Exploring the Effects of Metadata Distribution on Filtered Approximate Nearest Neighbor Search

Kyung Jae Lee
University of Toronto

Jiongan Mu
University of Toronto

1 Abstract

In this paper, we investigate the impact of metadata distribution on the performance of filtered approximate nearest neighbor search in vector databases. By generating multiple variants of the SIFT1M [6] dataset, each augmented with a synthetic "popularity" attribute sampled from various statistical distributions, we analyze recall and latency of filtered queries for fixed selectivity values across these distributions. Our study considers systems employing pre-filtering and post-filtering strategies (ElasticSearch [4] and Chroma [3], respectively), as well as VBase [14] which uses a novel query method specifically designed for relational queries. Our experiments reveal that filtered query performance is primarily influenced by filter selectivity, with metadata distribution having minimal impact. However, for discrete distributions with concentrated probability mass, such as the Zipfian distribution, the presence of duplicate metadata values can distort the effective selectivity, even at the specified threshold. This distortion of the effective selectivity can cause significant deviations from the expected performance trend at that selectivity level. Additionally, we discuss the architectural nuances of the evaluated systems to explain some results that digress from the overall trend.

We make our experiments reproducible at: https://github.com/dlrudwo1269/w25_csc2233_project.

2 Introduction

Modern AI applications leverage embedding models to transform various types of data into dense vector representations [1, 10, 12]. These vectors serve as the backbone for downstream tasks such as search [7] and recommendation [8]. To adhere to the strict latency requirements of such tasks, the vectors are typically stored in a vector database, where approximate nearest neighbor search (ANNS) is performed to efficiently retrieve the top-K most relevant results. As applications leveraging top-K ANNS evolve, there is a growing need to support more complex query types, including filtered queries, which return only the results satisfying the specified metadata constraints. However, the introduction of filters can have a larger-than-expected impact on the recall and latency due to their interaction (or non-interaction) with the underlying ANNS index. Consequently, understanding the factors that influence filtered query performance is critical for designing efficient retrieval techniques.

Prior work on filtered queries suggests that filter selectivity, that is, the proportion of items that pass the filter, significantly impacts query performance [5, 14]. As a result, studies evaluating filtered query performance often control the selectivity when benchmarking. In this work, we take a different perspective by investigating how changes in metadata distribution influence the performance of filtered queries. **Specifically, we aim to determine whether variations in the distribution of a filtered attribute affect search performance, or whether filter selectivity remains the dominant factor.** To the best of our knowledge, no prior work has explored this aspect.

3 Related Work

There has been much work to develop novel methods to support filtered queries. Filtered-DiskANN [5] proposes a graph-based index that incorporates both the vectors and metadata during index construction to improve the efficiency of filtered queries. This allows efficiently skipping nodes that do not match the filter. VBase [14] is another vector database that unifies ANNS with relational query processing. It achieves this by leveraging a property known as relaxed monotonicity, which is observed in efficient ANNS indexes, such as HNSW and IVFFlat.[1] Relaxed monotonicity ensures that after an initial traversal phase, the distance to the query vector increases monotonically, enabling early query termination. These papers measure performance of filtered queries at different levels of filter selectivity, but do not consider the distribution of the metadata.

[1] Neither HNSW nor IVFFlat *strictly* satisfies relaxed monotonicity at the query termination step but approximately follows the pattern.

Telmai's blog post illustrates how missing or mislabeled metadata can degrade ANNS results [13]. However, this discussion focuses on data quality issues rather than the impact of metadata distribution on search performance.

4 Methodology

4.1 Vector Databases Considered

Filtered queries in vector databases are often implemented via one of two strategies: pre-filtering, where the filter is applied before retrieving results; and post-filtering, where the filter is applied after retrieval [2]. Since these two approaches are widely supported across vector databases, we evaluate one system from each category: ElasticSearch [4] for pre-filtering and ChromaDB [3] for post-filtering. Additionally, we include VBase to provide further insight into one of the novel alternative systems for filtered queries.

ElasticSearch implements pre-filtering by applying the filter during index traversal, skipping nodes that do not match the filter but still expanding to its neighbors. This is in contrast to first filtering the dataset and then performing a brute-force scan on the filtered results. We deliberately focus on ElasticSeach's method of pre-filtering because the alternative method does not leverage the vector index at all. In such cases, it is clear that the query performance will only depend on the selectivity of the filter, not by the distribution of metadata, making it less relevant to our investigation.

4.2 Dataset and Pre-processing

We perform evaluation on the SIFT1M [6] dataset. The SIFT1M dataset consists of 1M base vectors and 10K queries. Each vector is 128-dimensional and comes from SIFT-descriptiors extracted from image patches. By default, there is no metadata associated with these vectors, so we generate multiple variants of the dataset by augmenting each base vector with a synthetic "popularity" attribute sampled from different distributions to use for filtering. The popularity attribute is sampled from the following distributions:

- Uniform $(a = 0, b = 10000)$
- Normal $(\mu = 5000, \sigma = 1000)$
- Log-normal $(\mu = \log 5000, \sigma = \log 1000)$
- Zipfian $(a = 2.0)$
- Zipfian $(a = 1.1)$

The 1st, 10th, 50th, 90th, and 99th percentile popularity values from each distribution are logged, later used to set the filters to control the selectivity of queries.

We include two parameter settings for the Zipfian distribution because the value of a significantly influences the selectivity, which in turn, impacts the search performance. a controls the rate of decay: higher values of a lead to a steeper drop-off and concentrates the probability mass on a small set of frequent items. We refer to the Zipfian distribution with $a = 1.1$ as "Zipfian Flat." Figures 1a and 1b show histograms of sampled popularity values from the two Zipfian distributions. Due to the discrete and heavily skewed nature of the Zipfian distribution, many duplicates exist at the smaller values. For example, in 1a we see that more than 50% of the values are equal to 1. Consequently, when applying a low selectivity filter, the presence of duplicates makes the effective selectivity higher. Since a valid Zipfian distribution requires $a > 1$, our "Zipfian Flat" setting represents one of the least skewed configurations. The other distributions do not suffer from this issue as they are continuous.

4.3 Query and Evaluation Setup

As mentioned in Section 2, the main goal is to determine whether distributional changes in the popularity attribute impact performance of filtered queries. To evaluate this, we issue single-column top-K queries with $K = 50$ using the 10K query vectors for each combination of database system, metadata distribution, and selectivity level (1%, 10%, 50%, 90%, 99%). For each combination, we measure the average recall as well as the average, median, and 99% latency. All queries run sequentially to avoid interference from other queries.

Algorithm 1 provides an example of a SQL query used for VBase, and we adapt the query accordingly for other systems that do not use the SQL interface.

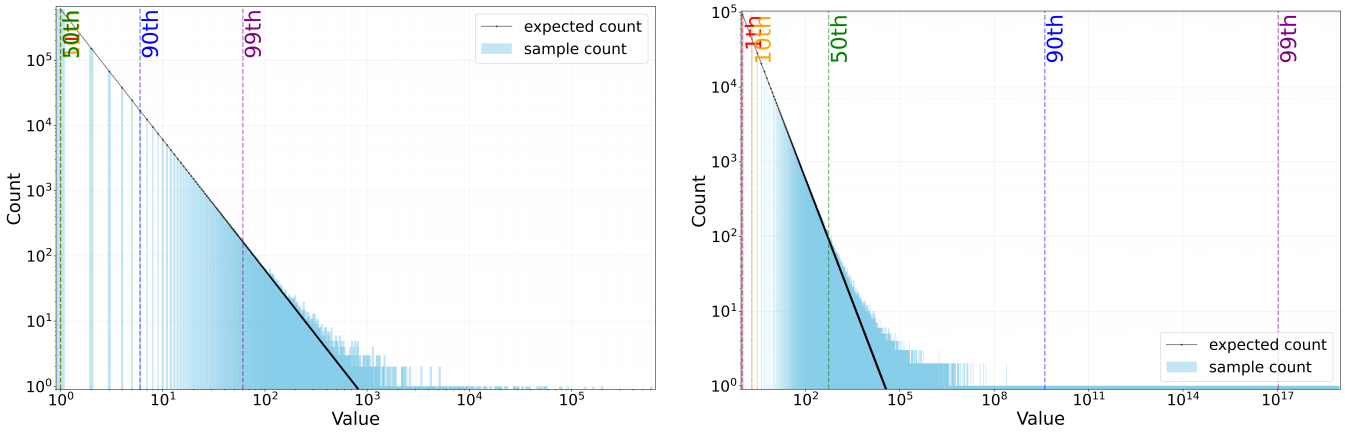
Algorithm 1 Single Vector Top-K + Numeric Filter

```
1: SELECT id FROM sift_table
2: WHERE popularity ≤ ${popularity_threshold}
3: ORDER BY sift_vector<*>ARRAY[{embedding}]
4: LIMIT 50;
```

ElasticSearch, ChromaDB, and VBase are each configured to utilize an HNSW index [9] with parameters $M = 16$, $ef\_construction = 200$, and $ef\_search = 64$ to remain consistent with the values in the original VBase paper (we are using a dataset of similar scale). All three systems also employ an additional index on the popularity column to optimize filtering on metadata. Note that VBase only uses its B-tree index on popularity when its query planner estimates that pre-filtering will be more efficient than the filtered query with the HNSW index.

4.4 Hardware Details

We run all experiments on the University of Toronto CSLab SLURM cluster [11], using compute nodes with 64 CPU cores, 128G of RAM and an AMD EPYC 9634 processor.



(a) Zipfian ($a = 2.0$)

(b) Zipfian Flat ($a = 1.1$)

Figure 1: Histogram of Popularity Values from Zipfian Distributions (Axes in Log Scale)

Each job has exclusive access to a compute node once assigned, ensuring a consistent evaluation environment. All jobs are able to run without fully utilizing the 128G of memory.

5 Results

5.1 ElasticSearch (Pre-Filtering)

The recall and latency results are summarized in Figures 2a and 3. We generally observe similar performance across distributions, except for the Zipfian and Zipfian Flat distributions, which show different trends.

Notably, the Zipfian Flat distribution shows significantly higher latency at selectivity 1 compared to other distributions, and the Zipfian distribution shows lower latency at selectivities 1 and 10. At first glance, this might seem counter-intuitive since any trend observed in the Zipfian Flat distribution is likely exacerbated for the Zipfian distribution. This behavior arises from how ElasticSearch's query planner performs pre-filtering for low-selectivity queries. Specifically, if fewer than ef_search candidates match the filter in a given segment, ElasticSearch bypasses the HNSW index and instead resorts to the alternative pre-filtering method of filtering the full dataset and performing a brute-force scan over the filtered set. This behavior is not well-documented online, but it was confirmed by an ElasticSearch engineer.

Because the Zipfian and Zipfian Flat distributions contain many duplicate values, the query planner executes queries on the HNSW index across all selectivities. In contrast, for other distributions, ElasticSearch performs a full scan over the filtered set when selectivity is 1. The high latency of Zipfian Flat at selectivity 1 occurs because its effective selectivity is too high to trigger a full scan but still low enough to significantly slow down HNSW index traversal. Meanwhile, the Zipfian distribution benefits from much higher effective selectivity when using the HNSW index, leading to lower latency at selectivities 1 and 10. Note that a higher selectivity will improve the latency, as more nodes encountered during traversal become eligible to be included in the final candidate set.

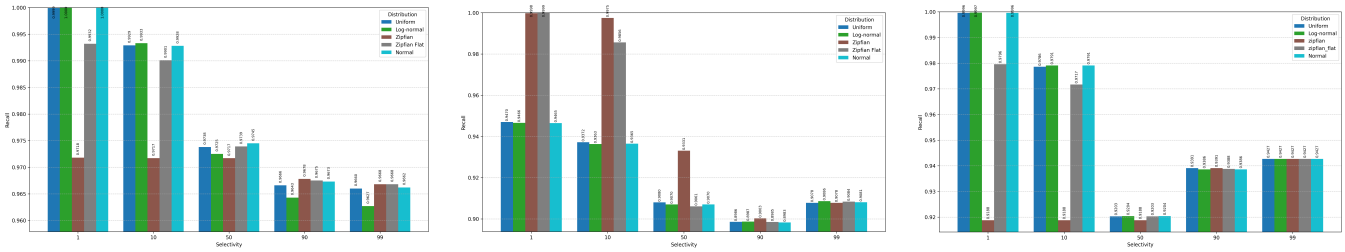
This behavior also explains the recall trends. For non-Zipfian distributions at selectivity 1, we observe perfect recall due to the full scan, whereas Zipfian and Zipfian Flat exhibit lower recall because they rely on approximate search with HNSW. Furthermore, as selectivity increases, fewer nodes in the index are traversed, leading to a decline in recall.

In conclusion, the differences observed for the Zipfian and Zipfian Flat distributions stem from the high number of duplicate values in these distributions, along with quirks of the ElasticSearch query planner. Other distributions all show similar performance.

5.2 ChromaDB (Post-Filtering)

For ChromaDB, we use a post-filtering approach, using a one-shot estimate of $50/selectivity$ for the number of results to retrieve in the initial ANNS. That is, when applying a filter with selectivity 0.01, we would initially retrieve $50/0.01 = 5000$ candidates. The corresponding latency and recall results are summarized in 2b and 4, respectively.

Since only the selectivity is used to estimate the number of candidates to retrieve, and the filter condition is not considered during index traversal, it is expected that query latency is influenced solely by selectivity, with the underlying metadata distribution playing no significant role. Our results confirm this expectation. Additionally, we observe that the distribution of the popularity attribute has minimal impact on recall, again except in the Zipfian and Zipfian Flat distributions. In this case, due to the presence of many duplicate values, even when filtering at the the low selectivity thresholds, a larger-than-expected proportion of vectors satisfy the filter. This



(a) ElasticSearch Recall Metrics (b) Chroma Recall Metrics (c) VBase Recall Metrics

Figure 2: Recall Metrics by Selectivity

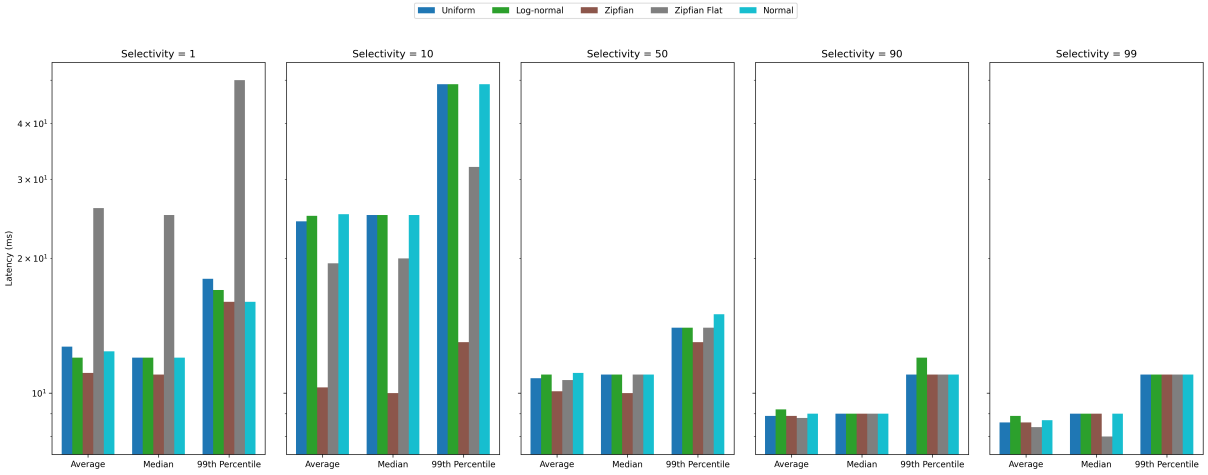


Figure 3: ElasticSearch Latency Metrics by Selectivity

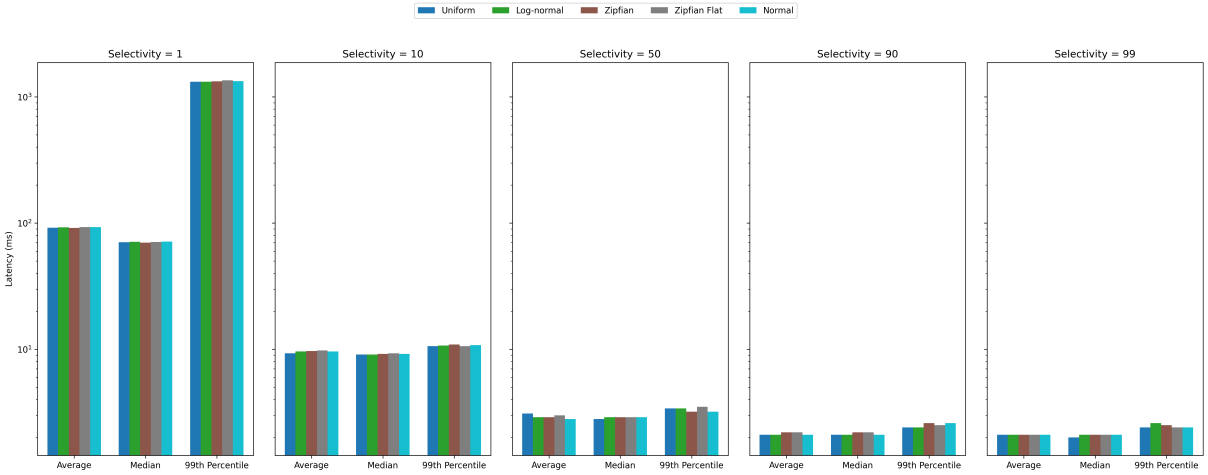


Figure 4: ChromaDB Latency Metrics by Selectivity

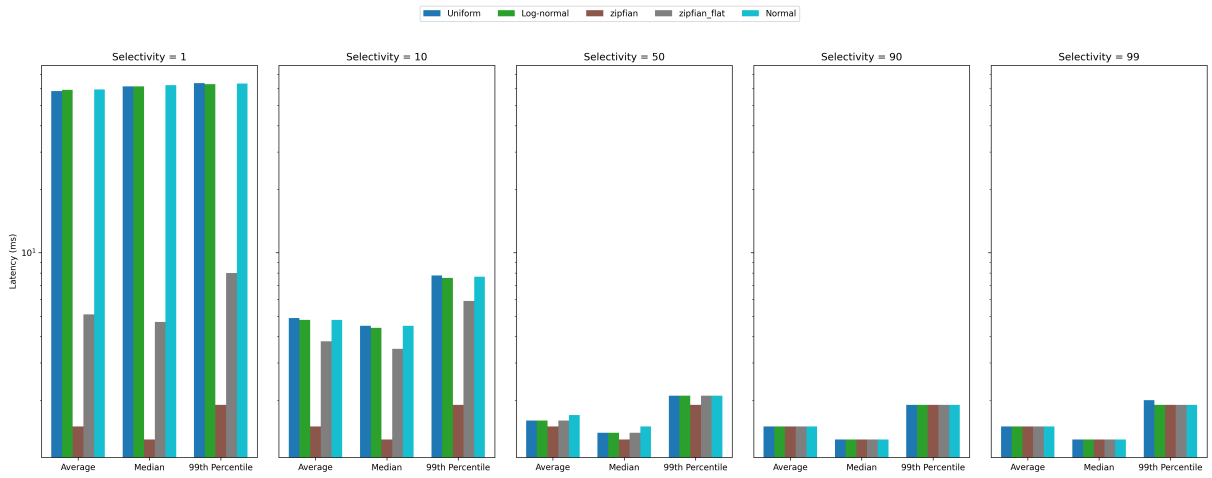


Figure 5: VBase Latency Metrics by Selectivity

effectively increases the number of candidate vectors considered, leading to higher recall in low-selectivity configurations.

In conclusion, distributional changes in metadata have minimal effect on performance, aside from the effect of duplicate values in the Zipfian distributions.

5.3 VBase

VBase utilizes the "relaxed monotonicity" property to preemptively terminate queries when further traversal of the HNSW index is unlikely to yield closer vectors. Additionally, it employs a query planner that determines the optimal execution strategy by estimating the costs of its scalar (B-tree) and vector (HNSW) index traversal. While the VBase paper states that the HNSW index will be used if the estimated selectivity is below 0.18, we found that the query planner never selects the HNSW index for all selectivities. Instead, it consistently opts to pre-filter with the scalar index followed by a full scan, leading to slower query performance. To address this, we explicitly disable the scalar index, forcing the use of the HNSW index. The resulting latency and recall metrics are reported in 5 and 2c, respectively.

When traversing the HNSW index, VBase repeatedly requests the `next` closest vector until it finds the requested number of vectors that match the filter. Similar to ElasticSearch, both Zipfian distributions exhibit distinct latency statistics at lower selectivities. Again, this behavior can be attributed to the presence of duplicates at the threshold levels for lower selectivities, which alters the effective selectivity. The consistency of latency metrics across selectivity levels 1, 10, and 50 in the Zipfian distribution (as the corresponding thresholds are all equal to 1.0 as shown in 1a), as well as the latency values converging in the higher selectivity levels, further illustrates this effect.

Recall follows a similar pattern: as selectivity increases, the performance gap across distributions narrows. At low selectivities, where fewer candidate rows are considered for Zipfian and Zipfian Flat, we observe lower recall values. This is in line with the findings in the original VBase paper.

In conclusion, distributional changes in metadata have minimal effect on performance, aside from the effect of duplicate values in the Zipfian distributions.

6 Limitations

Some limitations of our work should be considered when interpreting the results.

First, our evaluation is conducted in a simplified setting with a single shard, where all queries run independently. This does not accurately reflect a production environment, and depending on the system setup coupled with database-specific behavior (as we saw with ElasticSearch's query planner in 5.1), the performance can differ. For example, in highly skewed distributions like the Zipfian, most query results may come from a single shard, which could affect load balancing and retrieval efficiency. Additionally, we only consider the HNSW index, and other indexing methods may exhibit different behaviors. Future work could include scaling up these experiments to multi-shard production environments or exploring alternative indexing strategies to generalize our findings.

Another limitation is that our metadata is synthetically generated and independent of the vector data, whereas in real-world applications, metadata and vector representations are often correlated. However, based on our results, if any performance differences arise in real-world datasets, they can likely be attributed to variations in the vector data itself rather than the metadata distribution.

7 Conclusion

In conclusion, this paper provides an evaluation of how metadata distribution influences the performance of filtered ANNS for various vector databases with different filtering strategies. Our experiments indicate that the distributional shifts in the metadata do not meaningfully impact the query performance, aside from when duplicate values in discrete and highly-skewed distributions distort the effective selectivity. The selectivity of the filter remains the dominant factor in influencing the query performance. When recall or latency metrics deviate from expected trends across selectivity levels, we analyze system-specific optimizations to better understand their impact on overall performance.

References

[1] Zeynep Akkalyoncu Yilmaz, Shengjin Wang, Wei Yang, Haotian Zhang, and Jimmy Lin. Applying BERT to document retrieval with birch. In Sebastian Padó and Ruihong Huang, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, pages 19–24, Hong Kong, China, November 2019. Association for Computational Linguistics.

[2] James Briggs. The missing where clause in vector search.

[3] Chroma. https://www.trychroma.com/.

[4] ElasticSearch. https://www.elastic.co.

[5] Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, Amit Singh, and Harsha Vardhan Simhadri. Filtered-diskann: Graph algorithms for approximate nearest neighbor search with filters. In *Proceedings of the ACM Web Conference 2023*, WWW '23, page 3406–3416, New York, NY, USA, 2023. Association for Computing Machinery.

[6] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

[7] Jie Li, Haifeng Liu, Chuanghua Gui, Jianyu Chen, Zhenyun Ni, and Ning Wang. The design and implementation of a real time visual search system on jd e-commerce platform, 2019.

[8] Defu Lian, Haoyu Wang, Zheng Liu, Jianxun Lian, Enhong Chen, and Xing Xie. Lightrec: A memory and search-efficient recommender system. In *Proceedings of The Web Conference 2020*, WWW '20, page 695–705, New York, NY, USA, 2020. Association for Computing Machinery.

[9] Yury A. Malkov and Dmitry A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *CoRR*, abs/1603.09320, 2016.

[10] Antoine Miech, Dimitri Zhukov, Jean-Baptiste Alayrac, Makarand Tapaswi, Ivan Laptev, and Josef Sivic. Howto100m: Learning a text-video embedding by watching hundred million narrated video clips, 2019.

[11] University of Toronto Computer Science Department. Slurm resource management guide, 2025. Accessed: 2025-02-14.

[12] Mattis Paulin, Matthijs Douze, Zaid Harchaoui, Julien Mairal, Florent Perronin, and Cordelia Schmid. Local convolutional features with unsupervised training for image retrieval. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 91–99, 2015.

[13] Telmai. Data quality for vector databases. https://www.telm.ai/blog/data-quality-for-vector-databases/, February 2025. Accessed: 2025-02-14.

[14] Qianxi Zhang, Shuotao Xu, Qi Chen, Guoxin Sui, Jiadong Xie, Zhizhen Cai, Yaoqi Chen, Yinxuan He, Yuqing Yang, Fan Yang, Mao Yang, and Lidong Zhou. VBASE: Unifying online vector similarity search and relational queries via relaxed monotonicity. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 377–395, Boston, MA, July 2023. USENIX Association.

8 Appendix

Distribution	Selectivity	Recall	Avg Latency	Median Latency	99th Latency
Uniform	1	0.947	92.0	70.6	1320.3
Uniform	10	0.9372	9.3	9.1	10.6
Uniform	50	0.908	3.1	2.8	3.4
Uniform	90	0.8986	2.1	2.1	2.4
Uniform	99	0.9078	2.1	2.0	2.4
Log-normal	1	0.9466	92.5	71.1	1320.6
Log-normal	10	0.9363	9.6	9.1	10.7
Log-normal	50	0.907	2.9	2.9	3.4
Log-normal	90	0.8987	2.1	2.1	2.4
Log-normal	99	0.9086	2.1	2.1	2.6
Zipfian	1	0.9998	91.6	70.0	1330.0
Zipfian	10	0.9975	9.7	9.2	10.9
Zipfian	50	0.9331	2.9	2.9	3.2
Zipfian	90	0.9003	2.2	2.2	2.6
Zipfian	99	0.9078	2.1	2.1	2.5
Zipfian Flat	1	0.9999	92.8	70.8	1352.1
Zipfian Flat	10	0.9856	9.8	9.3	10.6
Zipfian Flat	50	0.9061	3.0	2.9	3.5
Zipfian Flat	90	0.8985	2.2	2.2	2.5
Zipfian Flat	99	0.9084	2.1	2.1	2.4
Normal	1	0.9465	93.0	71.3	1331.9
Normal	10	0.9365	9.6	9.2	10.8
Normal	50	0.907	2.8	2.9	3.2
Normal	90	0.8983	2.1	2.1	2.6
Normal	99	0.9081	2.1	2.1	2.4

Table 1: Chroma Performance Metrics Across Selectivity and Distribution

Distribution	Selectivity	Recall	Avg Latency	Median Latency	99th Latency
Uniform	1	0.9999	12.7	12.0	18.0
Uniform	10	0.9929	24.2	25.0	49.0
Uniform	50	0.9738	10.8	11.0	14.0
Uniform	90	0.9666	8.9	9.0	11.0
Uniform	99	0.966	8.6	9.0	11.0
Log-normal	1	1.0	12.0	12.0	17.0
Log-normal	10	0.9933	24.9	25.0	49.0
Log-normal	50	0.9725	11.0	11.0	14.0
Log-normal	90	0.9643	9.2	9.0	12.0
Log-normal	99	0.9627	8.9	9.0	11.0
Zipfian	1	0.9718	11.1	11.0	16.0
Zipfian	10	0.9717	10.3	10.0	13.0
Zipfian	50	0.9717	10.1	10.0	13.0
Zipfian	90	0.9678	8.9	9.0	11.0
Zipfian	99	0.9668	8.6	9.0	11.0
Zipfian Flat	1	0.9932	25.9	25.0	50.0
Zipfian Flat	10	0.9901	19.5	20.0	32.0
Zipfian Flat	50	0.9739	10.7	11.0	14.0
Zipfian Flat	90	0.9675	8.8	9.0	11.0
Zipfian Flat	99	0.9668	8.4	8.0	11.0
Normal	1	1.0	12.4	12.0	16.0
Normal	10	0.9928	25.1	25.0	49.0
Normal	50	0.9745	11.1	11.0	15.0
Normal	90	0.9673	9.0	9.0	11.0
Normal	99	0.9662	8.7	9.0	11.0

Table 2: Elastic Performance Metrics Across Selectivity and Distribution

Distribution	Selectivity	Recall	Avg Latency	Median Latency	99th Latency
Uniform	1	0.9996	58.4	61.4	63.6
Uniform	10	0.9786	4.9	4.5	7.8
Uniform	50	0.9203	1.6	1.4	2.1
Uniform	90	0.9391	1.5	1.3	1.9
Uniform	99	0.9427	1.5	1.3	2.0
Log-normal	1	0.9997	59.2	61.4	62.9
Log-normal	10	0.9791	4.8	4.4	7.6
Log-normal	50	0.9204	1.6	1.4	2.1
Log-normal	90	0.9386	1.5	1.3	1.9
Log-normal	99	0.9427	1.5	1.3	1.9
Zipfian	1	0.9188	1.5	1.3	1.9
Zipfian	10	0.9188	1.5	1.3	1.9
Zipfian	50	0.9188	1.5	1.3	1.9
Zipfian	90	0.9391	1.5	1.3	1.9
Zipfian	99	0.9427	1.5	1.3	1.9
Zipfian Flat	1	0.9796	5.1	4.7	8.0
Zipfian Flat	10	0.9717	3.8	3.5	5.9
Zipfian Flat	50	0.9203	1.6	1.4	2.1
Zipfian Flat	90	0.9388	1.5	1.3	1.9
Zipfian Flat	99	0.9427	1.5	1.3	1.9
Normal	1	0.9996	59.4	62.2	63.4
Normal	10	0.9791	4.8	4.5	7.7
Normal	50	0.9204	1.7	1.5	2.1
Normal	90	0.9386	1.5	1.3	1.9
Normal	99	0.9427	1.5	1.3	1.9

Table 3: VBase Performance Metrics Across Selectivity and Distribution