Beyond Keywords: AI-powered Text Search with pgvector for PostgreSQL

Belma Canik

(She/Her)
Senior Database Specialist TAM

Agenda

GenAI

Vector concept

PostgreSQL as vector store with pgvector

Demo lab: AI powered similarity search using pgvector

Questions

Generative AI

Artificial intelligence (AI)

Any technique that allows computers to mimic human intelligence using logic, if-then statements, and machine learning

Machine learning (ML)

A subset of AI that uses machines to search for patterns in data to build logic models automatically

Deep learning (DL)

A subset of ML composed of deeply multi-layered neural networks that perform tasks like speech and image recognition

Generative AI

Powered by **large models** that are pre-trained on vast corpora of data and commonly referred to as **foundation models (FMs)**

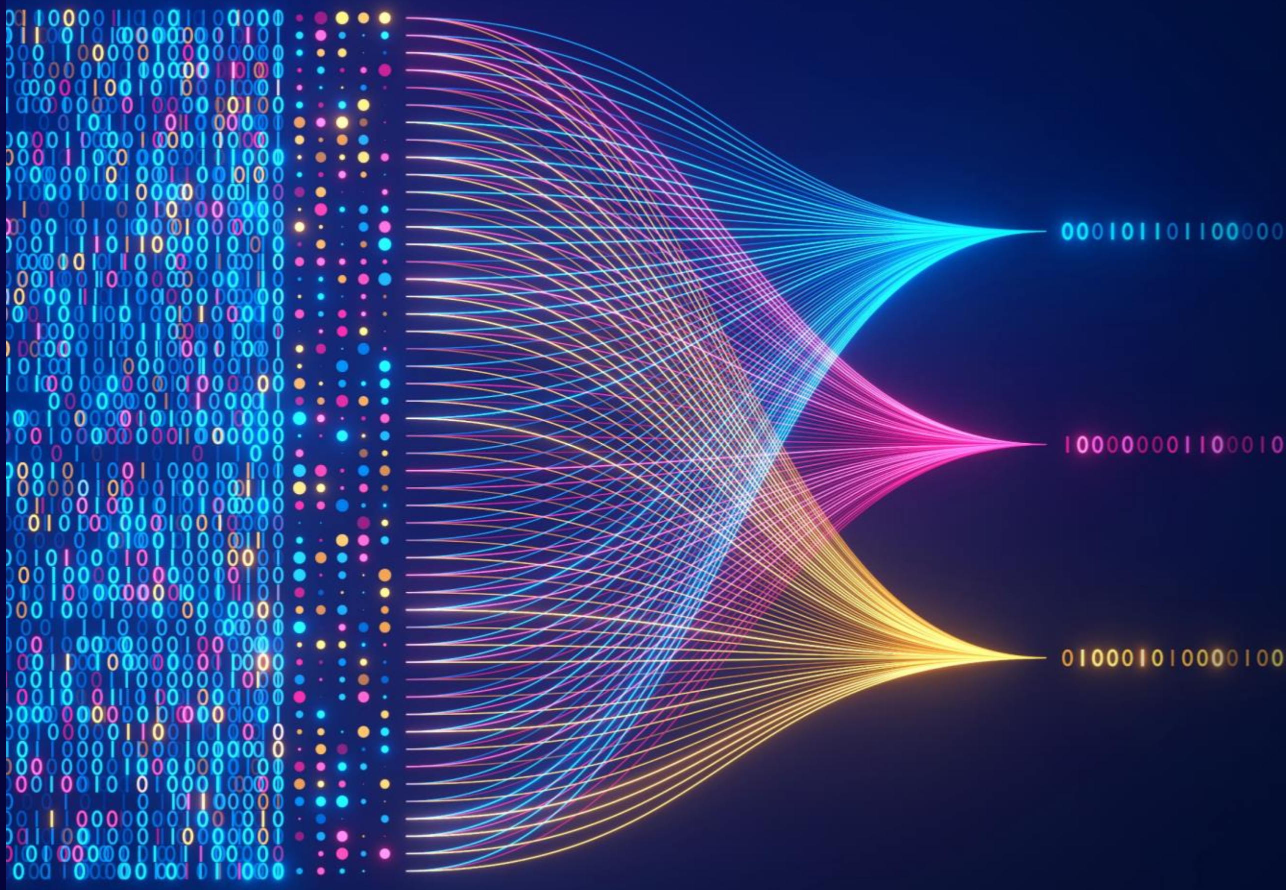
Generative AI is powered by foundation models (FMs)

Pre-trained on vast amounts of unstructured data

Contain large number of parameters that make them capable of learning complex concepts

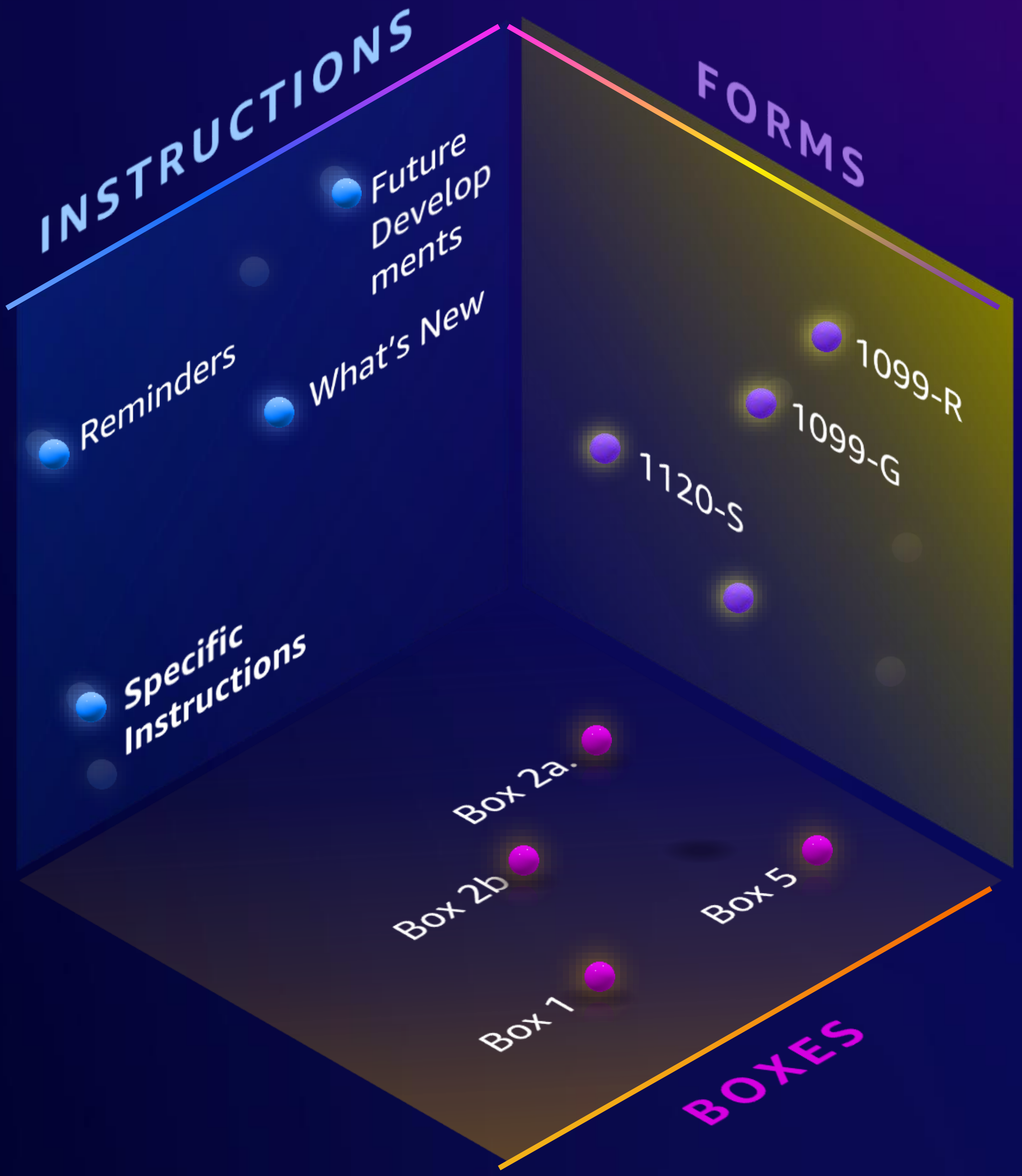
Can be applied in a wide range of contexts

Customize FMs using your data for domain-specific tasks



Vector space

VECTORS ARE ORGANIZED IN CLUSTERS



What is a **vector embedding** ?

- A numerical representation of words or sentences, used in NLP
- NLP models can easily perform tasks such as querying, classification, and applying machine learning algorithms on textual data

"I am at PG Conference 2025"

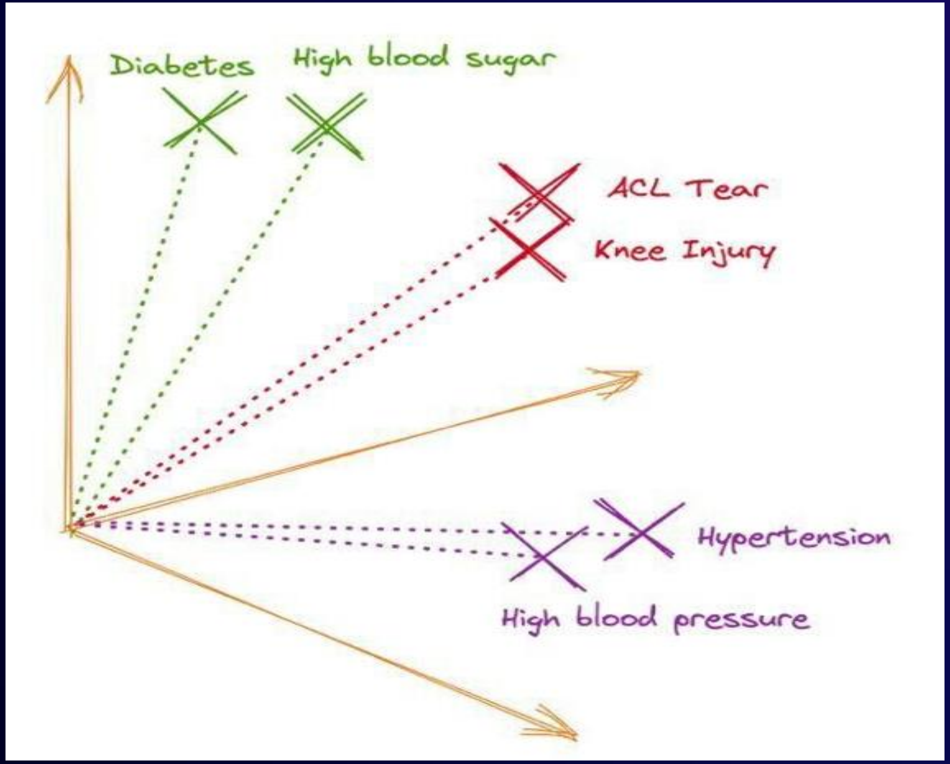
[0.743, 0.720, -0.325, 0.195, 0.835, -0.945]

n-dimensional vector

What are vector embeddings?

Unstructured data has to be vectorized into vectors to be used in generative AI applications

An example of Generated embedding in vector space



ˈpivədl

PostgreSQL as a vector store

Why use PostgreSQL for vector searches?

- Existing client libraries work without modification
- Convenient to co-locate app and AI/ML data in same database
- PostgreSQL acts as persistent transactional store while working with other vector search systems

Note: Postgres, PostgreSQL, and the Slonik Logo are trademarks or registered trademarks of the PostgreSQL Community Association of Canada, and used with their permission

Native vector support and challenges

ARRAY data type

Multiple data types (int4, int8, float4, float8)
"Unlimited" dimensions
No native distance operations
Can add using Trusted Language Extensions + PL/Rust
No native indexing

Cube data type

- float8 values
- Euclidean, Manhattan, Chebyshev distances
- K-NN GiST index – exact nearest neighbor search
- Limited to 100 dimensions

What is pgvector?

- An open-source extension
- support for storage, indexing , searching, metadata with choice of distance

vector data types

Co-locate with embeddings

Exact nearest neighbor
Approximate nearest neighbor (ANN)

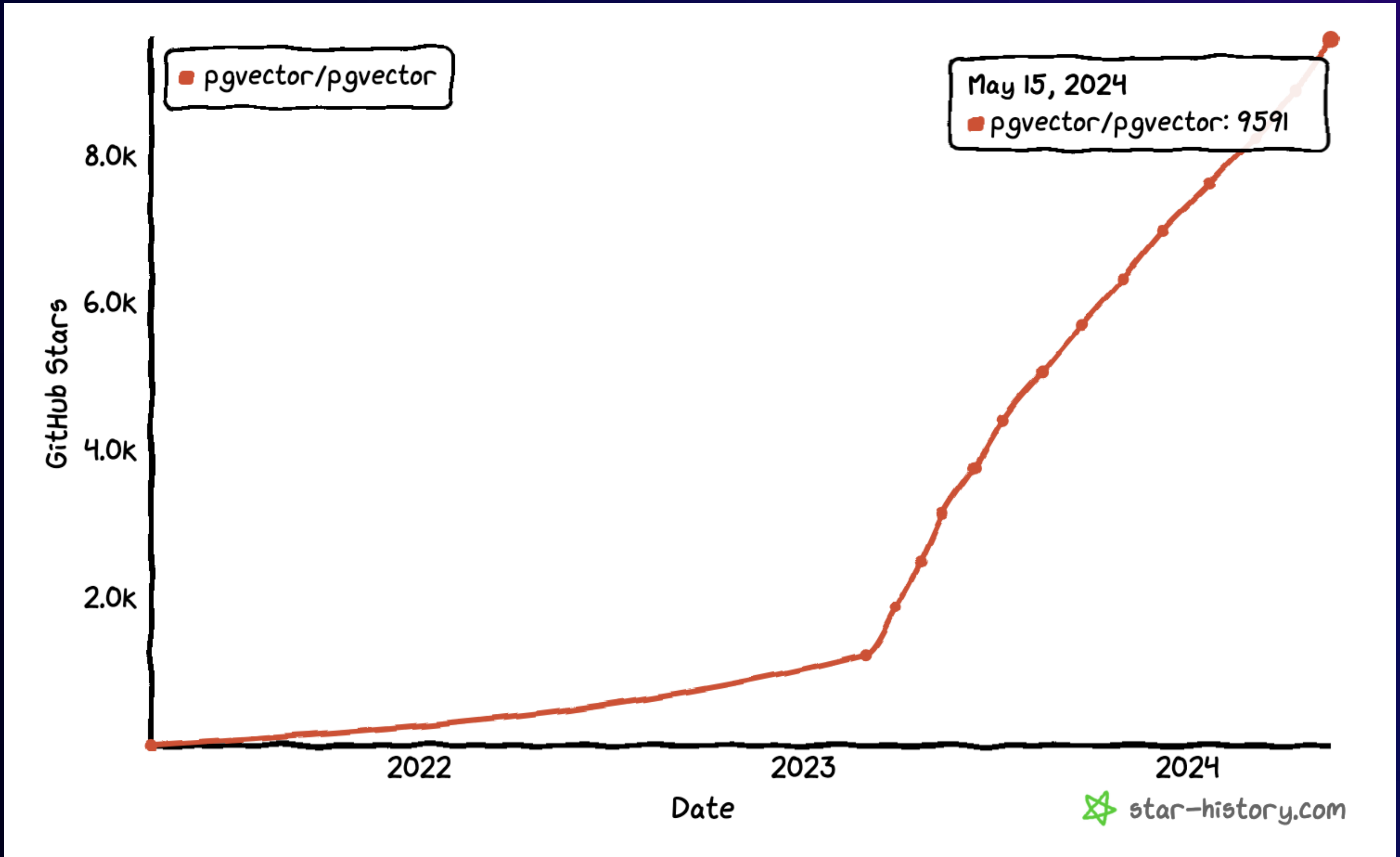
Supports IVFFlat/HNSW indexing

Distance operators (<->, <=>, <#>, <+>, <~>, <%>)

github.com/pgvector/pgvector
Note: <+>, <~>, and <%> operators available only from pgvector version 0.7.0

Aurora PostgreSQL 15.3, 14.8, 13.11, 12.15 and higher
Amazon RDS PostgreSQL 15.2 and higher

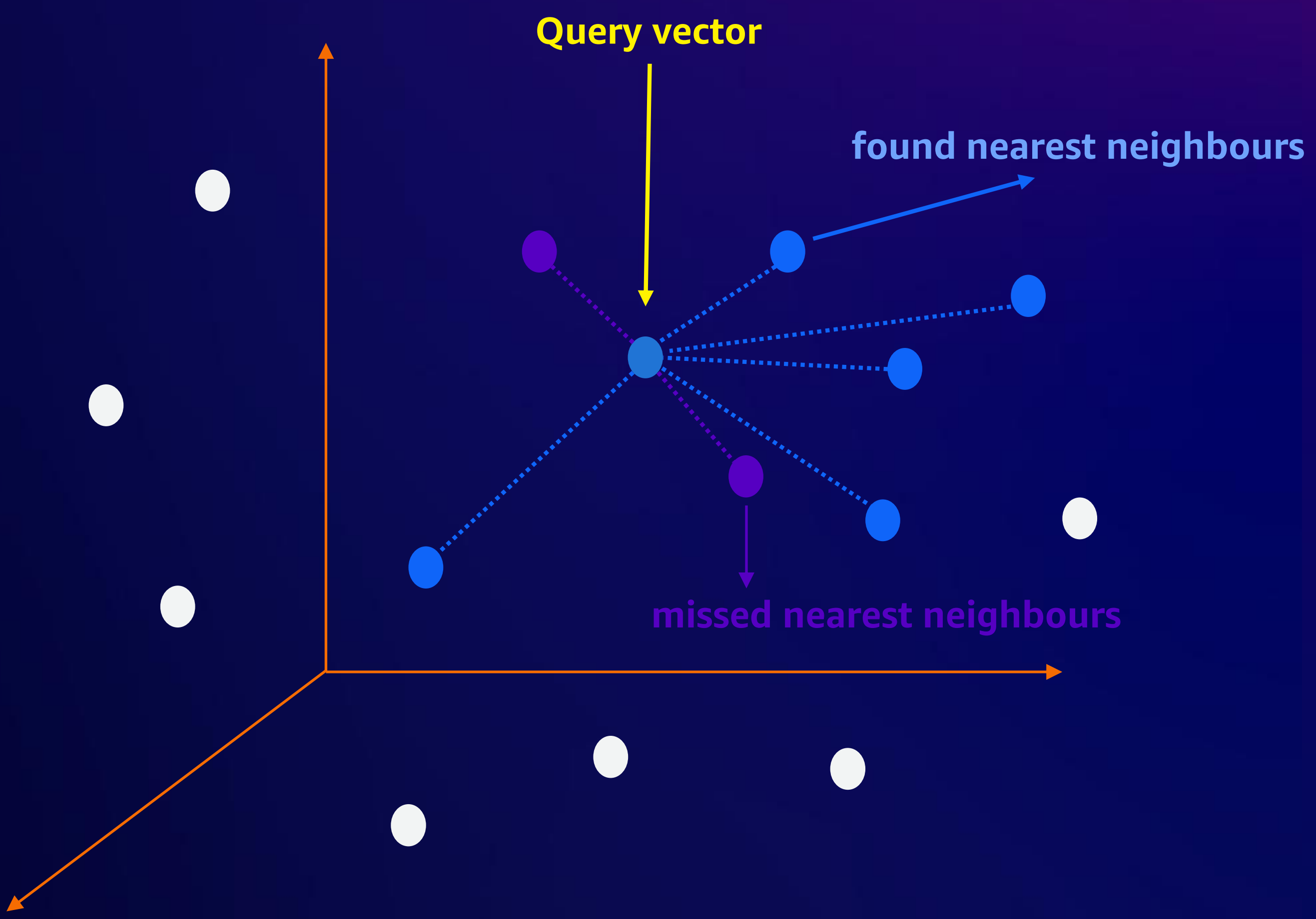
Pgvector Popularity



https://github.com/pgvector/pgvector

Approximate nearest neighbor (ANN)

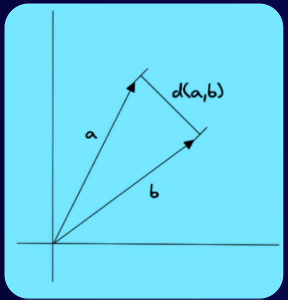
- Find similar vectors without searching all of them
- Faster than exact nearest neighbor
- “Recall” – % of expected results



pgvector: offers distance operations

Euclidean (L2) (vector_l2_ops)

Useful for counts / measurements
Recommendation Systems

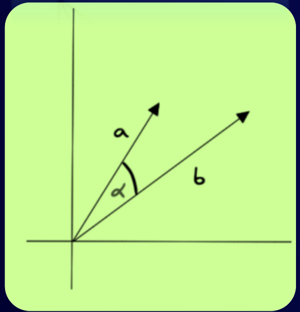


$$d(\mathbf{p},\mathbf{q}) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

<->

Cosine Similarity (vector_cosine_ops)

Useful for semantic search and document classification

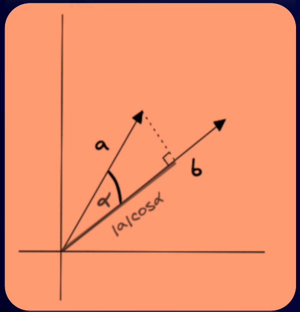


$$sim(\mathbf{a},\mathbf{b}) = \frac{\mathbf{a}\cdot\mathbf{b}}{||\mathbf{a}||\cdot||\mathbf{b}||}$$

<=>

Dot Product (vector_ip_ops)

Useful for collaborative filtering



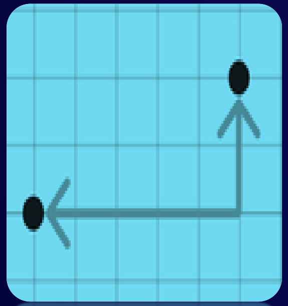
$$\mathbf{a}\cdot\mathbf{b} = |\mathbf{a}||\mathbf{b}|cos\alpha$$

<#>

pgvector: new operations from version 0.7.0

Manhattan Distance (L1) aka Taxicab (vector_l1_ops)

HNSW indexing
Useful for city navigation / speech recognition and image processing

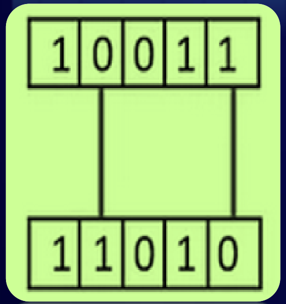


$$Manhattan = \sum_{i=1}^{n} |x_i - y_i|$$

<+>

Hamming (binary vectors) (bit_hamming_ops)

Useful for error correction / dissimilarity between two binary vectors or strings

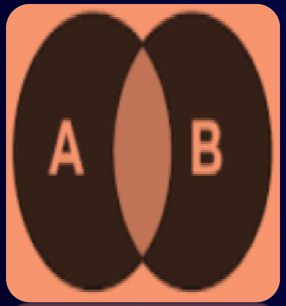


sum(vect0r1 != vector2)

<~>

Jaccard Distance(binary vectors) (bit_jaccard_ops)

Image recognition with labelled data / Compare text patterns in documents based on the overlap of words.



$$Jaccard = \frac{Intersection\ (A, B)}{Union\ (A, B)}$$

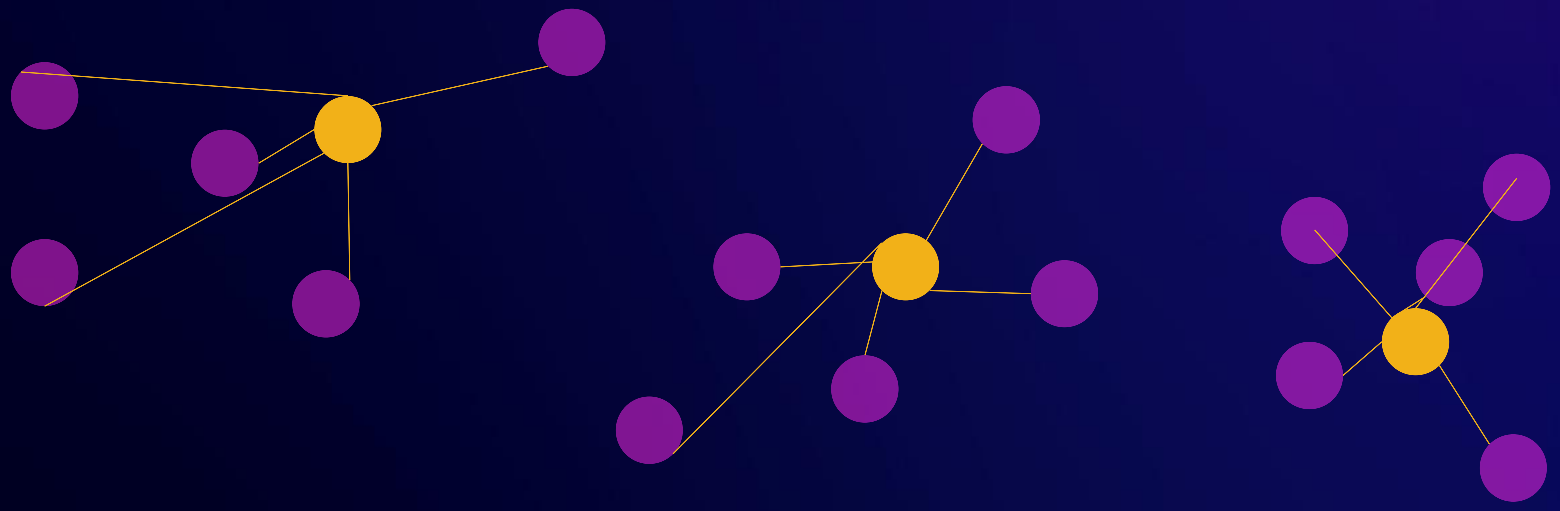
<%>

Inverted File with Flat Compression (IVFFlat) Index

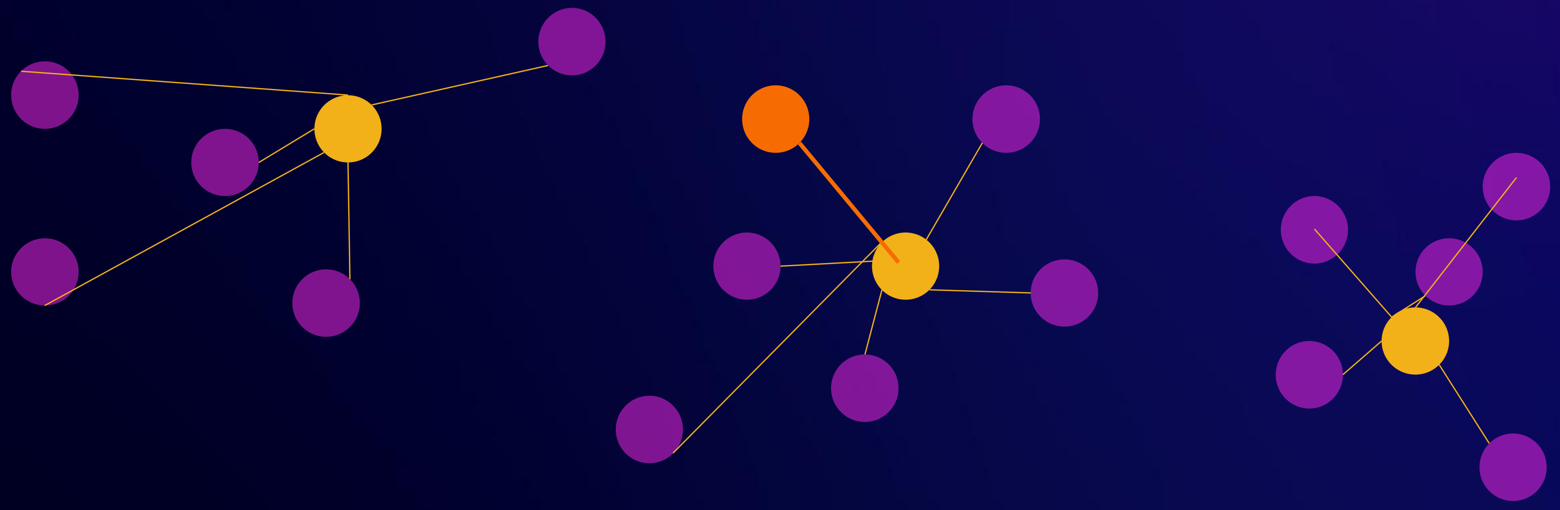
Building an IVFFlat index

Building an IVFFlat index: Find centers

Building an IVFFlat index: Assign lists

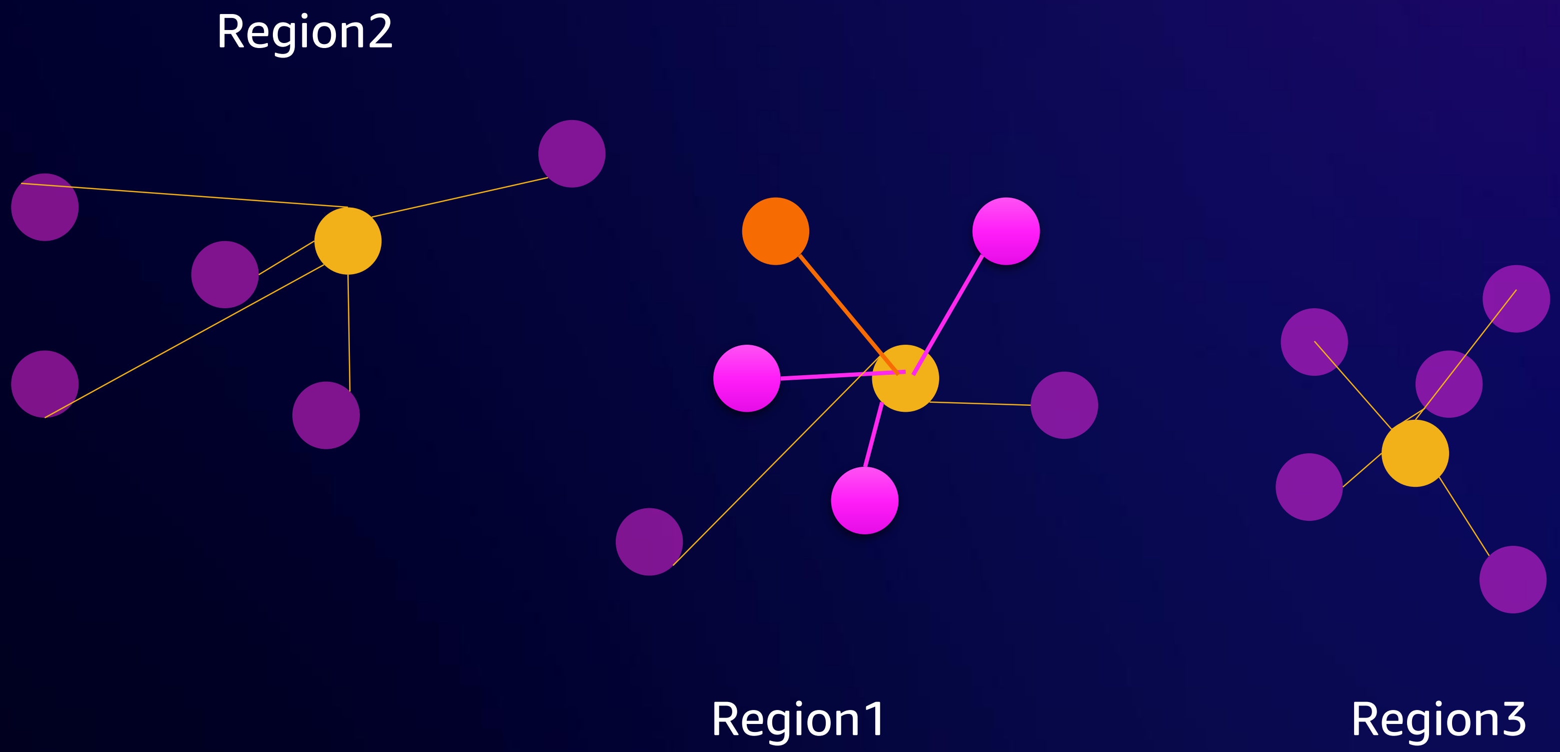


Querying an IVFFlat index



```
SELECT id FROM products ORDER BY $1 <-> embedding LIMIT 3
```

Querying an IVFFlat index




```
SELECT id FROM products ORDER BY $1 <-> embedding LIMIT 3
```

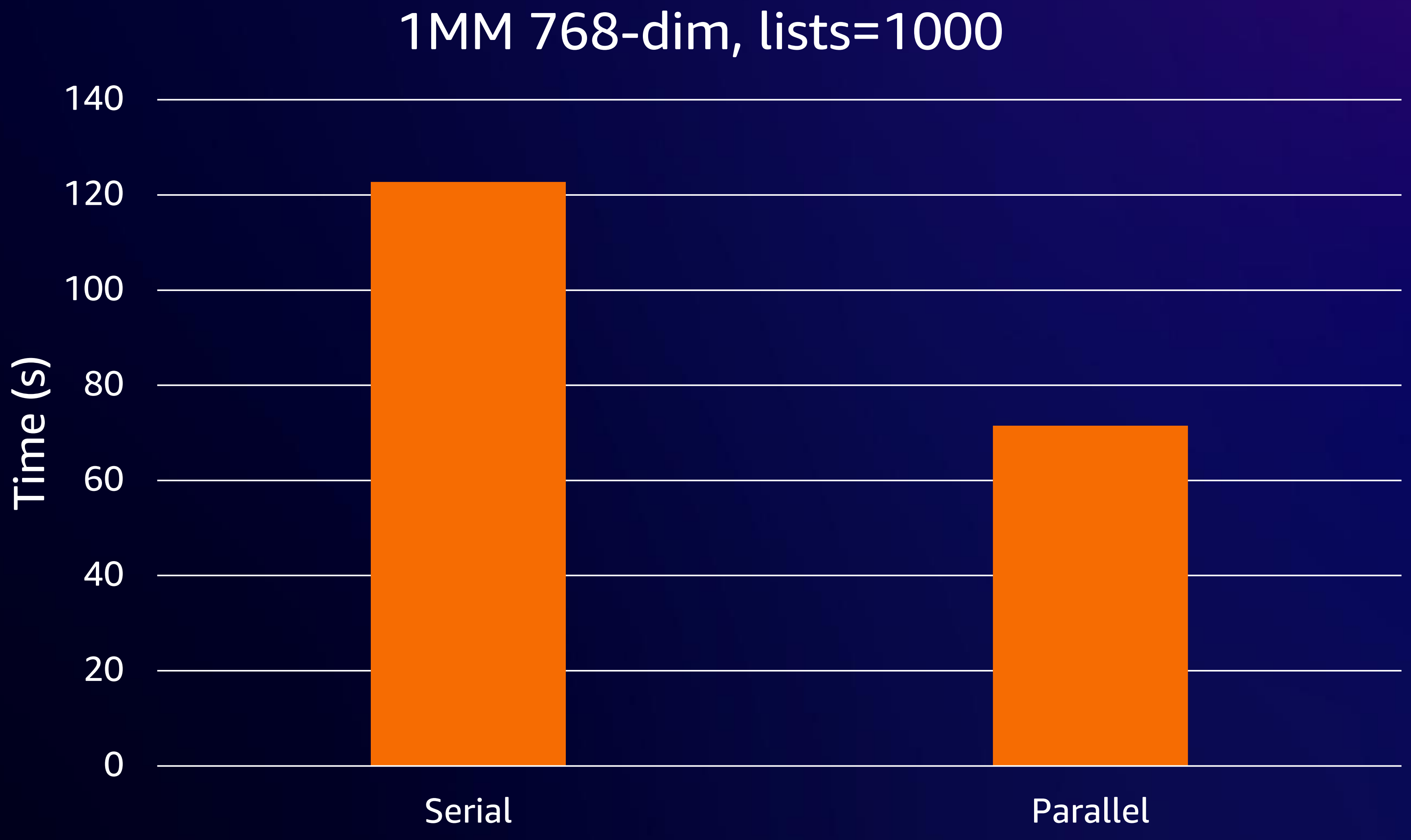
IVFFlat query parameters

- `ivfflat.probes (1 by default)`
  - Number of lists to search during a query
  - More lists leads to more relevant results
  - More lists requires more time
  - probes = sqrt(lists)

Best practices for building IVFFlat indexes

- Choose value of lists to maximize recall but minimize effort of search
  - < 1MM vectors: # vectors(rows) / 1000
  - > 1MM vectors: √(# vectors(rows))
- May be necessary to rebuild when adding/modifying vectors in index
- Use parallelism to accelerate build times
- Increase maintenance_work_mem for faster index creation
- Faster build times, less memory but lower query performance

Using parallelism to accelerate IVFFlat builds



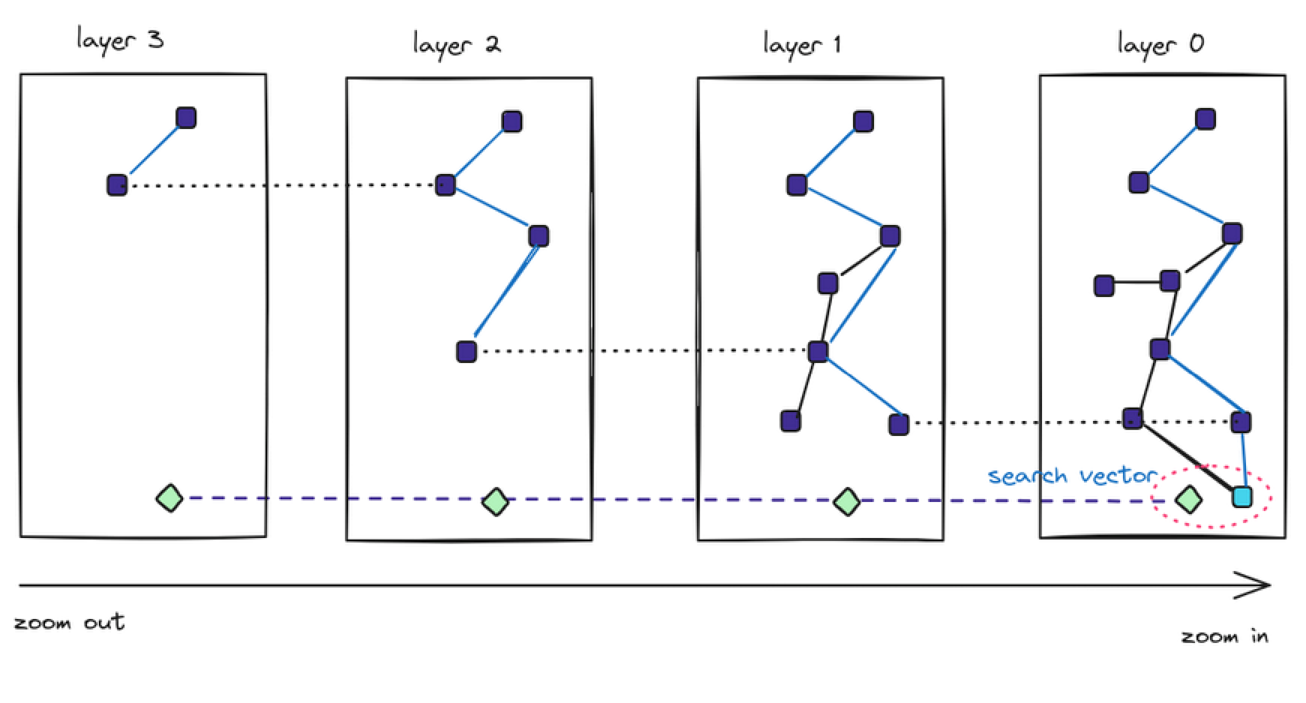
Performance strategies for IVFFlat queries

- Increasing `ivfflat.probes` increases recall, decreases performance
- Lowering `random_page_cost` on a per-query basis can induce index usage
- Set `shared_buffers` to a value that keeps data (table) in memory
- Increase `work_mem` on a per-query basis

Hierarchical Navigable Small World (HNSW) Index

HNSW (Hierarchical Navigable Small Worlds)

- a multi-layered graph-based indexing
- the skip lists and navigable small world algorithms



HNSW creates a graph with multiple layers

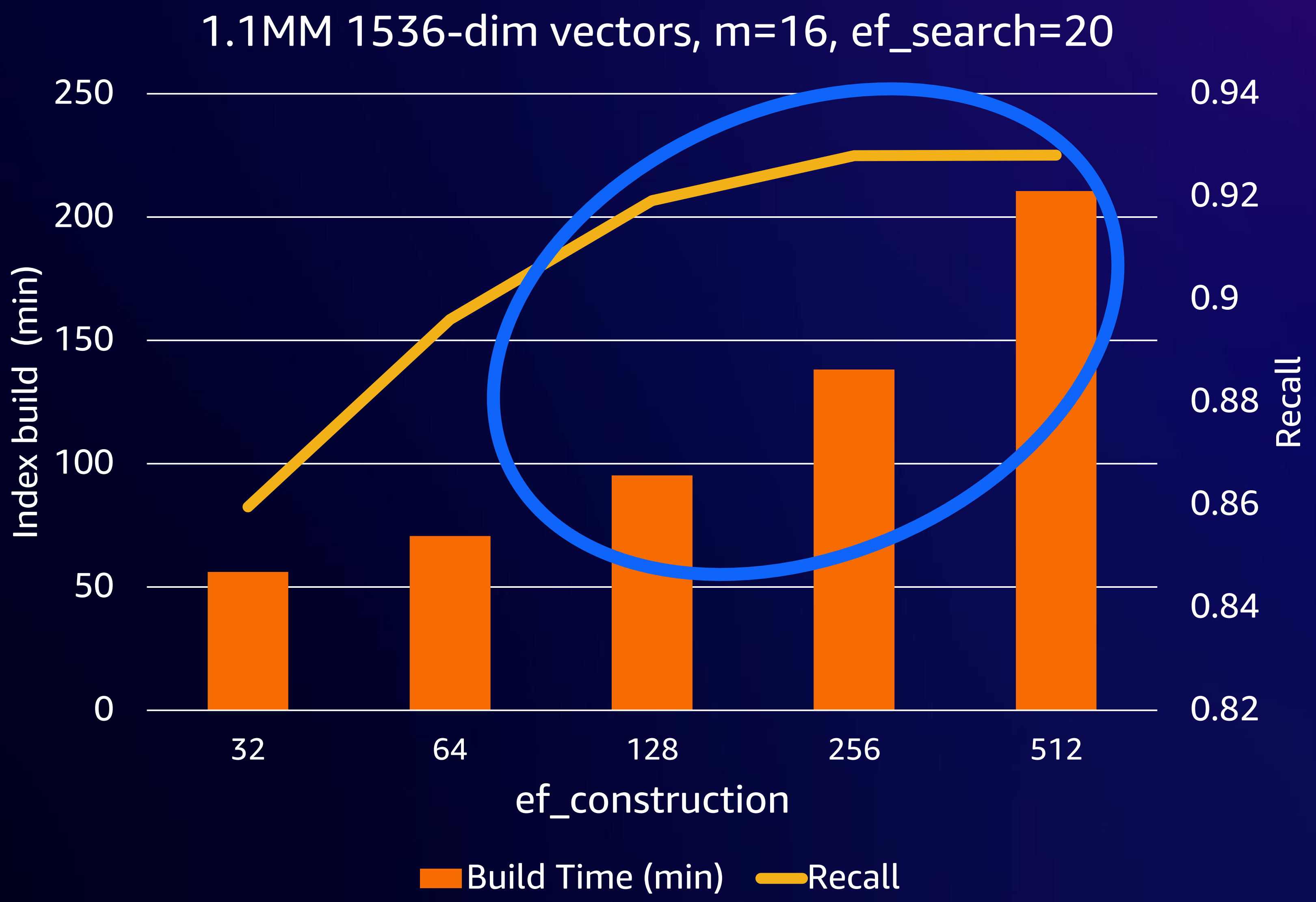
Source: https://tembo.io/blog/vector-indexes-in-pgvector/#hnsw

https://jkatz05.com/post/postgres/pgvector-hnsw-performance/

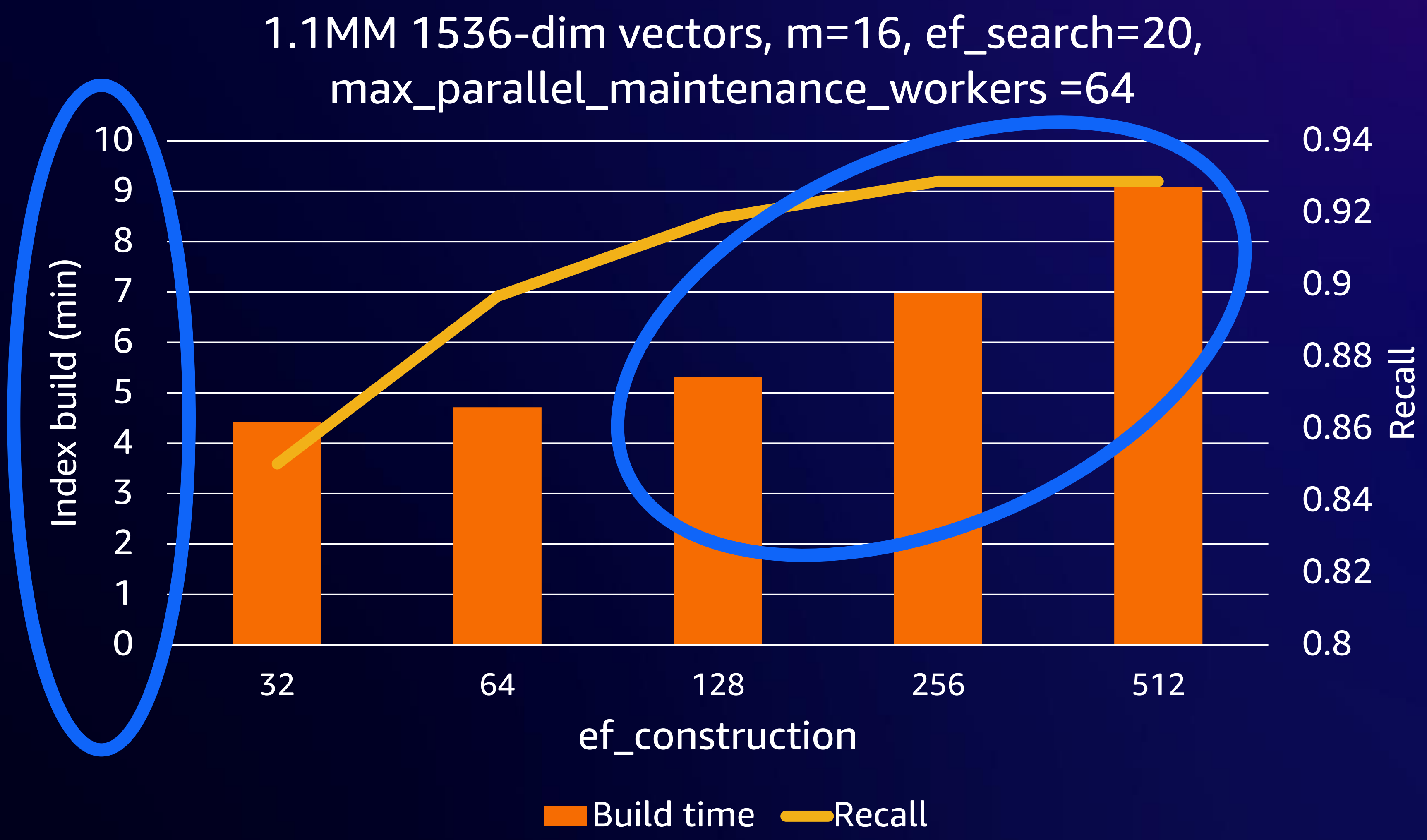
HNSW index building parameters

- `m`
- Maximum number of bidirectional links between indexed vectors
  - Default: 16
- `ef_construction`
  - Number of vectors to maintain in “nearest neighbor” list
  - Default: 64

Choosing m and ef_construction (serial)



Choosing m and ef_construction (parallel)



Note: Paralleism supported from version 0.6.0

Best practices for HNSW indexes

Building HNSW indexes

- Default values `(m=16,ef_construction=64)` usually work
- (pgvector 0.5.1) Start with empty index and use concurrent writes to accelerate builds

Performance strategies

- Index building has biggest impact on performance/recall
- Increasing `hnsw.ef_search( default 40)` increases recall, decreases performance

Which index do I choose?

If you care more about index size, then choose IVFFlat

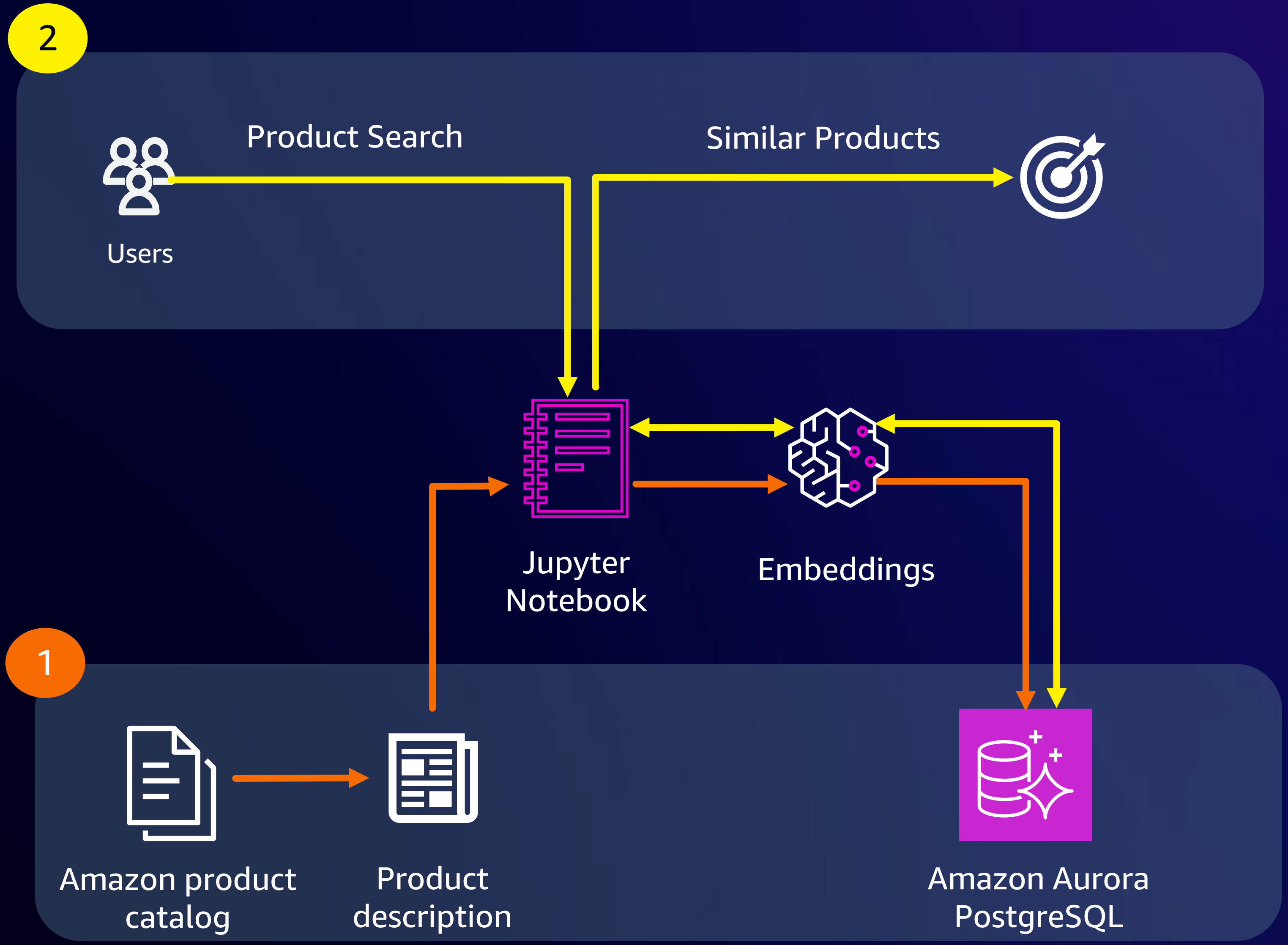
If you care more about index build time, then select IVFFlat

If you care more about high performance/recall, then choose HNSW

If you expect vectors to be added or modified, then select HNSW

Demo lab: AI powered similarity search using pgvector

How vector embeddings are used



Questions

Thank You

Belma Canik
https://www.linkedin.com/in/belma-canik-38b66648/