Google Cloud

Vector Indexing
A Comparative Analysis

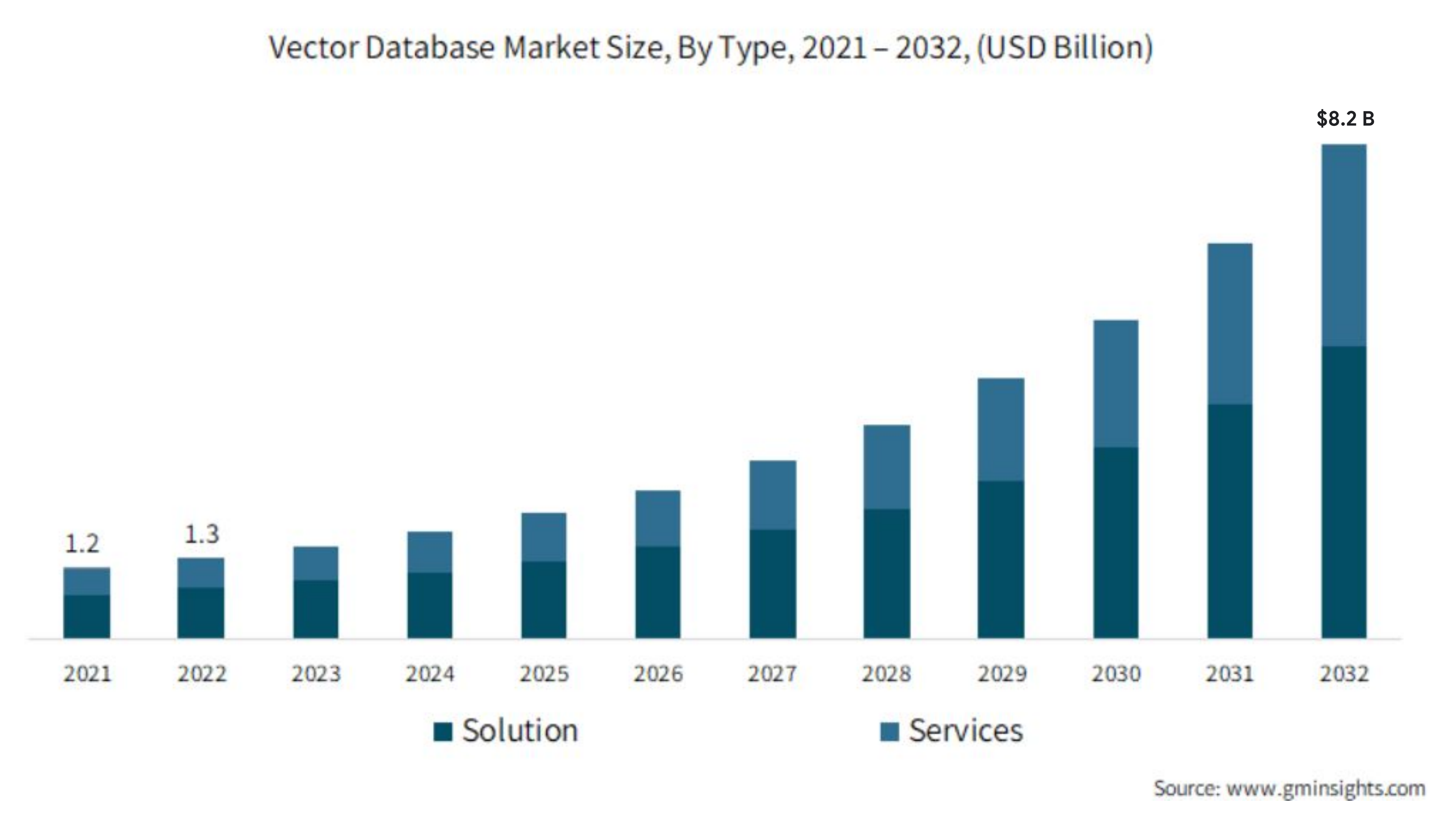
6 March 2025

PGConf India

Today's Speaker

Abhijeet Rajurkar

APAC Databases Lead - Google Cloud


Vector Database Market Size, By Type, 2021 – 2032, (USD Billion)
$8.2 B
1.2
1.3
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
Solution
Services
Source: www.gminsights.com

Databases bridge the gap between LLMs and enterprise Gen AI apps

Databases:

- Provide the most **up-to-date** data
- Can efficiently store and search **vector** embeddings
- Are your **trusted** and familiar data store

PostgreSQL pgvector Extension

Easiest way to get started with a vector use cases

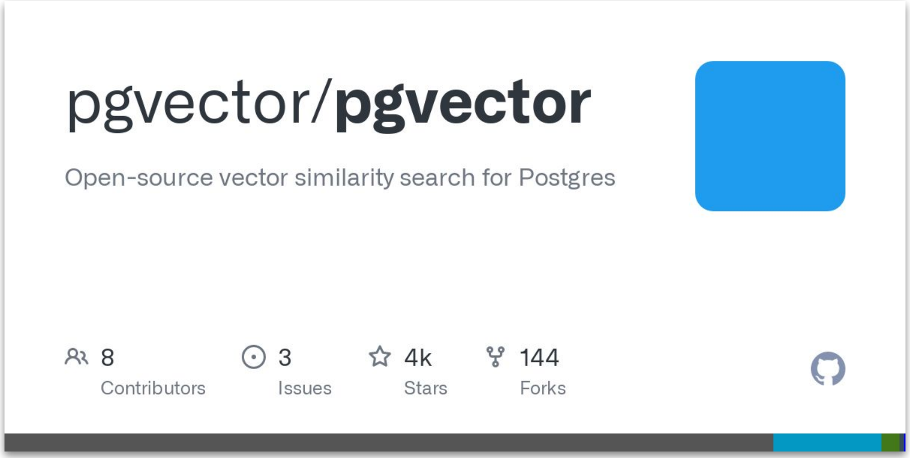
- Co-locate vectors and operational data
- Use a familiar database technology
- Open-source solution

PG Vector Supported databases from Google

- Cloud SQL for PostgreSQL
- AlloyDB (10x faster, 4x larger vectors)

Limitations

- <= 2000 dimensions per embedding if indexed
- Performance degrades at large scale
- Shares resources with operational DB

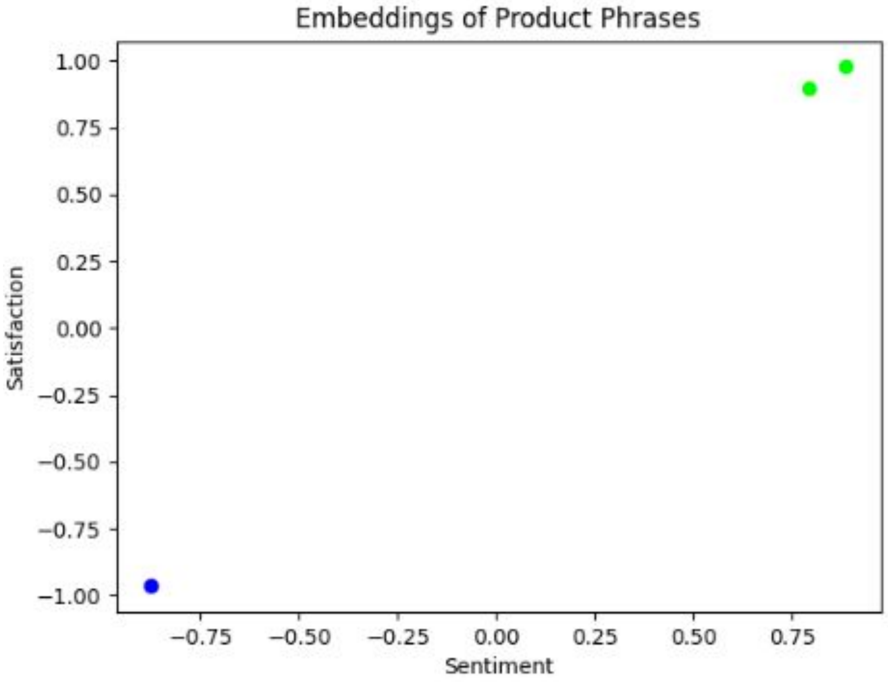


What are Embeddings?

An embedding is a **mathematical representation** of a word, phrase, or other object (pixels in an image, sound waves in an audio file, etc), stored in **vector format**.

Embeddings graphed on the right:

- "I love this product" (green)
- "This product makes my life so easy" (green)
- "This product arrived damaged" (blue)

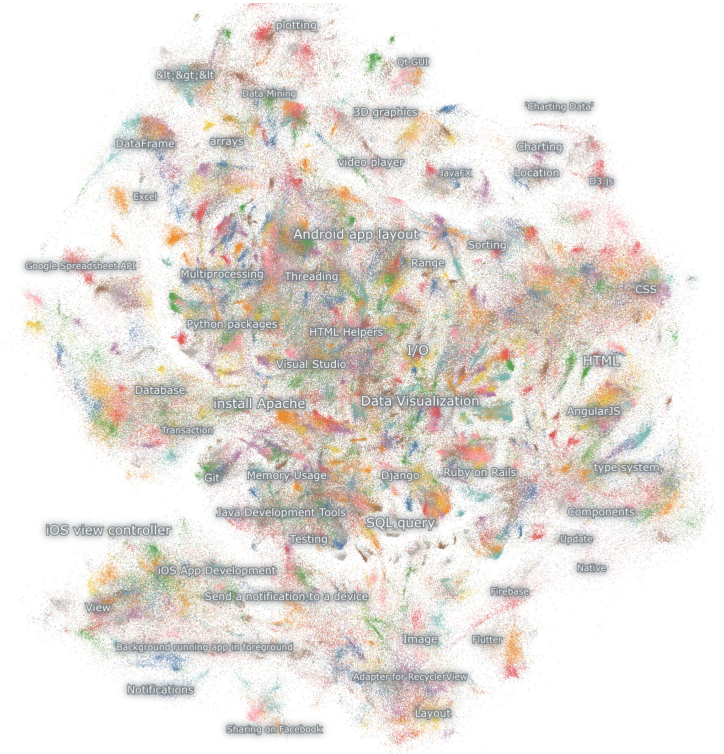


Embeddings space

An **embedding space** is the **x-dimensional** space in which embeddings are defined wherein the **relative distance** between points holds **semantic meaning**.

Embeddings space graphed on the right:

8 million Stack Overflow questions



Embeddings of 8 million Stack Overflow questions
[Visualized by Nomic AI Atlas]

Getting value out of unstructured data with embeddings

Data	Deep learning model	Embedding (vector) representations	Serving index

Embedding Model

[0.2, 0.5, 1.2, ..., 0.4, 0.05, 0.6]

Images, videos, text, songs, time-series, etc.

Pre-trained custom encoders

Vectors of numbers representing the semantic structure of an entity

Similar objects clustered together

A search on the embedding space

What kind of content the **document** or **image** has?

What are **products** similar to this?

Who are **users** who behave like this user?

Any other **music** has the same taste as this?

Is there any **IoT** device outputs similar malfunctional signals?

Vector Embedding in Postgres

With pgvector extension

- Use your database to store and index vector embeddings generated by large language models (LLMs)
- Efficiently find similar items using exact and approximate nearest neighbor search
- Leverage relational database data and features to further enrich and process the data

ANN & KNN

Vector indexes differ from traditional indexes

ANN and KNN
ANN : Approximate Nearest Neighbours KNN: K-Nearest Neighbours

- Vector is a data type
- Without a vector index, you would do an exhaustive "K-Nearest Neighbours" (KNN) search (brute-force)
- ANN-indexes provide an "approximate" answer that is aimed to be "good-enough", but at the fraction of cost

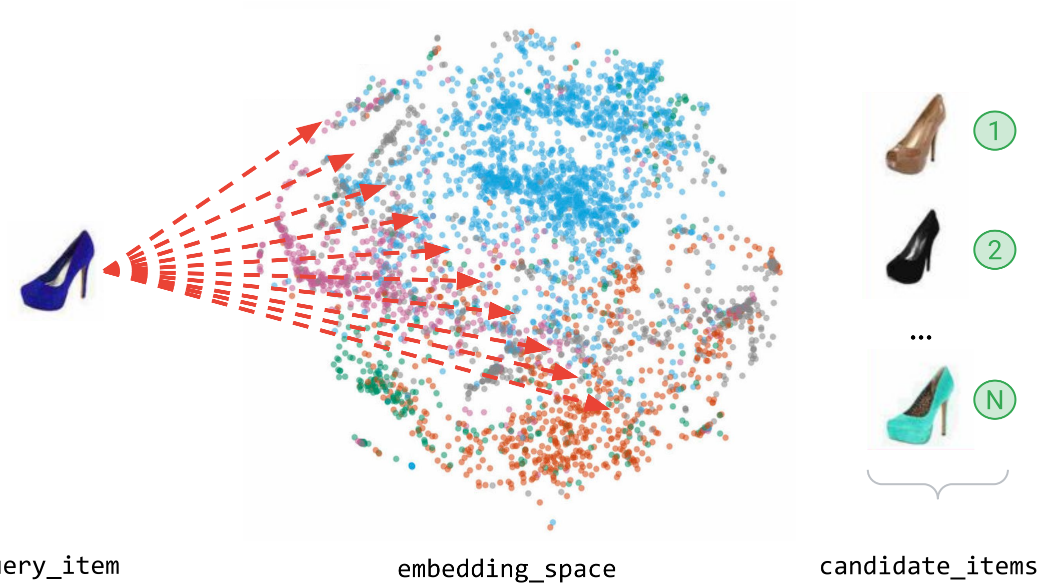
By default, pgvector performs exact nearest neighbor search

Proprietary + Confidential

Retrieval & Similarity Search

Given a query, search a corpus of items for the most relevant candidate item(s)

query → search → rank



Brute force (exhaustive) search infeasible in large database of million or billions of items

wasted computation
⟶ only a small subset is relevant
⟶ real-time ranking is impossible

PG Vector Indexing

Two types of Index supported IVFFLAT & HNSW

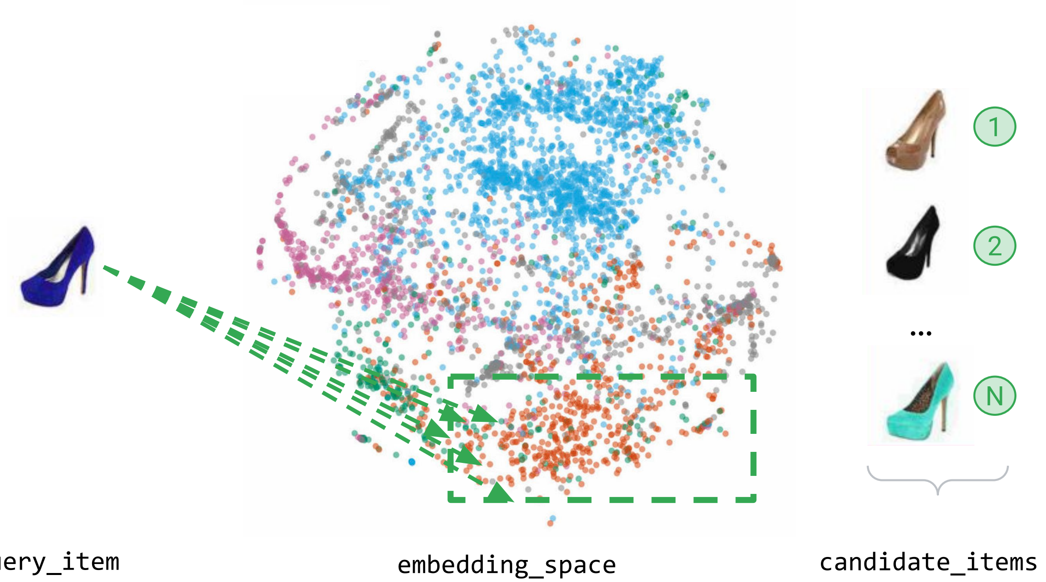
- Inverted File with Flat compression
- Hierarchical Navigable Small Worlds

Proprietary + Confidential

Retrieval & Similarity Search

Given a query, search a corpus of items for the most relevant candidate item(s)

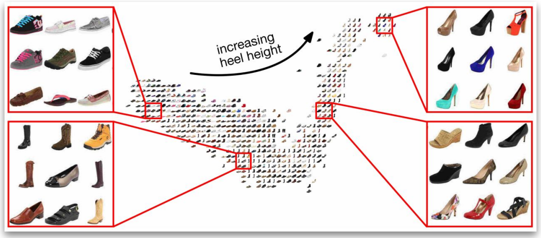
query → search → rank



Approximate Search

index structure optimized for efficient retrieval
→ divides dataset into subsets
→ **limits search to subset** of candidate items (sub-linear)

Construct index in a way that orients similar items closer to each other ★

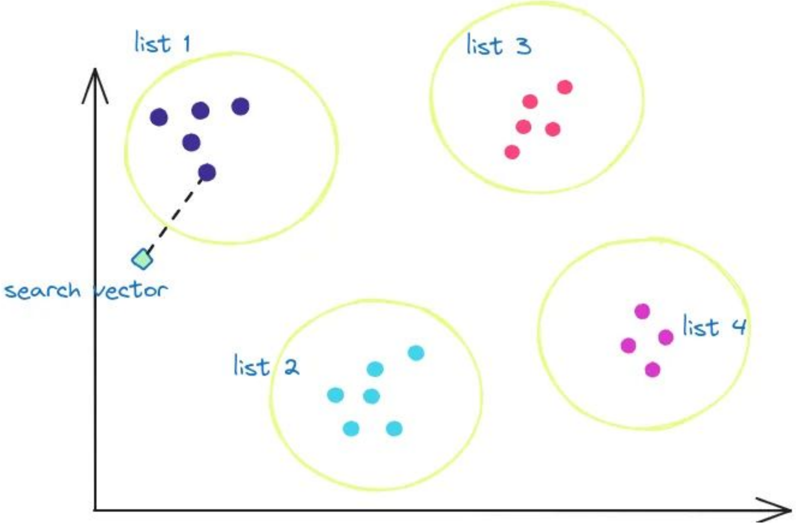


Inverted File with Flat Compression (IVFFlat)

An IVFFlat index divides vectors into **lists**, and then searches a subset of those lists that are closest to the query vector

Benefits

- Simple implementation
- Faster build times
- Uses less memory
- Tunable lists and probes for recall optimization

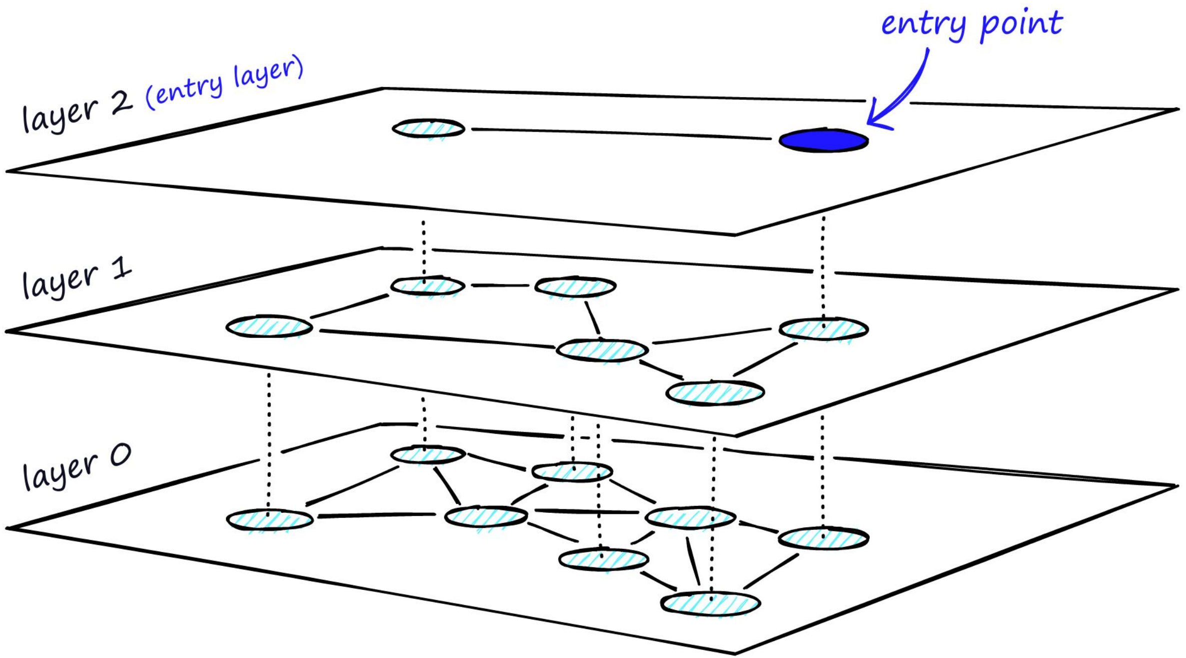


Limitations

- Lower performance (speed / recall)
- Requires a k-means training phase, so you must create the index *after* representative data is loaded.
- May require reindexing as data changes.

HNSW Explained

HNSW maintains multiple layers of NSWs in hierarchical format



Source: *pinecone.io/learn/series/faiss/hnsw/*

Hierarchical Navigable Small Worlds (HNSW)

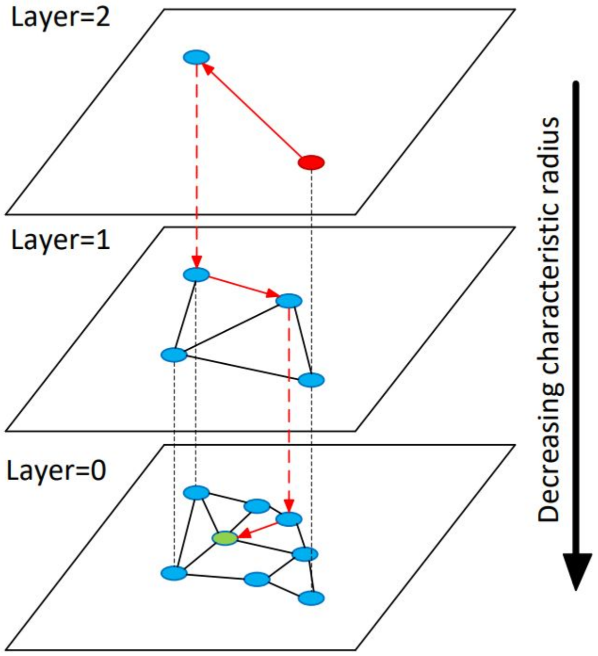
HNSW combines small world graph theory and skip lists to produce a high-performance vector index

Benefits

- Most popular vector similarity search index algorithm due to better query performance.
- No training phase, so index can be created before data is populated.
- Tunable connections per layer and candidate list size

Limitations

- Consumes more storage/memory (indexes are often larger than source data)
- Slower index builds



Source: *bit.ly/3RMJsN3*

Choose the right Index Type

HNSW vs IVFFlat

Query Speed

HNSW is faster

Recall

HNSW typically gives higher recall for same QPS

Caution: Be cognizant of data and query characteristics that can affect recall, e.g.:

* Query Selectivity
* Sparse vs Dense vectors

Index Building

IVFFlat is faster

Memory Usage

IVFFlat is lower

Incremental Data Changes

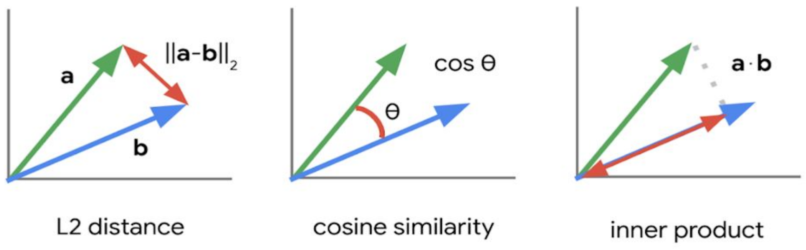
HNSW handles these well

IVFFlat is more sensitive, and may need more frequent re-building

Proprietary + Confidential

Index creation - Options

Distance Metric	pgvector Operator	Measures	When to Choose
Cosine Distance	vector_cosine_ops	Similarity of direction (angle)	Text similarity, high-dimensional data, normalized vectors
L2 Distance	vector_l2_ops	Straight-line distance (direction + magnitude)	Geometric similarity, magnitude of vectors is important, data without normalization
Inner Product	vector_ip_ops	Projection of one vector onto another	Normalized vectors (for efficiency equivalent to cosine), relevance ranking (with normalization), specific embedding models



Calculating vector similarity

Cosine is common and often a good default choice

Proprietary + Confidential

DB Setup

Configuration

vCPUs	Memory	SSD storage
4	32 GB	71 GB

- Enterprise Plus edition
- Data Cache is enabled (375 GB)
- Database version is PostgreSQL 14.17
- Auto storage increase is enabled
- Automated backups are enabled
  Stored in Region: asia-east2 (Hong Kong)
- Point-in-time recovery is enabled

Table Rows = 162,528

Proprietary + Confidential

Table Structure

```
        Column          |       Type        | Collation | Nullable | Default
------------------------+-------------------+-----------+----------+---------
 id                     | character varying |           | not null |
 updated_at_in_sec      | integer           |           |          |
 publisher_name         | character varying |           |          |
 title                  | character varying |           |          |
 title_translated       | character varying |           |          |
 description            | character varying |           |          |
 description_translated | character varying |           |          |
 image_id               | character varying |           |          |
 language               | character varying |           |          |
 country                | character varying |           |          |
 article_url            | character varying |           |          |
 image_url              | character varying |           |          |
 expiry_in_hours        | integer           |           |          |
 uid                    | integer           |           |          |
 title_translated_emb   | vector(768)       |           |          |
 data                   | character varying |           |          |
 data_embedding         | vector(768)       |           |          |
Indexes:
    "seqi_pkey" PRIMARY KEY, btree (id)
```

Proprietary + Confidential

IVFFlat

```
CREATE INDEX ON emb2
USING ivfflat(data_embedding vector_cosine_ops) WITH (lists = 1000);
ERROR:  memory required is 351 MB, maintenance_work_mem is 64 MB
Time: 966.945 ms
genai=>
CREATE INDEX ON emb2
USING ivfflat(data_embedding vector_cosine_ops) WITH (lists = 100);
CREATE INDEX
Time: 10987.383 ms (00:10.987)
```

Proprietary + Confidential

HNSW

With default params (m = 16, ef_construction = 64)

```
genai=> CREATE INDEX ON emb2
USING hnsw(data_embedding vector_cosine_ops);
NOTICE:  hnsw graph no longer fits into maintenance_work_mem after 16755 tuples
DETAIL:  Building will take significantly more time.
HINT:  Increase maintenance_work_mem to speed up builds.
CREATE INDEX
Time: 132424.201 ms (02:12.424)
```

CPU 74%

With params (m = 32, ef_construction = 128)

```
CREATE INDEX ON emb2
USING hnsw(data_embedding vector_cosine_ops)
WITH (m = 32, ef_construction = 128);

NOTICE:  hnsw graph no longer fits into maintenance_work_mem after 14748 tuples
DETAIL:  Building will take significantly more time.
HINT:  Increase maintenance_work_mem to speed up builds.


CREATE INDEX
Time: 466643.649 ms (07:46.644)
genai=>
```

CPU 81%

Proprietary + Confidential

HNSW

HNSW with Default - But for L2 distance

```
genai=> CREATE INDEX ON emb2
USING hnsw(data_embedding vector_l2_ops);
NOTICE:  hnsw graph no longer fits into maintenance_work_mem after 16761 tuples
DETAIL:  Building will take significantly more time.
HINT:  Increase maintenance_work_mem to speed up builds.
CREATE INDEX
Time: 144018.396 ms (02:24.018)
```

HSNW with Default for Inner Product

```
genai=> CREATE INDEX ON emb2
USING hnsw(data_embedding vector_ip_ops);
NOTICE:  hnsw graph no longer fits into maintenance_work_mem after 16755 tuples
DETAIL:  Building will take significantly more time.
HINT:  Increase maintenance_work_mem to speed up builds.


CREATE INDEX
Time: 132719.478 ms (02:12.719)
```

Proprietary + Confidential

HNSW

HNSW L2 & IP - m=32 & ef_construction=128

```
genai=> CREATE INDEX ON emb2
USING hnsw(data_embedding vector_l2_ops)
WITH (m = 32, ef_construction = 128);
NOTICE:  hnsw graph no longer fits into maintenance_work_mem after 14748 tuples
DETAIL:  Building will take significantly more time.
HINT:  Increase maintenance_work_mem to speed up builds.



CREATE INDEX
Time: 512083.698 ms (08:32.084)
```

```
genai=> CREATE INDEX ON emb2
USING hnsw(data_embedding vector_ip_ops)
WITH (m = 32, ef_construction = 128);
NOTICE:  hnsw graph no longer fits into maintenance_work_mem after 14755 tuples
DETAIL:  Building will take significantly more time.
HINT:  Increase maintenance_work_mem to speed up builds.




CREATE INDEX
Time: 467931.197 ms (07:47.931)
genai=>
```

What is wrong with the following statement ?

```
CREATE INDEX ON emb2
USING hnsw(data_embedding
vector_cosine_ops)
WITH (m = 64, ef_construction = 32);
```

Proprietary + Confidential

Query With & Without Index

```sql
SELECT id, publisher_name, data FROM emb2 ORDER BY data_embedding <=> embedding('text-embedding-005',
'delhi capital')::vector LIMIT 10;
```

RESULTS

id	publisher_name	data
6c9a980bc3-55e40558-3907-5...	My Khel Kann...	{"id":"6c9a980bc3-55e40558-3907-5397-bfe6-3fe1bcc96d8f","updated...
ce036ccddc-3e791d43-daf0-5f...	Maha Sports	{"id":"ce036ccddc-3e791d43-daf0-5f5c-b1d7-9f4d8663b7f8","updated_...
13013f6186-e8f9a9f2-f244-568...	My Khel Telugu	{"id":"13013f6186-e8f9a9f2-f244-5686-9418-1a4f3dd29229","updated_...
a270977e8f-2d716a70-7cbb-59...	CricTracker	{"id":"a270977e8f-2d716a70-7cbb-5924-b48e-d9b85c378286","update...
6711b8ad9a-606f2cdf-01b9-5f...	ವಿಸ್ತಾರ ನ್ಯೂಸ್	{"id":"6711b8ad9a-606f2cdf-01b9-5faa-9531-b22953ceccf2","updated...

With Index

```sql
SELECT id, publisher_name, data FROM emb2 ORDER BY data_embedding <=> embedding('text-embedding-005',
'delhi capital')::vector LIMIT 10;
```

RESULTS

id	publisher_name	data
s-7a7a9m025s7a3g-bbec930...	msn Austria	{"id":"s-7a7a9m025s7a3g-bbec9303-2b93-5436-8b20-45290fd1c651","...
cf747cc039-f21c76fb-5f9d-5b...	オリコン	{"id":"cf747cc039-f21c76fb-5f9d-5baf-9f59-425a38cc2d87","updated_...
cf747cc039-bcdd8f0b-d681-5...	オリコン	{"id":"cf747cc039-bcdd8f0b-d681-5f73-8703-e5d980460624","updated...
1bd968f18e-ccb26578-ed58-5...	ABP Nadu	{"id":"1bd968f18e-ccb26578-ed58-56a1-8fb3-752d168acc2e","updated...

Without Index

At the Time of Creation HNSW & IVFFLAT

Parameter	Index Type	Impact on Increasing the parameter value	
		Benefit	Cost
m	HNSW	Give higher recall and/or Support higher dimensionality	Increased memory usage and Query time
ef_construction	HNSW	Improved recall (up to an extent)	Increased index build time
Lists	IVFFLAT	More the value higher the accuracy	Higher value add build time

Run Time HNSW & IVFFLAT

Parameter	Index Type	Impact on Increasing the parameter value	
		Benefit	Cost
Probe	IVFFLAT	Can set per query	More the probe higher the execution time
ef_search	HNSW	Higher recall	Lower QPS

Reference index parameters

Typical values - for reference

- `M`: [8, 16, 24, 32, 48, 64]
- `ef_construction`: [64, 128, 256, 512]
- `Target recall =` 0.95
- List = 1000
- probe= 5

Size your instance for Vectors

Vector data size

Guiding calculations for your vector data

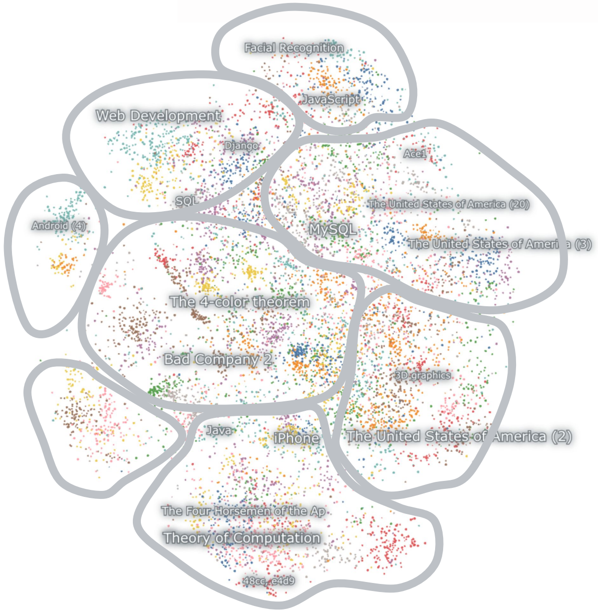
- #Dimensions of the model used for embedding generation
- Vertex AI text-embedding-005 - 768
- OpenAI text-embedding-3-small - 1536
- Per vector size: d * 4 + 8, where d = #dimensions

Which new index type Google has introduced for vector data in AlloyDB?

ScaNN

Scalable Approximate Nearest Neighbor

- ANN separates embeddings space into clusters, enabling fast and scalable search
- ScaNN: ANN library published by Google in 2020
- Foundation for many Google services such as Search, Play, Youtube



https://github.com/google-research/google-research/tree/master/scann

AlloyDB AI vector search performance

AlloyDB AI's ScaNN index compared to the HNSW index in standard PostgreSQL

up to
4x
faster vector queries

up to
8x
faster index creation

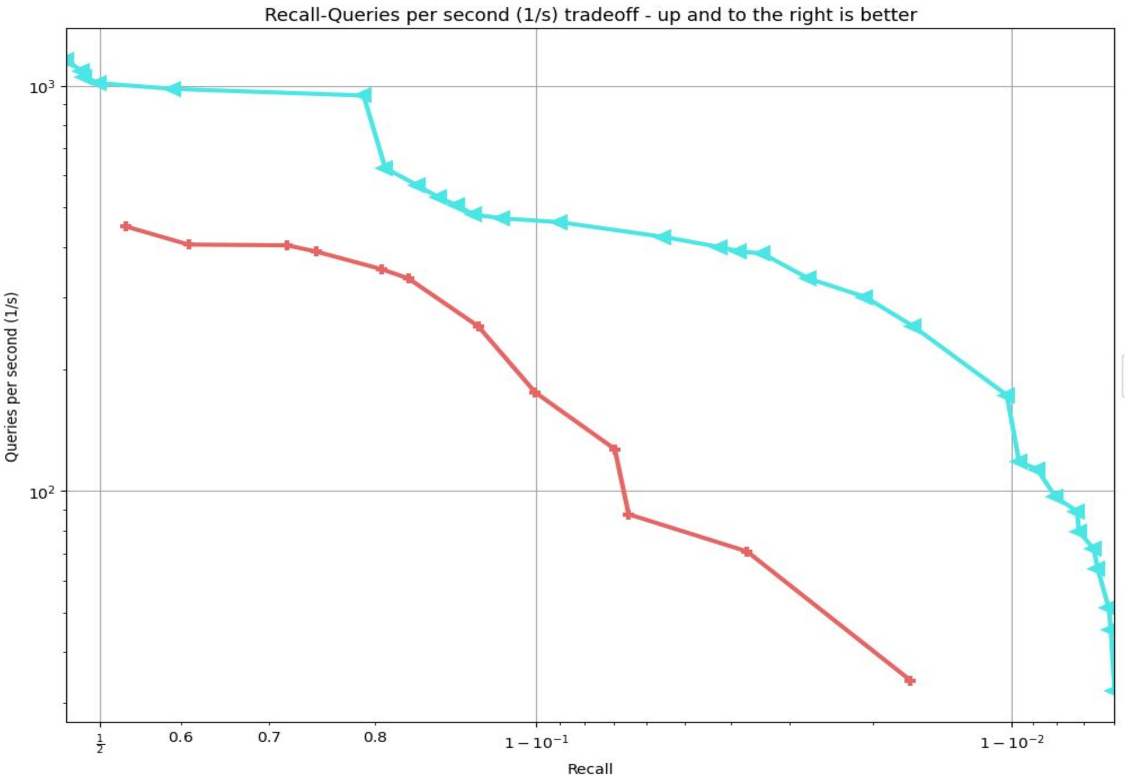
up to
10x
higher write throughput

typically uses
3-4x
less memory

Source: Google Cloud performance tests, March 2024

ScaNN for AlloyDB Performance

- Glove-100 recall/throughput curve
  - ScaNN for AlloyDB
  - pgvector HNSW



Tips for Index Building

Memory

HNSW is memory intensive. So:

*`maintenance_work_mem` = memory estimation formulae shared earlier + sufficient buffer

* ❗ Caution: As soon as the graph spills over to disk, you will see drastic slowdown in index build. Watch out for pgvector warnings at build time

Parallelize

Pgvector added support for parallel index build in 0.6.0

* SET `max_parallel_maintenance_workers` = #vCPUs

Regression Tests

If building on production instances, it is easy to consume a lot of resources (CPU, memory) and overload the server.

Perf tests!

Ref: pgvector docs - https://github.com/pgvector/pgvector/pull/422/files

Day 2 Operations

Track index Usage and Recall:

- Are your indexes being used?
- Is your recall drifting?
- Have my query shapes changed?

Handling Data Drift

- Has my vector distribution changed? (Esp IVFFlat)
- Have I switched embedding model, or model version?

Reindexing to retain target Recall-QPS characteristics

- Or do I just need a vacuum cleaner? (Esp HNSW)

Proprietary + Confidential

Something not right here?

```
         Column          |       Type        | Collation | Nullable | Default
-------------------------+-------------------+-----------+----------+---------
 id                      | character varying |           | not null |
 updated_at_in_sec       | integer           |           |          |
 publisher_name          | character varying |           |          |
 title                   | character varying |           |          |
 title_translated        | character varying |           |          |
 description             | character varying |           |          |
 description_translated  | character varying |           |          |
 image_id                | character varying |           |          |
 language                | character varying |           |          |
 country                 | character varying |           |          |
 article_url             | character varying |           |          |
 image_url               | character varying |           |          |
 expiry_in_hours         | integer           |           |          |
 uid                     | integer           |           |          |
 title_translated_emb    | vector(768)       |           |          |
 data                    | character varying |           |          |
 data_embedding          | vector(768)       |           |          |
Indexes:
    "emb2_pkey" PRIMARY KEY, btree (id)
    "emb2_data_embedding_idx" hnsw (data_embedding vector_l2_ops) WITH (m='32', ef_construction='128')
    "emb2_data_embedding_idx1" hnsw (data_embedding vector_ip_ops) WITH (m='32', ef_construction='128')
    "emb2_data_embedding_idx2" ivfflat (data_embedding vector_cosine_ops)
```

Thank you