IVF^2 Index: Fusing Classic and Spatial Inverted Indices for Fast Filtered ANNS

Ben Landrum [1 2] **Magdalen Dobson Manohar** [3] **Mazin Karjikar** [2] **Laxman Dhulipala** [2]

Abstract

The rise of vector embeddings as a crucial tool in search, recommendation, and large language model applications has created significant interest in complex search queries over vectors, such as restricted vector search based on per-vector metadata ("filtered ANNS").

The NeurIPS'23 BigANN competition's Filter track evaluated submissions based on query throughput above a target level of recall on a 10M vector dataset, with binary per-vector metadata (labels), and with query predicates requiring equality on either one or two specified labels for all vectors returned. Existing state of the art approaches for filtered ANNS struggle to perform such 'AND' queries, which require all returned vectors to have a set of specified binary labels.

Perhaps surprisingly, we find that a more combinatorial view of the problem leads to highly efficient solutions, approaching and sometimes even exceeding the throughput of unfiltered search on the full dataset. We present the IVF^2 index, a novel approach to indexing vectors to serve filtered queries which leverages classical and inverted file indices in tandem to dramatically reduce the number of vectors needing to be considered before comparing any of them to the query vector. We demonstrate empirically strong results on the competition dataset, exceeding the throughput of the runner-up submission by a factor of 1.97x and the organizer provided baseline by a factor of 11.58x.

[1]Department of Computer Science, Cornell University, Ithaca, NY, USA [2]Department of Computer Science, University of Maryland, MD, College Park, USA [3]Carnegie Mellon University, Pittsburgh, PA, USA. Correspondence to: Ben Landrum <blandrum@cs.cornell.edu>.

Proceedings of the 1st Workshop on Vector Databases at International Conference on Machine Learning, 2025.

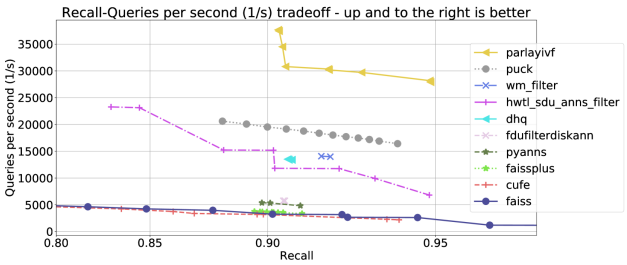


Figure 1. The results from the Filter track of the NeurIPS 2023 Big ANN Benchmarks competition, taken from (Simhadri et al., 2024). IVF^2 is represented here by the 'parlayivf' entry.

1. Introduction

Approximate Nearest Neighbor Search (ANNS) has recently emerged as a fundamental primitive in modern databases, information-retrieval systems, and machine learning. ANNS has numerous applications today due to the development of high-dimensional and high-quality "semantic" learned representations of complex objects such as text (Greene et al., 2022), images (Dosovitskiy et al., 2020), and graphs (Narayanan et al., 2017). State-of-the-art ANNS systems can index billions of vectors while enabling fast approximate nearest-neighbor queries over those vectors (Subramanya et al., 2019; Malkov & Yashunin, 2020), and have been widely deployed in industry to power state-of-the-art search (Jegou et al., 2023; Douze et al., 2024).

In real-world deployments of ANNS, the vectors often come with accompanying metadata, which is typically used to convey some type of categorical information. For example, in a shopping application, the vectors may correspond to learned embeddings of items, and have additional metadata indicating attributes such as their color, the product category, or size. In such applications, users may wish to find objects that are most similar to their query, but also subject to inflexible constraints such as product size, using the categories to *filter* the points that are returned.

This requirement for incorporating categorical information has led to the development of the *Filtered Approximate Nearest Neighbor Search (Filtered ANNS)* problem, which is to retrieve the nearest points in the database for a query

that match a given set of *filters*, i.e., a *predicate* over the points. In the simplest case, which already requires non-trivial algorithmic ideas to solve (Gollapudi et al., 2023), the predicate is to match a single (binary) filter f, i.e., to retrieve the nearest points only among those matching f. In practice, predicates can involve OR filters (disjunction) and AND filters (conjunction), i.e., finding points satisfying $(f_1 \vee f_2 \ldots \vee f_k)$, or points matching $(f_1 \wedge f_2, \wedge \ldots f_k)$. More formally, let n_f be the number of filters associated with points in the database. In Filtered ANNS each point $d_i \in D$ has a set of labels $f_i \subseteq [n_f]$, and a predicate P for the query specifying elements that f_i must have or exclude in order to be a valid nearest neighbor of q.

Previous work addressing the Filtered ANNS problem focuses on building a single *monolithic* index, such as a graph- or partition- based index, and modifying the way the index is built in order to answer filter queries more efficiently. For example, in the case of graph-based indices, Filtered-DiskANN (Gollapudi et al., 2023) takes steps to ensure that the subgraph G_{f_i} corresponding to points satisfying each filter f_i is connected within the overall graph; on the other hand, CAPS (Gupta et al., 2023) adapts a partition-based index by organizing points according to their filter information within each partition. A major issue with such monolithic indices is that there is no straightforward way to handle the combinatorial explosion introduced by supporting AND (conjunction) queries between labels, as well as further arbitrary predicates. The most natural approach to the Filtered ANNS problem would be to build an *inverted index* where filter f_i maps to a posting list of points matching this filter, and route incoming queries to the appropriate posting list. However, all of the prior work assumes that building a separate posting list per label would use a prohibitive amount of memory (Gollapudi et al., 2023; Gupta et al., 2023), due to the fact that points would have to be duplicated over labels, but more catastrophically that each posting list itself would likely require a corresponding spatial index in order to answer queries efficiently.

In this paper, we carefully revisit the classic inverted index solution and the assumption that its memory cost would be prohibitive for filtered search. A key observation underpinning this work is that real-life filter frequencies follow a *power-law distribution*: that is, only a small number of labels f_i correspond to "large" posting lists that require memory-intensive spatial indices in order to answer queries efficiently. The remaining filters, in the "small" case of 10-20 thousand points or fewer, can be efficiently indexed using a flat posting list which is exhaustively searched, and thus requires very little memory to index. Leveraging this insight, in this paper we present the IVF^2 index, a natural and surprisingly powerful approach to solving the Filtered ANNS problem on both AND and OR predicates. IVF^2 is based on carefully combining classical inverted indices with spatial indices, allowing spatial indices constructed on points associated with labels to effectively accelerate filter predicates that were not materialized explicitly at build time.

We used IVF^2 in our winning submission to the NeurIPS 2023 Big ANN Benchmarks competition's Filter track, which exceeded the performance of the runner-up open source submission by a factor of **1.97x** . A full breakdown of the competition and its results can be found in (Simhadri et al., 2024). We test IVF^2 against state-of-the-art existing algorithms, and on two recently released real-world filter datasets and one dataset with synthetically generated filters. We demonstrate strong empirical performance, with both high throughput and high accuracy. On the SIFT-label dataset we are 1.7x faster than the existing state-of-the-art, CAPS (Gupta et al., 2023) at 85% recall, and on the YFCC dataset we are more than 10x faster at 85% recall than the FAISS baseline created for the NeurIPS 2023 Big ANN Benchmarks competition's Filter track (Simhadri et al., 2024). In all cases we show significantly stronger performance than existing baselines. We also provide ablations showing the effects of different optimizations and performance in different filter classes to help illustrate the reasons for the strong performance of IVF^2.

1.1. Contributions

We present the IVF^2 index, a novel approach to indexing vectors with boolean label metadata to serve filtered queries requiring one or two labels. Our contributions include:

(1) We characterize the typical distribution of real-world boolean labels.
(2) We describe the IVF^2 index and motivate its design.
(3) We ablate components of the index and discuss the impact of these ablations on its query performance, build time, and memory consumption.
(4) We compare IVF^2 to two strong open source baselines tuned expressly for the evaluation dataset by their original authors.
(5) We demonstrate that the IVF^2 index can achieve query performance comparable to unfiltered search with a state-of-the-art search graph.

Limitations and Broader Impacts. While search and ranking based on embeddings can be used in unethical ways, this work studies the topic of faster algorithms for existing problems, and does not meaningfully enhance any capacities for unethical use that may already be present in the subject area. The technical limitations of this work are chiefly constrained by available datasets, which are limited in size and in complexity of filter predicates. Given access to larger real-world datasets with more complicated predicates, we could further validate the robustness and generality of our algorithmic ideas.

1.2. Scope and Existing Algorithms

While filtered search is a core component of commercial vector databases such as Pinecone (pin, 2023), Weaviate (wea, 2023), QDrant (Qdrant, 2024), Milvus (Project, 2023), and many more, there is little existing academic work on filtered nearest neighbor search. In this section we summarize the existing academic work based on what types of filter predicates it is tailored to.

OR predicates. Two existing works provide algorithms tailored for the case of a single label or an OR of two or more labels. FilteredDiskANN (Gollapudi et al., 2023) modifies the DiskANN (Subramanya et al., 2019) graph index by building a separate DiskANN graph index for each label and then taking their union to merge them into one graph. They control the degree of the final graph using a filter-aware pruning strategy that attempts to ensure that the subgraph corresponding to each label remains connected. FilteredDiskANN modifies the graph search routine to only visit vertices that satisfy one or more of the labels being searched for; they furthermore provide a (less performant) dynamic version of the algorithm that builds the final index using a filter-aware search and prune on each inserted point. Similarly, the Unified Navigating Graph (UNG)(Cai et al., 2024) constructs a DiskANN graph index over points associated with each label, and uses an additional 'Label Navigating Graph' to connect these constituent graphs to each other based on superset relations between label sets. Since our work is focused primarily on answering AND predicates, we do not directly compare to either approach.

AND predicates. AnalyticDB-V (Wei et al., 2020) supports AND queries as well as arbitrary predicates by retrieving all candidates that satisfy the predicate from a central database and then varying the search strategy based on the cardinality of the set. NHQ (Wang et al., 2022) supports filter queries by modifying the distance function to treat points with label matches as closer together, thus making them more likely to be returned during a filtered search. CAPS (Gupta et al., 2023) answers AND queries by building an IVF index over the dataset and then building a Huffman tree over the filters in each bucket. They search by first selecting buckets closer to the query point and then by using the Huffman trees to quickly identify points that satisfy the filter predicate. For the NeurIPS23 Big ANN Benchmarks competition (Simhadri et al., 2024), the FAISS library (Jégou et al., 2011) provided an IVF-based implementation that used bitmaps to identify elements satisfying the filter predicates in each bucket. In our work, we directly compare against both CAPS, the FAISS library implementation, and one of the top entries in the competition. We do not compare against AnalyticDB-V and NHQ due to low query performance in the former case and lack of code availability in the latter case.

2. Preliminaries

Graph-based ANNS. Graph-based ANNS works by building a directed graph where the vertices correspond to points in the dataset and edges are drawn based on connecting nearby points to each other. A search for the nearest neighbor of a query point q starts at a designated start point s, computes the distance to q from all of s's out-neighbors $N_{\text{out}}(s)$ in the graph, and hops to the closest vertex to q in $N_{\text{out}}(s)$. The algorithm continues searching in this manner until the distance to q ceases to improve, i.e. a local maxima is reached. This *greedy search* or *beam search* can be modified to search for k-nearest neighbors by maintaining a queue of length L of the k nearest neighbors instead of a single nearest neighbor. Some well-known ANNS graph algorithms include DiskANN, HNSW, and NSG (Subramanya et al., 2019; Fu et al., 2019; Malkov & Yashunin, 2020), but many more such algorithms abound in the literature (Muñoz et al., 2019; Fu et al., 2022; Zhang et al., 2022; Lu et al., 2022; Harwood & Drummond, 2016; Dong et al., 2011; McInnes, 2020; Iwasaki, 2016; Iwasaki & Miyazaki, 2018; Boytsov & Naidan, 2013; ann, 2016; Chen et al., 2018; Ren et al., 2020; ope, 2022; n22, 2021; ves, 2022; kat, 2022). Graph-based algorithms are widely regarded to be the state-of-the-art data structure for achieving high recall on datasets in the range of 10 million to 1 billion points (Dobson et al., 2023; Wang et al., 2021).

Vamana Search Graph. Used extensively in the remainder of this paper, the Vamana search graph construction algorithm was introduced as part of DiskANN (Subramanya et al., 2019). It approximates the Sparse Neighborhood Graph from (Arya & Mount, 1993), differing primarily from the construction procedure described there by the addition of a parameter α to further promote sparsity, and restricting candidate edges for a point to those traversed in a query to that point over the existing graph. Given a list of candidate neighbors for a point x sorted by distance to x, we add the nearest candidate as a neighbor, and then remove all candidates which are less than α times closer to x than they are to the newly added neighbor. This process continues until the candidate list is empty.

3. The IVF^2 Index

3.1. Power-Law Distributed Labels

Labels which are *power-law distributed* or more rigorously *Zipfian*, have frequencies distributed in such a way that for any cutoff x, and multiple α, if there are y labels with at least x occurrences, it's expected that there are $\frac{y}{\alpha}$ labels with at least αx occurrences. This pattern is common in real world data, being observed most famously in English-language word frequencies (Zipf, 1949), but also in the populations of cities (Cottineau, 2017), citation counts of scientific pub-

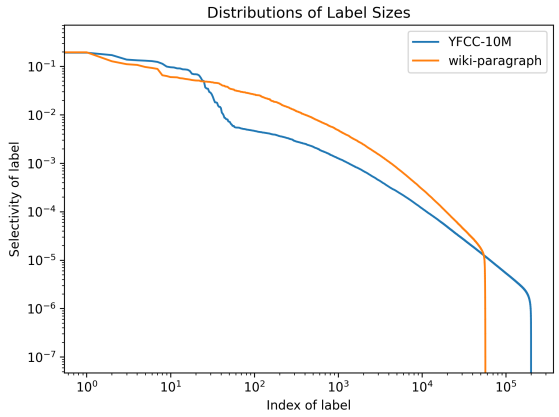


Figure 2. The cumulative density functions of our two datasets with naturally derived labels; a point (x, y) shows that the x-th largest label in a dataset is associated with at least a fraction y of the points.

lications (Silagadze, 1999), and nonzero entries of sparse vector representations of text (Bruch et al., 2023).

Figure 2 shows that this pattern appears in both of our experimental datasets with natural labels, as a perfectly Zipfian distribution would appear linear on log-log axes. This is unsurprising, as both have labels derived from English text corpora, captions on photographs for YFCC-10M and the text of English language Wikipedia articles for wiki-paragraphs. YFCC-10M also includes labels corresponding to administrative regions where a photograph was taken, which also obey this pattern as a downstream effect of the distribution of city sizes.

This distribution of label frequencies motivated the design of the IVF^2 index. Because the vast majority of labels in a dataset can be expected to be small and therefore efficient to search without indexing, we can afford to build more expensive indices for the large cardinality labels.

Table 1. Summary of Notation

Symbol	Description
n	Size of the full dataset
d	Dimensionality of the vectors in the dataset
f_i	A specific label in the dataset
$\|f_i\|$	Number of points associated with label f_i
C	Cutoff size between 'large' and 'small' labels
k	Number of clusters in IVF index
s	Target size of clusters in per-label IVF indices
$C_{\text{bitvector}}$	Cutoff for constructing a bit vector for large labels
$N_{\text{target points}}$	Target number of join candidates from a large label
C_{tiny}	Cutoff size for using bitvector join in AND queries

3.2. Construction

We build an inverted file index over labels, where an index representing the points associated with label f_i is accessible in constant time from a pointer at index i in an array built for this lookup. For each f_i, the index built depends on the number of points $|f_i|$ associated with it. This allows us to reserve the construction of memory and build-time intensive data structures for the largest labels.

This approach of independently indexing each label increases the minimal memory footprint by an amount proportional to the number of point-label connections present in the dataset, as the vectors being indexed are stored *only once*. We define a hyperparameter C representing the cutoff in size between 'large' and 'small' labels. If the memory footprint of the index is not a concern, C would ideally be the point at which more sophisticated indexing no longer offers a speedup over exhaustive search; in practice C is the lowest value allowing the resulting set of indices to fit in memory on the machine used for querying, which is significantly smaller on typical machines.

Small Labels The power-law distribution of label frequencies means that the vast majority of labels f_i will satisfy $|f_i| < C$ (for reasonable values of C) and are associated with few points in the dataset. The points associated with small labels are indexed with a sorted array, which allows both minimal memory footprint and is convenient for the use of a linear join between the points of two labels.

Large Labels For f_i with $|f_i| > C$, we construct an index over the points in the large label f_i with three components:

(1) An IVF index
(2) A Vamana search graph
(3) A dense vector of bits b_i of length n where $\forall j \in [n], b_i[j] = 1 \Leftrightarrow j \in f_i$

IVF Index. We construct an IVF index over the points of f_i with k-means clustering (MacQueen, 1967) initialized by a hierarchical clustering splitting recursively on random hyperplanes. We store an array of associated indices and a centroid in $\mathbb{R}^d$ for each partition. A hyperparameter s is used to determine k for a given f_i, with $k = \lfloor \frac{|f_i|}{s} \rfloor$.

Search Graph. In addition to the IVF index used for AND queries, we construct a Vamana search graph (Subramanya et al., 2019) over the points in f_i. We find that as $|f_i|$ increases, the max degree of a Vamana graph over the points associated with f_i should increase to preserve recall. To better fit graph construction to the size of the label in question, we define *weight classes*: disjoint size regimes above the cutoff with distinct build and search parameters optimized for datasets in that size regime.

Bit Vector. We construct a bit vector encoding the member-

ship of points in a large f_i for fast constant-time membership lookup. The memory footprint of this bit vector is linear in the size of the full dataset, and is the only component of the index which has superlinear memory footprint as the size of the full dataset n increases. As a result, to be more adaptable to large n, we define a separate cutoff $C_{\text{bitvector}}$, below which large labels do not construct a bit vector.

Materialized Join. Often, two large labels will have an intersection with more than C points. In this case, we construct a search graph on the points in the intersection to accelerate queries constrained to the intersection of those labels.

3.3. Querying

Query behavior of the index is determined by the form of the query (single label vs. AND) and the size(s) of the labels referenced by the query

3.3.1. Single-Label Filter

In the case where the filter on a query is only dependent on a single label, a query reduces trivially to a k-NN query on the index for that label.

Small Labels. For f_i with $|f_i| < C$, we only store a sorted array of the indices of points associated with f_i. A k-NN query on such an array is simply an exhaustive search over the points referenced by this array of indices.

Large Labels. For f_i with $|f_i| \geq C$, we have a Vamana search graph over the points associated with f_i built when constructing the index. When queried with beam search, this provides a SOTA k-NN index (Dobson et al., 2023) over the points associated with f_i.

3.3.2. AND Filter.

The AND filter case describes queries which restrict search to points which are in the intersection of f_i and f_j for some distinct i, j. With the exception of a special case where the bit vector of the larger filter is used to accelerate the computation of the intersection, candidates are found independently for each filter, and we exhaustively search the intersection of these sets.

Small Label Join Candidates. Because we only store the indices of the points associated with small labels, the candidates returned for a small label in this case are all points associated with the label, as we have no datastructure for efficiently restricting the candidates by proximity to the query, and want to restrict the candidates by performing the join before doing distance comparisons.

Large Label Join Candidates. To collect join candidates from large labels, we leverage the assumption that the true nearest neighbors matching the predicate will be relatively close to the query vector among the points associated with the larger label. To find candidate points close to the query while minimizing distance comparisons, we compare the query vector to the centroids of the IVF index constructed over the label's points, and add indices of points in the nearest clusters to a sorted array until we have at least $N_{\text{target points}}$ candidates, a query parameter.

Bitvector Join. In the case where an AND query is between a large label and an especially small label of size less than C_{tiny}, a query parameter, the intersection is computed exactly by checking the bitvector associated with the large label for each point associated with the small label. This has the advantages of avoiding the overhead of comparing to the centroids of the large label's IVF index and providing exact results for the query.

Sorted Queries. Batched queries with filters have the convenient property that an ordering which maximizes temporal locality can be computed far more easily than would be possible with vector-only queries. In order to leverage this property, we lexicographically sort queries by the label(s) they constrain search to before processing them. In the parallel setting where we perform queries, this causes naive partitioning of queries into jobs in a work-stealing scheduler to group together queries using the same label-specific index in jobs which will begin in the work queue of the same core. In a serial setting or within a set of queries executed together on a given core, this has the advantage of allowing frequently accessed indices associated with a large filter to remain in L1 cache between queries.

4. Experimental Setup

We run construction and querying for experiments on a 2.10GHz 4 socket Intel Xeon machine with 96 cores and two way hyperthreading, 132 MiB L3 cache, and 1.47 TB of RAM. Querying is done with 8 cores to simulate the smaller machines which would typically be used to serve such an index in a production setting, and construction with the full machine. Statistics on datasets can be found in Table 2.

We run our algorithm and baselines in the big-ann-benchmark framework used for the NeurIPS'23 Practical Vector Search Challenge benchmark competition, which facilitates straightforward comparisons with consistent evaluation and provides implementations of baselines tuned by their authors on the YFCC-10M dataset. When possible, we modify baselines to run on our other dataset, although only the FAISS baseline was amenable to the maximum inner product search used for the wiki-10M dataset.

YFCC-10M. The YFCC-10M dataset consists of CLIP(Radford et al., 2021) embeddings of a 10 million image subset of the YFCC-100M dataset (Thomee et al., 2016). It was released under a CC by 4.0 license for the NeurIPS

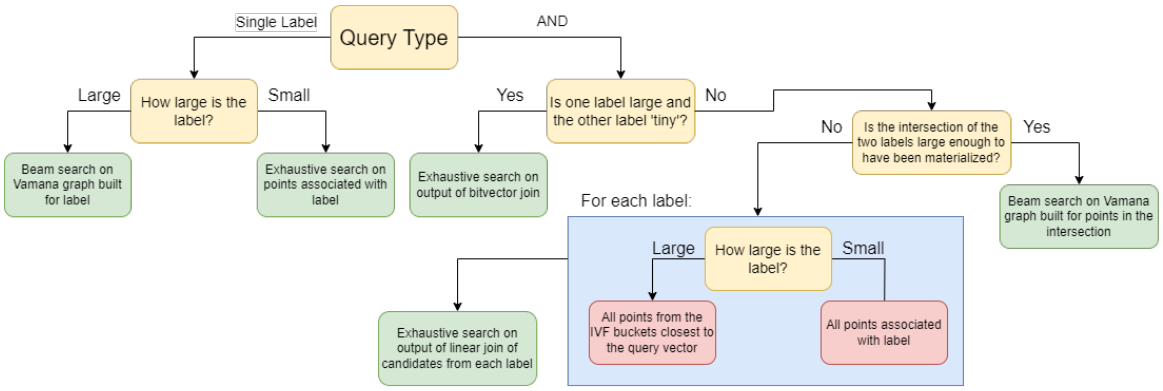


Figure 3. Flowchart illustrating the methods used to resolve queries.

Table 2. Dataset statistics. Num Assoc shows the number of point-label associations in the dataset. 80% Pareto is the portion of labels responsible for 80% of all point-label associations in the dataset. n_q is the number of queries, and $n_\cap$ is the number of AND queries with a filter conditioning on two labels.

Dataset	n	d	n_f	Max $\lvert f_i \rvert$	Mean $\lvert f_i \rvert$	Num. Assoc.	80% Pareto	n_q	$n_\cap$
YFCC-10M	10M	192	200,386	3,386,745	540	108M	0.054	100,000	38,374
wiki-para	10M	768	57,388	2,194,824	4,584	263M	0.091	10,000	5,080
SIFT	1M	128	5	500,254	400,000	2M	0.60	10,000	10,000

2023 Big ANN Benchmarks competition (Simhadri et al., 2024). Labels are derived from metadata associated with the original images, including the year and country in which an image was taken, and keywords associated with the content of the image. Query vectors were generated by embedding held out elements of the dataset, and filters are 1-2 labels which must be present in points returned by the search. Each vector has 192 dimensions, and the values are quantized to unsigned 8 bit integers.

wiki-paragraphs. The wiki-paragraphs dataset was constructed by embedding 35 million paragraphs from English Wikipedia with the cohere.ai `multilingual-22-12` embedding model (cohere.ai, 2022), which is released under an Apache 2.0 License. Labels were generated by taking all words which appear at least 1000 times in the full corpus, removing syntactic words such as 'the' which appear in virtually all paragraphs in the dataset and are not representative of a realistic search scenario, and annotating each paragraph with the words it contains. from the resulting set Query vectors were generated by embedding paragraphs from Simple English Wikipedia with the same model. For each vector, both single-label and AND queries were generated, and chosen at random for an approximately equal proportion of each in the dataset. Each filter was chosen at random from the words present in the paragraph, and were confirmed to each match at least 10 vectors from a randomly sampled subset of 1 million data vectors. Each vector has 768 dimensions, each of which is represented with a 32 bit float.

SIFT-label. In order to compare with CAPS (Gupta et al., 2023), we also provide a limited evaluation of ParlayANN on the SIFT dataset (Muja & Lowe, 2009), which is released under a CC0 license. Following the procedure used in the aforementioned work, we generated two random labels for each data point.

5. Experiments

We validate the design of the IVF^2 index by showing strong query performance in each of the distinct cases we resolve queries to, and showing comparatively that specific optimizations had a measurable positive effect on the performance of the index. We experimentally demonstrate strong query performance on all 3 datasets we evaluate against, with stronger QPS across different recall regimes than the baselines. We also show that in a high-recall regime, the IVF^2 index achieves QPS closer to a state-of-the-art unfiltered baseline than to comparable filtered baselines, and beyond 95.24% recall achieves higher QPS than the unfiltered baseline.

5.1. Optimizing the index

We implemented several practical optimizations to improve index performance and address weaknesses. For each such optimization, we recall a short description of its function and justify its effectiveness experimentally.

Weight classes. As discussed in 3.2, we use different graph construction parameters depending on the size of the label being indexed. This is primarily an extension of the broader ethos of the index: smaller sets require less indexing because they are easier to search. Because many labels fall under the smaller weight classes, reducing the size of these graphs by decreasing the maximum degree significantly reduces the memory consumption of the index.

To validate the effectiveness of this choice, we compared the query performance of an index that builds all graphs with the parameters of the largest weight class to a modification of the full IVF^2 index which uses 2.5% less memory and builds 2.1% faster on YFCC-10M. For all Pareto optimal configurations of the index without weight classes, there exists a configuration of the high memory index which achieves a QPS and recall which are both strictly greater.

Bitvector. As described in 3.2, we accelerate joins between our smallest labels and large labels with a bitvector associated with each large label storing the membership of each point from the full dataset therein. While the memory consumed by this optimization is asymptotically worse than other components of the index,

Sorted Queries. Applying the sorted queries optimization discussed in paragraph 3.3.2 significantly improves performance by improving the temporal locality of the queries, as can be seen in Figure 4. We note that we observed similar improvements on the wiki-10M dataset.

Materialized Joins. The materialized joins described in 3.2 strengthen what would otherwise be a comparatively low-recall regime, and justify their increased memory footprint. In Figure 11, we show the effect of including materialized-joins, and find that they provide several orders of magnitude higher QPS at the same level of recall.

Construction. The index takes 19.05 minutes to build on YFCC-10M, and has a memory footprint of 18.046 GB, including 2.866 GB for the dataset. On wiki-paragraphs, the index takes 196.4 minutes to build and has a footprint of 154.361 GB, including 32.13 GB for the dataset.

5.2. QPS vs Recall in Different Query Regimes

We validate the design of our index by demonstrating that it is competitive in virtually every case to which a query can be resolved. Fig. 10 shows that of the distinct query regimes, only large $\times$ large has lower QPS at a given recall than any of the filtered baselines on YFCC-10M, and all but the small $\times$ large queries in the 'bitvector' regime are also similarly higher performing than the unfiltered Vamana baseline.

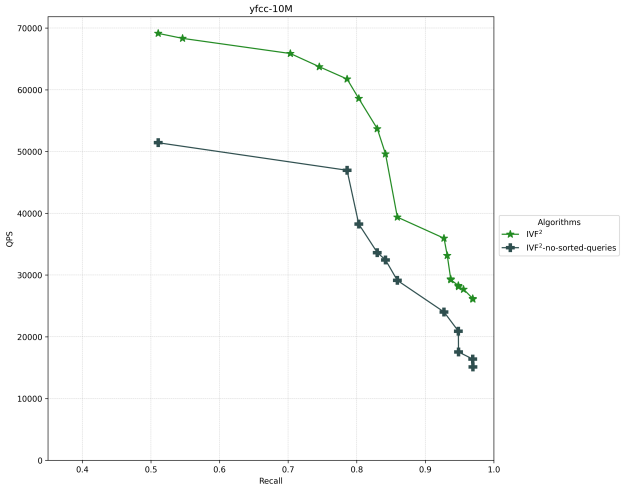


Figure 4. Effect on the QPS vs. recall curves for the IVF^2 index when using the sorting queries optimization to improve temporal locality on the YFCC-10M dataset.

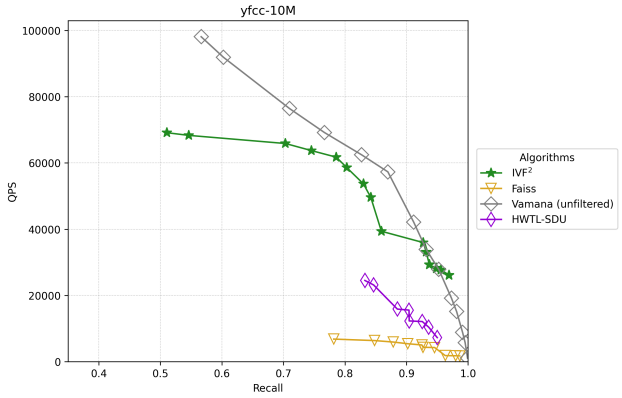


Figure 5. QPS vs. recall for IVF^2 and filtered baselines compared to a Vamana graph serving unfiltered queries.

5.3. Comparisons against other algorithms

We compare against a FAISS baseline (released under an MIT license) (Jegou et al., 2023) tuned for our setting by the organizers of the big-ann-benchmarks competition (Simhadri et al., 2024), and HWTL-SDU (accessed through its submission to the NeurIPS 2023 Big ANN Benchmarks Competition (Simhadri et al., 2024), which is MIT licensed), a top open source submission to the big-ann-benchmarks competition. For comparisons to unfiltered search, we use the implementation of DiskANN from ParlayANN (Dobson et al., 2023), which is MIT licensed.

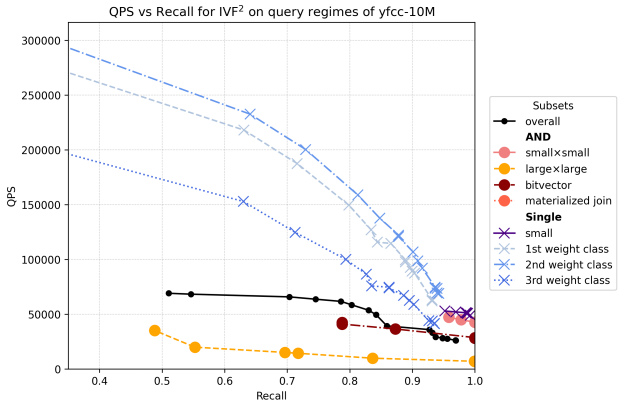


Figure 6. QPS vs. recall for IVF^2 on queries falling in each independently handled regime of queries. See Fig. 8 for the same analysis on the Wikipedia dataset.

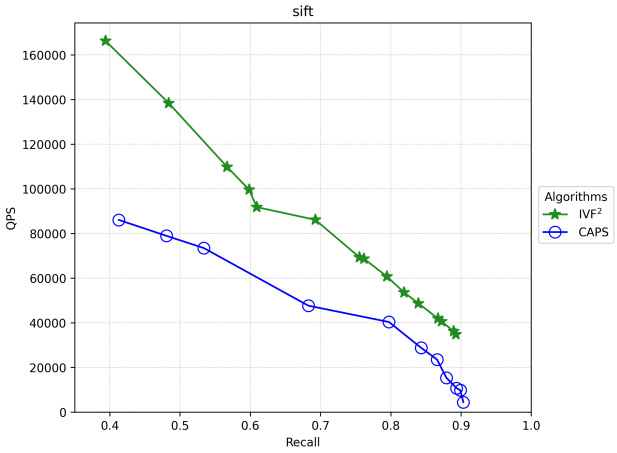


Figure 7. QPS vs. recall of IVF^2 and CAPS on the SIFT dataset.

A good-faith effort was made to compare against published baselines. We do not compare to NHQ (Wang et al., 2022), as the GitHub repo containing their code is no longer public, and a request for code to compare against was not answered. We attempted to compare to CAPS(Gupta et al., 2023), which is released under an Apache 2.0 license, on our datasets, but their codebase is configured for Euclidean distances only, so we were unable to use it on the wiki-paragraphs dataset, and we were not able to run CAPS on YFCC-10M, due to the codebase not supporting points with variable numbers of labels. We thus provide comparisons with CAPS only on the SIFT-label dataset, which can be seen in Fig. 7. We achieve uniformly higher QPS and recall, with **78.32%** higher QPS at 85% recall. We attribute the higher performance of IVF^2 as compared to CAPS to a few chief attributes of our design: first, by creating an inverted index over the filter labels, IVF^2 prunes the search space and thus the number of required distance comparisons before the search begins. CAPS, on the other hand, identifies candidates via spatial search *before* searching for appropriate filter membership, which can lead to a need to evaluate a larger number of candidates. Secondly, while CAPS treats each filter label uniformly, we vary the algorithm used based on the size of the filter, allowing us to save compute on smaller labels and reserve it for larger labels.

5.4. Comparison to unfiltered search

In principle, because the filters provide a powerful way to select vectors relevant to a query, it is reasonable to imagine that there must exist a regime where using an inverted index to answer filtered queries is faster than answering unfiltered queries on the same dataset. A comparison of our algorithm and the filtered baselines to a Vamana search graph[1] performing unfiltered search on the same dataset and query vectors can be seen in Fig. 5. While the other filtered methods are not competitive with the unfiltered search, IVF^2 has comparable QPS for recall above 0.75, and **exceeds the QPS of the Vamana graph for recall above 95.24%**

Further plots comparing our method with unfiltered search are presented in the supplemental material.

6. Conclusion

We present IVF^2, a novel and highly competitive index for filtered ANNS based on an "IVF"-style approach that builds a carefully chosen index on each label. Perhaps surprisingly, our evaluation showed that filtered search can be as fast, or even faster than state-of-the-art unfiltered search on the same dataset, and suggests that the restrictions imposed by filtered ANNS can actually improve the efficiency of retrieval in practice. Compared to CAPS, a state-of-the-art existing filtered ANNS baseline, IVF^2 achieves 78% higher QPS at a high recall level of 85%. Compared with an existing baseline for filtered ANNS from FAISS, IVF^2 obtains 6.63× higher throughput at 90% recall. An interesting direction for future work is to study how to support arbitrary boolean predicates in a filtered ANNS solution; we believe that ideas from the design of IVF^2 will be relevant in such systems.

References

Kgraph: A library for approximate nearest neighbor search. Webpage, 2016. URL https://github.com/aaalgo/kgraph.

N2. Webpage, 2021. URL https://github.com/kakao/n2.

[1] R = 64, L = 128, α = 1.15

Vald: A highly scalable distributed vector search engine. Webpage, 2022. URL https://github.com/vdaas/vald.

Opensearch k-nn. Webpage, 2022. URL https://github.com/opensearch-project/k-NN.

vespa. Webpage, 2022. URL https://github.com/vespa-engine/vespa.

Pinecone: Vector database for vector search, 2023. URL https://www.pinecone.io/.

Weaviate: The ai native vector database, 2023. URL https://weaviate.io/.

Arya, S. and Mount, D. M. Approximate nearest neighbor queries in fixed dimensions. In *ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 271–280. ACM/SIAM, 1993.

Boytsov, L. and Naidan, B. Engineering efficient and effective non-metric space library. In *Similarity Search and Applications (SISAP)*, volume 8199 of *Lecture Notes in Computer Science*, pp. 280–293. Springer, 2013.

Bruch, S., Nardini, F. M., Ingber, A., and Liberty, E. Bridging dense and sparse maximum inner product search. *arXiv preprint arXiv:2309.09013*, 2023.

Cai, Y., Shi, J., Chen, Y., and Zheng, W. Navigating Labels and Vectors: A Unified Approach to Filtered Approximate Nearest Neighbor Search. *Proceedings of the ACM on Management of Data*, 2(6):1–27, December 2024. ISSN 2836-6573. doi: 10.1145/3698822. URL https://dl.acm.org/doi/10.1145/3698822.

Chen, Q., Wang, H., Li, M., Ren, G., Li, S., Zhu, J., Li, J., Liu, C., Zhang, L., and Wang, J. *SPTAG: A library for fast approximate nearest neighbor search*, 2018. URL https://github.com/Microsoft/SPTAG.

cohere.ai. Cohere wikipedia-22-12-en embeddings. https://huggingface.co/datasets/Cohere/wikipedia-22-12-en-embeddings, 2022. Accessed: 2024-03-30.

Cottineau, C. Metazipf. a dynamic meta-analysis of city size distributions. *PLOS ONE*, 12(8):1–22, 08 2017. doi: 10.1371/journal.pone.0183919. URL https://doi.org/10.1371/journal.pone.0183919.

Dobson, M., Shen, Z., Blelloch, G. E., Dhulipala, L., Gu, Y., Simhadri, H. V., and Sun, Y. Scaling graph-based ANNS algorithms to billion-size datasets: A comparative analysis. *CoRR*, abs/2305.04359, 2023. doi: 10.48550/ARXIV.2305.04359. URL https://doi.org/10.48550/arXiv.2305.04359.

Dong, W., Charikar, M., and Li, K. Efficient k-nearest neighbor graph construction for generic similarity measures. In Srinivasan, S., Ramamritham, K., Kumar, A., Ravindra, M. P., Bertino, E., and Kumar, R. (eds.), *Proceedings of the 20th International Conference on World Wide Web (WWW)*, pp. 577–586. ACM, 2011.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P.-E., Lomeli, M., Hosseini, L., and Jégou, H. The faiss library, 2024.

Fu, C., Xiang, C., Wang, C., and Cai, D. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc. VLDB Endow.*, 12(5):461–474, 2019.

Fu, C., Wang, C., and Cai, D. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(8):4139–4150, 2022.

Gollapudi, S., Karia, N., Sivashankar, V., Krishnaswamy, R., Begwani, N., Raz, S., Lin, Y., Zhang, Y., Mahapatro, N., Srinivasan, P., Singh, A., and Simhadri, H. V. Filtered-diskann: Graph algorithms for approximate nearest neighbor search with filters. In Ding, Y., Tang, J., Sequeda, J. F., Aroyo, L., Castillo, C., and Houben, G. (eds.), *Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, pp. 3406–3416. ACM, 2023.

Greene, R., Sanders, T., Weng, L., and Neelakantan, A. New and improved embedding model. Webpage, 2022. URL https://openai.com/blog/new-and-improved-embedding-model.

Gupta, G., Yi, J., Coleman, B., Luo, C., Lakshman, V., and Shrivastava, A. CAPS: A practical partition index for filtered similarity search. *CoRR*, abs/2308.15014, 2023. doi: 10.48550/ARXIV.2308.15014. URL https://doi.org/10.48550/arXiv.2308.15014.

Harwood, B. and Drummond, T. Fanng: Fast approximate nearest neighbour graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5713–5722, 2016.

Iwasaki, M. Pruned bi-directed k-nearest neighbor graph for proximity search. In *Similarity Search and Applications (SISAP)*, volume 9939 of *Lecture Notes in Computer Science*, pp. 20–33, 2016.

Iwasaki, M. and Miyazaki, D. Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data. *CoRR*, abs/1810.07355, 2018. URL http://arxiv.org/abs/1810.07355.

Jégou, H., Douze, M., and Schmid, C. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, 2011.

Jegou, H., Douze, M., Johnson, J., Hosseini, L., Deng, C., and Guzhva, A. Faiss wiki. Webpage, 2023. URL https://github.com/facebookresearch/faiss/wiki.

Lu, K., Kudo, M., Xiao, C., and Ishikawa, Y. Hvs: Hierarchical graph structure based on voronoi diagrams for solving approximate nearest neighbor search. *Proc. VLDB Endow.*, 15(2):246–258, 2022.

MacQueen, J. Some methods for classification and analysis of multivariate observations. In Cam, L. M. L. and Neyman, J. (eds.), *Berkeley Symposium on Mathematical Statistics and Probability, 1967*, volume 5.1, pp. 281–297, 1967. URL https://api.semanticscholar.org/CorpusID:6278891.

Malkov, Y. A. and Yashunin, D. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(4):824–836, 2020.

McInnes, L. Pynndescent for fast approximate nearest neighbors. Webpage, 2020. URL https://pynndescent.readthedocs.io/en/latest/.

Muja, M. and Lowe, D. G. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proceedings of the Fourth International Conference on Computer Vision Theory and Applications (VISAPP)*, pp. 331–340. INSTICC Press, 2009.

Muñoz, J. A. V., Gonçalves, M. A., Dias, Z., and da Silva Torres, R. Hierarchical clustering-based graphs for large scale approximate nearest neighbor search. *Pattern Recognit.*, 96, 2019.

Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., and Jaiswal, S. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

Project, T. M. Milvus: open source vector database built for scalable similarity search, 2023. URL https://milvus.io/.

Qdrant. Qdrant: High-performance, massive-scale vector database for ai. https://github.com/qdrant/qdrant, 2024. Accessed: 2024-04-29.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning transferable visual models from natural language supervision, 2021.

Ren, J., Zhang, M., and Li, D. HM-ANN: efficient billion-point nearest neighbor search on heterogeneous memory. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

Silagadze, Z. K. Citations and the zipf-mandelbrot's law, 1999.

Simhadri, H. V., Aumüller, M., Ingber, A., Douze, M., Williams, G., Manohar, M. D., Baranchuk, D., Liberty, E., Liu, F., Landrum, B., Karjikar, M., Dhulipala, L., Chen, M., Chen, Y., Ma, R., Zhang, K., Cai, Y., Shi, J., Chen, Y., Zheng, W., Wan, Z., Yin, J., and Huang, B. Results of the Big ANN: NeurIPS'23 competition, September 2024. URL http://arxiv.org/abs/2409.17424. arXiv:2409.17424 [cs].

Subramanya, S. J., Devvrit, F., Simhadri, H. V., Krishnaswamy, R., and Kadekodi, R. Diskann: Fast accurate billion-point nearest neighbor search on a single node. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pp. 13748–13758, 2019.

Thomee, B., Shamma, D. A., Friedland, G., Elizalde, B., Ni, K., Poland, D., Borth, D., and Li, L.-J. Yfcc100m: the new data in multimedia research. *Communications of the ACM*, 59(2):64–73, January 2016. ISSN 1557-7317. doi: 10.1145/2812802. URL http://dx.doi.org/10.1145/2812802.

Wang, M., Xu, X., Yue, Q., and Wang, Y. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proc. VLDB Endow.*, 14(11):1964–1978, 2021.

Wang, M., Lv, L., Xu, X., Wang, Y., Yue, Q., and Ni, J. Navigable proximity graph-driven native hybrid queries with structured and unstructured constraints. *CoRR*, abs/2203.13601, 2022. doi: 10.48550/ARXIV.2203.13601. URL https://doi.org/10.48550/arXiv.2203.13601.

Wei, C., Wu, B., Wang, S., Lou, R., Zhan, C., Li, F., and Cai, Y. Analyticdb-v: A hybrid analytical engine towards query fusion for structured and unstructured data. *Proc. VLDB Endow.*, 13(12):3152–3165, 2020. doi: 10.14778/3415478.3415541. URL http://www.vldb.org/pvldb/vol13/p3152-wei.pdf.

Zhang, J., Ma, R., Song, T., Hua, Y., Xue, Z., Guan, C., and Guan, H. Hierarchical satellite system graph for approximate nearest neighbor search on big data. *ACM/IMS Trans. Data Sci.*, 2(4), 2022.

Zipf, G. K. *Human Behaviour and the Principle of Least Effort*. Addison-Wesley, 1949.

A. Additional QPS vs. Recall Plots

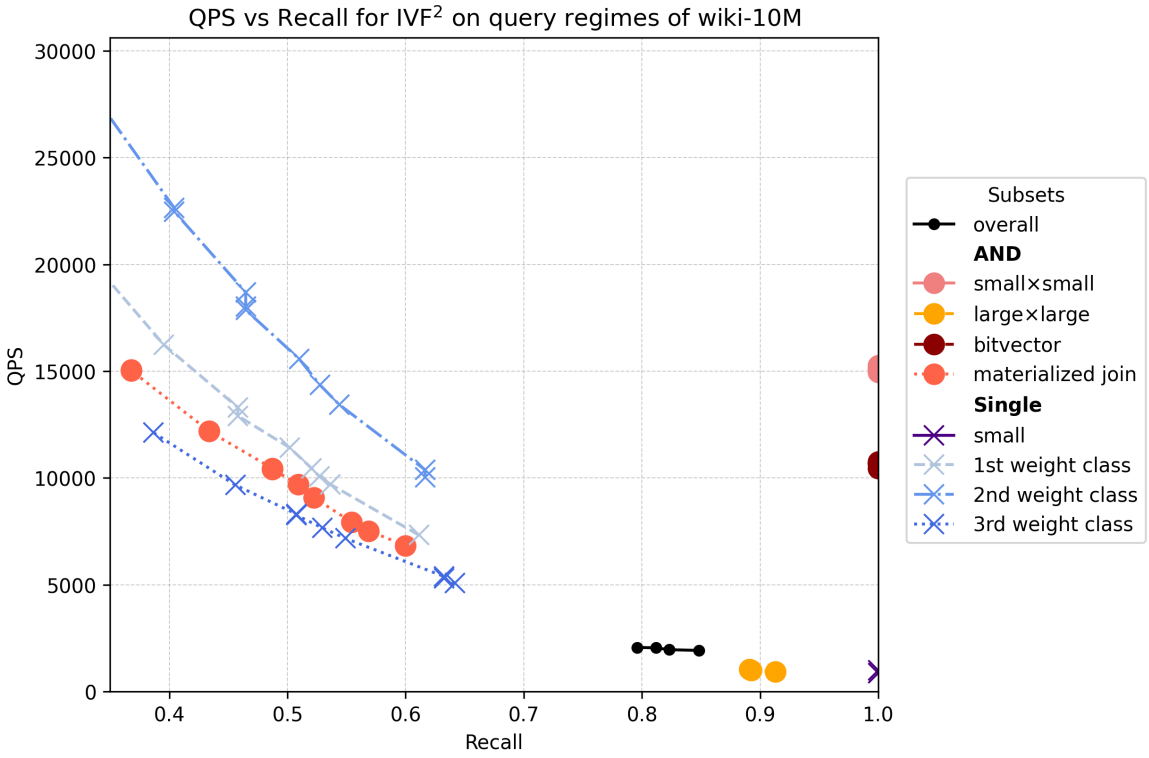
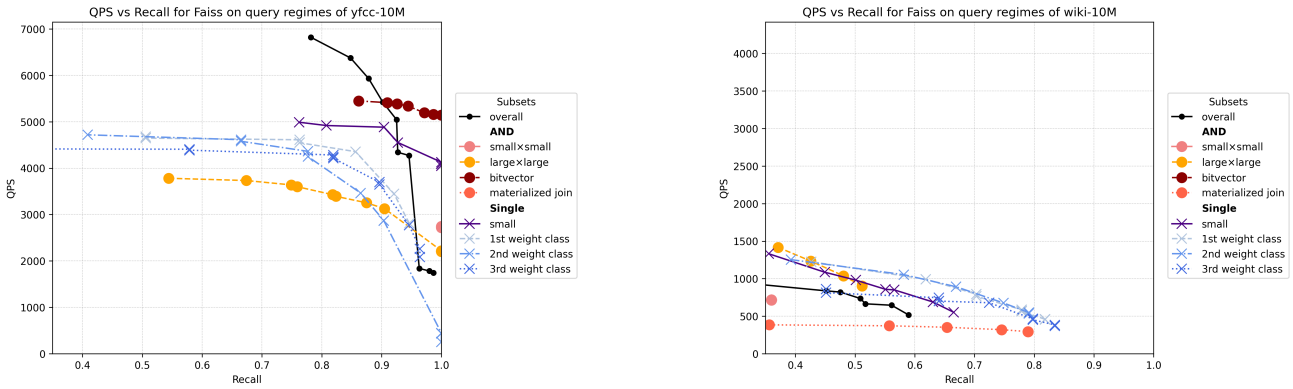


Figure 8. QPS vs. recall for IVF^2 on queries falling in each independently handled regime of queries.



(a) YFCC dataset

(b) Wikipedia dataset

Figure 9. QPS vs. recall of the Faiss baseline on the query cases IVF^2 differentiates, provided to show that differences in performance across query cases are primarily caused by the design of the index instead of variable query difficulty.

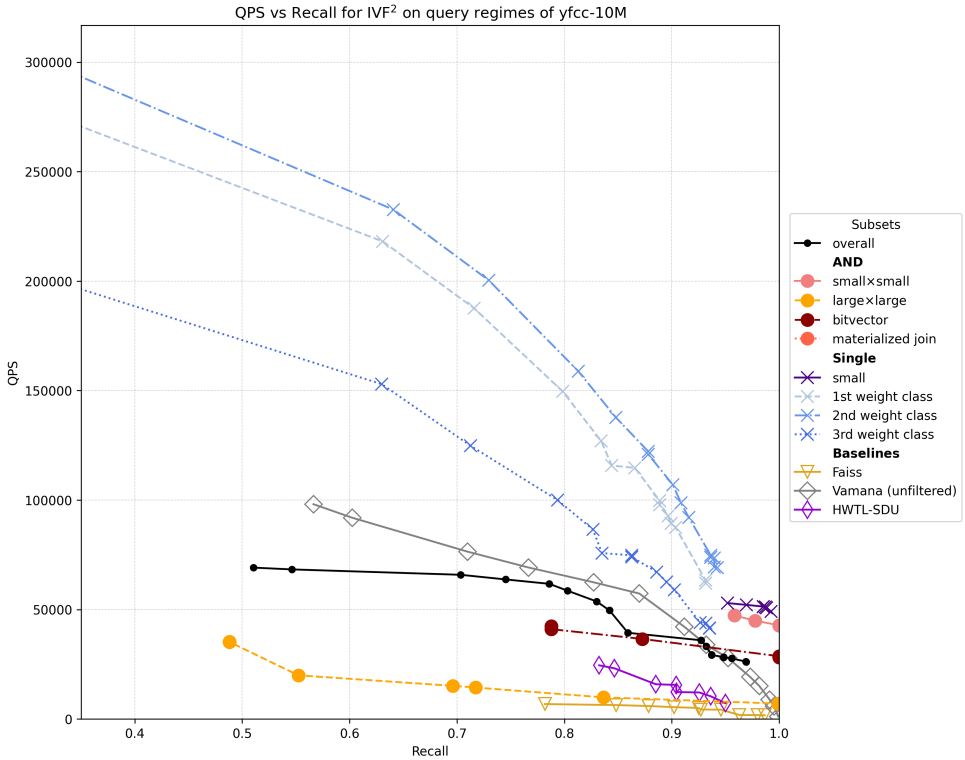


Figure 10. Comparison of IVF^2 on distinct query regimes of YFCC-10M to filtered and unfiltered baselines.

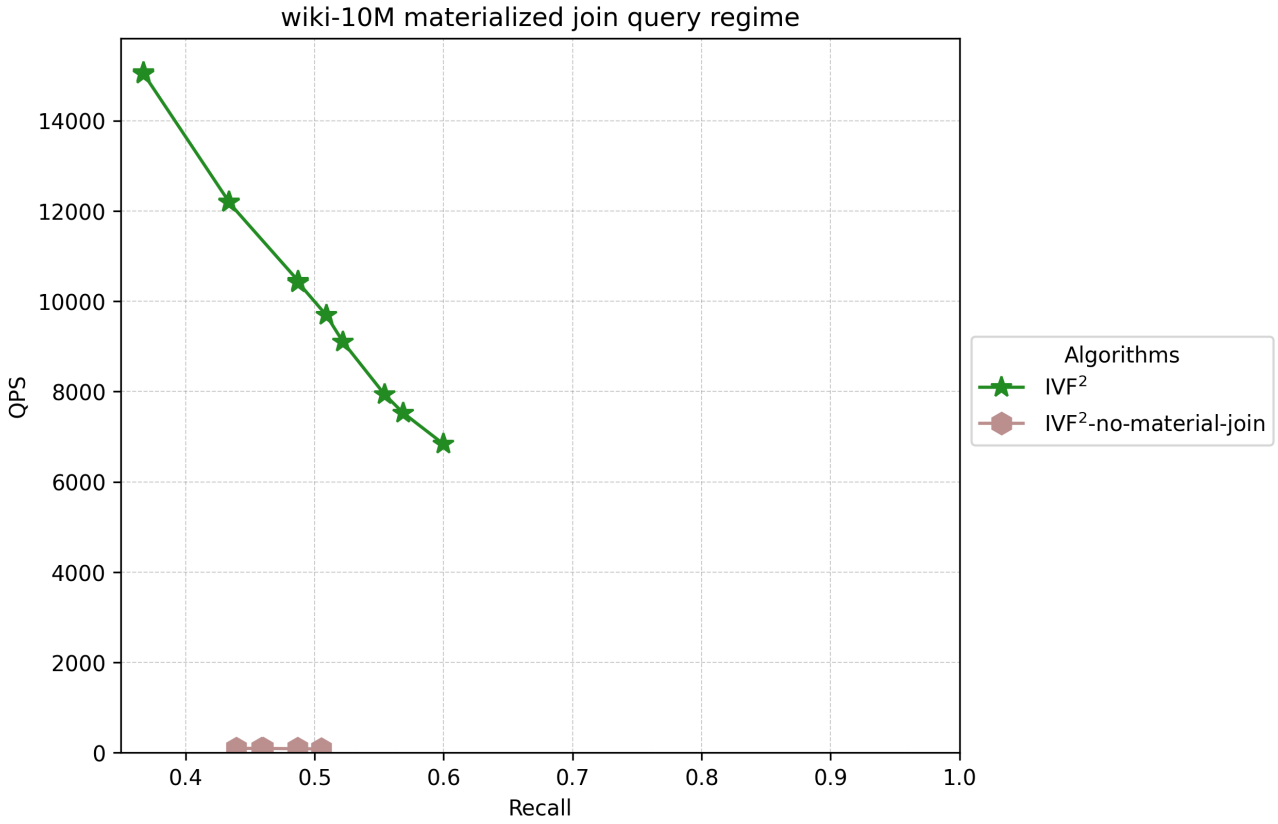


Figure 11. QPS vs. recall of the full IVF^2 index vs an ablation with no materialized joins on the query regime of the wiki-10M dataset which is handled by materialized joins. There is no equivalent plot for the YFCC-10M dataset, as none of the provided queries fall within the materialized join regime.