ParlayANN: Scalable and Deterministic Parallel Graph-Based Approximate Nearest Neighbor Search Algorithms

Magdalen Dobson Manohar, Zheqi shen, Guy E. Blelloch, Laxman Dhulipala, Yan Gu, Harsha Vardhan Simhadri, Yihan Sun

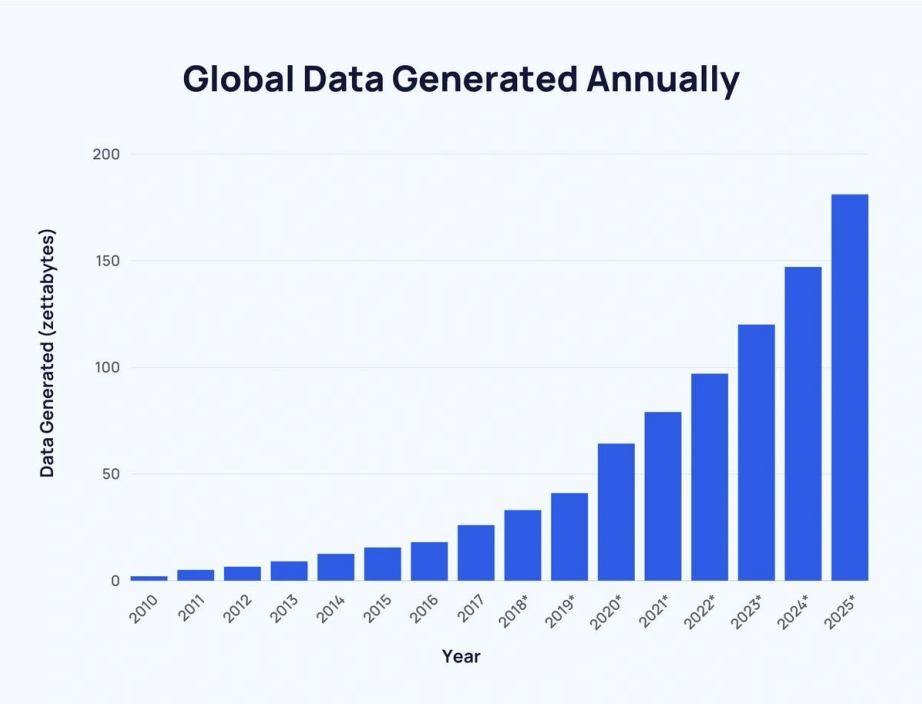
PPoPP 2024

2024. 10. 23

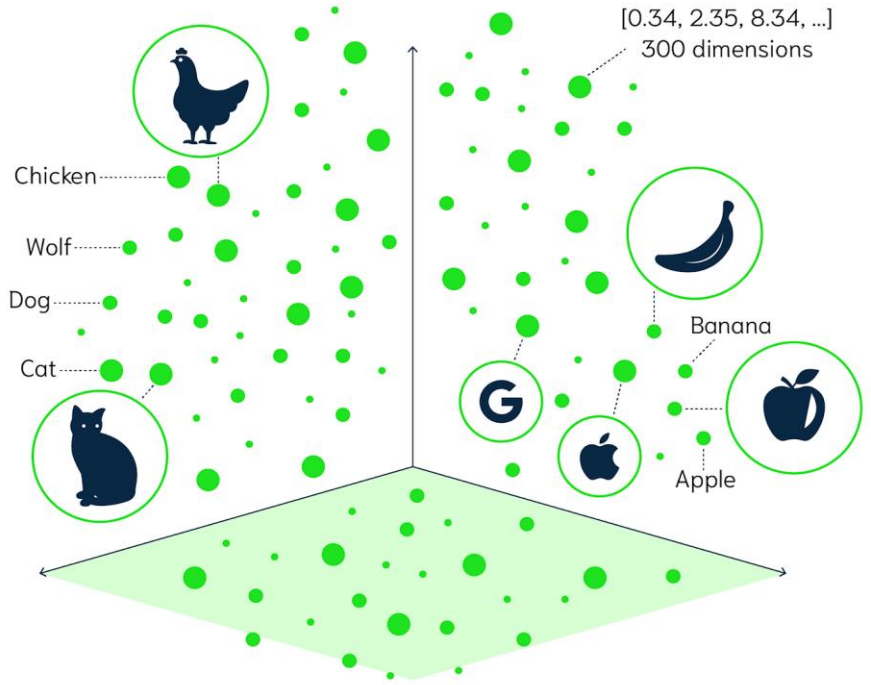
Hojin Shin (Dankook University)
ghwls03s@gmail.com or hojin03s@dankook.ac.kr

Contents

1. Introduction

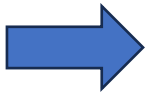


source:https://www.statista.com/statistics/871513/worldwide-data-created/



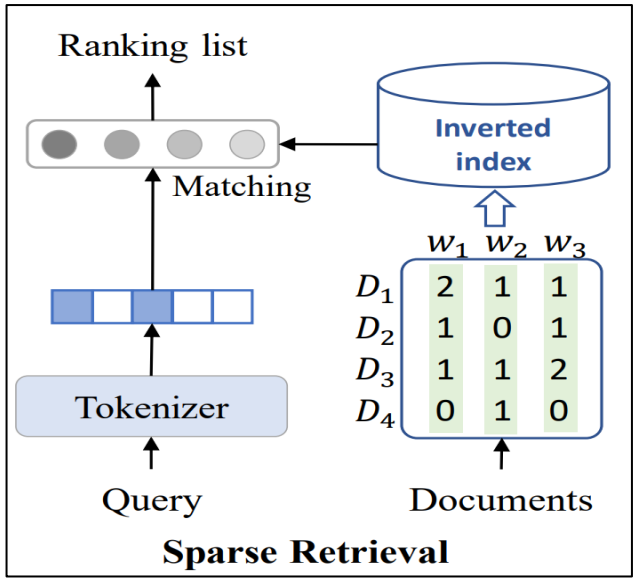
source:https://medium.com/@serkan_ozal/vector-similarity-search-53ed42b951d9

Constantly growing amount of the
available information resources



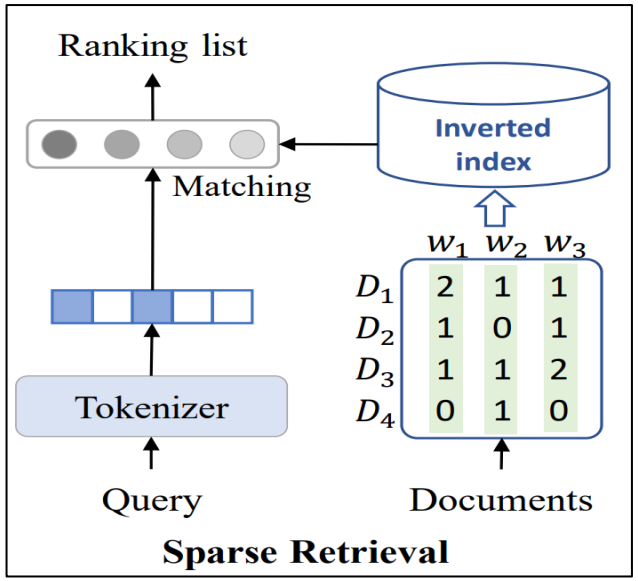
High demand in scalable and efficient
similarity search data structures

1. Introduction



Consider ***only some words*** in the target documents and search terms among a large number of words

1. Introduction



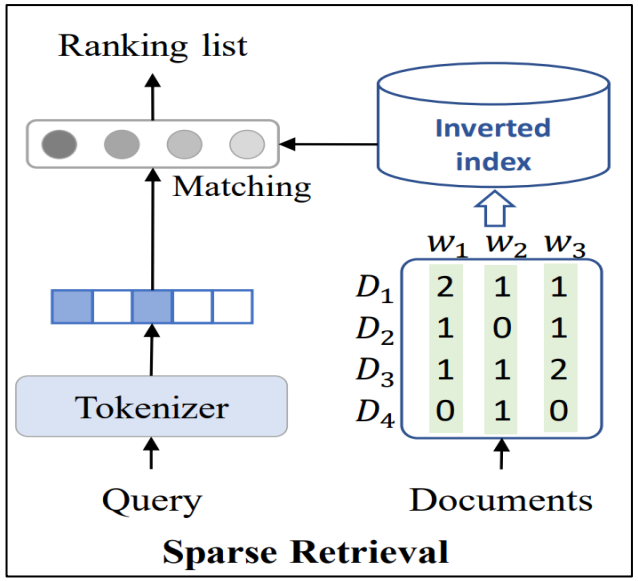
Consider ***only some words*** in the target documents and search terms among a large number of words

have ***similar meanings but do not match***

경기도민의 수는 얼마?

경기도의 인구는 00명

1. Introduction



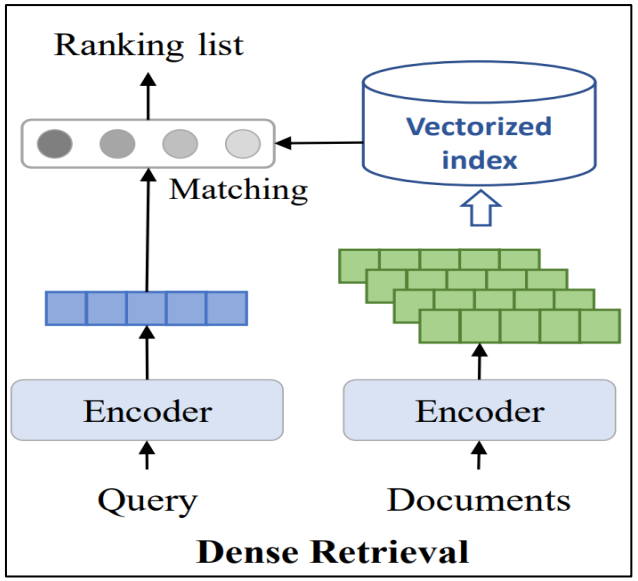
Consider ***only some words*** in the target documents and search terms among a large number of words

have ***similar meanings but do not match***

경기도민의 수는 얼마?

✘

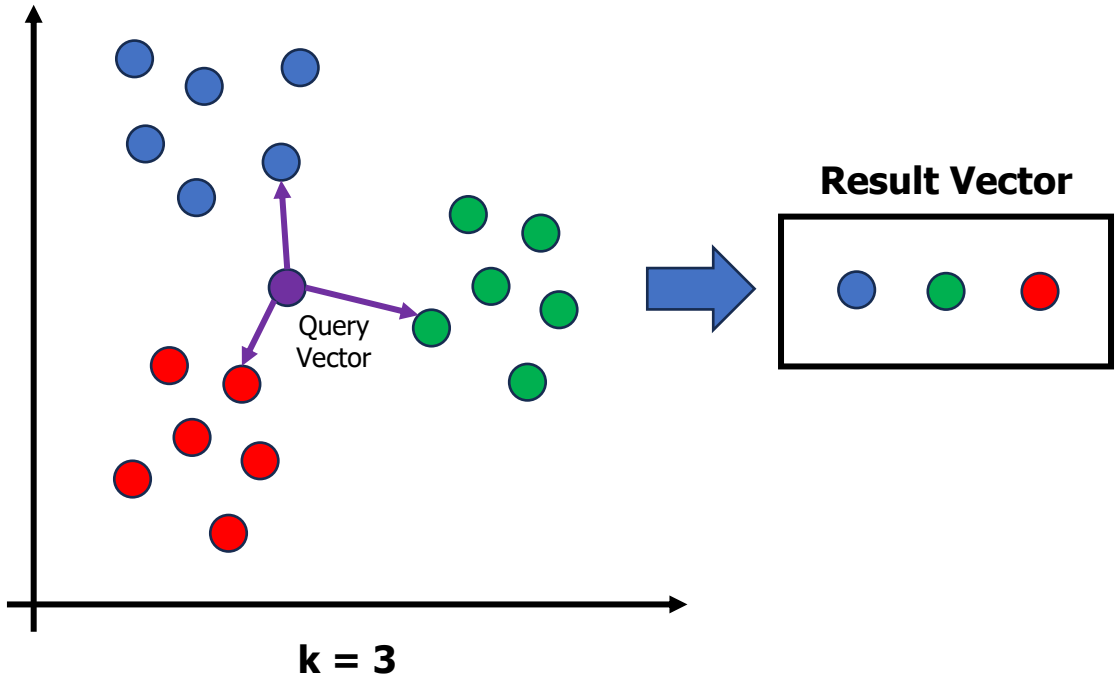
경기도의 인구는 00명



Continuous ***vector spaces***, capturing ***semantic similarity*** beyond exact term matching

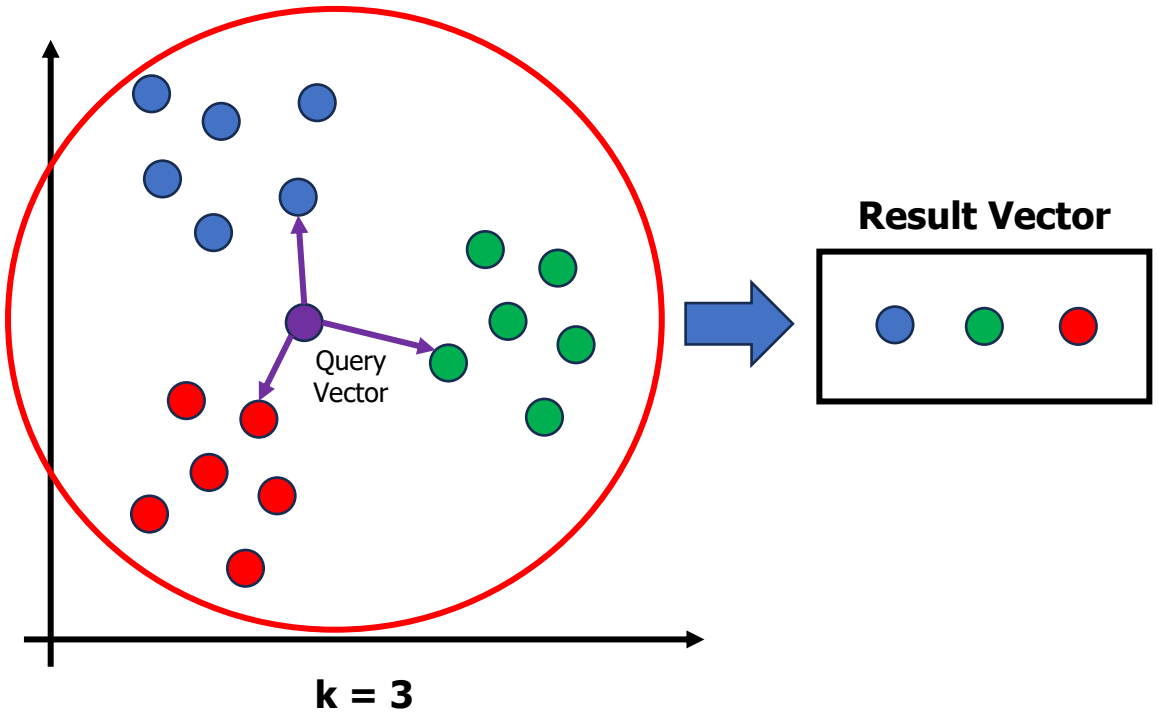
1. Introduction

Similarity Search (K-NNS)



1. Introduction

Similarity Search (K-NNS)



Strengths

Exact nearest neighbor search
➔ High ***Accuracy***

Limitations

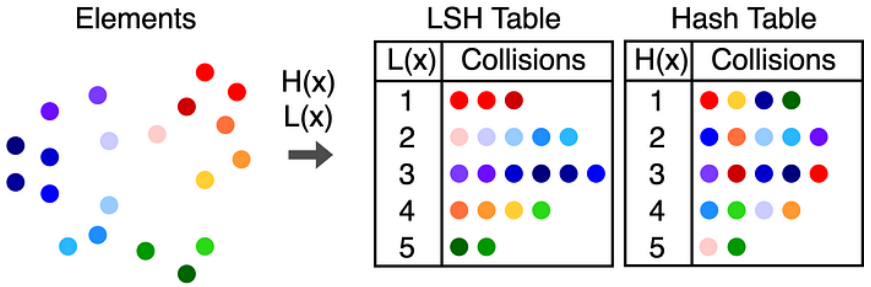
The number of stored elements making it ***infeasible*** for large-scale dataset (***High-D***)

➔ Whole search

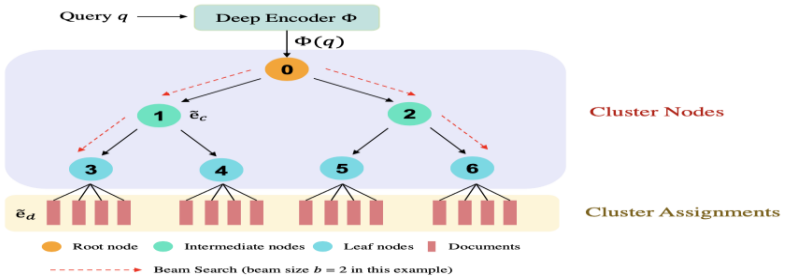
1. Introduction

Emergence of Approximate NNS

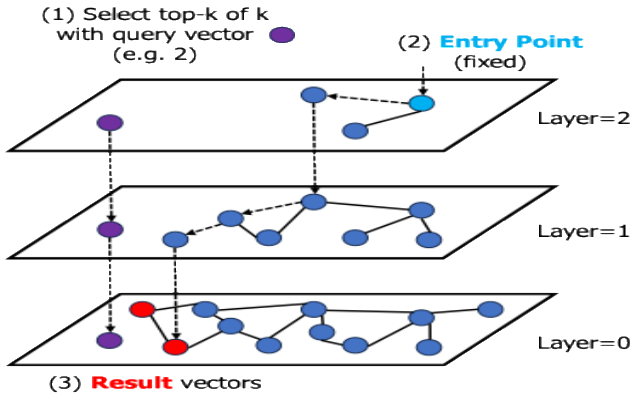
- Hash-based Approximate NNS
- Tree-based Approximate NNS
- Graph-based Approximate NNS



LSH



JTR

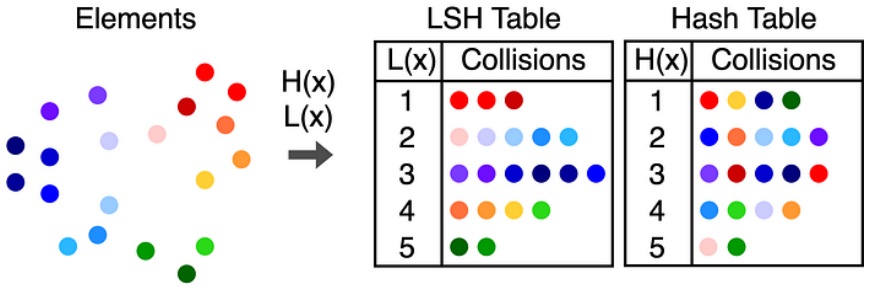


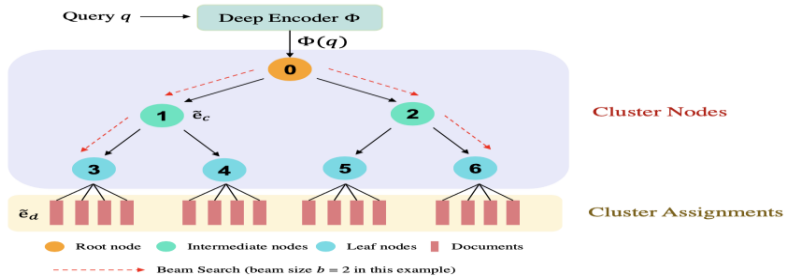
HNSW

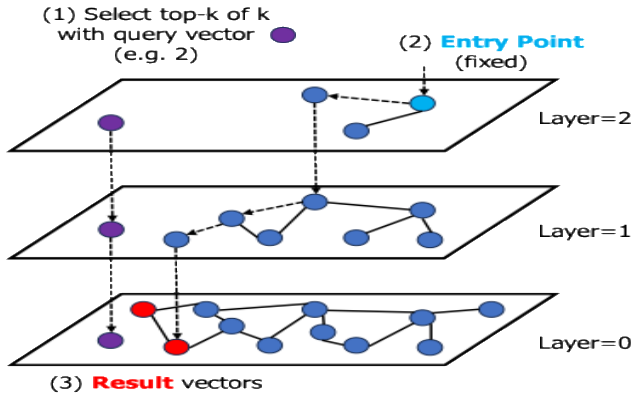
1. Introduction

Emergence of Approximate NNS

- Hash-based Approximate NNS → LSH
- Tree-based Approximate NNS → JTR
- Graph-based Approximate NNS → HNSW

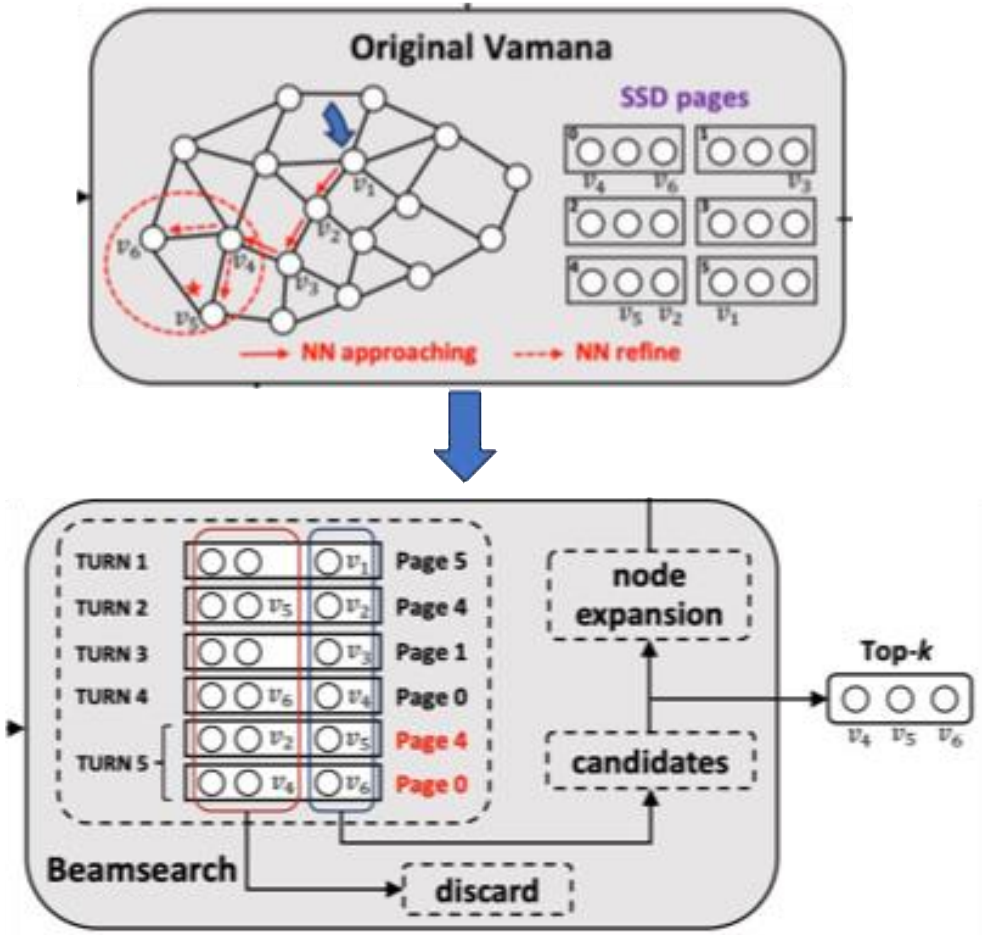




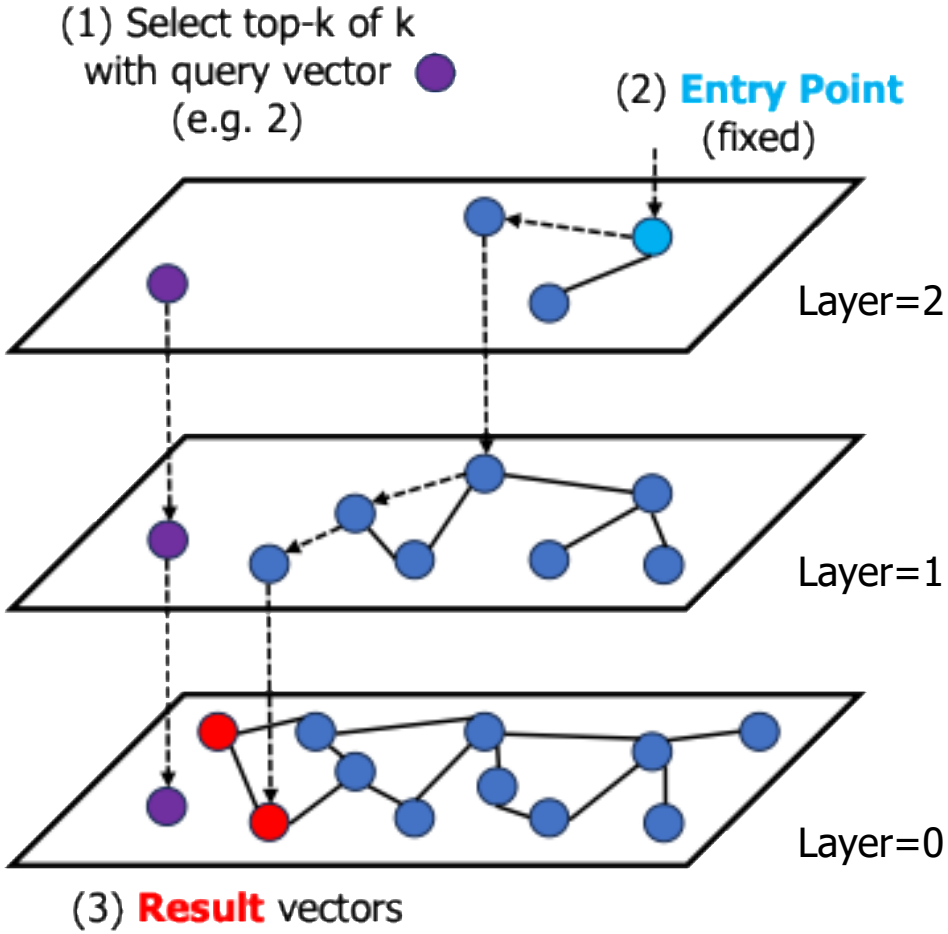


Graph ➔ best *throughput-recall* tradeoffs

2. Background

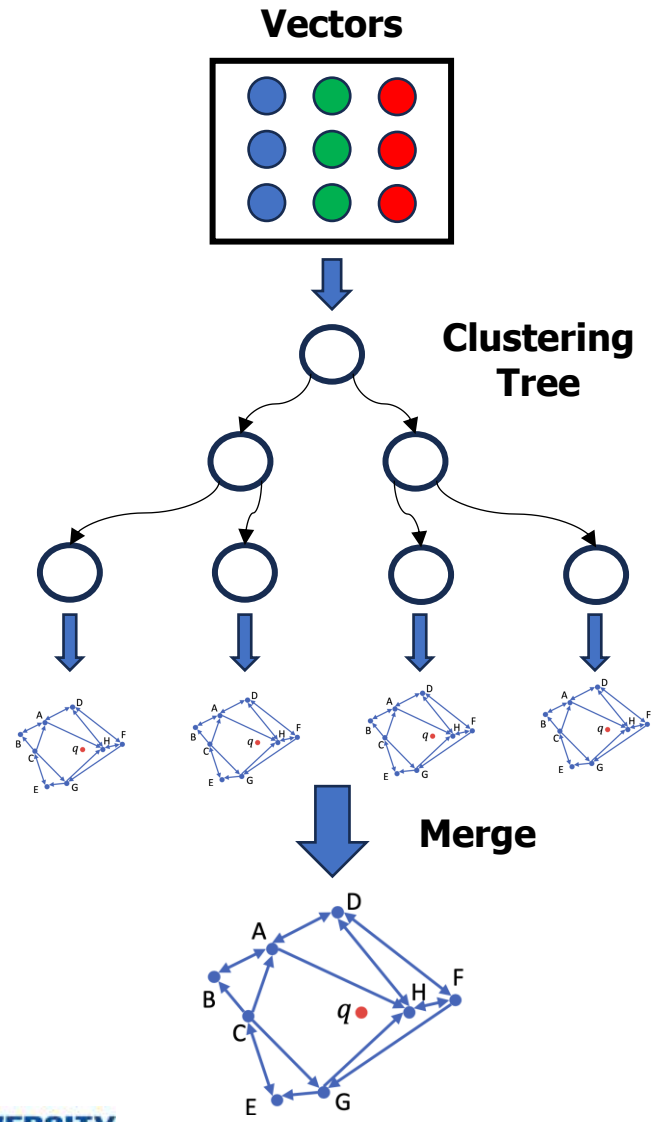


DiskANN (Vamana)



HNSW
Hierarchical Navigable Small World

2. Background



HCNNG (Hierarchical Clustering-based Nearest Neighbor Graph)

Hierarchical Clustering-Based Graphs for Large Scale Approximate Nearest Neighbor Search

Javier Vargas Muñoz[a,*], Marcos A. Gonçalves[b], Zanoni Dias[a], Ricardo da S. Torres[a]

[a] University of Campinas, Av. Albert Einstein, 1251, Campinas, São Paulo, Brazil
[b] Universidade Federal de Minas Gerais, Av. Antônio Carlos, 6627, Belo Horizonte, Minas Gerais, Brazil

PyNNDescent (Python Nearest Neigbhor Descent)

Efficient K-Nearest Neighbor Graph Construction for Generic Similarity Measures

Wei Dong wdong@cs.princeton.edu
Moses Charikar moses@cs.princeton.edu
Kai Li li@cs.princeton.edu

Department of Computer Science, Princeton University
35 Olden Street, Princeton, NJ 08540, USA

3. Motivation

Performance Issues on Large Datasets

- High-dimensional vector embeddings
  ➔ Essential element
- Transform into ***millions or billions*** of vectors
- Graph-based ANNS suffer from ***scalability*** issues on large datasets
- During parallel processing due to ***use of locks (per-point locks)***

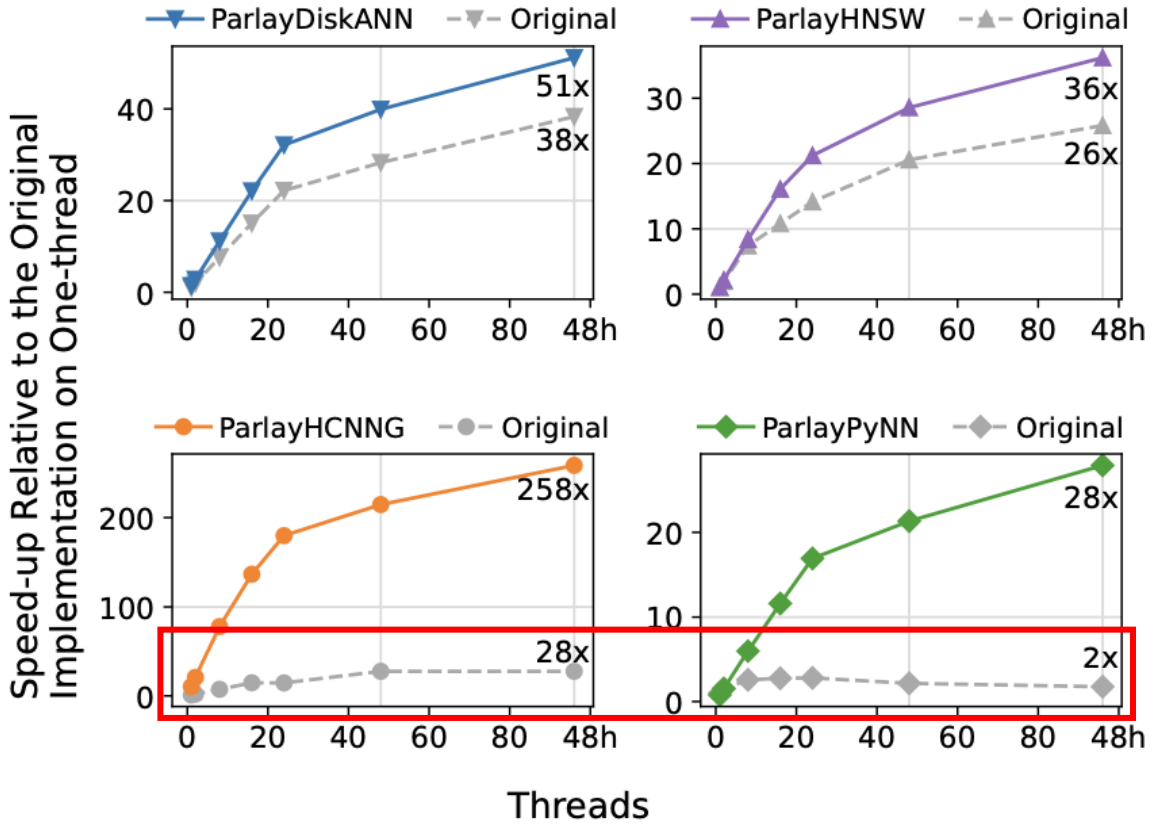
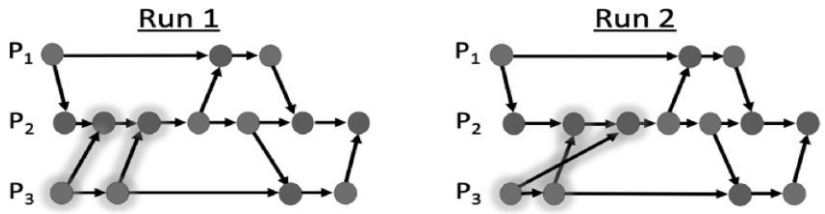


Figure 1. Scalability of original and our new implementations of four ANNS algorithms on various number of threads. Within each subfigure, all numbers are *speedup numbers relative to the original implementation on one thread*. Higher is better. Results were tested on a machine with 48 cores using dataset BIGANN-1M (10^6 points). "48h": 48 cores with hyperthreads. The two implementations in the same subfigure always use the same parameters and give similar query quality (recall-QPS curve).

3. Motivation

Use of Locks Limitations

- Existing parallel ANNS use locks to prevent ***data conflicts***
- Locks are effective in maintaining ***data consistency***
- The execution results depending on ***the order*** in which the locks are acquired
- Application (persistence, crash recovery)
- ***Non-determinism***



Existing Benchmarks Limitations

- Most of the existing studies evaluate relatively ***small dataset*** of millions of data

A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search

Mengzhao Wang[1], Xiaoliang Xu[1], Qiang Yue[1], Yuxiang Wang[1,*]
[1]Hangzhou Dianzi University, China
{mzwang,xxl,yq,lsswyx}@hdu.edu.cn

ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms*

Martin Aumüller[1], Erik Bernhardsson[2], and Alexander Faithfull[1]
[1] IT University of Copenhagen, Denmark, {maau,alef}@itu.dk
[2] Better, mail@erikbern.com

Table 3: Statistics of real-world datasets.

Dataset	Dimension	# Base	# Query	LID [34, 57]
UQ-V [5]	256	1,000,000	10,000	7.2
Msong [2]	420	992,272	200	9.5
Audio [4]	192	53,387	200	5.6
SIFT1M [1]	128	1,000,000	10,000	9.3
GIST1M [1]	960	1,000,000	1,000	18.9
Crawl [3]	300	1,989,995	10,000	15.7
GloVe [48]	100	1,183,514	10,000	20.0
Enron [75]	1,369	94,987	200	11.7

Dataset	Data/Query Points	Dimensionality	Metric
SIFT	1 000 000 / 10 000	128	Euclidean
GLOVE	1 183 514 / 10 000	100	Angular
NYTimes	234 791 / 10 000	256	Euclidean
Rand-Angular	1 000 000 / 1 000	128	Angular
SIFT-Hamming	1 000 000 / 10 000	256	Hamming
NYTimes-Hamming	234 791 / 10 000	128	Hamming

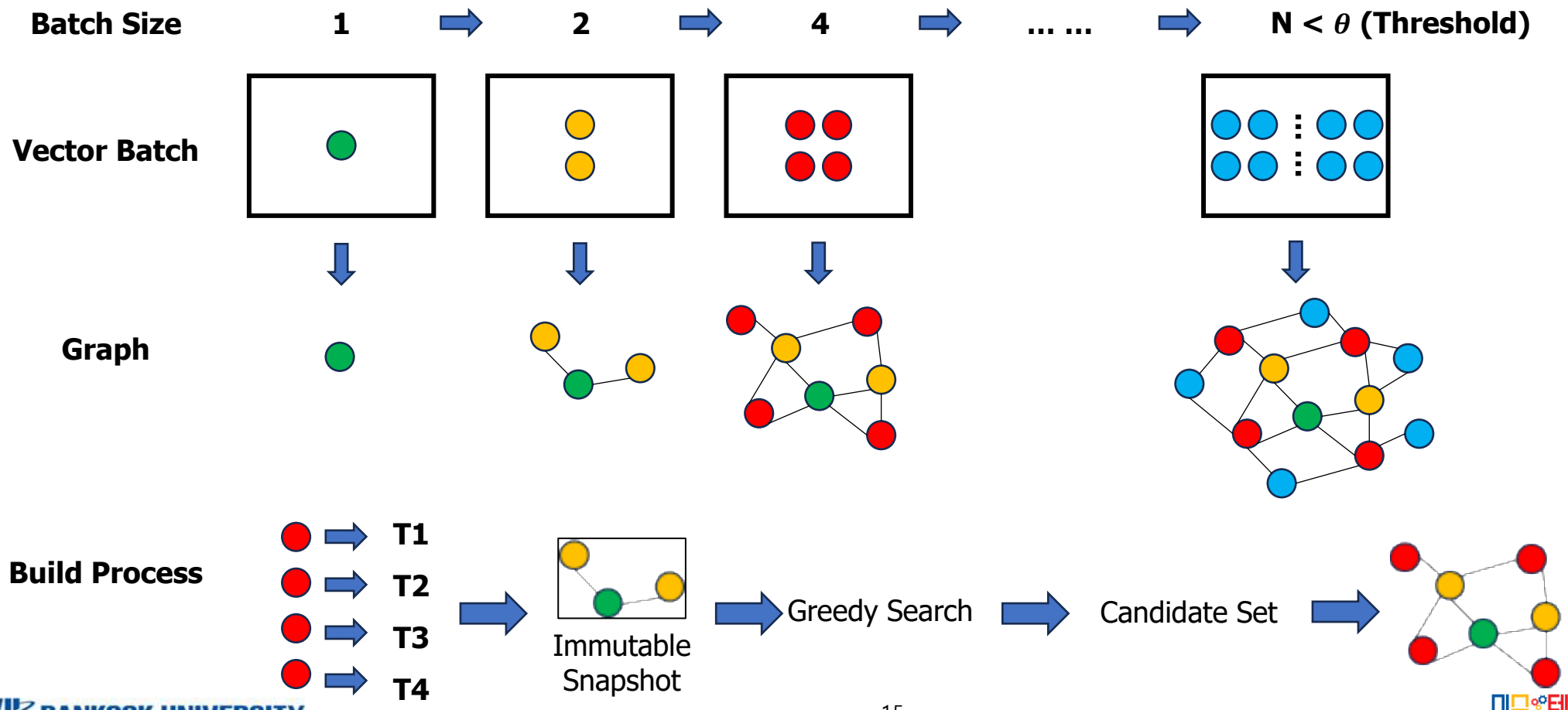
Table 3. Datasets under consideration

4. ParlayANN

New Techniques: *Prefix Doubling*

(1) Batch Insertion

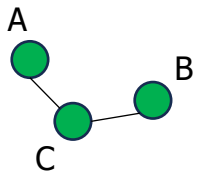
- Divide the data points into ***batches*** and insert them incrementally/independently



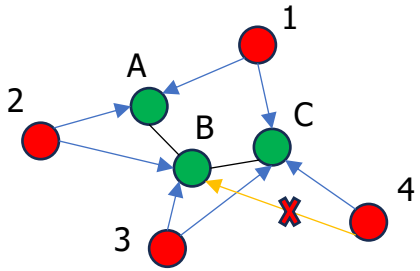
4. ParlayANN

(2) Pruning and SemiSort

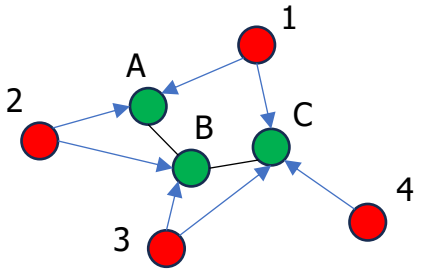
- Remove ***unnecessary*** connections and reduce the complexity of the graph
- ***Groups items by key***, arranging with the same key in a sequence



Existing Points



Pruning



SemiSort

Semisort Process

4. ParlayANN

Algorithm 3: batchBuild($\mathcal{P}$, s, R, L).

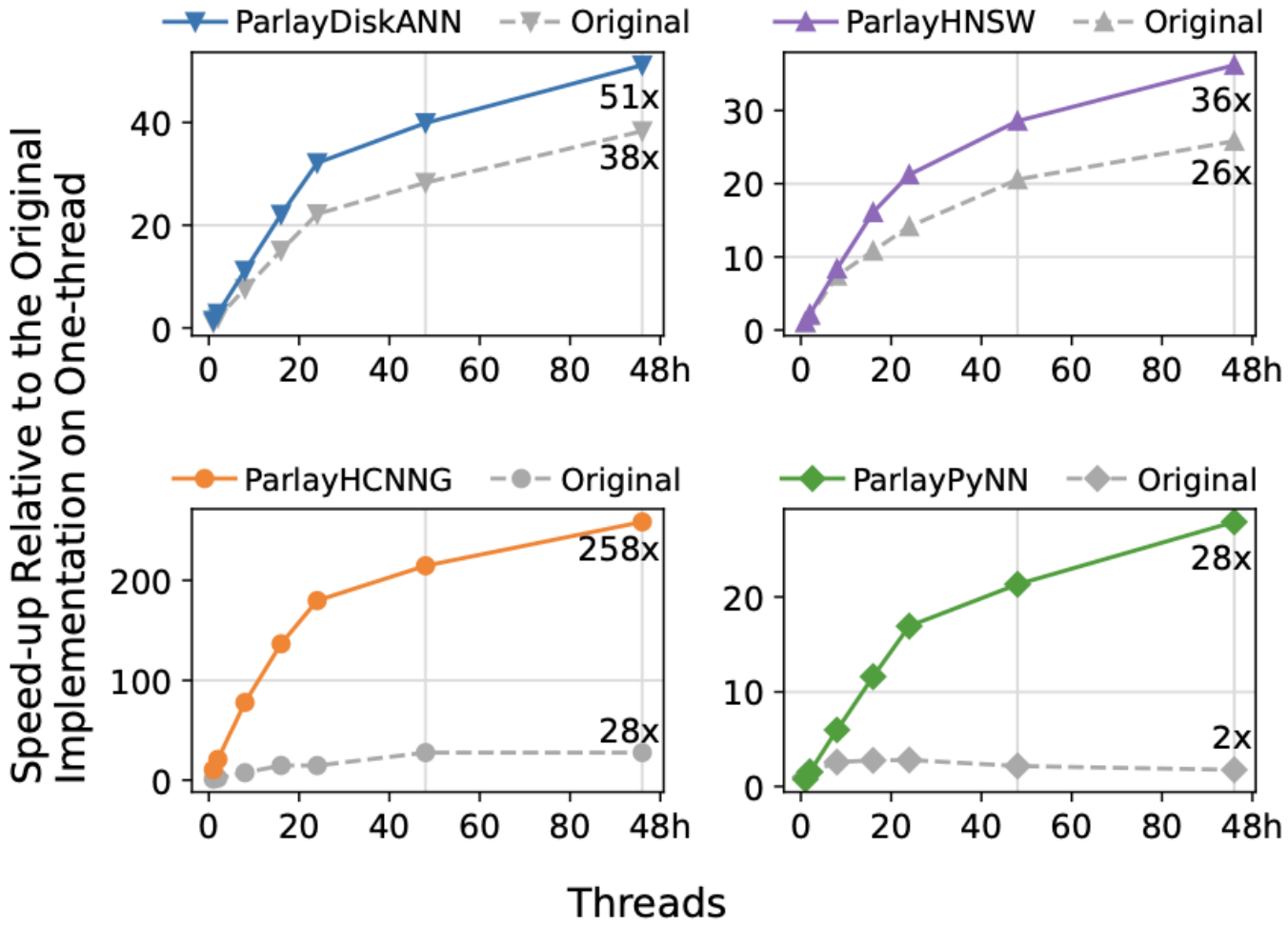
Input: Point set $\mathcal{P}$, starting point s, beam width L, degree bound R.

Output: An ANN graph consisting of all points in $\mathcal{P}$.

```
start ← 1
while start ≤ |P| do                                   // Prefix-doubling
    end ← min(start × 2, start + θ, |P|)               // θ: batch size upper bound
    BatchInsert(P[start..end])                         Batch Insertion
    start ← end + 1
Function BatchInsert(P')    // Insert a batch P' to the current index
    parallel for p ∈ P' do
        V, K ← greedySearch(p, s, L, 1)
        N_out(p) ← prune(p, V, R)                      Pruning
    B ← ⋃_{p∈P'} N_out(p)                              // All (existing) affected points
    parallel for b ∈ B do                              SemiSort
        // N: all points in P' that added b as their neighbors
        N ← {p | p ∈ P' ∧ b ∈ N_out(p)}
        N_out(b) ← N_out(b) ∪ N
        if |N_out(b)| > R then N_out(b) ← prune(b, N_out(b), R)
```

5. Evaluation

Comparison with multi-thread



5. Evaluation

Comparison with another library

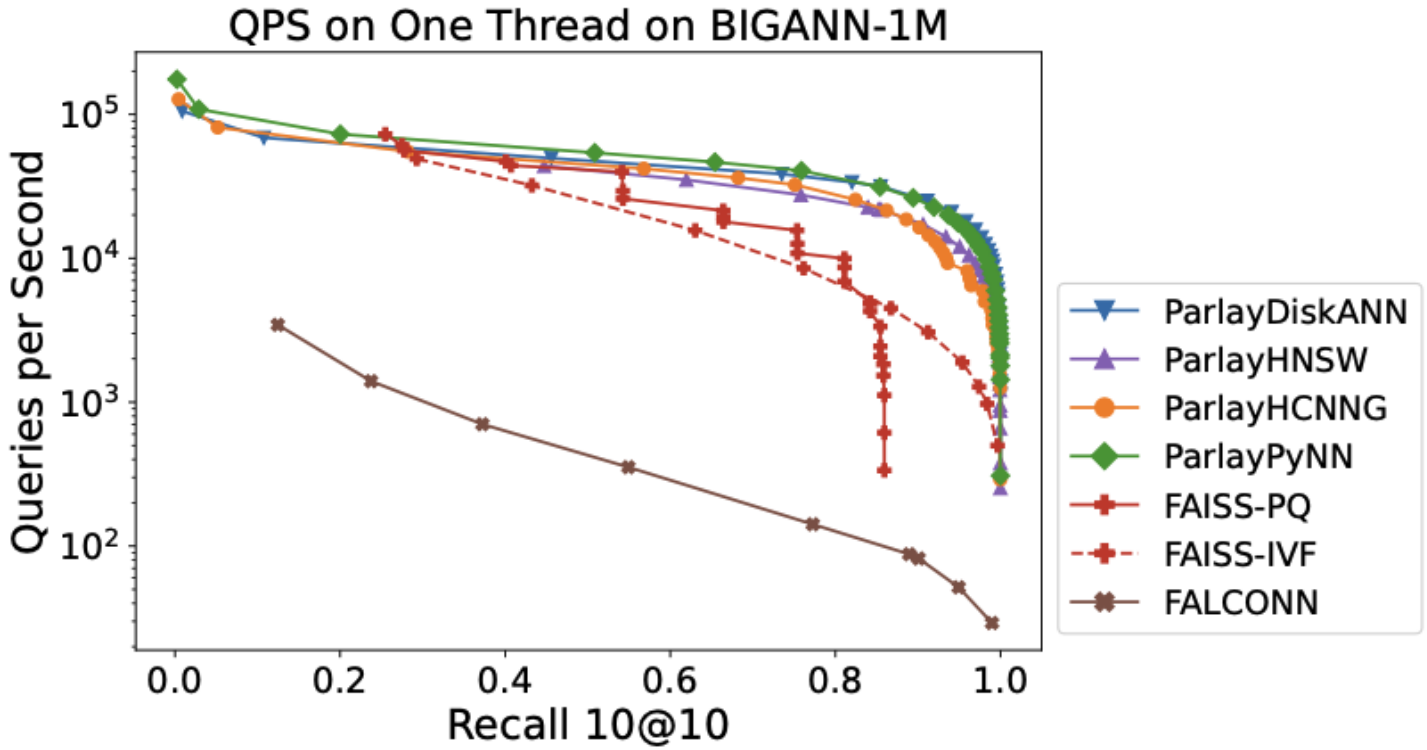
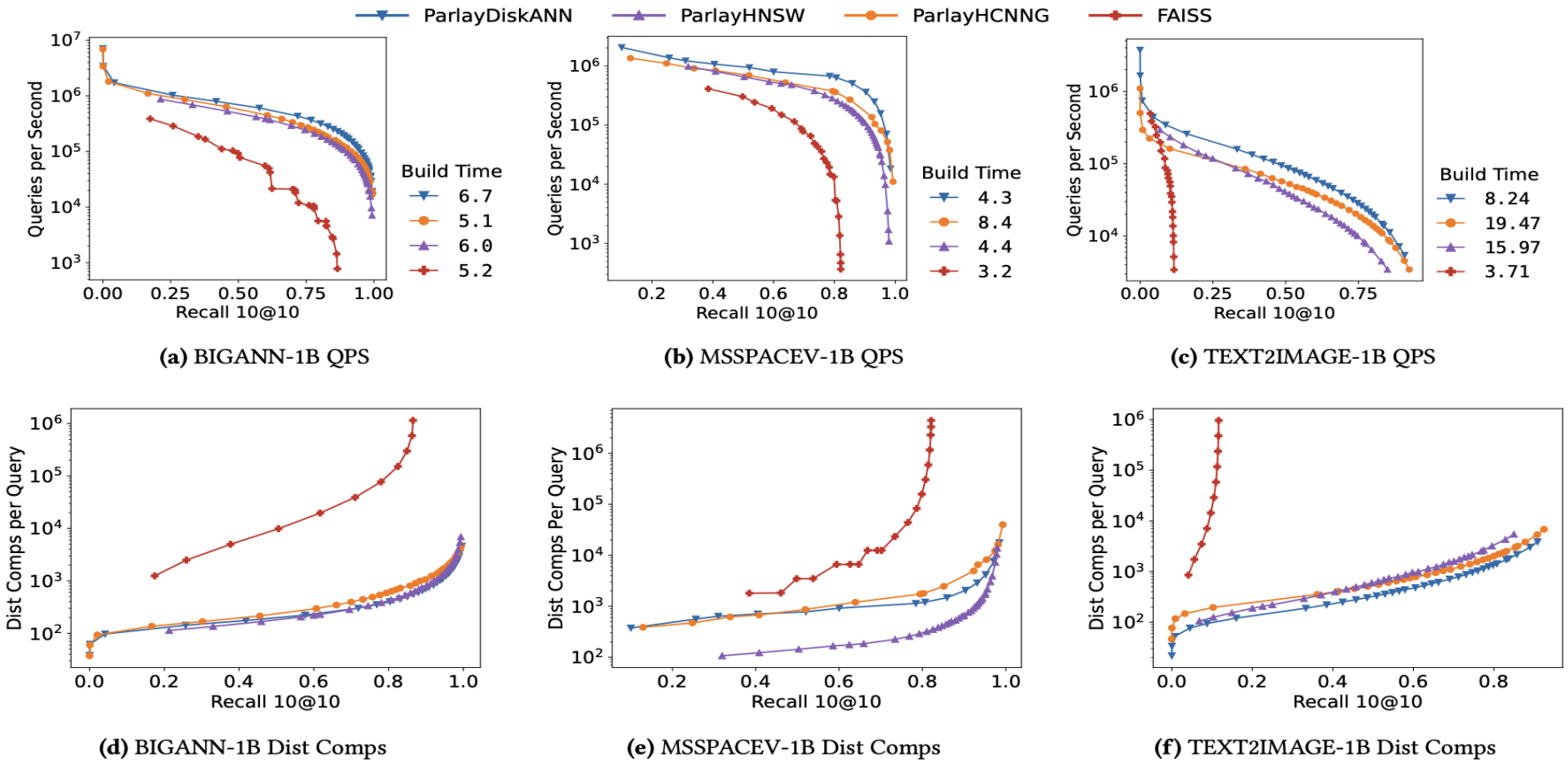


Figure 5. QPS on a single thread on BIGANN-1M. Shown to compare with ANN-benchmarks.

5. Evaluation

Comparison with Billion-scale Dataset



(a) BIGANN-1B QPS **(b)** MSSPACEV-1B QPS **(c)** TEXT2IMAGE-1B QPS

(d) BIGANN-1B Dist Comps **(e)** MSSPACEV-1B Dist Comps **(f)** TEXT2IMAGE-1B Dist Comps

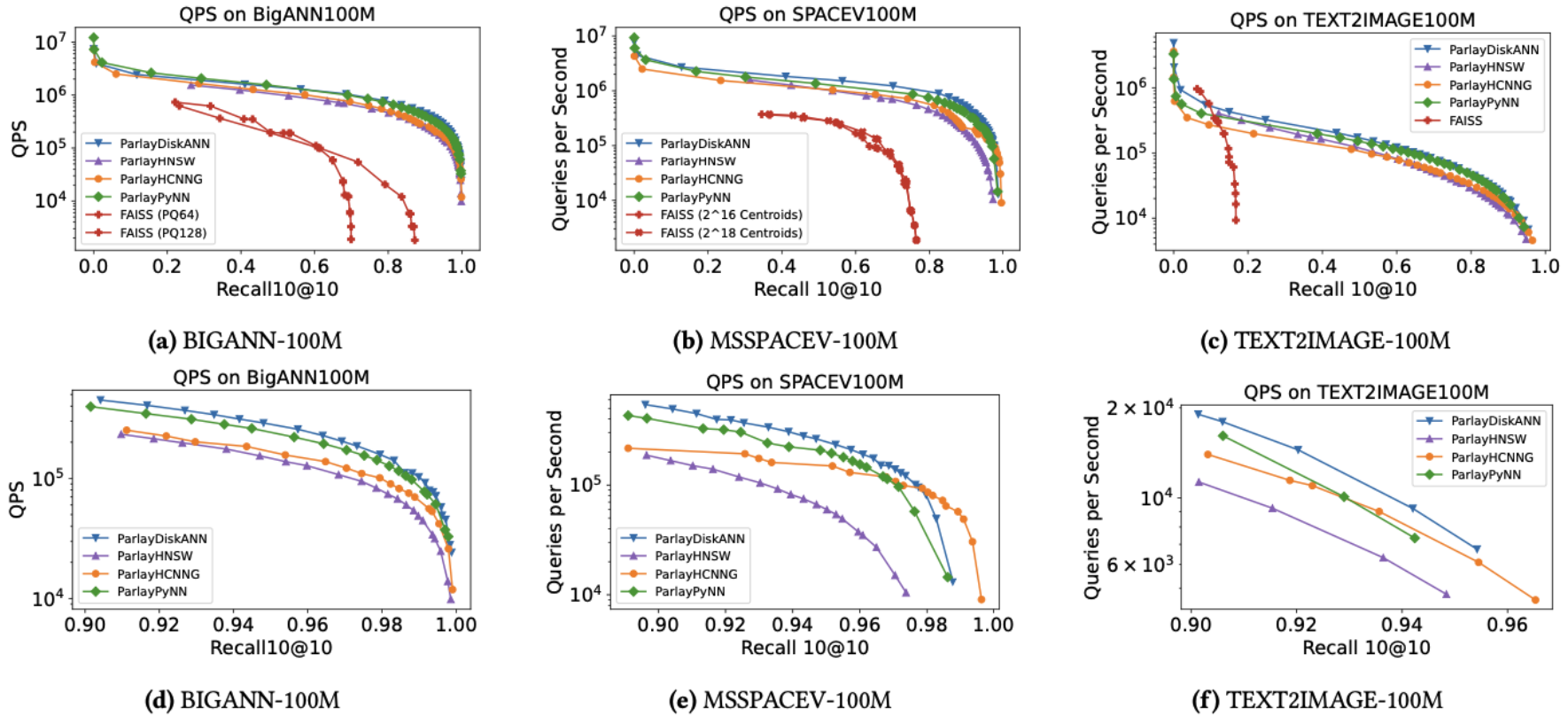
Figure 3. Build time (hours), QPS, recall, and distance comparisons for all algorithms on billion-size datasets.

5. Evaluation

Comparison with Million-scale Dataset

	BIGANN	MSSPACEV	TEXT2IMAGE
DiskANN	.42	.35	.70
HNSW	.35	.37	.94
HCNNG	.45	.77	1.75
pyNNDescent	.42	.73	1.23
FAISS	.19	.13	.22

Table 1. Build times (hours) on hundred million scale datasets.



(a) BIGANN-100M **(b)** MSSPACEV-100M **(c)** TEXT2IMAGE-100M

(d) BIGANN-100M **(e)** MSSPACEV-100M **(f)** TEXT2IMAGE-100M

Figure 4. QPS-recall curves on all 100-million size datasets. The first row shows the overall QPS/recall curve, while the second row zooms into a higher-recall regime. The build times are given in Tab. 1

6. Conclusion

Present **ParlayANN**, which implements four **parallel deterministic** graph-based ANNS algorithms

ParalyANN that scale to **billion-scale inputs** on a single machine with high recall

Avoid the use of locks, achieve better scalability than existing implementations

Outperformed existing **non-graph** implementations in the ability of achieving high recall

ParlayANN: Scalable and Deterministic Parallel Graph-Based Approximate Nearest Neighbor Search Algorithms

Magdalen Dobson Manohar, Zheqi shen, Guy E. Blelloch, Laxman Dhulipala, Yan Gu, Harsha Vardhan Simhadri, Yihan Sun

PPoPP 2024

Thank you !

2024. 10. 23
Hojin Shin (Dankook University)
ghwls03s@gmail.com or hojin03s@dankook.ac.kr