FreshDiskANN: A Fast and Accurate Graph-Based ANN Index for Streaming Similarity Search [arXiv'21]

Authors: Aditi Singh, et al. (Microsoft Research & CMU)

Presented by: Yifang Tian (ECE Ph.D., yifang.tian@mail.utoronto.ca)

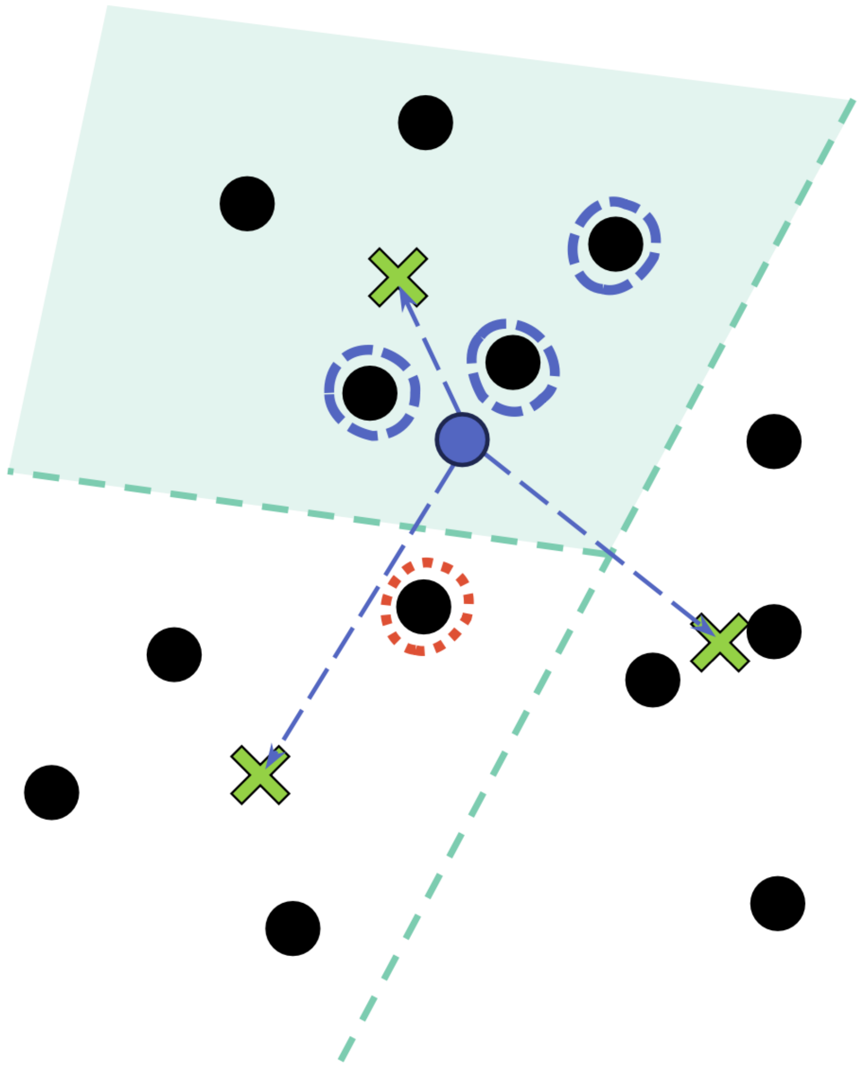
Presentation date: 2025-01-29

UNIVERSITY OF TORONTO

DEFY GRAVITY

Background – Introduction to ANNS and Their Challenges

What is Approximate Nearest Neighbor Search (ANNS)?



[Reference: Professor Gabel's lecture 2 slides for Vector Database Querying]

Real-world applications:

- Recommendation systems
- Image retrieval
- NLP embeddings

Challenges:

- Curse of dimensionality
- Scalability for billion-point datasets

Infeasible to put all data into into RAM!

UNIVERSITY OF TORONTO

Background - DiskANN Overview

DiskANN [Subramanya, NeurIPS'19]: Scalable ANNS on SSDs

- **Vamana algorithm**
- **Indexing more with fewer RAM**
- **Better memory efficiency, latency, and recall (FAISS and HNSW)**

- **Static Indexing**
- **Real-World Constraints**
- **Limited Adaptability**

UNIVERSITY OF TORONTO

Current Challenge – Deletion is hard

HNSW, NSG, and Vamana -

- Delete Policy A:
  - Remove all edges
- Delete Policy B:
  - Remove all edges
  - Add all in/out pairs
  - Prune

The graph becomes sparse!

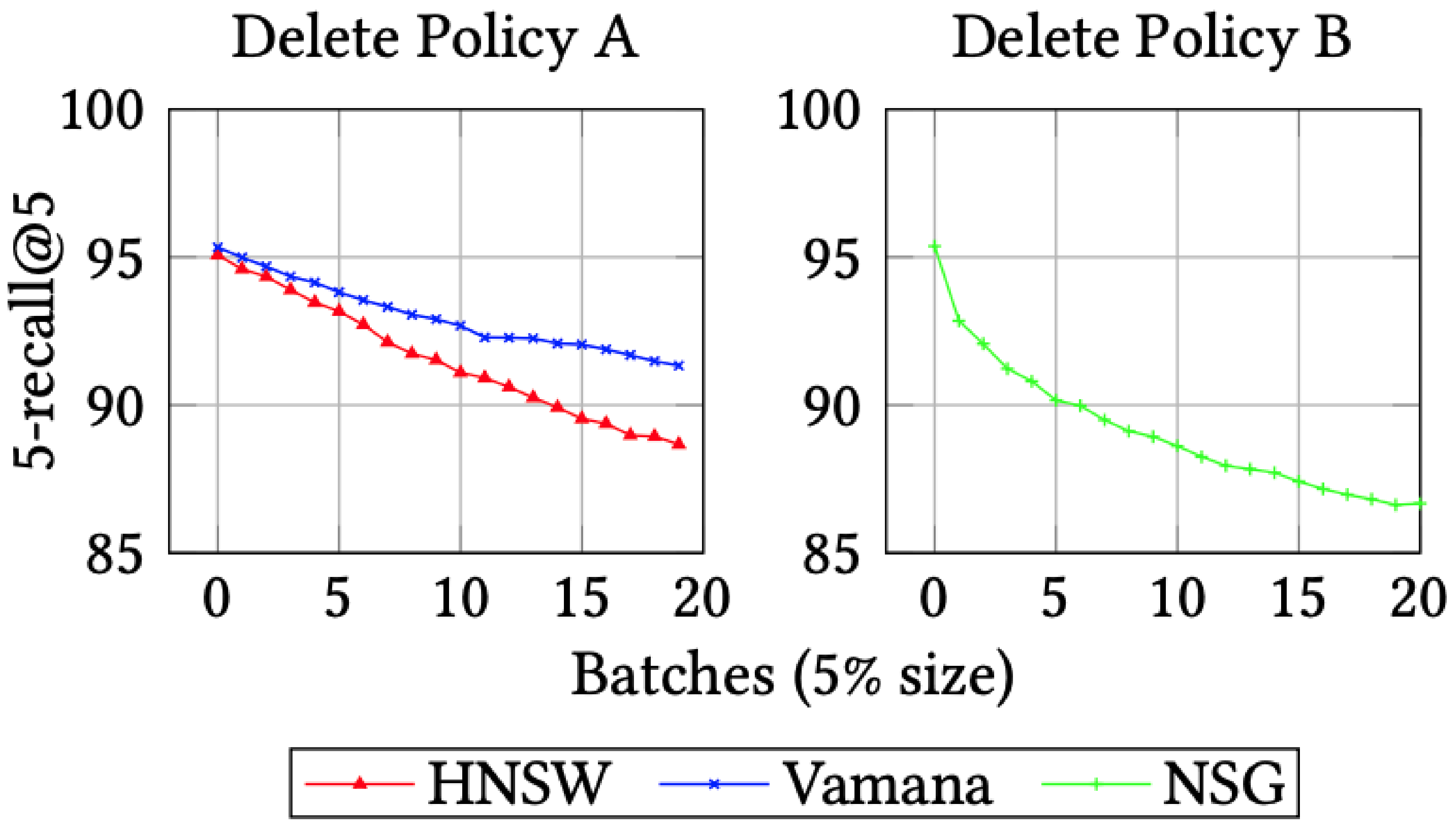


Figure 1. Search recall over 20 cycles of deleting and re-inserting 5% of SIFT1M dataset with statically built HNSW, Vamana, and NSG indices with L_s = 44, 20, 27, respectively.

Main contribution – Cost Reduction in Maintaining Freshness with FreshDiskANN

Intuition:

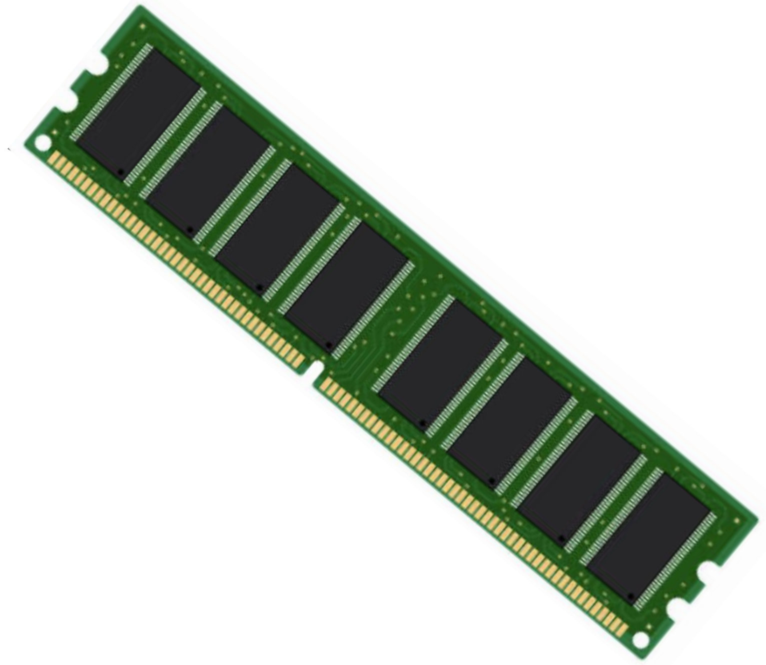
- Vectors are books!
- New books: **temporary bookshelf** (in-memory index)
- Most books: **library's vault** (SSD-based index)
- Borrowed books: **Gone book list** (Delete List)
- Merging books: Library is less busy
- Book-borrowing: **on the fly**

UNIVERSITY OF TORONTO

Methodology Overview - FreshDiskANN

- **Hybrid Architecture**: SSD + in-memory
- **Dynamic Updates**:
  - Real-time insertions and deletions.
  - High recall and low latency during updates.
- **Querying:** From both places then merge result
- **Key Innovations**:
  - **FreshVamana Algorithm**
  - **StreamingMerge Component**

Long-term (SSD)



Temporary (Memory)

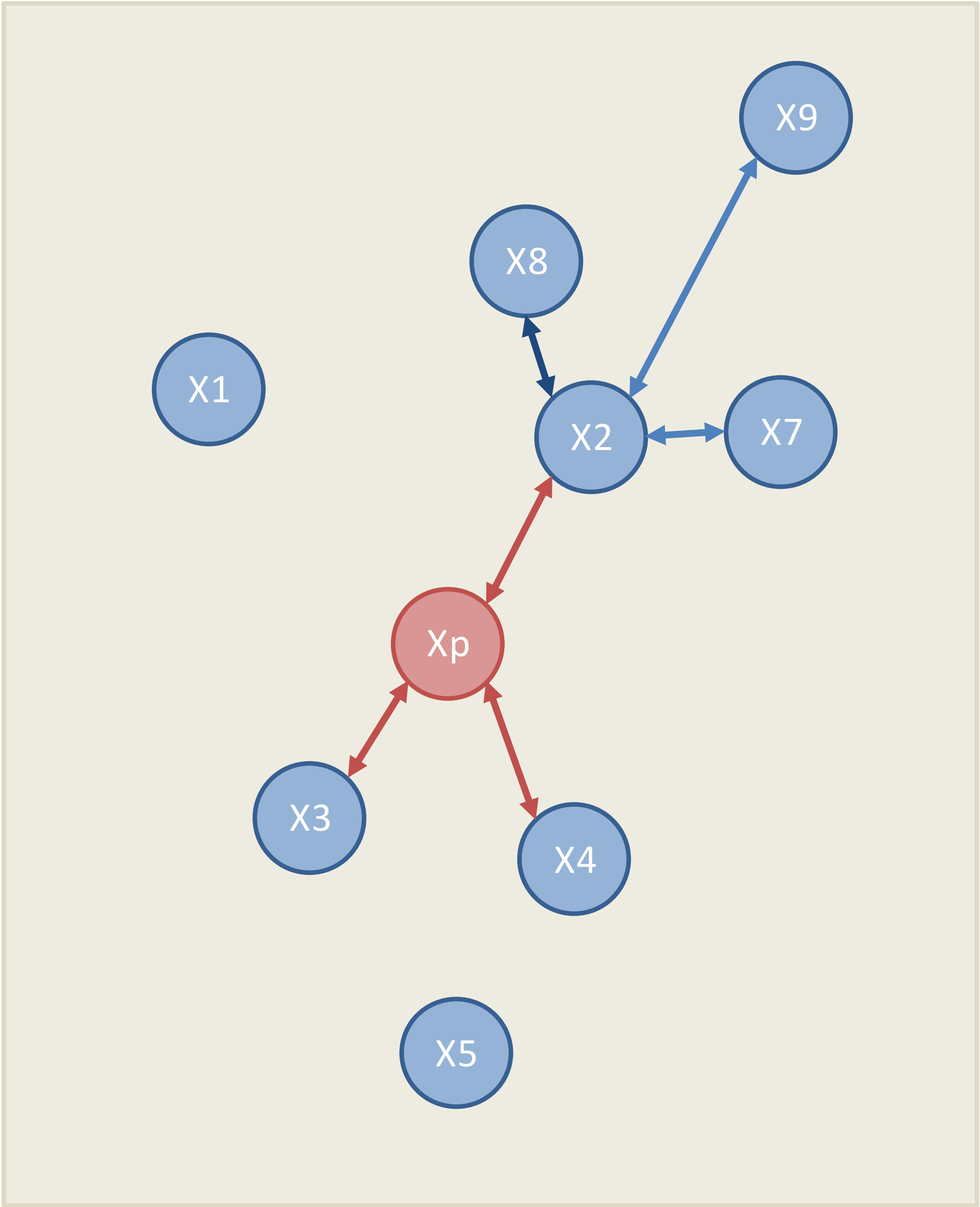
UNIVERSITY OF TORONTO

Methodology - Real-Time Graph Construction with FreshVamana Algorithm

SSD	I. Initiation	• Build graph with Vamana • Satisfy the α-RNG property
In-memory	II. Insertion	• GreedySearch for nearest neighbors • RobustPrune to update edges
In-memory	III. Deletion	• Remove the point • Rewire its neighbors
From in-memory to SSD	IV. Merge	• Periodically merge the in-memory index with the SSD-resident index

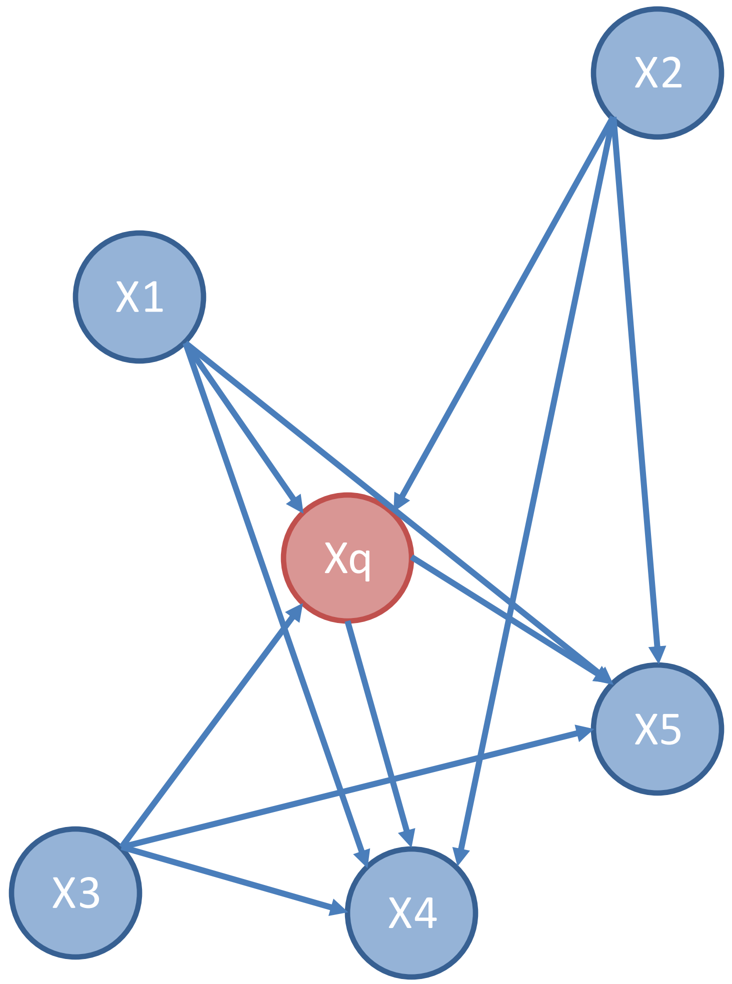
Methodology – Insertion in FreshVamana Algorithm

1. Query Nearest Neighbors: {X1, X2, X3, X4, X5}
2. Generate Candidate Out-Neighbors: {X2, X3, X4}
3. Add Bi-Directional Edges: (Xp <-> X2), (Xp <-> X3), (Xp <-> X4)
4. Prune Over-Degree Nodes: Prune X2
5. Concurrency Control with lock



Methodology – Deletion in FreshVamana Algorithm

1. Lazy Deletion:
   - Stay there with In-neighbours (X1, X2) and out-neighbours (X3, X4, X5)
   - Add to Delete List
2. Search Adjustments
   - Filter result with Delete List
3. Delete Consolidation
   - Batch update
   - Delete node / add edges
   - Prune



Delete List:

UNIVERSITY OF TORONTO

Methodology – StreamingMerge in FreshDiskANN System

I. Initiation

- Temporary index is full

II. Deletion

- Small patch with the point and neighbours
- Robust Prune and write back

III. Insertion

- Find small patch neighbours
- Store delta in memory
- NOT directly write back! (excessive I/O)

IV. Patch

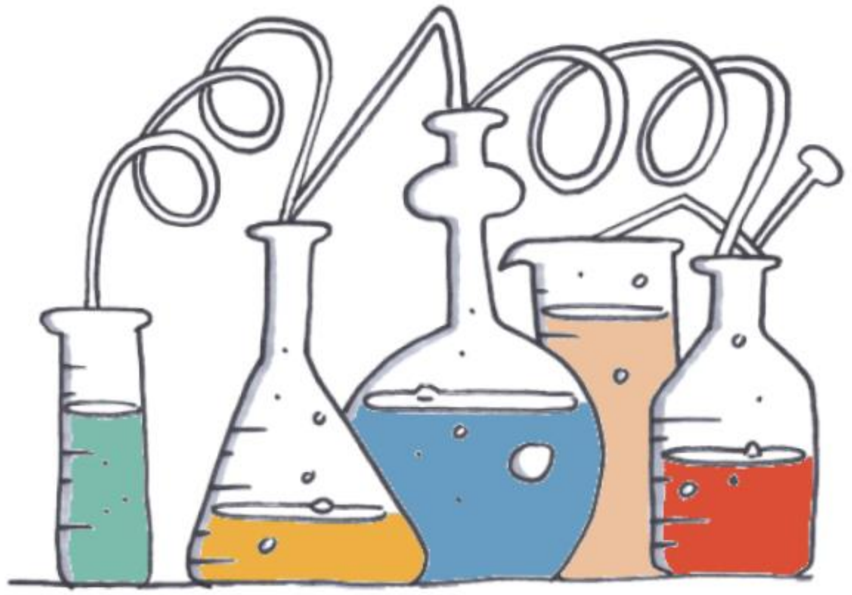
- Patch-by-patch for each delta
- Integrates backward edges
- Write to SSD

UNIVERSITY OF TORONTO

Evaluation Setup

- **Datasets**:
  - **SIFT1B** and other.
  - Real-world dynamic workloads.
- **Metrics**:
  - Recall
  - Query Latency

- **Baseline Comparisons**:
  - **DiskANN** and static ANNS systems.
- **Infrastructure**:
  - Single-node SSDs
  - 64GB RAM



UNIVERSITY OF TORONTO

Evaluation – Recall Stability of StreamingMerge

- Dataset: SIFT100M and SIFT1B
- Statically build with 80M points / 800M points
- StreamingMerge
  - 8M / 80M insertion
  - 8M / 80M deletion
- Run 40 cycles

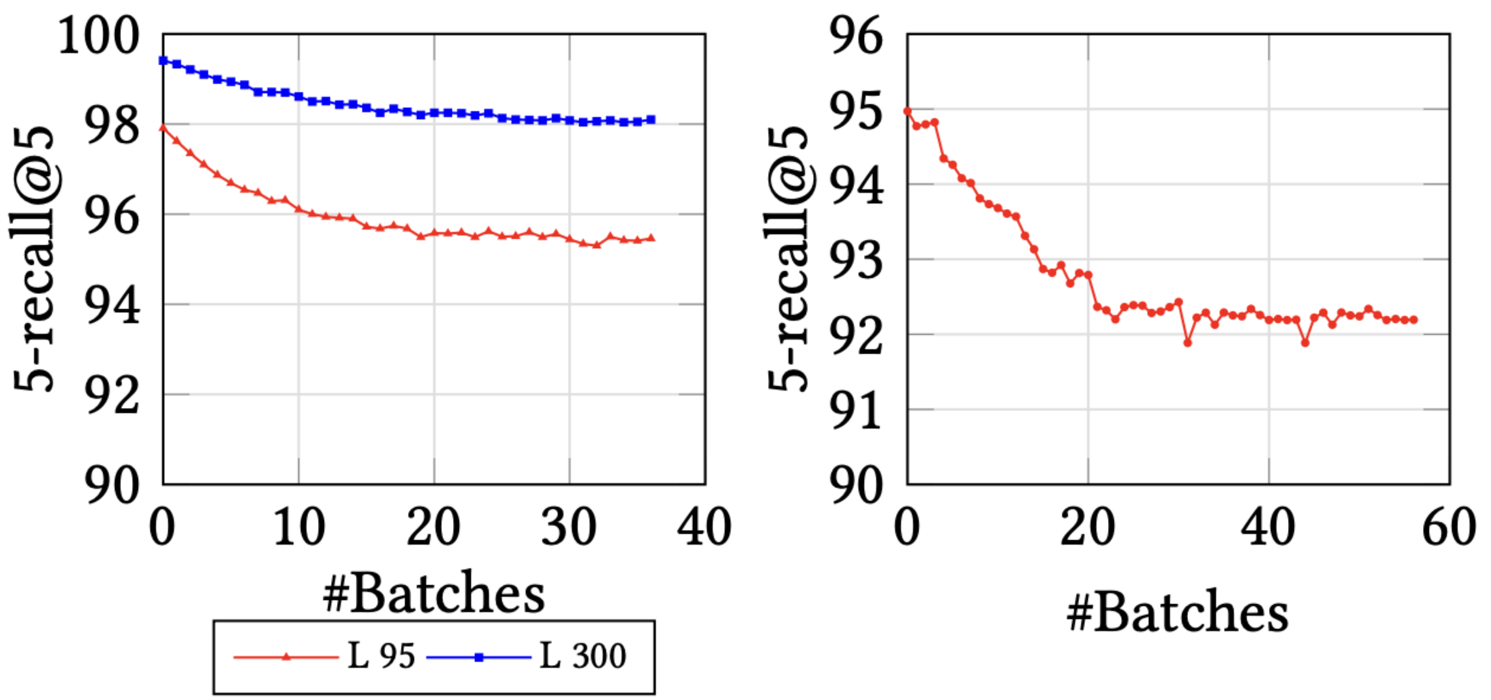


Figure 4. Recall evolution over multiple cycles of `StreamingMerge` in *steady-state* over (left) 80M point index with 10% deletes and inserts and (right) 800M point index with 30M insertions and deletions.

Evaluation – Stage 1: Ramp Up

- Build while querying
- Start with 100M
- Temp memory cap at 30M
- Insert until 800M
- Took 3 days
- Latency: 5 ms when no merging, 15 ms when merging

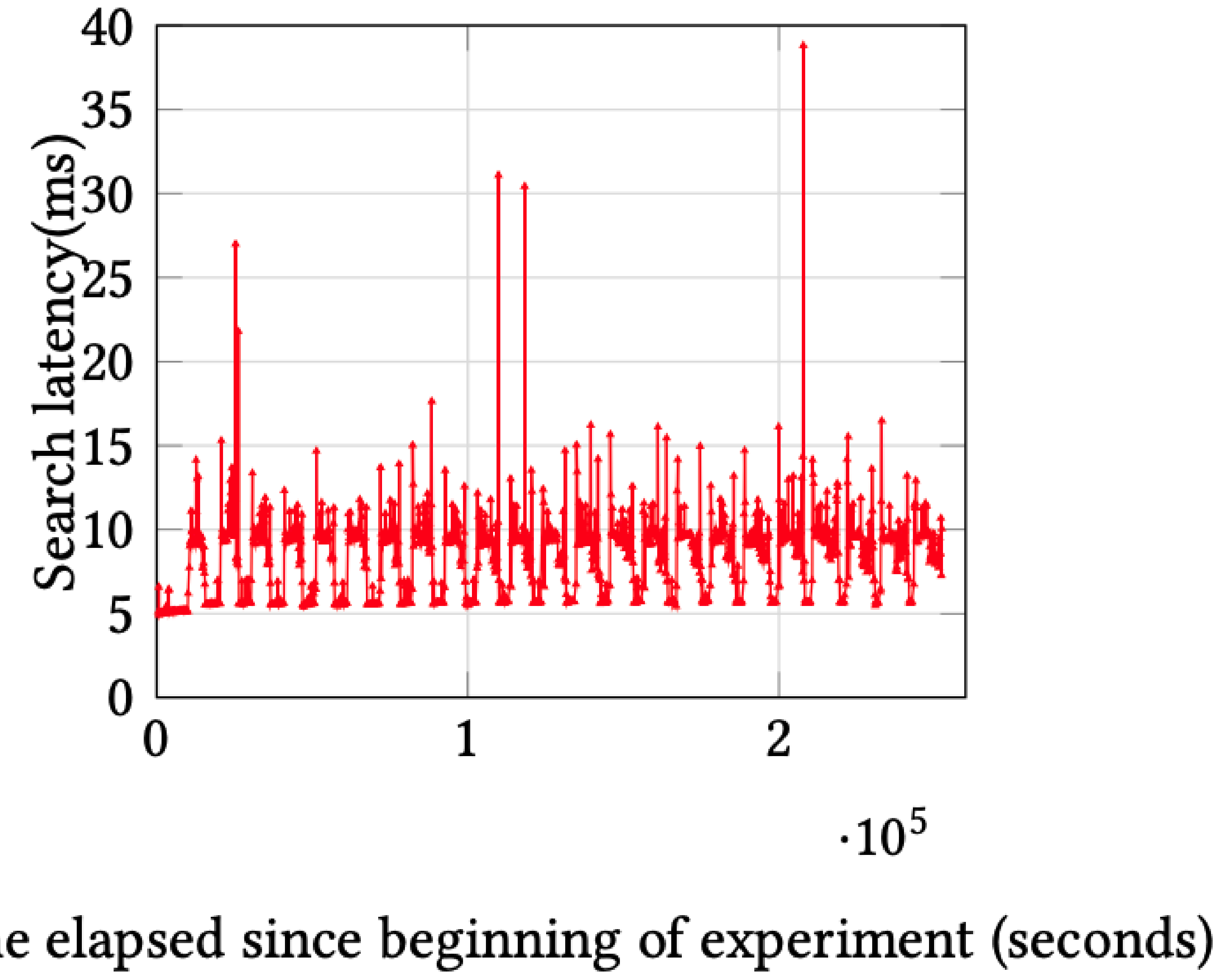


Figure 5. Search latencies for $Ls = 100$ (always $> 92\%$ 5-recall@5) over the course of ramping up an index to size 800M. Each point is mean latency over a 10000-query batch.

UNIVERSITY OF TORONTO

Results – Stage 2: Steady State Query Performance (Recall vs Latency)

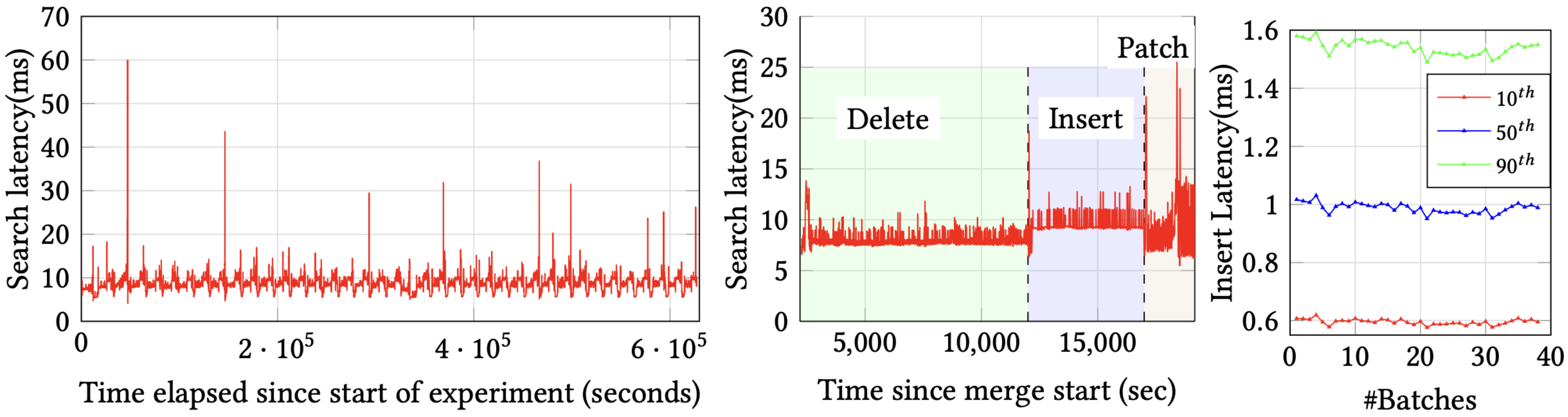


Figure 6. Mean latency[4] measurements for the week-long *steady-state* experiment with an 800M FreshDiskANN index processing concurrent inserts, deletes, and periodic background merge. (left) Search latency with $Ls = 100$ over the entire experiment; (middle) Search latency during one `StreamingMerge` run, zoomed in from the left plot; (right) 10^{th}, 50^{th} and 90^{th} percentile insert latency over the entire experiment.

- **High recall (> 95%) at 5-10 ms query latency**
- **Stability across updates**
- **Handles well in dynamic scenarios**

UNIVERSITY OF TORONTO

Results - Much Faster Index Build Time than DiskANN

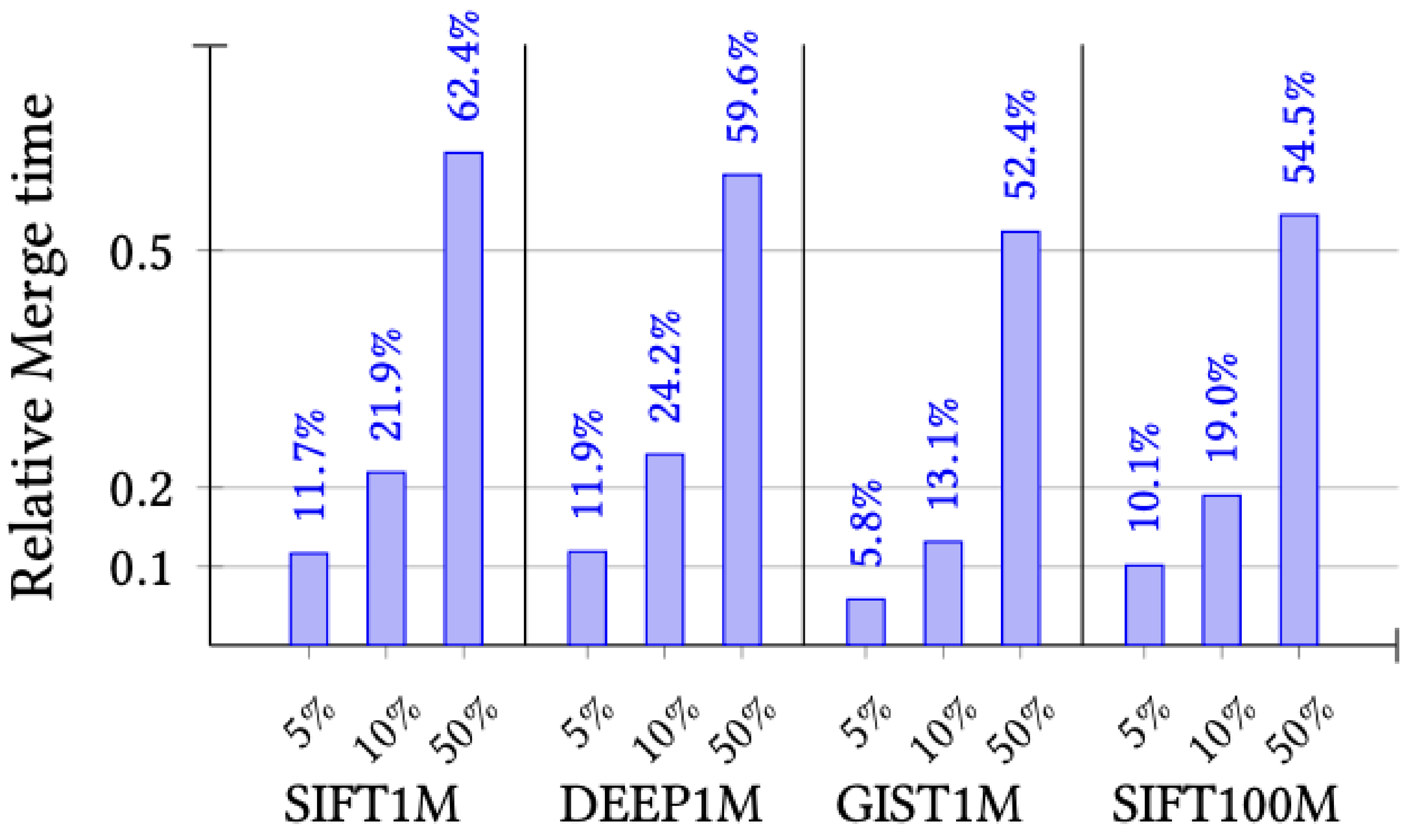


Figure 11. Time taken to merge delete and re-insert of 5%, 10%, and 50% of index size into a FreshVamana index, expressed relative to index rebuild time for Vamana.

Table 2. Full build time with DiskANN (96 threads) versus FreshDiskANN (40 threads) to update a 800M index with 30M inserts and deletes

Dataset	DiskANN(sec)	StreamingMerge (sec)
SIFT800M	83140 s	15832 s

Patching is great!

UNIVERSITY OF TORONTO

Results – Query Latency during StreamingMerge

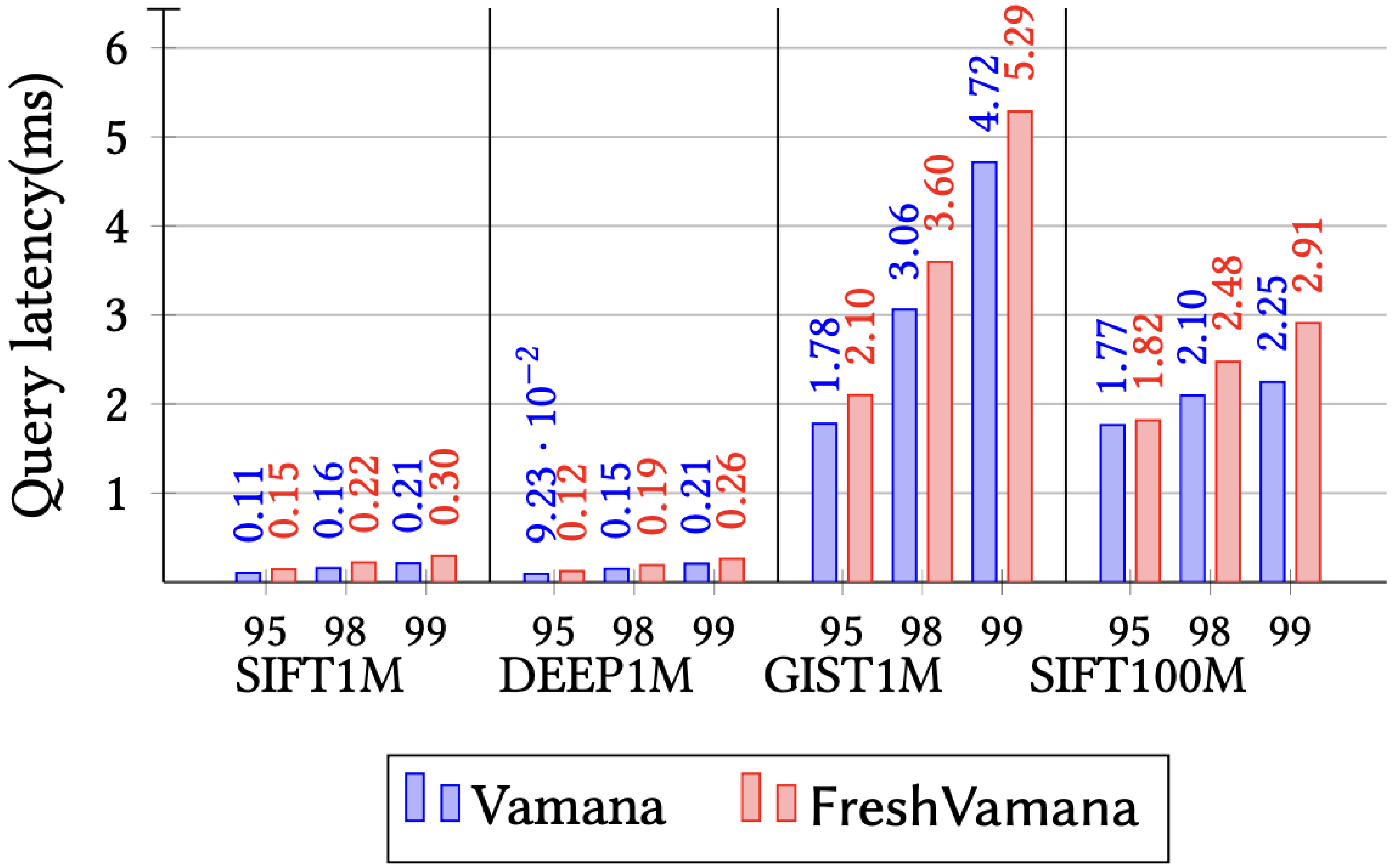


Figure 16. Query latency for Vamana and build-time normalized FreshVamana 10-recall@10 at 95%, 98%, and 99%.

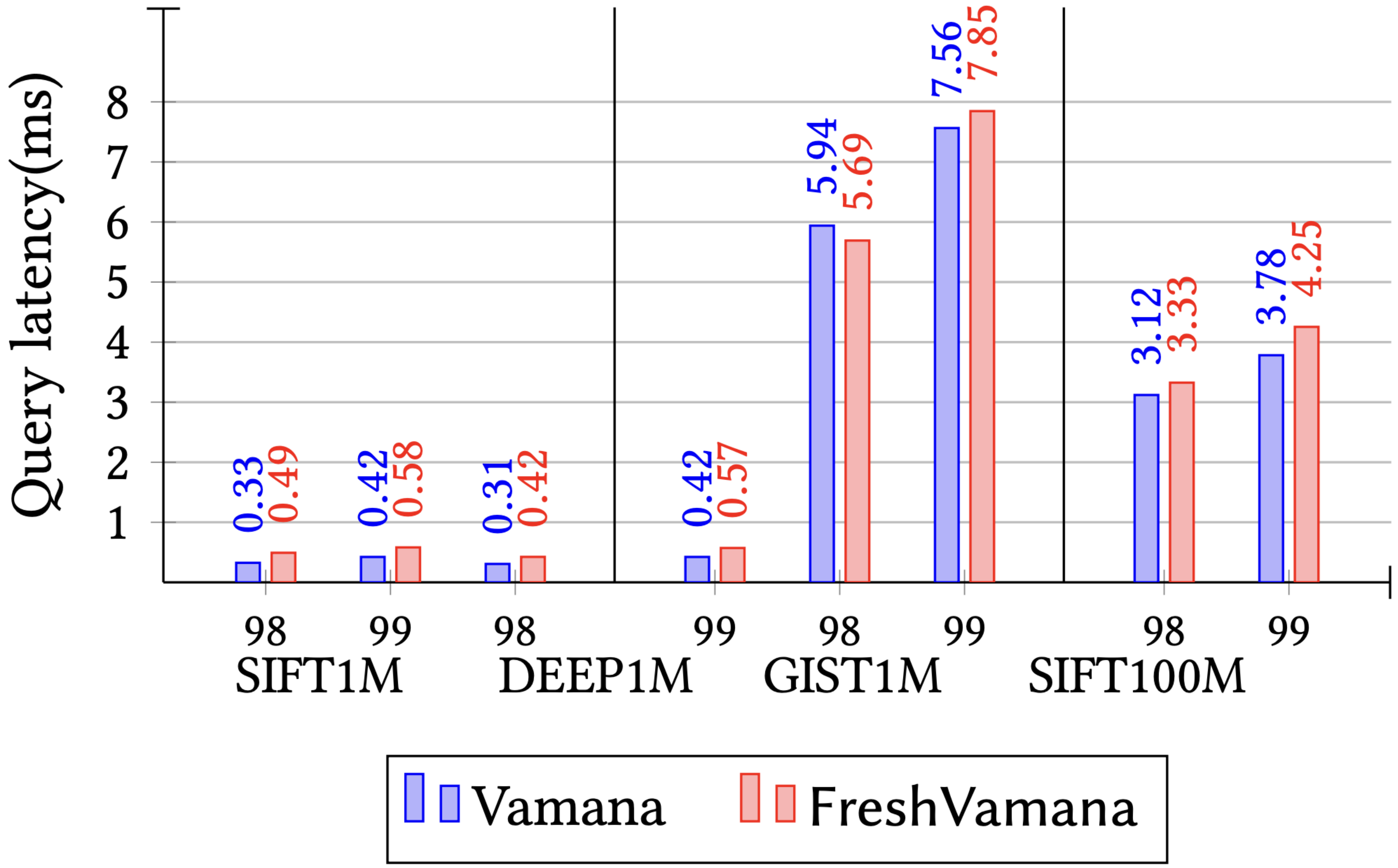


Figure 17. Query latency for Vamana and build-time normalized FreshVamana 100-recall@100 at 98%, and 99%.

UNIVERSITY OF TORONTO

FreshDiskANN vs. DiskANN

Category	FreshDiskANN	DiskANN
Updates	Real-time	Static
Index Design	Memory (temp) + SSD (long-term)	only SSD
Query Latency	Slightly higher	Baseline
Use cases	Dynamic, real-time applications	Static, pre-built datasets

UNIVERSITY OF TORONTO

Conclusion

- **Contributions**:
  - Scalable, real-time ANNS system
  - Efficient hybrid design: SSD + memory
  - High recall, low latency, and dynamic update in real-time
  - Suitable for dynamic environments
- **My feedback**:
  - Better explanation needed, NO architecture diagram
  - Not enough performance comparison with other ANNS (like HNSW or FAISS)
  - Figures are mixed

Thank you!

UNIVERSITY OF TORONTO