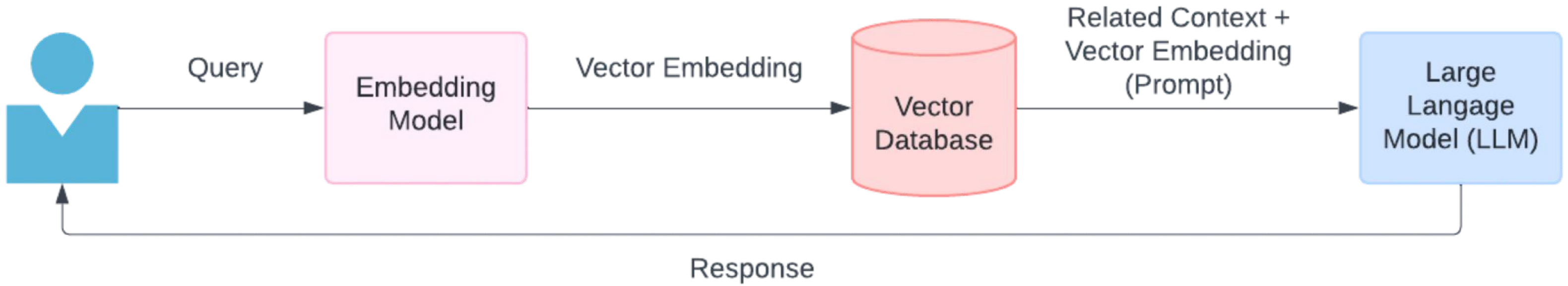
Towards Optimizing Search and Indexing in Vector Databases

Jayjeet Chakraborty, Heiner Litz
Center for Research in Systems and Storage
University of California, Santa Cruz

UC SANTA CRUZ
BaskinEngineering
Center for Research in Systems and Storage
NSF

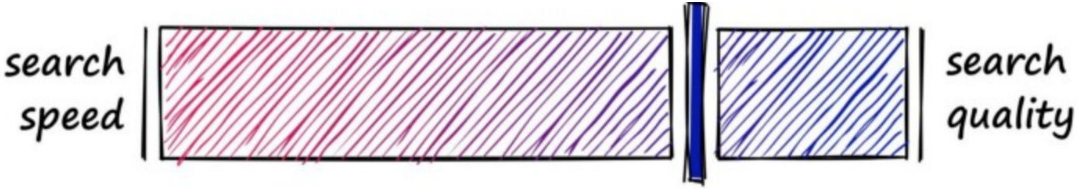
Motivation



- ❖ Vector searches sit on the critical path in RAG applications; Getting fast responses require searches to be highly performant
- ❖ Vector datasets often have billions/trillions of vectors; Getting high-performance at that scale is difficult
- ❖ Having an in-depth understanding of the performance characteristics of vector indexing and search algorithms is crucial !

Vector Search

- Finding tokens closely related to a query, requires performing a nearest neighbor search on a set of feature vectors
- Traditionally, KNN algorithms have been used for nearest neighbor searches,
  - But with billions/trillions of vectors, KNN search durations become unrealistic quickly
  - ANN (Approximate Nearest Neighbor) algorithms allows **faster searches by trading some accuracy** using a combination of techniques such as space-partitioning, indexing, and quantization



Vector Similarity Metrics

- Popular metrics used for calculating the distance between two vectors in the euclidean space include
  - Cosine Similarity
  - Dot Product
  - Manhattan Distance (L1)
  - **Euclidean Distance (L2)**

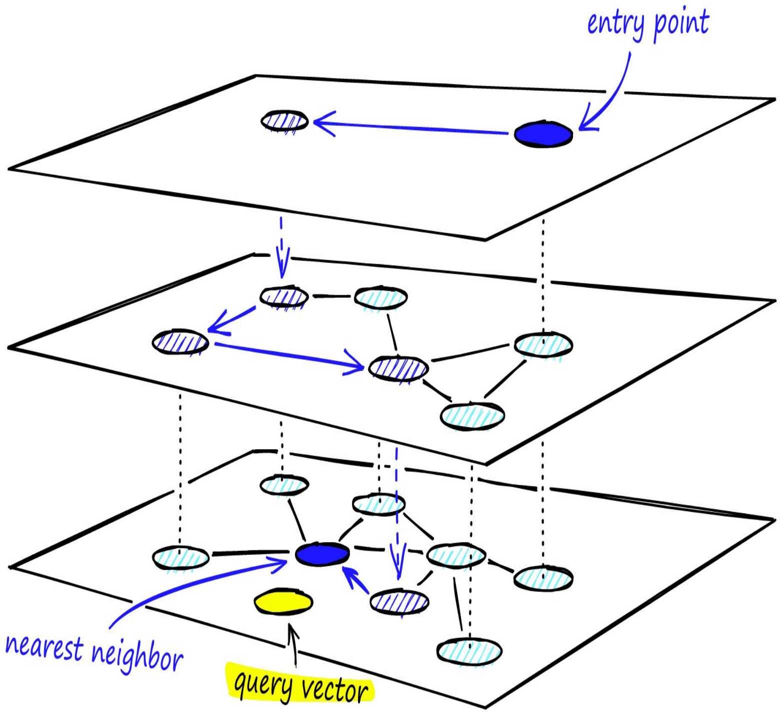
$$d(\mathbf{a}, \mathbf{b}) = \sqrt{(\mathbf{a_1} - \mathbf{b_1})^2 + (\mathbf{a_2} - \mathbf{b_2})^2 + \ldots + (\mathbf{a_n} - \mathbf{b_n})^2}$$

Vector Indexing Techniques

- **Given below are some widely-used vector indexing algorithms**
  - Flat
  - IVF or Inverted File Index (Clustering-based)
  - **HNSW or Hierarchical Navigable Small World** (Graph-based)
  - LSH or Locality Sensitive Hashing (Hashing-based)
  - Examples of other indexing algorithms include
    - Microsoft DiskANN
    - Google ScaNN
    - Spotify ANNOY

Hierarchical Navigable Small World

CRSS



- ❖ We start from layer 0,
  - ➢ pick a entry node
  - ➢ compare its neighbors with the query vector
  - ➢ move to the neighbor closest to the query vector
  - ➢ Once we find a local minima, we move to that exact node in the next layer and start the search again
  - ➢ The local minima that we find in the last layer is the closest one to our query vector

Vector Data Management Systems

- **Some widely-used vector data management systems include**
  - Client-Server
    - Milvus, Qdrant, Weavite
  - Embedded
    - LanceDB, Deeplake, Chroma
  - Extensions
    - PgVector
  - Libraries
    - **FAISS (Meta), hnswlib,** usearch

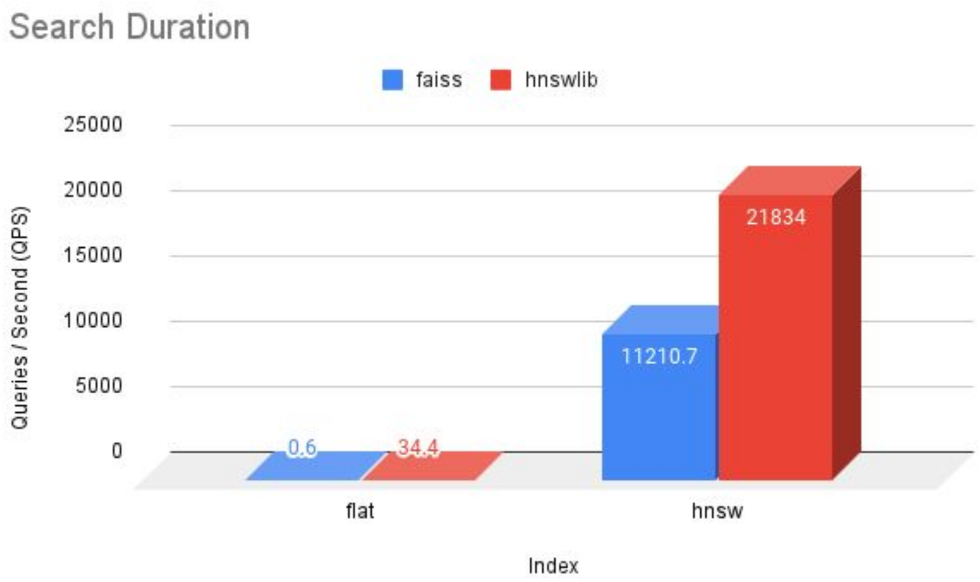
Machine Information

- **CPU:**
  - Intel(R) Xeon(R) Silver 4114 @ 2.20GHz
    - 2 Sockets
    - 10 Cores / Socket
    - Hyperthreading enabled
    - **L2:** 20 MB, **L3:** 27.5 MB
- **Memory:**
  - 192 GB ECC DDR4

Benchmark and Profile Setup

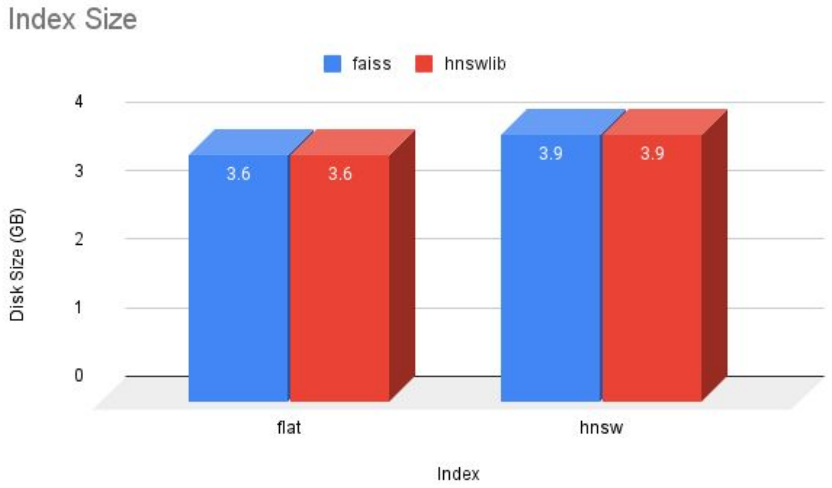
- **Dataset**
  - GIST dataset (http://corpus-texmex.irisa.fr/)
  - **Learn Set:** 1,000,000 x 960 / 3.84 GB
  - **Query Set:** 500,000 x 960 / 1.92 GB
- **Indexes**
  - Flat
  - HNSW (**M = 32** / **efSearch = 16** / **efConstruction = 40**)
- **TopK**
  - 10

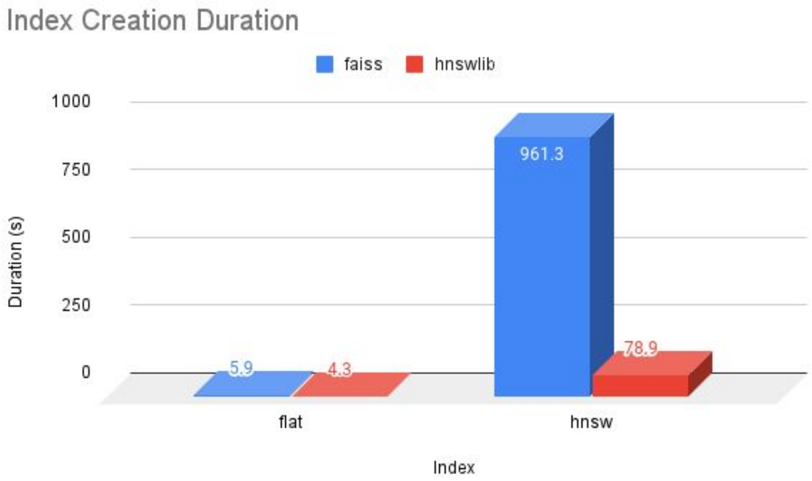
FAISS vs hnswlib



hnswlib is faster at vector searches than FAISS

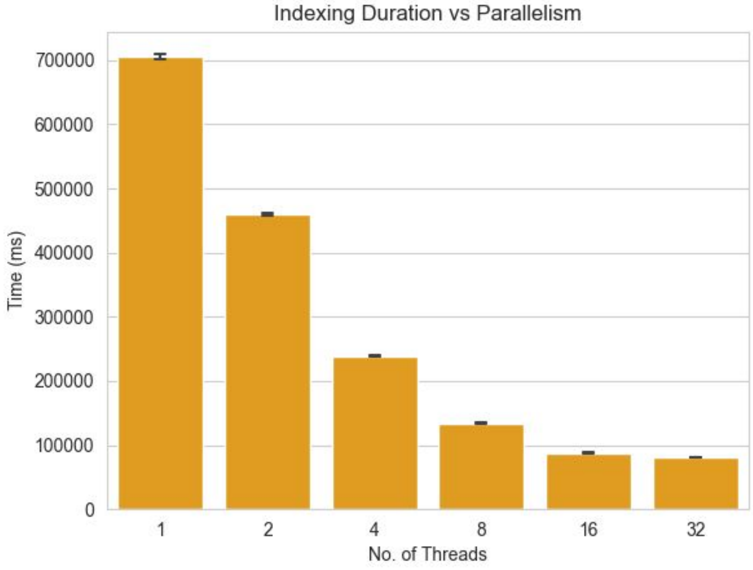
FAISS vs hnswlib

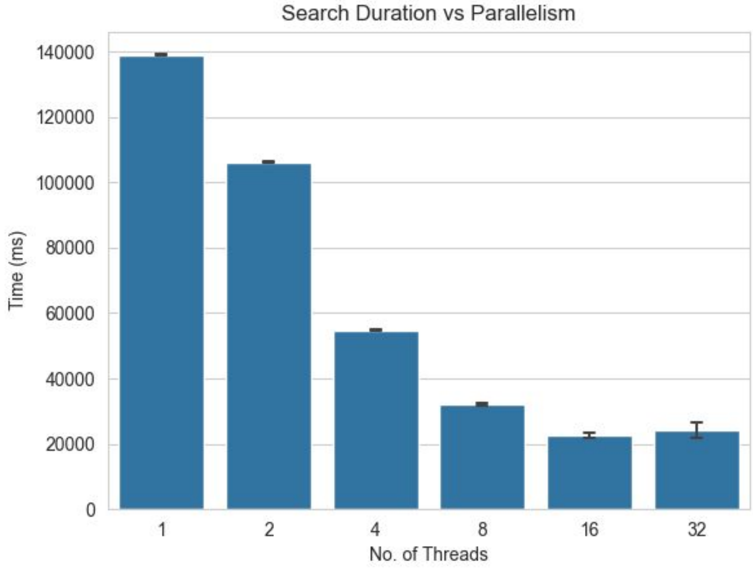




Although index sizes are similar, FAISS is slower in indexing than hnswlib

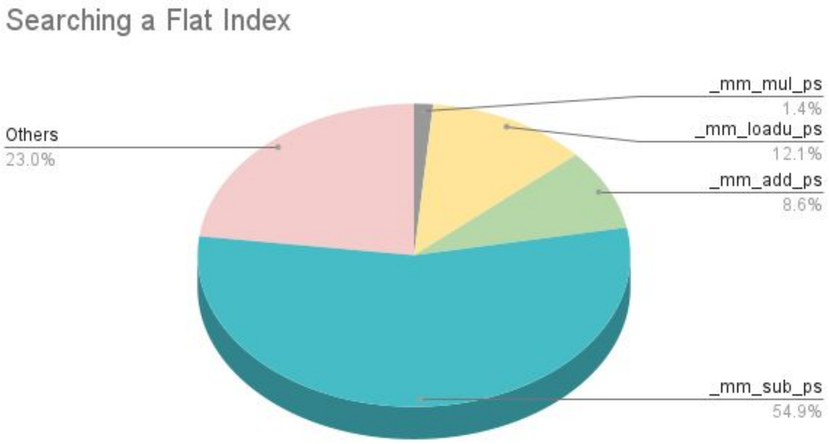
Effect of parallelism in hnswlib

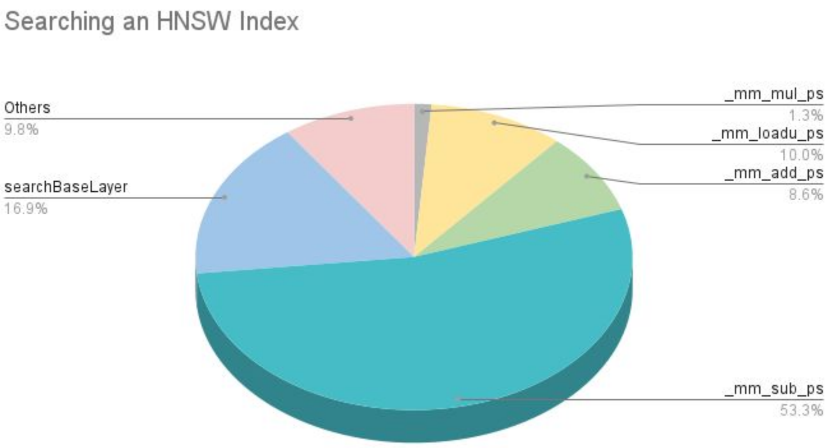




Index and Search operations can scale well with increasing parallelism, hence opportunities for using GPUs, FPGAs

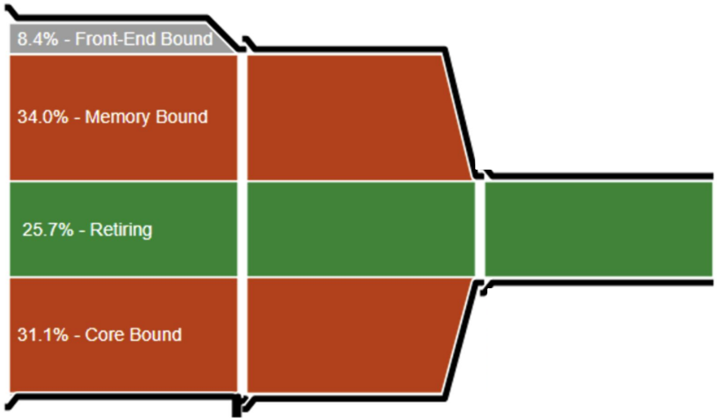
Profiling hnswlib searches: **CPU Hotspots**



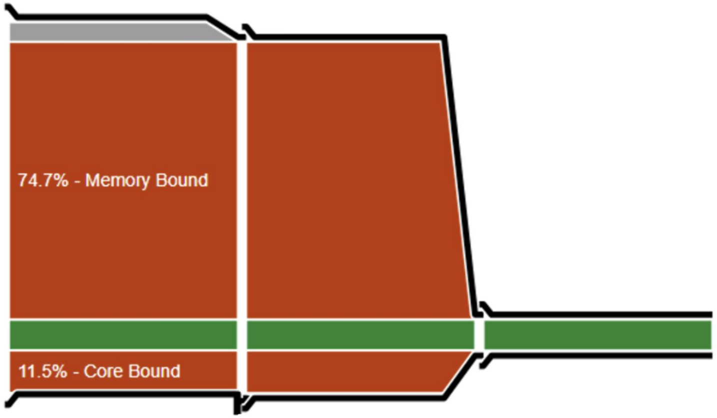


Distance calculation operations (subtractions in this case) dominate the total search duration

Profiling hnswlib searches: **Micro-Architecture**



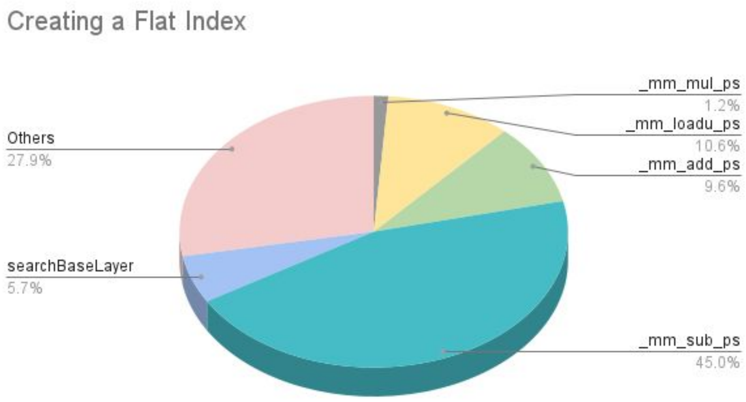
Searching a Flat Index



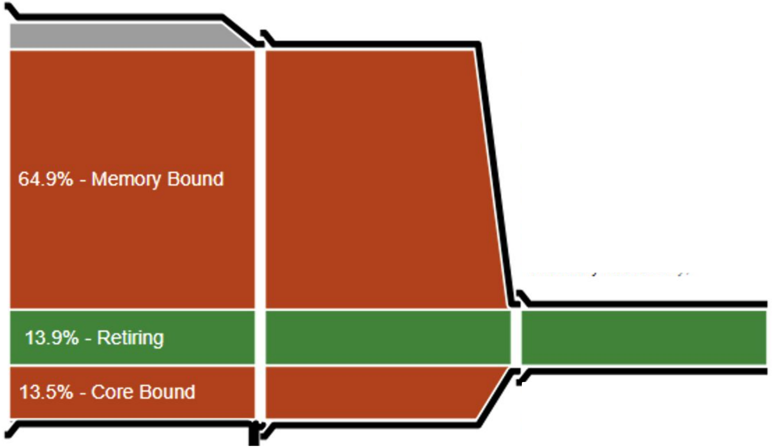
Searching a HNSW Index

Search operations are highly memory bandwidth bound, more so on an HNSW index than on a Flat index

Profiling hnswlib indexing



CPU Hotspots



Micro-Architecture

Index operations have similar performance characteristics to searches being memory bandwidth bound and dominated by distance calculations

Conclusion and Next Steps

- **Conclusion**
  - Vector search libraries are still not in their most performant state and needs optimizations
  - Understanding the performance characteristics of vector indexing/searches is necessary to find opportunities for acceleration using modern hardware
  - Vector indexing/search operations are **memory bandwidth bound** and the query duration is dominated by **distance calculations**
- **Next Steps**
  - Study GPU-based indexes, for example NVIDIA Cagra
  - Perform billion-scale experiments (for example, with the BigANN dataset)
  - Explore using GPUs, specialized accelerators, and CXL memory expanders for billion-scale searches
  - Look at end-to-end performance characteristics by profiling databases besides libraries

Thank You

Questions?

jayjeetc@ucsc.edu

jayjeetc.github.io

https://github.com/JayjeetAtGithub