Empowering RoseDB With Vector Index

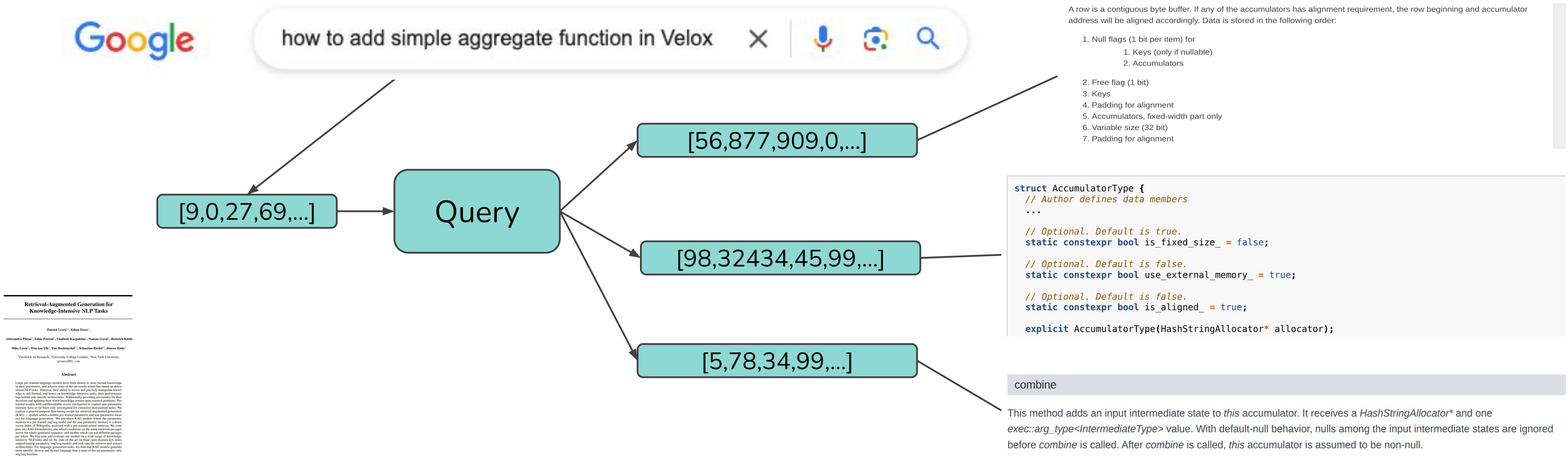
Hanzhi Wu, Real Chen, Zhubo Zhou

An efficient vector index, when building on top of an existing database, can be very EASY

Background

🔥 Large Language Models (**LLM**) use **RAG** to enhance the text generation process

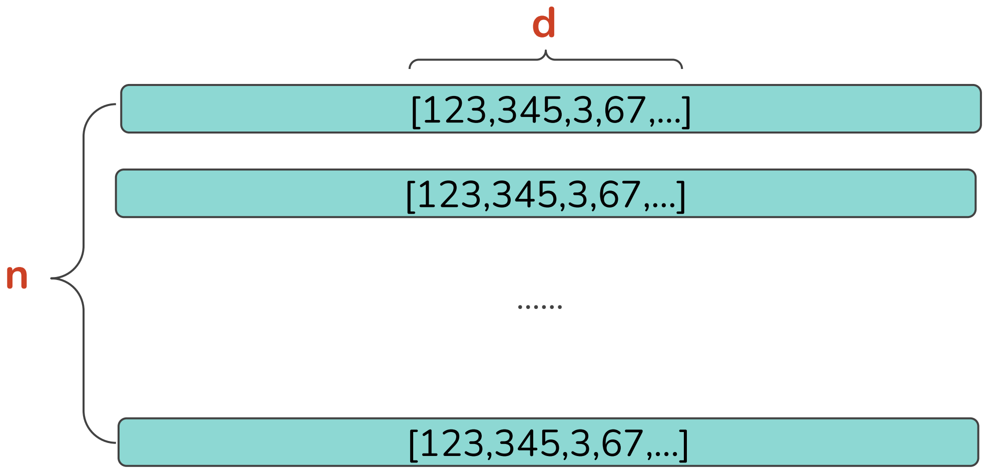
- Involves searching (from an external database) for **closest matching vector(s)** to the query vector produced by the LLM



Background

However, many existing key-value stores lack the ability to **query nearest neighbors** of a given vector efficiently

- Brute-force search: **O(n * d * log k)**



for every query, need to construct a priority queue using distances between the given vector and all vectors in the database

Solution: Vector Index

- **Fast retrieval** of **nearest neighbors** of a given vector within a large database

- Highly **scalable** with the increase of vector dimensions

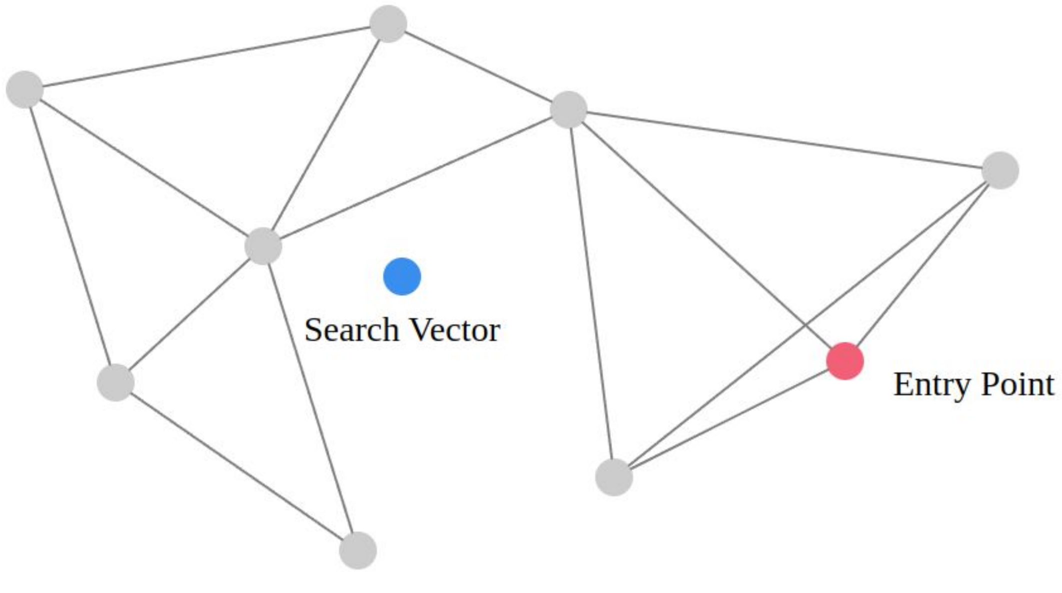
- LLM only needs answer that is good enough - Approximately closest vectors would be fine

NSW (Navigable Small Worlds) Index

GetVectors: Which is basically Dijkstra

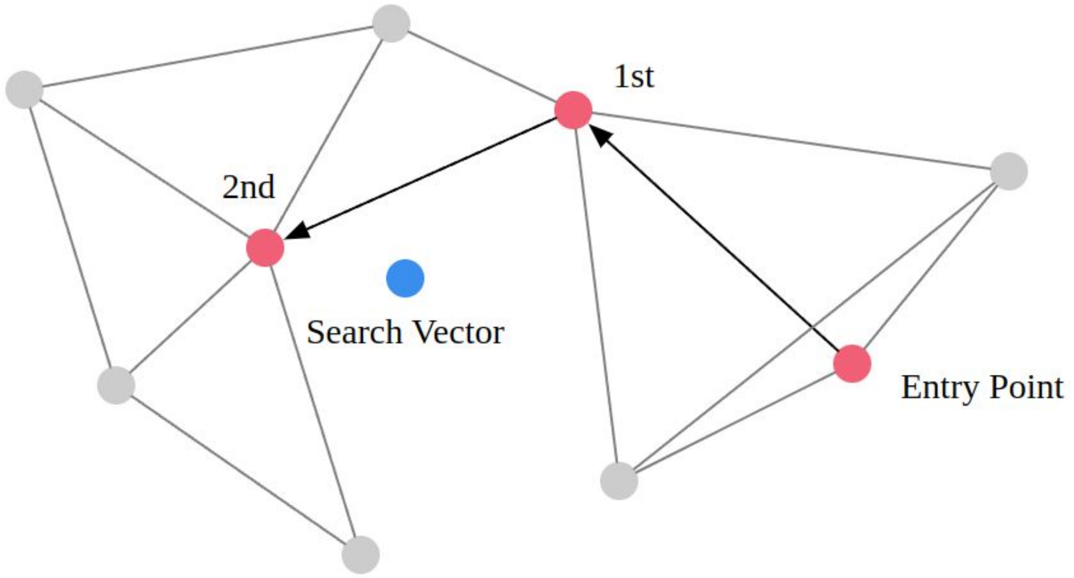
```
C <- entry_points as min heap on distance
W <- entry_points as max heap on distance
visited <- entry_points as unordered set
while not C.empty():
    node <- pop C (nearest element in C)
    if dist(node) > dist(top W): # top W is the furthest element in W
        break
    for neighbor in neighbors of node:
        if not visited neighbor:
            visited += neighbor
            C += neighbor
            W += neighbor
            retain k-nearest elements in W
return W
```

NSW (Navigable Small Worlds) Index



Picture from https://skyzh.github.io/write-you-a-vector-db/cpp-06-01-nsw.html

NSW (Navigable Small Worlds) Index



Picture from https://skyzh.github.io/write-you-a-vector-db/cpp-06-01-nsw.html

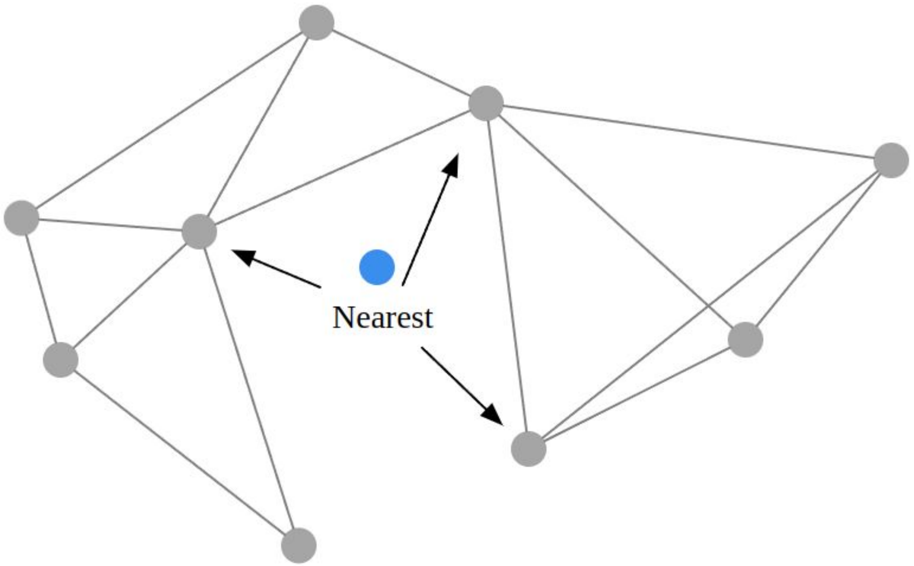
NSW (Navigable Small Worlds) Index

Put / Delete Vector: which is also EASY

Picture from https://skyzh.github.io/write-you-a-vector-db/cpp-06-01-nsw.html

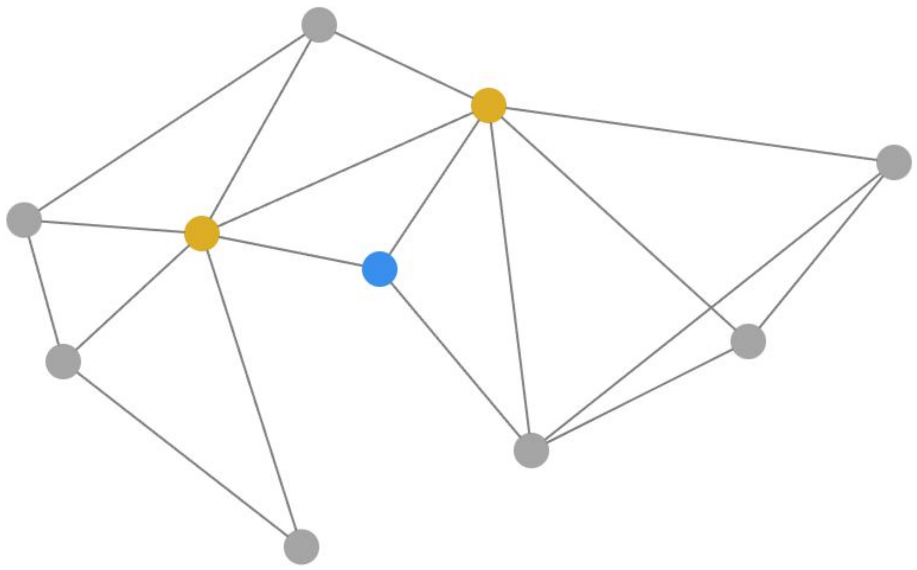
NSW Index: Put Vector

Get **m** nearest neighbors



Picture from https://skyzh.github.io/write-you-a-vector-db/cpp-06-01-nsw.html

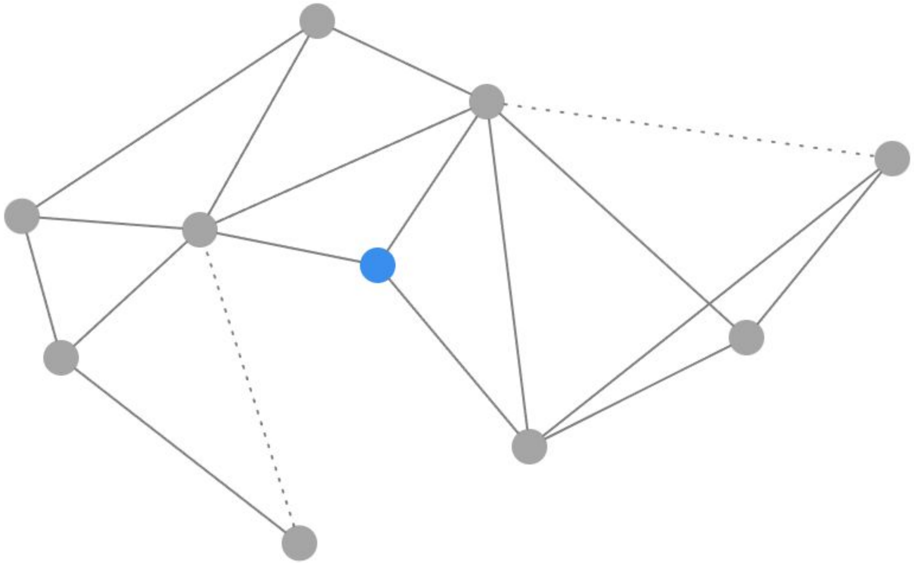
NSW Index: Put Vector



Picture from https://skyzh.github.io/write-you-a-vector-db/cpp-06-01-nsw.html

NSW Index: Put Vector

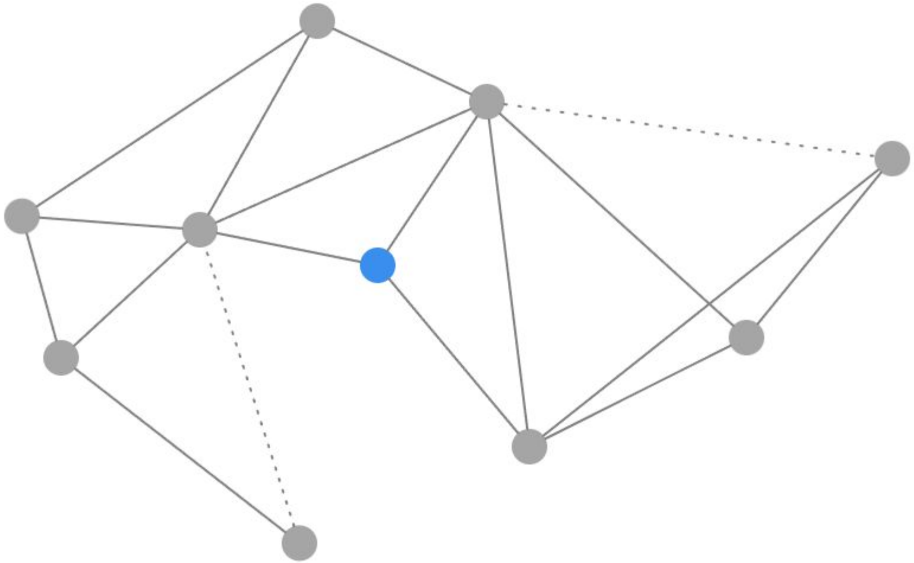
But graph might get too complicated ...
So cut off some edges according to **max_m**



Picture from https://skyzh.github.io/write-you-a-vector-db/cpp-06-01-nsw.html

NSW Index: Delete Vector

Delete is simply reverse the put operations



Picture from https://skyzh.github.io/write-you-a-vector-db/cpp-06-01-nsw.html

NSW Index: Parameters

Two parameters that can be configured by users

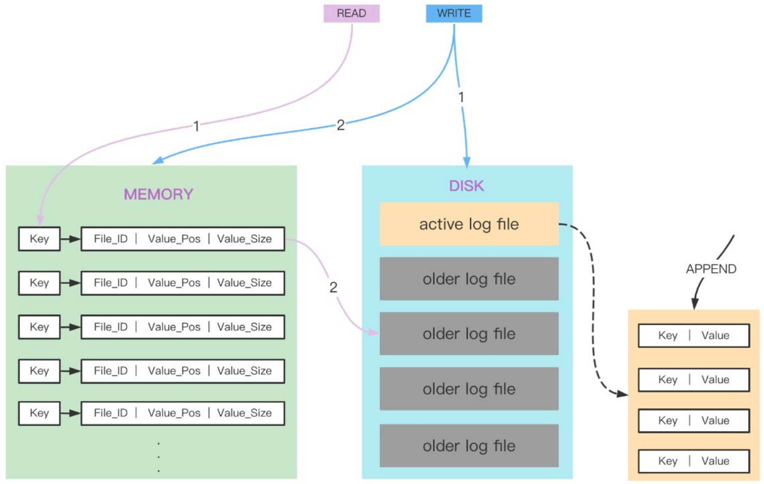
m and **max_m**

Picture from https://skyzh.github.io/write-you-a-vector-db/cpp-06-01-nsw.html

Rose DB

Integrate vector indexes to existing databases RoseDB

- A bitcask based lightweight key-value database
- Can be embedded to many existing systems



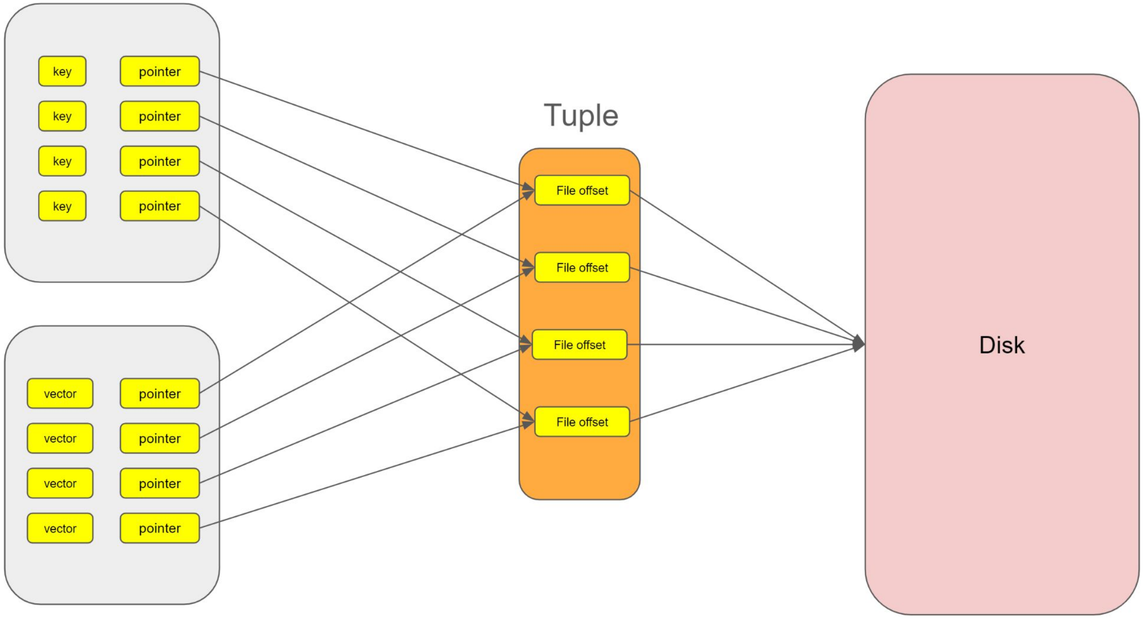
Observations From NSW Index

- Only support three operations
  - Get: get vectors “mathematically” close to the target
  - Put
  - Delete
- There is no:
  - Exact Get
  - Iterator

Picture from https://skyzh.github.io/write-you-a-vector-db/cpp-06-01-nsw.html

Integration

B+ tree index



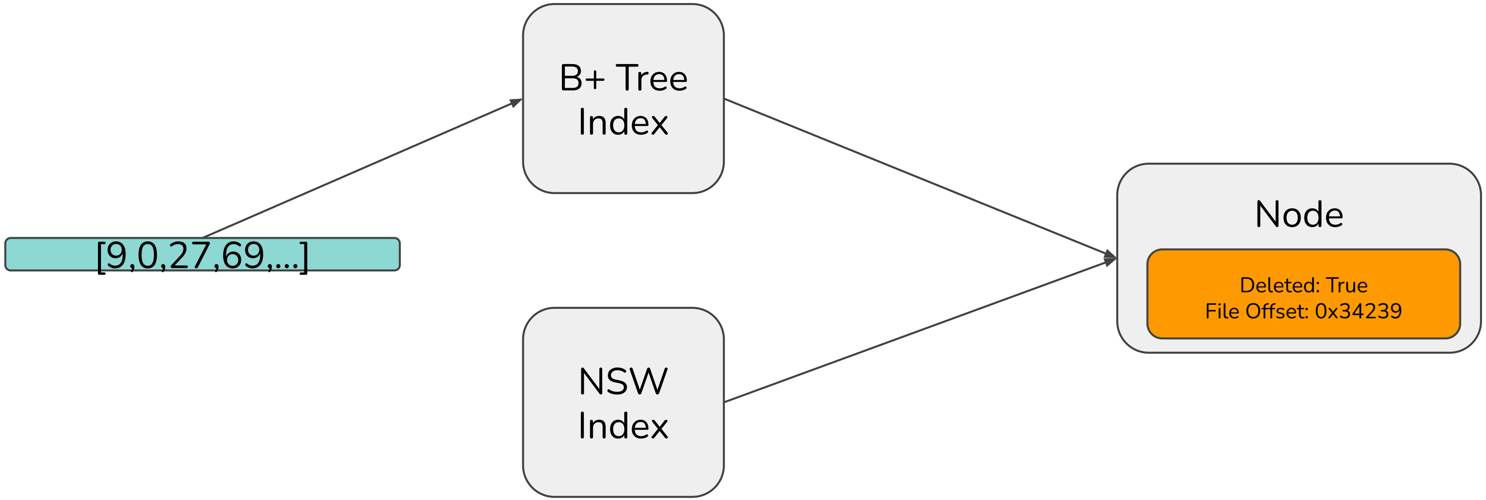
NSW index

Optimization - Lazy Delete

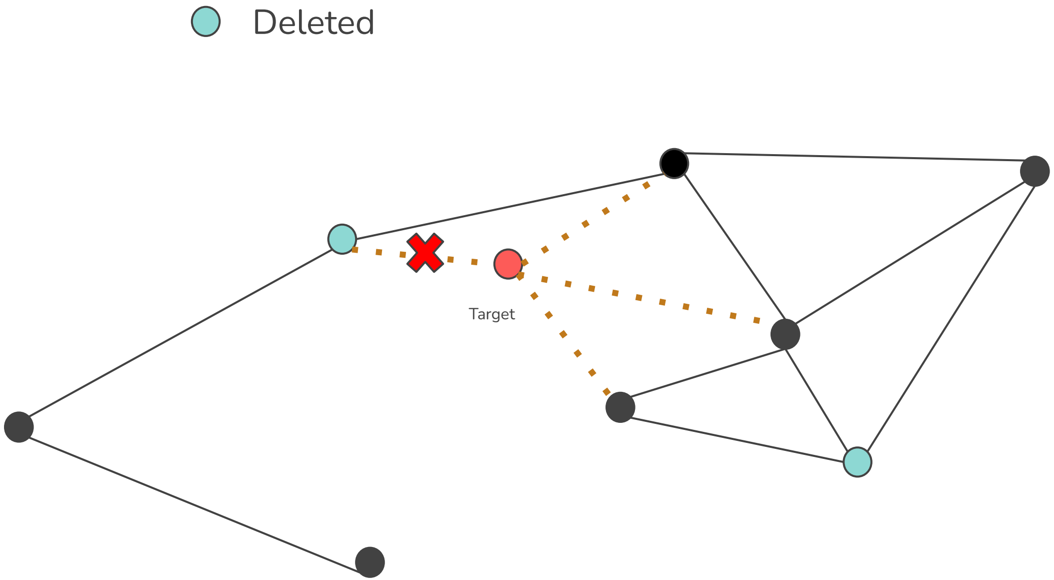
- Observation from NSW index
  - Delete operations are super expensive
  - Taking Exclusive Latch which impact the performance
- But Delete operations in B+ tree are super fast

Optimization - Lazy Delete

- Only delete vectors from B+ Tree
- Don’t delete the vector from graph right away

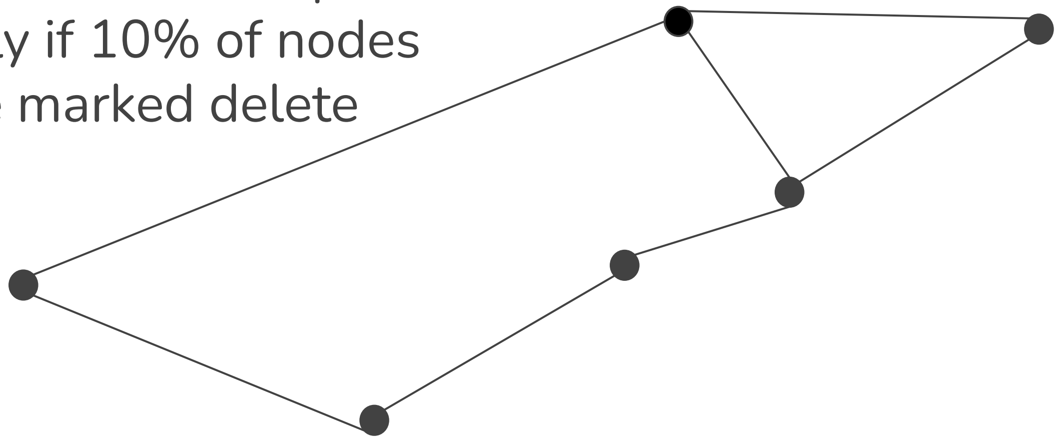


Optimization - Lazy Delete



Optimization - Lazy Delete

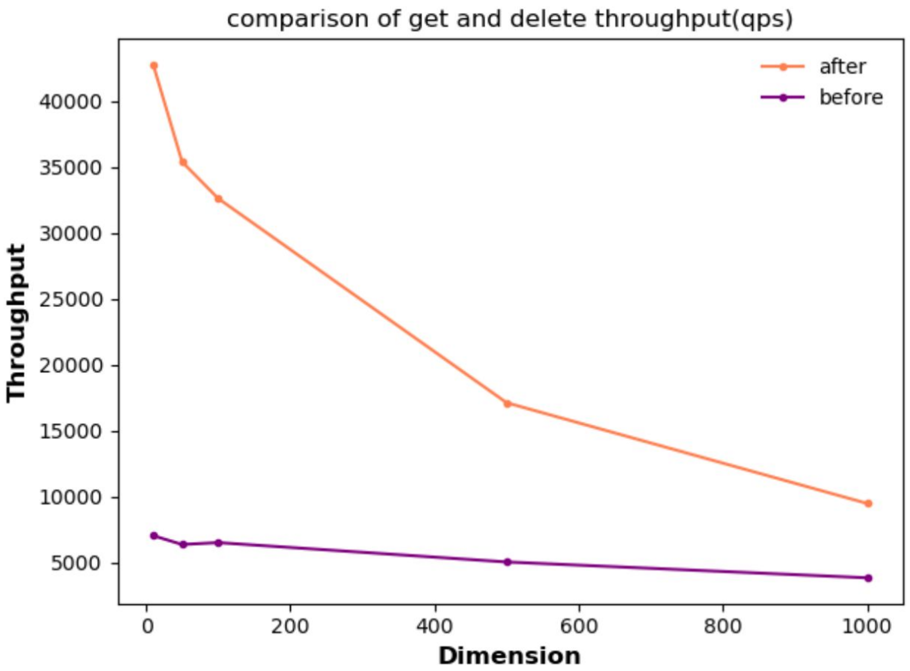
Reconstruct Graph only if 10% of nodes are marked delete



Optimization Evaluation: Setup

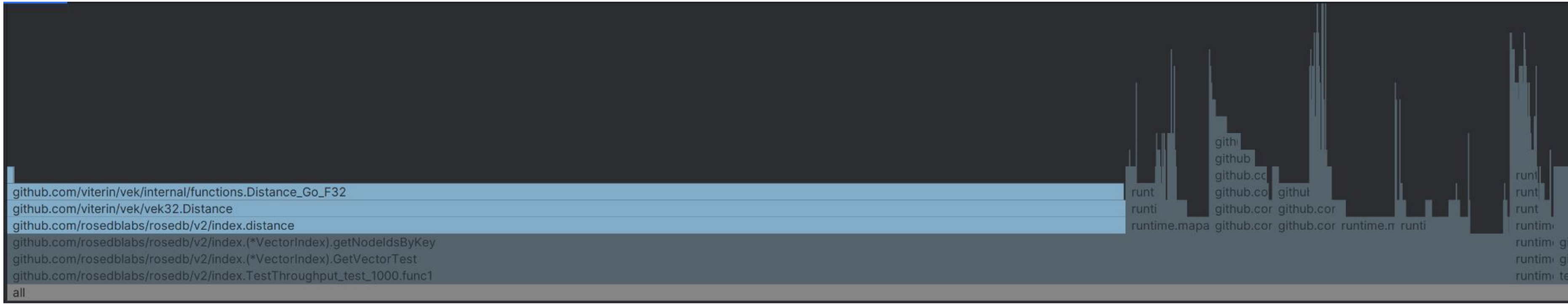
- Get 10000 vectors while 30% of total vectors is being deleted
    - All run **concurrently** with the following dimensions
        - 10
        - 50
        - 100
        - 500
        - 1000

Optimization Evaluation



Optimization - Vectorized(SIMD) Execution

- In NSW index
  - Tons of vector calculations: subtract, addition, pow, sqrt...
- SIMD execution can speed up calculations



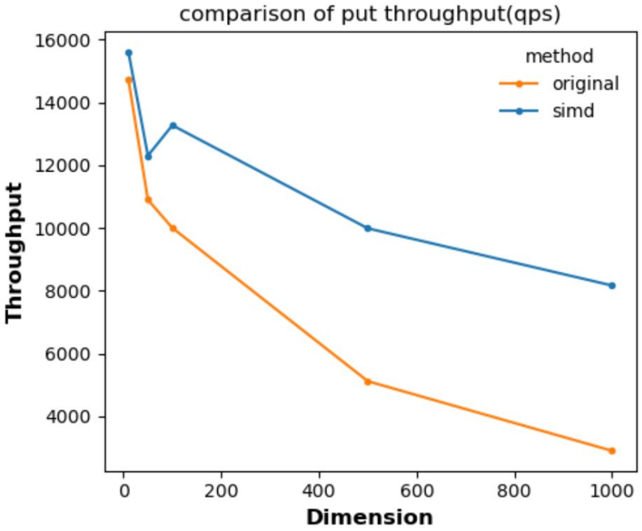
SIMD Evaluation: Environment Set Up

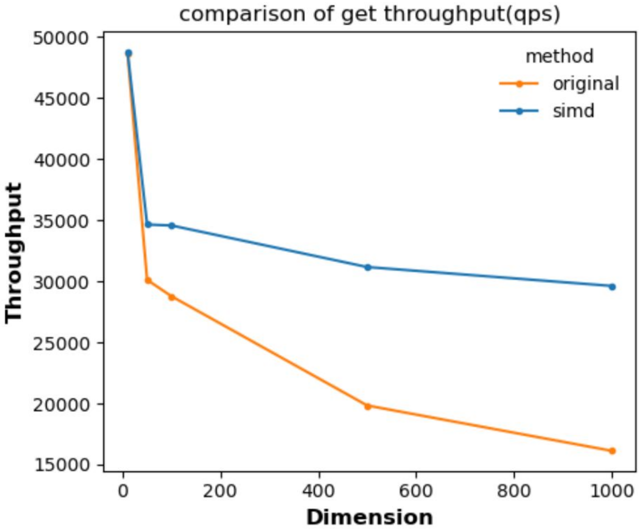
Arch: AMD 64 with the following SIMD register:
SSE2 SSE41 AVX2 BMI1 FMA SSSE3 SSE42 AES CX16 RDRAND AVX RDSEED SSE3 PCLMULQDQ ADX POPCNT ERMS BMI2 OSXSAVE

Test Set: get 200000 times from 10000 vectors with the following dimensions:

- 10
- 50
- 100
- 500
- 1000

Optimization - Vectorized(SIMD) Execution





Lessons Learned

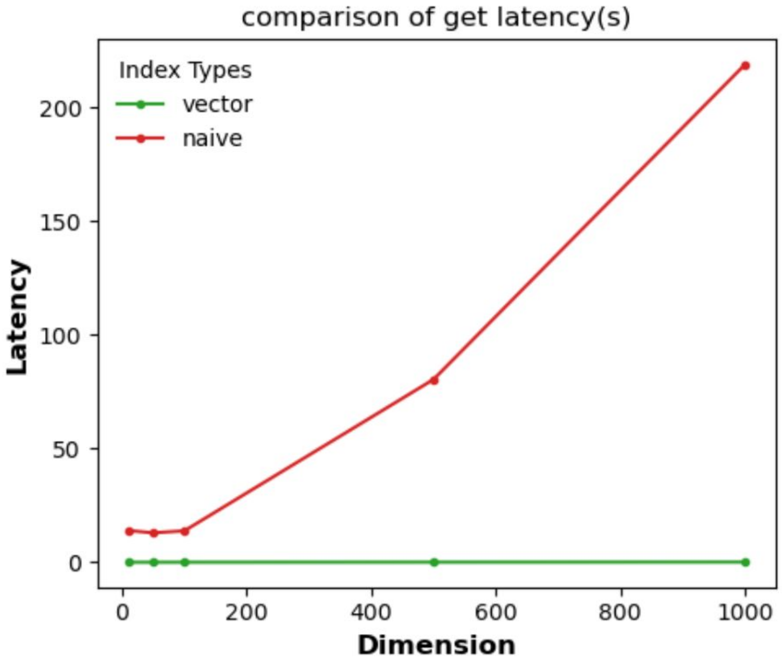
- NSW cannot be standalone index
  - Limited operations support
  - Bad put/delete performance
- Opens up optimization windows for integrations
  - Lazy delete
  - Lazy put could also be possible
- SIMD executions could be super helpful

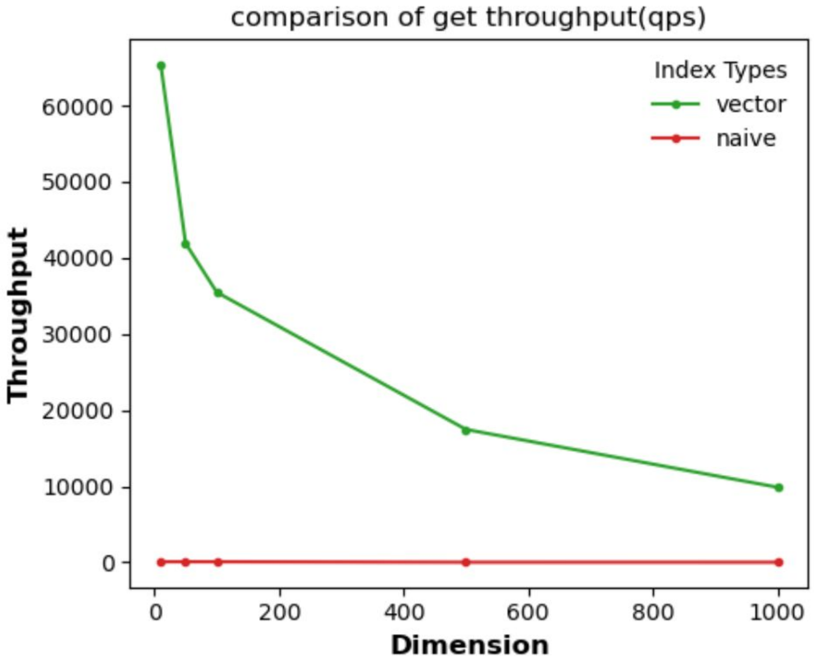
Evaluation against Naive: Environment Set Up

Arch: Apple M1 without SIMD acceleration
Test Set: get 1000 vectors from 5000 vectors with following dimensions

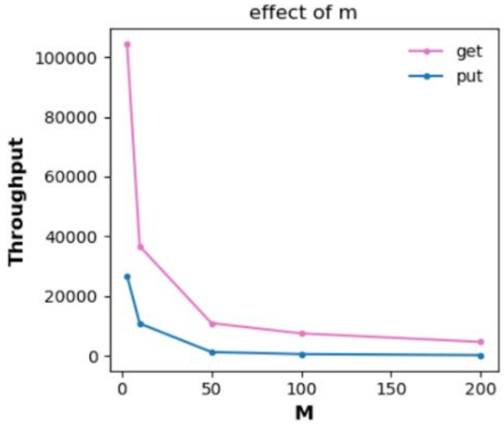
- 10
- 50
- 100
- 500
- 1000

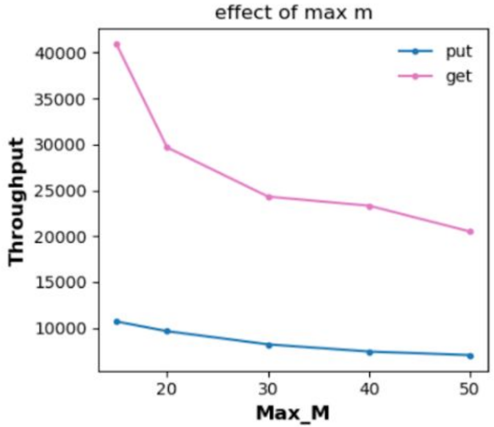
Evaluation: Naive KNN vs NSW Index

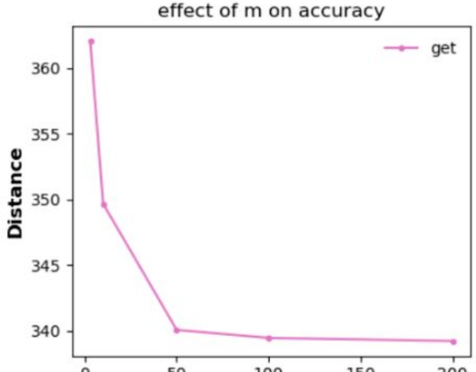


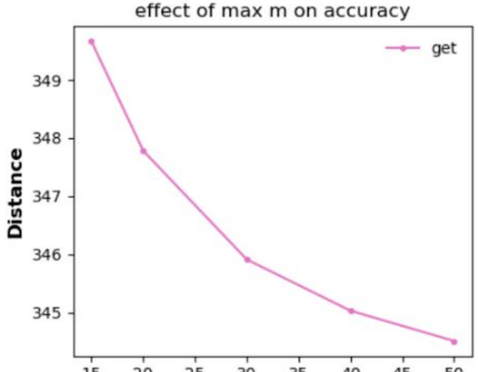


Evaluation Against Different Parameter


effect of m
get
put
Throughput
100000
80000
60000
40000
20000
0
0
50
100
150
200
M


effect of max m
put
get
Throughput
40000
35000
30000
25000
20000
15000
10000
20
30
40
50
Max_M


effect of m on accuracy
get
Distance
360
355
350
345
340


effect of max m on accuracy
get
Distance
349
348
347
346
345

Conclusion

NSW is **AWESOME**!!

The most important takeaway is:

An efficient vector index, when building on top of an existing database, can be very EASY