1725a PCIe SSD.

Parameters: We initiated all of our experiments with $R = 64$, $L = 128$, $\alpha_1 = 1.0$, $\alpha_2 = 1.2$. For all PQ-based quantization, we made use of 8-bit encoding (256 centroids per chunk) and use $T' = 10$, $T = 80000$ and $\phi = 100$. We train the APQ/AOPQ pivots on a sample of 100K base and 100K query points. For RAM-resident compressed vectors, we set the number of chunks $M = \frac{D}{4}$, where D was the dimensionality of the vectors of a given dataset. For the disk-resident quantized vectors, we used $M = D$. The build-time RAM budget was software limited to 200 GBs. During search, we set a beam width of 4, and set the number of search threads to 16.

Comparison with Existing Compression Schemes: The comparison of our work is limited to PQ and OPQ because these two techniques have the lowest search-time compute footprints. As observed from Fig. 2 and 3 in Martinez et al. [32] and Fig. 4 in Amara et al. [5], PQ and OPQ are about 10×-100× faster than the other approaches, making them suitable for low-latency disk-based ANNS. APQ and AOPQ have the same search-time compute footprints as PQ and OPQ respectively. Recently introduced SCANN [20] also offers the same compute requirements as PQ, however further analysis [49] shows that the reported accuracy gains have been due to its usage of 4-bit encoding (16 centroids per chunk), which vanish when compared against a 4-bit PQ baseline [3].

Figure 9 demonstrates the contribution of all three approaches towards improving the accuracy vs. latency boundary of the large scale ANNS. Taken together, these techniques yield upto 40% improvement in latency, or 15% improvement in recall.

6 CONCLUSION

We have presented and evaluated a new framework for OOD ANNS. We extended the graph construction algorithm of DiskANN, making it suitable for search with OOD queries, while maintaining ID query search qualities. We investigated how current Product Quantization schemes are inefficient in handling the nuances of OOD ANNS, and provided an improved formulation which leads to better performance. We also parallelized a state-of-the-art graph reordering algorithm, and adapted it to disk-based ANNS scenario. By combining all of our contributions, we have established a new state-of-the-art ANNS solution for the OOD setting, and verified our results on three large-scale datasets.

ACKNOWLEDGEMENTS

We would like to thank Ranajoy Sadhukhan, Neel Karia, Siddharth Gollapudi, Gopal Srinivasa and Nipun Kwatra for their helpful feedback and discussions. We would also like to thank Kriti Aggarwal,

Owais Khan Mohammed, Xiaochuan Ni, Subhojit Som for providing the Turing Text-to-Image dataset.

REFERENCES

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283. https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[2] Facebook AI. 2020. SimSearchNet++. https://ai.facebook.com/blog/heres-how-were-using-ai-to-help-detect-misinformation/

[3] Facebook AI. 2021. Turing Bletchley: A Universal Image Language Representation model by Microsoft. https://github.com/facebookresearch/faiss/wiki/Indexing-1M-vectors#4-bit-pq-comparison-with-scann

[4] Liefu Ai, Junqing Yu, Tao Guan, and Yunfeng He. 2014. Efficient approximate nearest neighbor search by optimized residual vector quantization. In *2014 12th International Workshop on Content-Based Multimedia Indexing (CBMI)*. 1–4. https://doi.org/10.1109/CBMI.2014.6849842

[5] Kenza Amara, Matthijs Douze, Alexandre Sablayrolles, and Hervé Jégou. 2022. Nearest Neighbor Search with Compact Codes: A Decoder Perspective. In *Proceedings of the 2022 International Conference on Multimedia Retrieval* (Newark, NJ, USA) *(ICMR '22)*. Association for Computing Machinery, New York, NY, USA, 167–175. https://doi.org/10.1145/3512527.3531408

[6] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. 2015. Practical and Optimal LSH for Angular Distance. In *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2015/file/2823f4797102ce1a1aec05359cc16dd9-Paper.pdf

[7] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms. *Inf. Syst.* 87, C (jan 2020), 13 pages. https://doi.org/10.1016/j.is.2019.02.006

[8] Artem Babenko and Victor Lempitsky. 2014. Additive Quantization for Extreme Vector Compression. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 931–938. https://doi.org/10.1109/CVPR.2014.124

[9] Philip Bachman, R Devon Hjelm, and William Buchwalter. 2019. Learning Representations by Maximizing Mutual Information Across Views. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/ddf354219aac374f1d40b7e760ee5bb7-Paper.pdf

[10] Dmitry Baranchuk, Artem Babenko, and Yury Malkov. 2018. Revisiting the Inverted Indices for Billion-Scale Approximate Nearest Neighbors. In *Proceedings of the European Conference on Computer Vision (ECCV)*. https://openaccess.thecvf.com/content_ECCV_2018/papers/Dmitry_Baranchuk_Revisiting_the_Inverted_ECCV_2018_paper.pdf

[11] Martin Bauw, Santiago Velasco-Forero, Jesus Angulo, Claude Adnet, and Olivier Airiau. 2021. Deep Random Projection Outlyingness for Unsupervised Anomaly Detection. https://doi.org/10.48550/ARXIV.2106.15307

[12] K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, and M. Varma. 2016. The extreme classification repository: Multi-label datasets and code. http://manikvarma.org/downloads/XC/XMLRepository.html

[13] Benjamin Coleman, Santiago Segarra, Anshumali Shrivastava, and Alex Smola. 2021. Graph Reordering for Cache-Efficient Near Neighbor Search. *arXiv preprint arXiv:2104.03221* (2021). https://arxiv.org/abs/2104.03221

[14] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M. Voorhees. 2020. Overview of the TREC 2019 deep learning track. In *Text REtrieval Conference (TREC)*. TREC. https://www.microsoft.com/en-us/research/publication/overview-of-the-trec-2019-deep-learning-track/

[15] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. 2014. Discriminative Unsupervised Feature Learning with Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger (Eds.), Vol. 27. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2014/file/07563a3fe3bbe7e3ba84431ad9d055af-Paper.pdf

[16] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search with the Navigating Spreading-out Graph. *Proc. VLDB Endow.* 12, 5 (jan 2019), 461–474. https://doi.org/10.14778/3303753.3303754

[17] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized Product Quantization for Approximate Nearest Neighbor Search. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2946–2953. https://doi.org/10.1109/CVPR.2013.379

[18] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 518–529. https://www.vldb.org/conf/1999/P49.pdf

[19] Jiafeng Guo, Yinqiong Cai, Yixing Fan, Fei Sun, Ruqing Zhang, and Xueqi Cheng. 2022. Semantic Models for the First-Stage Retrieval: A Comprehensive Review. *ACM Transactions on Information Systems* 40, 4 (oct 2022), 1–42. https://doi.org/10.1145/3486250

[20] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 3887–3896. https://proceedings.mlr.press/v119/guo20h.html

[21] R. Hadsell, S. Chopra, and Y. LeCun. 2006. Dimensionality Reduction by Learning an Invariant Mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, Vol. 2. 1735–1742. https://doi.org/10.1109/CVPR.2006.100

[22] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. 2017. Squeeze-and-Excitation Networks. https://doi.org/10.48550/ARXIV.1709.01507

[23] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning Deep Structured Semantic Models for Web Search Using Clickthrough Data. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management* (San Francisco, California, USA) *(CIKM '13)*. Association for Computing Machinery, New York, NY, USA, 2333–2338. https://doi.org/10.1145/2505515.2505665

[24] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/09853c7fb1d3f8ee67a61b6bf4a7f8e6-Paper.pdf

[25] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data* 7 (2019), 535–547. https://arxiv.org/pdf/1702.08734.pdf

[26] Herve Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128. https://doi.org/10.1109/TPAMI.2010.57

[27] Yannis Kalantidis and Yannis Avrithis. 2014. Locally Optimized Product Quantization for Approximate Nearest Neighbor Search. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2329–2336. https://doi.org/10.1109/CVPR.2014.298

[28] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Online, 6769–6781. https://doi.org/10.18653/v1/2020.emnlp-main.550

[29] Fuhui Long, Hongjiang Zhang, and David Dagan Feng. 2003. Fundamentals of content-based image retrieval. In *Multimedia information retrieval and management*. Springer, 1–26.

[30] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

[31] Julieta Martinez, Joris Clement, Holger H. Hoos, and James J. Little. 2016. Revisiting Additive Quantization. In *ECCV (2)*. 137–153. https://doi.org/10.1007/978-3-319-46475-6_9

[32] Julieta Martinez, Shobhit Zakhmi, Holger H. Hoos, and James J. Little. 2018. LSQ++: Lower running time and higher recall in multi-codebook quantization. In *Proceedings of the European Conference on Computer Vision (ECCV)*. https://openaccess.thecvf.com/content_ECCV_2018/papers/Julieta_Martinez_LSQ_lower_runtime_ECCV_2018_paper.pdf

[33] Alexander Meinke, Julian Bitterwolf, and Matthias Hein. 2021. Provably Robust Detection of Out-of-distribution Data (almost) for free. https://doi.org/10.48550/ARXIV.2106.04260

[34] John P Miller, Rohan Taori, Aditi Raghunathan, Shiori Sagawa, Pang Wei Koh, Vaishaal Shankar, Percy Liang, Yair Carmon, and Ludwig Schmidt. 2021. Accuracy on the Line: on the Strong Correlation Between Out-of-Distribution and In-Distribution Generalization. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 7721–7735. https://proceedings.mlr.press/v139/miller21b.html

[35] Stanislav Morozov and Artem Babenko. 2019. Unsupervised Neural Quantization for Compressed-Domain Similarity Search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. https://openaccess.thecvf.com/content_ICCV_2019/papers/Morozov_Unsupervised_Neural_Quantization_for_Compressed-Domain_Similarity_Search_ICCV_2019_paper.pdf

[36] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation Learning with Contrastive Predictive Coding. https://doi.org/10.48550/ARXIV.1807.03748

[37] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, 8748–8763. https://proceedings.mlr.press/v139/radford21a.html

[38] Mrinal Rawat, Ramya Hebbalaguppe, and Lovekesh Vig. 2021. PnPOOD : Out-Of-Distribution Detection for Text Classification via Plug andPlay Data Augmentation. https://doi.org/10.48550/ARXIV.2111.00506

[39] Yandex Research. 2021. Text-to-Image-1B: Benchmarks for Billion-Scale Similarity Search. https://research.yandex.com/blog/benchmarks-for-billion-scale-similarity-search

[40] Alexander Robey, Hamed Hassani, and George J. Pappas. 2020. Model-Based Robust Deep Learning: Generalizing to Natural, Out-of-Distribution Data. https://doi.org/10.48550/ARXIV.2005.10247

[41] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management* (Shanghai, China) *(CIKM '14)*. Association for Computing Machinery, New York, NY, USA, 101–110. https://doi.org/10.1145/2661829.2661935

[42] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamy, Gopal Srinivasa, Suhas Jayaram Subramanya, and Jingdong Wang. 2022. Results of the NeurIPS'21 Challenge on Billion-Scale Approximate Nearest Neighbor Search. https://doi.org/10.48550/ARXIV.2205.03763

[43] Saurabh Tiwary. 2021. Turing Bletchley: A Universal Image Language Representation model by Microsoft. https://www.microsoft.com/en-us/research/blog/turing-bletchley-a-universal-image-language-representation-model-by-microsoft/

[44] Ji Wan, Dayong Wang, Steven Chu Hong Hoi, Pengcheng Wu, Jianke Zhu, Yongdong Zhang, and Jintao Li. 2014. Deep learning for content-based image retrieval: A comprehensive study. In *Proceedings of the 22nd ACM international conference on Multimedia*. 157–166.

[45] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 14, 11 (jul 2021), 1964–1978. https://doi.org/10.14778/3476249.3476255

[46] Hao Wei, Jeffrey Xu Yu, Can Lu, and Xuemin Lin. 2016. Speedup Graph Processing by Graph Ordering. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 1813–1828. https://doi.org/10.1145/2882903.2915220

[47] Zhisheng Xiao, Qing Yan, and Yali Amit. 2021. Do We Really Need to Learn Representations from In-domain Data for Outlier Detection? https://doi.org/10.48550/ARXIV.2105.09270

[48] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *International Conference on Learning Representations (ICLR)*. https://www.microsoft.com/en-us/research/publication/approximate-nearest-neighbor-negative-contrastive-learning-for-dense-text-retrieval/

[49] Takuma Yamaguchi. 2021. Similarity Search: ScaNN and 4-bit PQ. https://medium.com/@kumon/similarity-search-scann-and-4-bit-pq-ab98766b32bd

[50] Ting Zhang, Chao Du, and Jingdong Wang. 2014. Composite Quantization for Approximate Nearest Neighbor Search. In *Proceedings of the 31st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 32)*, Eric P. Xing and Tony Jebara (Eds.). PMLR, Bejing, China, 838–846. https://proceedings.mlr.press/v32/zhangd14.html

A APPENDIX

A.1 ANNS Preliminaries

ANNS algorithms typically work in two phases, namely the build and the search phase. During build phase, they construct a data structure (called index) over the input database (also called base dataset) of points and associated vectors. The index is then used for rapidly sifting through the database while looking for the nearest neighbors of a query vector, during the search phase.

Graph ANNS algorithms construct a graph index, where the edges constructed by the build algorithm are guided by the vector-to-vector distances between the base dataset of points. A graph node thus comprises a base point, its associated vector and the out-neighbors of the base point.

Existing graph algorithms are derived from four base classes of graphs [45]: Delaunay Graphs (DGs), Minimum Spanning Trees (MSTs), K-Nearest Neighbor Graphs (k-NNGs) and Relative Neighborhood Graphs (RNGs). DGs are expensive to construct, by virtue of being almost fully connected in high dimensions, and hence do not find much practical use. MSTs guarantee node reachability, but at the cost of long traversal paths. k-NNGs, by definition, only construct short-ranged edges, and do not guarantee global connectivity. RNGs aim to balance the construction of short and long-ranged edges, and inspire the current state-of-the-art graph-based approaches such as Vamana and HNSW.

State-of-the-art graph-based indices [16, 24, 30] use data-dependent index construction to achieve 10×-100× more query efficiency over data-agnostic methods like LSH [6, 18], as measured by the number of index points accessed to achieve a certain recall (**Figure 10**).

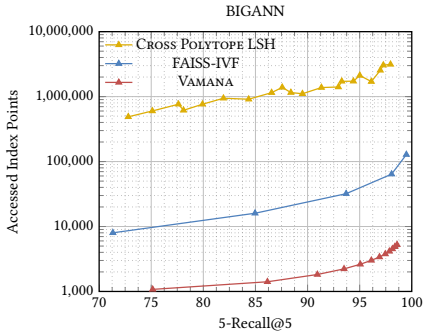


Figure 10: Accessed Index Points vs. Recall for BIGANN dataset (100M scale).

Scaling ANNS solutions to billion-scale datasets requires one to adapt the built index to be disk-resident, while supporting low-latency search, because of RAM constraints. As such, disk-based solutions load compressed representations of the dataset vectors in RAM for fast and approximate distance estimates during search. The original, higher precision dataset vectors are stored as a component of the graph nodes on the disk. Once the compressed vector search has shortlisted a bunch of candidate nearest neighbors of a query, the disk-resident graph nodes, pertaining to these candidates, are paged into RAM through I/O requests for accurately re-ranking the candidates with higher precision vectors.

A.2 Compression with Product Quantization

Let $\mathcal{X} = \{x_i \in \mathbb{R}^D | i = 1, \ldots, N\}$ be the set of N base dataset vectors that we wish to compress, and $C = \{\mu_i \in \mathbb{R}^D | i = 1, \ldots, K\}$ be a set of K learnable vectors (called pivots).

Quantization, in a classical sense, aims to learn a mapping $\mathcal{S} : \mathcal{X} \rightarrow C$ such that the overall reconstruction error is minimised:

$$\min \sum_{i=1}^{N} \|x_i - \mathcal{S}(x_i)\|_2^2, \ \mathcal{S}(x_i) \in C \tag{5}$$

where $\mathcal{S}(x_i)$ is the pivot (out of K pivots) closest to the x_i.

This mapping is typically learnt through the K-Means clustering algorithm, where the cluster centroids serve as pivots, and are generated using an alternating minimization approach. We can then use the logarithm of the index values of the K pivots, to assign bit-codes to individual base vectors for compression. The length of the bit codes is, thus, logarithmic in the number of pivots ($\log_2 K$).

The computational complexity of this approach is linear in K. Hence, generating a large number of pivots, which is desirable for minimising the reconstruction error, is computationally expensive through naive K-Means. This prohibits the generation of compressed codes with larger bit-lengths.

In Product Quantization (PQ), we can divide each base vector into M concatenated sub-vectors (commonly referred to as "chunks"), each of dimension $\frac{D}{M}$, and quantize each chunk with K unique pivots. The classical formulation for PQ minimizes the following objective function:

$$\min \sum_{j=1}^{M} \sum_{i=1}^{N} \|x_i^j - \mu\|_2^2, \ \mu \in \mathbb{R}^{\frac{D}{M}} \tag{6}$$

where x_i^j is the j^{th} chunk of the i^{th} vector, and μ is the closest pivot (out of the K pivots in the j^{th} chunk) to this sub-vector. Since each chunk has K unique pivots, we have a total of $M \times K$ pivots which generate bit-codes of length $M \log_2 K$, whereas the naive technique would have required K^M pivots to achieve the same bit-code length.

Optimized Product Quantization (OPQ) goes a step further, and learns an orthonormal matrix $\mathcal{R}$ for transforming the vector space of the dataset. It thus minimizes:

$$\min \sum_{j=1}^{M} \sum_{i=1}^{N} \|(\mathcal{R}x_i)^j - \mathcal{R}\mu\|_2^2, \ \mu \in \mathbb{R}^{\frac{D}{M}}, \ \mathcal{R}^T\mathcal{R} = \mathcal{I} \tag{7}$$

A.3 Why Do Existing PQ Schemes Fail In OOD Setting?

Existing PQ schemes only consider the squared euclidean norm of the second multiplicative term as an upper-bound of the overall distance distortion. By triangle inequality, we have:

$$\underbrace{\|q^j - x_i^j\|_2}_{\text{Actual}} \leq \underbrace{\|q^j - \mu\|_2}_{\text{Estimated}} + \underbrace{\|x_i^j - \mu\|_2}_{\text{Clustering Error}} \tag{8}$$

$$\underbrace{(\|q^j - x_i^j\|_2}_{\text{Actual}} - \underbrace{\|q^j - \mu\|_2)^2}_{\text{Estimated}} \leq \underbrace{\|x_i^j - \mu\|_2^2}_{\text{Clustering Error}} \tag{9}$$

Hence, through the above analysis, the argument is made that one only needs to minimize the squared clustering objective in Equation 3, as it upper bounds the total distortion.

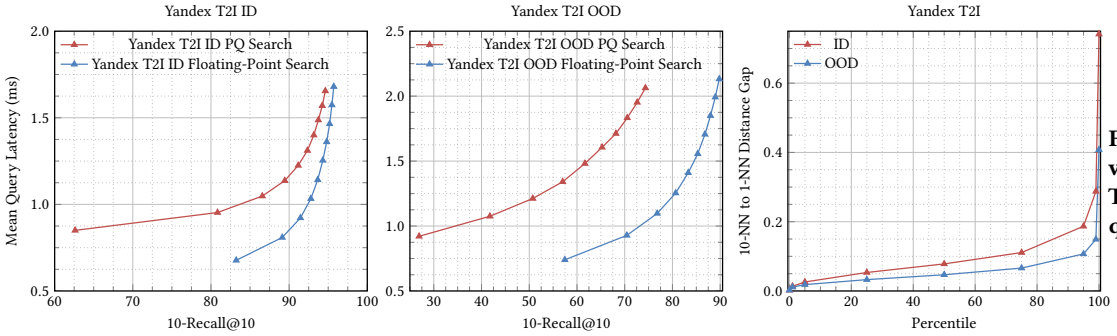


Figure 11: Latency vs. Recall for Yandex T2I on ID and OOD queries at 10M scale.

A natural question to ask is, why do existing PQ variants work well in the ID setting despite this upper-bound minimization approach? It is primarily because of two reasons. Firstly, as the graph traversal, initiated from a far off entry node, converges to a local region around an ID query, the first multiplicative term in Equation 3 progressively becomes smaller, to the extent that it no longer stays relevant. This is because the queries are in close proximity to their nearest base points in this scenario. Hence, the distortions progressively get smaller, and one only needs to focus on minimizing the second multiplicative term. Secondly, the difference in the true distances between a query and its n-th and $(n+1)$-th near neighbors increases significantly, as n gets smaller. This practically ensures that, even with distortions in the distance estimates, the search algorithm isn't easily confused between what appears to be near (based on a distorted estimate), and what is actually near (based on true distance) to a given query.

Extending the above argument, we can identify why existing PQ schemes fail to perform well in the OOD setting. The first multiplicative term in Equation 3 remains dominant for the entirety of the search process, since the query lies far away from any of the actual near points in the graph.

The average distance of a query to a base point is much larger in comparison to the average distance between two base points, or two queries. For high dimensional datasets, this means that for an OOD query, every base point appears to be roughly at the same distance from itself. Hence, the difference in the true distances between a query and its n-th and $(n+1)$-th near neighbors is quite small for any value of n. Both of these phenomena, together, are able to trick the search algorithm into mistaking a far off point for a near neighbor and vice versa, leading to poor recall (**Figure 11**).

A.4 Analysis for Inner Product Metric

For the inner product metric, following a similar approach, we have:

$$\langle q^j, x_i^j \rangle = \langle q^j, x_i^j - \mu + \mu \rangle = \langle q^j, \mu \rangle + \langle q^j, x_i^j - \mu \rangle \tag{10}$$

Thus, chunk-wise error between actual and estimated distance is:

$$\underbrace{\langle q^j, x_i^j \rangle}_{\text{Actual}} - \underbrace{\langle q^j, \mu \rangle}_{\text{Estimated}} = \langle q^j, x_i^j - \mu \rangle \tag{11}$$

Hence Equation 11 depicts the total distortion between the true inner product and the estimate distance. A positive value of the distortion term signifies that the base point appears less suitable to the query than it actually is, and correspondingly, a negative value of distortion term signifies that the base point appears more suitable. We can simplify the above as:

$$\langle q^j, x_i^j - \mu \rangle = \|q^j\| \|x_i^j - \mu\| \cos\theta \tag{12}$$

where, θ is the angle between the two inner product vectors.

As before, we minimize the distortion between the true inner product and the estimate, by considering the following loss objective:

$$\mathcal{L}_{\text{APQ-IP}}(x_i^j, \mu, \mathcal{Q}_{x_i}) = \frac{1}{|\mathcal{Q}_{x_i}|} \sum_{q \in \mathcal{Q}_{x_i}} \begin{cases} \langle q, x_i^j - \mu \rangle, & \text{if } x_i \in \mathrm{H}_{T'}(q) \\ |\langle q, x_i^j - \mu \rangle|, & \text{otherwise} \end{cases} \tag{13}$$

where, $\mathcal{Q}_{x_i}$ and $\mathrm{H}_{T'}(q)$ are obtained for the inner product metric as previously described.

A.5 Limitations of APQ and AOPQ

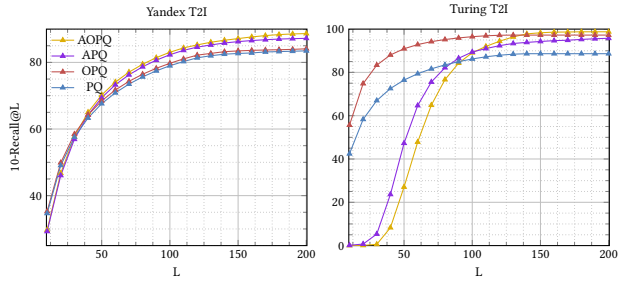


Figure 12: 10-Recall@L scores for T2I datasets at 10M scale.

At a high level, PQ and OPQ aim to minimize the quantization error for all base points with equal priority. Hence, this leads to large distortions on average, which are distributed more or less equally among the base points. APQ and AOPQ aim to minimize the quantization error over a chosen subset of base points, chalked out by their "closeness" to the query sample set. As such, while APQ and AOPQ lead to lower distortions for this chosen subset, they incur much higher distortions, for base points not considered to be "close". APQ and AOPQ are thus unsuitable for linear search routines not utilising re-ranking. Additionally, in the case of linear search with re-ranking, they also seem to underperform at low search budgets (**Figure 12**). This has no bearing for graph-based indices, as graph traversals are highly efficient and only look at a tiny fraction of the indexed dataset, which alleviates this problem.