Project 2: HNSW

Author
Kangwei Zhu
NetID: kangwei2

1 Project Environment & Acknowledgement

I largely used LLMs such as ChatGPT and Claude to write and debug codes. LLMs also helped paraphrasing and fixing typo in this report. The project was run and tested under:

- **OS**: Linux 6.17.5-arch1-1,
- **CPU**: AMD Ryzen™ AI 9 HX 370
- **Memory**: 32GB.

It is worth noting that during evaluation, the additional data logging and data presistence in code introduced runtime overhead for both algorithms. Nevertheless, the observed performance trends remain relatively accurate and representative of their true behavior.

2 HNSW vs LSH

In this experiment, I compared the performance of **Hierarchical Navigable Small Worlds (HNSW)** and **Locality Sensitive Hashing (LSH)** on the **SIFT1M** dataset. Both methods were evaluated using *Recall@1* and *Queries Per Second (QPS)* as primary metrics. The trade-off between accuracy and throughput was analyzed by varying the key parameters of each algorithm: `efSearch` for HNSW and `nbits` for LSH.

Experiment Configuration

- **Dataset:** SIFT1M (1M train vectors, 10K query vectors, 128 dimensions)
- **HNSW parameters:** $M = 32$, $efConstruction = 200$, $efSearch \in \{10, 50, 100, 200\}$
- **LSH parameters:** $nbits \in \{32, 64, 512, 768\}$

The evaluation considers two primary metrics across all configurations:

- **Recall@1:** The percentage of queries in which the true nearest neighbor appears as the top-1 result.
- **QPS (Queries Per Second):** The number of queries handled per second, indicating query throughput.

As shown in Figure 1, the plot visualizes how Recall@1 varies with QPS for both methods, where each annotated marker corresponds to a specific parameter configuration (`efSearch` for HNSW, `nbits` for LSH).

Observations and Conclusion.

HNSW exhibits a strong and smooth trade-off between recall and throughput. Increasing `efSearch` from 10 to 200 raises Recall@1 from 0.85 to 0.99, while QPS decreases from about 96K to 9.7K. This demonstrates that HNSW can flexibly balance accuracy and latency through the `efSearch` parameter. The improvement in recall occurs because a larger `efSearch` value expands the candidate

exploration during query traversal, allowing the search to reach deeper layers of the graph and identify closer neighbors. However, this increased traversal also introduces more distance computations per query, leading to higher latency and reduced throughput.

By contrast, LSH achieves much lower recall under all configurations. Even when increasing `nbits` from 32 to 768, Recall@1 only improves from 0.008 to 0.40, while QPS drops from 17K to under 2K. This indicates that LSH's discriminative power grows slowly with larger hash sizes, while query efficiency declines rapidly. The limited recall arises because LSH relies on random hash projections that approximate similarity in a probabilistic manner; as dimensionality increases, many true neighbors fall into different hash buckets. Increasing `nbits` reduces these collisions but at the cost of more hash computations and sparser bucket lookups, which significantly slows down query processing.

To conclude, HNSW provides a more effective balance between accuracy and speed on high-dimensional datasets like SIFT1M. LSH performs faster at lower bit counts but fails to achieve sufficient recall for high-precision applications, making HNSW the more suitable choice when accuracy is essential.

3 Benchmarking HNSW with increasing dataset sizes

3.1 QPS vs Recall

To examine the scalability of HNSW, we varied the connectivity parameter $M \in \{4, 8, 12, 24, 48\}$ for four datasets: *coco-t2i*, *coco-i2i*, *lastfm*, and *mnist*. We fixed **efConstruction** = 200 and **efSearch** = 200 to ensure a balanced trade-off between index quality and runtime efficiency.

COCO-T2I. As M increases, recall improves significantly from 0.8331 (M=4) to 0.9738 (M=48), while QPS drops from 17.2k to 6.8k. This clearly shows the recall–throughput trade-off typical of HNSW: denser connectivity leads to better accuracy but slower query traversal.

COCO-I2I. A similar pattern is observed: recall rises from 0.9623 (M=4) to 0.9895 (M=48), but QPS falls from 14.7k to 7.2k. The smaller recall gain compared to COCO-T2I suggests that image–image embeddings are more internally consistent, making recall saturate earlier.

LastFM. The *lastfm* dataset achieves both high recall and extremely high throughput. Recall improves slightly from 0.9769 to 0.9913, while QPS remains between 112k–161k across all M. This indicates that user–item embeddings in LastFM have moderate dimensionality and well-clustered neighborhoods, enabling efficient nearest-neighbor search.

MNIST. The *mnist* dataset reaches near-perfect recall even for small M: from 0.9957 (M=4) to 0.9998 (M=24). QPS decreases moderately (18.1k to 8.1k). This demonstrates that MNIST's low-dimensional and highly separable embeddings make HNSW less sensitive to M variation.

Overall, the QPS–Recall relationship shows that higher M consistently improves recall at the expense of throughput, but the effect is dataset-dependent: high-dimensional COCO data exhibit steep trade-offs, while structured, low-dimensional datasets such as MNIST and LastFM remain efficient across all settings.

3.2 Index Build Time vs Recall

Index build time grows approximately linearly with M, consistent with the $O(MN \log N)$ construction complexity of HNSW. Below, we detail dataset-specific trends.

COCO-T2I. Build time increases from 5.16s (M=4) to 10.69s (M=48), while recall improves by 0.14. The substantial growth reflects high feature dimensionality and dense connectivity during link creation.

COCO-I2I. A similar trend occurs: build time grows from 5.12s to 10.79s. Despite a smaller recall increment, the cost of graph construction rises sharply, confirming that high-dimensional embeddings require more candidate comparisons during insertion.

LastFM. For LastFM, build time grows gently from 3.00s (M=4) to 4.23s (M=48), with recall improving from 0.9769 to 0.9913. This moderate increase suggests that the dataset's embedding space is easier to partition, yielding faster neighbor selection even at high M.

MNIST. MNIST has overall build times (2.7–5.4s). Recall saturates near 1.0 early, and increasing M provides minimal gain but noticeable cost increase. This again reflects that low-dimensional, well-separated vectors require fewer edges to achieve full connectivity.

Across datasets, both QPS and build time are shaped by intrinsic properties: high-dimensional data (COCO) lead to expensive graph traversal and construction, while compact or structured spaces (MNIST, LastFM) deliver high recall and throughput with minimal computational cost. In practice, moderate M values (12–24) provide a good balance for complex, high-dimensional datasets, while smaller M (4–8) suffice for low-dimensional or structured data.

4 HNSW vs DiskANN

4.1 Experimental Setup

This experiment compares two approximate nearest neighbor (ANN) search algorithms — **HNSW** (Hierarchical Navigable Small World Graph) and **DiskANN** — on the `SIFT1M` dataset. The goal is to analyze how varying key parameters impacts query latency and recall, and to visualize the latency–recall tradeoff for both algorithms.

Dataset and Metrics We used the public `SIFT1M.hdf5` dataset, which contains 1 million 128-dimensional feature vectors. The evaluation metrics are:

- **Recall@1**: the fraction of queries whose nearest neighbor matches the ground truth;
- **Latency (ms)**: average query response time per vector;
- The x-axis plots $(1 - Recall@1)$, and the y-axis shows latency, to clearly illustrate the accuracy–efficiency tradeoff.

HNSW Configuration For HNSW, we varied the `efSearch` parameter while keeping other parameters constant:

- $M = 32$
- $efConstruction = 200$
- $efSearch \in \{10, 50, 100, 200\}$

DiskANN Configuration For DiskANN, we varied the graph degree (`R`) and the search complexity parameter (`L`) jointly:

- $(R, L) \in \{(20, 30), (30, 50), (40, 70), (60, 100)\}$

Other settings such as in-memory and disk memory size were kept constant since the main objective was to observe the latency–recall trend.

4.2 Results and Discussion

Figure 4 shows that both algorithms exhibit a clear tradeoff between latency and recall.

HNSW Behavior As `efSearch` increases from 10 to 200, recall improves from 0.8483 to 0.9925, while latency increases almost tenfold (from 0.0097 ms to 0.0990 ms). This occurs because a larger `efSearch` expands the dynamic search list, leading to more graph traversal and hence higher recall but longer search time. The shape of the HNSW curve is steep at small `efSearch` values and gradually flattens, indicating diminishing returns in recall improvement at high search effort.

DiskANN Behavior For DiskANN, recall improves from 0.55 to 0.90 as (R, L) increase from $(20, 30)$ to $(60, 100)$, while latency rises from 0.20 ms to 0.95 ms. Higher R increases graph connectivity, and larger L expands the candidate list during search. Since DiskANN is disk-based, the latency cost grows faster due to increased I/O operations compared to the in-memory HNSW.

Comparison and Insights HNSW achieves higher recall at significantly lower latency in this setup, owing to its fully in-memory structure and efficient graph navigation. DiskANN, while slower, is designed for large-scale, disk-resident datasets and thus scales better when data exceeds RAM capacity. In scenarios where memory is limited but high recall is still required (e.g., billion-scale retrieval systems), DiskANN is preferred. Conversely, for latency-critical applications (e.g., real-time recommendation or visual search), HNSW remains the superior choice.

4.3 Summary

- Increasing `efSearch` (HNSW) or (R, L) (DiskANN) improves recall but increases latency.
- HNSW exhibits sub-millisecond latency even at high recall, whereas DiskANN trades speed for scalability.
- The plotted curves capture the fundamental tradeoff between accuracy and efficiency across both ANN frameworks.

References

[1] Erikbern (no date) ERIKBERN/Ann-benchmarks: Benchmarks of approximate nearest neighbor libraries in Python, GitHub. Available at: https://github.com/erikbern/ann-benchmarks?tab=readme-ov-filedata-sets (Accessed: 03 November 2025).

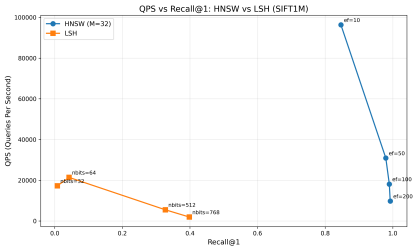

Figure 1: QPS vs Recall@1 comparison between HNSW and LSH on the SIFT1M dataset.

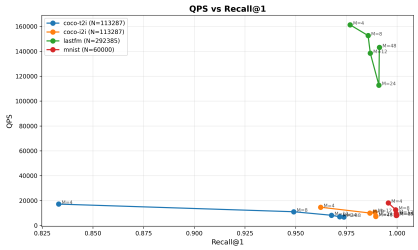

Figure 2: QPS vs Recall@1 across datasets and varying M values.

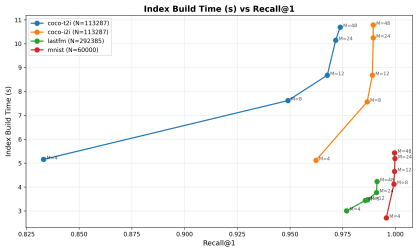

Figure 3: Index Build Time vs Recall@1 across datasets and varying M values.

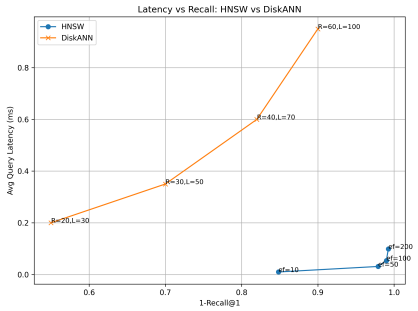

Figure 4: Latency vs Recall@1 for HNSW and DiskANN on SIFT1M. The x-axis represents $1 - Recall@1$, and the y-axis shows average query latency (ms). Each point is annotated with its corresponding parameter value.