



Accelerating Vector Search with RAPIDS cuVS

Nathan Stephens, Product Manager, NVIDIA

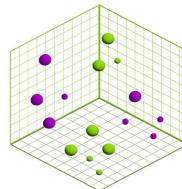


<https://github.com/nwstephens/ASU-GPU-Day-2024>

Contents

Vector Search

Overview



Embeddings

Vector Search

Applications

Algorithms

IVF-Flat

IVF-PQ

CAGRA

RAPIDS

Notebooks



TL/DR

Why GPU accelerated vector search matters

1. Vector Search is commonly used with AI

Examples: Retrieval augmented generation (RAG), recommenders, genomics, finance, chemistry, finance, and cybersecurity.

2. Vector data types are everywhere

Nearly all data sources will support them if they don't already.

3. GPUs increase throughput, lower latency, and decrease build times

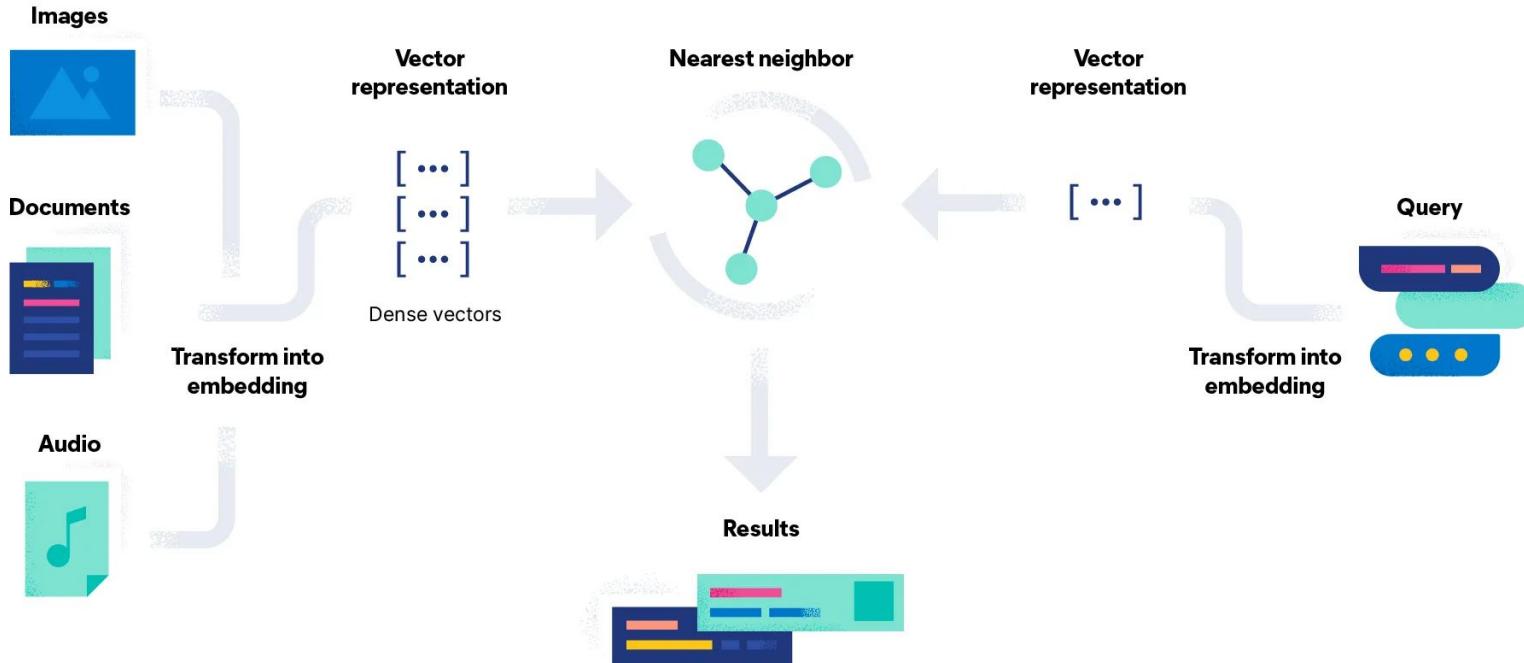
GPUs can increase performance by +10X and cut index build times from days to minutes.

Vector Search Overview



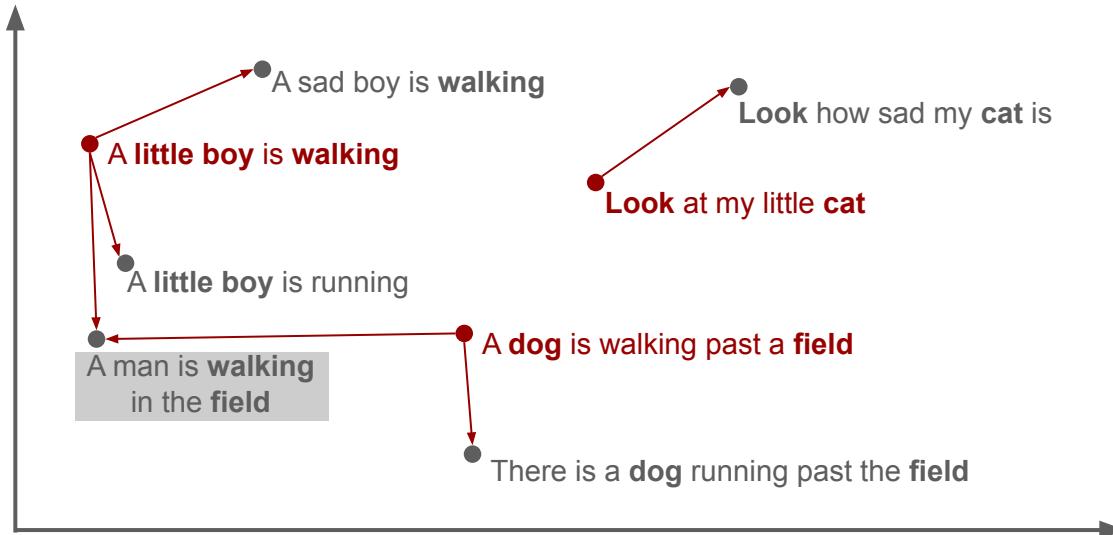
Vector Search Overview

Nearest Neighbor Analysis



Vector Search Overview

Example of Text Vectors



Vector Search Overview

What is an embedding?

Embedding: Numerical representation of typically non-numerical data objects

Text

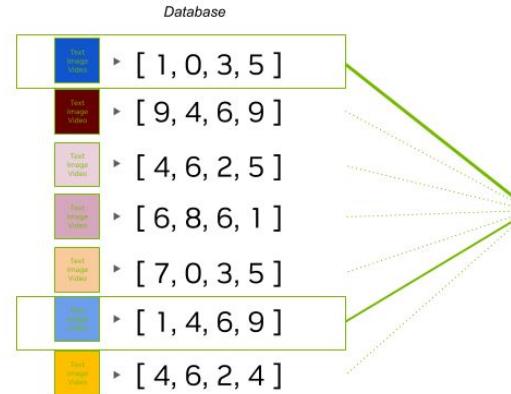
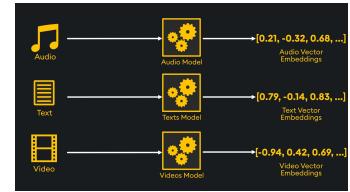
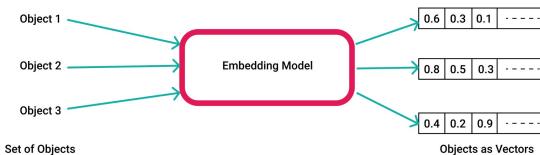
- TF-IDF (Term Frequency — Inverse Document Frequency)
- Word2Vec
- BERT (Bidirectional Encoder Representations from Transformers)

Audio

- Spectrogram-based Representations
- MFCCs (Mel Frequency Cepstral Coefficients)
- Convolutional Recurrent Neural Networks (CRNNs)

Image

- Convolutional Neural Networks
- Transfer Learning with Pre-trained CNNs like ResNet and VGG
- Autoencoders



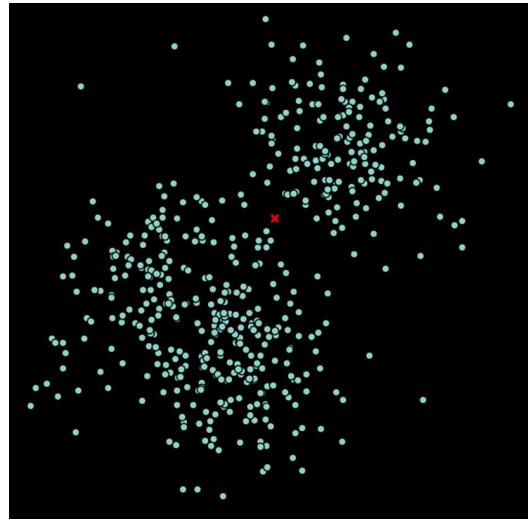
Request
[1, 3, 6, 8]

"Find the two most similar images, documents, or videos"

Vector Search Overview

Nearest Neighbor Search

Question: How to find the closest k points to the red marker?



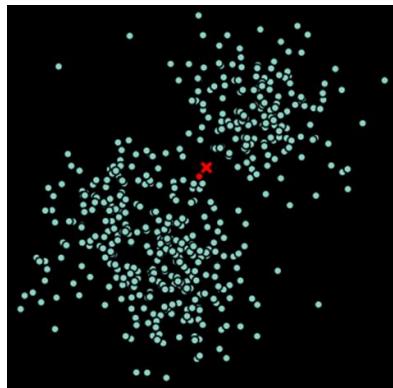
Vector Search Overview

Nearest Neighbor Search

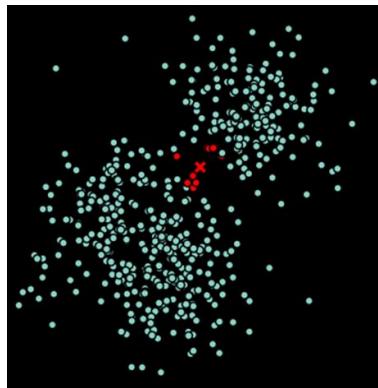
Question: How to find the closest k points to the red marker?

Answer: Measure the distance to all points and choose the shortest!

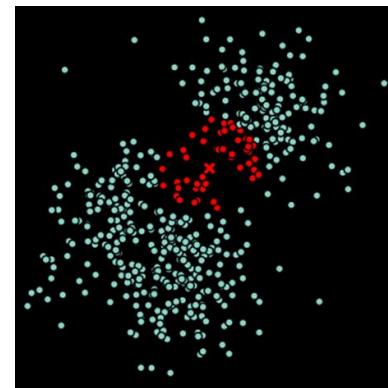
K=1



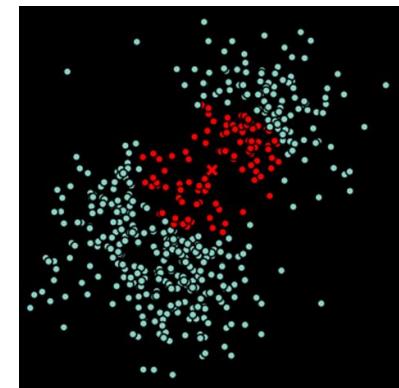
K=10



K=50

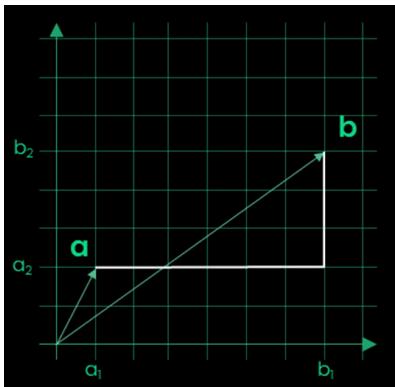


K=100

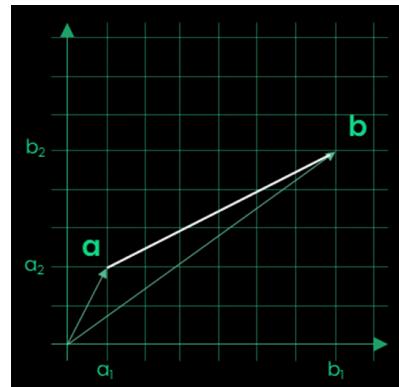


Vector Search Overview

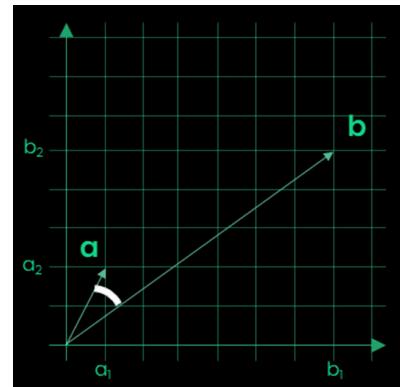
Example Distance Metrics



L1
(Taxi Distance)



L2 or Euclidean
Distance



Cosine
Similarity

Vector Search Overview

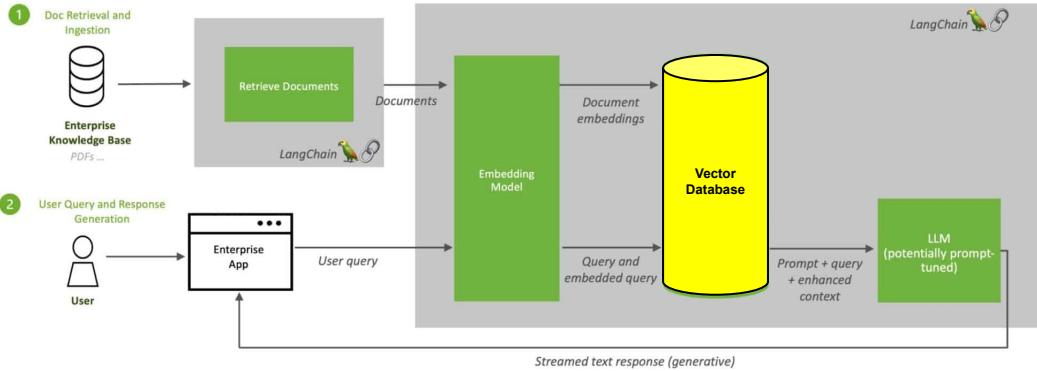
Applications

- **Recommender systems.** Provide personalized suggestions according to what a user has shown interest in or interacted with.
- **Finance.** Fraud detection models vectorize user transactions, making it possible to determine whether those transactions are similar to typical fraudulent activities.
- **Cybersecurity.** Uses embeddings to model and search behaviors of bad actors and anomalous activities.
- **Genomics.** Finds similar genes and cell structures in genomics analysis, such as single-cell RNA analysis.
- **Chemistry.** Models molecular descriptors or fingerprints of chemical structures to compare them or find similar structures in a database.

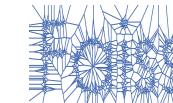
Vector Search Overview

Retrieval Augmented Generation (RAG)

RAG Workflow



Vector Search Technologies



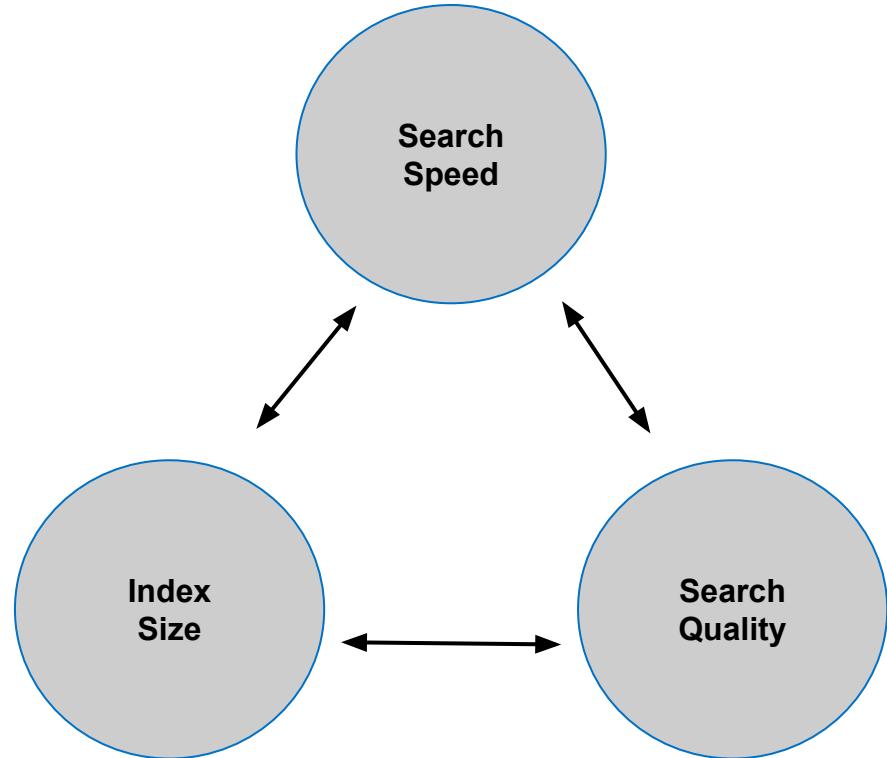
Vector Search Algorithms



Vector Search Challenges

Competing priorities cause computational challenges and tradeoffs

- Higher throughput requires larger batch sizes but latency requirements may require small batch sizes.
- Indexing is slow and choice of index affects search quality, size, and speed.
- Higher quality search requires larger storage and is more thorough, reducing speed.
- Higher dimensionality embeddings can improve quality at the expense of storage and speed.
- Larger datasets increase computational needs, reducing speeds and increasing storage size.



RAPIDS cuVS Overview

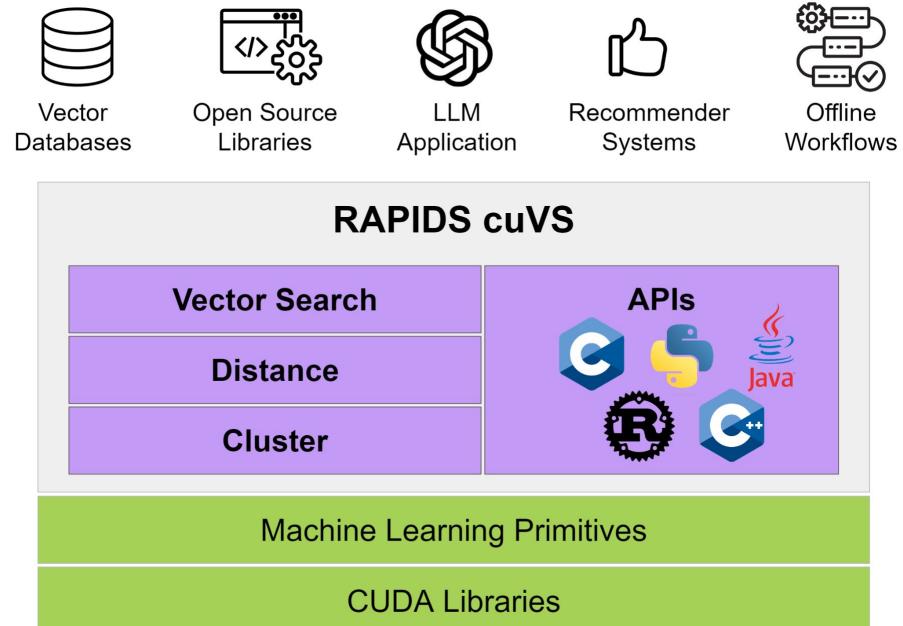
Open-Source GPU Accelerated Vector Search and Clustering

Vector search:

- Underpins many data mining and AI applications
- Requires efficient handling of massive workloads.

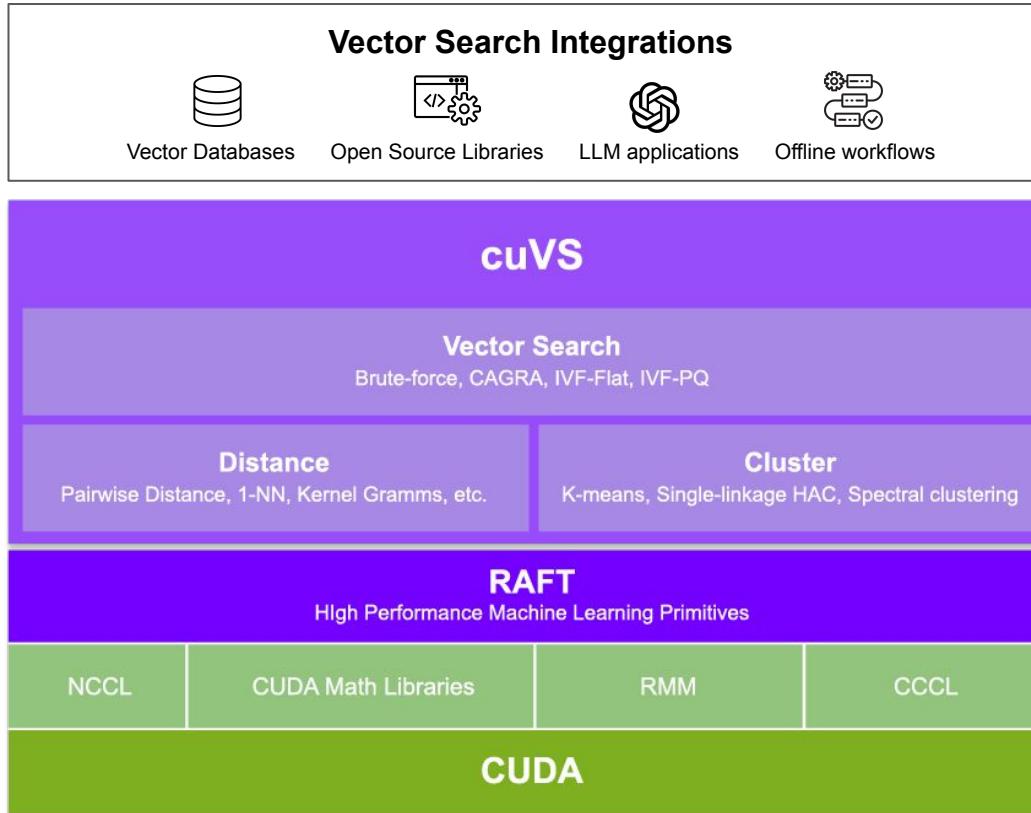
RAPIDS cuVS contains optimized algorithms for:

- Approximate nearest neighbors
- Clustering



RAFT to cuVS Migration

Winter 2024 - Summer of 2024



Approximate Nearest Neighbors

Building an Index

Brute force is sloooowwww for large data

- **Solution:** Build an index!
- **Examples:** Inverted file index (IVF) or graph based (HNSW)
- Gives **approximate** results (less than 100% recall)

1	1	5557	15117	14860	14493	14265	1487	1482	1479	1473	1472	1471	1470	1469	1468	1467	1466	1465	1464	1463	1462	1461	1460	1459	1458	1457	1456	1455	1454	1453	1452	1451	1450	1449	1448	1447	1446	1445	1444	1443	1442	1441	1440	1439	1438	1437	1436	1435	1434	1433	1432	1431	1430	1429	1428	1427	1426	1425	1424	1423	1422	1421	1420	1419	1418	1417	1416	1415	1414	1413	1412	1411	1410	1409	1408	1407	1406	1405	1404	1403	1402	1401	1400	1399	1398	1397	1396	1395	1394	1393	1392	1391	1390	1389	1388	1387	1386	1385	1384	1383	1382	1381	1380	1379	1378	1377	1376	1375	1374	1373	1372	1371	1370	1369	1368	1367	1366	1365	1364	1363	1362	1361	1360	1359	1358	1357	1356	1355	1354	1353	1352	1351	1350	1349	1348	1347	1346	1345	1344	1343	1342	1341	1340	1339	1338	1337	1336	1335	1334	1333	1332	1331	1330	1329	1328	1327	1326	1325	1324	1323	1322	1321	1320	1319	1318	1317	1316	1315	1314	1313	1312	1311	1310	1309	1308	1307	1306	1305	1304	1303	1302	1301	1300	1299	1298	1297	1296	1295	1294	1293	1292	1291	1290	1289	1288	1287	1286	1285	1284	1283	1282	1281	1280	1279	1278	1277	1276	1275	1274	1273	1272	1271	1270	1269	1268	1267	1266	1265	1264	1263	1262	1261	1260	1259	1258	1257	1256	1255	1254	1253	1252	1251	1250	1249	1248	1247	1246	1245	1244	1243	1242	1241	1240	1239	1238	1237	1236	1235	1234	1233	1232	1231	1230	1229	1228	1227	1226	1225	1224	1223	1222	1221	1220	1219	1218	1217	1216	1215	1214	1213	1212	1211	1210	1209	1208	1207	1206	1205	1204	1203	1202	1201	1200	1199	1198	1197	1196	1195	1194	1193	1192	1191	1190	1189	1188	1187	1186	1185	1184	1183	1182	1181	1180	1179	1178	1177	1176	1175	1174	1173	1172	1171	1170	1169	1168	1167	1166	1165	1164	1163	1162	1161	1160	1159	1158	1157	1156	1155	1154	1153	1152	1151	1150	1149	1148	1147	1146	1145	1144	1143	1142	1141	1140	1139	1138	1137	1136	1135	1134	1133	1132	1131	1130	1129	1128	1127	1126	1125	1124	1123	1122	1121	1120	1119	1118	1117	1116	1115	1114	1113	1112	1111	1110	1109	1108	1107	1106	1105	1104	1103	1102	1101	1100	1099	1098	1097	1096	1095	1094	1093	1092	1091	1090	1089	1088	1087	1086	1085	1084	1083	1082	1081	1080	1079	1078	1077	1076	1075	1074	1073	1072	1071	1070	1069	1068	1067	1066	1065	1064	1063	1062	1061	1060	1059	1058	1057	1056	1055	1054	1053	1052	1051	1050	1049	1048	1047	1046	1045	1044	1043	1042	1041	1040	1039	1038	1037	1036	1035	1034	1033	1032	1031	1030	1029	1028	1027	1026	1025	1024	1023	1022	1021	1020	1019	1018	1017	1016	1015	1014	1013	1012	1011	1010	1009	1008	1007	1006	1005	1004	1003	1002	1001	1000	999	998	997	996	995	994	993	992	991	990	989	988	987	986	985	984	983	982	981	980	979	978	977	976	975	974	973	972	971	970	969	968	967	966	965	964	963	962	961	960	959	958	957	956	955	954	953	952	951	950	949	948	947	946	945	944	943	942	941	940	939	938	937	936	935	934	933	932	931	930	929	928	927	926	925	924	923	922	921	920	919	918	917	916	915	914	913	912	911	910	909	908	907	906	905	904	903	902	901	900	899	898	897	896	895	894	893	892	891	890	889	888	887	886	885	884	883	882	881	880	879	878	877	876	875	874	873	872	871	870	869	868	867	866	865	864	863	862	861	860	859	858	857	856	855	854	853	852	851	850	849	848	847	846	845	844	843	842	841	840	839	838	837	836	835	834	833	832	831	830	829	828	827	826	825	824	823	822	821	820	819	818	817	816	815	814	813	812	811	810	809	808	807	806	805	804	803	802	801	800	799	798	797	796	795	794	793	792	791	790	789	788	787	786	785	784	783	782	781	780	779	778	777	776	775	774	773	772	771	770	769	768	767	766	765	764	763	762	761	760	759	758	757	756	755	754	753	752	751	750	749	748	747	746	745	744	743	742	741	740	739	738	737	736	735	734	733	732	731	730	729	728	727	726	725	724	723	722	721	720	719	718	717	716	715	714	713	712	711	710	709	708	707	706	705	704	703	702	701	700	699	698	697	696	695	694	693	692	691	690	689	688	687	686	685	684	683	682	681	680	679	678	677	676	675	674	673	672	671	670	669	668	667	666	665	664	663	662	661	660	659	658	657	656	655	654	653	652	651	650	649	648	647	646	645	644	643	642	641	640	639	638	637	636	635	634	633	632	631	630	629	628	627	626	625	624	623	622	621	620	619	618	617	616	615	614	613	612	611	610	609	608	607	606	605	604	603	602	601	600	599	598	597	596	595	594	593	592	591	590	589	588	587	586	585	584	583	582	581	580	579	578	577	576	575	574	573	572	571	570	569	568	567	566	565	564	563	562	561	560	559	558	557	556	555	554	553	552	551	550	549	548	547	546	545	544	543	542	541	540	539	538	537	536	535	534	533	532	531	530	529	528	527	526	525	524	523	522	521	520	519	518	517	516	515	514	513	512	511	510	509	508	507	506	505	504	503	502	501	500	499	498	497	496	495	494	493	492	491	490	489	488	487	486	485	484	483	482	481	480	479	478	477	476	475	474	473	472	471	470	469	468	467	466	465	464	463	462	461	460	459	458	457	456	455	454	453	452	451	450	449	448	447	446	445	444	443	442	441	440	439	438	437	436	435	434	433	432	431	430	429	428	427	426	425	424	423	422	421	420	419	418	417	416	415	414	413	412	411	410	409	408	407	406	405	404	403	402	401	400	399	398	397	396	395	394	393	392	391	390	389	388	387	386	385	384	383	382	381	380	379	378	377	376	375	374	373	372	371	370	369	368	367	366	365	364	363	362	361	360	359	358	357	356	355	354	353	352	351	350	349	348	347	346	345	344	343	342	341	340	339	338	337	336	335	334	333	332	331	330	329	328	327	326	325	324	323	322	321	320	319	318	317	316	315	314	313	312	311	310	309	308	307	306	305	304	303	302	301	300	299	298	297	296	295	294	293	292	291	290	289	288	287	286	285	284	283	282	281	280	279	278	277	276	275	274	273	272	271	270	269	268	267	266	265	264	263	262	261	260	259	258	257	256	255	254	253	252	251	250	249	248	247	246	245	244	243	242	241	240	239	238	237	236	235	234	233	232	231	230	229	228	227	226	225	224	223	222	221	220	219	218	217	216	215	214	213	212	211	210	209	208	207	206	205	204	203	202	201	200	199	198	197	196	195	194	193	192	191	190	189	188	187	186	185	184	183	182	181	180	179	178	177	176	175	174	173	172	171	170	169	168	167	166	165	164	163	162	161	160	159	158	157	156	155	154	153	152	151	150	149	148	147	146	145	144	143	142	141	140	139	138	137	136	135	134	133	132	131	130	129	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19

IVF-Flat

Inverted File Index

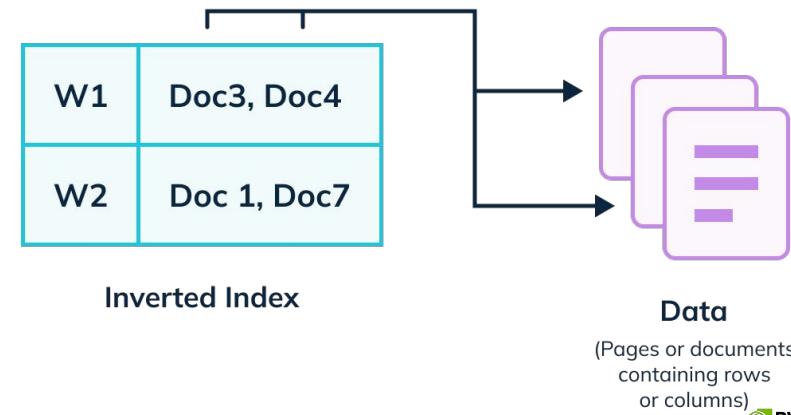
IVF-Flat

- Flat == unmodified vectors (i.e. no compression)
- Simple knobs to reduce the overall search space
- Trade-off accuracy for speed.

Forward Index (~ table of contents)



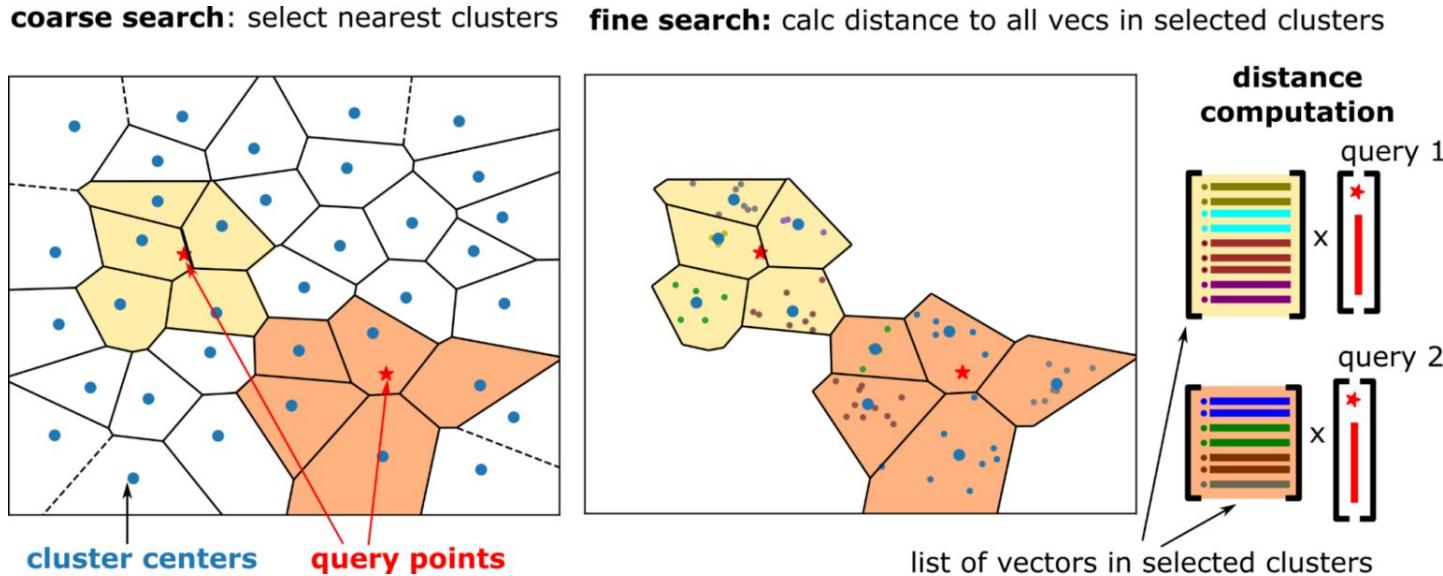
Inverted Index (~ back of the book index)



IVF-Flat

Coarse Search and Fine Search

The top-k neighbors from each probed cluster are selected, which results in **n_probes * k_neighbor** candidates for each query. This is reduced to the k-nearest neighbors.



IVF-Flat

Tuning parameters for index building

Tuning n_lists parameter

- **n_lists** parameter determines the number of clusters to partition the training dataset.
- **n_probes** determines the number of closest clusters to search through to compute the nearest neighbors for a set of query points.
- $n_lists == \sqrt{n_samples}$ is a good starting point (where $n_samples$ is the number of vectors in the dataset).
- $n_lists == n_probes$ is like an exact (brute force) search

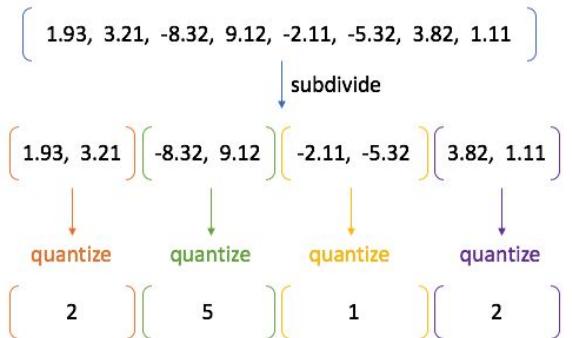
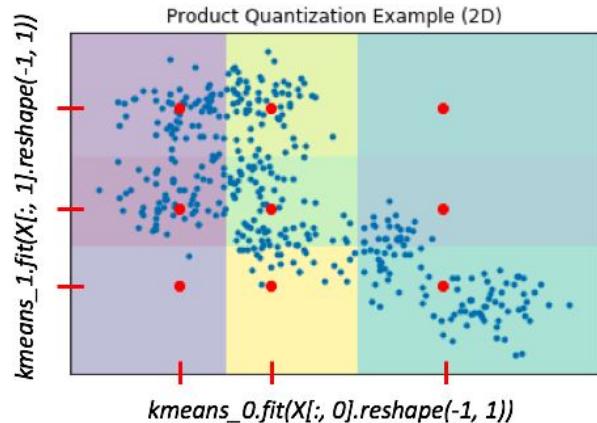
Optimizations

- **High throughput.** cuVS uses Tensor Cores to accelerate the k-means clustering during index building.
- **Improved algorithm.** cuVS uses a balanced hierarchical k-means clustering, which clusters the dataset efficiently even as the number of vectors reaches hundreds of millions.
- **Optimized top-k candidates.** cuVS has highly optimized methods to select the top-k candidates.

IVF-PQ

Product Quantization

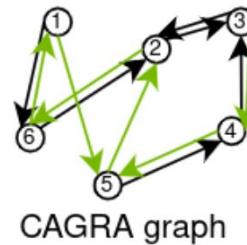
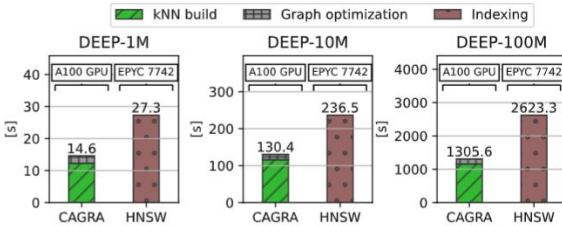
- Lower PQ bits (4-8) provide **better compression** and more **efficient use of shared memory**
- Configurable lookup table and distance precision provide **faster computation** and **efficient use of shared memory**
- Support for **reduced precision** (uint8 and int8)
- Supports custom predicate **pre-filter**
- **pq_dim** sets the target dimensionality of the compressed vector.
- **pq_bits** sets the number of bits for each vector element after compression.



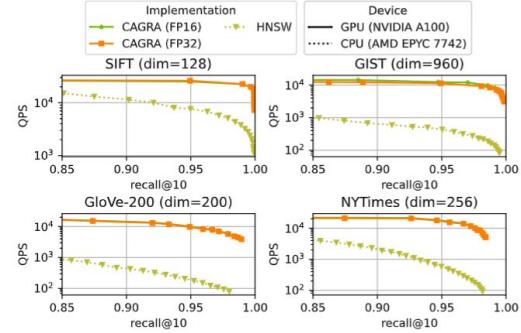
CAGRA

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

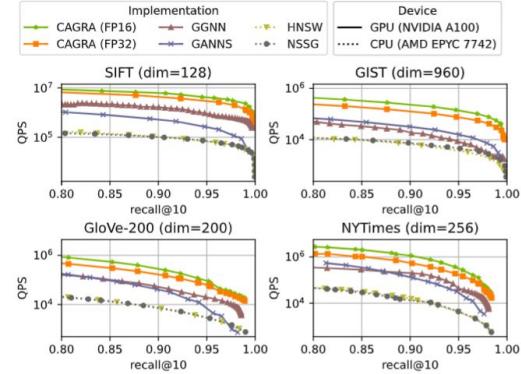
- *Individual queries* parallelized during search
- Setting records for both *single query* and *large batch* performance
- *Higher throughput* than existing GPU Graph ANNs and *lower latency* than SOTA CPU Graph ANNs



Single query at a time

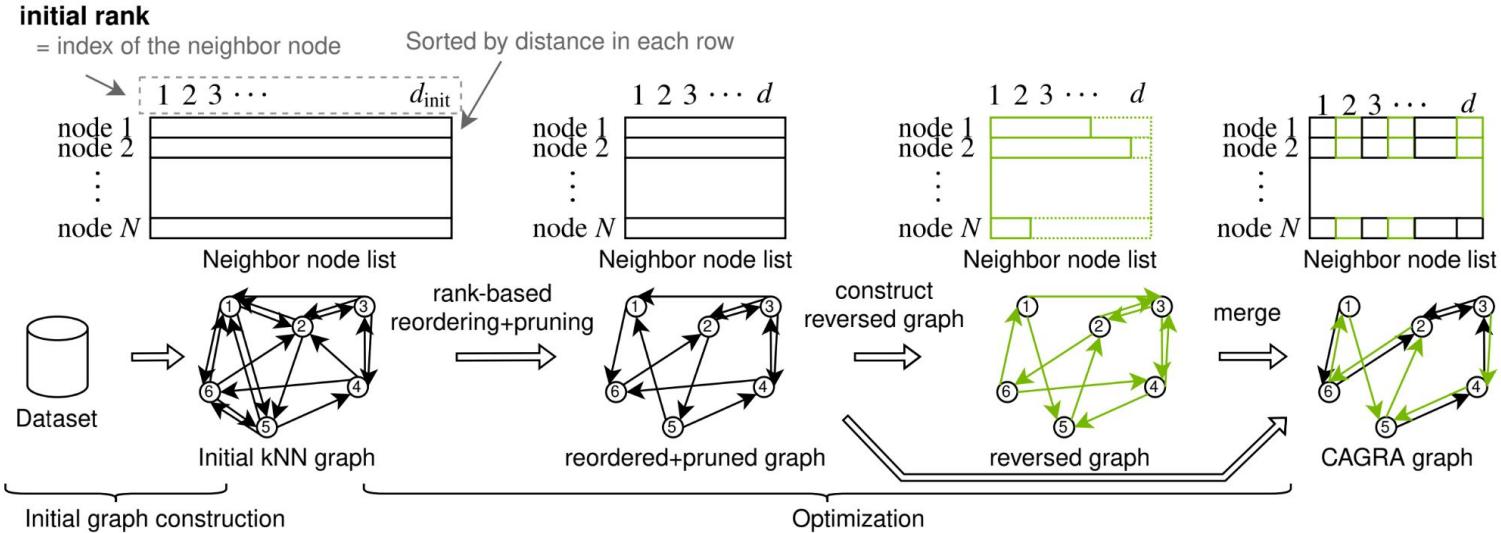


Batches of 10k queries



CAGRA

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



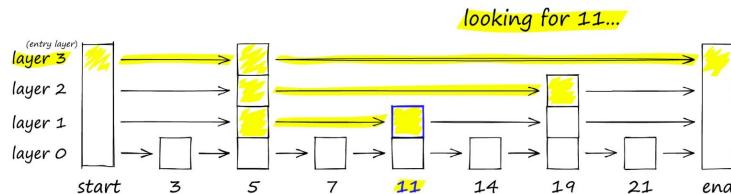
["CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search on the GPU"](#), Ootomo et al., 2023

CPU
Only

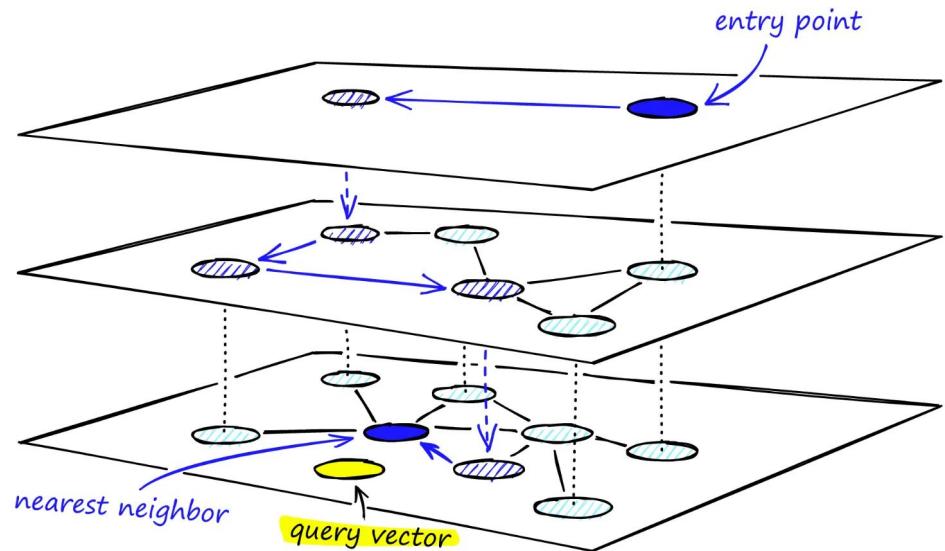
HNSW Overview

Hierarchical Navigable Small World (HNSW) algorithm

Skip List



HNSW Graph

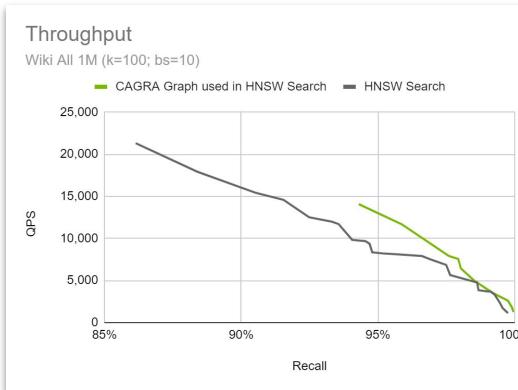
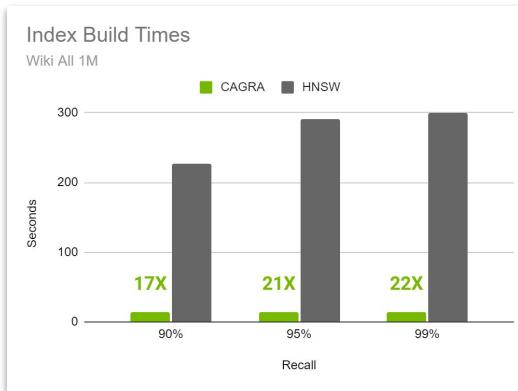


CAGRA+HNSW

Building index on GPU and searching on CPU

Organizations that prefer to use CPU infrastructure for search can still benefit from GPU accelerated build times, because CAGRA indexes built on GPUs are compatible with HNSW.

- HNSW can be slightly modified to accept CAGRA graphs
- CAGRA graphs are then deployed to HNSW on the CPU
- The result is the CAGRA graph often outperforms HNSW on CPU search – especially for larger dimensional data

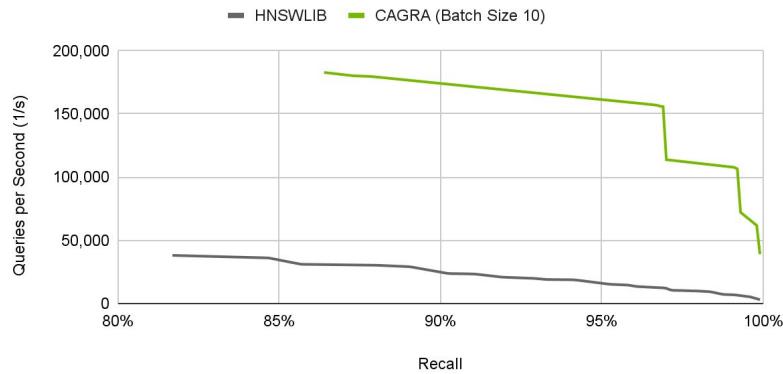


CAGRA (GPU) vs. HNSW (CPU)

Small batches (bs=10): Low latencies drive higher throughput

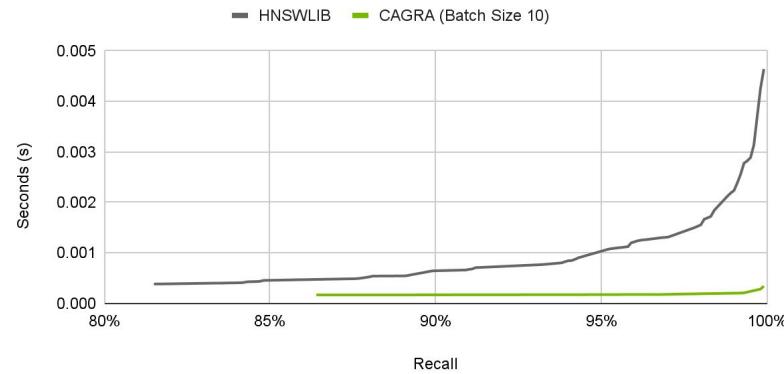
Throughput

BIGANN (10M; 128 Dim; k=100)



Latency

BIGANN (10M; 128 Dim; k=100)

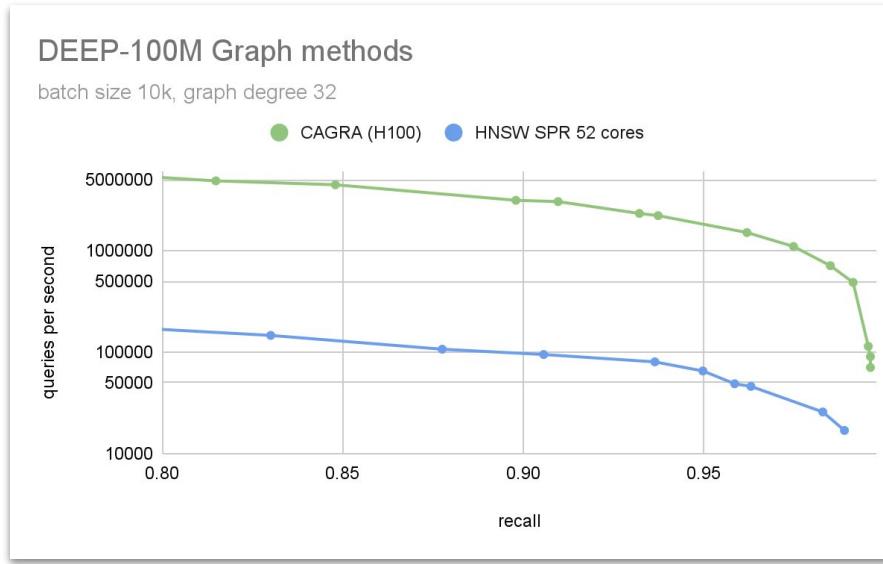


Recall	CAGRA	HNSW	Speedup
90%	174,153	24,816	7.0X
95%	161,356	16,008	10.1X
99%	108,029	6,940	15.6X

Recall	CAGRA	HNSW	Speedup
90%	0.000171	0.000647	3.8X
95%	0.000176	0.001035	5.9X
99%	0.000204	0.002236	11.0X

CAGRA (GPU) vs. HNSW (CPU)

Large batches (bs=10k): Increased parallelism drives larger speedups



Recall	CAGRA	HNSW	Speedup
90%	3,133,965	97,451	32X
95%	1,861,029	64,988	29X
99%	574,209	16,939	34X

CAGRA Index Build Times

Compared to HNSW

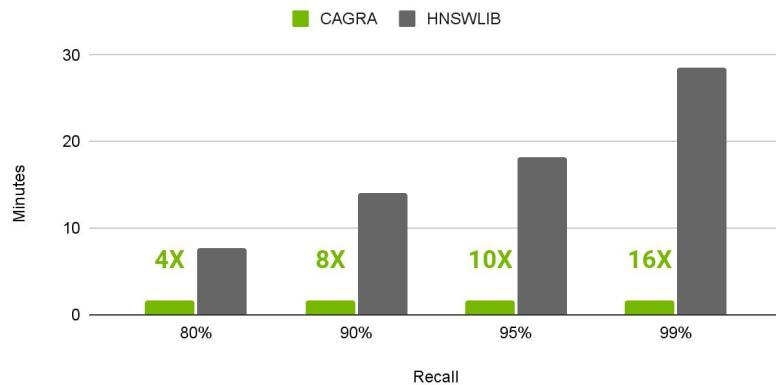
GPUs are especially good at offline processes where large batch sizes can be used

Index build time increases as:

- Recall increases
- Dimensions increases
- Embeddings increase

BIGANN 10M (128 Dim) Build Time

A10g vs AMD Graviton2



Wiki All 1M (768Dim) Build Time

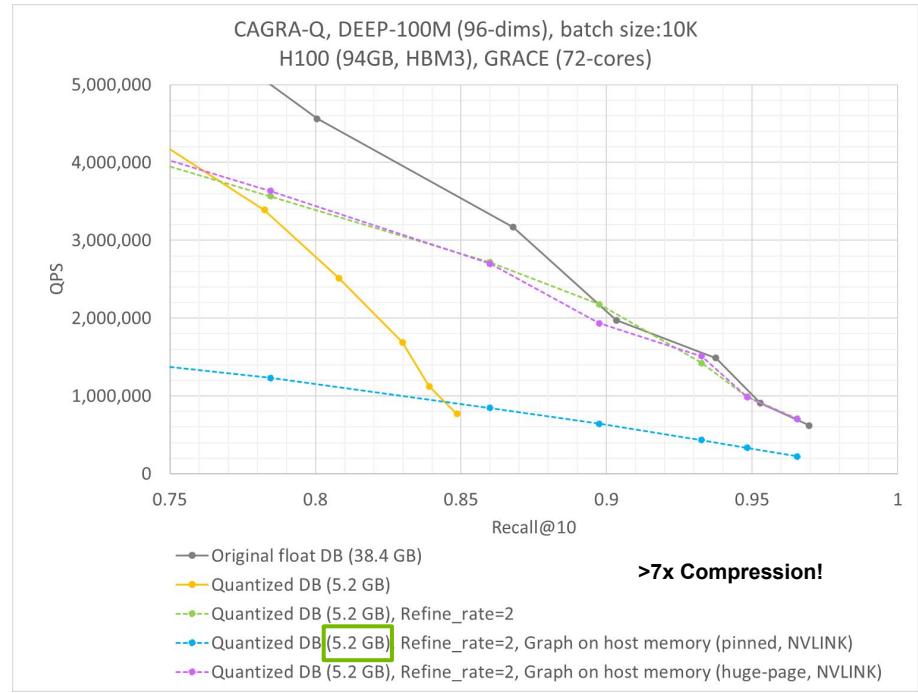
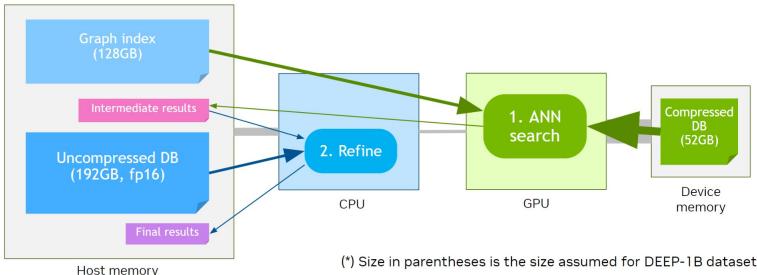
A10g vs Intel Xeon Ice Lake



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



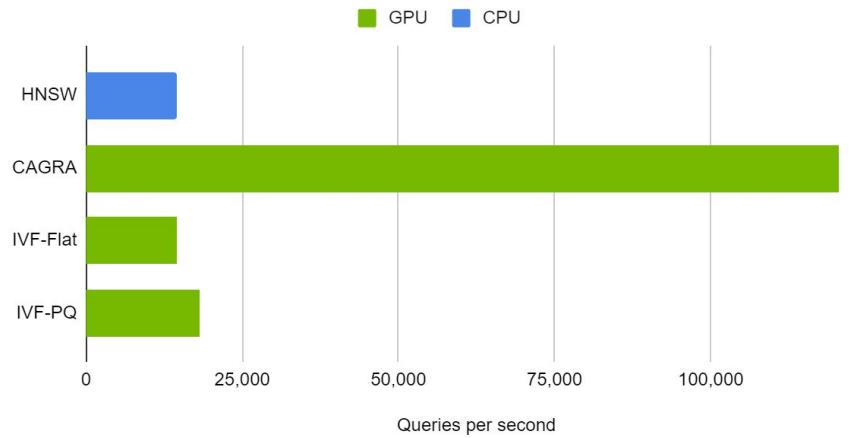
- CAGRA-Q makes a great companion for *Grace Hopper* and improved chip-to-chip (C2C) bandwidth.
- TLDR; Compressed dataset on device and graph stored in huge page pinned memory has *equivalent performance* to original dataset and graph stored on device at high recall levels.

Scaling up cuVS

Deep 100M; 96 Dimensions; Single GPU; 38GB

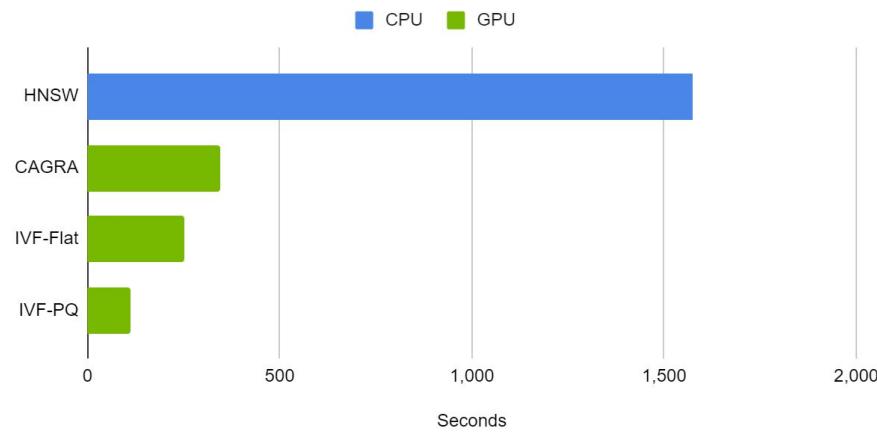
8X Higher Throughput vs CPU (Higher is better)

DEEP 100M dataset; k = 10; batch size = 10; 95% recall



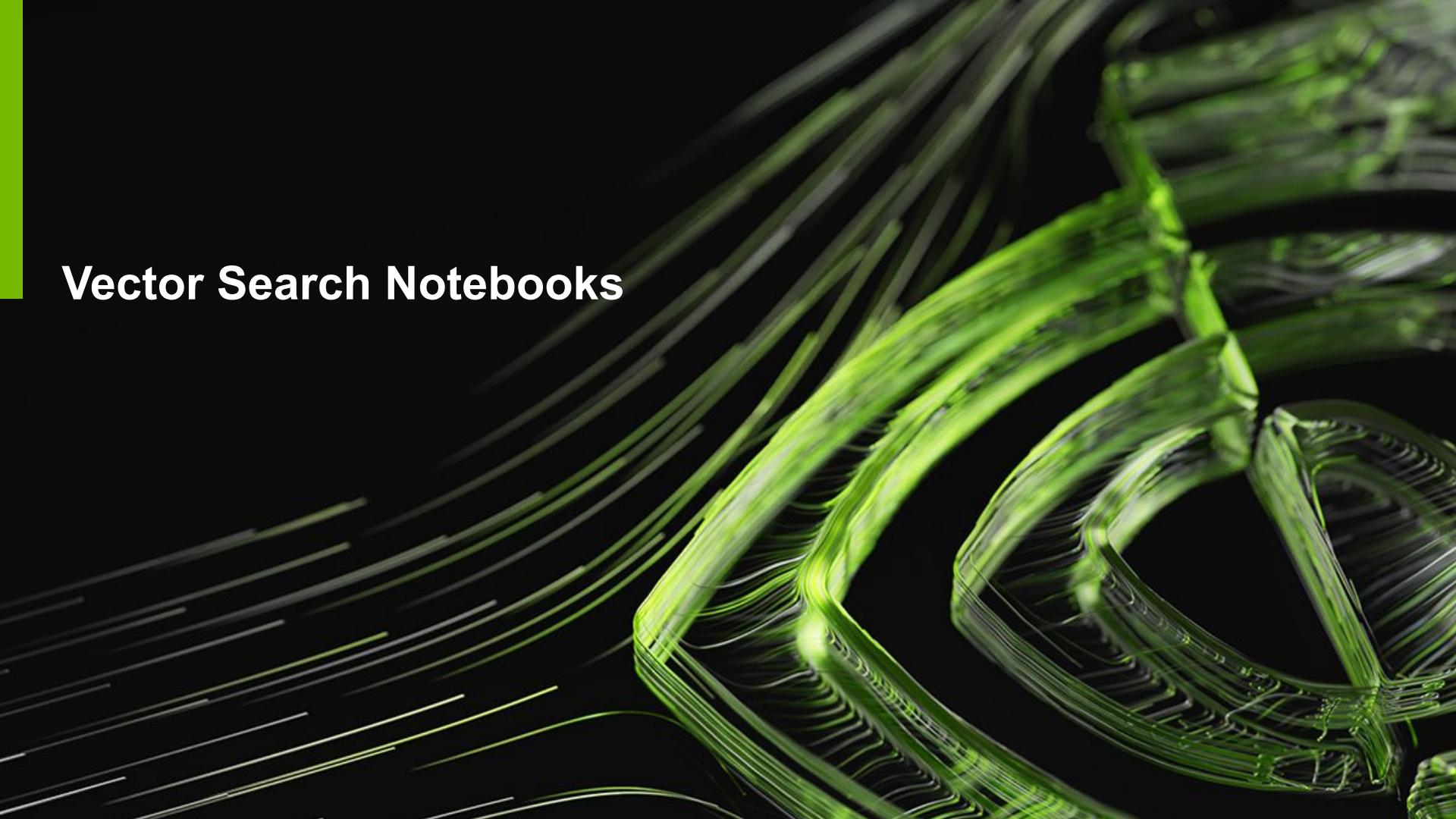
5X Faster Index Build Time vs CPU (Lower is better)

DEEP 100M dataset; k = 10; batch size = 10; 95% recall



NVIDIA DGX H100 SXM GPU vs Intel Sapphire Rapids 8480CL CPU (224 threads)

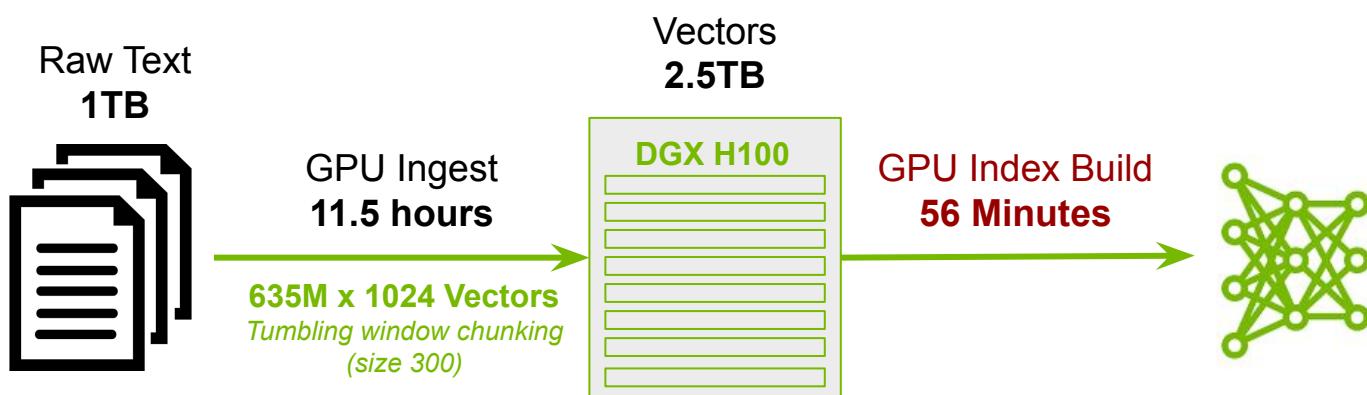
Vector Search Notebooks



Scaling cuVS - Multiple GPUs (1TB)

Massive-scale indexing on a DGX H100

GPU
Benchmark



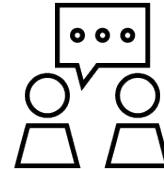
CPU
Estimate

GPU Index Build Time	56 Min
Number of GPUs	× 8
GPU / CPU Performance	× 20
Minutes per day	÷ 1440 Min/Day
CPU Estimated Index Build	6.22 Days

Assumes weak scaling

Resources

A Variety of Ways to Get Up & Running



More about RAPIDS cuVS

- RAPIDS [GTC Talk](#)
- cuVS IVF-PQ [GTC Talk](#)
- cuVS CAGRA [arXiv](#)
- NVIDIA Tech [Blog](#)

Discussion & Support

- Check the [RAPIDS cuVS GitHub](#)
- [C++ API](#) documentation
- [Python API](#) documentation
- Talk to [NVIDIA Services](#)



@RAPIDSai



<https://github.com/rapidsai/cuvs>



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

RAPIDS

<https://rapids.ai>

