Accelerating Vector Search with RAPIDS RAFT

Summarizing the benefits, challenges, and possibilities with RAPIDS RAFT

Corey J. Nolet

About Me

- Past 5 years at Nvidia: Data scientist and principal engineer on the RAPIDS ML team
- Lead engineering for vector search, machine learning, and data mining primitives
- Prior to Nvidia: Built massive-scale exploratory data science and real-time analytics platforms for big-data and HPC environments in the defense industry.

Agenda

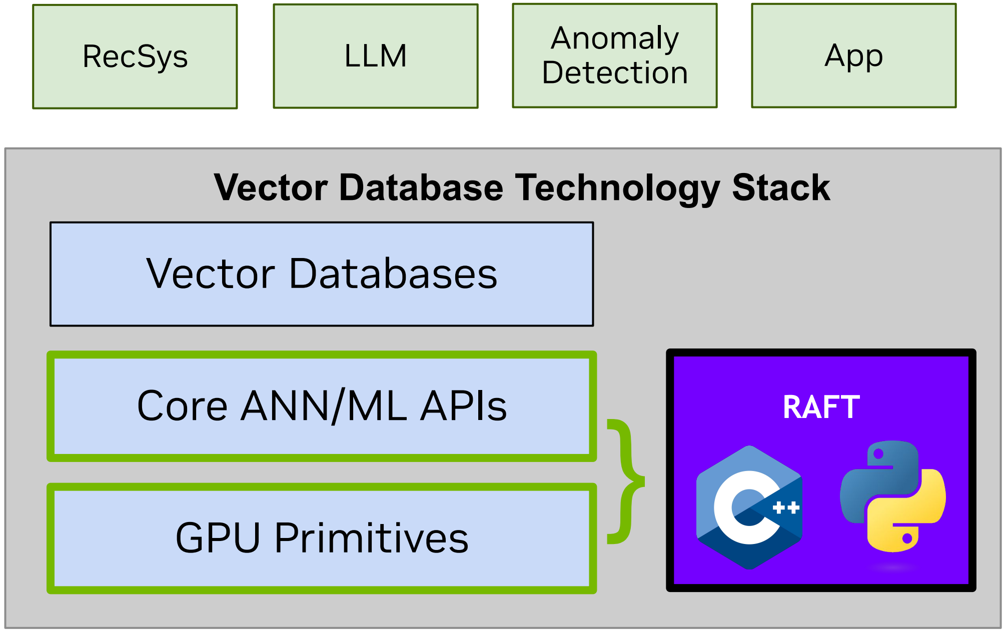
- What is RAPIDS RAFT?
- Benchmarking Performance
    - Price/Performance
- Notable algorithms in RAFT
- CAGRA
- Release Roadmap

What is RAPIDS RAFT?

RAPIDS RAFT Overview

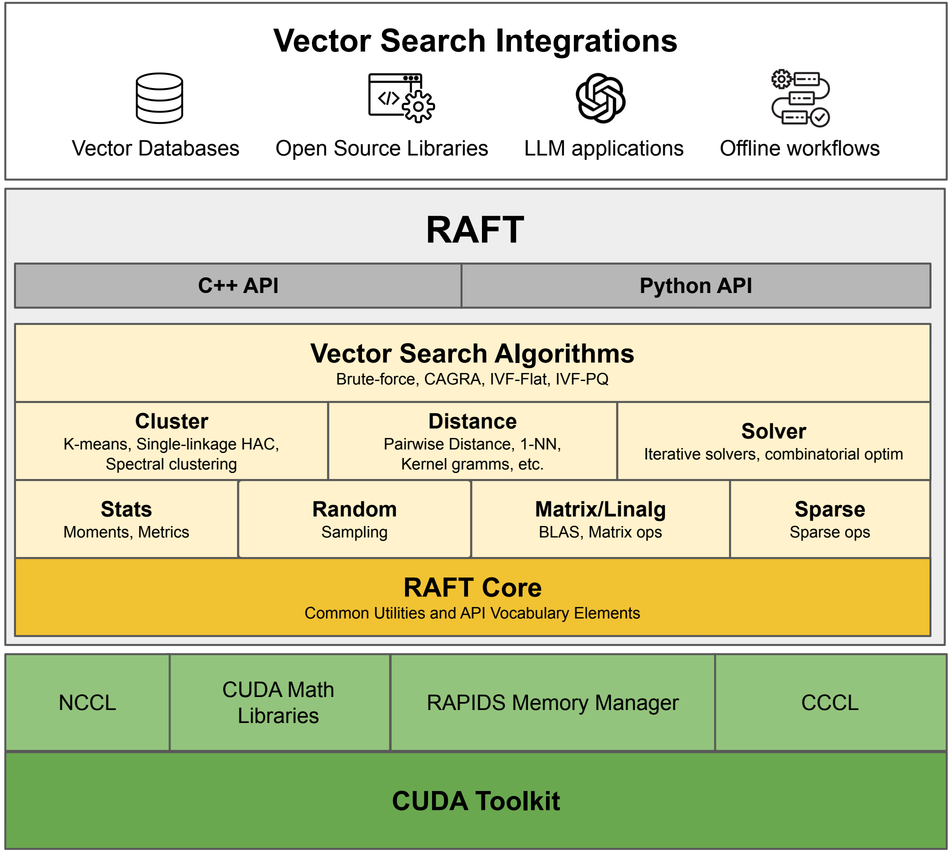
Accelerated, Composable Building Blocks for ML & Vector Search

- RAFT contains *ready-to-use APIs and composable building blocks*
  - Sparse and dense matrix operations, nearest neighbors, clustering, iterative solvers, and more…
- *Fastest* Approximate and Exact Nearest Neighbors
  - Core ANN APIs: IVF-PQ, IVF-Flat, CAGRA (graph-based)
- Friendly, consistent *C++17* and *Python* APIs with a header-only library and Apache 2.0 license



RAPIDS RAFT Overview

Toolbox of Accelerated, Composable Building Blocks for ML & Data Analytics



RMM

Unifying memory management across the GPGPU ecosystem

- Framework for defining ***composable*** memory allocation resources
- Unlocks ability to ***build end-to-end workflows***, comprised of different libraries, to share memory and allocators
- Centralized memory management provides ***zero-copy interoperability*** across different libraries
- Enables sharing a ***device memory pool*** across supported libraries in the ecosystem
- Working to bring RMM support to FAISS!

https://github.com/rapidsai/rmm

dask Numba CuPy RAPIDS PyTorch

A little background...

GPU-accelerated nearest neighbors at Nvidia

- 2018
  - Nvidia announces RAPIDS for GPU-accelerated data science!
  - RAPIDS cuML library starts using FAISS for nearest neighbors search on the GPU
    - Nearest neighbors and pairwise distances are useful for many ML algorithms- clustering, manifold learning, class imbalance, classification, pre-processing, filtering
- 2021
  - Nvidia joins Big-ann Benchmarks ‘21 competition and wins first place alongside Intel
- 2022
  - RAPIDS open sources Big-ann Benchmarks implementation through RAFT library
    - Initial implementations include IVF-Flat, IVF-PQ, random ball cover, and brute-force
  - FAISS agrees to use RAFT as a back-end for GPU-accelerated vector search
- 2023
  - RAPIDS introduces graph-based vector search algorithm CAGRA

Benchmarking

Methodology

Making a fair comparison between CPU and GPU

	General-purpose CPU	General-purpose GPU
Parallelism/per-core trade-off	Limited parallelism but faster pre-core	Massive parallelism but slower per-core
Concurrency implementation	Threads	CUDA streams
Best when used for:	I/O and fast general-purpose operation	Heavy compute and massive parallelism
Query performance	Threading	Batching (and CUDA streams)
Build performance	Threading	Batching

GPUs excel at tasks that require ***high data throughput*** or ***low latency.***

Methodology

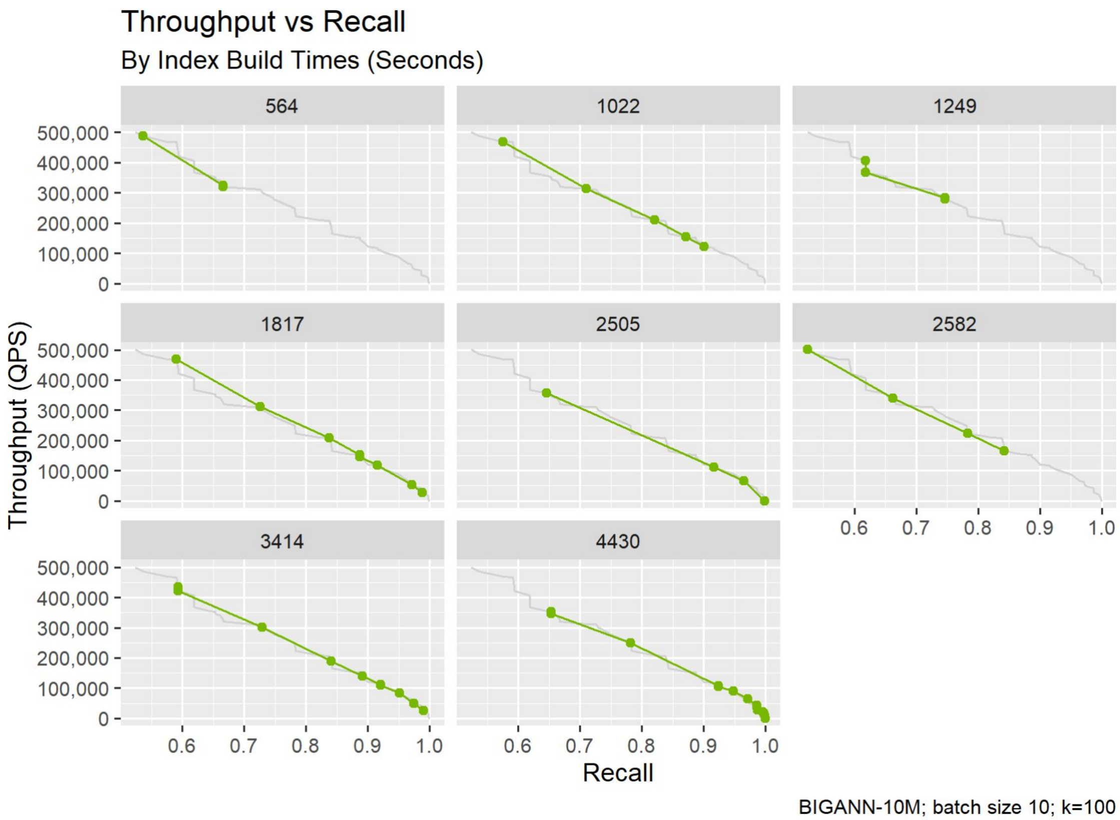
Making a fair comparison between CPU and GPU

In general, we

- measure both latency (single-threaded one-at-a-time) and throughput (saturate available hardware)
- compare CPU single-query at a time to GPU at different batch sizes (usually 1, 10, 100, 10k)
- don’t measure time to copy queries to device memory (on the order of single-digit microseconds)
- always compare index build times based on achieved throughput/latency and recall levels
- compare end-to-end walltime for both latency and throughput (to make sure we don’t ignore CPU idle time)

Measuring index build times

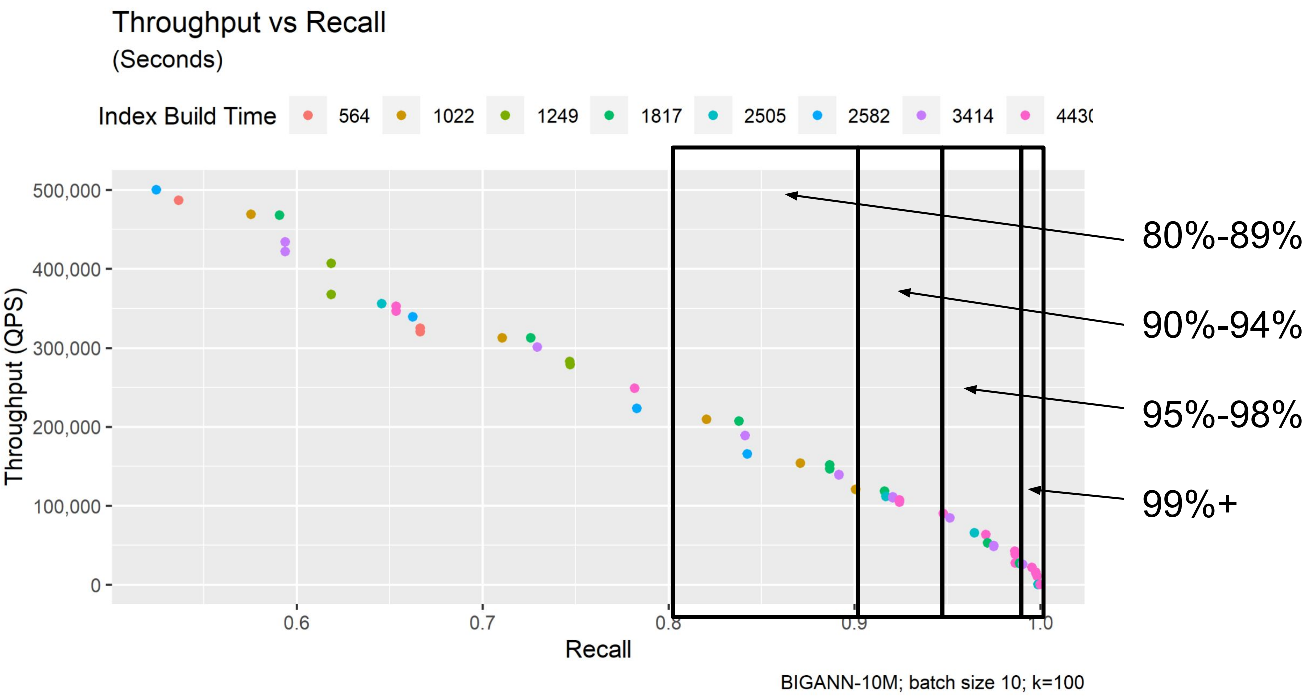
Index build times for HNSW on Big-ANN 10M



Which one's the most fair to report?

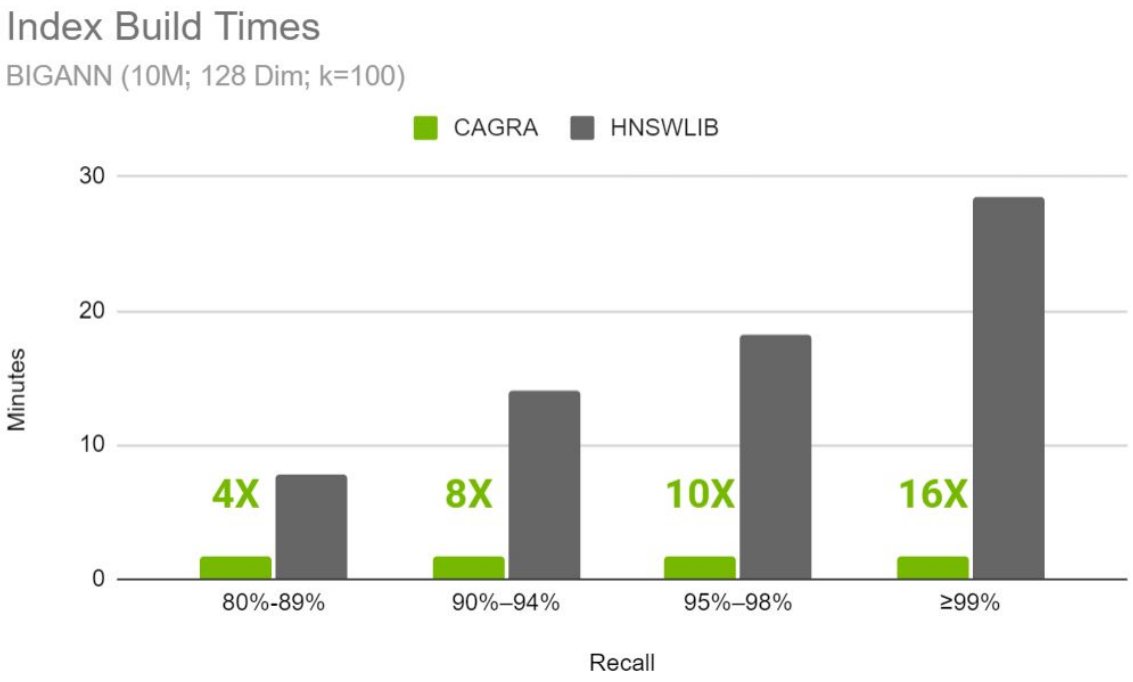
Measuring index build times

Index build times for HNSW on Big-ANN 10M



Measuring index build times

Index build times for HNSW on Big-ANN 10M



Average build times across target recall windows.

Measuring search times

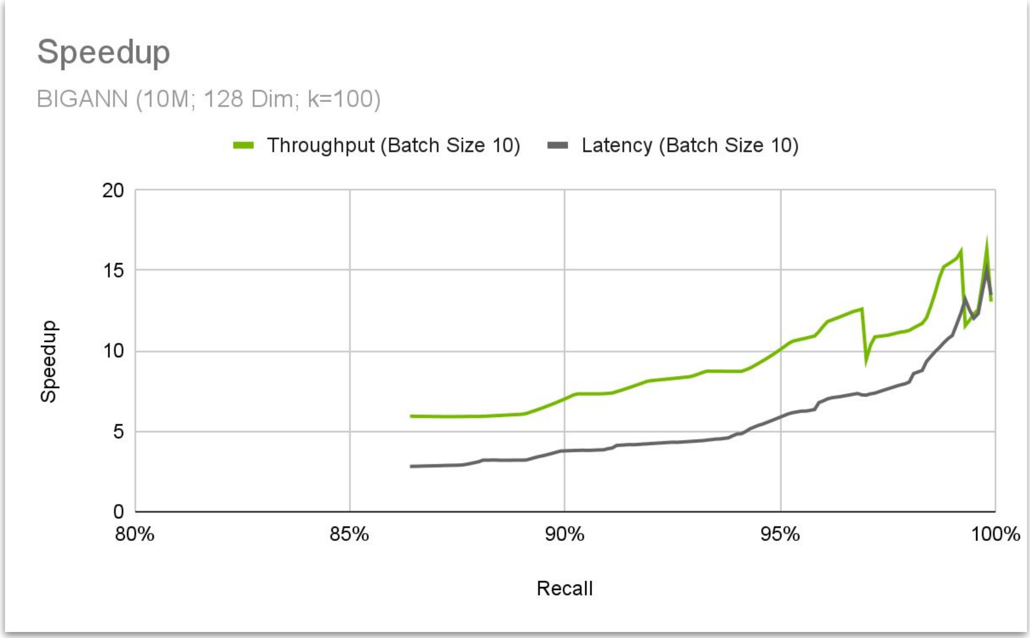
Search times for CAGRA and HNSW on Big-ANN 10M

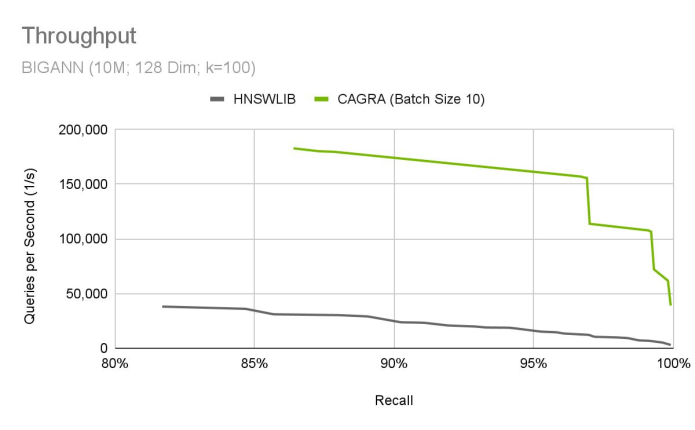
	CPU	GPU
Instance	r6g.4xlarge	g5.2xlarge
RAM	128 Gb	32 Gb
vCPU	16	8
GPU	–	A10G
GPU Memory	–	24 Gb
Price	$0.8064	$1.212

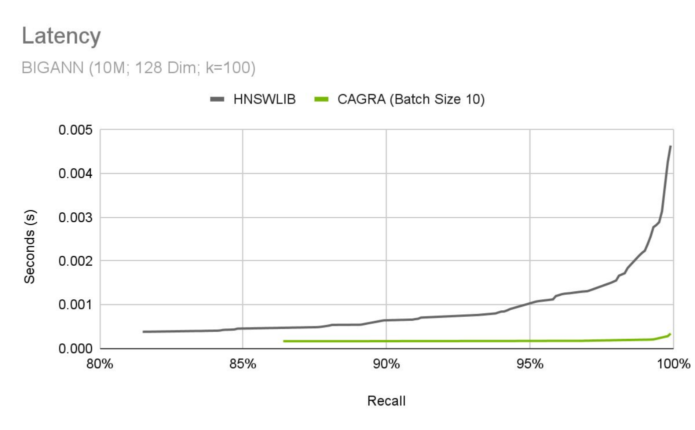
GPU($) / CPU($) = 1.50

Measuring search times

Search times for CAGRA and HNSW on Big-ANN 10M

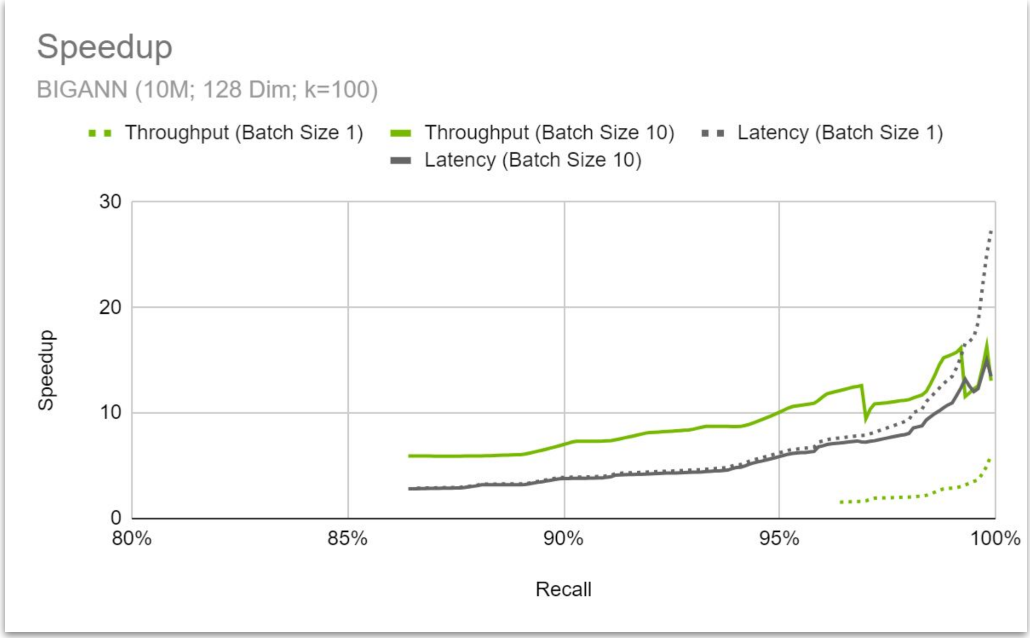


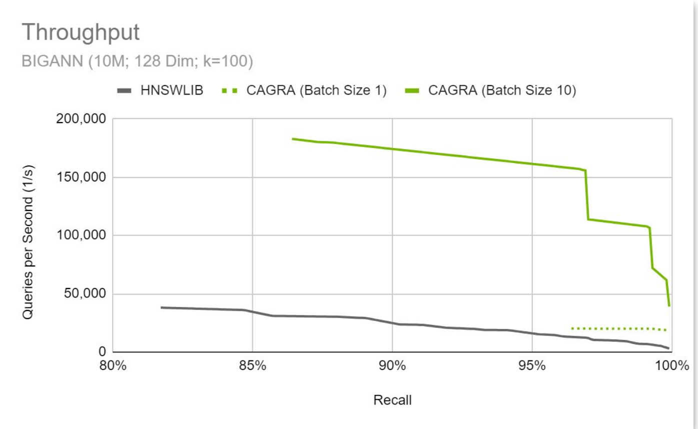


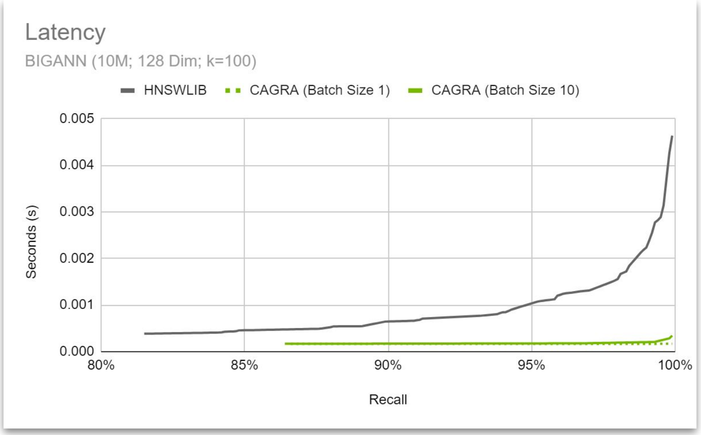


Measuring search times

Search times for CAGRA and HNSW on Big-ANN 10M







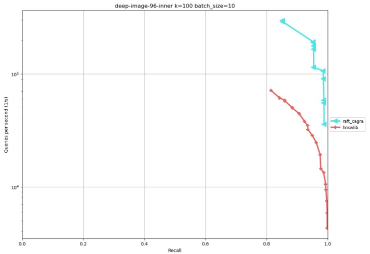
RAFT ANN Benchmarks

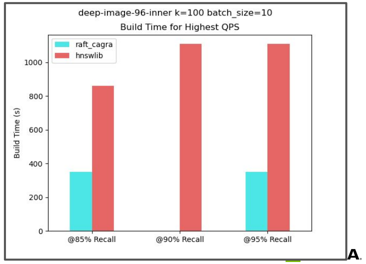
Reproducible benchmarking for state-of-the-art ANN comparison

- *CUDA-friendly* reproducible benchmarking tool to compare state-of-the-art ANN implementations at C++ level
- Heavily inspired by https://ann-benchmarks.com/
- *Conda* package and *Docker* containers available
- Measures both ***latency*** and ***throughput*** by saturating hardware
- Tools for users to ***reproduce ANN benchmarks*** on their own hardware, data, and algorithms.
- Learn more in the RAFT ANN Benchmarks documentation

```
name: raft_ivf_pq
groups:
  base:
    build:
      nlist: [500, 1024, 1648, 3200, 6400, 100000]
      pq_dim: [128, 64, 32]
      pq_bits: [8, 6]
      ratio: [1]
      niter: [25]
    search:
      nprobe: [1, 5, 10, 50, 100, 200, 500, 1000, 2000]
      internalDistanceDtype: ["float", "half"]
      smemLutDtype: ["float", "fp8", "half"]
```

Produces standardized charts and CSV files to compare performance





Wiki-all Dataset

Benchmarking Vector Search for RAG/LLM at Scale

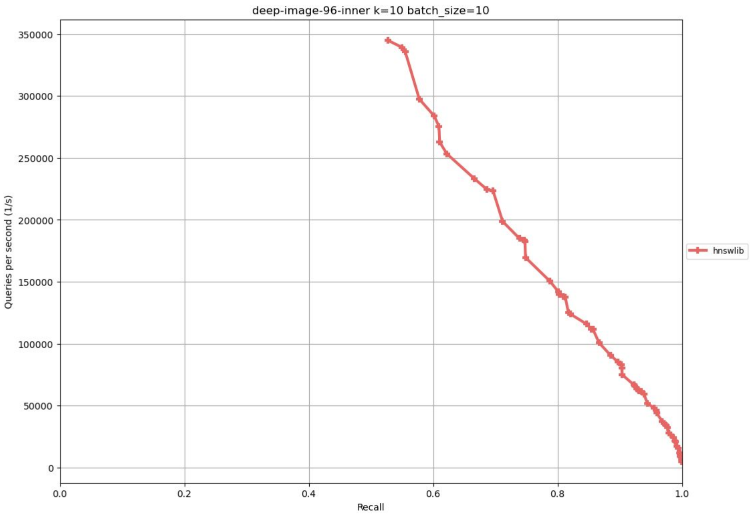
- Composed of *English* wiki texts from Kaggle and *multi-lingual* wiki texts from Cohere wikipedia.
- For testing *at scale* with *large dimensions*
  - Full dataset larger than a single GPU
  - Forces distributed or out-of-core solutions
- 768 dimensional dataset of *LLM embeddings* to benchmark vector search for RAG/LLM
- *Supported by RAFT* ANN Benchmarking tool
- Free and publicly available: https://docs.rapids.ai/api/raft/nightly/wiki_all_dataset/

# Vectors	Size
88M	**251GB**
10M	29GB
1M	2.9GB

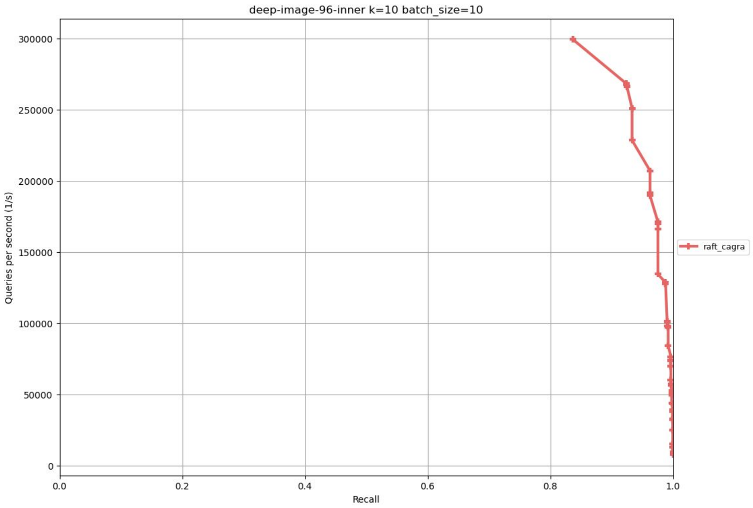
Price / Performance

Deep 10M | throughput | price-perf

C2-standard-30
Intel Cascade Lake 15-core
$1.32/hr



G2-standard-32
Nvidia L4
$1.80/hr

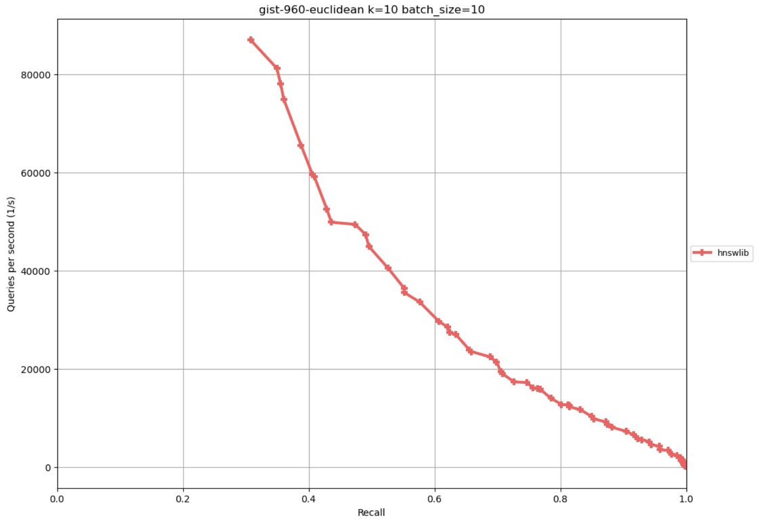


Same performance for ~57% less

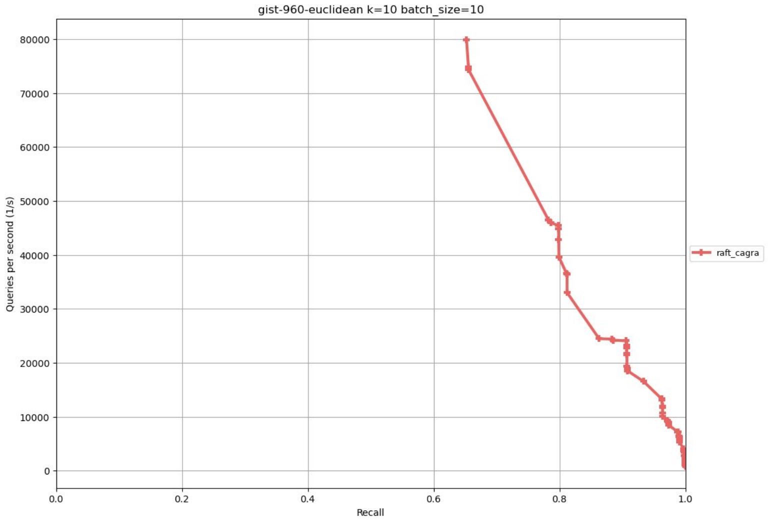
Gist | throughput | price-perf

K: 10

C2-standard-60
Intel Cascade Lake 30-core
$2.57/hr

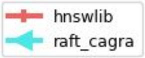


G2-standard-32
Nvidia L4
$1.80/hr



Same performance for ~65% less

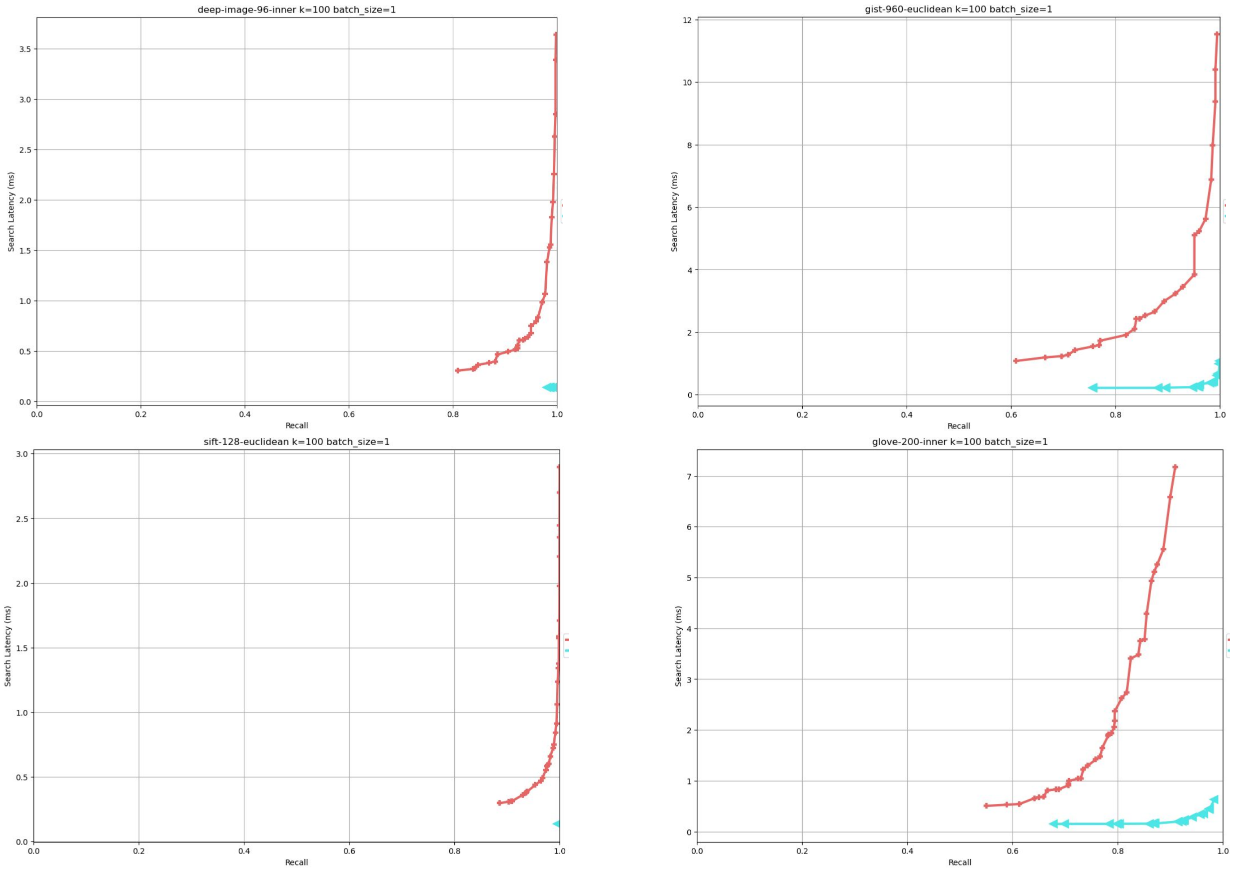
latency | price-perf



RAFT CAGRA
G2-standard-32
Nvidia L4
$1.80/hr

HNSWLIB
C2-standard-30
Intel Cascade Lake 15-core
$1.32/hr

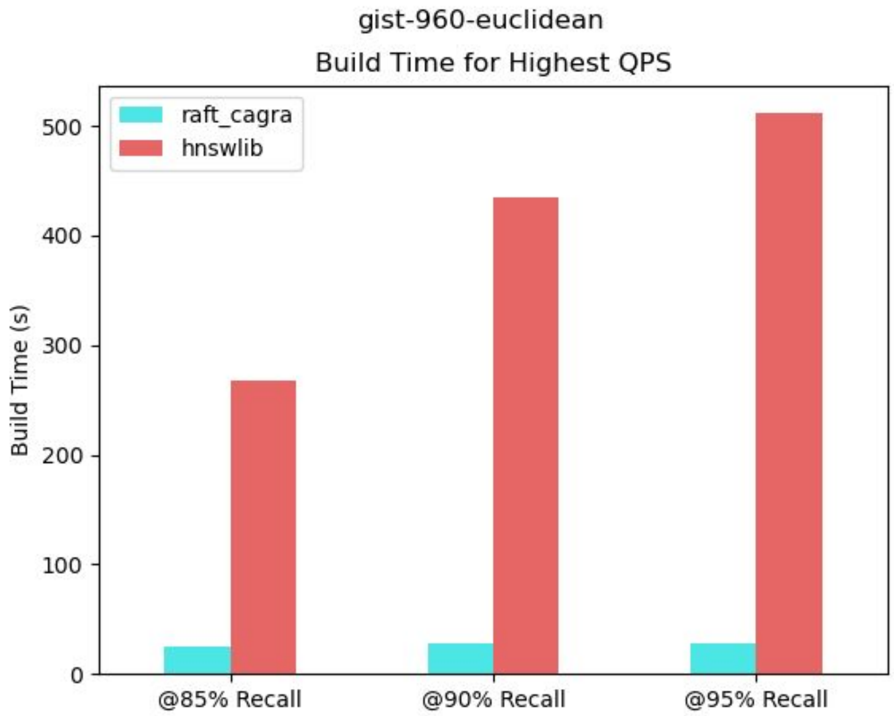
6x-10x Lower latency



Gist | index build | price-perf

GPU (L4): $0.01
CPU (15-core): $0.16
CPU (30-core): $0.32

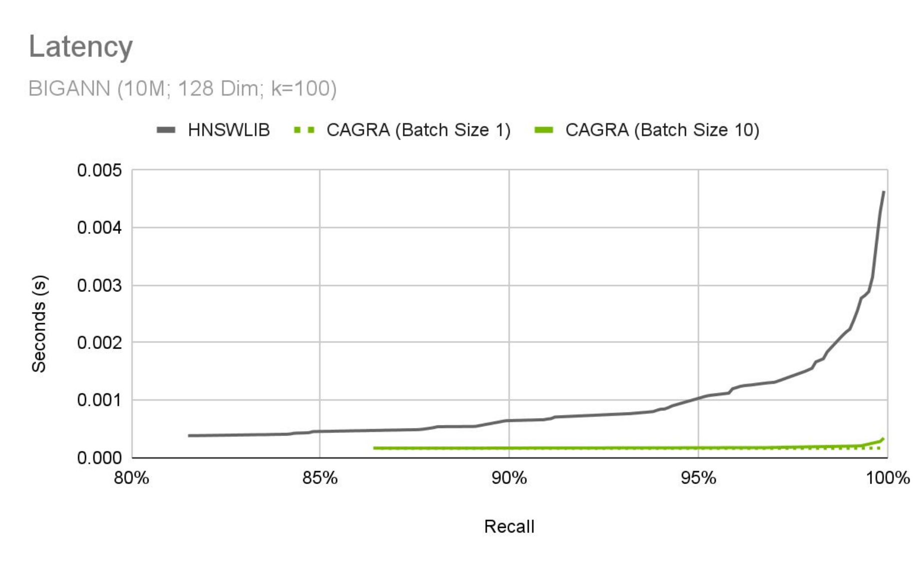
Note: All available threads were used to build HNSW index but build times were about the same on 15-core and 30-core.

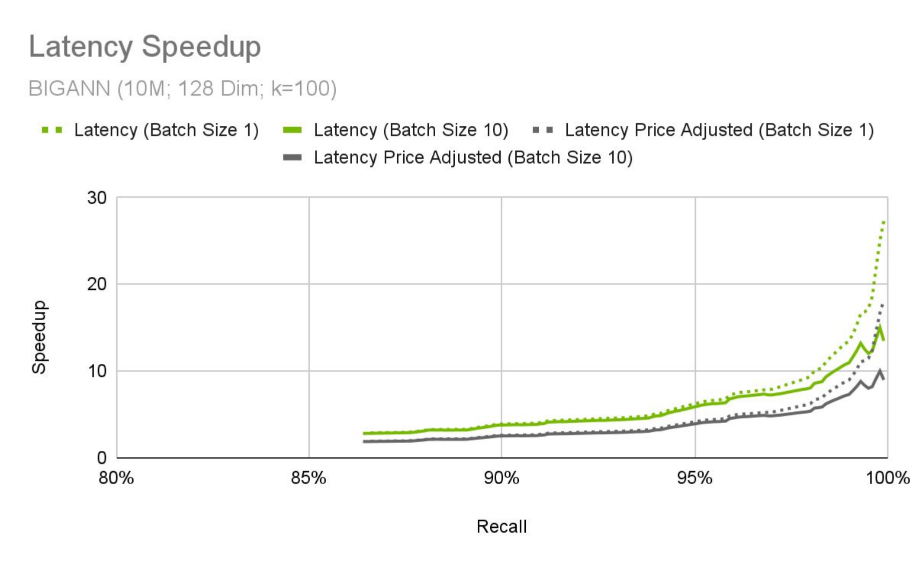


Same performance for 16x-32x less

Latency Price Performance

Batch size 1 and 10 (BIGANN-10M, 128 dimension)



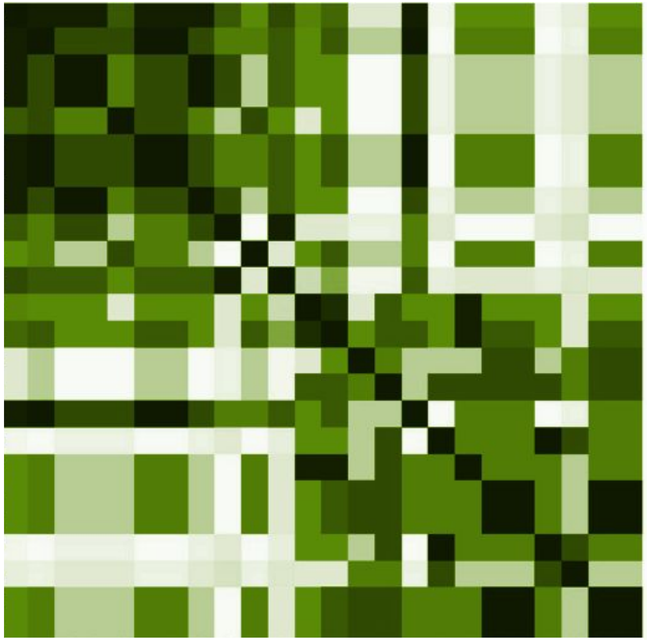


Notable Algorithms in RAFT

Pairwise distances

Every spatial library needs them!

- Flexible, ***composable building blocks*** that live at the heart of vector search.
- Uses CUTLASS GEMM for ***tensor cores***
- Element-wise epilogue operations (such as norm-based expansion functions) ***fused*** with GEMM.
- Kernel gramm API for constructing ***reproducing kernels*** (useful for kernel methods like Kernel PCA, Kernel density and SVM)



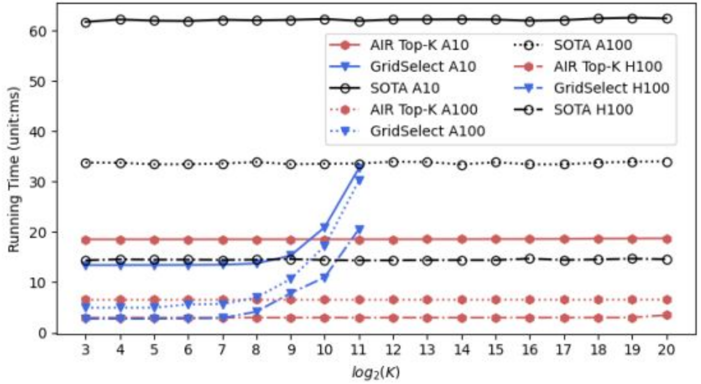
K-Selection

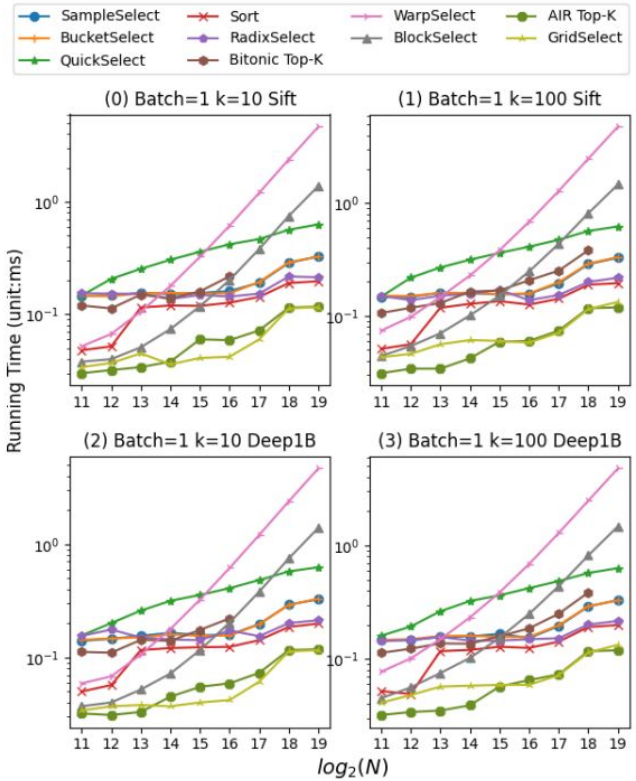
AirTopK: Adaptive and Iteration-fused Radix Top-K

- Minimizes CPU-GPU communication and device data access

GridSelect: Improved WarpSelect (from FAISS)

- Shared queue and parallel two-step insertion to decrease the frequency of costly operations





"Parallel Top-K Algorithms on GPU: A Comprehensive Study and New Methods", Zhang et al., SC23

Fusing Distances and K-Selection

- Special optimizations when k < 64
- Compute distance and k-selection in ***single "fused" kernel*** to eliminate additional memory transfers.
- K-selection done in ***registers*** for 1-NN and shared memory for k-NN.
- Important computation in some ***clustering algorithms***, (e.g. k-means and single-linkage clustering).

Fused k-NN Primitive

Index Rows	Query Rows	GPU-FAISS	cuSLINK
100K	100K	261ms	**143ms**
200K	200K	783ms	**537ms**
400K	400K	2706ms	**2017ms**
1M	1M	1.607s	**1.218s**

Fused 1-NN Primitive

Index Rows	Query Rows	Cols	GPU-FAISS	cuSLINK
100K	100	128	98.4ms	**0.55ms**
100K	100	256	95.6ms	**0.967ms**
100K	1k	64	96.6ms	**1.85ms**
100K	1K	128	98.9ms	**3.39ms**
100K	1K	256	104ms	**6.46ms**
100K	10K	64	126ms	**17ms**
100K	10K	128	146ms	**32ms**
100K	10K	256	156ms	**62.2ms**

"cuSLINK: Single-linkage Agglomerative Clustering on the GPU", Nolet et al., ECML-PKDD23

Semirings, Distances, and Sparse k-NN

	Distance	MovieLens Baseline	MovieLens RAFT	scRNA Baseline	scRNA RAFT	NY Times Bag of Words Baseline	NY Times Bag of Words RAFT	SEC Edgar Baseline	SEC Edgar RAFT
Dot Product Based	*Correlation*	130.57	**111.20**	**207.00**	235.00	**257.36**	337.11	134.79	**87.99**
	Cosine	131.39	**110.01**	**206.00**	233.00	**257.73**	334.86	127.63	**87.96**
	Dice	130.52	**110.94**	**206.00**	233.00	**130.35**	335.49	134.36	**88.19**
	Euclidean	131.93	**111.38**	**206.00**	233.00	**258.38**	336.63	134.75	**87.77**
	Hellinger	129.79	**110.82**	**205.00**	232.00	**258.22**	334.80	134.11	**87.83**
	Jaccard	130.51	**110.67**	**206.00**	233.00	**258.24**	336.01	134.55	**87.73**
	Russel-Rao	130.35	**109.68**	**206.00**	232.00	**257.58**	332.93	134.31	**87.94**
Non-Trivial Metrics	Canberra	3014.34	**268.11**	4027.00	**598.00**	4164.98	**819.80**	505.71	**102.79**
	Chebyshev	1621.00	**336.05**	3907.00	**546.00**	2709.30	**1072.35**	253.00	**146.41**
	Hamming	1635.30	**229.59**	3902.00	**481.00**	2724.86	**728.05**	258.27	**97.65**
	Jensen-Shannon	7187.27	**415.12**	4257.00	**1052.00**	10869.32	**1331.37**	1248.83	**142.96**
	KL Divergence	5013.65	**170.06**	4117.00	**409.00**	7099.08	**525.32**	753.56	**87.72**
	Manhattan	1632.05	**227.98**	3904.00	**477.00**	2699.91	**715.78**	254.69	**98.05**
	Minkowski	1632.05	**367.17**	4051.00	**838.00**	5855.79	**1161.31**	646.71	**129.47**

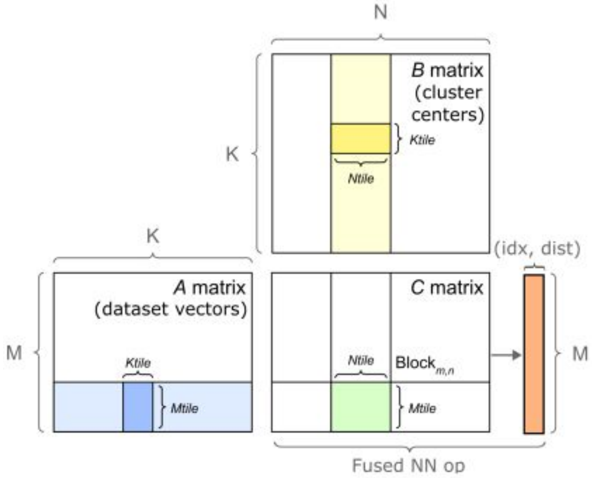
- Uses the framework of ***algebraic semirings*** popular in graph analytics
- Novel and state-of-the-art SpMV (sparse matrix-vector) for computing ***pairwise distance*** and tiled k-NN
- Uses same ***k-selection routines*** from dense brute-force kNN

Distance	Formula	NAMM	Norm	Expansion
Correlation	$1 - \frac{\sum_{i=0}^{k}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=0}^{k} x_i-\bar{x}^2}^2 \sqrt{\sum_{i=0}^{2} y_i-\bar{y}^2}^2}$		L_1,L_2	$1 - \frac{k\langle x\cdot y\rangle - \|x\|\|y\|}{\sqrt{(k\|x\|_2-\|x\|^2)(k\|y\|_2-\|y\|^2)}}$
Cosine	$\frac{\sum_{i=0}^{k} x_i y_i}{\sqrt{\sum_{i=0}^{k} x_i^2}\sqrt{\sum_{i=0}^{k} y_i^2}}$		L_2	$1 - \frac{\langle x\cdot y\rangle}{\|x\|_2^2\|y\|_2^2}$
Dice-Sorensen	$\frac{2\lvert\sum_{i=0}^{k} x_i y_i\rvert}{(\sum_{i=0}^{k} x)^2+(\sum_{i=0}^{k} y)^2}$		L_0	$\frac{2\langle x\cdot y\rangle}{\lvert x\rvert^2+\lvert y\rvert^2}$
Dot Product	$\sum_{i=0}^{k} x_i y_i$			$\langle x \cdot y\rangle$
Euclidean	$\sqrt{\sum_{i=0}^{k} \lvert x_i - y_i\rvert^2}$		L_2	$\|x\|_2^2 - 2\langle x \cdot y\rangle + \|y\|_2^2$
Canberra	$\sum_{i=0}^{k} \frac{\lvert x_i-y_i\rvert}{\lvert x_i\rvert+\lvert y_i\rvert}$	$\{\frac{\lvert x-y\rvert}{\lvert x\rvert+\lvert y\rvert}, 0\}$		
Chebyshev	$\sum_{i=0}^{k} \max(x_i - y_i)$	$\{\max(x-y), 0\}$		
Hamming	$\frac{\sum_{i=0}^{k} x_i \neq y_i}{k}$	$\{x \neq y, 0\}$		
Hellinger	$\frac{1}{\sqrt{2}}\sqrt{\sum_{i=0}^{k}(\sqrt{x_i}-\sqrt{y_i})^2}$			$1 - \sqrt{\langle\sqrt{x}\cdot\sqrt{y}\rangle}$
Jaccard	$\frac{\sum_{i=0}^{k} x_i y_i}{(\sum_{i=0}^{k} x_i^2+\sum_{i=0}^{k} y_i^2-\sum_{i=0}^{k} x_i y_i}$		L_0	$1 - \frac{\langle x\cdot y\rangle}{(\|x\|+\|y\|-\langle x\cdot y\rangle)}$
Jensen-Shannon	$\sqrt{\frac{\sum_{i=0}^{k} x_i \log\frac{x_i}{\mu_i}+y_i\log\frac{y_i}{\mu_i}}{2}}$	$\{x\log\frac{x}{\mu}+y\log\frac{y}{\mu}, 0\}$		
KL-Divergence	$\sum_{i=0}^{k} x_i \log(\frac{x_i}{y_i})$			$\langle x \cdot \log\frac{x}{y}\rangle$
Manhattan	$\sum_{i=0}^{k} \lvert x_i - y_i\rvert$	$\{\lvert x-y\rvert, 0\}$		
Minkowski	$(\sum_{i=0}^{k} \lvert x_i - y_i\rvert^p)^{1/p}$	$\{\lvert x-y\rvert^p, 0\}$		
Russel-Rao	$\frac{k-\sum_{i=0}^{2} x_i y_i}{k}$			$\frac{k-\langle x\cdot y\rangle}{k}$

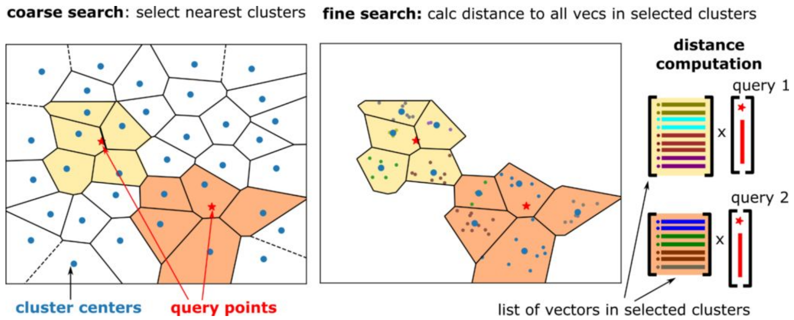
"GPU Semiring Primitives for Sparse Neighborhood Methods", Nolet et al., MLSys22

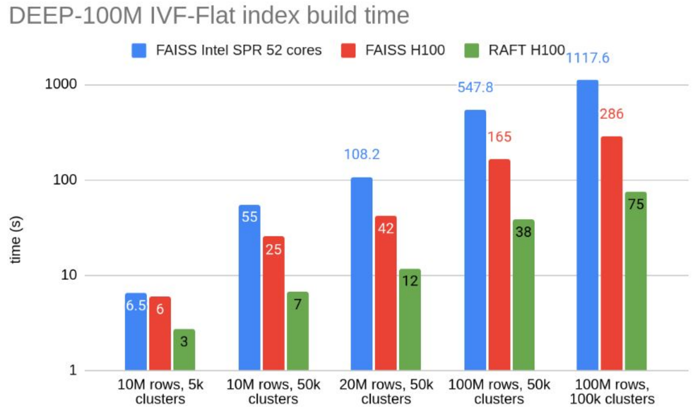
Balanced / Hierarchical K-means

- Uses Fused 1-NN Primitive to compute closest centroids
- Vectors more ***uniformly distributed*** across clusters
- Utilizes ***tensor cores***

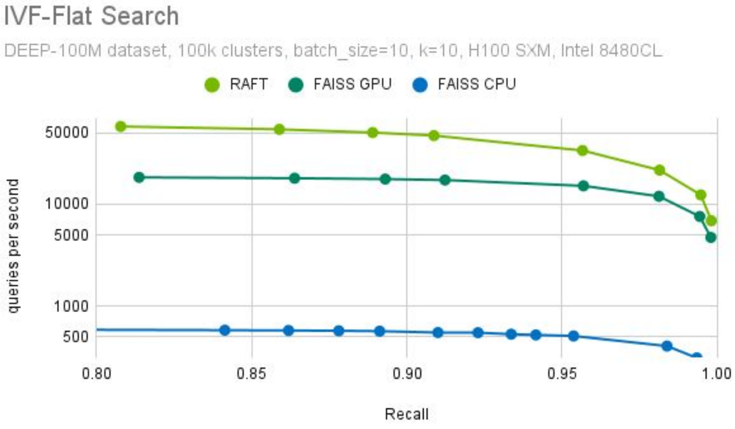


IVF-Flat





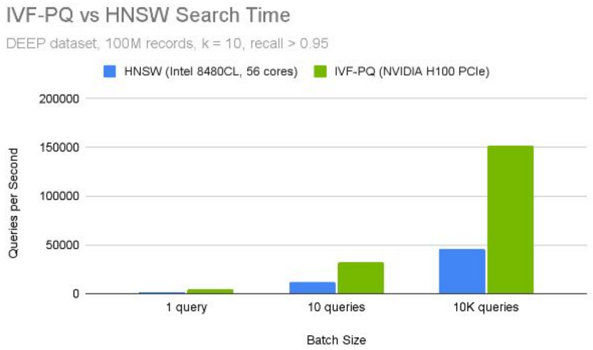
- Uses balanced k-means implementation
- Balanced clustering uses tensor cores to speed up computation
- Vectorized interleaved layout ***improves memory reads***
- Support for ***8-bit datatypes*** (uint8 and int8)
- Supports custom predicate ***pre-filter***
- Improved performance over FAISS GPU for ***small batch sizes***



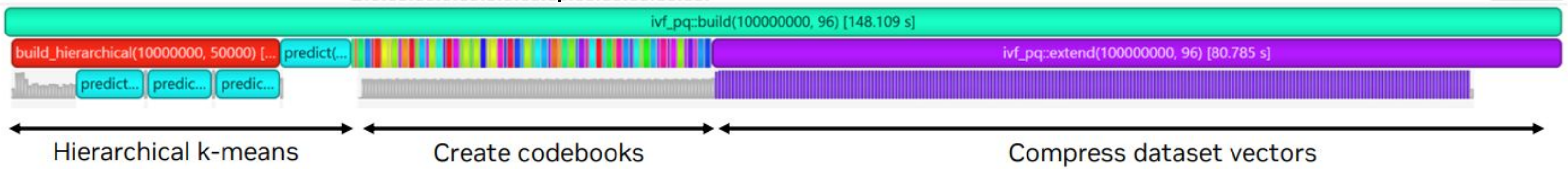
NVIDIA

IVF-PQ

- Lower PQ bits (4-8) provide ***better compression*** and more ***efficient use of shared memory***
- Configurable lookup table and distance precision provide ***faster computation*** and ***efficient use of shared memory***
- Support for ***reduced precision*** (uint8 and int8)
- Supports custom predicate ***pre-filter***



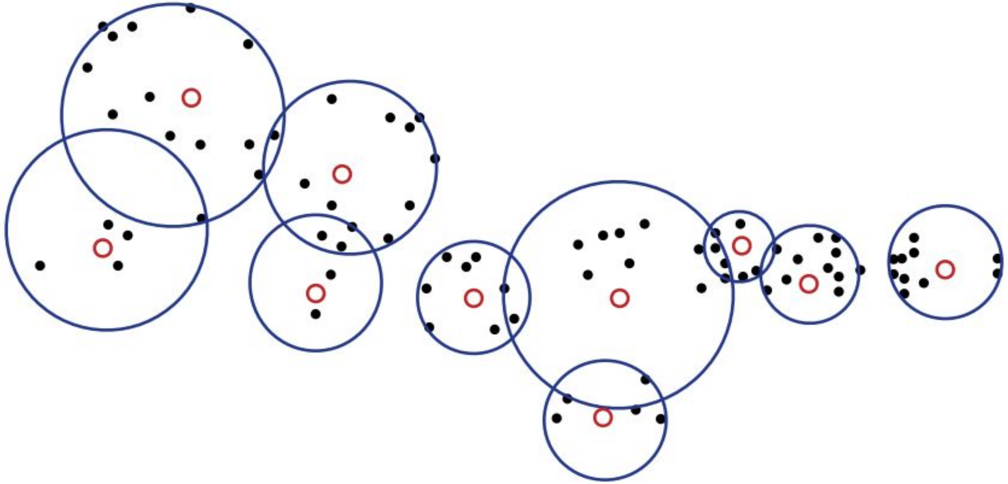
Deep-100M w/ K-means trained on 10%



Random Ball Cover

Reduces to an ***inverted file index*** where the number of probes are computed

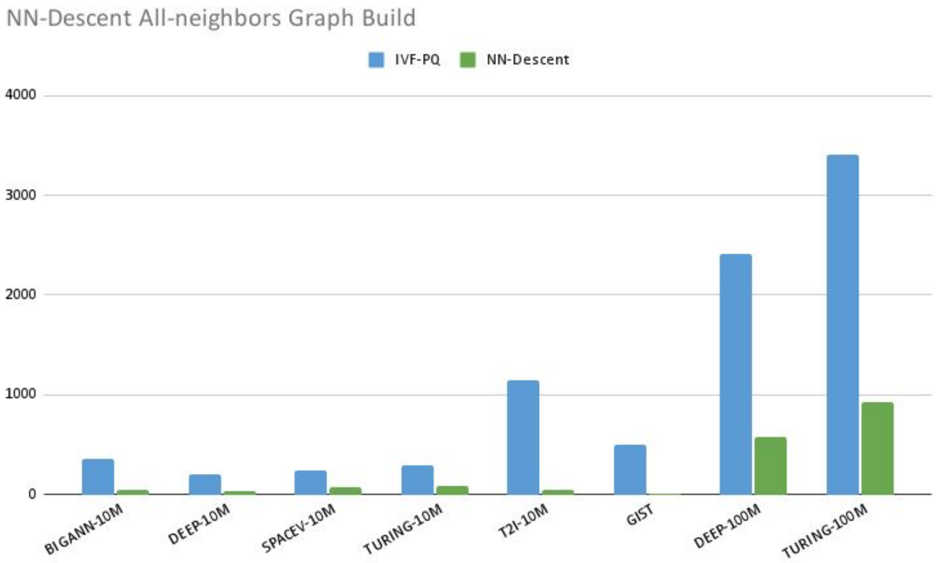
- Choose centroids uniformly at random and find closest index points to each (1-nn)
- Use ***triangle inequality*** during search to compute probes for each query point
- Use ***IVF-flat*** algorithm to search closest probes. Can be both exact and approximate
- Can be used for ***k-NN and eps-NN***



"[Accelerating Nearest Neighbors Search on Manycore Systems]", Lawrence Cayton, 2011

Nearest Neighbors Descent

- Useful for ***accelerated all-neighbors*** graph construction
- Currently used to build ***CAGRA graph***
- Utilizes ***tensor cores***, resulting in speedup from original paper
- Graph sampling and updating are offloaded to CPU, ***reducing GPU memory*** usage

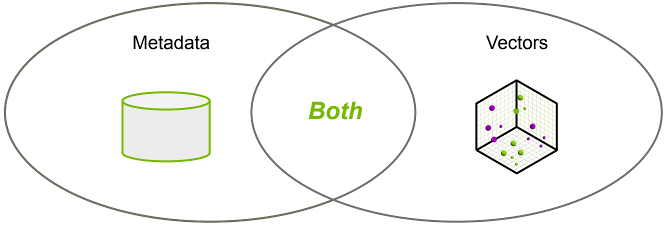


"Fast k-NN Graph Construction by GPU-based NN Descent", Wang et al., CIKM21

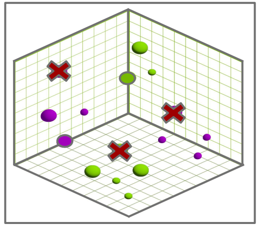
Pre-filtering

Improved pre-filtering unlocks advanced search capabilities

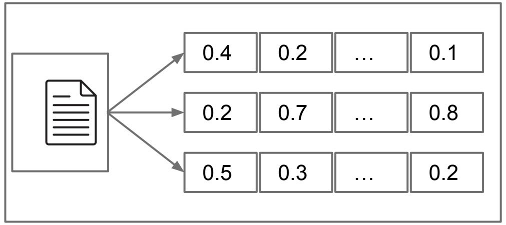
Hybrid Search



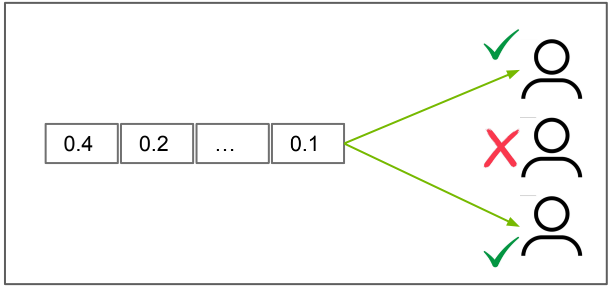
Vector Removal



Multi-valued Keys



Access Controls



- Accepts ***predicate function*** to filter vectors during search
- Filtering primitives ***optimized for GPU*** (eg. bitset, bitmask, hash table, bloom filter)

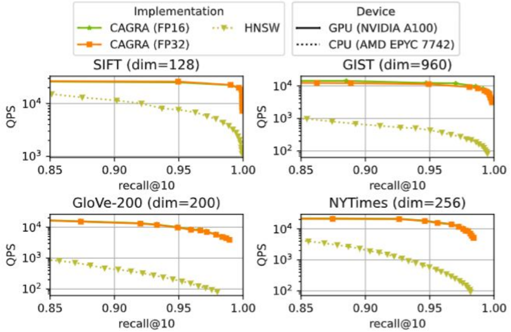
CAGRA

CAGRA

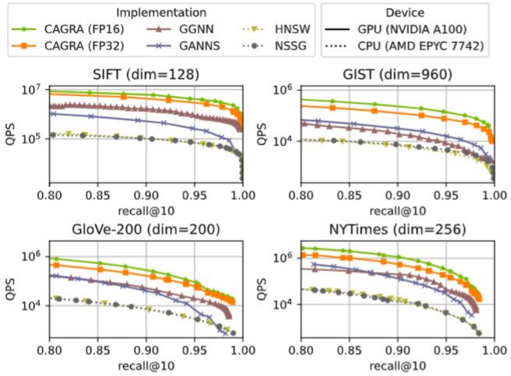
GPU-Accelerated State-of-the-Art Graph-Based ANN

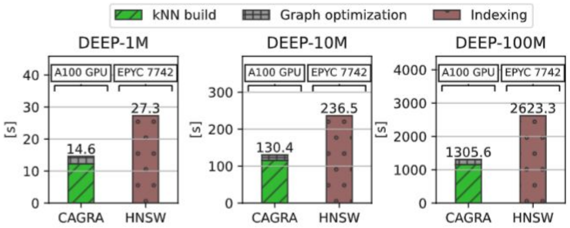
- ***Individual queries*** parallelized during search
- Setting records for both ***single query*** and ***large batch*** performance
- ***Higher throughput*** than existing GPU Graph ANNs and ***lower latency*** than SOTA CPU Graph ANNs

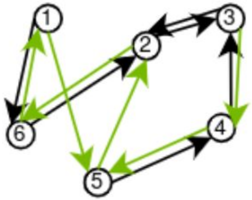
Single query at a time



Batches of 10k queries







CAGRA graph

“[CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search on the GPU]”, Ootomo et al., 2023

CAGRA

GPU-Accelerated State-of-the-Art Graph-Based ANN

- Step 1: ***Build*** initial k-NN Graph
    - Use fast ANN method like NN-Descent (or IVF-PQ)

- Step 2: ***Optimize*** k-NN Graph
    - Reduce degree of the k-nn graph (reducing size) while enhancing reachability
    - Enhance reachability
        - Use strongly connected components
            - smaller value enhances reachability
        - Average 2-hop node count (number of nodes that can be reached in 2 hops)
            - larger value improves exploration

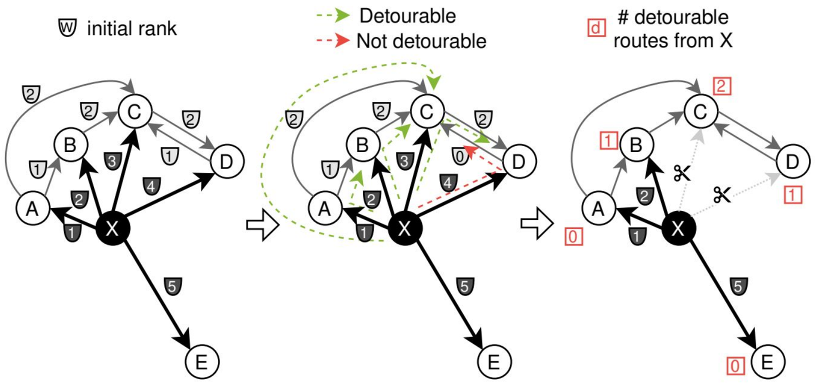
"CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search on the GPU", Ootomo et al., 2023

CAGRA

GPU-Accelerated State-of-the-Art Graph-Based ANN

Graph Optimization

- ***Reorder*** edges by rank and ***prune***
  - increase diversity
- ***Reverse*** edge addition
  - improve reachability and reduce strong connected components



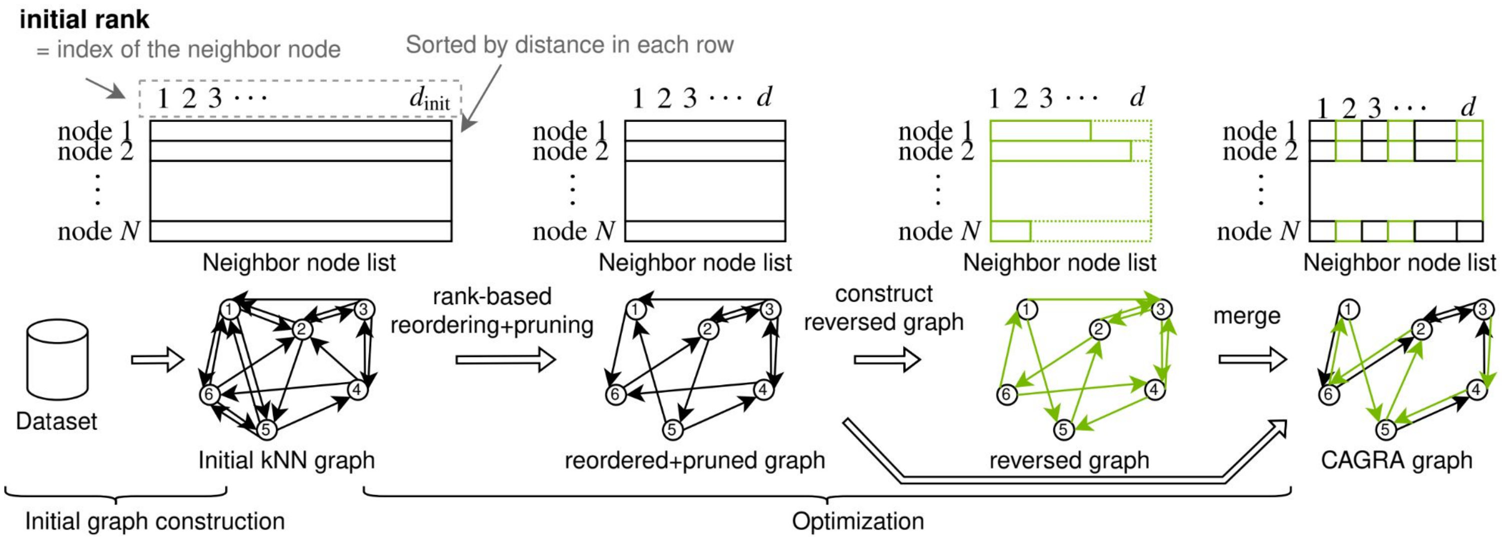
Detourable routes classified according to:

$$(e_{X\to Z}, e_{Z\to Y}) \text{ s.t. } \max(w_{X\to Z}, w_{Z\to Y}) < w_{X\to Y}$$

"CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search on the GPU", Ootomo et al., 2023

CAGRA

GPU-Accelerated State-of-the-Art Graph-Based ANN



"CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search on the GPU", Ootomo et al., 2023

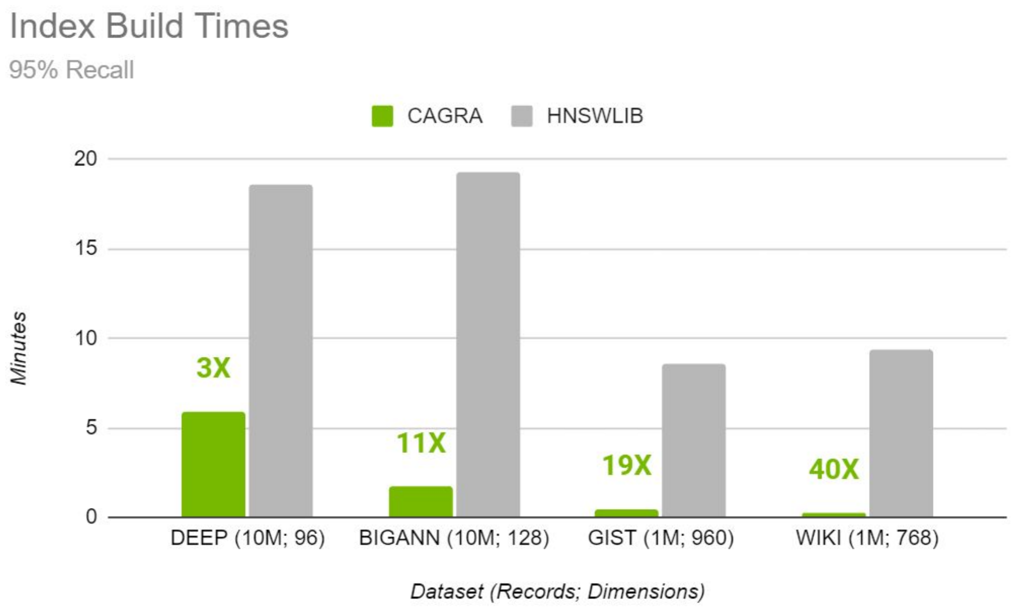
CAGRA

GPU-Accelerated State-of-the-Art Graph-Based ANN

Build speedup scales with

1. Number of dimensions
2. Number of vectors
3. Recall level

Build times based on nn-descent strategy

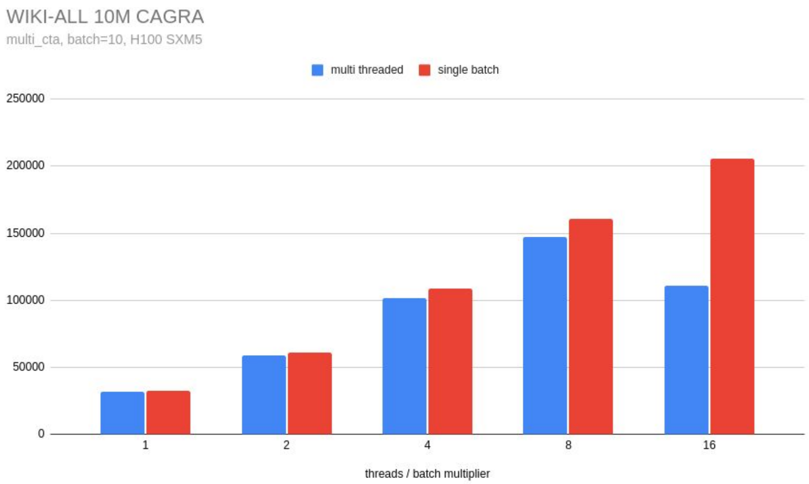


"CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search on the GPU", Ootomo et al., 2023

CAGRA

GPU Scaling in throughput mode

- Throughput mode improves GPU utilization for small batches
- Performance of submitting all queries in a single batch stays similar to using 8x threads / cuda streams.
- Throughput shrinks almost 2x with 16x threads / cuda streams.



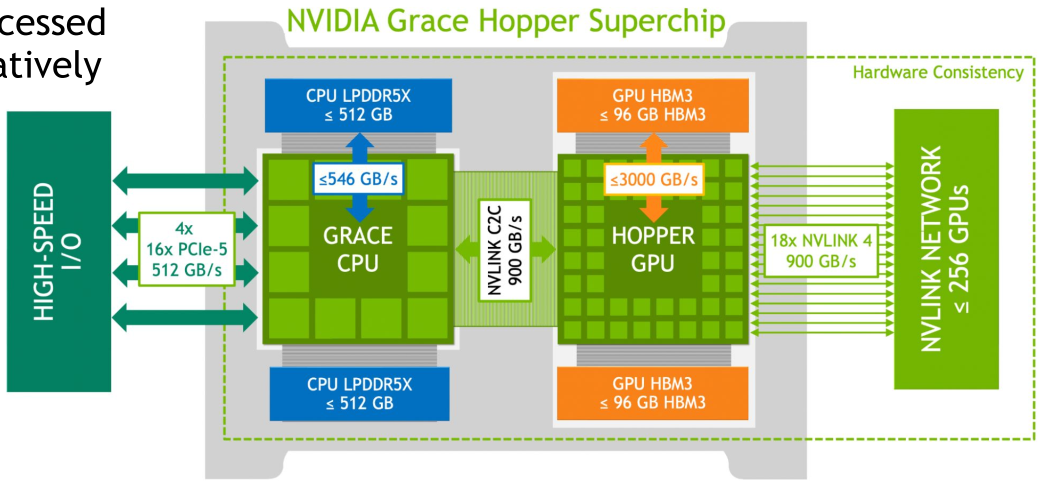
"CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search on the GPU", Ootomo et al., 2023

NVIDIA

Vector Search with Grace Hopper

Optimal performance for huge indexes

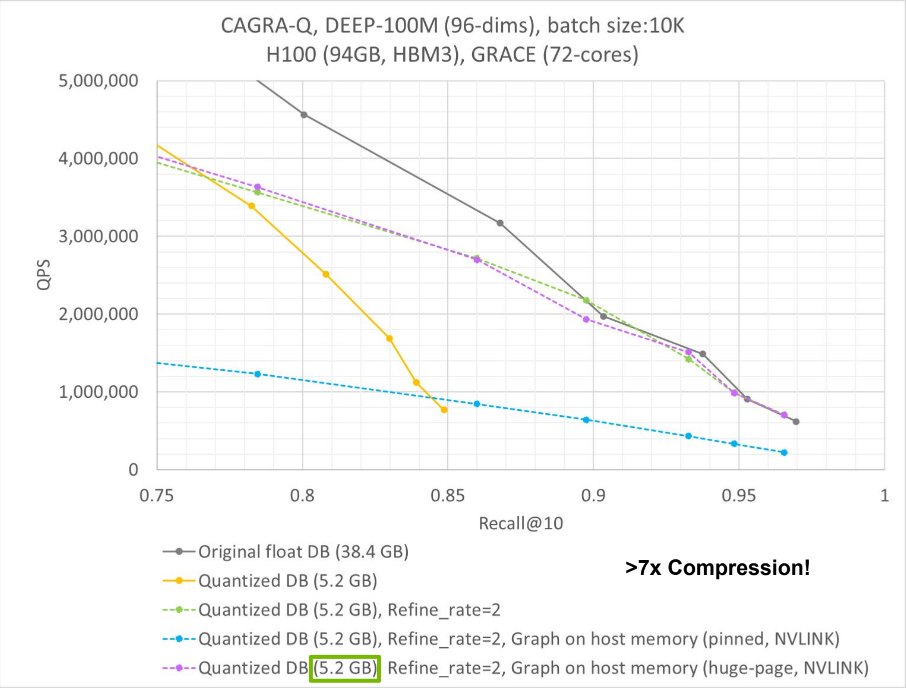
- High-speed (900 GB/s) C2C memory link allows "spilling" of large indexes from device to host memory
- 512GB of host memory allows storage of huge indexes in memory with fast retrieval
- Upcoming optimizations will keep most-accessed index memory on device but still offer relatively fast access to entire index through C2C

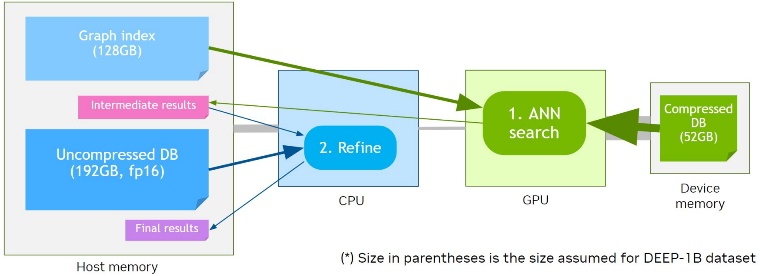


CAGRA-Q

CAGRA + Quantization for improved scale

- CAGRA requires ***original training vectors*** to compute distances
- Can keep original dataset in ***host memory*** (this can be slow)
- CAGRA-Q ***compresses original dataset*** so it can be stored on device for faster search
- Original dataset kept in host memory and used ***only for reranking*** to improve recall





- CAGRA-Q makes a great companion for ***Grace Hopper*** and improved chip-to-chip (C2C) bandwidth.
- **TLDR;** Compressed dataset on device and graph stored in huge page pinned memory has ***equivalent performance*** to original dataset and graph stored on device at high recall levels.

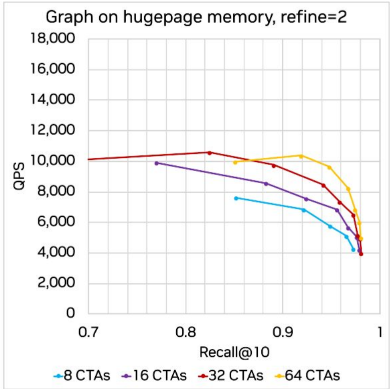
CAGRA-Q

CAGRA + Quantization for improved scale

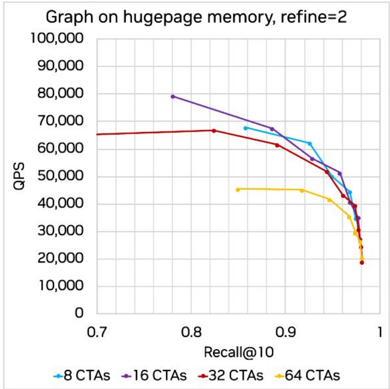
- CAGRA-Q ***compresses original dataset*** so it can be stored on device for faster search
- Original dataset kept in host memory and used ***only for reranking*** to improve recall

Wiki-all-88M (251GB), Compressed: 17GB, Graph: 11GB

Batch size 1

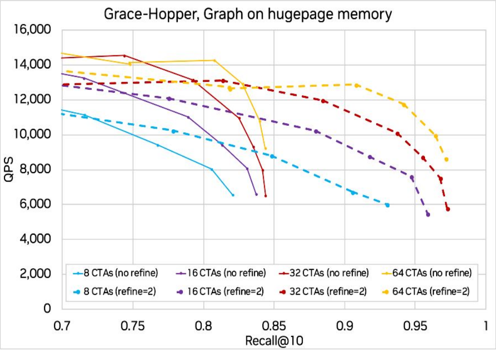


Batch size 10

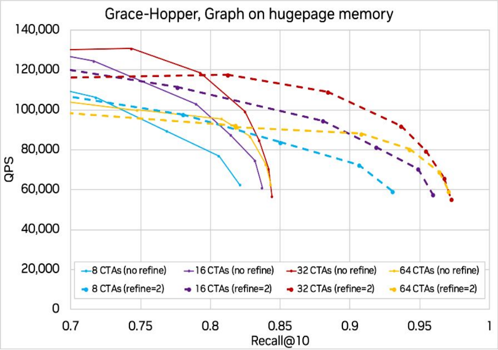


Deep-1B (384GB), Compressed: 52GB, Graph: 128GB

Batch size 1



Batch size 10

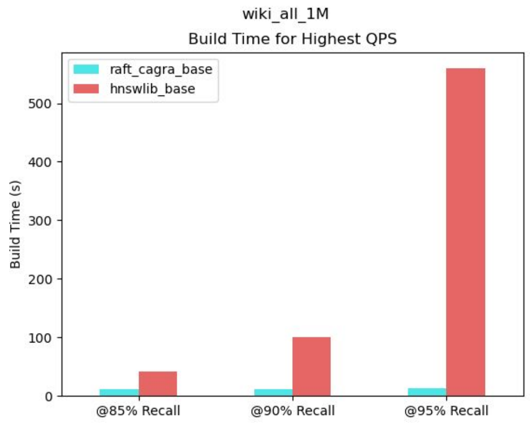


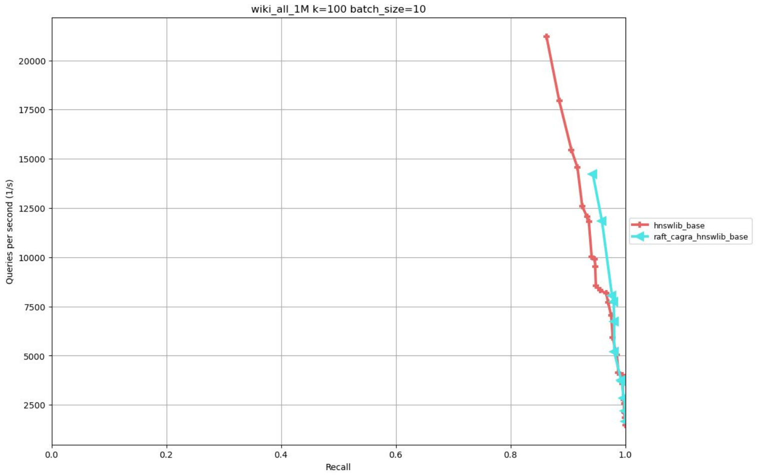
- CAGRA-Q makes a great companion for ***Grace Hopper*** and improved chip-to-chip (C2C) bandwidth.
- **TLDR;** Compressed dataset on device and graph stored in huge page pinned memory has ***equivalent performance*** to original dataset and graph stored on device at high recall levels.

CAGRA+HNSW

Building index on GPU and searching on CPU

- ***Training and updating*** indexes faster on the GPU
- Some organizations have pre-existing ***CPU infrastructure*** dedicated to search



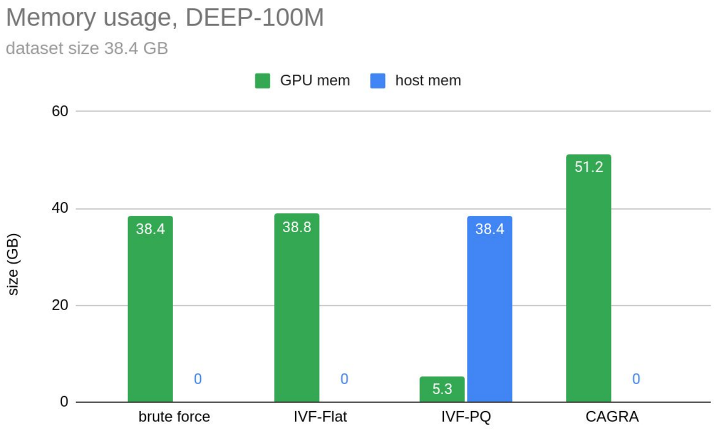


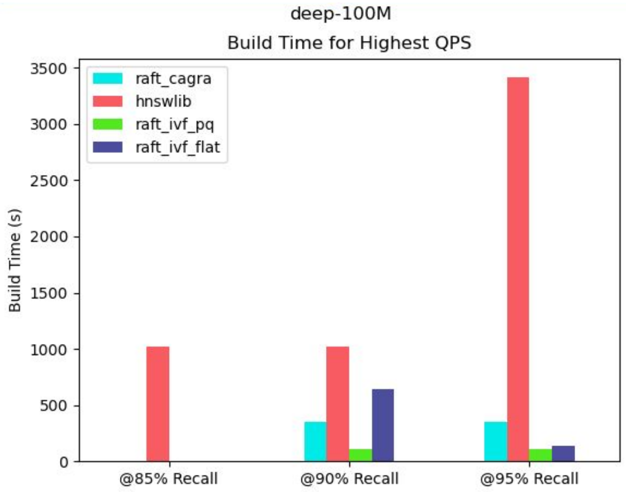
- We can ***search CAGRA graph on CPU*** using HNSW
- Tests are demonstrating ***comparable performance*** (sometimes better) even when CAGRA is used only as the base graph
- This capability is ***available to test*** in RAFT ANN Benchmarks and will soon have a first-class API

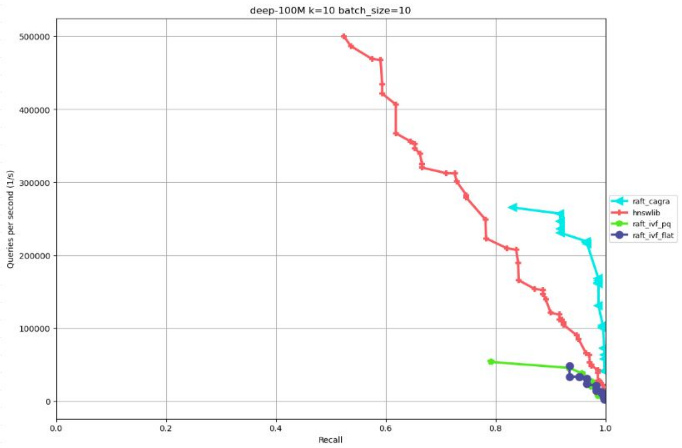
"Graph-based Nearest Neighbor Search: From Practice to Theory", Prokhorenkova et al., ICML '20

Scaling to 100M

Comparing trade-offs at scale for 95% recall







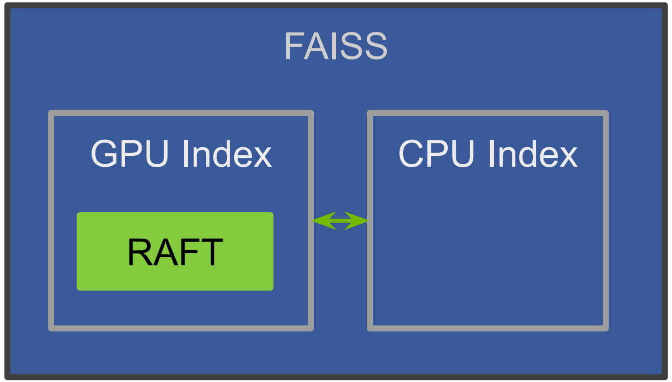
Integrations

FAISS is a great way to get RAFT

RAFT will become a GPU backend for FAISS

Benefits of using FAISS as a library for vector search

- It's *easy to integrate*
- It supports *CPU* and *GPU* interoperability
- It provides *multi-GPU* for improved scale and throughput
- Its APIs have become a *standard*



A new GPU backend for FAISS

Modernizing existing GPU capabilities

- Working to make RAFT ***the default*** back-end for FAISS on the GPU
- RAFT will ***continue to improve*** GPU performance and features, even as new hardware architectures and CUDA versions are released
- When building FAISS from source, RAFT can be enabled using a ***compile-time option***
- Will soon have a ***faiss-gpu-raft*** Conda package

FAISS Integration Timeline

- v23.08 ✓ Brute Force / Flat
- v23.10 ✓ IVF-flat
- v23.12 IVF-PQ
- v24.02 CAGRA

Initial GPU Acceleration Partners

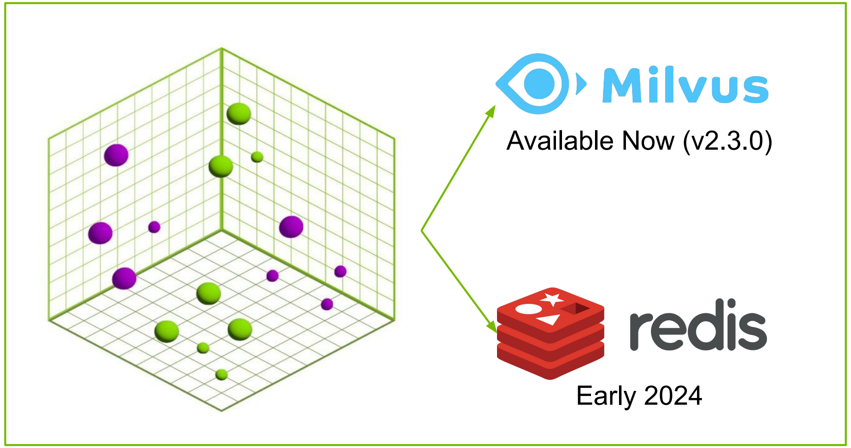
RAFT is empowering the ecosystem

- *Milus* already integrated RAFT in v2.3.0. Expecting updated version of RAFT in the next release and going forward.

- *Redis* will have RAFT integrated by end of year, with an enterprise offering in 2024.

- Five other *independent software vendors* in the process of integrating RAFT.

- All of the *cloud service providers* are in the process of evaluating RAFT. We are assisting with price performance estimations.



Release Roadmap

RAFT Vector Search Roadmap

Key initiatives

Features	Description	Version
CAGRA and HNSW interoperability	Train an index on GPU and deploy it to CPU.	24.04
CAGRA reduced precision support	Improves scale and performance on a single GPU.	24.02
Multi-GPU ANN index API	Train and search an index across multiple GPUs in a single node.	24.04
C API	Enable third party adoption (in addition to the C++ and Python APIs)	24.02 24.04 24.06
Multi-valued keys	Support multi-valued keys	24.04
Dynamic batching	Dispatches queries within a given latency budget.	24.06

Resources

A Variety of Ways to Get Up & Running

More about RAPIDS RAFT

- RAPIDS GTC Talk
- RAFT IVF-PQ GTC Talk
- RAFT CAGRA arXiv
- NVIDIA Tech Blog

Discussion & Support

- Check the RAPIDS RAFT GitHub
- C++ API documentation
- Python API documentation
- Talk to NVIDIA Services

@RAPIDSai

https://github.com/rapidsai/raft

https://rapids.ai/slack-invite/

RAPIDS

https://rapids.ai