An Experimental Survey on Algorithms for Approximate Nearest Neighbor Search with Attribute Filtering (Appendix)

CCS Concepts

• **Information systems → Retrieval efficiency**.

Keywords

Approximate Nearest Neighbor, Filtering, Survey

ACM Reference Format:

. 2018. An Experimental Survey on Algorithms for Approximate Nearest Neighbor Search with Attribute Filtering (Appendix). In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Appendix 1: Complexity

In some cases, complexity analysis is prohibitive [1], as no algorithm can guarantee sub-linear query efficiency in the worst case. Nevertheless, we can still analyze the additional time and memory overhead introduced by filtering techniques, relative to the underlying ANN search algorithm. In this section, we focus on graph-based methods, including Filtered-DiskANN, AIRSHIP, NHQ, SeRF, ACORN, β-WST, iRangeGraph, and UNIFY.

Table 1 presents our complexity analysis, where Γ denotes the selectivity of a query. To avoid the intricacies of detailed complexity discussions for each base index, we adopt the following notations: the indexing complexity is represented as n^α, typically with $1 < \alpha < 2$, and the query complexity as n^λ, where $0 < \lambda < 1$. The index size for graph-based methods is denoted as Mn, where M is a hyper-parameter that controls the degree of each vertex. For all HNSW-based algorithms, the base indexing complexity is $n\log(n)$, and the query complexity is $\log(n)$ [2].

For any algorithm that adjusts the pruning strategy, the impact on indexing complexity is typically minimal. Consequently, the indexing complexity remains n^α for methods such as AIRSHIP, VG in Filtered-DiskANN, NHQ, SeRF, and UNIFY. However, in the case of SVG within Filtered-DiskANN—which merges multiple subgraphs into a single graph—the indexing time increases to Ln^α, where L denotes the total number of attribute values.

Filtered-DiskANN offers two indexing strategies. The Filtered Vamana Graph retains the original indexing complexity, introducing no significant overhead. In contrast, the Stitched Vamana Graph builds a separate subgraph for each categorical attribute value, resulting in an overall indexing complexity of $O(Ln^\alpha)$, where L is the number of distinct categorical values. Despite this, its memory complexity remains Mn, constrained by the maximum degree parameter M.

Table 1: Complexity of Graph-based FANN Algorithms

Algorithm	Index Time Complexity	Query Time Complexity	Memory Complexity
FDiskANN-VG	n^α	$(\Gamma n)^\lambda$	Mn
FDiskANN-SVG	Ln^α	$(\Gamma n)^\lambda$	Mn
NHQ	n^α	n^λ	Mn
SeRF	$nM^2\log(n)$	$(\Gamma n)^\lambda$	$Mn\log(n)$
DSG	$nM^2\log(n)$	$(\Gamma n)^\lambda$	$Mn\log(n)$
ACORN	$n\log(n)$	$\log(\Gamma n)$	Mn
β-WST	$n^d\log(n)$	$(\Gamma n)^\lambda$	$Mn\log(n)$
iRangeGraph	$n^d\log(n)$	$(\Gamma n)^\lambda$	$Mn\log(n)$
UNIFY	$n\log(n)$	pre-:Γn joint-:$\log(\Gamma n)$ post-:$\log(n)$	$L'MSn$

NHQ and ACORN do not significantly change the way they construct index, so their indexing, query time complexity, and memory complexity stay unchanged.

For SeRF, determining the upper bound timestamp incurs no additional cost in any aspect. However, to perform arbitrary range queries like $r_k = [od(l_k), od(u_k + 1)]$, SeRF searches for the top K neighbors($\log(n)$ time complexity) for each vector ($O(n)$) at for any $od(l_k)$ that $od(l_k) < od(u_k + 1)$(takes about $O(n)$). This leads to a worst-case time cost of $O(n^2\log(n))$ and a memory cost of $O(Mn^2)$. Fortunately, the average index time cost is $O(M^2n\log(n))$ and the memory footprint is $O(Mn\log(n))$.

2 Appendix 2: Experiment Settings

Algorithm Settings.

In this study, we have gathered the latest algorithms for both range and label-based FANN to evaluate their performance. Our primary focus is on analyzing the advantages and disadvantages of their filtering architectures, which is why most traditional ANN systems are excluded from our experiments. Below, we list all the algorithms considered in our analysis.

Overall configuration. To make the experiment compares equally, we uses the same setting as much as possible for all algorithm. We use $M = 40$ and $ef\_construction = 1000$ for all graph based algorithms. For IVFPQ methods, we take $ncentroids = 4\sqrt{N}$ and $partition\_M = d/2$. But Both Milvus-IVFPQ and Faiss-IVFPQ failed at 1% and 0.1% selectivity in SIFT and Spacev, so in these cases, we take $partition\_M = d$ to ensure the they can fetch 90% recall.

Faiss. Faiss uses `is_member()` to check whether or not a vector's attribute matches our restriction. To define `is_member()` for all

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

vectors and all queries, it is necessary to compare query restrictions with all attributes. However, it is inefficient and unnecessary to compare each item. Specifically, `is_member()` is used only when ANN index scans on the vector, which is unnecessary to check others. So we exclude the computation time for building `is_member()` for each query, but only take ANN search time into our consideration. To accelerate its computing efficiency, we enable AVX2 to accelerate L2 distance computation. In Faiss HNSW, We further static the overall distance computation, instead of the original the number of bottom layer computation.

Besides, we implement the brute force computation for range Filtering ANN search. Instead of using `is_member()`, we checks whether its attribute fits query range.

In our experiment, we use brute-force search to generate the ground truth. Faiss-HNSW and Faiss-IVFPQ are used as baselines for comparison with the filtering ANN algorithms.

ACORN. ACORN is designed based on faiss library, but it designed a high efficient `is_member()` function, which simply stores true or false for each item. Like Faiss, We enable AVX128 acceleration, and overlook the computation overhead of `is_member()`. We use it for both label and range query since it supports arbitrary FANN query tasks. ACORN uses $M\gamma$ to present its $ef\_construction$, so we take $\gamma = ef\_construction/M = 25$.

Milvus. We use Milvus 2.5.9 Standalone version in our experiments, with Milvus_HNSW and Milvus_IVFPQ representing Milvus Filtering ANN. The default partition size is set to 64. Since Milvus is a highly integrated ANN system, it is not possible to directly obtain Comparisons Per Query (CPQ) for HNSW queries. As a result, we report Query Per Second (QPS) as the performance metric.

Filtered DiskANN. We include the Filtered Vamana Graph (VG) and Stitched Vamana Graph (SVG) in our experiments, using $\alpha = 1.2$ as recommended.

NHQ. We include NHQ_nsw and NHQ_kgraph in our experiments, as these two methods demonstrate the best performance in their respective papers and align closely with the core ideas of this work. Note that NHQ_kgraph relies on more than 9 hyperparameters, including L, $iter$, S, R, $RANGE$, PL, B, kg_M, and L_search, making it quite complex. For consistency, I set $RANGE = M$, $PL = efconstruction$, and $Lsearch = efsearch$, while the remaining parameters are $L = 100$, $iter = 12$, $S = 10$, $R = 300$, $B = 0.4$, $kg\_M = 1$.

SeRF. SeRF uses M differently; it stores pruned neighbors as segmented neighbors, meaning it's not necessary to set M as large as in other methods. Therefore, we assign $M = 8$ for SeRF as recommended, while both ef_max and $efconstruction$ are set to 1000. For all algorithms, constructing graph index will sacrifice the accuracy. This paper does not guarantee the accuracy of building index in parallel, but it is unfair to compare them in different indexing parallelization, so we enable the same index scale for them. Besides, we enable SSE L2 computation acceleration for SeRF.

DSG DSG operates similarly to SeRF and shares the same hyperparameters, so we set M and ef_max to the same values as those used for SeRF. Similar to SeRF, we enable parallel index building and SSE distance computation acceleration.

β-WST β-WST offers several methods for constructing a BST and querying on them. In this paper, we focus on the super-optimized post-filtering (WST_opt) and Vamana tree (WST_vamana) methods, as they demonstrate competitive performance. Additionally, these methods use the $split\_factor$ to control the splitting scale and the $shift\_factor$ to manage the overlap size. We set $split\_factor = 2$ to align the scale with that of $iRangeGraph$, and $shift\_factor = 0.5$ as recommended.

iRangeGraph. iRangeGraph uses the fewest hyperparameters, requiring only M, $ef\_construction$ for index construction. Therefore, we set these to the same values as the global ones. We enable SSE acceleration for iRangeGraph.

UNIFY. UNIFY requires B to set the number of slots inside the index, we set it to 8 as recommended. We enable combined filtering method for unify, setting $low\_threshold = 5\%$ and $high\_threshold = 50\%$, which means if the selectivity of a query is lower than 5%, we use skip table to find results; if it is between 5% and 10%, we use hybrid filtering (the core algorithm of UNIFY) to perform ANN search; if the selectivity is larger than 50%, we use post filtering to find the results. To estimate hybrid filtering performance on low and high selectivity, we also enables UNIFY-hybrid to search for any selectivity using joint-filtering, and UNIFY-CBO to search using pre-/joint-/filtering.

3 Appendix 3: Algorithm Performance on Different Selectivity Ranging From 1% to 100%

We evaluate the performance of the algorithms across a wide range of selectivity levels, from 1% to 100%, to comprehensively assess their behavior. The average number of Comparisions Per Query (CPQ) is used as one of the primary performance metric. This metric effectively reflects the quality of graph-based Filtering ANN indices, as all algorithms are designed to minimize computational overhead. Importantly, CPQ abstracts away implementation-specific factors such as the exact distance computation or filtering techniques. Therefore, a lower CPQ indicates better performance.

3.1 Experiments for range filtering ANN search

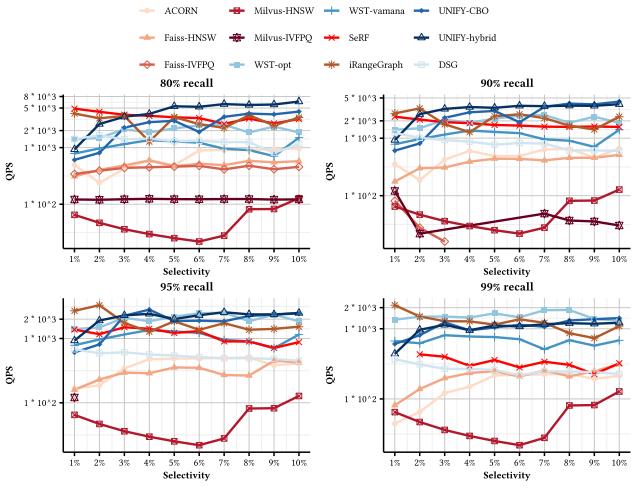


Figure 1: QPS for range query with selectivity from 1% to 10% in SIFT

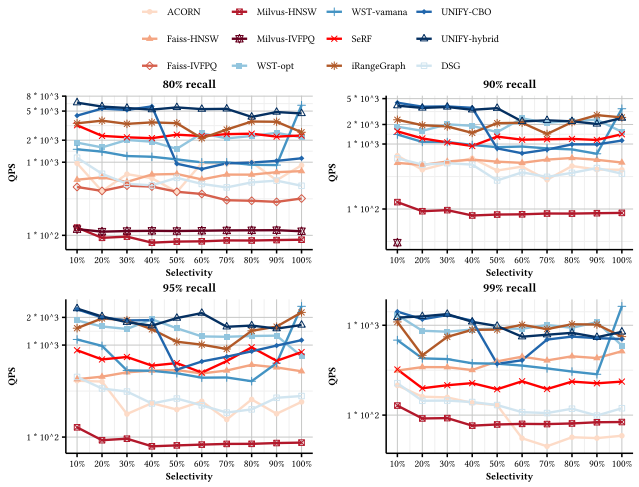


Figure 2: QPS for range query with selectivity from 10% to 100% in SIFT

Figure 1 presents the QPS results for range ANN search with selectivity levels from 1% to 10%, while Figure 2 extends the analysis to selectivity levels from 10% to 100%. Overall, most algorithms exhibit a consistent performance trend across both QPS and CPQ metrics—namely, higher QPS generally corresponds to lower CPQ, indicating more efficient computation.

In Figure 1, UNIFY-hybrid, WST-opt, and iRangeGraph demonstrate the best performance, with UNIFY-hybrid slightly outperforming the others. In contrast, as shown in Figure 2, WST-vamana achieves the highest QPS at higher selectivity levels, benefiting from the elimination of multi-subgraph searches. Meanwhile, WST-CBO shows a decline in performance when selectivity exceeds 50%, due to the overhead introduced by its post-filtering strategy.

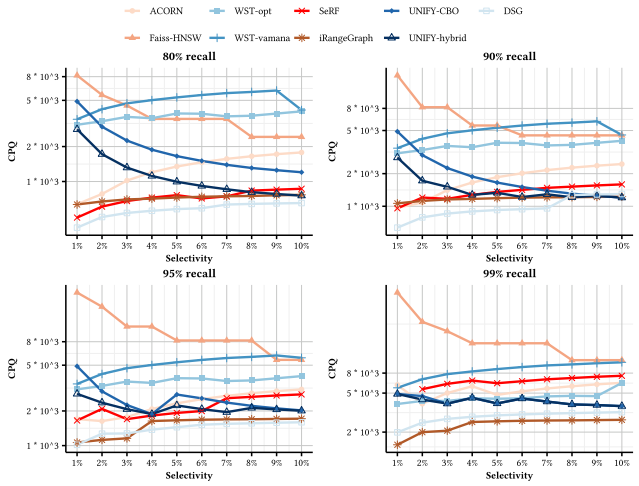


Figure 3: CPQ for range query with selectivity from 1% to 10% in SIFT

Figure 3 and Figure 4 shows similar but stable performance report. iRangeGraph achieved the best performance instead of UNIFY,

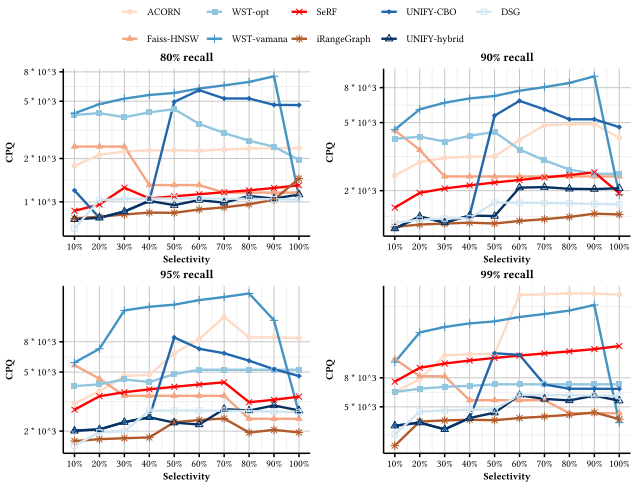


Figure 4: CPQ for range query with selectivity from 10% to 100% in SIFT

mainly because of the different implementation in terms of code details.

One different is that DSG achieved significant better performance than QPS showed. This mainly because the extra cost on edge selection. Even though SeRF have the same strategy to DSG, it has less edges than that, making it more efficient. Similar case apply to ACORN, which searches two-hop neighbors to find suitable edges, making it significantly inefficient at 1% selectivity, but its CPQ is still low and comparable.

3.2 Experiments for label filtering ANN search

We conduct our experiment on the selectivity from 1% to 100% to estimate their performance. Figure 5, 6, 7 and 8 shows our experiment on those methods.

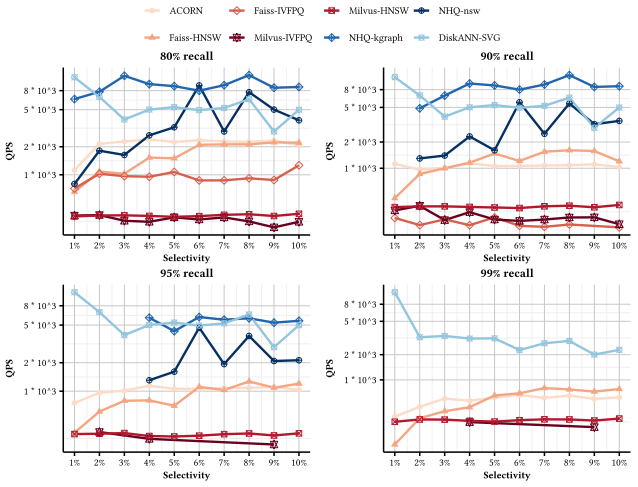


Figure 5: QPS for label query with selectivity from 10% to 100% in SIFT

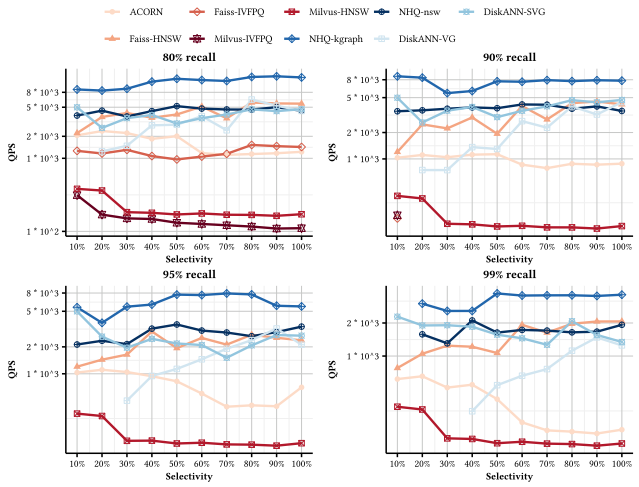


Figure 6: QPS for label query with selectivity from 10% to 100% in SIFT

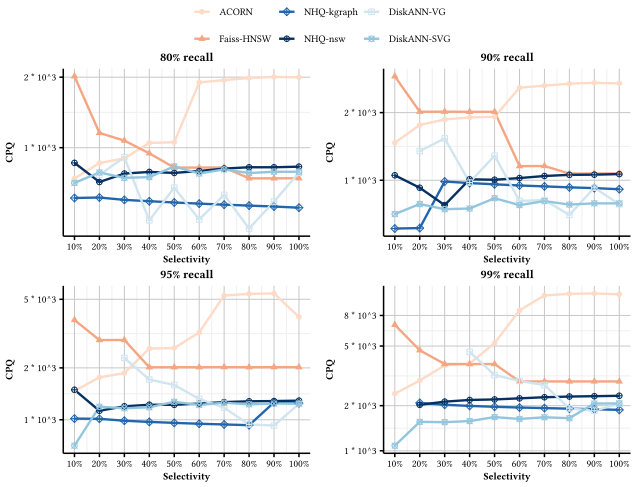


Figure 8: CPQ for label query with selectivity from 10% to 100% in SIFT

NHQ-kgraph shows the best performane across all selectivity. While Faiss-IVFPQ achieved better QPS at 1% selectivity, showing IVF-based method outstanding tolerance to low selectivity.

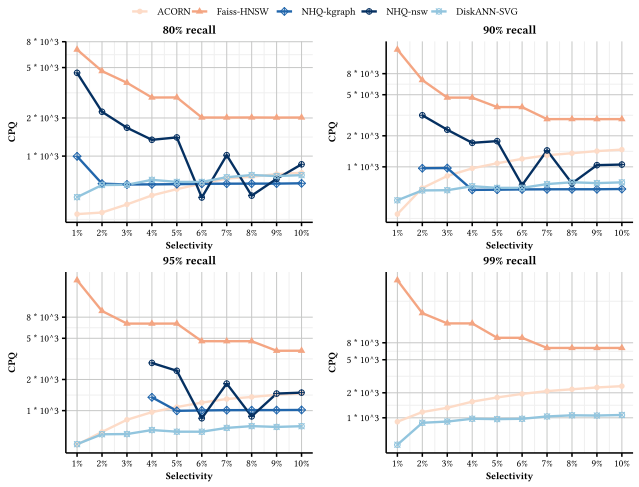


Figure 7: CPQ for label query with selectivity from 1% to 10% in SIFT1

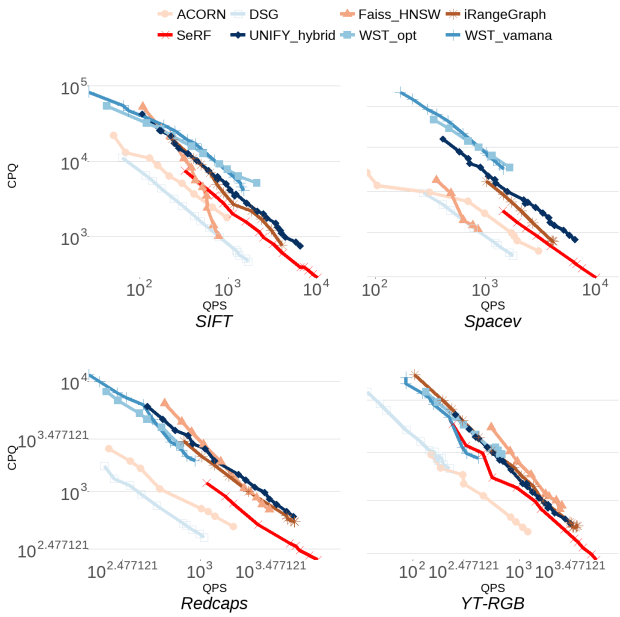


Figure 9: CPQ vs QPS at 80% recall and 10% selectivity

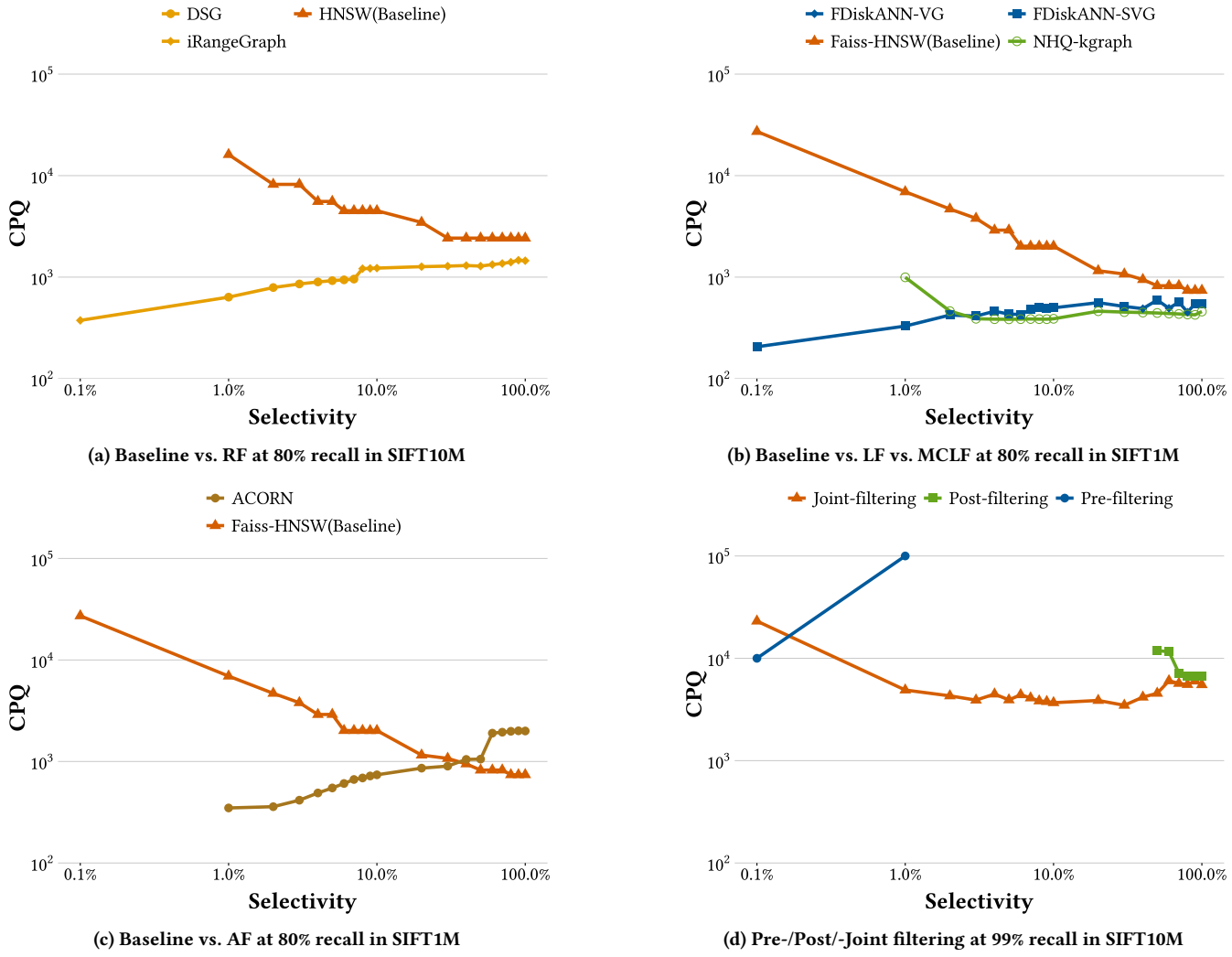
4 Appendix 4: CPQ vs. QPS

Figure 9 shows the relationship between CPQ and QPS in different configurations(for example, different ef_search value). All algorithms show negative power relationship. All algorithms are parallel to each other, indicating that CPQ directly reflect the performance of QPS. Not that comparing CPQ-QPS between different algorithms are invalid since they are not guaranteed to get the same recall at the same QPS. Besides, CPQ provides more stable metric on all graph-based experiments.

Even if QPS and CPQ are colsely connected in certain selectivity, it is not guaranteed to be comparable across algorithms and selectivity levels.

5 Appendix 5: Comparision

Figure 10 shows the comparison of baseline (Faiss-HNSW) and the best performance in range filtering (RF), label filtering (LF), multi cardicality label filtering (MCLF) and arbitrary filtering (AF), and the comparison among pre-/post-/joint-filtering methods. For all



(a) Baseline vs. RF at 80% recall in SIFT10M

(b) Baseline vs. LF vs. MCLF at 80% recall in SIFT1M

(c) Baseline vs. AF at 80% recall in SIFT1M

(d) Pre-/Post/-Joint filtering at 99% recall in SIFT10M

Figure 10: Baseline comparison for different filtering ANN scenarios with selectivity from 10% to 100% in SIFT. The lower CPQ the better

filtering ANN methods, the greater selcivity is, the closer performance will be. For range and label filtering cases, the filtering ANN algorithms is shown significant better performance than baseline.

In figure 10a, What is out of our expectation is, DSG performs well with the selectivity locates at 1%~10%. To analyse its reason, we take 9 into consideration. For the same ef_search, DSG got lower QPS and CPQ than other methods(DSG's line is significantly parallel but lower than other methods). This indicates that even though DSG constructs a large index, less edges are legal for a search range. In other words, the edges are strictly classified, making search task performs well at low selectivity. However, DSG failed to achieve competitive performance with extremely small and large selectivity, mainly due to the low quality of its connection.

In figure 10b, we split Filtered DiskANN methods and NHQ methods apart, because they focus on different filtering scenario. NHQ failed in 0.1% recall, because its heuristic distance estimation method failed when its connection is no longer guaranteed by graph index. DiskANN Stitched shows outstanding performance thorough out all selectivities, showing the effective improvement on constructing subgraphs for all labels.

However, arbitrary filtering performance is still limited, especially when the selectivity exceeds 50%. This mainly because ACORN's pruning method drops RNG connectivity and scalability, making searching less efficient when most one-hop edges can be used.

In figure 10d, all methods are based on UNIFY. Pre-filtering is linear scan, post-filtering is search on UNIFY complete index and then filter out matched candidates, while joint-filtering is UNIFY filtering algorithm. In SIFT10M, pre-filtering works only when selectivity is extremely small(less than 0.5%, but this value varies when dataset scale changes). While post-filtering get a relative effective when selectivity is larger than 70%, which is different than suggested setting in UNIFY(10% and 50% respectively). In conclusion, Joint-filtering

Table 2: CPQ comparison of different entry point selection strategies in SIFT10M (the original algorithm serves as baseline for its variants)

Algorithm	Selectivity			
	1%	10%	50%	100%
UNIFY	**default**	**default**	**default**	**default**
UNIFY-middle	-0.14%	0.10%	1.09%	2.21%
UNIFY-left	-0.14%	0.10%	1.09%	2.21%
UNIFY-right	-0.14%	0.10%	1.09%	2.21%
SeRF	**default**	**default**	**default**	**default**
SeRF-left	0.00%	0.00%	0.00%	0.00%
SeRF-right	0.00%	0.00%	0.00%	0.00%
DSG	**default**	**default**	**default**	**default**
DSG-left	0.00%	0.00%	0.00%	0.00%
DSG-right	0.00%	0.00%	0.00%	0.00%

has been always better than post-filtering, while pre-filtering is better when we need almost 100% recall, otherwise joint-filtering is better.

6 Appendix 6: entry point experiments

Inspired by SeRF and DSG, which overlook the hierarchical structure and perform search directly on the bottom layer, we investigate the role of hierarchy in the context of filtering ANN. In this section, we adjust UNIFY to select entry points directly from the bottom layer, aiming to analyze the impact of hierarchical design in filtering scenarios. Specifically, we seek to verify two aspects: (1) the influence of hierarchical traversal from top entry points down to the bottom layer, and (2) the effect of selecting different entry points within the bottom layer itself.

Inspired by SeRF and DSG, We define three different methods for selecting entry points from a sorted vector list V, given a range query q_k. Let the matched subset of V_k be located within the indices $[l, r]$, such that the size of the range is $|V_k| = r - l + 1$. The three methods of choosing entry points are:

$$EP = \begin{cases} \{v_{l+i\times|V_r|/ep}\} & \text{Middle} \\ \{v_{l+i}\} & \text{Left} \\ \{v_{l+|V_r|-i}\} & \text{Right} \end{cases} \quad (1)$$

Table 2 presents our entry point selection experiments, where UNIFY (short for UNIFY-hybrid) is evaluated under different conditions. The results demonstrate a significant performance impact when avoiding heretical entry points at high selectivity. Notably, as selectivity decreases to 10%, UNIFY actually benefits from omitting heretical entry points.

On the other hand, as long as the entry point lies within the queried subset, the method of selecting entry points from the bottom layer has minimal impact. Both UNIFY, SeRF, and DSG's -left and -right entry points exhibit negligible performance differences.

7 Appendix 7: Search Parameters

Unlike traditional ANN, filtering ANN must handle varied selectivity, leading to searches over subsets of different sizes. As a result, the search parameter varies with selectivity, an aspect that has not been thoroughly discussed in previous studies.

In this appendix, I introduce the search parameters that achieved the best performance for each algorithm. For IVF-based methods, we provide *nprobe*; for graph-based methods, we provide ef_search. For algorithms that do not use ef_search, we specify L_search for NHQ-kgraph and Filtered DiskANN, and $Beam\_size$ for β-WST.

Table 3 shows the part of hyper-parameters that reached our best results. Milvus and Faiss methods show unstable parameter choice, where the higher selectivity, the lower value they choose. This case apply to ACORN and DiskANN.

Besides, the larger dataset, the larger parameter value. Besides, Youtube-RGB shows the hardest case since all algorithms have to apply the large parameter to achieve the recall.

SeRF, DSG show stable hyper-parameter choice, which is a strong advantage. However, iRangeGraph and WST-opt have to assign larger hyper-parameter to larger selectivity, which is opponent to Milvus and Faiss, because it have to search on larger subindex.

UNIFY have to assign ef_search and AL to search, making its hyper-parameter more unstable.

8 Appendix 8: M in SeRF

Figure 11 shows that hyper-parameter M in SeRF has limited influence, while larger M even get worse at low selectivity.

Figure 12 shows that the low recall at SIFT is no relationship with stitching operation, but only related to the natural quality of Vamana Graph, which can not guarantee the quality of each subgraph, so the stitched index is of low quality too.

9 Appendix 9: Thread number to SeRF and DSG

Figure 13 shows that thread number show little effect to their performance.

10 Appendix 10: Recall/QPS for All Algorithms

References

[1] Piotr Indyk and Haike Xu. 2023. Worst-case performance of popular approximate nearest neighbor search implementations: Guarantees and limitations. *Advances in Neural Information Processing Systems* 36 (2023), 66239–66256.

[2] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009

Table 3: Search parameters for range filtering ANN algorithm

Algorithm	Selectivity															
	SIFT				Spacev				Redcaps				Youtube-RGB			
	0.1%	1%	10%	50%	0.1%	1%	10%	50%	0.1%	1%	10%	50%	0.1%	1%	10%	50%
Milvus-IVFPQ	100	1	30	20	300	150	80	50	80	20	20	20	30	20	20	20
Milvus-HNSW	18	60	10	10	12	150	40	80	40	60	40	40	200	200	60	300
IVFPQ	300	150	80	50	800	300	150	150	200	50	20	20	80	50	20	20
HNSW	-	300	80	30	-	400	100	40	-	300	60	18	-	1000	300	200
ACORN	-	100	60	80	-	100	100	150	-	80	18	40	-	300	60	300
SeRF	-	150	80	35	-	400	400	300	-	100	40	40	-	-	-	300
DSG	-	40	80	80	-	150	150	150	-	18	15	20	-	40	60	60
WST-vamana	12	20	12	15	18	1000	40	40	15	10	10	10	12	20	600	800
WST-opt	12	20	20	40	10	20	18	20	15	18	10	15	20	40	200	300
UNIFY	-	-	20	20	-	-	18	40	-	-	10	40	-	-	40	40
UNIFY-hybrid	9	18	20	20	10	12	18	40	8	10	10	10	10	20	40	60
iRangeGraph	15	20	40	80	40	20	60	60	10	12	10	18	12	60	150	200

Table 4: Search parameters for label filtering ANN algorithm

Algorithm	Selectivity															
	SIFT				Spacev				Redcaps				Youtube-RGB			
	0.1%	1%	10%	50%	0.1%	1%	10%	50%	0.1%	1%	10%	50%	0.1%	1%	10%	50%
Milvus-IVFPQ	150	150	30	20	200	300	200	150	50	20	30	20	20	20	20	10
Milvus-HNSW	1000	20	40	20	18	15	40	40	18	12	12	10	10	200	150	100
IVFPQ	300	300	80	50	800	300	150	100	200	50	20	20	100	50	20	20
HNSW	-	300	80	40	-	500	80	40	-	300	40	18	-	1000	300	200
ACORN	-	80	40	60	-	400	100	150	-	60	18	40	-	1000	60	300
DiskANN	-	-	-	150	-	-	-	1000	-	-	-	100	-	-	-	-
DiskANN-Stitched	300	-	-	-	20	40	60	40	20	20	20	20	20	1200	1400	1000
NHQ-kgraph	-	100	100	100	-	-	100	100	-	-	100	100	-	-	100	100
NHQ-nsw	-	-	-	100	-	-	1400	200	-	-	-	60	-	-	1200	1200

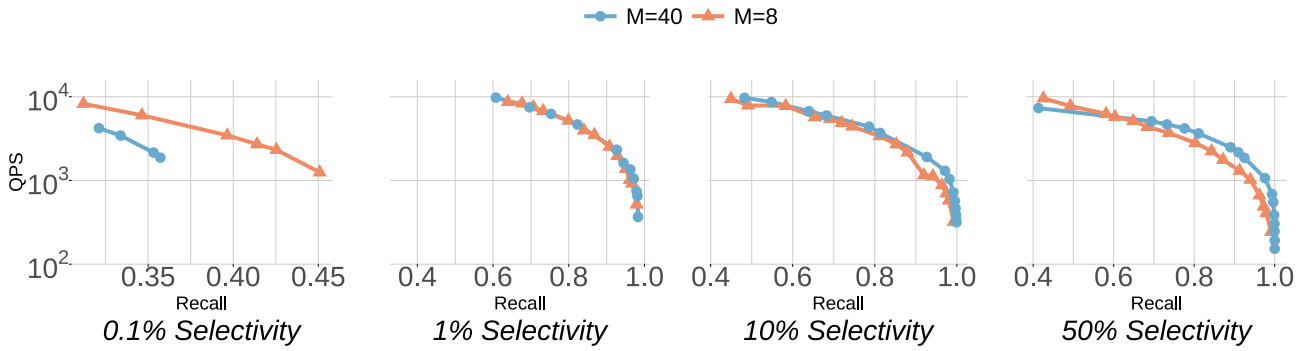


Figure 11: SeRF recall/qps in Redcaps with different M

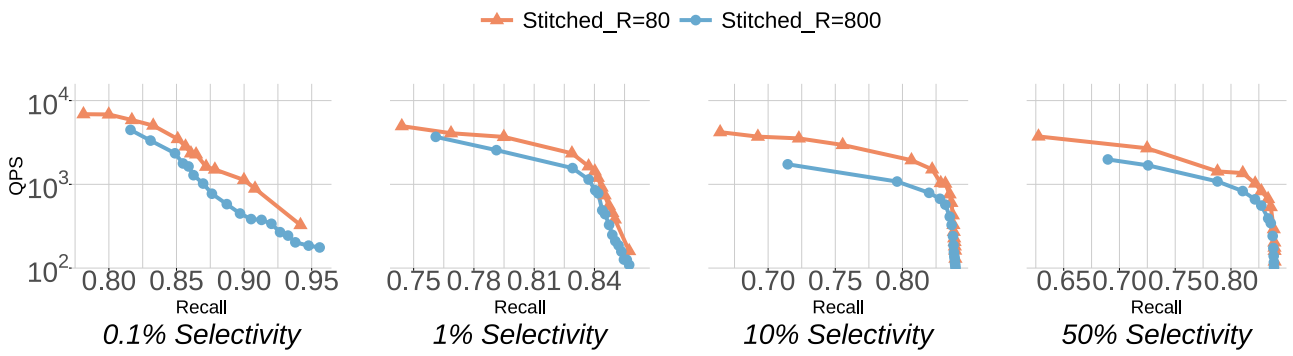


Figure 12: Filtered DiskANN Stitched recall/qps in SIFT with different Stitched_R

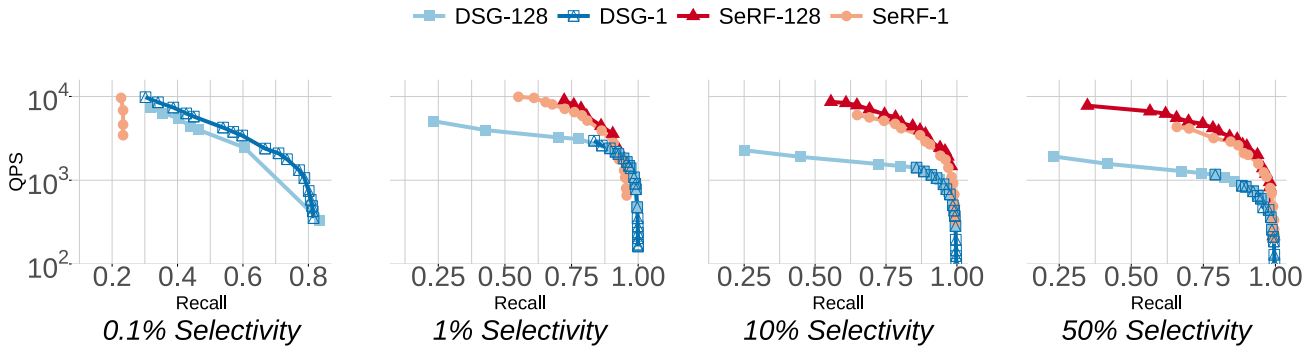
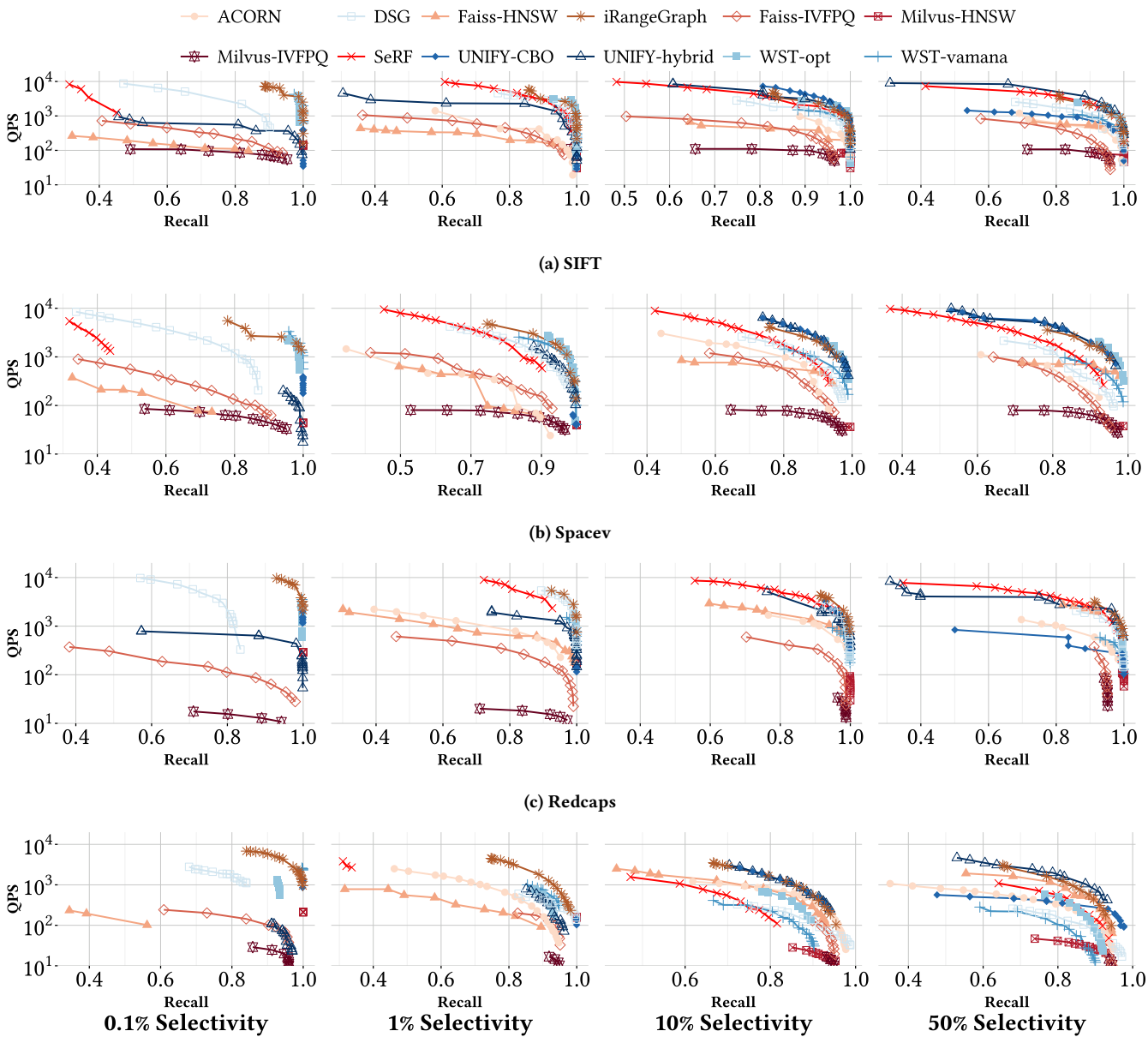


Figure 13: SeRF and DSG recall/qps in redcaps with different thread number



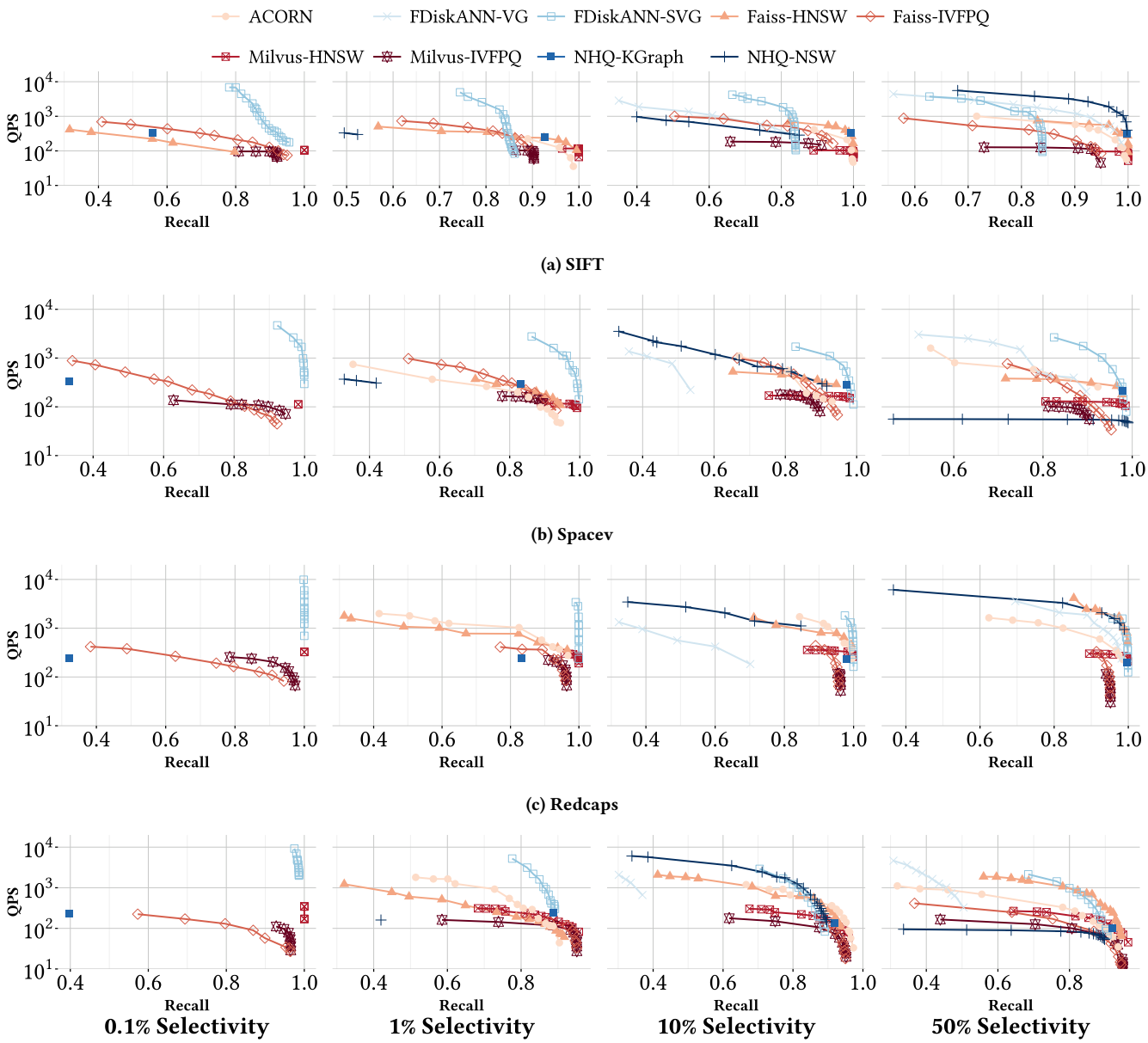
(a) SIFT

(b) Spacev

(c) Redcaps

(d) Youtube-RGB

Figure 14: Recall/QPS for range filtering ANN algorithms.



(a) SIFT

(b) Spacev

(c) Redcaps

(d) Youtube-RGB

Figure 15: Recall/QPS for label filtering ANN algorithms.