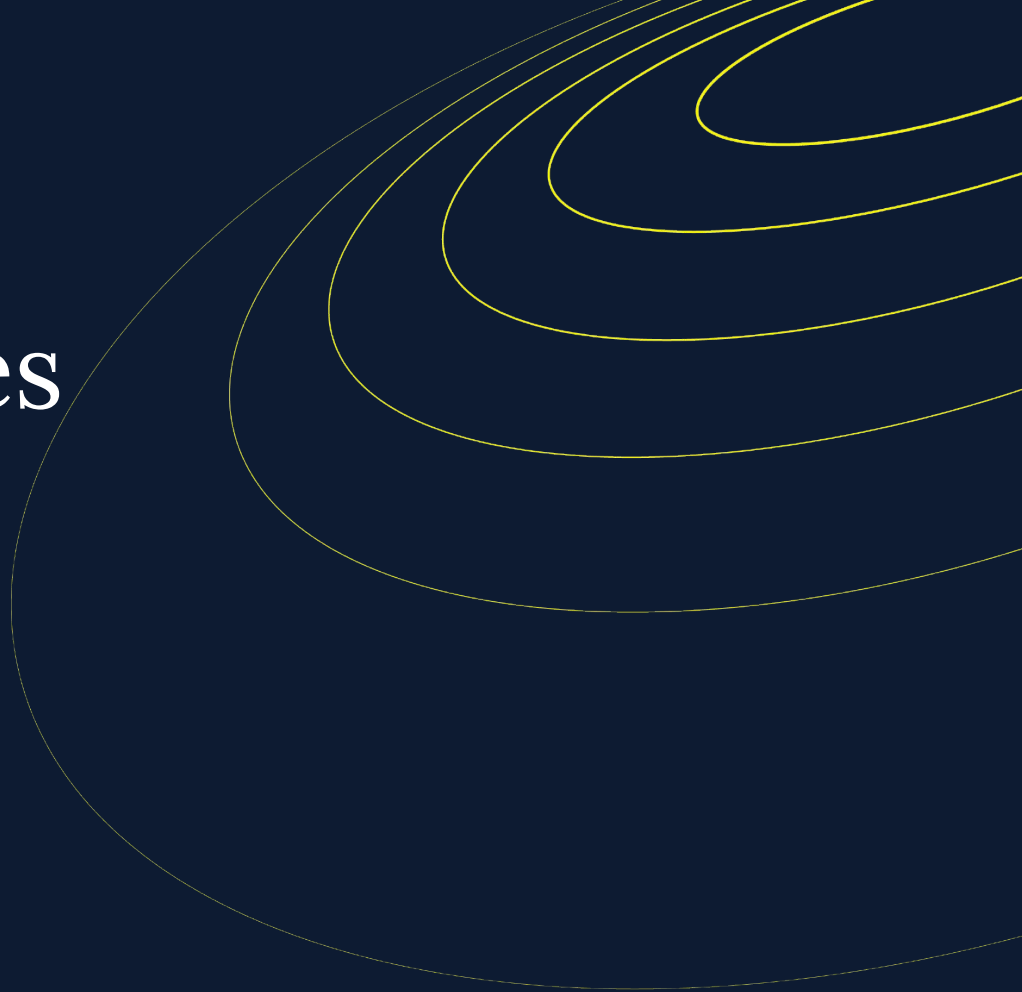


Aerospike

Vector Databases Demystified

How they work and
why they're important

Art Anderson
Director of Developer Advocacy

Vector Databases Demystified

create-a-vector-db.netlify.app

https://github.com/aerospike-examples/create-a-vector-database

While you wait, please help create a data set by visiting this url and create a product or two for a fictional bank dealing with mythological creatures

For example:

***Type**: Mortgage Product*
***Name**: Swamp Leasing*
***Description**: Lending for the leasing of swamps. Primarily focused around murky swamps for ogres, but other creatures may use this product.*

Use your imagination, make it kooky and funny and not rude but focused around mythological creatures please!

Agenda

01 Large Language Models (LLMs)

02 Vectors and Semantic Search

03 Implementing a Vector Database

04 Commercial Vector Databases

05 HNSW Algorithm

06 Use of Vector Databases

01 Large Language Models

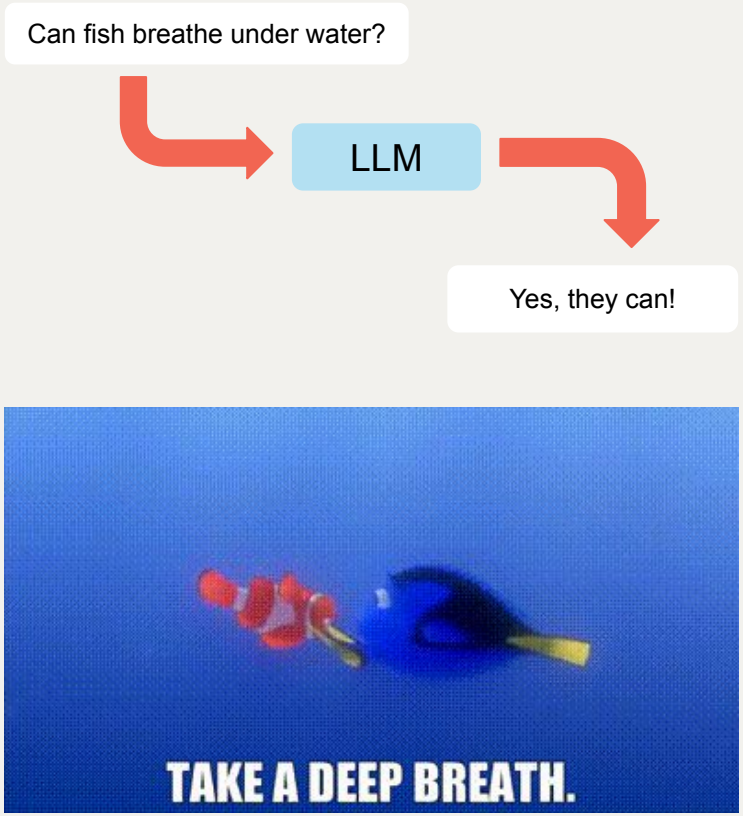
What is an LLM anyway?

LLMs

Large Language Models (LLMs) are a type of artificial intelligence that uses deep learning techniques and massively large data sets to understand, summarize and generate new content.

Recently popularized through Chat-GPT, Llama, Gemini, etc.

Billions of calculations per interaction!



Prompt Engineering

```
PROMPT = '''\
You are a happy chatbot designed to answer the following question
in a whimsical manner.

Question: Do fish breathe air?
'''
```

```
PROMPT = '''\
You are an angry chatbot designed to answer the following question
in a gruff manner.

Question: Do fish breathe air?
'''
```


Using Large Language Models

gemini-1.5-flash-001

```python
from google.oauth2 import service_account
import vertexai
from vertexai.generative_models import GenerativeModel

# Credentials to google account to allow login
credentials = service_account.Credentials.from_service_account_file("auth.json")
vertexai.init(project="aero-devrel", location="us-central1", credentials=credentials)

model = GenerativeModel("gemini-1.5-flash-001")

chat = model.start_chat(response_validation=False)
result = chat.send_message(
    content="Do fish breathe air?",
    stream=False
)
print(result)
print(result.candidates[0].content.parts[0].text)
```


Take a look!

Let’s play around with an LLM

Improving LLM Output

LLMs are prone to:

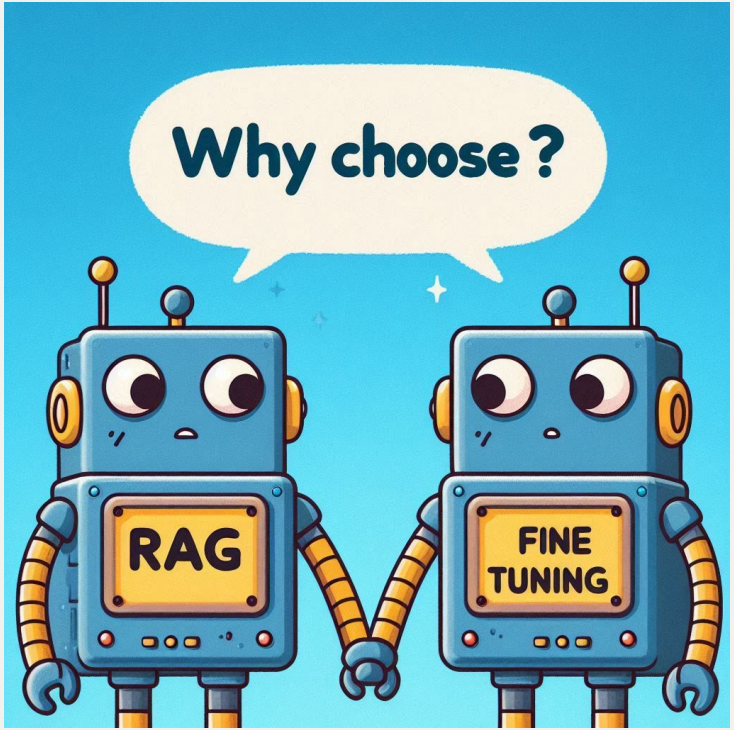
- Stale data (only know that they're trained on)
- Non domain specific
- Hallucinations

How do I create a secondary index in Aerospike?

LLM

Hallucination!

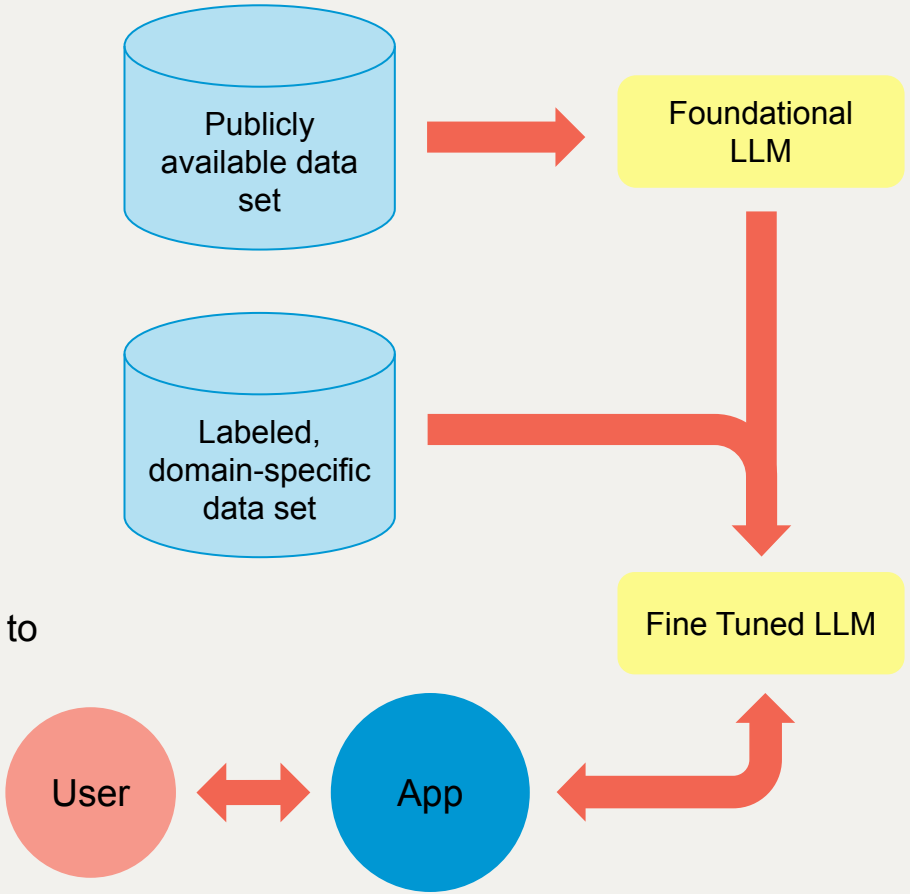
Indexes in Aerospike are created using the Secondary Index Administration Tool



Fine tuning LLMs

Domain of the data scientist

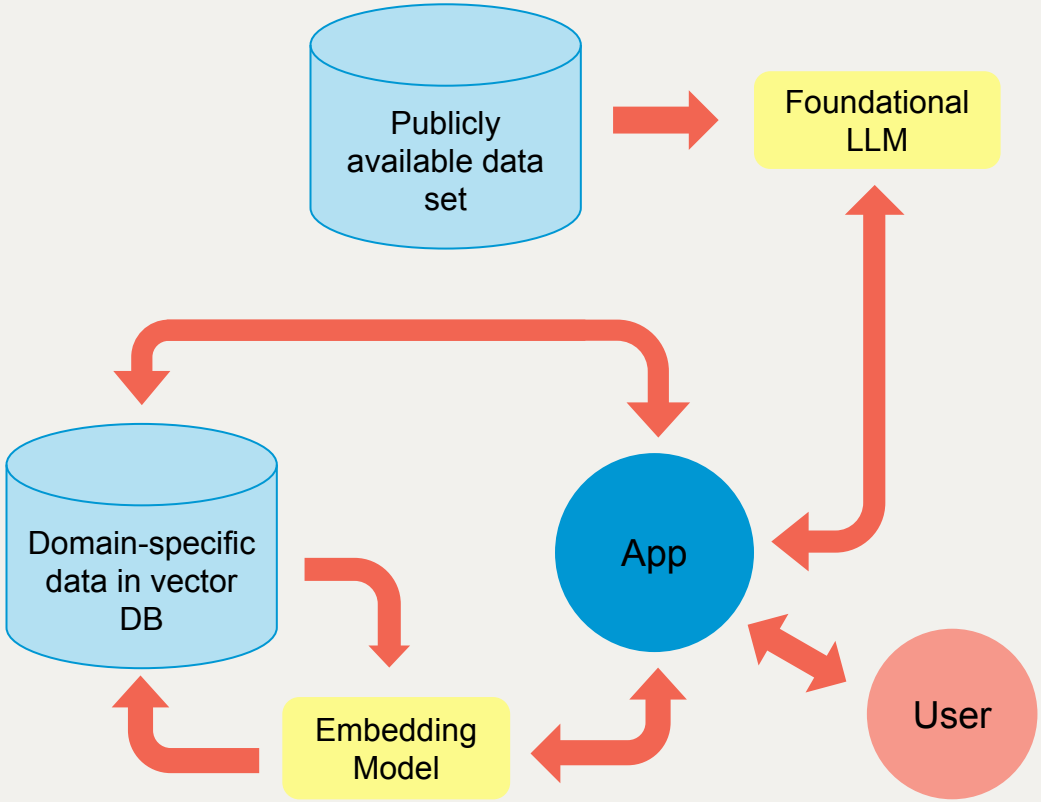
- Adjusts the model's parameters and embeddings based on new data
- Helps model perform better for specific tasks
  - Eg adapting to legal domain
- Requires lots of time, compute and labeling
- Harder to adapt to new information due to retraining



Retrieval Augmented Generation

Domain of the developer

- Allows stricter access control to proprietary data
- More cost-effective and scalable
- Rapid access to latest data



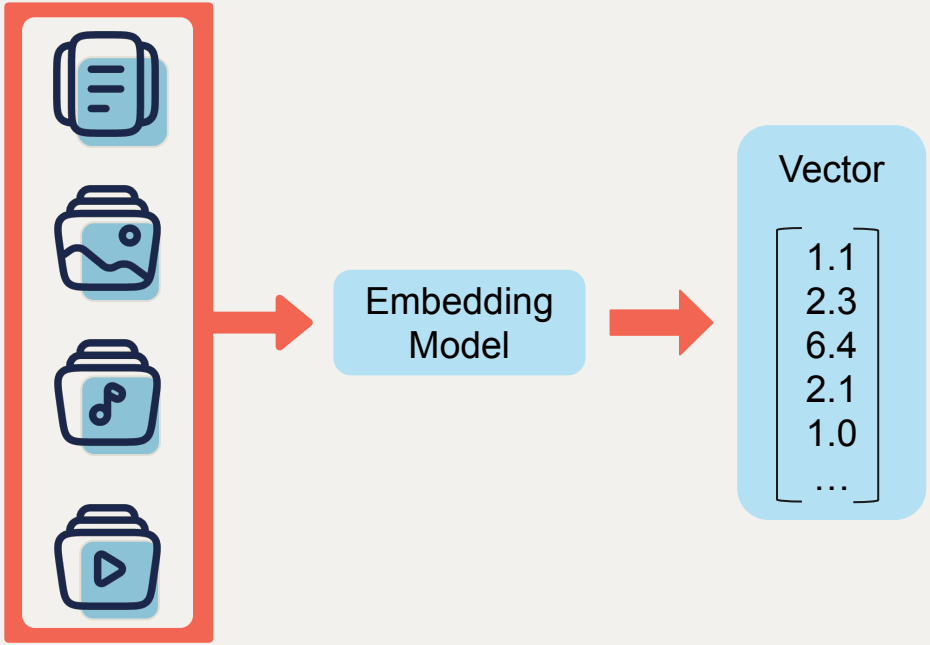
Hallucinations!

Prevention through prompts

02 Vectors and Semantic Search

How does a semantic search work?

Semantic search with vectors



Vector embeddings capture the semantic meaning of source data.

- Reduces high volume data according to meaning
- Similar to a “lossy” algorithm like JPEG compression

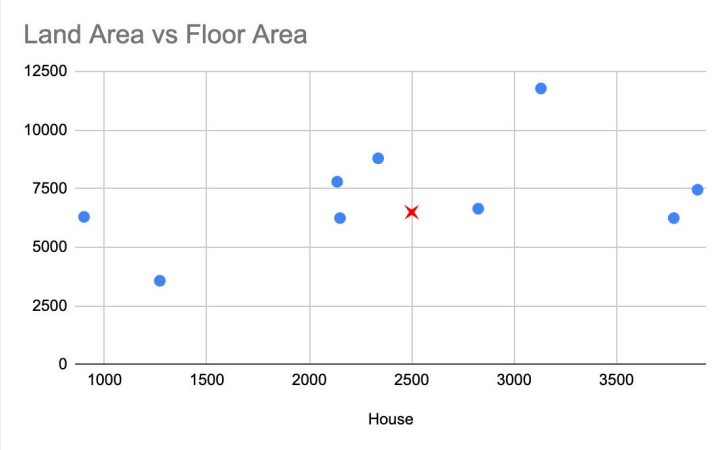
Vector example

Consider searching for a house with the following criteria:

- You want about 2,500ft^2 floor area and 6,500ft^2 land area
- Floor space is twice as important compared to land area

Of the 9 houses below, which one should you choose?

House	Land Area	Floor Area
1	8800	2336
2	7800	2136
3	6300	900
4	6250	2150
5	6250	3780
6	3575	1270
7	11780	3131
8	7460	3896
9	6650	2825



Embedding example

To turn each house into a vector:

1. Normalize both land area and floor area by dividing each one by the largest value in the column
   - Gives a value in the range `[0,1]` for each value
   - E.g., house 1 land area is 8,800ft^2, largest land area is 11,780ft^2, 8800/11780 = 0.7470

2. Since floor area is worth twice as much as land area, divide land area by 2 (weighting)

3. Vector for each house is:
   `[W Land, W Floor]`

House	Land Area	Floor Area	*N* Land	*N* Floor	*W* Land	*W* Floor
1	8800	2336	0.7470	0.5996	**0.3735**	**0.5996**
2	7800	2136	0.6621	0.5483	**0.3311**	**0.5483**
3	6300	900	0.5348	0.2310	**0.2674**	**0.2310**
4	6250	2150	0.5306	0.5518	**0.2653**	**0.5518**
5	6250	3780	0.5306	0.9702	**0.2653**	**0.9702**
6	3575	1270	0.3035	0.3260	**0.1517**	**0.3260**
7	**11780**	3131	1.0000	0.8036	**0.5000**	**0.8036**
8	7460	**3896**	0.6333	1.0000	**0.3166**	**1.0000**
9	6650	2825	0.5645	0.7251	**0.2823**	**0.7251**

Semantic Search example

Now we need to find which vector in the data set is closest to the desired house.

Apply same algorithm to the desired house:

```
[0.5 * 6500 / 11780, 2500 / 3896] = [0.2579, 0.6417]
```

There are various definitions of “closest”. Two examples are:

Cosine similarity (normalized dot product)

- Gives result in range [-1,1].
- Bigger numbers are closer, two identical vectors gives 1.0

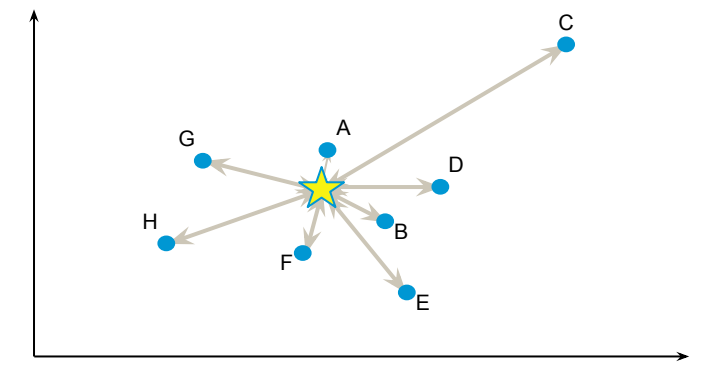
Euclidean distance

- Smaller numbers are closer

Test different algorithms for best results!

Euclidean Distance

For this example, A is closest to the target, C is furthest from the target.



- Distance as “measured by a ruler”
- Good for low-dimensional vector spaces
- Good for text spaces

Normally “Squared Euclidean Distance” is used for efficiency.

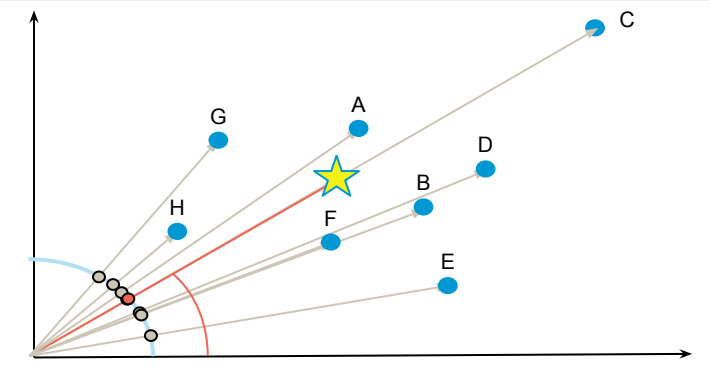
```
def squaredEuclidean(vect1, vect2):
    magnitude = 0.0
    for i in range(len(vect1)):
        length = vect1[i] - vect2[i]
        magnitude += length * length
    return magnitude
```

$$Distance = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

$$Distance = (x1 - x2)^2 + (y1 - y2)^2$$

Cosine Similarity

For this example, C is closest to the target, E is furthest from the target.



- Angle between the vectors as seen by an observer at the origin.
- Distance to point is ignored
- Good for high-dimensional vector spaces

Good for retrieval of the most similar texts for a given document.

```
def cosineSimilarity(vect1, vect2):
    dotProduct = 0.0
    magnitude1 = 0.0
    magnitude2 = 0.0
    for i in range(len(vect1)):
        dotProduct += vect1[i] * vect2[i]
        magnitude1 += vect1[i] * vect1[i]
        magnitude2 += vect2[i] * vect2[i]
    return dotProduct / sqrt(magnitude1 * magnitude2)
```

$x = Weighted\ Land$

$y = Weighted\ Floor$

$$Distance = \frac{x \cdot y}{||x|| * ||y||} = \frac{x1 * x2 + y1 * y2}{\left(\left(\sqrt{x1^2 + x2^2}\right) * \left(\sqrt{y1^2 + y2^2}\right)\right)}$$

x · y = dot product of x and y
||x|| = magnitude of x

Results

To get the closest house to the requirements:

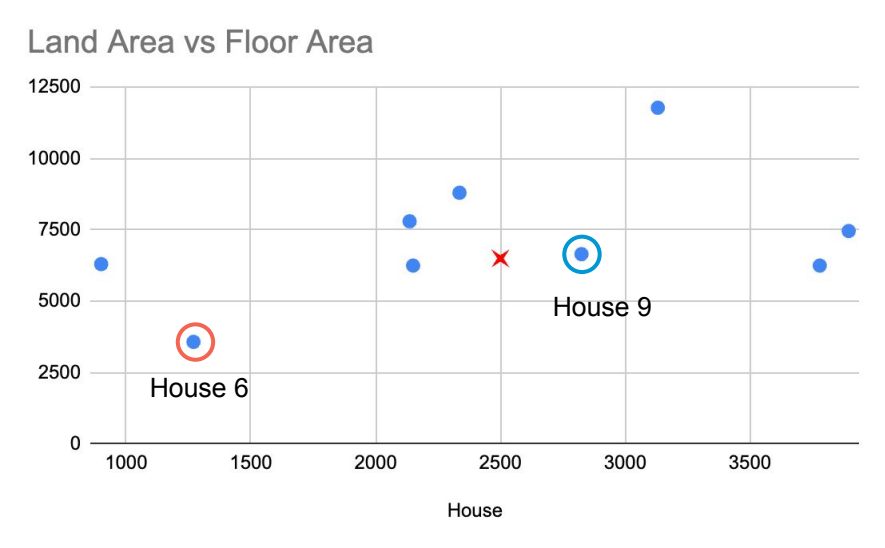
1. Calculate similarity for every house against our target house
2. Sort the results from largest to smallest similarity score
3. Pick the best one

House	Land Area	Floor Area	*N* Land	*N* Floor	*W* Land	*W* Land	Cosine Similarity	Squared Euclidean
1	8800	2336	0.7470	0.5996	0.3735	0.5996	0.9886	0.0113
2	7800	2136	0.6621	0.5483	0.3311	0.5483	0.9906	0.0118
3	6300	900	0.5348	0.2310	0.2674	0.2310	0.8995	0.1687
4	6250	2150	0.5306	0.5518	0.2653	0.5518	0.9991	0.0082
5	6250	3780	0.5306	0.9702	0.2653	0.9702	0.9903	0.1081
6	3575	1270	0.3035	0.3260	0.1517	0.3260	**0.9996**	0.1151
7	11780	3131	1.0000	0.8036	0.5000	0.8036	0.9887	0.0765
8	7460	3896	0.6333	1.0000	0.3166	1.0000	0.9951	0.1301
9	6650	2825	0.5645	0.7251	0.2823	0.7251	0.9994	**0.0070**

Target house has 6,500ft^2 land area, 2,500ft^2 floor area.area

Results

House	Land Area	Floor Area	*N* Land	*N* Floor	*W* Land	*W* Floor	Cosine Similarity	Squared Euclidean
6	3575	1270	0.3035	0.3260	0.1517	0.3260	**0.9996**	0.1151
9	6650	2825	0.5645	0.7251	0.2823	0.7251	0.9994	**0.0070**



Best result:

Squared Euclidean

Notes

- The previous example used vectors of 2 dimensions.
- Other house attributes (price, number of bedrooms, age, neighborhood quality, etc) can be introduced.
- Real world vectors can have hundreds to thousands of dimensions.
- Each vector in the search has the same number of dimensions.
- Each data set was normalized so vectors were in the range `[0, 1]`. This is not a requirement.
- In real models, the weights are determined by AI models like CNNs.
- The algorithm that creates the vectors on the data must be the same one used to create vectors for search.

03 Implementing a Vector DB.

Embedding Model

Hugging Face
https://huggingface.co/

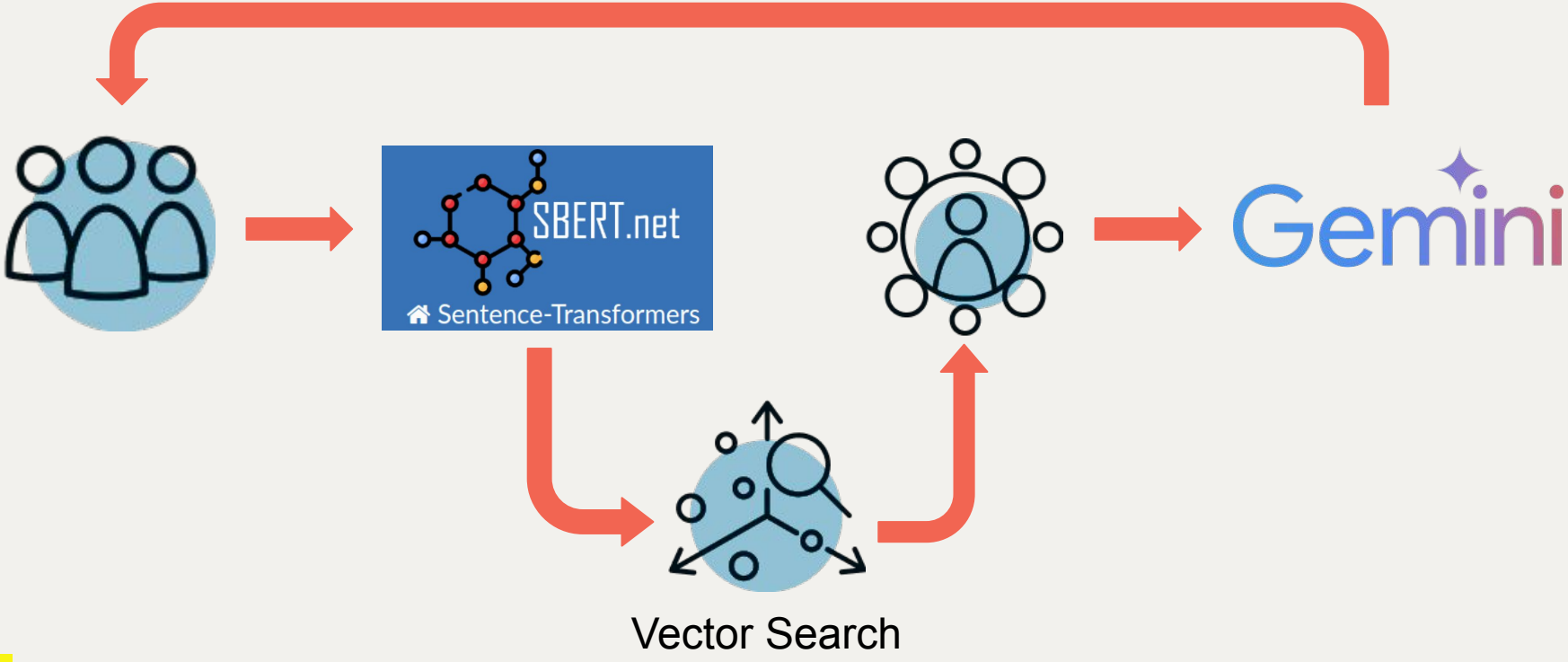
all-MiniLM-L6-v2

```
sentence = "This is a long sentence which we want to transform"
from sentence_transformers import SentenceTransformer
model = SentenceTransformer("all-MiniLM-L6-v2")
return model.encode(sentence)
```

```
[-4.22671344e-03  1.08605370e-01  5.85176945e-02  7.73526449e-03
  1.49376178e-02  3.88745330e-02  1.62545759e-02 -2.23493334e-02
  6.18864857e-02  5.37480740e-03  6.67040721e-02 -2.60523316e-02
  ...]
```

*384 dimensions

RAG application flow



Creating vectors!

See the RAG in action

04 Commercial Vector Databases

Advantages of a commercial database

Drawbacks of our database

Computational Scalability

- Comparing 2 vectors takes at least 1,000 operations (384 dimensions).
- Brute force on every vector for each comparison
- Sorting the results

Memory Scalability

- Each vector requires 384 floats ~= 3kB data

1 billion vectors => 1 trillion ops per comparison
~1.3TB DRAM

Our API:

```
set_comparator(comparator, ordering)
add_entry(data)
similarity_search(vector, count)
```


Commercial Vector Databases

Computational Scalability

- Typically pre-compute an index on vectors with a Nearest Neighbor search algorithm
- Dramatic reduction in computational requirements
- May still require 1000s of lookups for one query

Memory Scalability

- Many still store all vectors in memory
- Some store vectors on SSDs with memory cache

Commercial API:

```
define_index(...)
add_entry(data)
similarity_search(vector, count)
```

Vector DBs are easy to get started with!

See https://github.com/aerospike-examples/rag-demo for a similar repository with full integration to a commercial vector database.

05 HNSW Algorithm

Hierarchical Navigable Small Worlds

HNSW are a layer-traversal index

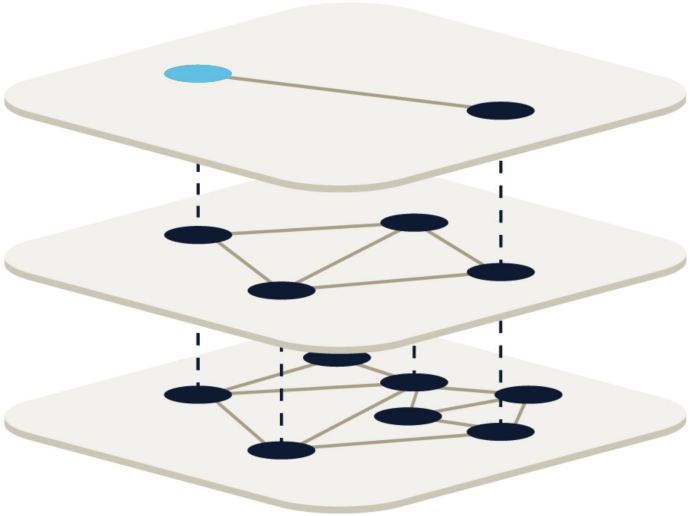
Start at higher levels:

- Fewer data points
- Each navigation moves further

Once desired locality is found

- Drill down to next index layer
- Repeat the process

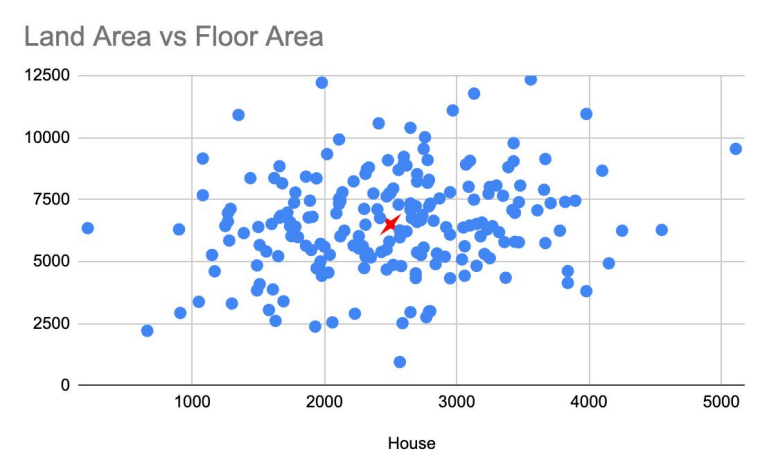
1 HNSW search can result in thousands of data point lookups for very large data sets.

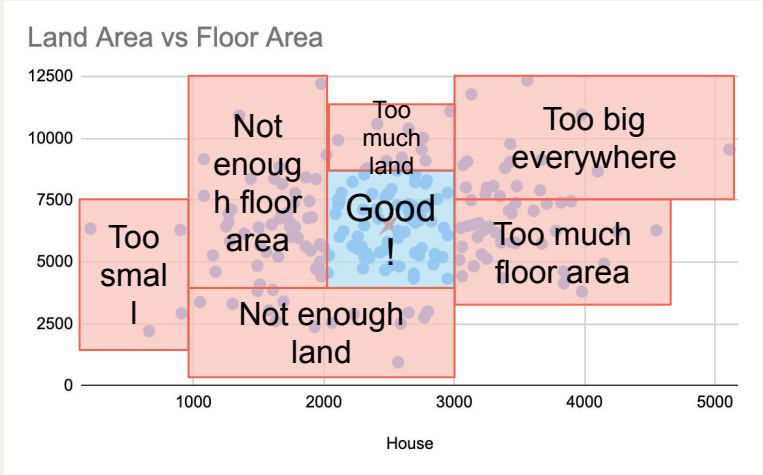


HNSW simple example

To continue the house example from before, say there are hundreds of points not just 9:

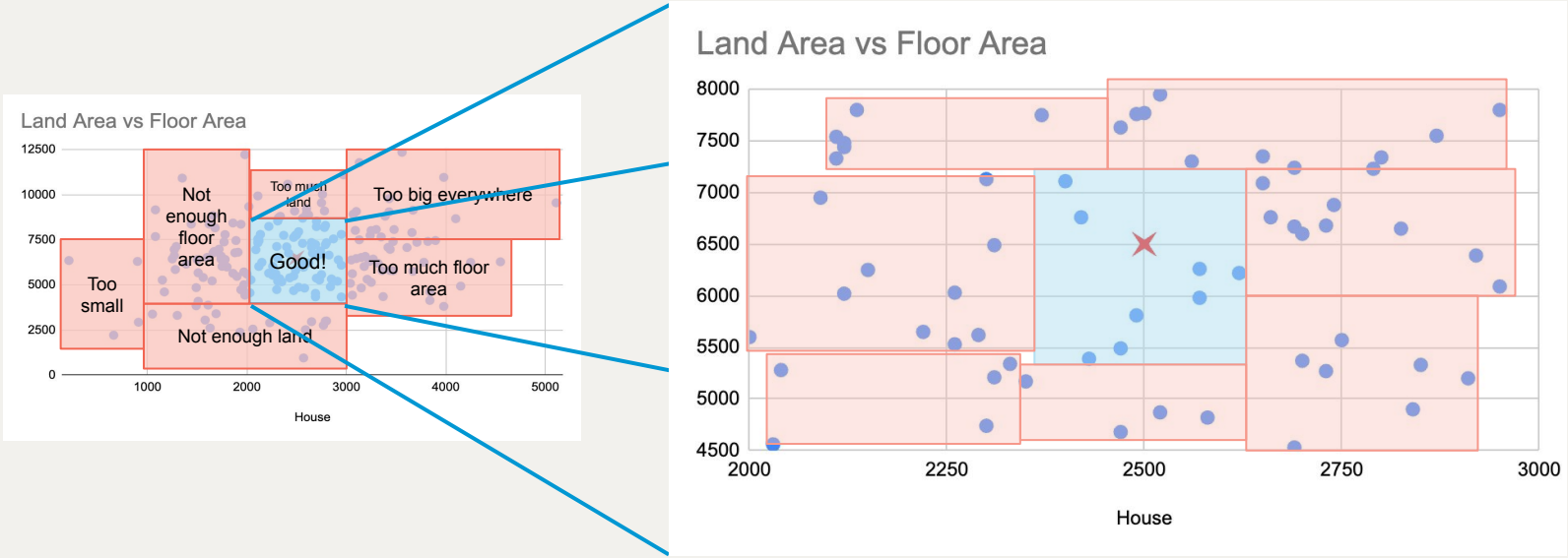
Intuitively you would break the graph into regions, focus on the desired region then drill down:





Drilling down

Once the best region is identified, drill into that region and repeat the process if necessary



In an HNSW, the regions are an index.

Once the number of points in region is small enough, use brute force.

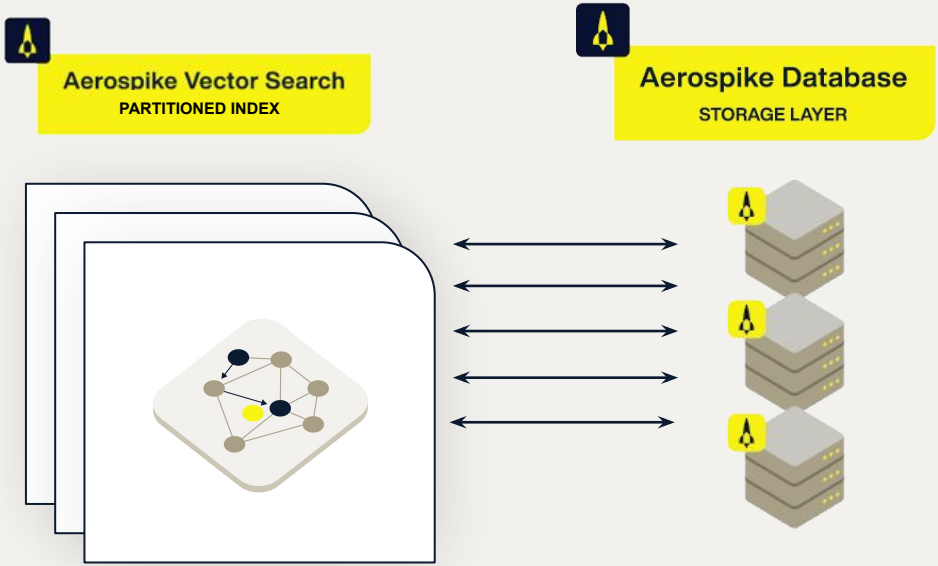
Aerospike implementation of HNSW

Storage

- Fast K-V retrieval
- Scalable to PB
- Very low latency
- Stores:
  - application data
  - vector search index

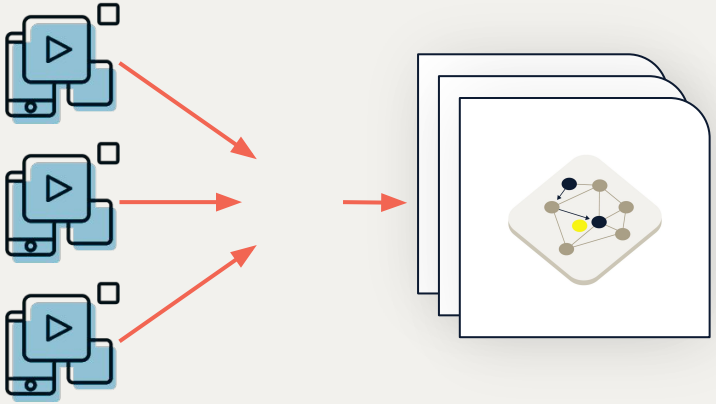
Index

- Partitioned, horizontally scalable
- Caches vector search off storage
- Self-healing
- Geometric cache to optimize cache-hit ratio



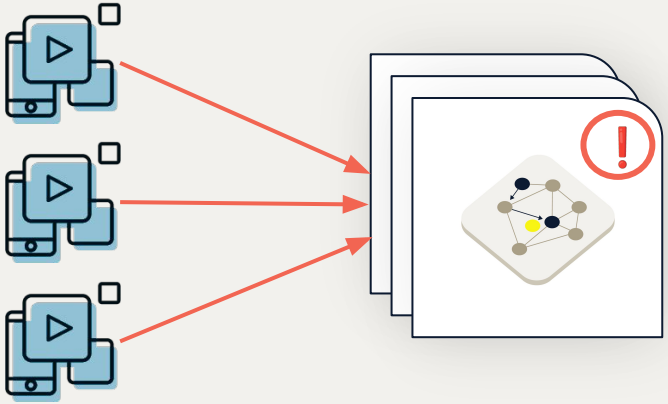
Scalable HNSW Challenge - Writes!

Serialized writes



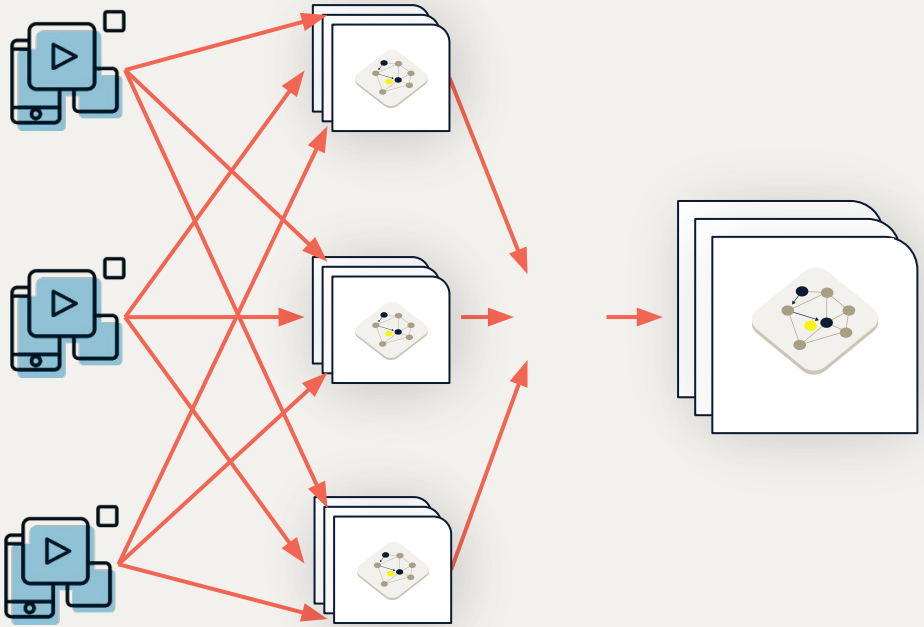
Writes preserve integrity of the index
Lock contention lowers write throughput
Non-scalable solution

Parallelized writes



Faster parallel writes
Index corruption highly likely

Aerospike solution



- Application writes to each shard broken into batches
- Each shard has locality to a particular section of the HNSW
- Each batch forms a unique HNSW with minimal lock contention
- Batches merged into main HNSW index
- Allows good parallelization whilst maintaining index integrity
- Self healing index processes can fix minor inconsistencies from node crashes, etc.
- BUT writes can be slower to reflect for reads (normally not an issue)

06 Use of Vector Databases

Similarity Search

Fraud Detection: Vector search can analyze transaction patterns by transforming data into vector representations, enabling real-time detection of anomalies and potential fraud.

Enhanced Search Capabilities: Vector search can improve the accuracy and relevance of search results within applications, and allows easy search across structured or unstructured data.

Transaction Categorization: Using vector embeddings to classify and categorize transactions can help users track spending patterns and generate better financial insights.

Risk Assessment: Assess credit risk from vector embeddings for customer data, helping them make informed lending decisions.

Predictive Analytics: By utilizing vector representations of historical data, you can build models that predict future trends, aiding in proactive decision-making.

RAG in the wild

Chatbots: Chatbots/virtual assistants can provide accurate, context-aware responses to user inquiries.

Regulatory Compliance: Ingestion of the latest regulatory requirements and generation of compliance reports or updates based on the latest regulations.

Risk Management: RAG can pull historical risk data and generate comprehensive risk assessments for loans or investments, aiding in decision-making processes.

Sentiment Analysis: By ingesting data from social media, news articles, and financial reports an idea of sentiment can be generated.

Loan and Mortgage Processing: Streamlining the loan application process by retrieving necessary documentation and generating personalized loan offers based on applicant profiles.

The Aerospike Community

Join the conversation and become a part of our community!

Try the Aerospike Vector Database for free!

Aerospike

Thank you