Analyzing the Performance of Vector Databases

Jayjeet Chakraborty

UC Santa Cruz

UC SANTA CRUZ | Baskin Engineering | Center for Research in Systems and Storage

About Me

- 3rd yr. PhD Student @ UC Santa Cruz
- Research Areas
  - Databases and Data Management
  - Programmable Storage Systems
  - Hardware Software Co-Design
- Working with the Centre for Research in Systems and Storage, UCSC
- Publications
  - Popper (CANOPIE-HPC ‘20)
  - Skyhook (CCGrid ‘22)
  - Apache Arrow Datafusion (SIGMOD ‘24)
- Internships
  - Princeton University (IRIS-HEP); Fall 2020, Winter 2021, Spring 202
  - InfluxData (InfluxDB); Summer 2023
  - NVIDIA (RAPIDs Team); Summer 2024

Background

Large Language Models (LLMs)

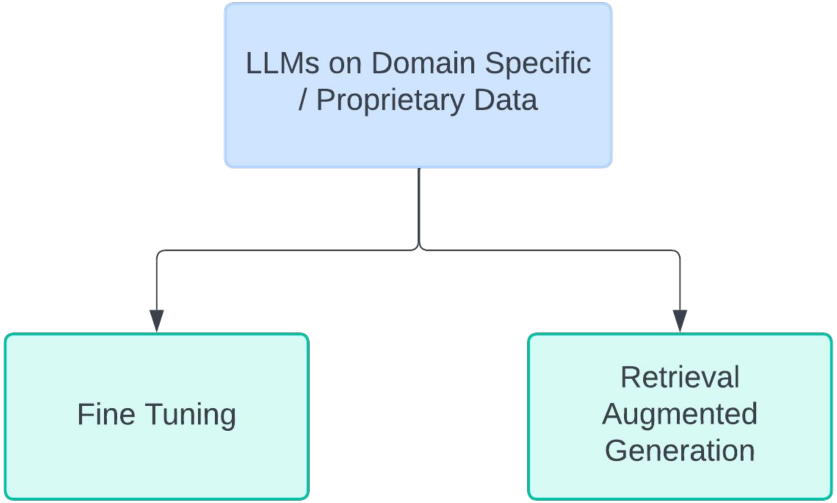
Massive “**Transformer-Decoder**” models trained on large amounts of data

Specifically focussed on language understanding and generation

Used for question answering, summarization, content creation, code development, translation, etc

Source: Google Images

Large Language Models (LLMs)



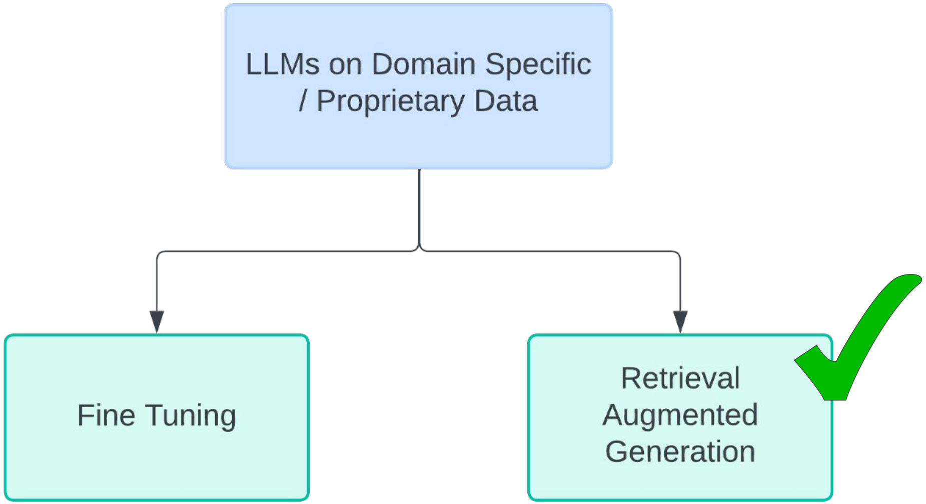
Fine Tuning

- High Infra Costs
- Time Consuming
- Training Data Dependent
- Loses Generic Capabilities

Retrieval Augmented Generation

- Minimal Costs
- Flexible Sources of Knowledge
- Retains Original Model Capabilities

Large Language Models (LLMs)



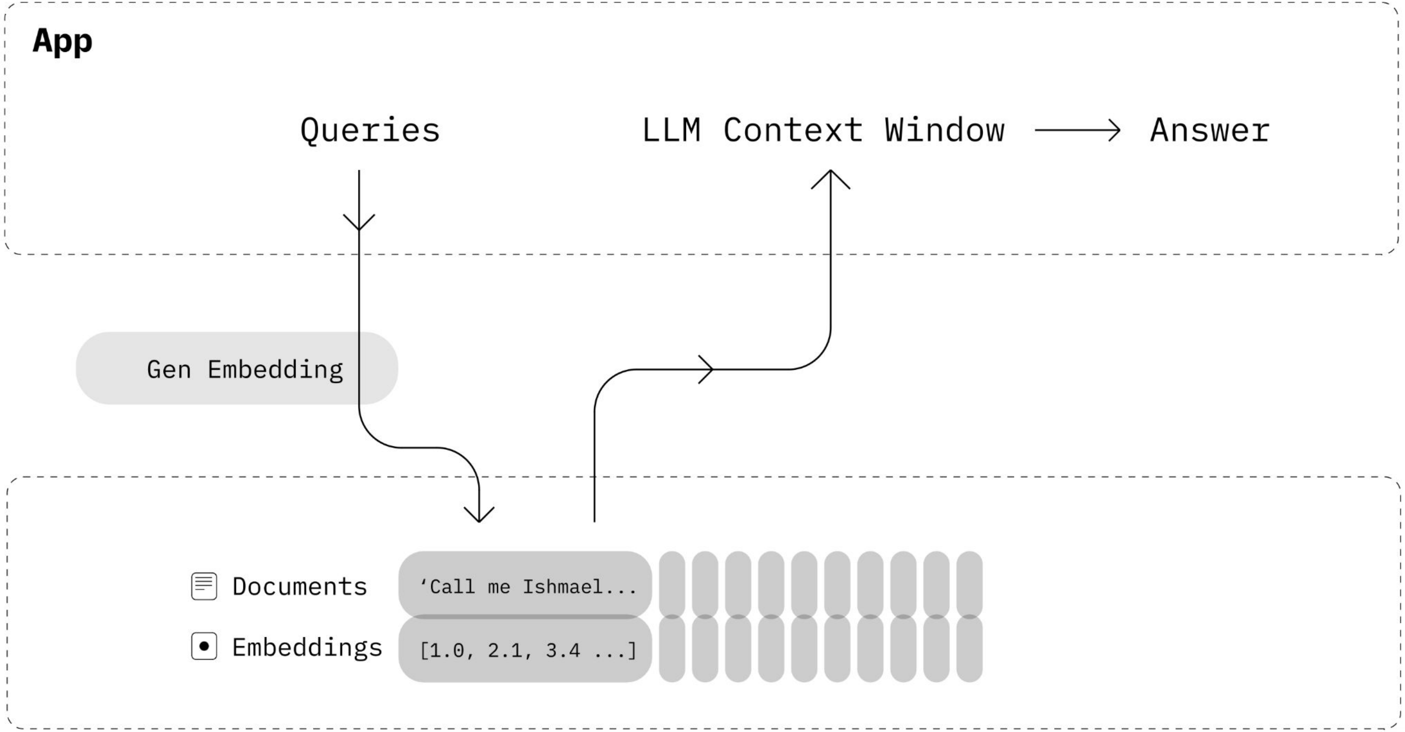
Fine Tuning

- High Infra Costs
- Time Consuming
- Training Data Dependent
- Loses Generic Capabilities

Retrieval Augmented Generation

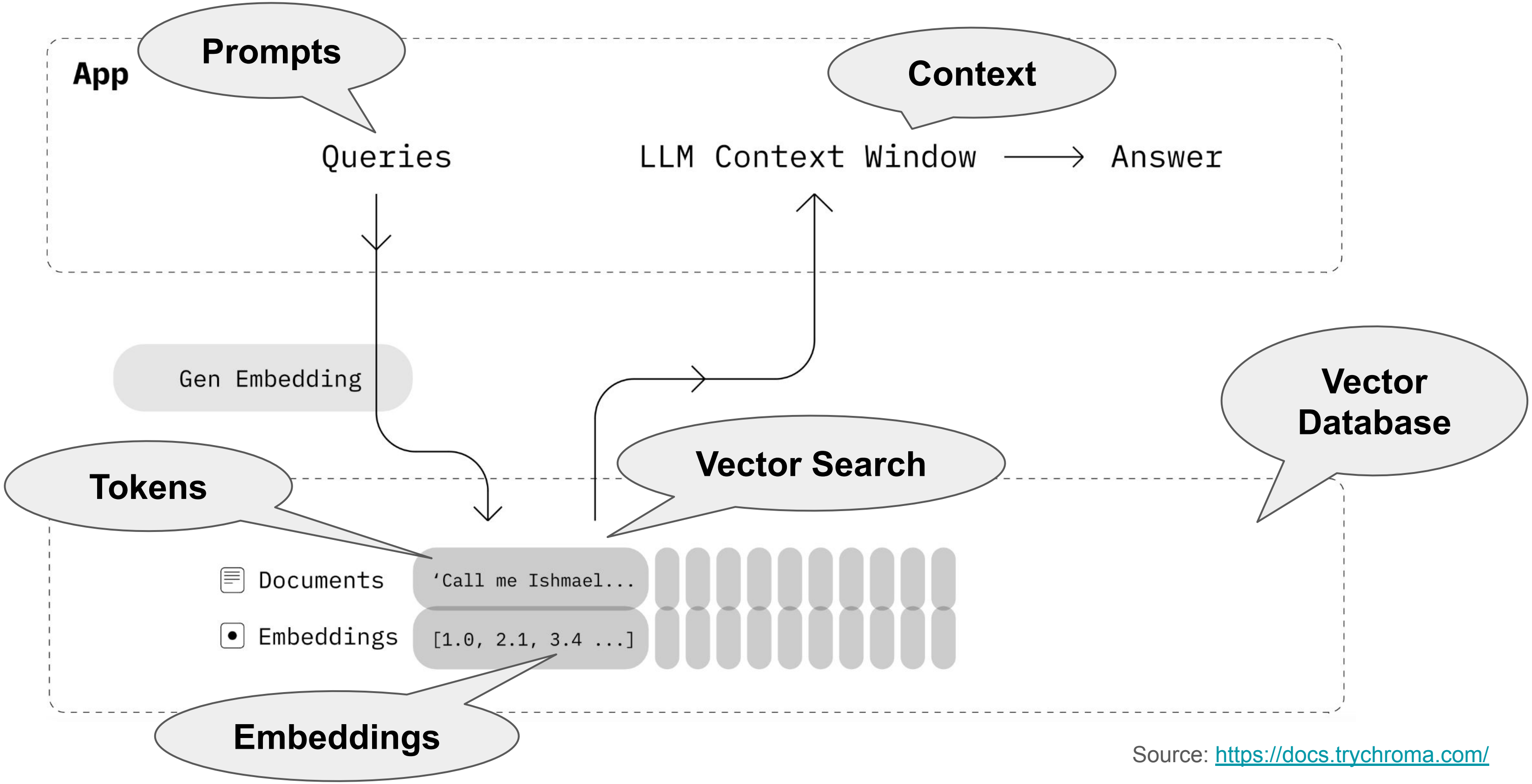
- Minimal Costs
- Flexible Sources of Knowledge
- Retains Original Model Capabilities

Retrieval Augmented Generation (RAG)



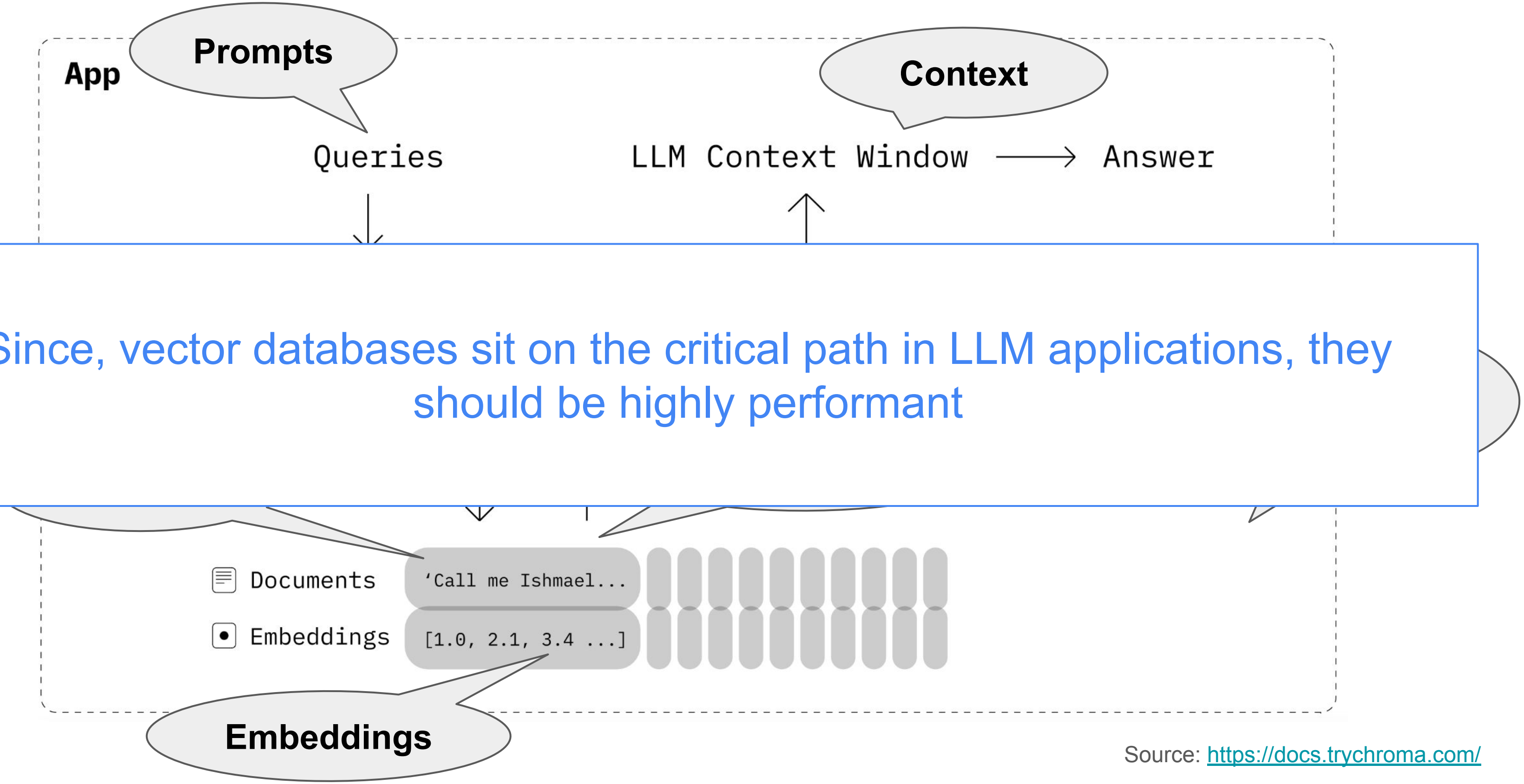
Source: https://docs.trychroma.com/

Retrieval Augmented Generation (RAG)



Source: https://docs.trychroma.com/

Retrieval Augmented Generation (RAG)



Since, vector databases sit on the critical path in LLM applications, they should be highly performant

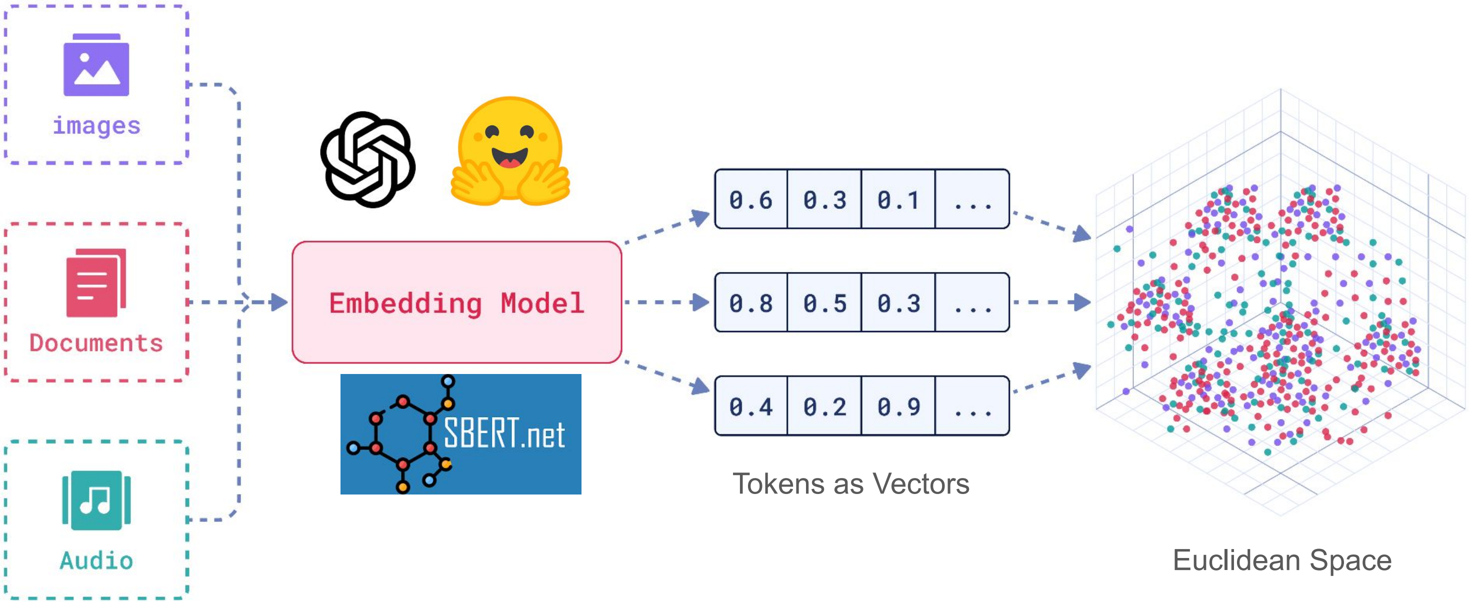
Source: https://docs.trychroma.com/

Tokens / Tokenization

- Tokens are the basic units of data processed by an LLM
  - Words
  - Subwords
  - Sentences
- Tokenization is the process of splitting large corpus of texts into smaller pieces or tokens
- Recommendations for Tokenization
  - Simple searches, smaller chunks; Complex searches, larger chunks
  - Larger chunks = More Context = But less tokens that fit in the LLM context window
- Examples of Tokenizers
  - nltk.tokenize
  - Hugging Face Tokenizer API
  - `BertTokenizer` and `AutoTokenizer` from transformers

NLTK

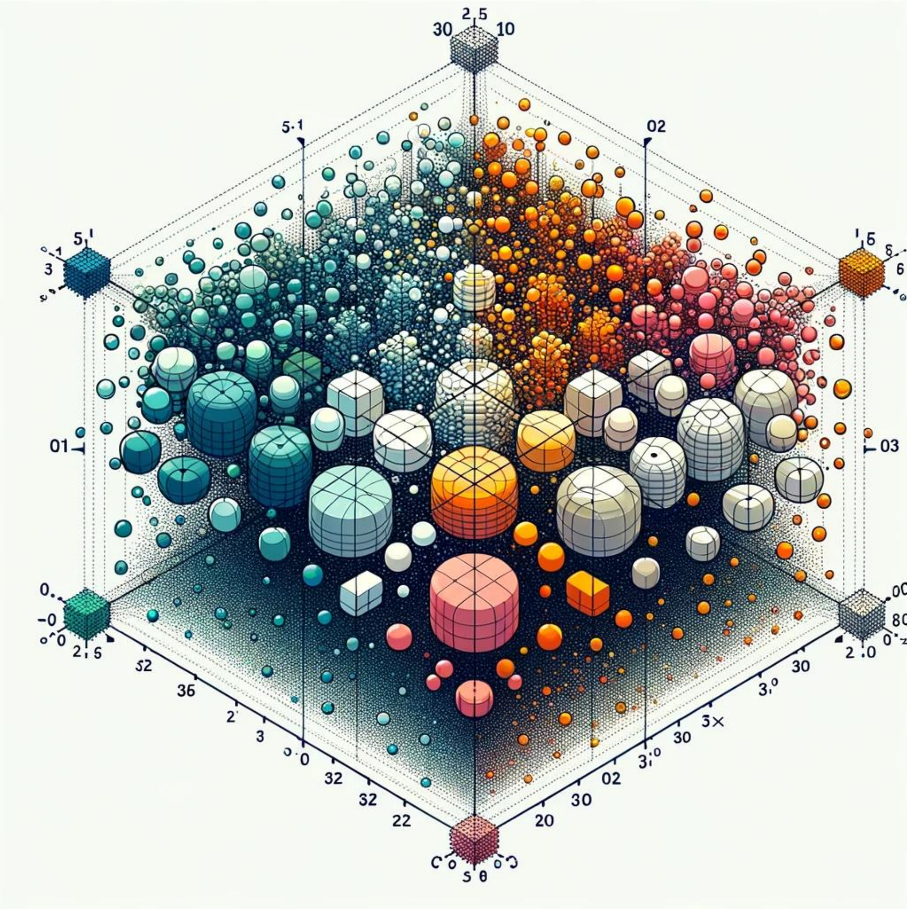
Embeddings



Tokens from Multimodal Data

Source: https://qdrant.tech/articles/what-are-embeddings/

Vector embeddings when laid out in a multi-dimensional space form clusters of **semantically similar** tokens



Source: Generated by DALL-E

Top Embedding Models

Retrieval English leaderboard 🔎

- **Metric:** Normalized Discounted Cumulative Gain @ k (ndcg_at_10)
- **Languages:** English

Metric: Normalized Discounted Cumulative Gain (NDCG)

Rank	Model	Model Size (Million Parameters)	Memory Usage (GB, fp32)	Average	ArguAna	ClimateFEVER	CQADupstackRetrieval	DBPedi
1	Linq-Embed-Mistral	7111	26.49	60.19	69.65	39.11	47.27	51.32
2	NV-Embed-v1			59.36	68.2	34.72	50.51	48.29
3	SFR-Embedding-Mistral	7111	26.49	59	67.17	36.41	46.49	49.06
4	voyage-large-2-instruct			58.28	64.06	32.65	46.6	46.03
5	gte-large-en-v1.5	434	1.62	57.91	72.11	48.36	42.16	46.3
6	GritLM-7B	7242	26.98	57.41	63.24	30.91	49.42	46.6
7	e5-mistral-7b-instruct	7111	26.49	56.89	61.88	38.35	42.97	48.89
8	LLM2Vec-Meta-Llama-3-supervis	7505	27.96	56.63	62.78	34.27	48.25	48.34
9	voyage-lite-02-instruct	1220	4.54	56.6	70.28	31.95	46.2	39.79
10	SE_v1			56.55	61.42	30.33	49.94	49.03
11	gte-Qwen1.5-7B-instruct	7099	26.45	56.24	62.65	44	40.64	48.04

Source: https://huggingface.co/spaces/mteb/leaderboard

Scale of Embeddings

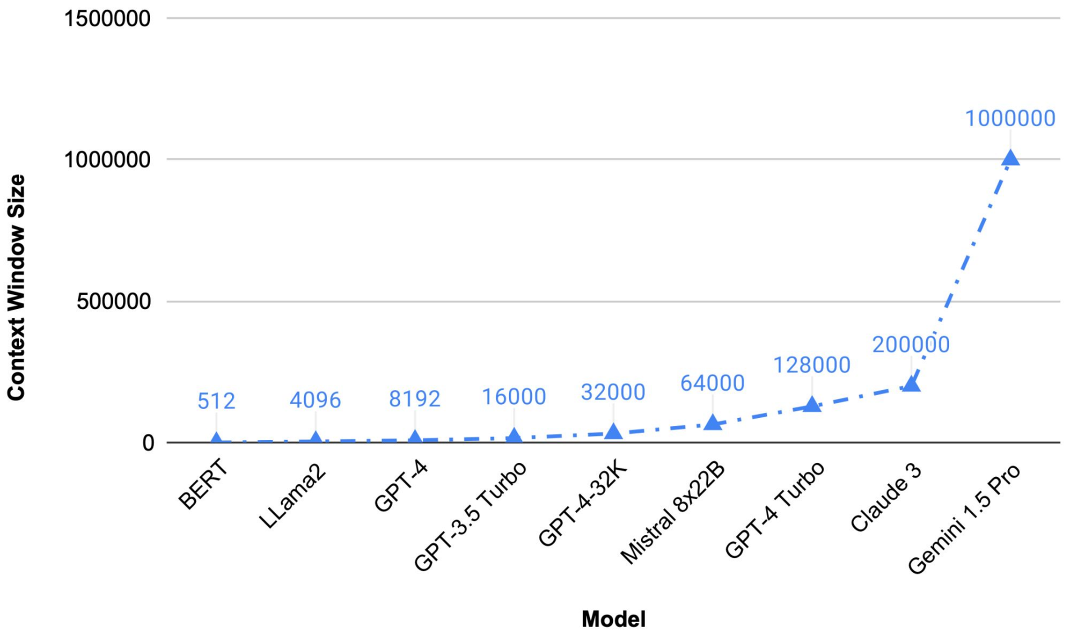
- OpenAI
  - text-embedding-3-small: 1536 dims
    - 1536 * 4 bytes = 6 KB
    - 6 KB * 1B = **6 TB**
    - 6 KB * 1T = **6 PB**
  - text-similarity-davinci-001: 12288 dims
    - 12288 * 4 bytes = 49 KB
    - 49 KB * 1B = **49 TB**
    - 49 KB * 1T = **49 PB**

Huge DRAM capacities required for processing billion / trillion scale vector datasets

Source: openai.com

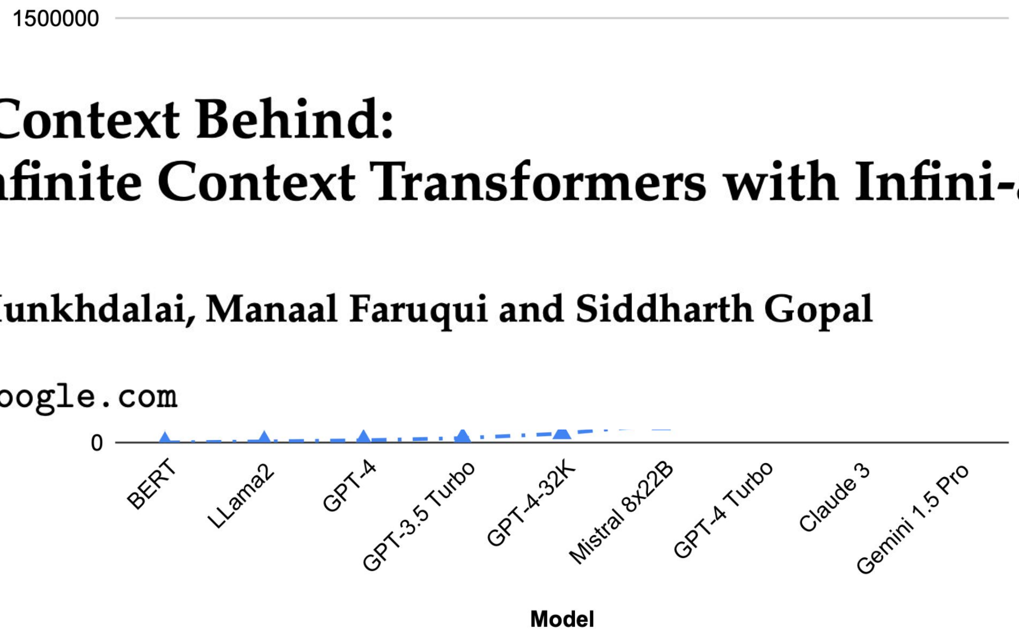
Context Windows in LLMs

The maximum amount of text in the form of “tokens” that an LLM can consider at any given time while generating a response



Context Windows in LLMs

The maximum amount of text in the form of "tokens" that an LLM can consider at any given time while generating a response



Leave No Context Behind:
Efficient Infinite Context Transformers with Infini-attention

Tsendsuren Munkhdalai, Manaal Faruqui and Siddharth Gopal
Google
tsendsuren@google.com

Vector Databases

- Indexes and stores high-dimensional vector embeddings and tokens for fast similarity searches and retrieval

- Consistency guarantees, multi-tenancy, cloud-native, CRUD, logging and recovery, serverless, etc

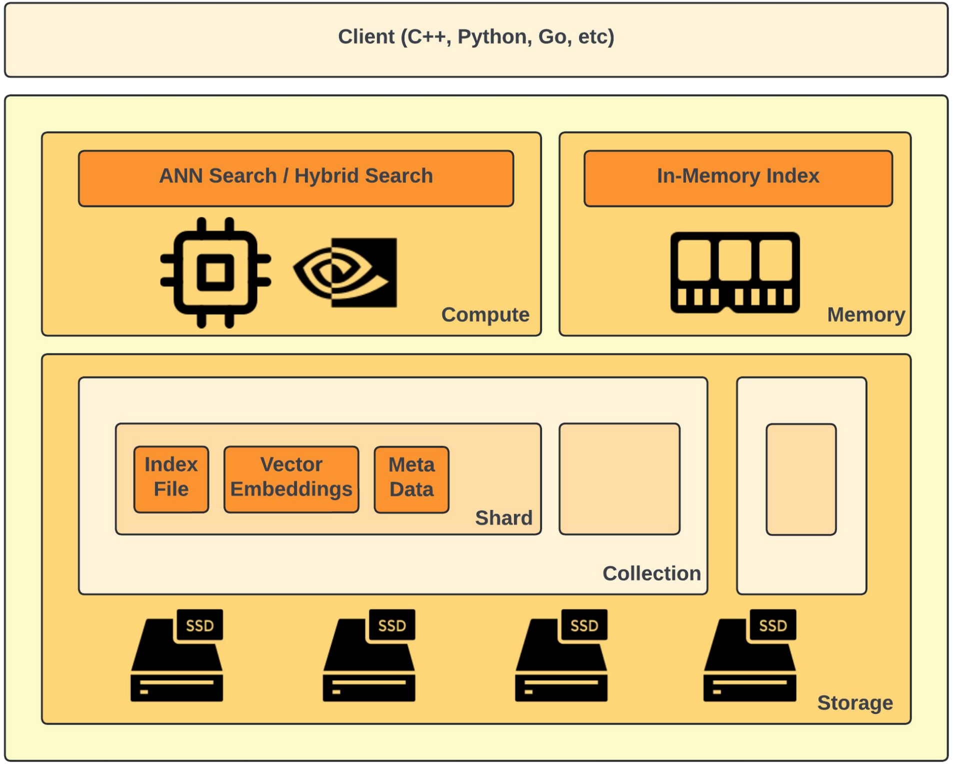
Source: Google Images

Vector Databases

How do vector databases compare to relational databases ?

	Relational Database	Vector Database
Indexing	B/B+ Tree, LSM-Tree	HNSW, IVF, LSH
Compute	Filter, Project, Aggregate, Sort	ANN, KNN, Hybrid Search
Data Access	Index loaded page-by-page	Entire index in memory
Query Interface	SQL, JDBC, ODBC	Python, C++, REST APIs
Performance Metric	Transactions / Second	Queries / Second, Recall

Architecture of Vector Databases



Types of Vector Databases

- Client-Server
  - Milvus, **Qdrant**, Weaviate
- Embedded
  - LanceDB, Chroma, DeepLake
- Extensions
  - **PGVector**, DuckDB Vector, Redis Vector Search
- Libraries
  - FAISS, **HNSWLIB**, usearch, NVIDIA Raft (CAGRA)

Indexing Algorithms in Vector Databases

Pinecone	Proprietary composite index
milvus / zilliz	Flat, Annoy, IVF, HNSW/RHNSW (Flat/PQ), DiskANN
Weaviate	Customized HNSW, HNSW (PQ), DiskANN (in progress...)
qdrant	Customized HNSW
chroma	HNSW
LanceDB	IVF (PQ), DiskANN (in progress...)
vespa	HNSW + BM25 hybrid
Vald	NGT
elasticsearch	Flat (brute force), HNSW
redis	Flat (brute force), HNSW
pgvector	IVF (Flat), IVF (PQ) in progress... HNSW

Source: http://thedataquarry.com/posts/vector-db-1/

Indexing Algorithms in Vector Databases

Pinecone	Proprietary composite index
milvus / zilliz	Flat, Annoy, IVF, HNSW/RHNSW (Flat/PQ), DiskANN
Weaviate	Customized HNSW, HNSW (PQ), DiskANN (in progress...)

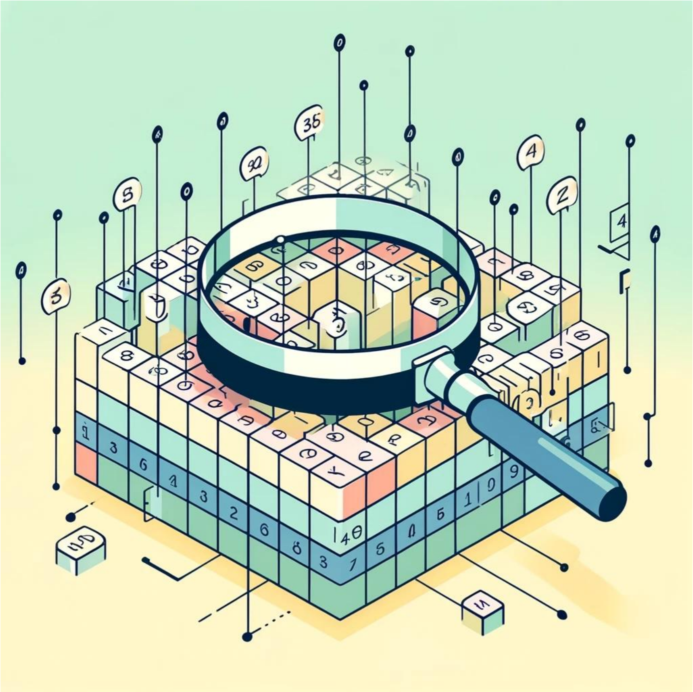
HNSW is the Most Widely Supported Indexing Algorithm

elasticsearch	Flat (brute force), HNSW
redis	Flat (brute force), HNSW
pgvector	IVF (Flat), IVF (PQ) in progress... HNSW

Source: http://thedataquarry.com/posts/vector-db-1/

Vector Search

- Finding tokens similar to a query using nearest neighbor searches
- Traditionally, **KNN** has been used
  - But on billions / trillions of data points, not feasible
- Using **ANN** (Approximate Nearest Neighbor) algorithms allows **trading off accuracy for search speed**
- Use Cases
  - Retrieval Augmented Generation, Recommendation Models, Classification, Clustering



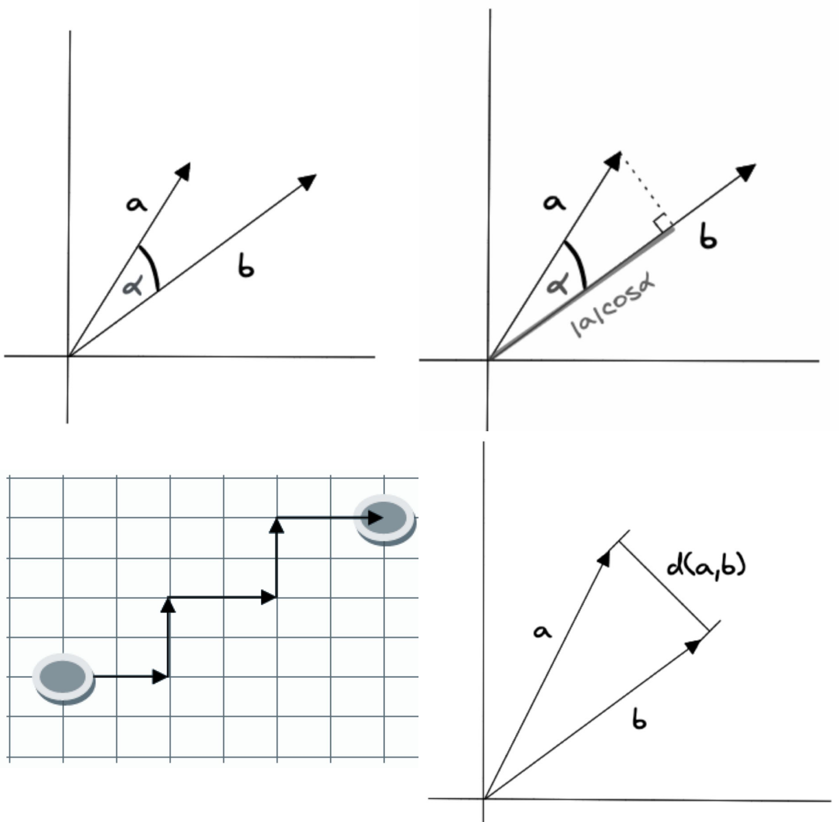
Source: Generated by DALL-E

Hybrid Search

- Hybrid search combines the results of a **vector search** and a **keyword search** by fusing the two result sets
- Uses inverted index for keyword lookups and vector index for vector similarity searches
- How does it work ?
  - First, both vector and keyword search are run independently on the dataset to generate unique top K result sets
  - Then, the above two lists are combined in a single unified list using a special re-ranking algorithm
    - **RRF (Reciprocal Rank Fusion)**: Give more weight to the top-ranked item in each individual list when building the fused list
    - Example: If a record X is ranked 3 in one list and 9 in the other, then its score would be 1/3 + 1/9 = 0.444
  - The items are ranked in descending order to their scores in the fused list

Vector Similarity Metrics

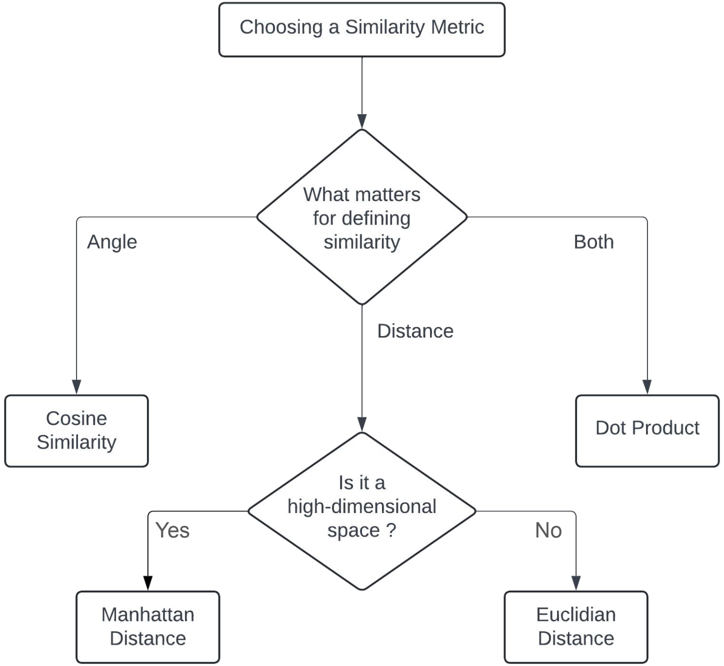
- Cosine Similarity
- Dot Product
- Manhattan Distance (L1)
- **Euclidean Distance (L2)**



Source: https://www.pinecone.io/learn/vector-similarity/

Vector Similarity Metrics

- Choosing a metric ?
  - To define similarity b/w two vectors, follow the chart given here
  - For indexing, use the similarity metric used to train your embedding model
  - For searches, use the similarity metric used to create your index
  - Calculation Speed
    - Manhattan > Cosine > Dot Product > L2

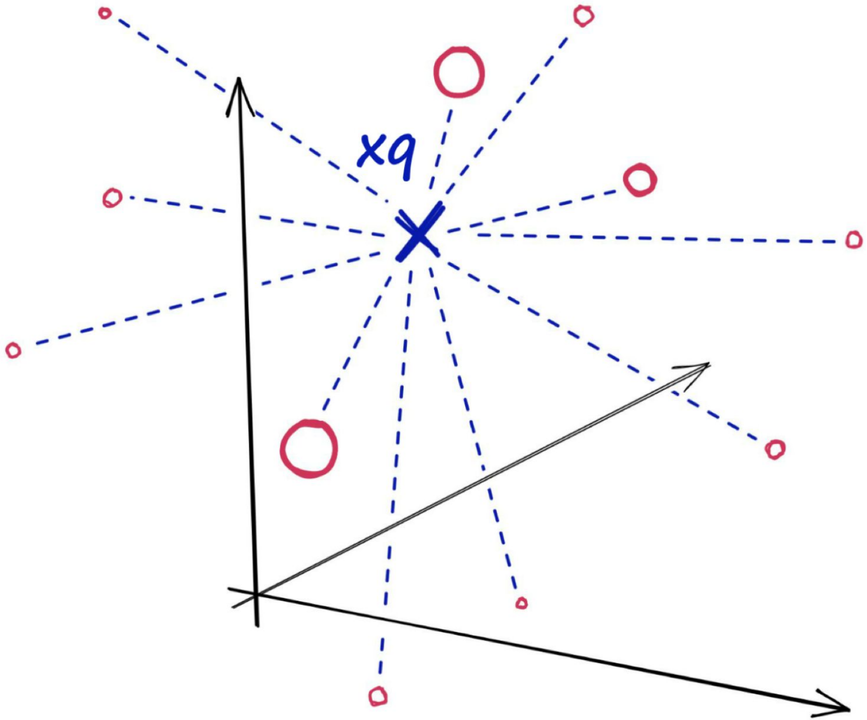


Vector Indexing Algorithms

- Flat
- IVF or Inverted File Index (clustering-based)
- LSH or Locality Sensitive Hashing (hashing-based)
- **HNSW or Hierarchical Navigable Small Worlds** (graph-based)
- Others
  - Microsoft DiskANN
  - Spotify ANNOY
  - Google ScaNN
  - NVIDIA Cagra

Flat Index

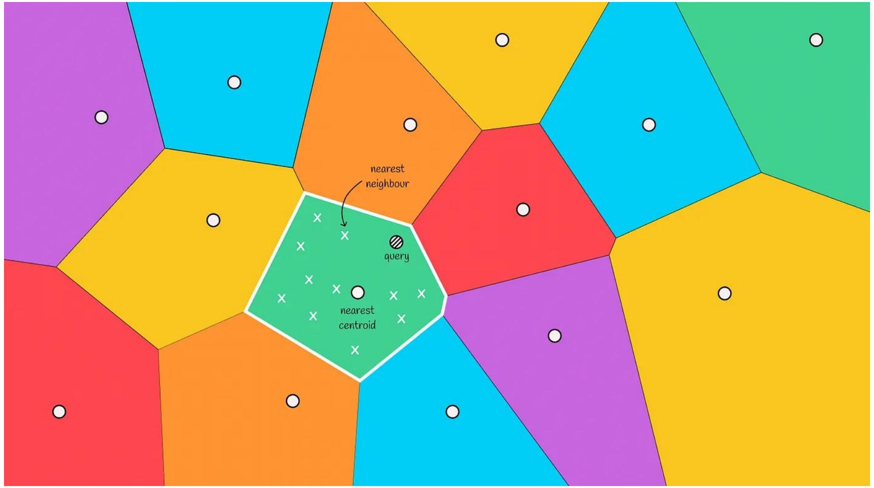
- ❖ Vectors simply laid out in space
- ❖ Perform simple K-nearest neighbors search
- ❖ Search duration grows linearly, but provides accurate results



Source: https://www.pinecone.io/learn/series/faiss/vector-indexes/

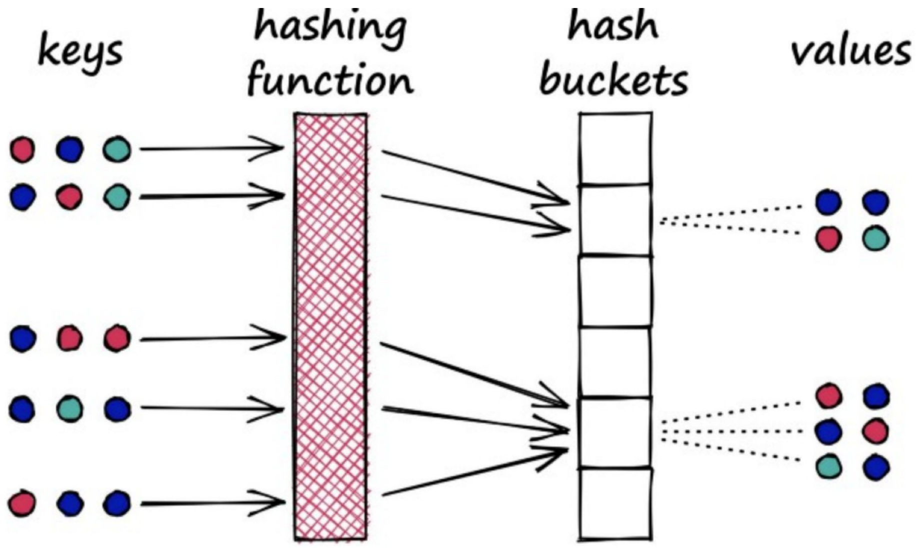
Inverted File Index (IVF)

- ❖ Cluster similar vectors using K-Means
- ❖ Find the centroid nearest to the query vector, then zoom into the cluster, and search for K-nearest neighbors
- ❖ Parameters
  - ➢ `n_list`: Number of cluster to create
  - ➢ `n_probe`: Number of nearest clusters to search



Locality Sensitive Hashing (LSH)

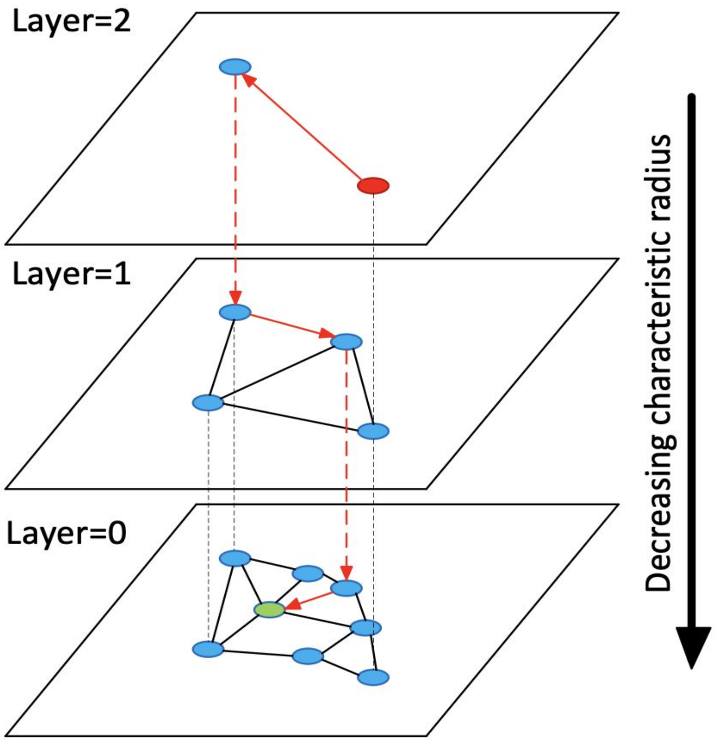
- ❖ Hash input vectors so that similar vectors land in the same bucket with high probability
- ❖ Query is hashed using the same hash function into the closest buckets within which KNN is performed
- ❖ Parameters
  - ➢ `nbits`: No. of bits used per stored vector (The number of hash buckets would be 2^{nbits})



Source: https://www.pinecone.io/learn/series/faiss/vector-indexes/

Hierarchical Navigable Small World (HNSW)

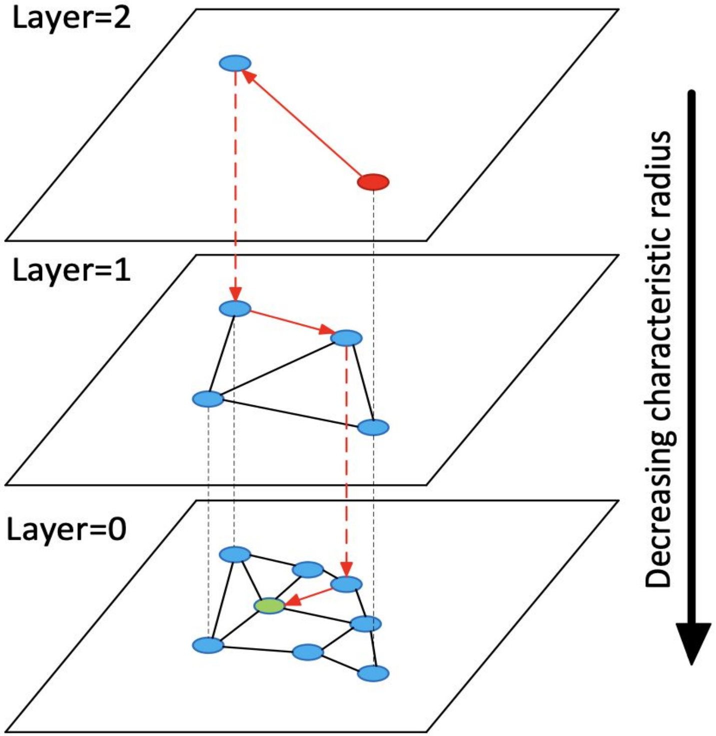
- Search
  - We start from the top layer by picking a node as the entrypoint
  - Compare its neighbors with the query vector and move the closest neighbor
  - Once we find a local minima, we move to the exact node in the next layer and start the search from there
  - The local minima that we find in the last layer is the nearest neighbor of our query vector
- Parameters
  - `ef_search`: The number of nearest neighbors searched for in every layer



Source: https://arxiv.org/abs/1603.09320

Hierarchical Navigable Small World (HNSW)

- Construction
  - Calculate a layer (L) for the incoming node using a probabilistic function
  - Assign the node to all the layers starting from L to 0
  - Run the search algorithm from the topmost layer, and connect the new node with its nearest neighbors
- Parameters
  - `ef_construction`: The number of nearest neighbors searched for in every layer
  - `M`: The maximum degree allowed for any node



Source: https://arxiv.org/abs/1603.09320

Benchmarking Vector Databases

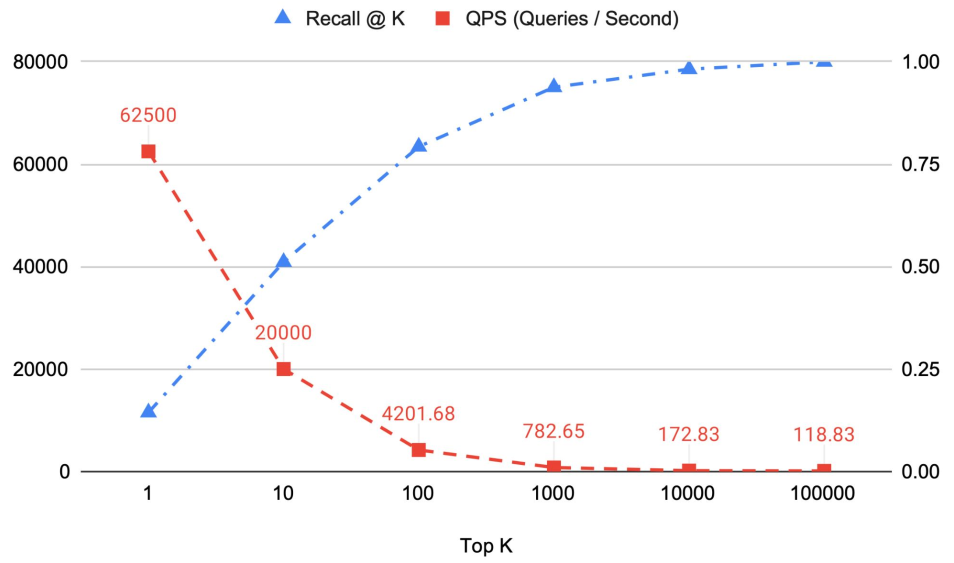
- Popular Datasets

Dataset	Dimensions	Train Set	Test Set	Distance Metric
DEEP1B	96	1B	10K	Cosine
SIFT 1M / 1B	128	1M / 1B	10K	Euclidean
GIST	960	1M	1K	Euclidean
GloVe	25	1.2M	10K	Cosine
dbpedia-openai	1536	1M	Need to split	Cosine

- Frameworks / Examples
  - ANN Benchmarks
  - Qdrant Vector DB Benchmarks
  - Ziliz VectorDBBench

Metrics for Comparison

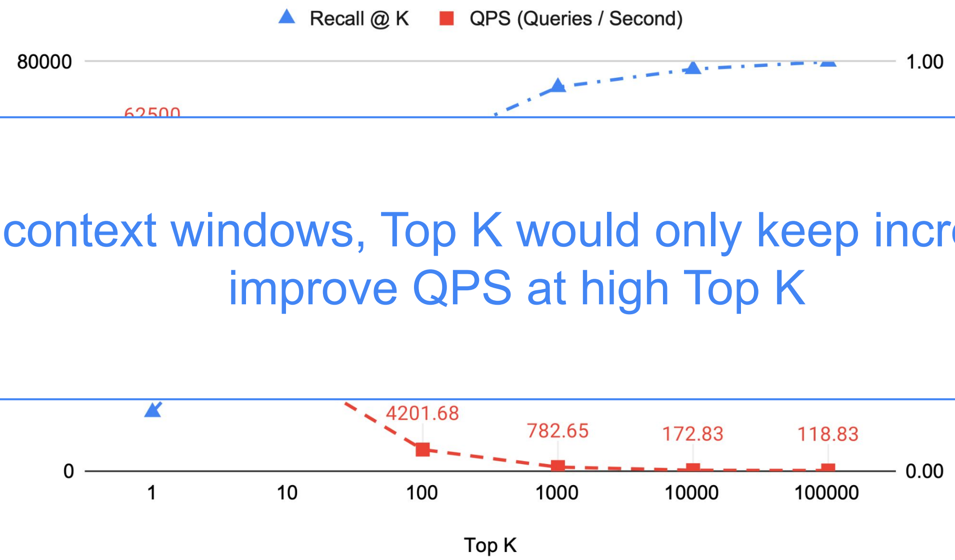
- Queries / Second (QPS)
- Recall @ K
  - # of *true* K-nearest neighbors / K



On a 100K learn / 1K query / GIST dataset / HNSWLIB

Metrics for Comparison

- Queries / Second (QPS)
- Recall @ K
  - # of *true* K-nearest neighbors / K



With bigger context windows, Top K would only keep increasing; Need to improve QPS at high Top K

On a 100K learn / 1K query / GIST dataset / HNSWLIB

Performance Comparison of Indexing Techniques

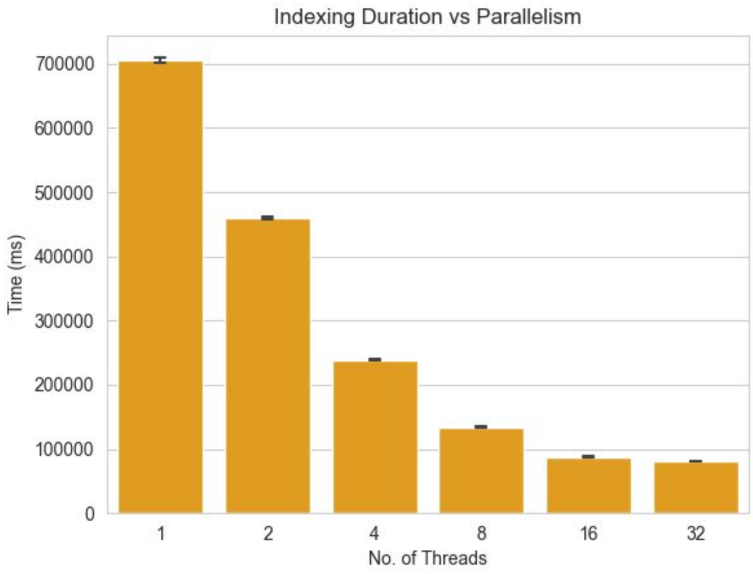
Index	Memory (MB)	Query Time (ms)	Recall @ 10
Flat (L2 or IP)	~500	~18	1.0
LSH	20 - 600	1.7 - 30	0.4 - 0.85
HNSW	600 - 1600	0.6 - 2.1	0.5 - 0.95
IVF	~520	1 - 9	0.7 - 0.95

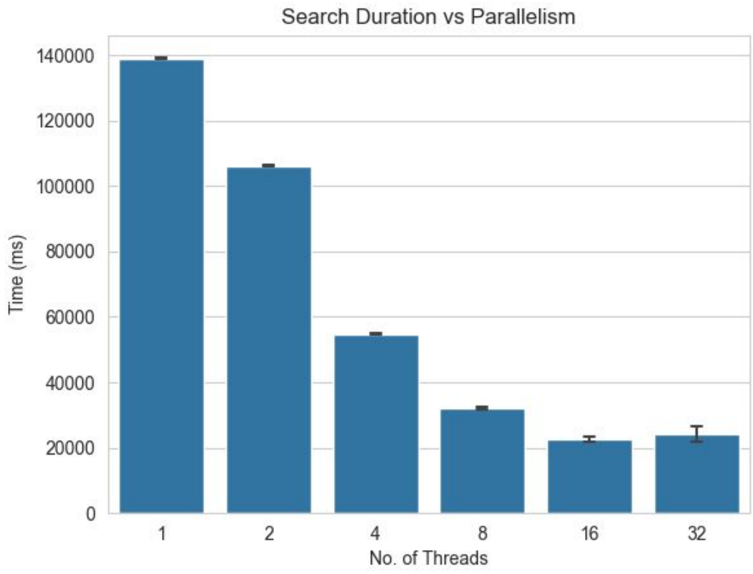
HNSW is the fastest algorithm with efficient memory usage and high recall

Source: https://www.pinecone.io/learn/series/faiss/vector-indexes/

SIFT-1M-128; K = 10

Effect Of Parallelism on HNSW

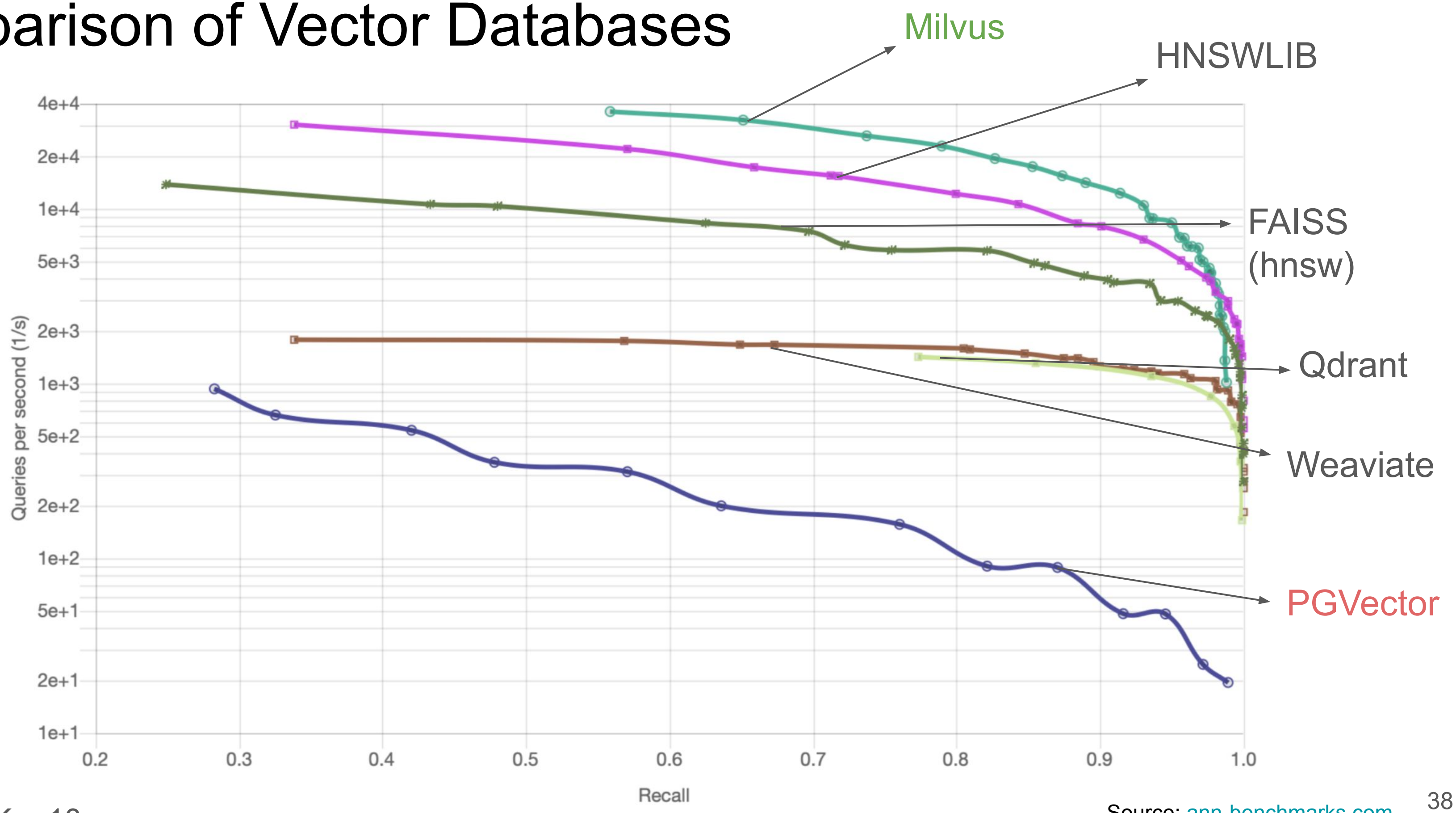




*Indexing and Search are **parallelizable** operations in HNSW*

GIST-1M-960 / HNSWLIB

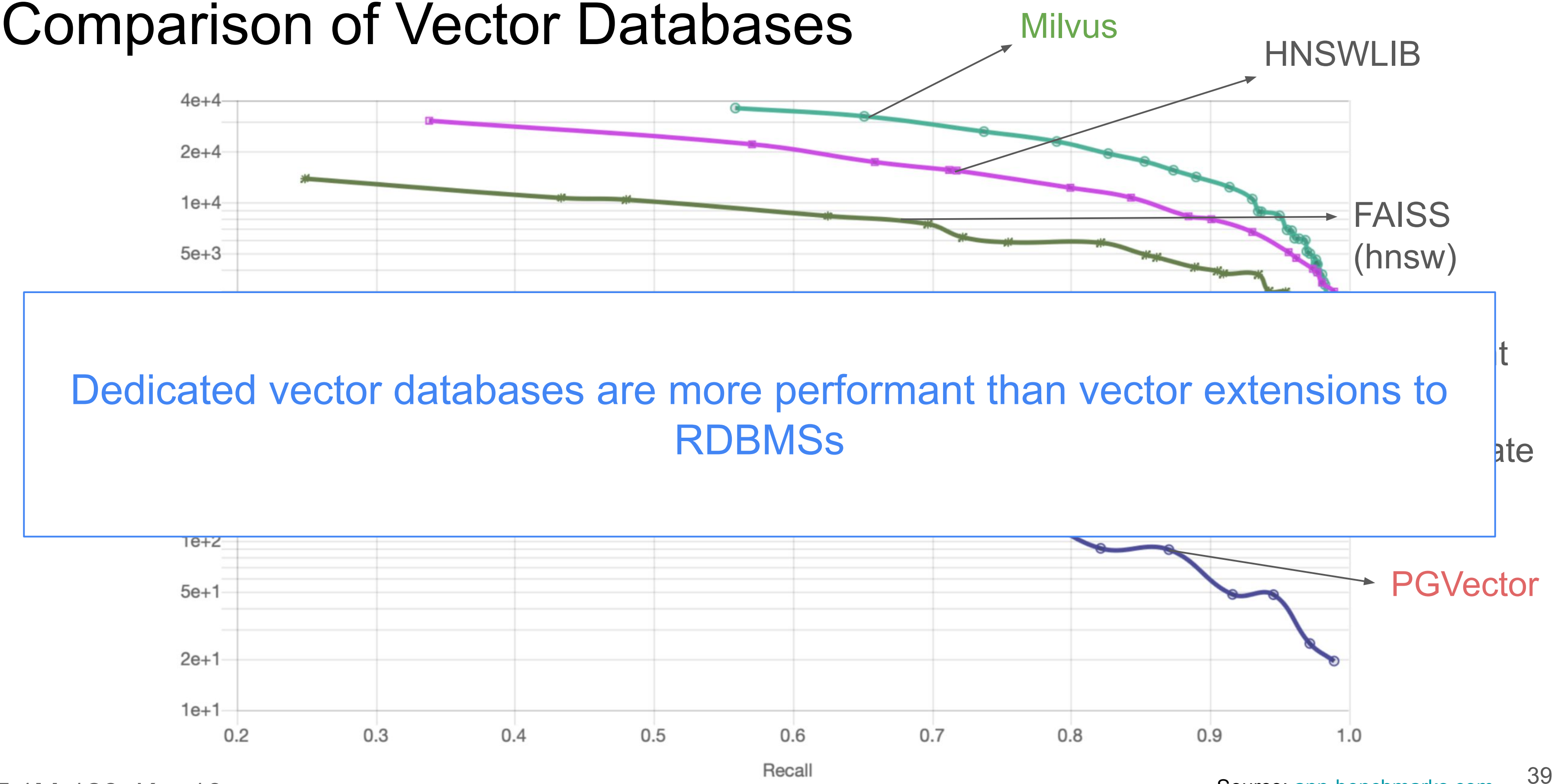
Comparison of Vector Databases



SIFT-1M-128; K = 10

Source: ann-benchmarks.com

Comparison of Vector Databases



SIFT-1M-128; K = 10

Source: ann-benchmarks.com

Profiles

(HNSWLIB, Qdrant, PGVector)

Machine Information

- Hardware
  - Processor
    - Dual Socket Intel Xeon Silver 4114 CPU @ 2.20 GHz
    - 10 Cores / Socket
    - Hyperthreading enabled
  - Cache
    - L1i / L1d: 640 KB
    - L2: 20 MB
    - L3: 27.5 MB
  - Memory
    - 192 GB DDR4
- Software
  - Intel VTune Profiler 2024.0.1
  - Perf 5.4.248

HNSWLIB: Performance Comparison

Dataset: GIST (960)

Train / Test Split: 100K / 1K

No. of Threads: 40

	Flat	**HNSW** (ef_cons = 64, ef_search = 100, M = 32)
Index Size	367 MB	393 MB
Index Duration	0.239 s	7.198 s
Query Duration (Top K = 100)	1.757 s	0.247 s

HNSWLIB: Performance Comparison

Dataset: GIST (960)

Train / Test Split: 100K / 1K

No. of Threads: 40

Index calculation is a highly expensive operation as compared to searches in HNSW

	Flat	**HNSW** (ef_cons = 64, ef_search = 100, M = 32)
Index Size	367 MB	393 MB
Index Duration	0.239 s	7.198 s
Query Duration (Top K = 100)	1.757 s	0.247 s

HNSWLIB: CPU Hotspot Analysis

Flat

```
91.76%  profile_hnswlib  profile_hnswlib     [.] hnswlib::L2SqrSIMD16ExtSSE
 3.36%  profile_hnswlib  libgomp.so.1.0.0    [.] omp_get_num_procs
 0.90%  swapper          [kernel.kallsyms]   [k] intel_idle
 0.79%  profile_hnswlib  [kernel.kallsyms]   [k] copy_user_enhanced_fast_string
 0.48%  profile_hnswlib  libc-2.31.so        [.] __memcpy_avx_unaligned_erms
 0.27%  profile_hnswlib  [kernel.kallsyms]   [k] _raw_spin_lock
 0.23%  profile_hnswlib  profile_hnswlib     [.] hnswlib::BruteforceSearch<float>::searchKnn
```

HNSW

```
45.59%  profile_hnswlib  profile_hnswlib      [.] hnswlib::L2SqrSIMD16ExtSSE
11.74%  profile_hnswlib  libgomp.so.1.0.0     [.] omp_get_num_procs
 5.93%  swapper          [kernel.kallsyms]    [k] intel_idle
 5.81%  profile_hnswlib  [kernel.kallsyms]    [k] copy_user_enhanced_fast_string
 4.82%  profile_hnswlib  profile_hnswlib      [.] hnswlib::HierarchicalNSW<float>::searchBas
 3.13%  profile_hnswlib  libc-2.31.so         [.] __memcpy_avx_unaligned_erms
 2.56%  swapper          [unknown]            [.] 0000000000000000
```

Distance calculations (L2) takes a major chunk of the execution time

HNSWLIB: Memory Access Analysis

Instruction	% of `L2SqrSIMD` in Flat	% of `L2SqrSIMD` in HNSW
`movups`	41.18	86.74
`addps`	43.43	7.27
`subps`	0.61	1.24
`mulps`	7.38	0.78

Distance calculation (L2) in HNSW is more memory bound than Flat due to aggressive prefetching

Qdrant: CPU Hotspot Analysis

Elapsed Time: 122.915s

CPU Time:	239.192s
Total Thread Count:	449
Paused Time:	0s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time	% of CPU Time
segment::spaces::simple_avx::dot_similarity_avx::he0656f57d193c9aa	qdrant	232.215s	97.1%
_LTsegment..vector_storage..raw_scorer..RawScorerImplLTTVector$C$TQueryScorer$GT$$u20asu20$segment..vector_storage..raw_scorer..RawScorerGT::peek_top_all::h7f7a0572eaed0ae9	qdrant	1.864s	0.8%
common::fixed_length_priority_queue::FixedLengthPriorityQueueLTTGT::push::h9555933f1b8054be	qdrant	0.808s	0.3%
epoll_wait	libc.so.6	0.760s	0.3%
_LTsegment..spaces..simple..DotProductMetric$u20$as$u20$segment..spaces..metric..MetricGT::similarity::h788ac744465c6b5a	qdrant	0.684s	0.3%
[Others]	N/A*	2.860s	1.2%

Flat

Elapsed Time: 122.332s

CPU Time:	87.370s
Total Thread Count:	200
Paused Time:	0s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time	% of CPU Time
segment::spaces::simple_avx::dot_similarity_avx::he0656f57d193c9aa	qdrant	46.463s	53.2%
epoll_wait	libc.so.6	4.458s	5.1%
segment::index::hnsw_index::hnsw::HNSWIndexLTTGraphLinksGT::search_with_graph::h5aad91e8607d78ab	qdrant	2.729s	3.1%
syscall	libc.so.6	2.607s	3.0%
write	libpthread.so.0	1.688s	1.9%
[Others]	N/A*	29.425s	33.7%

*N/A is applied to non-summable metrics

HNSW

Distance calculations (Dot Product) dominate the execution time

Qdrant: Memory Access Analysis

Function / Call Stack	CPU Time	Memory Bound	Loads	Stores	LLC Miss Count	Average Latency (cycles)	Module
segment::spaces::simple_avx::dot_similarity_avx::he06	217.755s	80.0%	112,072,553,108	10,047,542	16,394,674,965	120	qdrant
_LTsegment..vector_storage..raw_scorer..RawScorer	2.040s	82.9%	3,950,180,460	5,093,750,253	0	7	qdrant
smp_call_function_single	1.415s	37.9%	20,053,973	10,047,542	0	586	vmlinux
swapgs_restore_regs_and_return_to_usermode	1.040s	0.0%	70,250,572	10,006,431	0	8	vmlinux

Flat

Function / Call Stack	CPU Time	Memory Bound	Loads	Stores	LLC Miss Count	Average Latency (cycles)	Module
segment::spaces::simple_avx::dot_similarity_avx::he06	28.205s	84.1%	8,834,440,834	25,485,511	1,087,687,953	69	qdrant
segment::index::hnsw_index::hnsw::HNSWIndexLTT	2.670s	62.0%	314,894,531	138,670,884	0	31	qdrant
serde_json::de::DeserializerLTRGT::parse_decima	2.250s	21.6%	954,465,540	808,302,238	0	7	qdrant
serde::de::Deserializer::__deserialize_content::h9d9545	1.875s	0.0%	1,034,131,175	587,516,780	0	30	qdrant
regex::backtrack::BoundedLTIGT::backtrack::hf360	1.795s	2.0%	1,392,025,584	621,750,220	0	7	qdrant

HNSW

Distance calculations (Dot Product) are highly memory-bound

PGVector: Performance Comparison

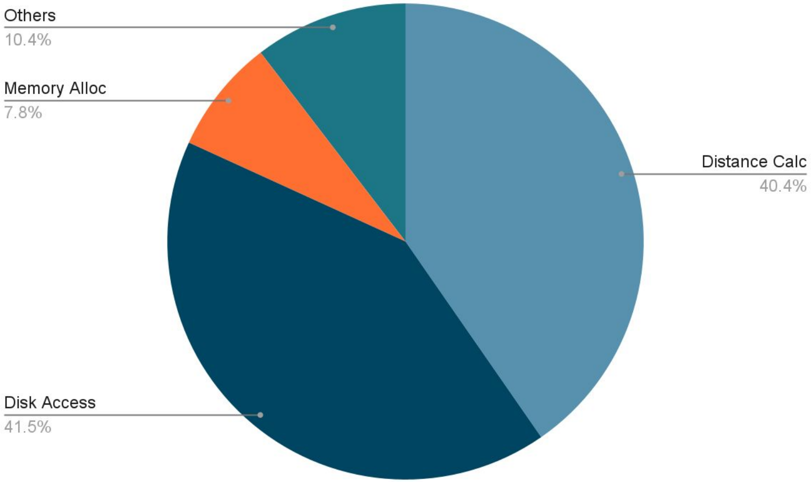
Dataset: dbpedia-entities-openai-1M (1536)

Train / Test Split: 100K / 1K

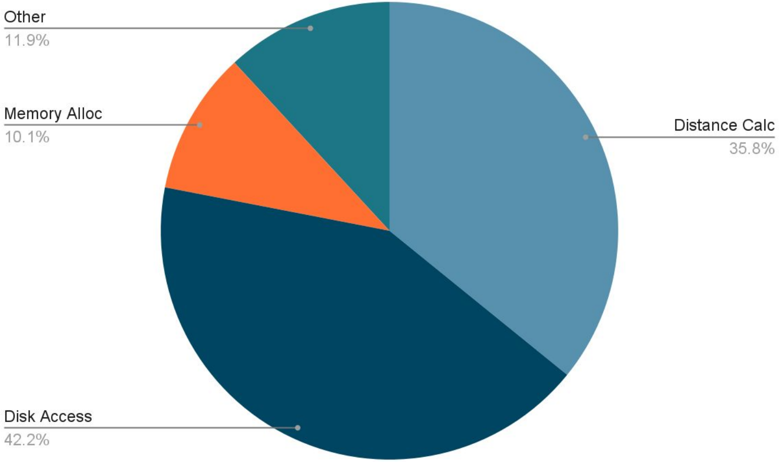
No. of Threads: 1

	IVF Flat (n_list = 100, n_probe = 10)	**HNSW** (ef_cons = 64, ef_search = 100, M = 32)
Index Size	782 MB	781 MB
Index Duration	27.50 s	150.24 s
Query Duration (Top K = 100)	147.63 s	89.82 s

PGVector: CPU Hotspot Analysis



IVF Flat



HNSW

After distance calculations, file reads comprise a significant chunk of the execution time

PGVector: Performance Comparison

Dataset: dbpedia-entities-openai-1M (1536)

Train / Test Split: 100K / 1K

No. of Threads: 1

Not much diff due to high overhead of disk access

	IVF Flat (n_list = 100, n_probe = 10)	**HNSW** (ef_cons = 64, ef_search = 100, M = 32)
Index Size	782 MB	781 MB
Index Duration	27.50 s	150.24 s
Query Duration (Top K = 100)	147.63 s	89.82 s

PGVector: Memory Access Analysis

```
Percent │   for (int i = 0; i < dim; i++)
        │     mov     $0x0,%eax
        │   float           distance = 0.0;
  0.06  │     vxorps  %xmm1,%xmm1,%xmm1
        │   for (int i = 0; i < dim; i++)
        │9:   cmp     %edi,%eax
  0.07  │   ↓ jge     26
        │   {
        │   float           diff = ax[i] - bx[i];
 25.44  │     movslq  %eax,%rcx
  0.74  │     vmovss  (%rsi,%rcx,4),%xmm0
  0.15  │     vsubss  (%rdx,%rcx,4),%xmm0,%xmm0
        │
        │   distance += diff * diff;
 23.49  │     vmulss  %xmm0,%xmm0,%xmm0
 49.96  │     vaddss  %xmm0,%xmm1,%xmm1
        │   for (int i = 0; i < dim; i++)
        │     inc     %eax
        │   ↑ jmp     9
        │   }
        │
        │   return distance;
        │   }
        │26:  vmovaps %xmm1,%xmm0
  0.08  │   ← retq
```

IVF Flat

compute ops dominate

```
Percent │   for (int i = 0; i < dim; i++)
  0.21  │     mov     $0x0,%eax
        │   float           distance = 0.0;
        │     vxorps  %xmm1,%xmm1,%xmm1
        │   for (int i = 0; i < dim; i++)
        │9:   cmp     %edi,%eax
        │   ↓ jge     26
        │   {
        │   float           diff = ax[i] - bx[i];
 26.24  │     movslq  %eax,%rcx
        │     vmovss  (%rsi,%rcx,4),%xmm0
        │     vsubss  (%rdx,%rcx,4),%xmm0,%xmm0
        │
        │   distance += diff * diff;
 27.25  │     vmulss  %xmm0,%xmm0,%xmm0
 46.12  │     vaddss  %xmm0,%xmm1,%xmm1
        │   for (int i = 0; i < dim; i++)
  0.18  │     inc     %eax
        │   ↑ jmp     9
        │   }
        │
        │   return distance;
        │   }
        │26:  vmovaps %xmm1,%xmm0
        │   ← retq
```

HNSW

PGVector: Memory Access Analysis

```
Percent │  for (int i = 0; i < dim; i++)
        │    mov     $0x0,%eax
        │  float             distance = 0.0;
   0.06 │    vxorps  %xmm1,%xmm1,%xmm1
        │  for (int i = 0; i < dim; i++)
        │ 9:  cmp     %edi,%eax
   0.07 │  ↓ jge     26
        │  {
        │  float             diff = ax[i] - bx[i];
  25.44 │    movslq  %eax,%rcx
   0.74 │    vmovss  (%rsi,%rcx,4),%xmm0
   0.15 │    vsubss  (%rdx,%rcx,4),%xmm0,%xmm0
        │
        │  distance += diff * diff;
  23.49 │    vmulss  %xmm0,%xmm0,%xmm0
  49.96 │    vaddss  %xmm0,%xmm1,%xmm1
        │  for (int i = 0; i < dim; i++)
        │    inc     %eax
```

```
Percent │  for (int i = 0; i < dim; i++)
   0.21 │    mov     $0x0,%eax
        │  float             distance = 0.0;
        │    vxorps  %xmm1,%xmm1,%xmm1
        │  for (int i = 0; i < dim; i++)
        │ 9:  cmp     %edi,%eax
        │  ↓ jge     26
        │  {
        │  float             diff = ax[i] - bx[i];
  26.24 │    movslq  %eax,%rcx
        │    vmovss  (%rsi,%rcx,4),%xmm0
        │    vsubss  (%rdx,%rcx,4),%xmm0,%xmm0
        │
        │  distance += diff * diff;
  27.25 │    vmulss  %xmm0,%xmm0,%xmm0
  46.12 │    vaddss  %xmm0,%xmm1,%xmm1
        │  for (int i = 0; i < dim; i++)
   0.18 │    inc     %eax
```

CPU is waiting for disk I/O, and can't issue memory load instructions, so its compute bound

IVF Flat   HNSW

Future Work

- Perform billion-scale experiments / profiles (Please provide me hardware !)
- Study GPU-based indexes and their performance characteristics
- Explore offloading distance calculations to specialized accelerators
- Leverage far memory to store larger-than-memory indexes
- Explore offloading vector search to computational storage devices

Thank You

Questions ?

jayjeetc@ucsc.edu
jayjeetc.github.io