Verifiable Nearest Neighbor Search

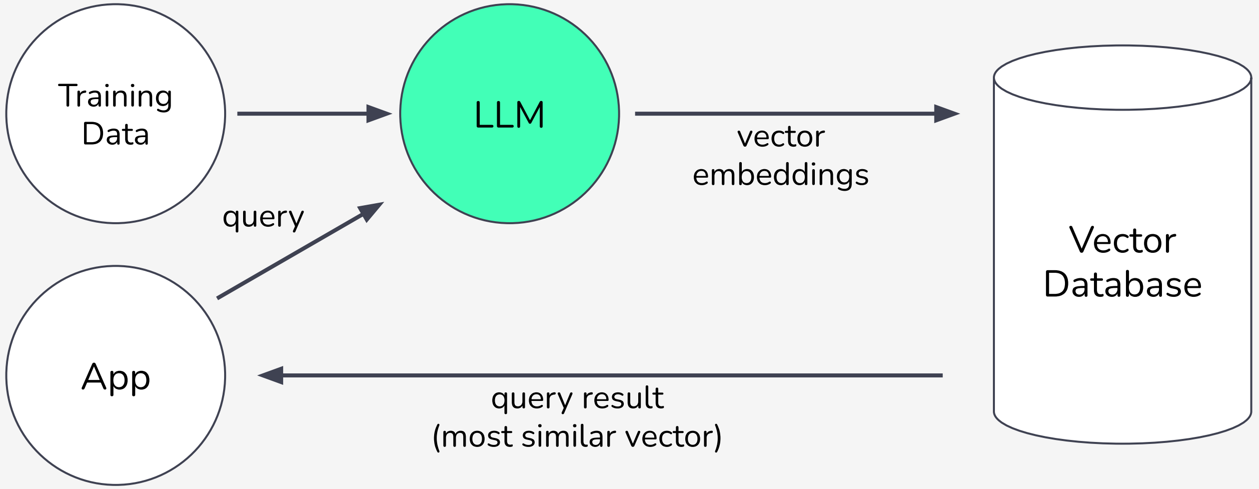
How to make verifiable queries to a vector database?

Erhan Tezcan
https://github.com/erhant/aligned-vnns

October 28, 2024

Vector Database

Midst of the AI revolution, applications are structuring large amounts of unstructured data using LLMs. Text, images, and sound are transformed into high-dimensional vectors called **embeddings**, mathematical objects representing semantic attributes of data.

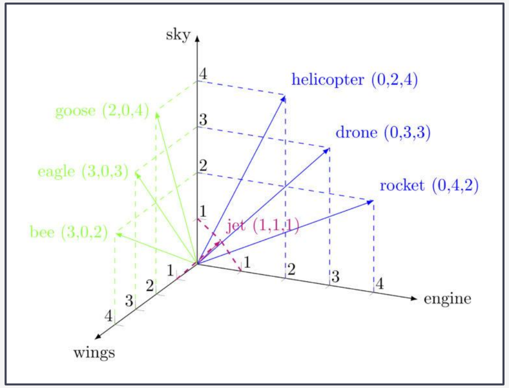


Vector Database

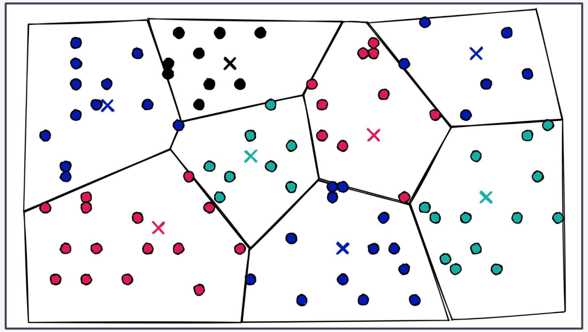
During a query, the VectorDB does a high-dimensional similarity search and returns the most similar vector to the query vector embedding.

Doing this search exhaustively (i.e. Flat Indexing) over the entire vector database is an option, although **expensive**!

There are slightly less accurate (i.e. **approximate**) but more efficient methods as well, denoted as ANN.



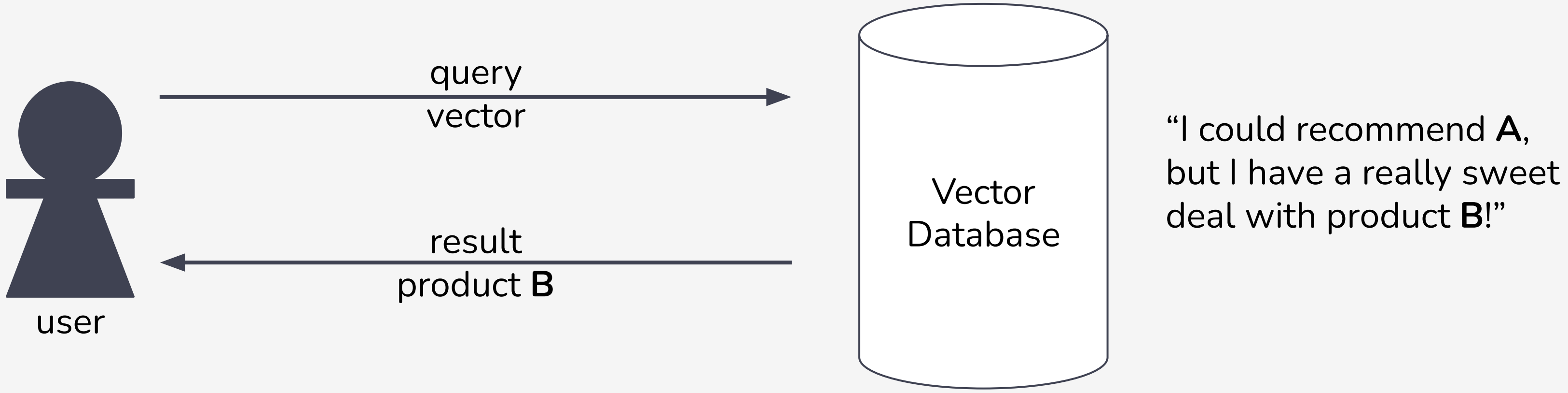
https://corpling.hypotheses.org/495



https://www.pinecone.io/learn/series/faiss/product-quantization/

Problem: Trust without Verification

Suppose you have a query vector derived from your interests, and a similarity search will recommend you relevant products. How do you make sure this is an **honest interaction**?


query
vector
result
product B
user
Vector
Database
"I could recommend A, but I have a really sweet deal with product B!"

Solution: Verifiable Computation

Zero-knowledge cryptography allows one to do “**verifiable computation**”.

A **verifiable Flat Indexing query** shall prove the following:

- The entire committed index was used while processing the query.
- The result respects the **similarity measure** agreed upon.

In our case, the vectors are high-dimensional (e.g 384 / 768 / 1024 / 1536) vectors with floating point values; making it really expensive to prove things over them within a zkVM.

Solution: Verifiable Computation

For the **similarity measure**, we are using **Euclidean distance**. While **Dot-Product** would be a bit more circuit friendly, it performs a lot worse.

$$\|\mathbf{a} - \mathbf{b}\| = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

Solution: Verifiable Computation

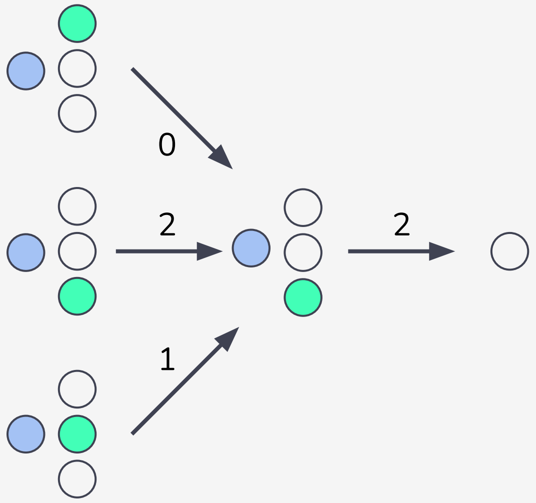
Even for just around 5-6 vectors with 100s of elements, the prover gets killed mid-war due to OOM (out-of-memory) problems.

We solve this issue with proof aggregation, with what we denote as **Recursive Similarity Search**.

Instead of searching over the entire Flat Index, we search batch by batch and continue eliminating non-similar vectors until we end up with just one vector in the end, being the result itself.

Recursive Similarity Search

Consider the following dataset* with 9 vectors and a query. With a batch size of 3, we create 4 proofs with commitments to input vectors ○, output vector 🟢, query vector 🔵 and the chosen index.



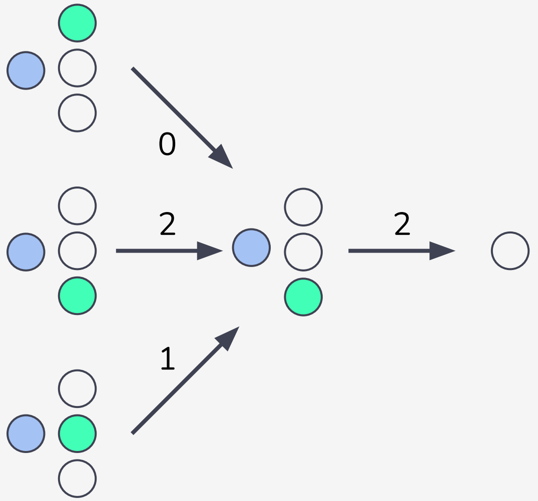
in(0,0), out(0,0), 0
in(1,0), out(1,0), 2
in(2,0), out(2,0), 1
in(0,1), out(1,0), 2

for each batch

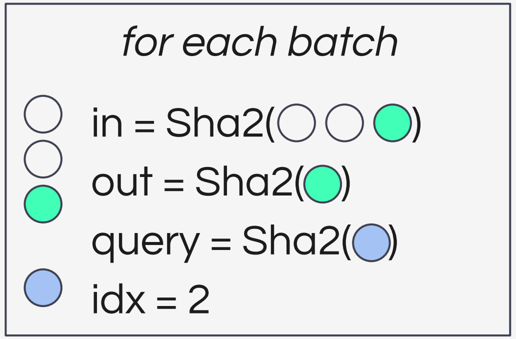
in = Sha2(○ ○ 🟢)
out = Sha2(🟢)
query = Sha2(🔵)
idx = 2

Recursive Similarity Search

The database owner can verify any **batch** by revealing the vectors in plaintext, with the same order as proved, so that the **hashes** and **commitment index** match. Connecting batches is done via the commitments of query and output vectors.



in(0,0), out(0,0), 0
in(1,0), out(1,0), 2
in(2,0), out(2,0), 1
in(0,1), out(1,0), 2

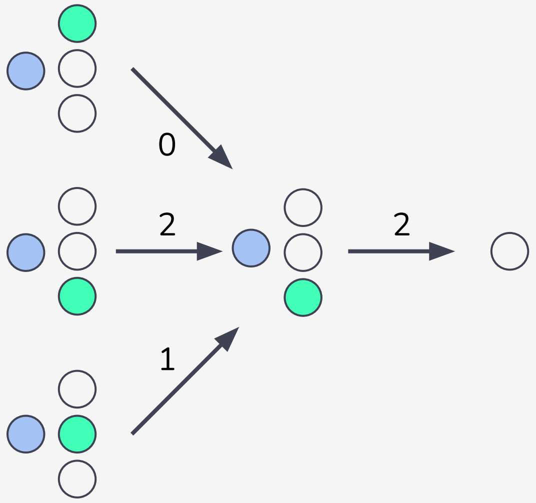


for each batch

in = Sha2(○○🟢)
out = Sha2(🟢)
query = Sha2(🔵)
idx = 2

Recursive Similarity Search

We could aggregate all proofs as well to obtain a final proof, which does work to some extent, it still gives OOM after certain amount of proofs to be aggregated. The final solution is to **send all proofs** to Aligned Layer one by one!



in(0,0), out(0,0), 0
in(1,0), out(1,0), 2
in(2,0), out(2,0), 1
in(0,1), out(1,0), 2

for each batch

○ ○ 🟢 🔵

in = Sha2(○ ○ 🟢)
out = Sha2(🟢)
query = Sha2(🔵)
idx = 2

Evaluations

For our evaluation, we will use the foods-med.json dataset in the repository, with the embedding vector of the text:

- *Japanese broth made with fermented soy*

The embedding vector is obtained using **all-minilm** model, and it has a dimension of 384.

The f32 values with little-endian bytes flattened has a SHA256 digest of:
583d29818a459e0a18e27309c32ec65591ecf8dec18e97c14d628e1805fc1787
Note that this is also our query commitment value as well in the proof.

See https://ollama.com/library/all-minilm for details on the used model.

Evaluations

With a batch size of 4 and vector dimension of 384 (again using **all-minilm** model) it takes around 5~6 minutes to generate a proof using a Macbook Air (M2) with 16GB RAM.

The total number of proofs generated for n vectors with batch size b is given by the following expression, where the smallest term is 1:

$$\left\lceil \frac{n}{b} \right\rceil + \left\lceil \frac{n}{b^2} \right\rceil + \left\lceil \frac{n}{b^3} \right\rceil + \ldots + 1$$

Evaluations

We ran an example over foods-med.json dataset with the embedding of “, which has 10 elements. Using a batch size of 4 we get:

- (4) + (4) + (2) = 3 proofs
- (3) = 1 proof

It took around 22 minutes in total.

$$\left\lceil \frac{10}{4} \right\rceil + \left\lceil \frac{10}{16} \right\rceil = 3 + 1$$

We have submitted all proofs to Aligned Layer:

```
./aligned.sh submit ./secrets/wallet.json ./data/foods-med.json
```

https://explorer.alignedlayer.com/batches/0x4c8552175b55bf515b7099563dc70c90ac004740ab46e38d52f0279d77191f81
https://explorer.alignedlayer.com/batches/0x934f4292323bdfcb513a5e9c646da2edf19f6f16352c0b293da9a12c89693885
https://explorer.alignedlayer.com/batches/0x9ab91e074c9a746dbc1ad1cea82ad47fbf9ef41c51e2eba7cb91c6f1fa343a06
https://explorer.alignedlayer.com/batches/0x6caefdb2f893f7e4f5b14d4aca837820ccea83ab3c15e87f1f12dd4bbad7ce23

Evaluations

To read the result, we check the output commitment of the final proof:
3197533e2a797a14eb2999ffd2fb814e4df38dc9f1d0aaa97a4581ede801d40a

Within foods-med.index.json we can lookup this value, and see that it belongs to the following item:

```
"data": {
   "name": "Sushi",
   "description": "Japanese vinegared rice with raw fish, light and fresh"
}
```

Et voila, that is our **verifiably** computed final result!

Future Work

(1) The project initially started as a verifiable Approximate Nearest-Neighbor calculator, which has plenty of algorithms with varying degrees of complexity (HNSW, ANNOY, PQIVF).

There are in-memory implementations for these, however, the zkVM got OOM errors rather easily with these, so instead we have opted for Flat Index (exhaustive similarity search).

As a future work, we could aim for more performant ANN methods to actually target production-level algorithms with this recursive approach.

Future Work

(2) To go even further, we could try different quantizations / distance metrics so that it is less costly to compute within the finite-field arithmetic powered zkVMs, but still has enough accuracy. This requires a bit of research!

(3) We could also test the current implementation in a more powerful prover machine, so that a more realistic database size could be benchmarked to see how long would it take to generate a full proof.

Fin

https://github.com/erhant/aligned-vnns

x: 0xerhant | **tg:** tgerhant