From Queries to Pints

Building a Beer Recommendation System with pgvector

Andrzej Nowicki

FOSDEM 2025

THE CONTENT OF THIS TALK IS INTENDED FOR INFORMATIONAL AND ENTERTAINMENT PURPOSES ONLY.
ENJOY ALCOHOLIC BEVERAGES RESPONSIBLY AND ALWAYS CONSUME ALCOHOL IN MODERATION.

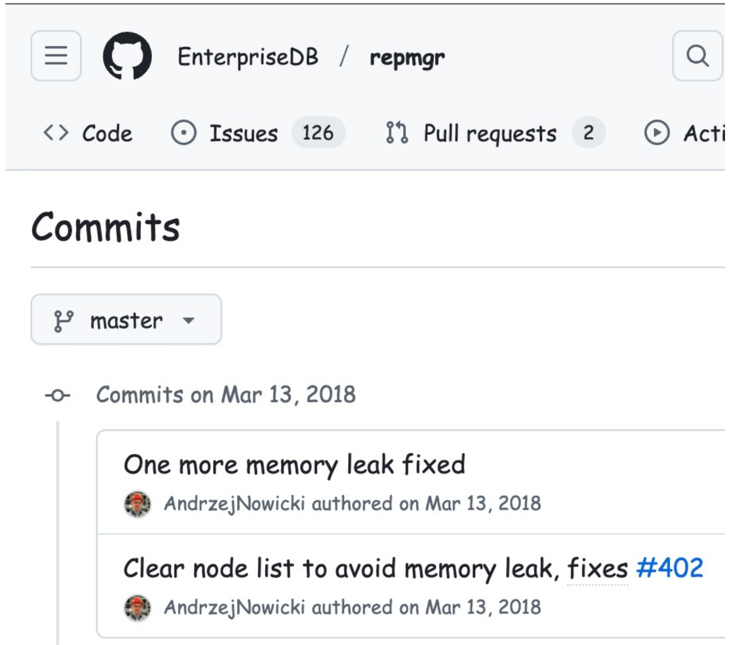
Andrzej Nowicki

12 years of Oracle DB exp, 8 years of PostgreSQL
Database Engineer @ CERN since 2020

andrzejnowicki

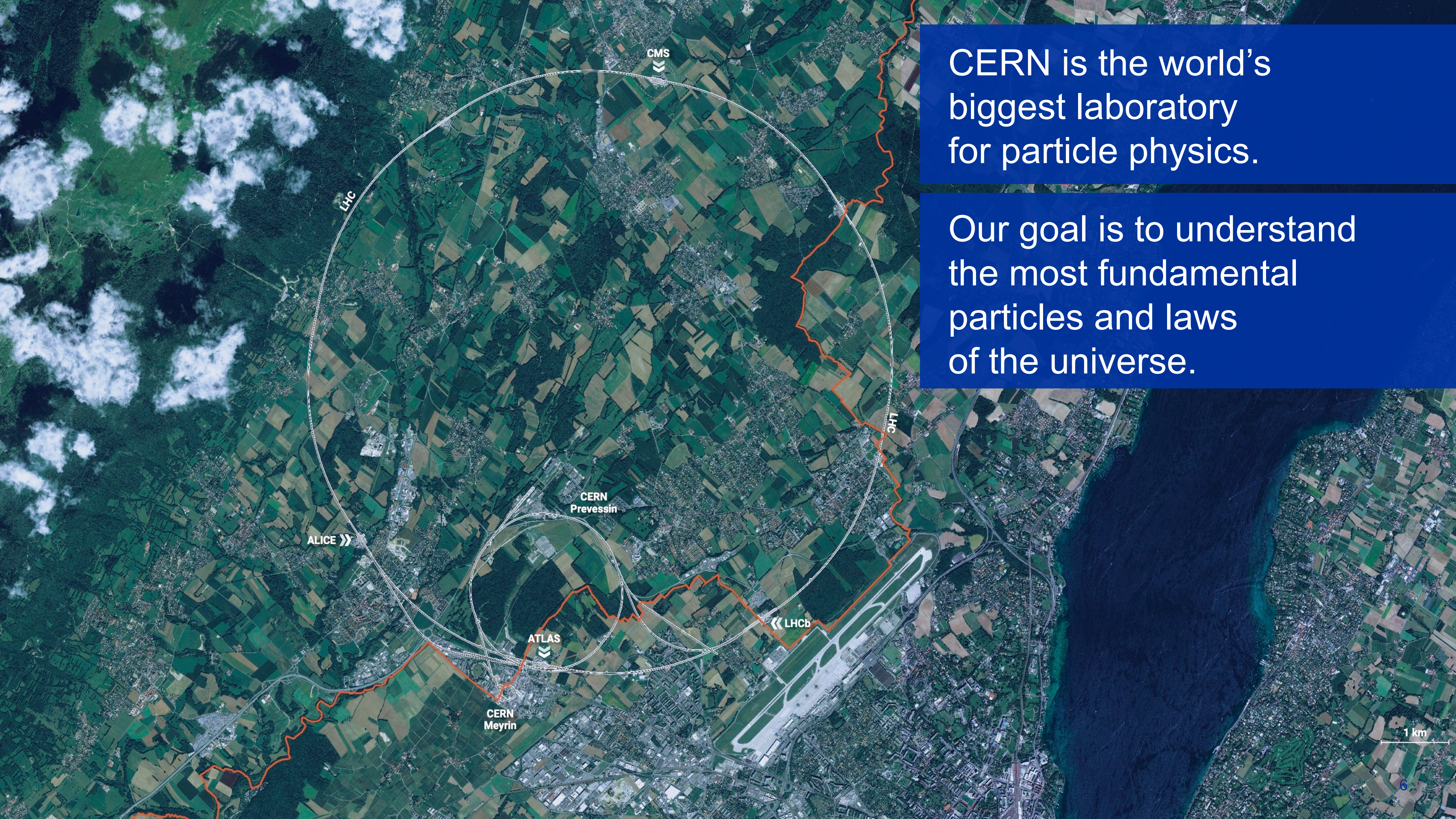
andrzej.nowicki@cern.ch

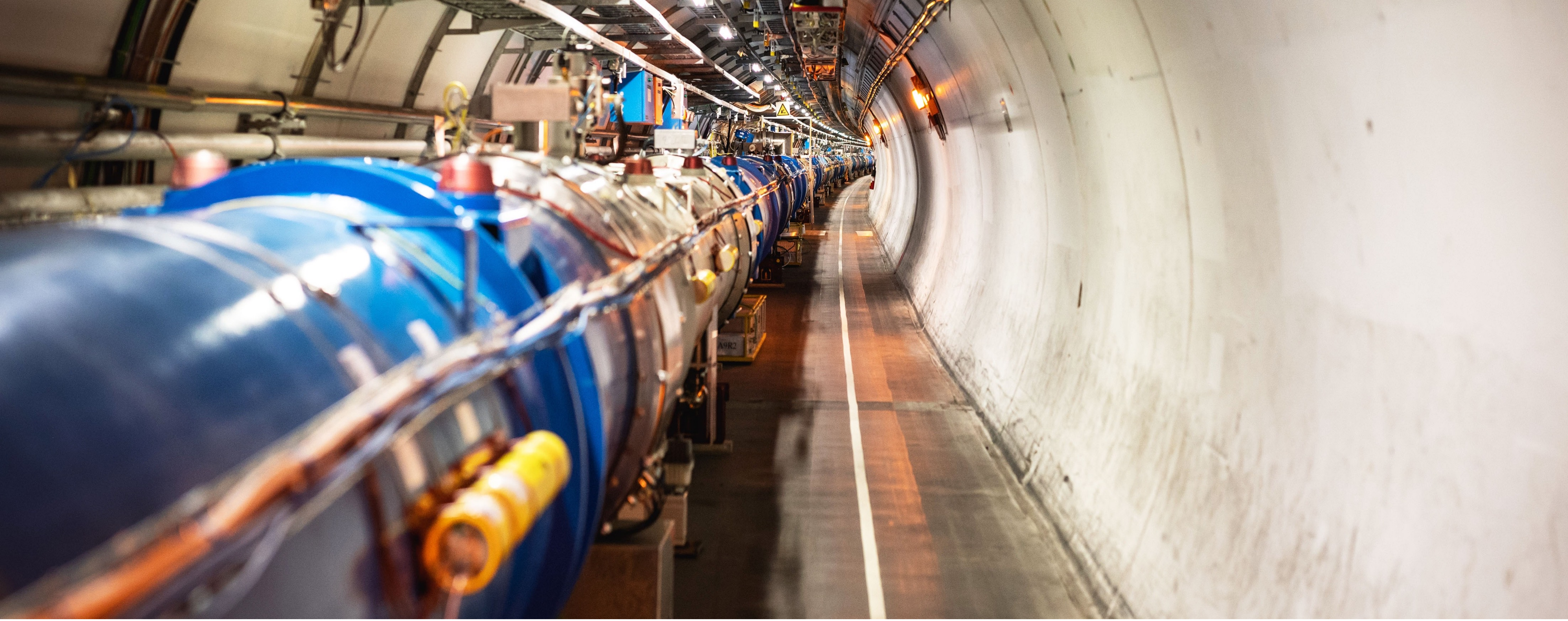
www.andrzejnowicki.pl



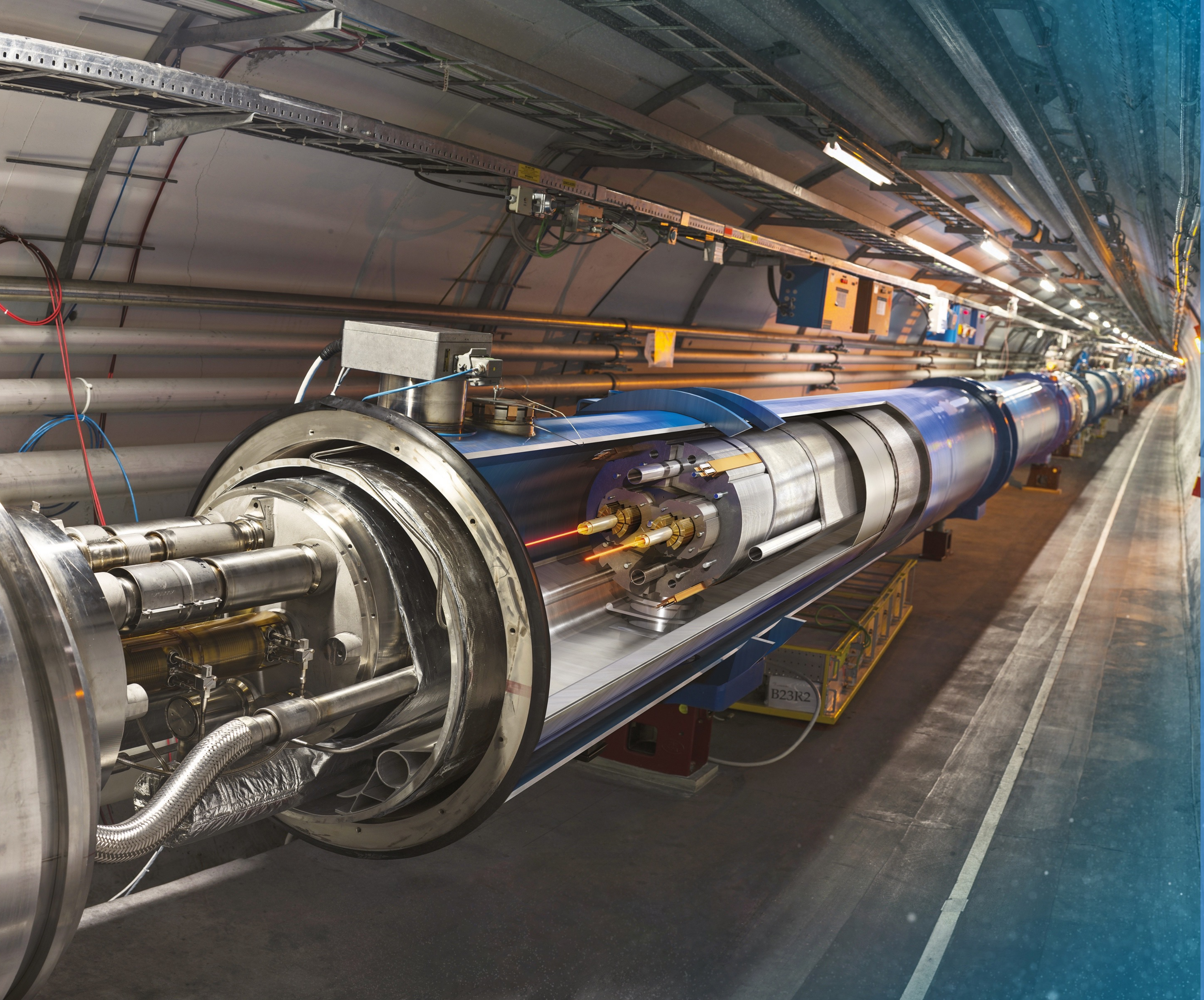
CERN is the world’s biggest laboratory for particle physics.

Our goal is to understand the most fundamental particles and laws of the universe.





Large Hadron Collider (LHC)

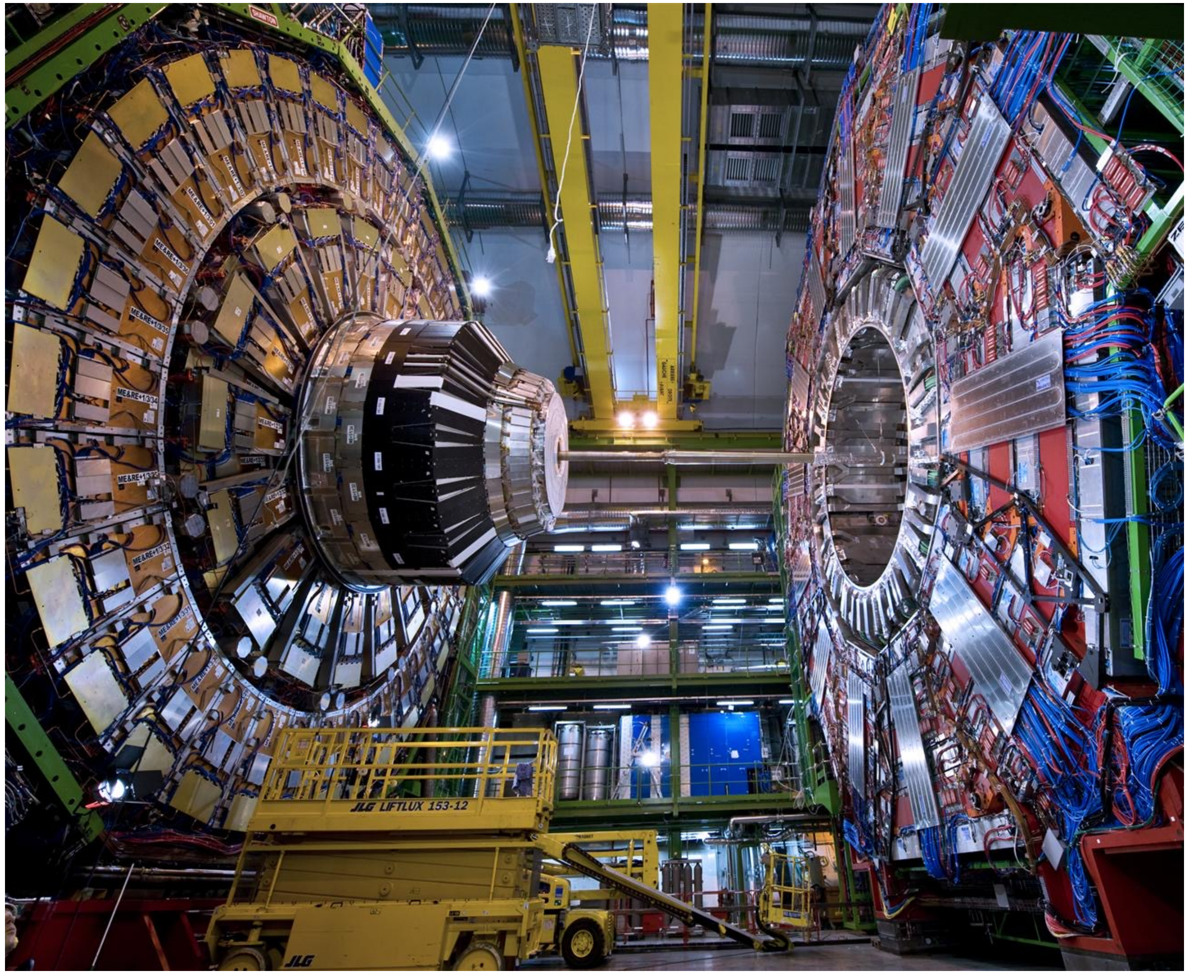


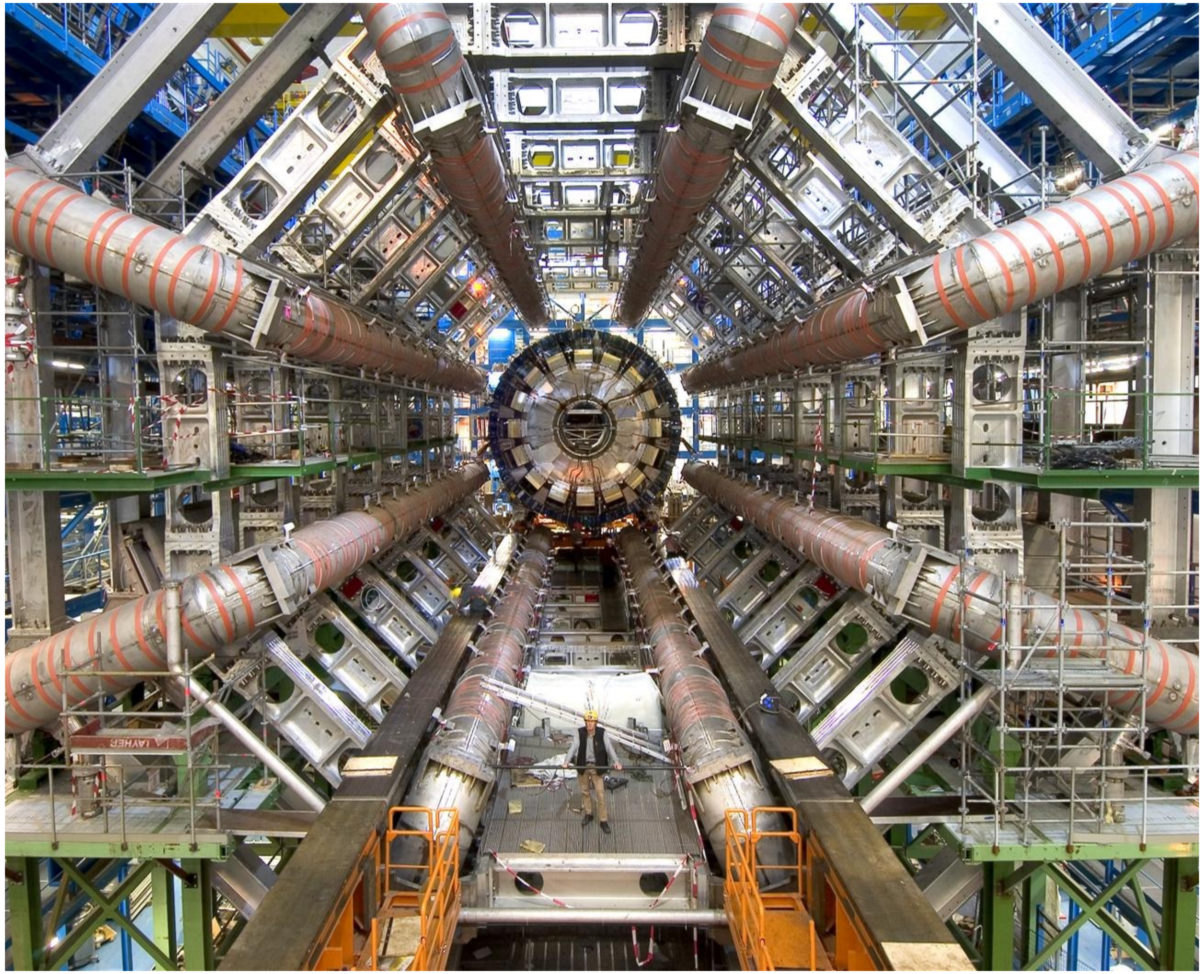
Large Hadron Collider (LHC)

- 27 km (17 mi) in circumference
- About 100 m (300 ft) underground
- Superconducting magnets steer the particles around the ring
- Particles are accelerated to close to the speed of light

CMS

ATLAS EXPERIMENT





IT @ CERN

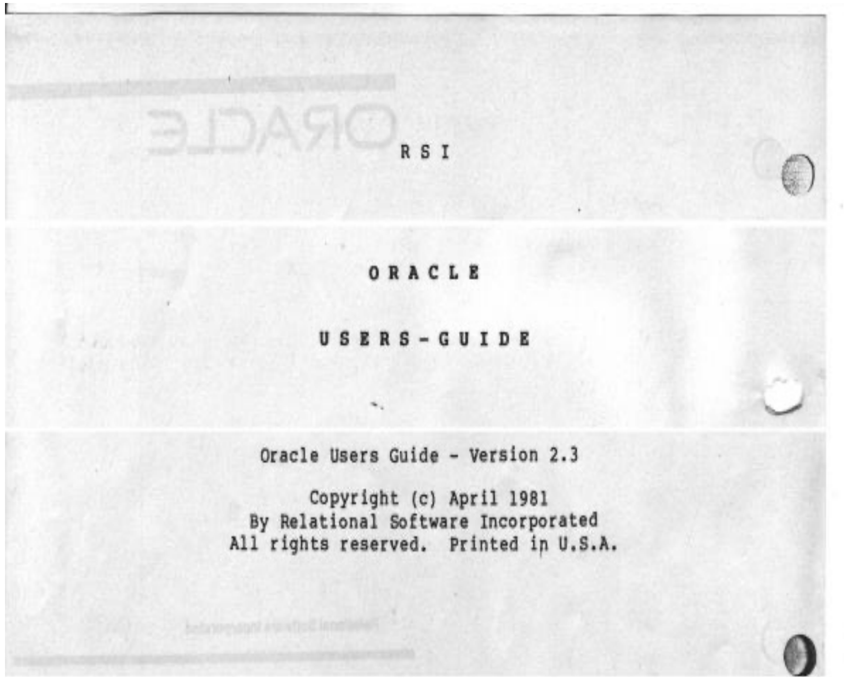
Databases at CERN

Oracle since 1982

- 105 Oracle databases, more than 11.800 Oracle accounts
- RAC, Active Data Guard, GoldenGate, OEM, RMAN, APEX, Cloud…
- Complex environment

Database on Demand (DBoD) since 2011

- ≈600 MySQL, ≈400 PostgreSQL, ≈200 InfluxDB
- Automated backup and recovery services, monitoring, clones, replicas
- HA MySQL clusters (Proxy + primary replica)



Size of the database environment

	Total size
Oracle	≈5 PB
DBoD (MySQL, PostgreSQL, InfluxDB)	≈150 TB
Backups	≈3 PB

VECTORS

~~Let's build a simple beer recommendation system~~

The content of this talk is intended for informational and entertainment purposes only.
Enjoy alcoholic beverages responsibly and always consume alcohol in moderation.
Please remember that alcohol consumption is not suitable for everyone, and there are many non-alcoholic options available for those who prefer them or are unable to consume alcohol.
I recommend exploring these alternatives as part of your beverage choices.
If you choose to consume alcohol, please ensure you are of legal drinking age in your location and never drink and drive or engage in activities that require full focus and coordination.
This talk is not intended to promote excessive drinking or irresponsible behaviour.
Always prioritize your health, well-being, and safety.

VECTORS

In AI, a vector is an ordered list of numbers (scalars) that can represent a point in a multidimensional space. Mathematically, a vector is often written as:

$$\mathbf{v} = (v_1, v_2, \ldots, v_{n-1}, v_n)$$

n is the dimensionality of the vector.

EMBEDDINGS

Embeddings are numerical representations of real-world objects that machine learning (ML) and artificial intelligence (AI) systems use to understand complex knowledge domains like humans do.

For example, a bird-nest and a lion-den are analogous pairs, while day-night are opposite terms. Embeddings convert real-world objects into complex mathematical representations that capture inherent properties and relationships between real-world data.

https://aws.amazon.com/what-is/embeddings-in-machine-learning/

EMBEDDING MODEL

An embedding model is a type of machine learning model designed to map high-dimensional or complex data (such as text, images, or categorical data) into lower-dimensional continuous vector spaces, known as embeddings. These embeddings capture the essential information or meaning of the data while preserving relationships between different data points in the original space.

How to put it all together?

Input

(movie, picture, text, etc.)

Embedding model

🚨

That's a simplification.
Normally you would cut the text in chunks and embed each chunk separately

Embedding

Vector

"Citrusy, sweet aroma"

`[0.329, 0.911, 0.21, 0.37, …]`

Vectors?

"Citrusy, sweet aroma"	*[0.329, 0.911, 0.21, 0.37, …]*
"Grapefruity taste, sweet aroma"	*[0.317, 0.818, 0.11, 0.36, …]*
"Harsh, spicy, roasted"	*[0.110, 0.010, 0.91, 0.87, …]*

Similar input should result in similar embedding (vector) values.

We can calculate distance between vectors to find similarity.

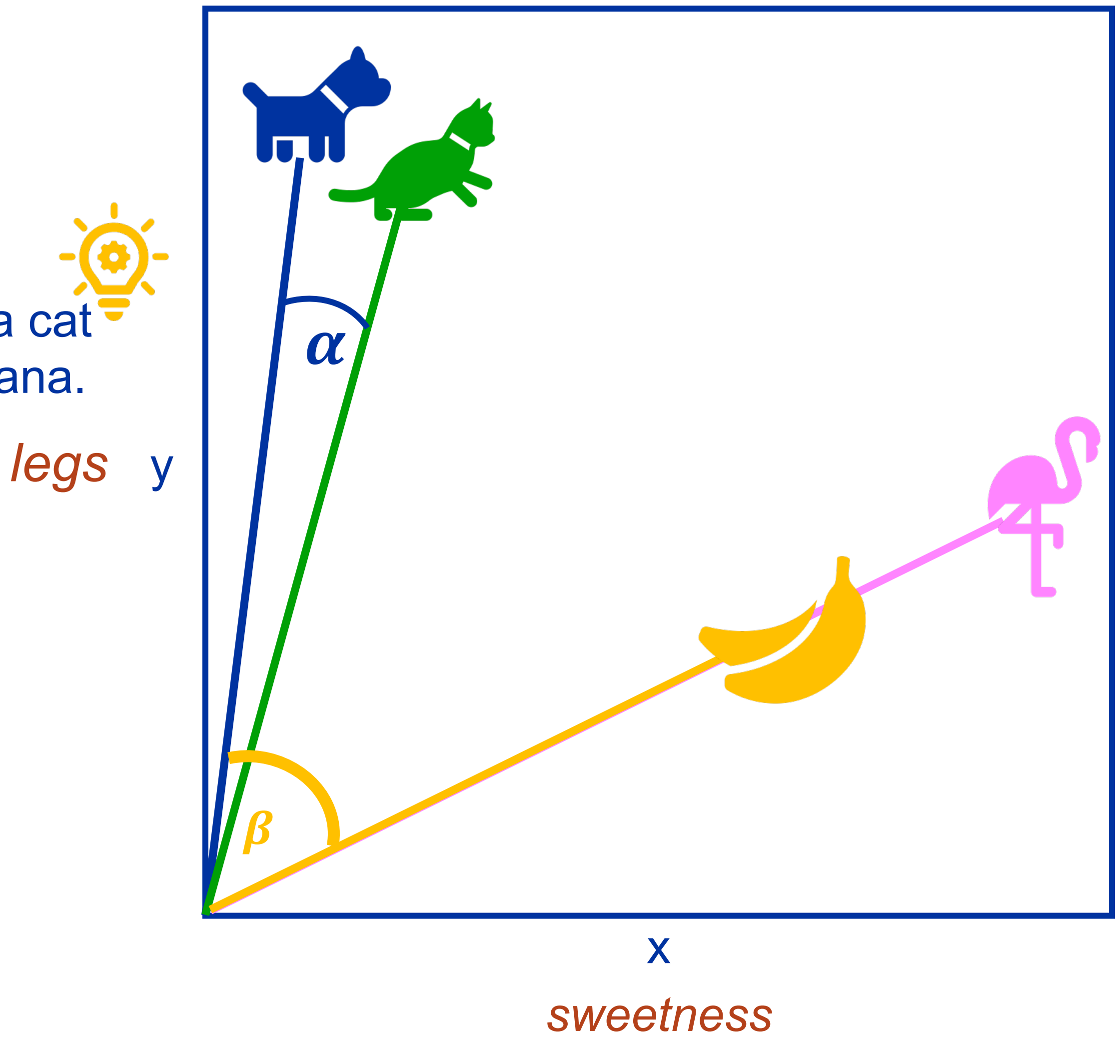
Our recommendation system will be based solely on similarity.

How to calculate similarity?

Cosine distance!

$\beta > \alpha$

A dog is more similar to a cat then it is similar to a banana.



Same thing hapens in the similarity search.

But we have 384 dimensions.

There are other methods.
More on that later.

There are some limitations of the similarity

higher number = more similar

“healthy” vs “unhealthy”	*0.6788*
“healthy” vs “not healthy”	*0.8208*
“dog” vs “banana”	*0.2532*
“I like beer” vs “Table partitioning is an amazing feature of RDBMS”	*0.0311*
“I like beer” vs “I like indexes in databases”	*0.2238*
“I like to index my data” vs “I like indexes”	*0.7497*

Healthy vs Unhealthy are similar because both are adjectives, related to the health status

The “opposite” is not well defined. What is the opposite of ”king”? Queen? Prince? Poor man? 🍌?

How do we handle the vectors in the db?

pgvector

pgvector

github.com/pgvector/pgvector

README License Security

pgvector

Open-source vector similarity search for Postgres

Store your vectors with the rest of your data. Supports:

- exact and approximate nearest neighbor search
- single-precision, half-precision, binary, and sparse vectors
- L2 distance, inner product, cosine distance, L1 distance, Hamming distance, and Jaccard distance
- any language with a Postgres client

Plus ACID compliance, point-in-time recovery, JOINs, and all of the other great features of Postgres

pgvector – HOWTO

1. **Build the extension (or download binaries)**
2. `> CREATE EXTENSION vector;`
3. `> ALTER TABLE beers ADD COLUMN embedding vector (...);`
4. **Add a library to your application code**
   Available for any language with a PG client (e.g. pgvector-python)

pgvector – queries

```
SELECT * FROM items ORDER BY embedding <=> '[3,1,2]' LIMIT 5;
```

But there’s more:

<-> L2 distance (Euclidean)

<#> (negative) inner product

<=> cosine distance

<+> L1 distance (added in 0.7.0, Manhattan)

<~> Hamming distance (binary vectors, added in 0.7.0)

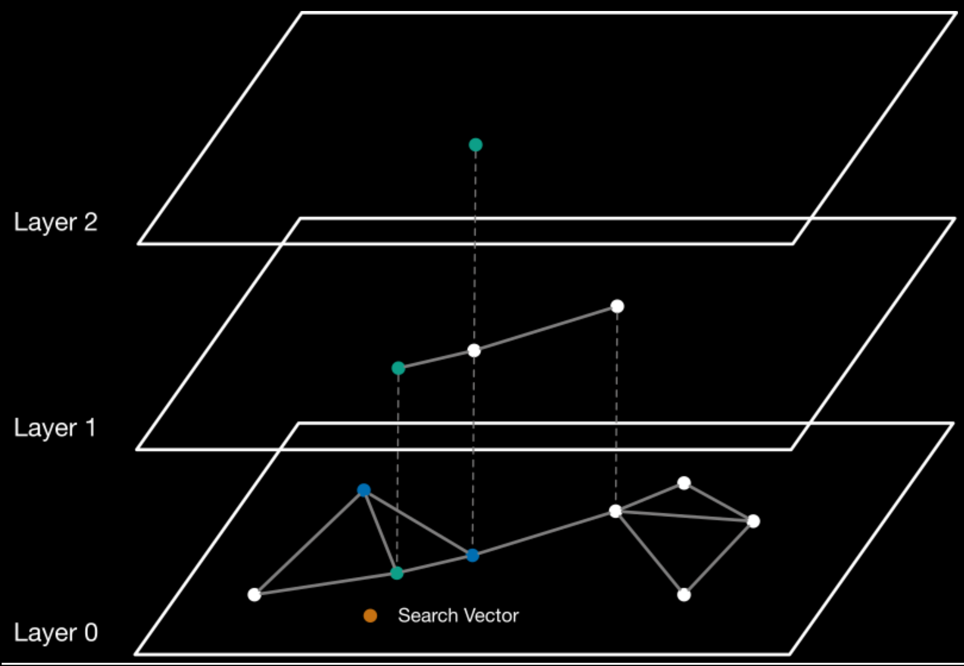
<%> Jaccard distance (binary vectors, added in 0.7.0)

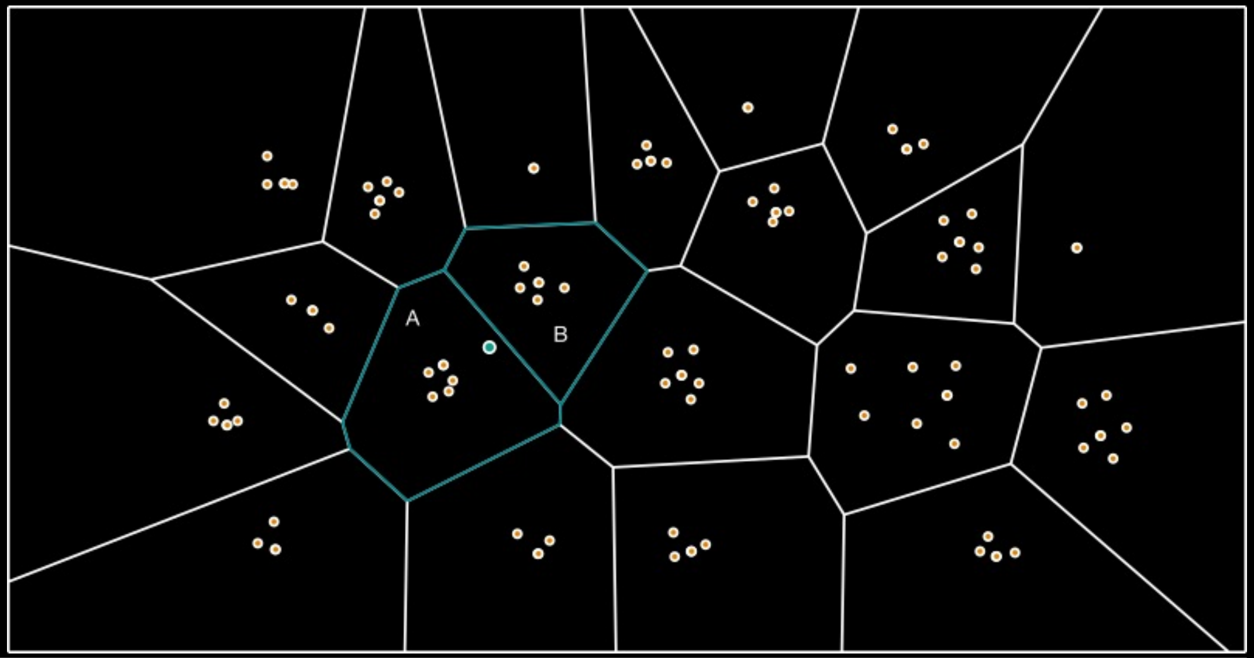
pgvector – indexes

By default, the nearest neighbour search will perform an exact search

There are two index types that you can use for approximate results:

- **Hierarchical Navigable Small World – HNSW**
- **InVerted File Flat - IVFFlat**





pgvector – indexes and filtering

```
SQL> SELECT *
     FROM beers
     WHERE category_id = 123
     ORDER BY embedding <-> '[3,1,2]'
     LIMIT 5;
```

With approximate indexes, the filtering is applied **after** the index is scanned.
It's possible that you'll get less than expected 5 rows.

For HNSW indexes, candidate list is 40 by default.
It's controllable, so you can adjust according to your filtering criteria.

You can also use Iterative Scan: `SET [hnsw/ivfflat].iterative_scan = relaxed_order;`
It will scan index more until enough results are found.

Enough theory

Let’s build a simple beer recommendation system

🍻 dataset

```
vector=# select id, beer_name, info from beers where id in (2707,2746,2612) ;

  id  |   beer_name    |                           info
------+----------------+------------------------------------------------------
 2612 | Massacre       | Imperial dark lager aged in bourbon barrels.
 2707 | Biere De Miele | Styled after a traditional Kolsch, this is an
                       | interpretation of a medieval Braggot,
                       | an ale fermented with honey
 2746 | Sun Drift      | Summon some sunshine with bright notes of citrus and
                       | black tea. A Brett-fermented ale with lemon zest and tea
```

This amazing dataset is available on Kaggle under creative commons license CC BY 4.0:

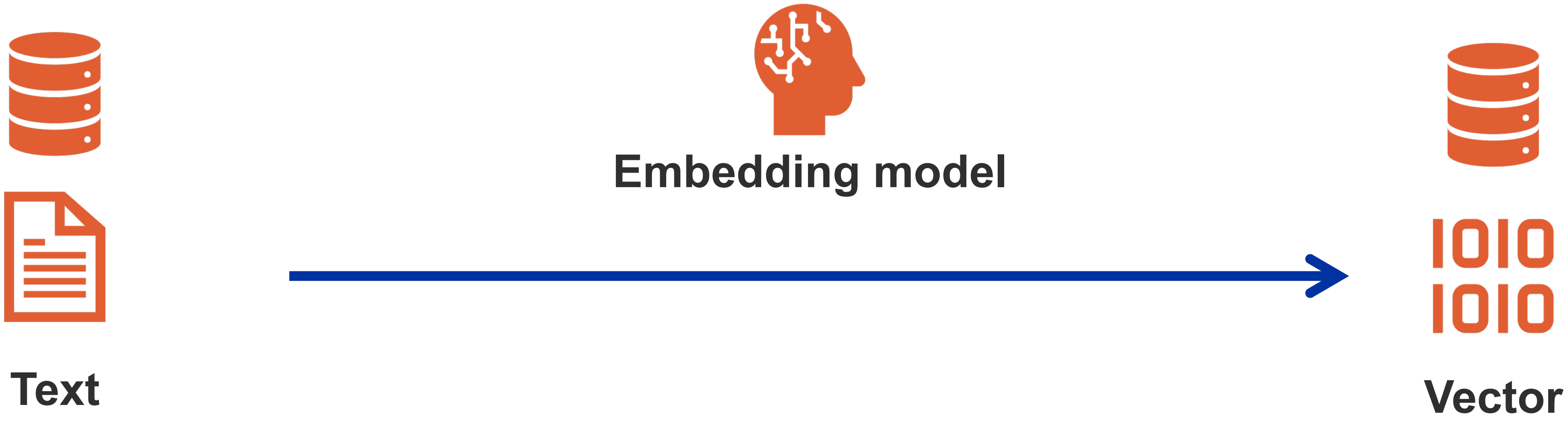
https://www.kaggle.com/datasets/ruthgn/beer-profile-and-ratings-data-set

VECTOR data type

```
vector=# \d beers
                                Table "public.beers”
  Column   |           Type           | Nullable |        Default
-----------+--------------------------+----------+----------------------
 id        | integer                  | not null | nextval('beers_id_seq'::regclass)
 beer_name | character varying(200)   |          |
 info      | character varying(4000)  |          |
 embedding | vector(384)              |          |

Indexes:
    "beers_pkey" PRIMARY KEY, btree (id)
```

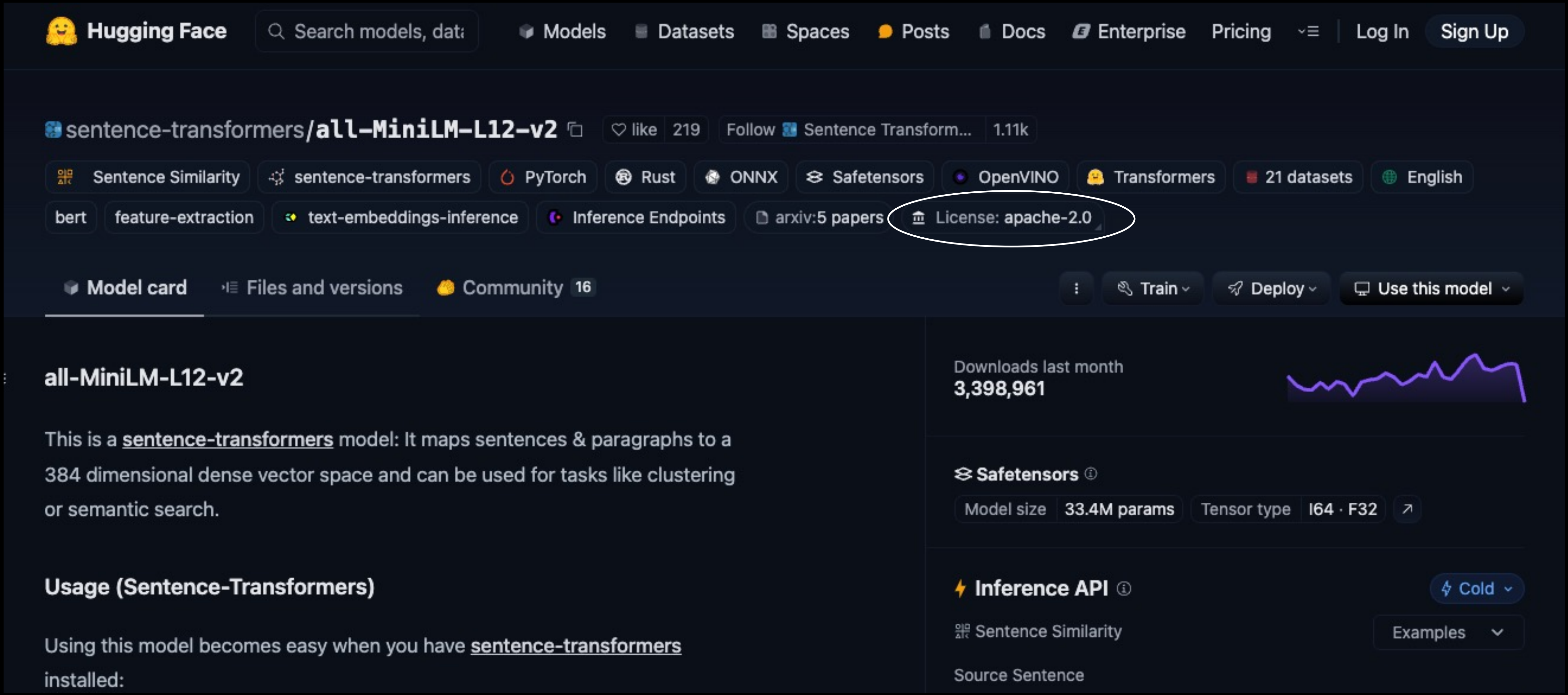
How to put it all together?



"Citrusy, sweet aroma"

[0.329, 0.911, 0.21, 0.37, …]

Embedding model



Embedding

```python
#!/bin/env python3

from sentence_transformers import SentenceTransformer

embedding_model = "sentence-transformers/all-MiniLM-L12-v2"

model = SentenceTransformer(embedding_model)

data = "rich blend of roasted barley"

embedding = list(model.encode(data))

print(embedding)
```

```
[-0.006417383, -0.022299055, -0.07196472, -0.038730085, 0.015408011,
0.011460664, 0.031957585, -0.14295837, -0.06265083, 0.047036696, 0.05393924,
-0.017266361, -0.060880985, -0.090641975, -0.018470088, 0.043274913,
0.10671821, -0.01918215, -0.017627805, 0.007417538, -0.094217524,
0.048147723, 0.007045083, -0.0059344354, 0.031551342, 0.0060908115, ...
```

Embedding Process

```
update beers set embedding = %s
            where id = %s;
```

Embedding 3361 beer descriptions

Embedding locally on Macbook M3 Pro (single threaded python)	**~43s**
Embedding locally on Macbook M3 Pro (Python's `multiprocessing.Pool` – 4 processes)	**~20s**

I used ChatGPT to parallelize my code

```python
    with connection.cursor() as cursor:
        # Loop over the rows and vectorize the data

        binds = []

        for id_val, info in cursor.execute(query_sql):
            # Create the embedding and extract the vector
            embedding = list(model.encode(info))

            # Record the array and key
            binds.append([embedding, id_val])

            print(info)


        # Do an update to add or replace the vector values
        cursor.executemany(
            update_sql,
            binds,
        )
```

```python
"""select id, info
   from beers
   order by 1"""
```

```python
"""update beers
   set embedding = %s
   where id = %s"""
```

VECTOR SEARCH

```
select beer_name, info
from beers
order by embedding <=> %s
limit %s;
```

```
import psycopg

from pgvector.psycopg import register_vector

from sentence_transformers import SentenceTransformer

    register_vector(connection)


            # Create the embedding and extract the vector
            embedding = model.encode(user_input)

                                """select beer_name, info
                                   from beers
                                   order by embedding <=> %s
                                   limit %s"""
            beers = []
            for (beer_name,info,) in cursor.execute(sql, [embedding, top]):
                beers.append((beer_name,info))

            for hit in beers:
              print(hit)
```

NO INDEXES

```
vector=# explain analyze select beer_name, info
         from beers
         where id <> 2363
         order by embedding <=> (select embedding from beers where id = 2363)
         limit 5;
                                                  QUERY PLAN
----------------------------------------------------------------------------------------------------------
 Limit  (cost=2064.52..2064.53 rows=5 width=357) (actual time=15.095..15.098 rows=5 loops=1)
   InitPlan 1
     ->  Index Scan using beers_pkey on beers beers_1
                          (cost=0.28..8.30 rows=1 width=1146) (actual time=0.013..0.014 rows=1 loops=1)
           Index Cond: (id = 2363)
   ->  Sort  (cost=2056.22..2064.62 rows=3360 width=357) (actual time=15.093..15.094 rows=5 loops=1)
         Sort Key: ((beers.embedding <=> (InitPlan 1).col1))
         Sort Method: top-N heapsort  Memory: 35kB
         ->  Seq Scan on beers  (cost=0.00..2000.41 rows=3360 width=357) (actual time=0.101..13.523
rows=3360 loops=1)
               Filter: (id <> 2363)
               Rows Removed by Filter: 1
 Planning Time: 0.710 ms
 Execution Time: 15.143 ms
```

```
vector=# create index on beers using hnsw (embedding vector_cosine_ops);
CREATE INDEX
vector=# explain analyze select beer_name, info
         from beers
         where id <> 2363
         order by embedding <=> (select embedding from beers where id = 2363)
         limit 5;
                                              QUERY PLAN
--------------------------------------------------------------------------------------------------------------
 Limit  (cost=482.29..497.44 rows=5 width=357) (actual time=3.172..3.261 rows=5 loops=1)
   InitPlan 1
     ->  Index Scan using beers_pkey on beers beers_1
                           (cost=0.28..8.30 rows=1 width=1146) (actual time=0.018..0.019 rows=1 loops=1)
           Index Cond: (id = 2363)
   ->  Index Scan using beers_embedding_idx on beers
                  (cost=473.99..10651.62 rows=3360 width=357) (actual time=3.169..3.255 rows=5 loops=1)
         Order By: (embedding <=> (InitPlan 1).col1)
         Filter: (id <> 2363)
         Rows Removed by Filter: 1
 Planning Time: 0.776 ms
 Execution Time: 3.317 ms
```

```
vector=# create index on beers using ivfflat (embedding vector_cosine_ops);
CREATE INDEX
vector=# explain analyze select beer_name, info
         from beers
         where id <> 2363
         order by embedding <=> (select embedding from beers where id = 2363)
         limit 5;
                                              QUERY PLAN
------------------------------------------------------------------------------------------------------------
 Limit  (cost=27.00..40.58 rows=5 width=357) (actual time=0.444..0.487 rows=5 loops=1)
   InitPlan 1
     ->  Index Scan using beers_pkey on beers beers_1
                           (cost=0.28..8.30 rows=1 width=1146) (actual time=0.017..0.019 rows=1 loops=1)
           Index Cond: (id = 2363)
   ->  Index Scan using beers_embedding_idx1 on beers
                  (cost=18.70..9144.62 rows=3360 width=357) (actual time=0.441..0.483 rows=5 loops=1)
         Order By: (embedding <=> (InitPlan 1).col1)
         Filter: (id <> 2363)
         Rows Removed by Filter: 1
 Planning Time: 0.724 ms
 Execution Time: 0.527 ms
```

VECTOR SEARCH

Prompt: 'lemon'

Sun Drift
Summon some sunshine with bright notes of citrus and black tea. A Brett-fermented ale with lemon zest and tea

Lemon Lager
Refreshingly cool taste produced with freshly squeezed lemon juice from Japanese Hiroshima Lemons, fermented and bottled as the perfect thirst-quencher, no matter what season.

Tocobaga Red Ale
Pours amber in color with notes of citrus and caramel. Citrus hop bitterness upfront with notes of caramel and an Amish bread sweetness. Citrus hop bitterness returns at the end for a long dry finish.75 IBU

Sorachi Ace
This is a saison featuring the rare Japanese-developed hop Sorachi Ace. The Sorachi Ace hop varietal is noted for its unique lemon zest/lemongrass aroma.

Femme Fatale Sudachi
A new version of Evil Twin?s classic brett fermented I.P.A. feauring Sudachi, an Asian citrus, for a nice citrusy note.

DEMO

DRINK RESPONSIBLY!

https://pixabay.com/photos/beer-draft-beer-happy-hour-beverage-2218900/

What about real life usage?

How to put it all together?

Input (e.g. text) → **Chunking** → **Chunks** → **Embedding model** → **Vector**

Two households, both alike in dignity
(In fair Verona, where we lay our scene),
From ancient grudge break to new mutiny,
Where civil blood makes civil hands unclean
…

Chunks	Vector
Two households, both alike in dignity	[0.329, 0.917, 0.211, 0.307, …]
(In fair Verona, where we lay our scene),	[0.129, 0.101, 0.561, 0.487, …]
From ancient grudge break to new mutiny,	[0.989, 0.091, 0.231, 0.962, …]
Where civil blood makes civil hands unclean	[0.439, 0.053, 0.513, 0.321, …]

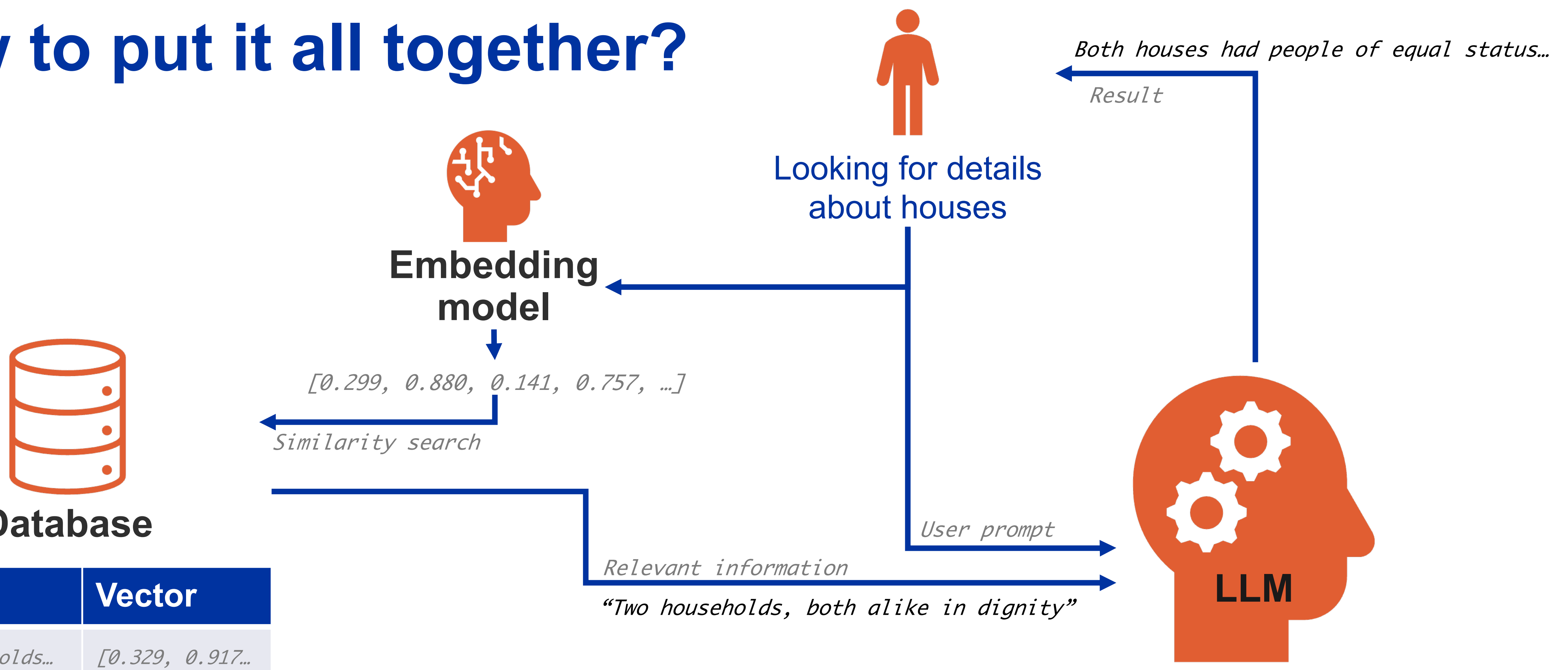
How to put it all together?

Chunks & **Vector** → **Database**

Chunks	Vector
Two households, both alike in dignity	*[0.329, 0.917, 0.211, 0.307, …]*
(In fair Verona, where we lay our scene),	*[0.129, 0.101, 0.561, 0.487, …]*
From ancient grudge break to new mutiny,	*[0.989, 0.091, 0.231, 0.962, …]*
Where civil blood makes civil hands unclean	*[0.439, 0.053, 0.513, 0.321, …]*

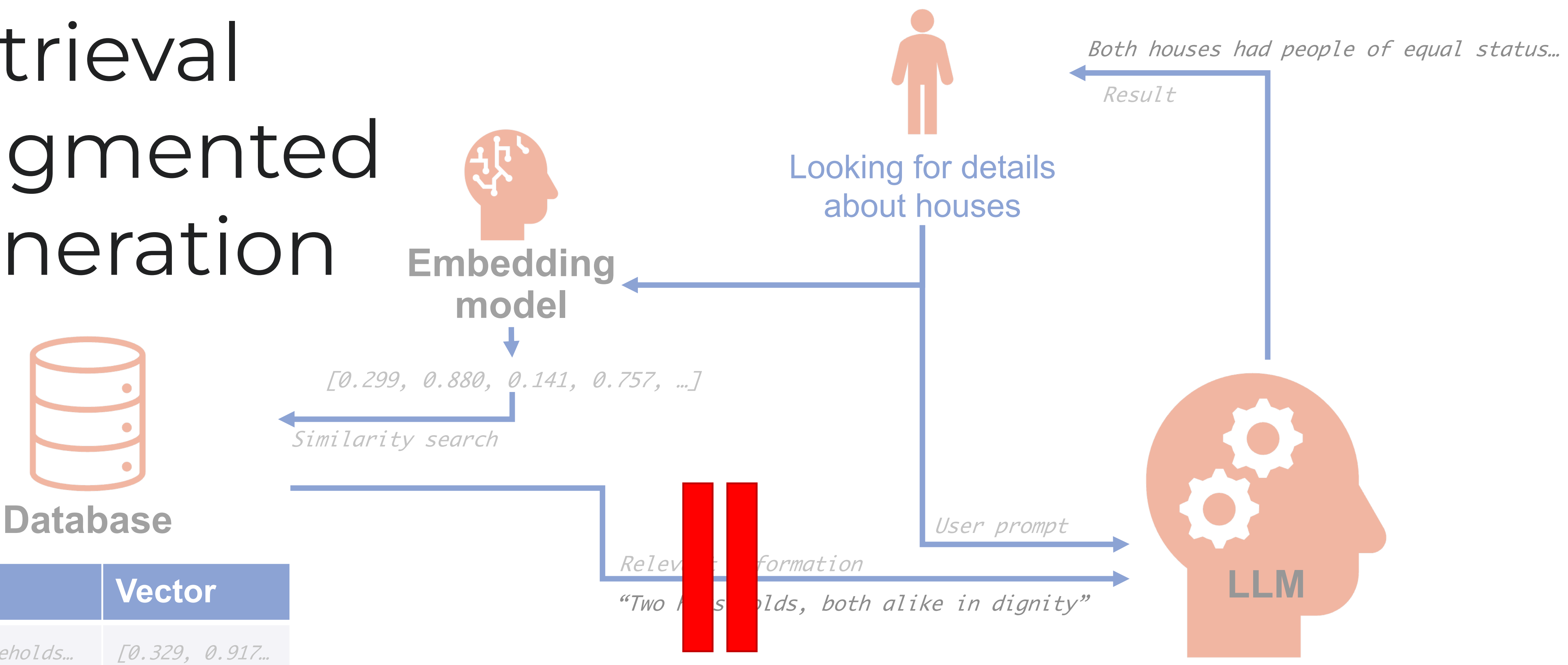
Text	Vector
Two households…	*[0.329, 0.917…*
(In fair Verona…	*[0.129, 0.101…*
From ancient…	*[0.989, 0.091…*
Where civil…	*[0.439, 0.053…*

How to put it all together?



Text	Vector
Two households…	*[0.329, 0.917…*
(In fair Verona…	*[0.129, 0.101…*
From ancient…	*[0.989, 0.091…*
Where civil…	*[0.439, 0.053…*

Retrieval **A**ugmented **G**eneration



Text	Vector
Two households…	*[0.329, 0.917…*
(In fair Verona…	*[0.129, 0.101…*
From ancient…	*[0.989, 0.091…*
Where civil…	*[0.439, 0.053…*

We would add a ReRank operation here
We can query from DB more information
Rank our information on relevance
Be selective in what we feed into the LLM

References

Writeup of CERN's
Internal Knowledge Chatbot

pgvector	https://github.com/pgvector/pgvector
IVFFlat & HNSW	https://skyzh.github.io/write-you-a-vector-db/
psycopg3	https://www.psycopg.org/psycopg3/
AccGPT	https://indico.cern.ch/event/1395528/contributions/5865654/attachments/2833642/4952053/AccGPT-IML_v2.pdf
Beer dataset	https://www.kaggle.com/datasets/ruthgn/beer-profile-and-ratings-data-set

***Romeo and Juliet* by W. Shakespeare**

Visit us if you're around Geneva!

https://visit.cern

We also have a booth!

Bulding F, Level 2

Thank you !

andrzejnowicki

andrzej.nowicki@cern.ch

www.andrzejnowicki.pl