Leveraging Vector Search for RAG in Generative AI

Kai Yu

Future & Past TechCasts:

Dec 12th

Unlocking Insights: Mastering Data Storytelling with Oracle Analytics

Presented by **Philip Godfrey**

Jan 9th

The Oracle AI Microservices Sandbox for RAG Rapid Prototyping

Presented by **Corrado De Bari, Mark Nelson, & John Lathouwers**

Jan 23rd

Leveraging Vector Search for RAG in Generative AI

Presented by **Kai Yu**

TechCast Archive

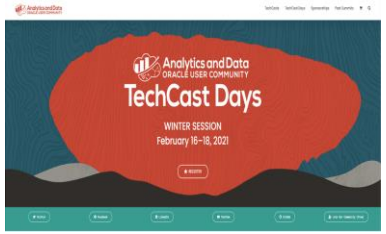
2024 | 2023 | 2022 | 2021 | 2020 | 2019

Date	Title	Presenter(s)	Replay	Download(s)
Nov 7	Gimme a Vector, Victor: Leveraging Vector Datatypes for Practical Generative AI Applications	Jim Czuprynski	Video	Slides
Oct 17	Our Favorite New Features in OAC	Dan Vlamis, Wayne Van Sluys, Cathye Pendley, Tim Vlamis, Mystery Guest: Gautam Pisharam	Video	Slides
Oct 3	Harnessing Oracle GoldenGate 23ai with AI Automation: A New Era of Intelligent Monitoring, Building, and Diagnostics	Bobby Curtis	Video	Slides
Sept 19	Discovering Oracle Fusion Data Intelligence	Peter Koutroubis & Jai Gangwani	Video	Slides
Sept 5	Oracle CloudWorld 2024 Session Analysis and Expert Agenda Recommendations	Roger Cressey, Dan Vlamis, Jim Czuprynski, Tim Vlamis, Cathye Pendley	Video	Slides
Aug 22	Our Favorite Features of OAC	Dan Vlamis, Tim Vlamis, Cathye Pendley, & Oracle Analytics Mystery Guest	Video	Slides

Submit a topic to share at **https://andouc.org/techcasts/**

Analytics and Data
ORACLE USER COMMUNITY

Let's Connect



Website
http://andouc.org/

Chat with the Experts
https://bit.ly/Join-ANDOUC-Slack

Watch Previous TechCasts
https://bit.ly/3qmGgHN

@AnalyticAndData

https://www.facebook.com/AnDOracleUserCommunity

https://www.linkedin.com/company/analytics-and-data-oracle-user-community

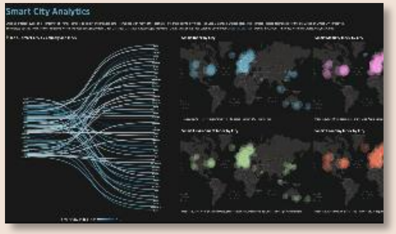
Spatial + Graph SIG
bit.ly/Spatial-Graph-LinkedIn

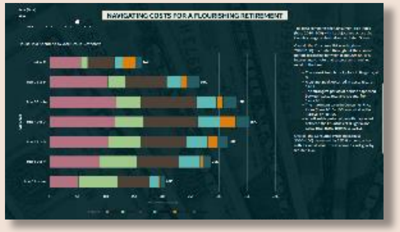
The Oracle Analytics

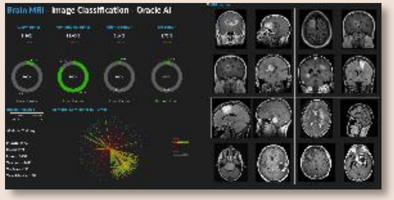
Data Visualization Challenge 2025

Get recognized for your Oracle Analytics skills!

Running February 3rd – 28th 2025







About Me: Kai Yu

- Independent Consultant, Ex Dell Technologies Distinguished Engineer
- 30 years experience in Tech Industry
- Specializing in Oracle Database/Apps, Cloud and AI/Machine Learning
- Author and Frequent Speaker at IEEE and Oracle Conferences
- IOUG Cloud Computing SIG Co-founder and VP
- Oracle ACE Director since 2010
- Oracle Cloud Infrastructure Generative AI Certified Professional
- OAUG Innovator of Year Award
- Oracle Technologist of the Year: Cloud Architect by Oracle Magazine
- My Blog: http://kyutechblog.wordpress.com/

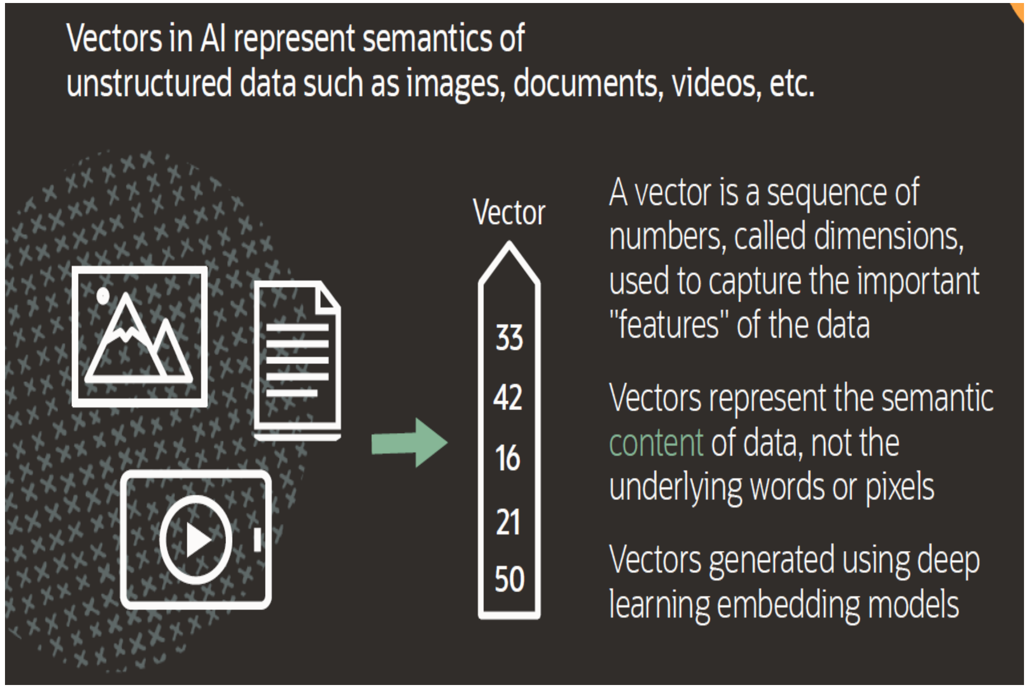
Agenda

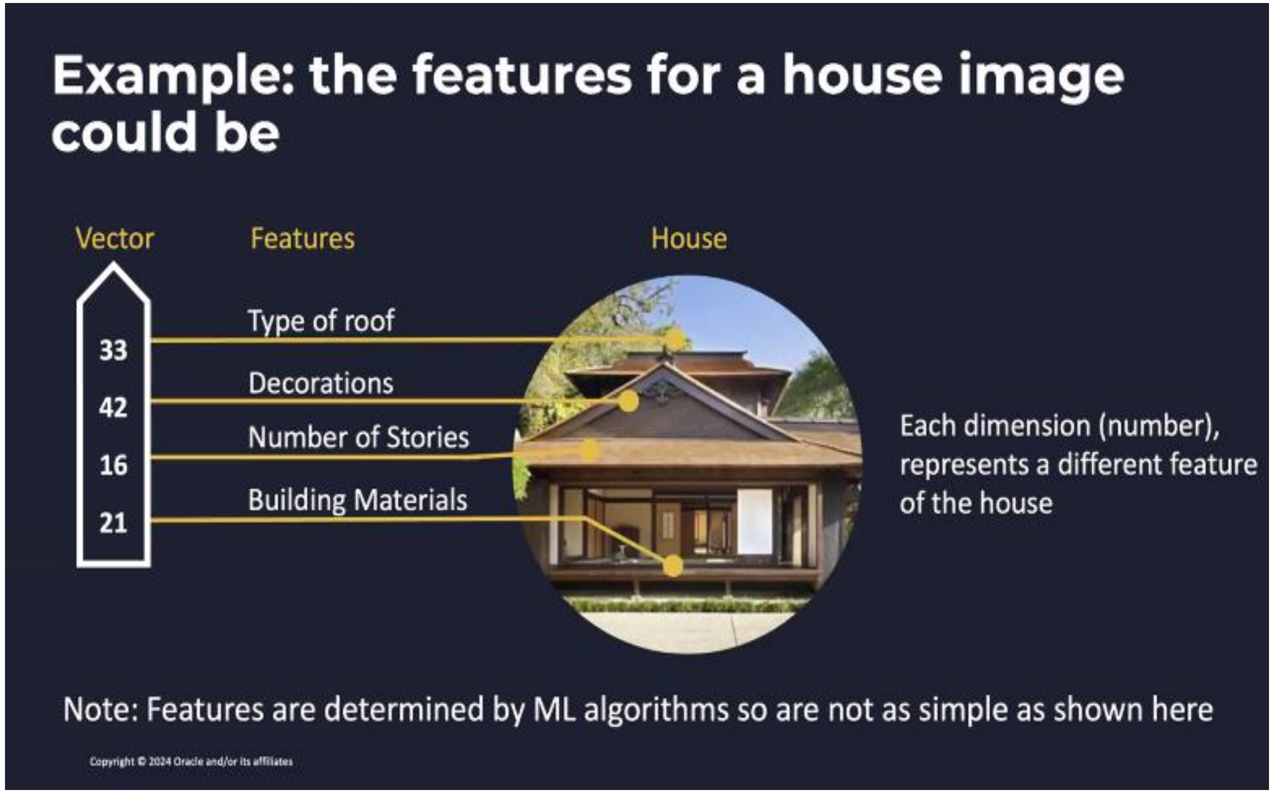
- Why vector and vector search?
- Vector search in Oracle database 23ai
- Generating vector embedding
- AI Vector Search for RAG in Generative AI

Vector and Vector Search

What are Vectors?

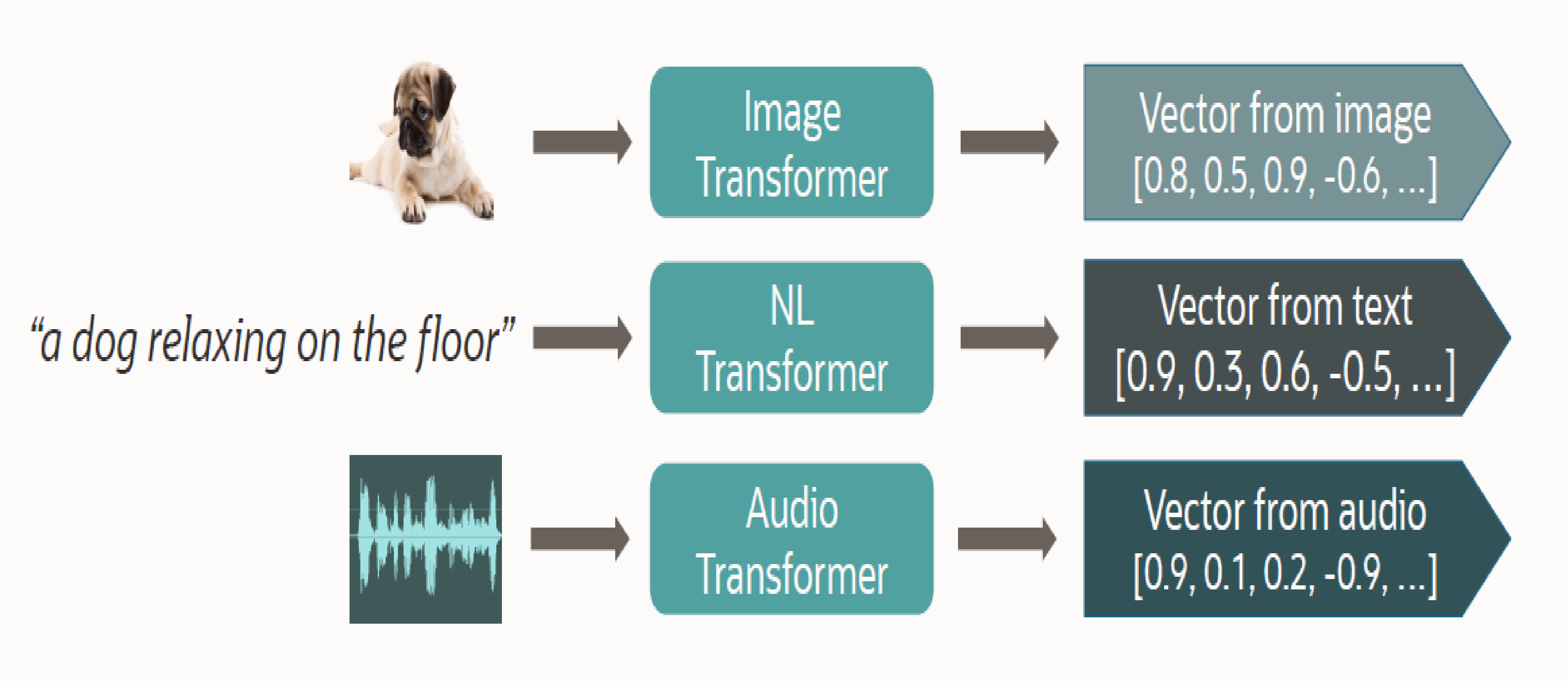
- Machine learning models are based on statistical calculation and they work with numbers.
- Before passing texts into the model process, you must first tokenize the words and converts the words into numbers.
- Vectors are used in AI to capture the semantics of data: Images, documents, videos, or even structured data





Vector Embeddings

Maps input to a multi-dimensional “concept space” as a vector of numbers



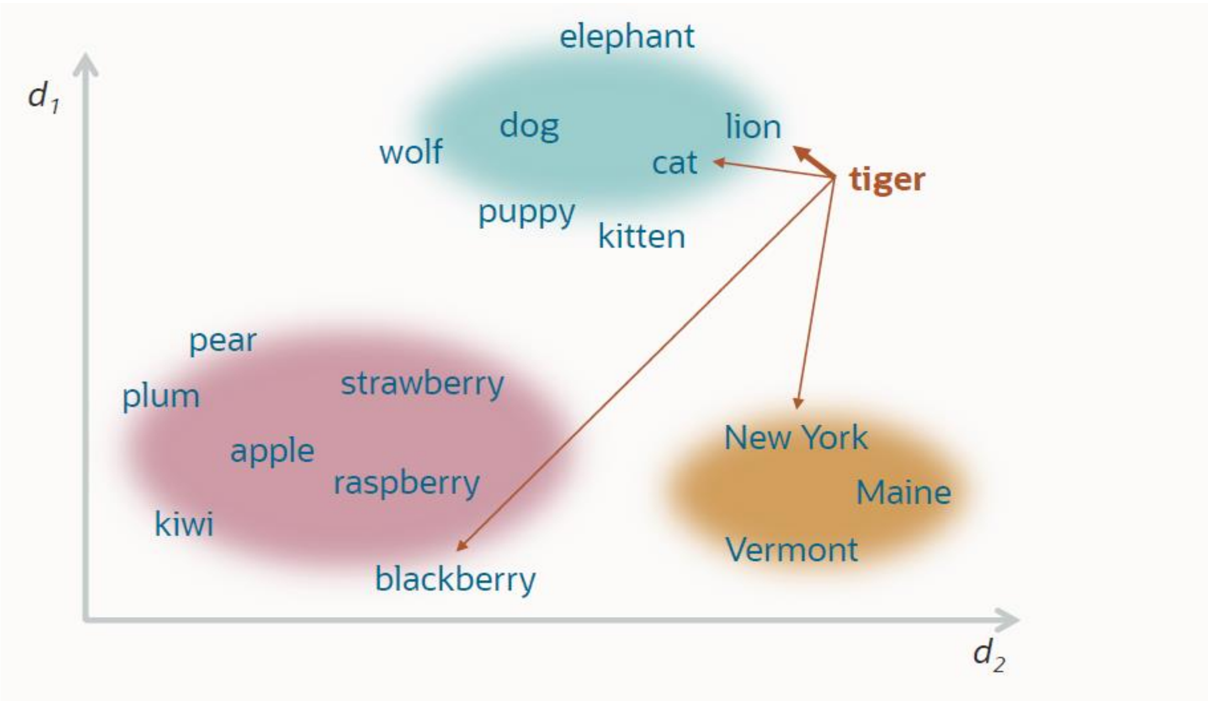
Vector Search based on the Similarity of Entities:

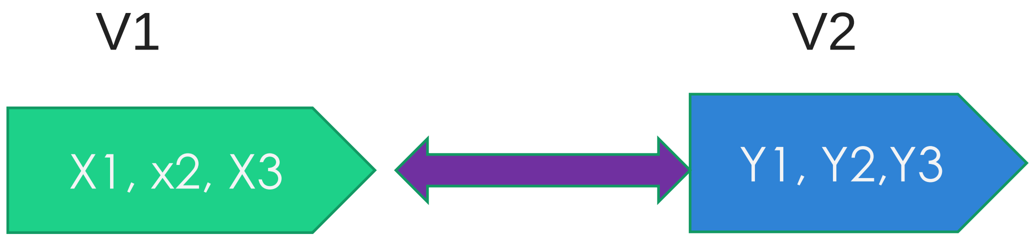
A new technology that enhances information retrieval by mapping queries to relevant data in your database based on semantics, instead of precise matches, using vectors to measure similarity

Word relatedness in two dimensions : Compare vectors to determine object relatedness or similarity

Similarity is based on the distance of two vectors : the mathematical distance between them

The more similar entities are, the shorter the distance between their vectors





Distance of two vectors V1 and V2 =
SQRT $((Y1 - X1)^2 + (Y2 - X2)^2 + (Y3 - X3)^2)$

There are many mathematical distance formulas (e.g., Euclidean, Cosine, Hamming)

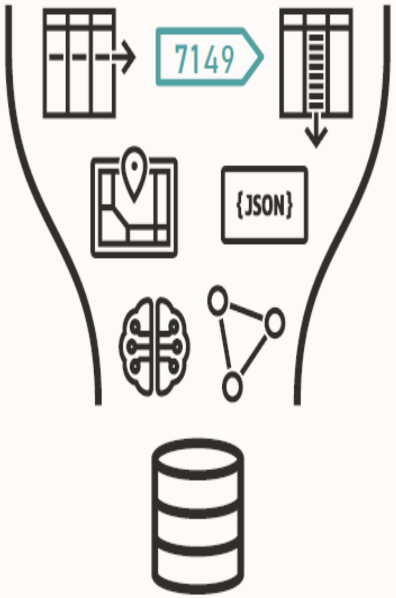
Oracle AI Vector Search with Oracle Database 23ai

AI Vector Search Feature in Oracle Database 23ai

AI Vector Search

An end-to-end solution for similarity search operations

- New VECTOR data type for storing vector embeddings
- New SQL syntax and functions expresses similarity search with ease
- New Approximate search indexes packaged and tuned for high performance and quality
- Vector similarity search in queries alongside business data about customers and products
- Handle vector and other workloads in same database



Create a table with vector data type:

```
create table my_vectortable
      (id number,
      datavec   VECTOR(3, FLOAT32)
       )
```

Vector DML operation:

Insert to the vector table:

```
   insert into my_vectortable (1,
      TO_VECTOR('[1.1, 2.2, 3.3]');
```

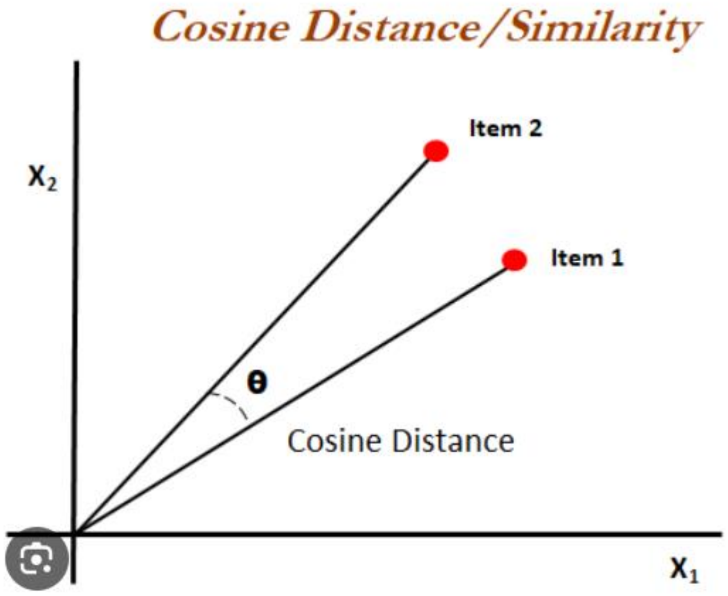
Vector Search SQL | Distance Function

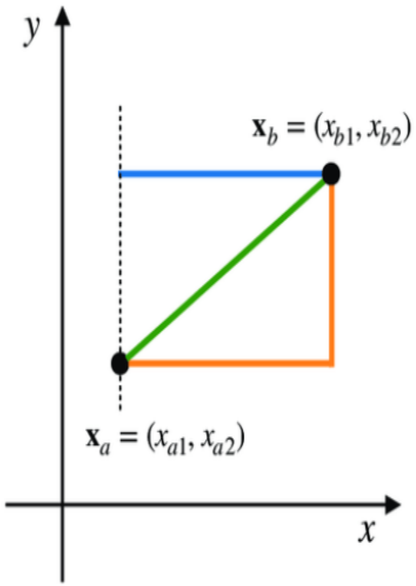
. NEW SQL Function to compute distance between vectors to gauge similarity

VECTOR_DISTANCE(VECTOR1, VECTOR2, <optional distance metric>)

- Different embedding models can use different distance metrics, but the basic concept remains the same
- The Distance between two vectors is smaller for entities that are more similar
- Distance functions supported in 23ai (specified in *metric*) are:

  COSINE (Default), EUCLIDEAN, EUCLIDEAN_SQUARED, HAMMING, MANHATTAN, DOT





$p = 2$ Euclidean distance

$$\|\mathbf{x}_a - \mathbf{x}_b\|_2 = (|x_{a1} - x_{b1}|^2 + |x_{a2} - x_{b2}|^2)^{\frac{1}{2}}$$

$p = 1$ Manhattan distance

$$\|\mathbf{x}_a - \mathbf{x}_b\|_M = |x_{a1} - x_{b1}| + |x_{a2} - x_{b2}|$$

$p = \infty$ Chebyshev distance

$$\|\mathbf{x}_a - \mathbf{x}_b\|_\infty = \max\{|x_{a1} - x_{b1}|, |x_{a2} - x_{b2}|\}$$

The dot product formula is:

$$A \cdot B = \sum_{i=1}^{n} A_i B_i$$

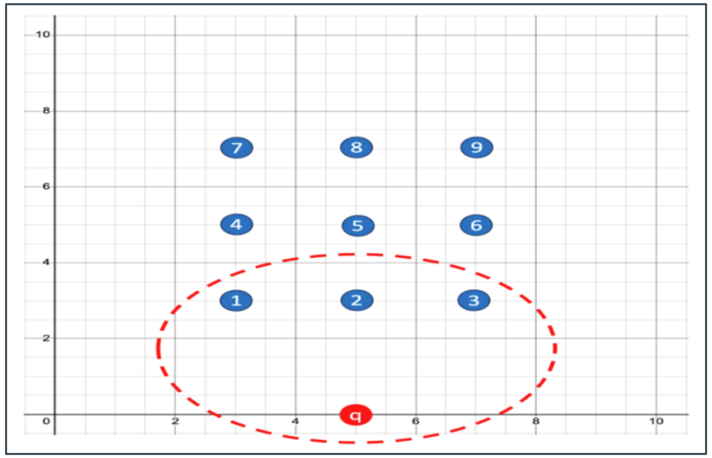
Oracle AI Vector Search with Oracle Database 23ai

- Similarity search using vector_distance function:

SELECT id, to_number(**vector_distance**(vector('[1.1, 2.2, 3.3]'), v)) distance FROM vt2

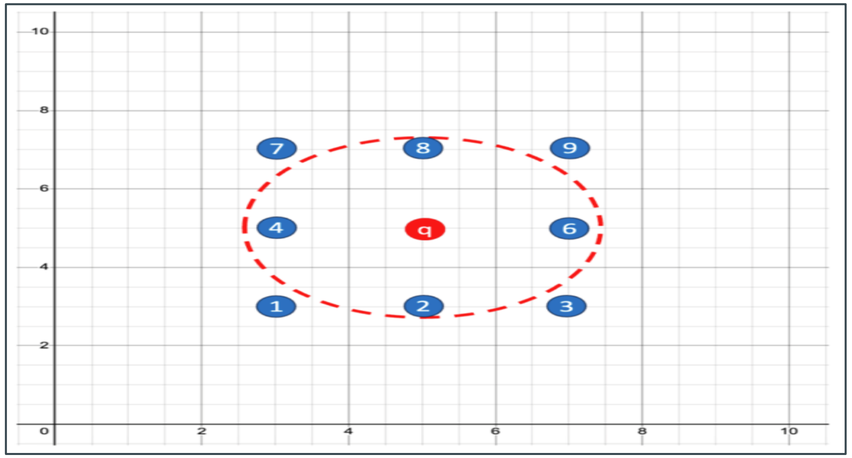
Look for closest vectors to a given vector(5,0)

```
SQL> SELECT id
  2  FROM   vt1
  3  ORDER  BY  vector_distance(
  4                 vector('[5, 0]'), v)
  5  FETCH FIRST 3 ROWS ONLY;

        ID
----------
         2
         1
         3

3 rows selected.
```



Look for closet Vectors to a given Vector(5,5)

```
SQL> SELECT id
  2  FROM   vt1
  3  ORDER  BY  vector_distance(vector('[5, 5]'), v)
  4  FETCH FIRST 5 ROWS ONLY;

        ID
----------
         5
         2
         8
         6
         4

5 rows selected.
```



Vector Indexes for Faster Similarity Searches

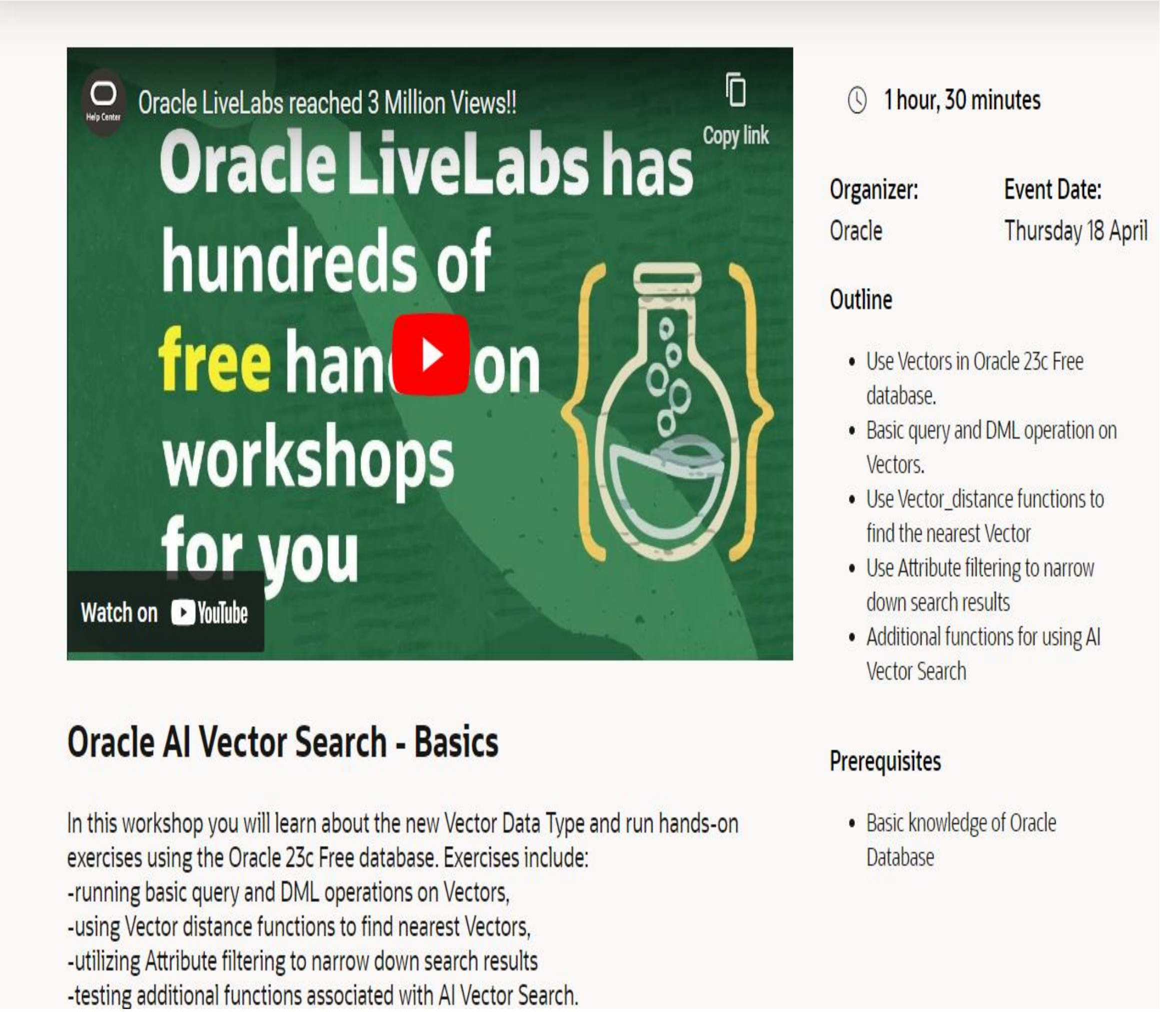
- Designed to accelerate similarity searches using high-dimensional vectors.
- Use techniques such as clustering, partitioning, and neighbor graphs to group vectors representing similar items
- Index Creation syntax:

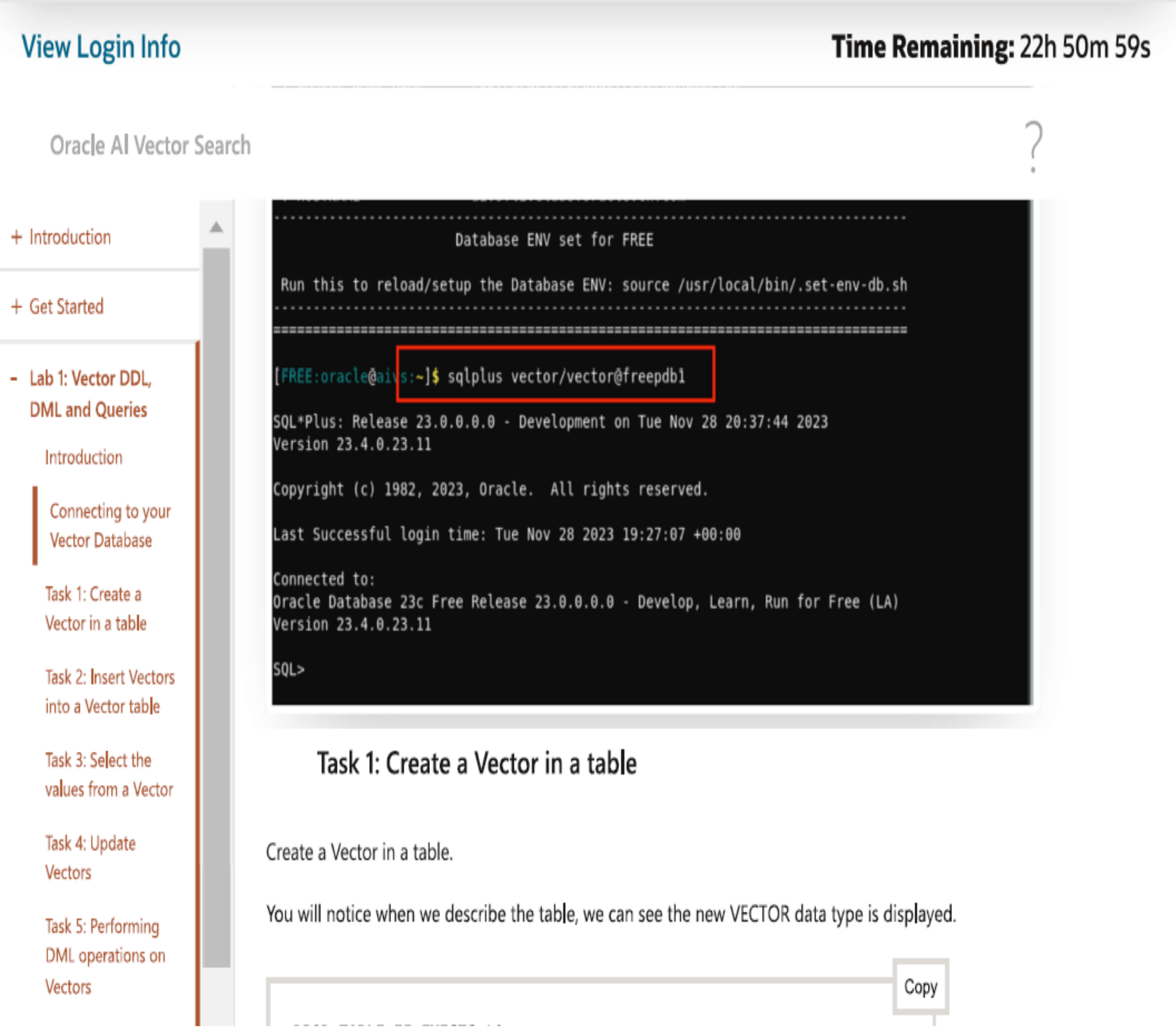
```
CREATE VECTOR INDEX  index_name ON table(vector_column)
ORGANIZATION [INMEMORY NEIGHBOR GRAPH | NEIGHBOR PARTITIONS]
DISTANCE COSINE | EUCLIDEAN | MANHATTAN | ...
TARGET_ACCURACY [<percent> | <Low-level parameters: efConstruction, nClusters
```

- **Neighbor Graph Vector Index**: Graph-based index where vertices represent vectors and edges between vertices represent similarity, In-Memory only index – highly efficient for both accuracy and speed
- **Hierarchical Navigable Small Worlds (HNSW):** a multi-layer in-memory graph index
- **Neighbor Partition Vector Index:** Partition-based index with vectors clustered into table partitions based on similarity

Oracle AI Vector Search Livelab

. LiveLab Link: https://apexapps.oracle.com/pls/apex/r/dbpm/livelabs/view-workshop?wid=1070&clear=RR,180&session=102975809866838

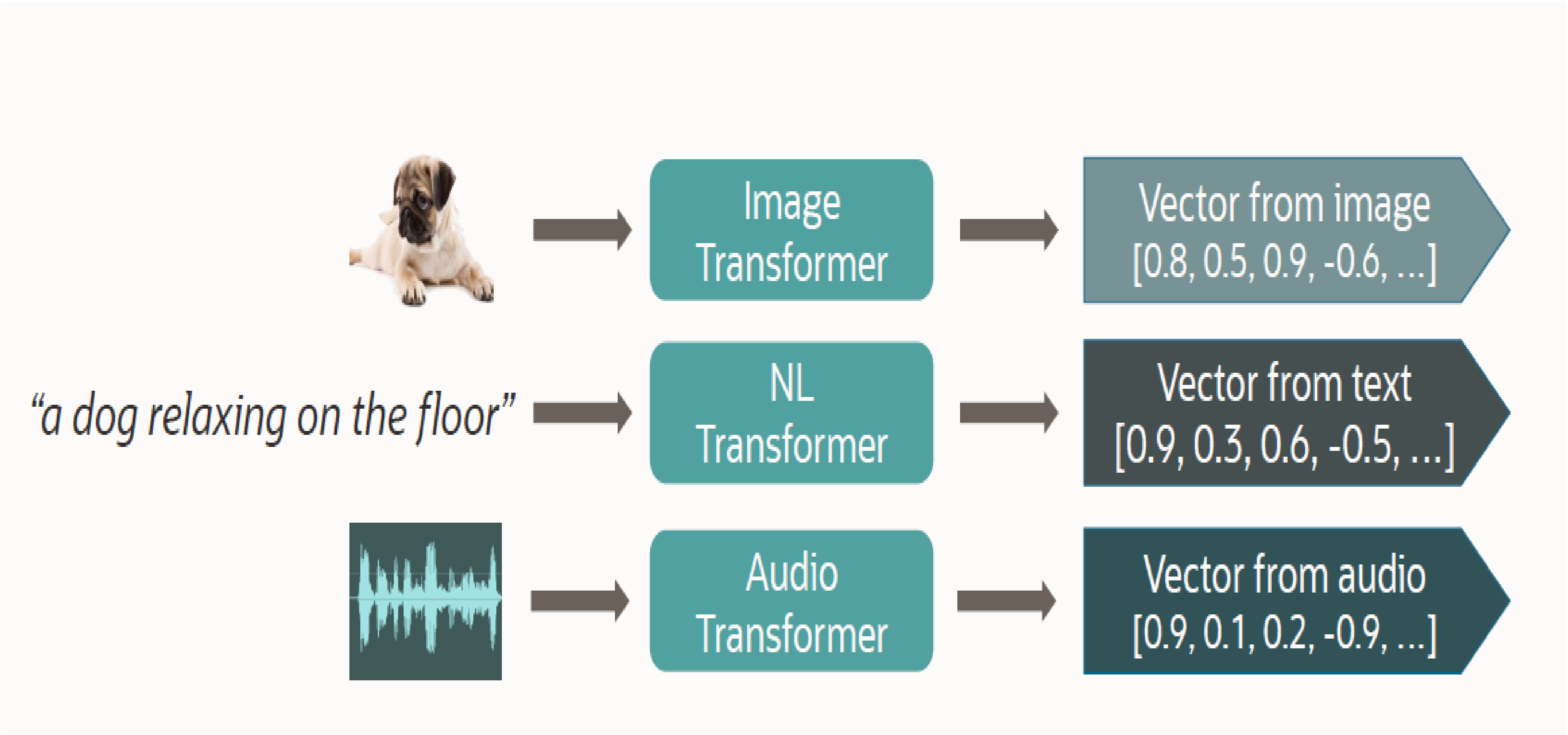




Vector Embeddings

Generating Vector Embeddings:

Used an embedding model to convert unstructured data into vectors: images, text, audio.



Generating Vector Embeddings:

Use Pre-Created Embeddings : Load vectors directly from external files into database into

VECTOR columns USING SQL*LOADER, or map the data as external tables

. Use an external embedding service such as **text-embedding-ada-002 model in Openai**

Generate vectors outside the database with AI model providers like Open-AI, Cohere, and Google

. Create a credential using the new CREATE_CREDENTIAL() API

```
DBMS_VECTOR.CREATE_CREDENTIAL("OPENAI_CRED", auth_params);
```

. PL/SQL APIs can perform REST callouts to model providers to generate embeddings

```
model_params := '{
        "provider": "openai",
        "credential_name": "OPENAI_CRED",
        "url": "https://api.openai.com/v1/embeddings",
        "model": "text-embedding-ada-002" }'
```

. Call the new UTL_TO_EMBEDDING() API to generate the vectors from the text

```
SELECT
 DBMS_VECTOR.UTL_TO_EMBEDDING(query_text, json(model_params))
 FROM document_text;
```

Database Resident Embedding Model

. Import ONNX embedding model from Huggingface , then generate embeddings with the ONNX embedding model

. Method 1: the OML4Py EmbeddingModel2OML package to Import pretrained models in ONNX Format into Oracle database

Sample codes: generate from preconfigureded model "sentence-transformers/all-MiniLML6-v2"

```
import oml
from oml.utils  import EmbeddingModel
em = EmbeddingModel(model_name="sentence-transformers/all-MiniLM-L6")
em.export2db("ALL_MINILM_L6")
 #or
em.export2file("all-MiniLM-L6.onnx",output_dir="/mydir")
```

* if all-MiniLM-L6.onnx file can be copied to other database servers and upload it to other databases.

Database Resident Embedding Model

- Method 2: Download the augmented model all-MiniLM-L12-v2 model in ONNX format all_MiniLM_L12_v2_augmented.zip through this (link) and unzip it to the model all_MiniLM_L12_v2.onnx

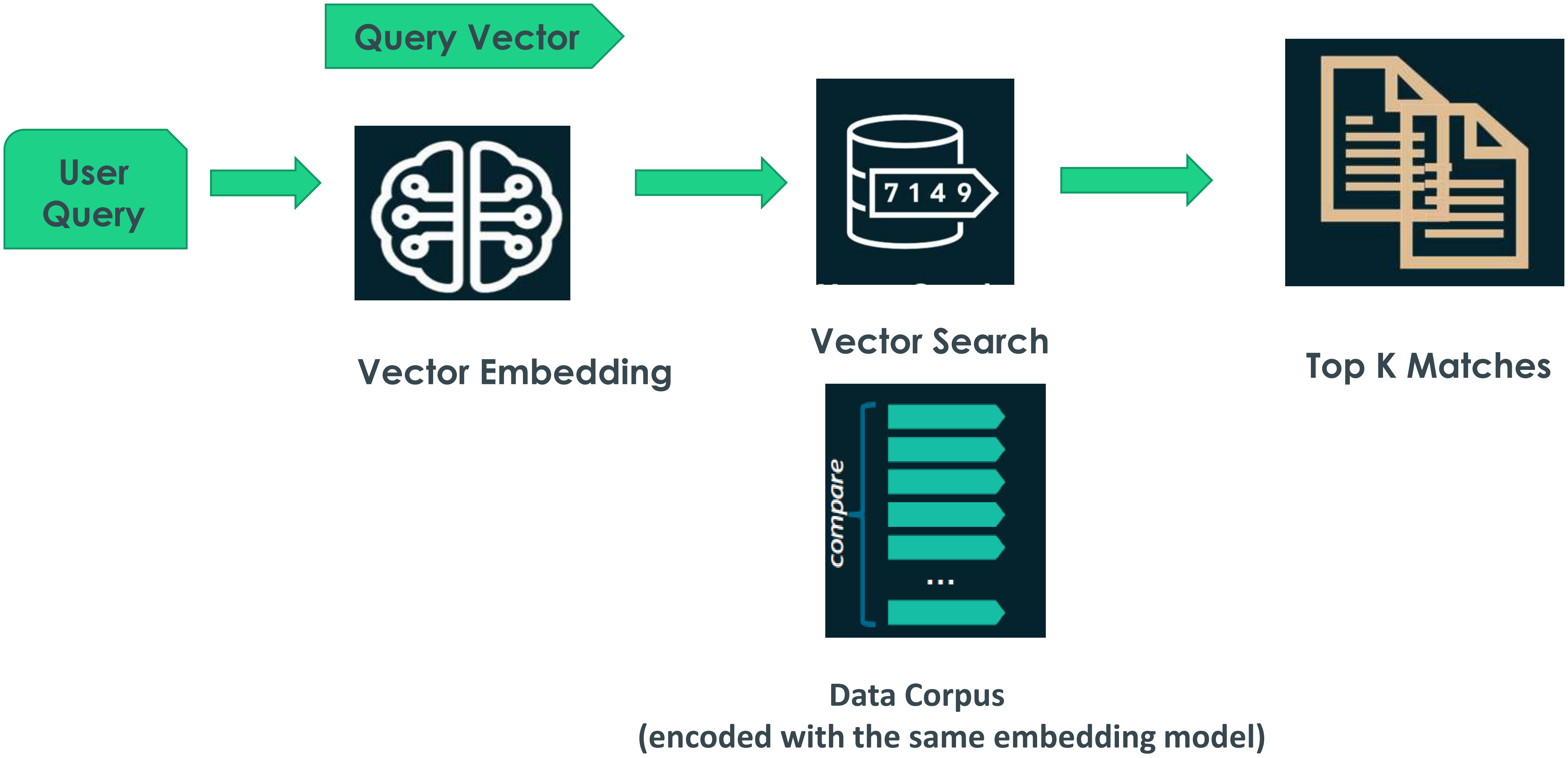
```
$ unzip all_MiniLM_L12_v2_augmented.zip
Archive:  all_MiniLM_L12_v2_augmented.zip
 inflating: all_MiniLM_L12_v2.onnx
inflating: README-ALL_MINILM_L12_V2_augmented.txt
```

Load the augmented model to Oracle 23ai database:

```
BEGIN
 DBMS_VECTOR.LOAD_ONNX_MODEL(
   directory => 'DM_DUMP',
   file_name => 'all_MiniLM_L12_v2.onnx',
   model_name => 'ALL_MINILM_L12_V2');
 END;
 /
```

Refer to this blog: https://blogs.oracle.com/machinelearning/post/use-our-prebuilt-onnx-model-now-available-for-embedding-generation-in-oracle-database-23ai

The Similarity Property Powers Vector Search



Implementing Vector Search in Oracle Database

- Generate a query vector for use in a similarity search :

```
ACCEPT text_input CHAR PROMPT 'Enter text: '
VARIABLE text_variable VARCHAR2(1000)
VARIABLE query_vector VECTOR
BEGIN
:text_variable := '&text_input';
SELECT vector_embedding(doc_model using :text_variable as data)
 into :query_vector;
END;
/
```

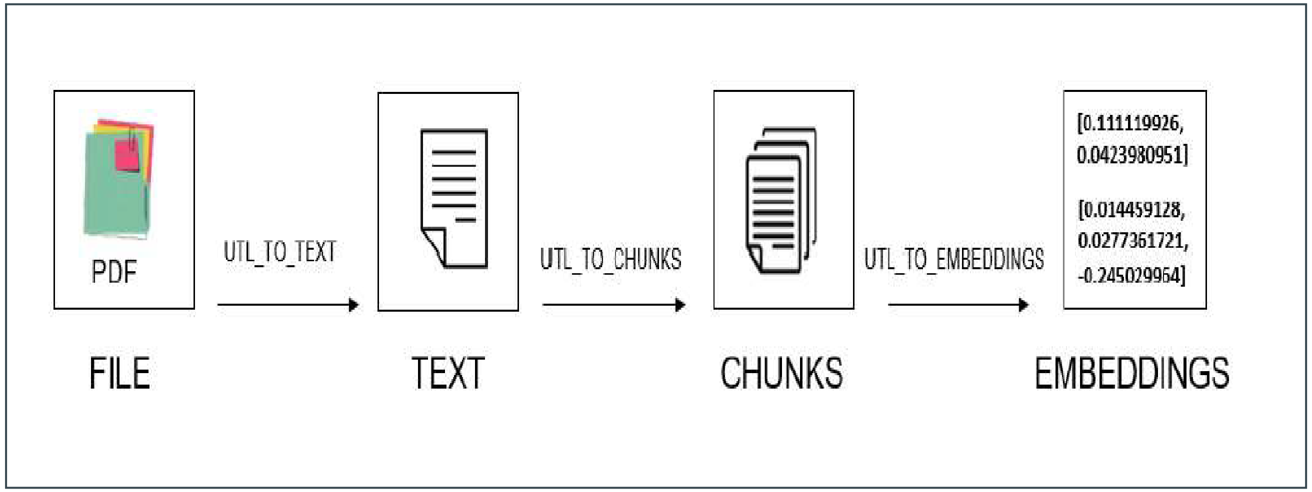
- Run a similarity search to find, the first four most relevant chunks that talk about a topic such as ‘Vector Search’

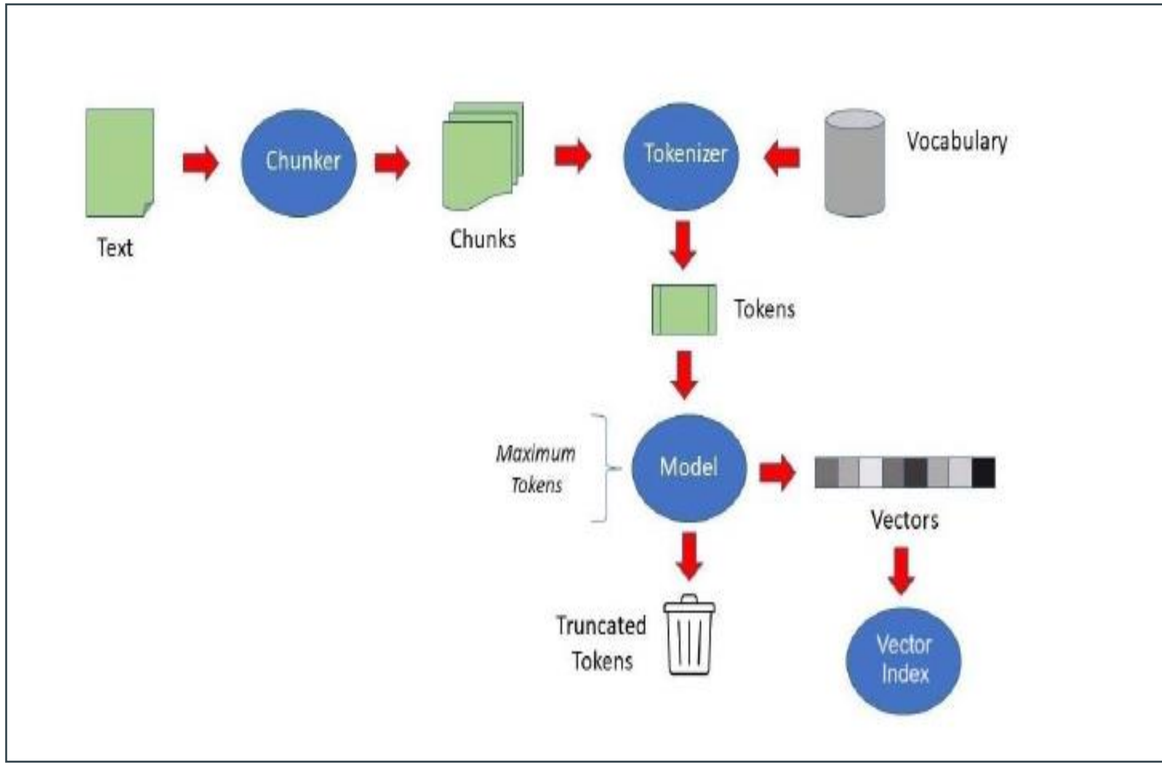
```
SELECT doc_id, chunk_id, chunk_data
FROM doc_chunks
ORDER BY vector_distance(chunk_embedding , :query_vector, COSINE)
FETCH FIRST 4 ROWS ONLY;
```

** refer to Oracle AI Vector Search User's Guide , 23ai, F87786-05, June 2024

Generating Embeddings of PDF Document

- Vector Utility PL/SQL packages to perform chunking, embedding, and text generation operations along with text processing and similarity search
    1. Converts a PDF file to plain text (using UTL_TO_TEXT)
    2. Splits the resulting text into appropriate-sized chunks (using UTL_TO_CHUNKS)
    3. Generates vector embeddings on each chunk (using UTL_TO_EMBEDDINGS) (To get an embedding, this function uses ONNX embedding models that you load into the database





Generating Embeddings of PDF document

- Chunking: breaking down large documents into smaller manageable pieces
- Impacts quality of retrieved information and generated responses
- What chunks created will eventually retrieved during inference.
- Chunk Size : too small : retrieve the too little
  too big: it may confuse the LLM

. Three main strategies:

  - Fixed-size: Equal-sized blocks
  - Semantic: Varying sizes based on contents
  - Hybrid: Combination of fixed and semantic approaches
  - Choice depends on data nature and expected

- Fixed-size Chunking
  - Divides text into chunks of predetermined size
  - Simple to implement
  - Consistent processing time
  - May break apart related information


Example (chunk Size: 50 words)
Source Doc:

Generating Embeddings of PDF document

- Semantic Chunking
  - Divides text based on meaning
  - Keeps related information together
  - More coherent chunks
  - Can be more complex to implement

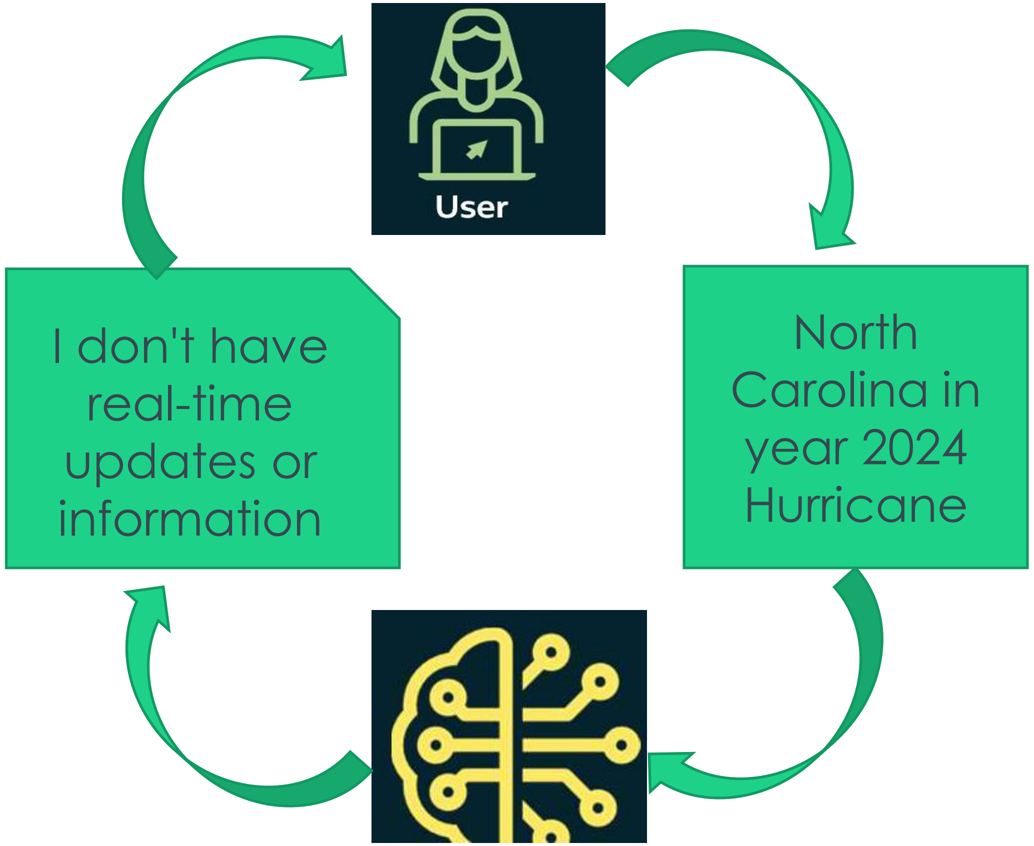
- Hybrid Chunking
  - Combines fixed-size and semantic approaches
  - Balances simplicity and coherence
  - Adaptable to different content types
  - Requires careful tuning

Refer to
From RAG-tag to RAG-nificent: Selecting Optimal algorithms for
 Conversational AI, Oracle Cloudworld presentation by Ago Canepa
Amir Rezaeian

AI Vector Search for Retrieval Augmentation Generation(RAG)

Oracle AI Vector Search for RAG

. Limitations of using LLMs to answer my questions



As of now, I don't have real-time updates or information about specific hurricanes in 2024. If there has been a hurricane affecting North Carolina in 2024, local news outlets and the National Hurricane Center would provide the most current information on the storm's path, impact, and any damages.

For the latest details, including forecasts, safety measures, and recovery efforts, I recommend checking reliable sources like the National Weather Service or local emergency management agencies. If you have other questions about hurricanes or need information on preparedness, feel free to ask!

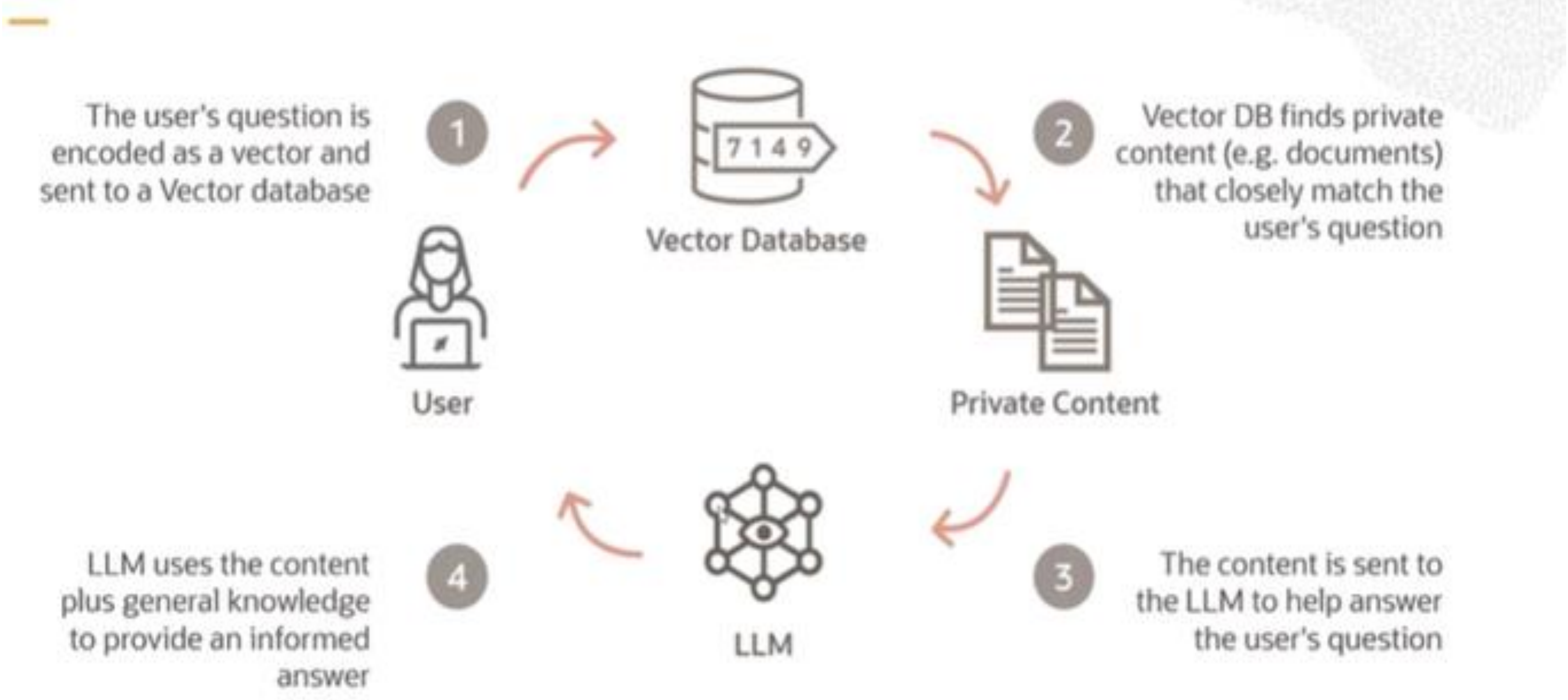
. Large Language Models (LLMs) are trained on a broad range of data from the internet.

. LLMs were trained with the data that were available in the internet at the of training.

. LLMs training has no access to private enterprise data

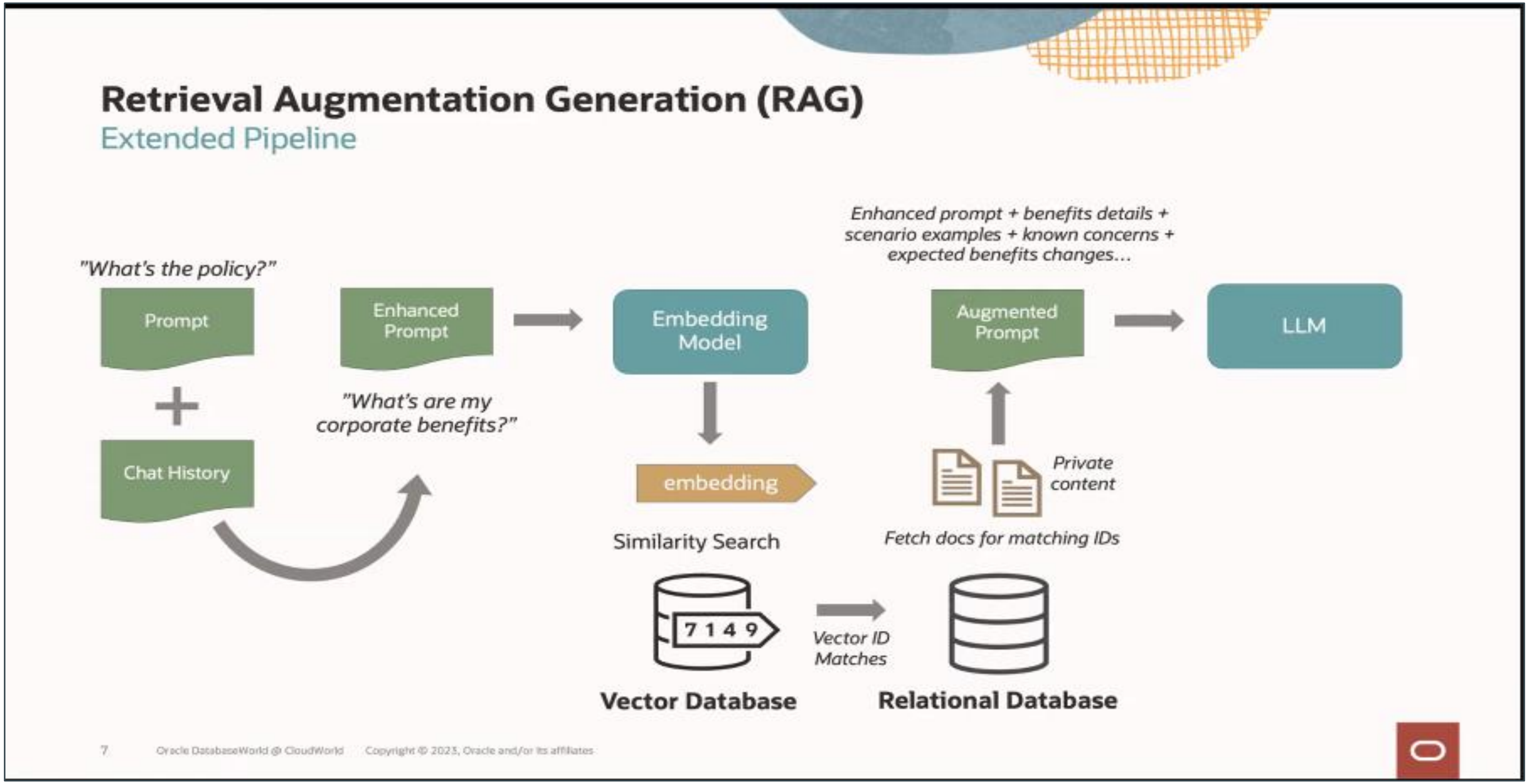
Oracle AI Vector Search for RAG

. Vector database augments Generative AI by retrieving detailed, often private contents needed to answer questions



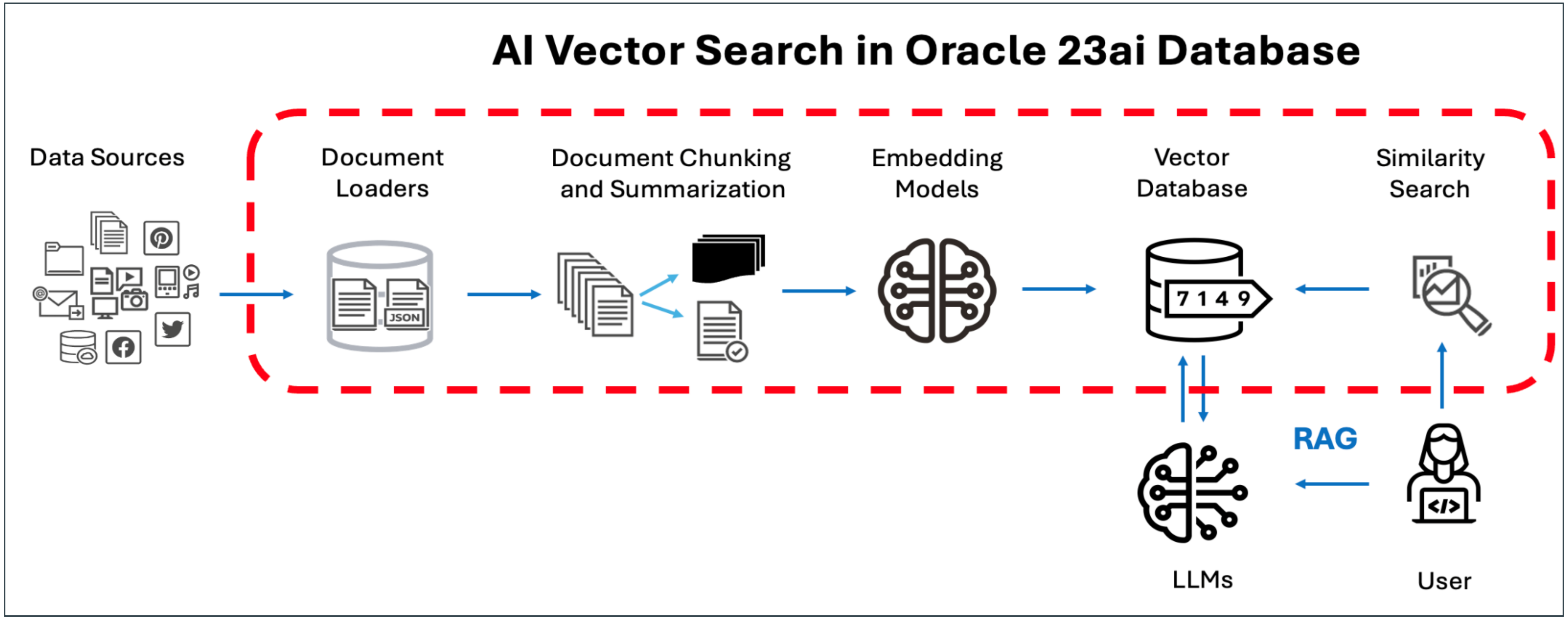
Oracle AI Vector Search for RAG

. Vector database in Retrieval Augmentation Generation(RAG)



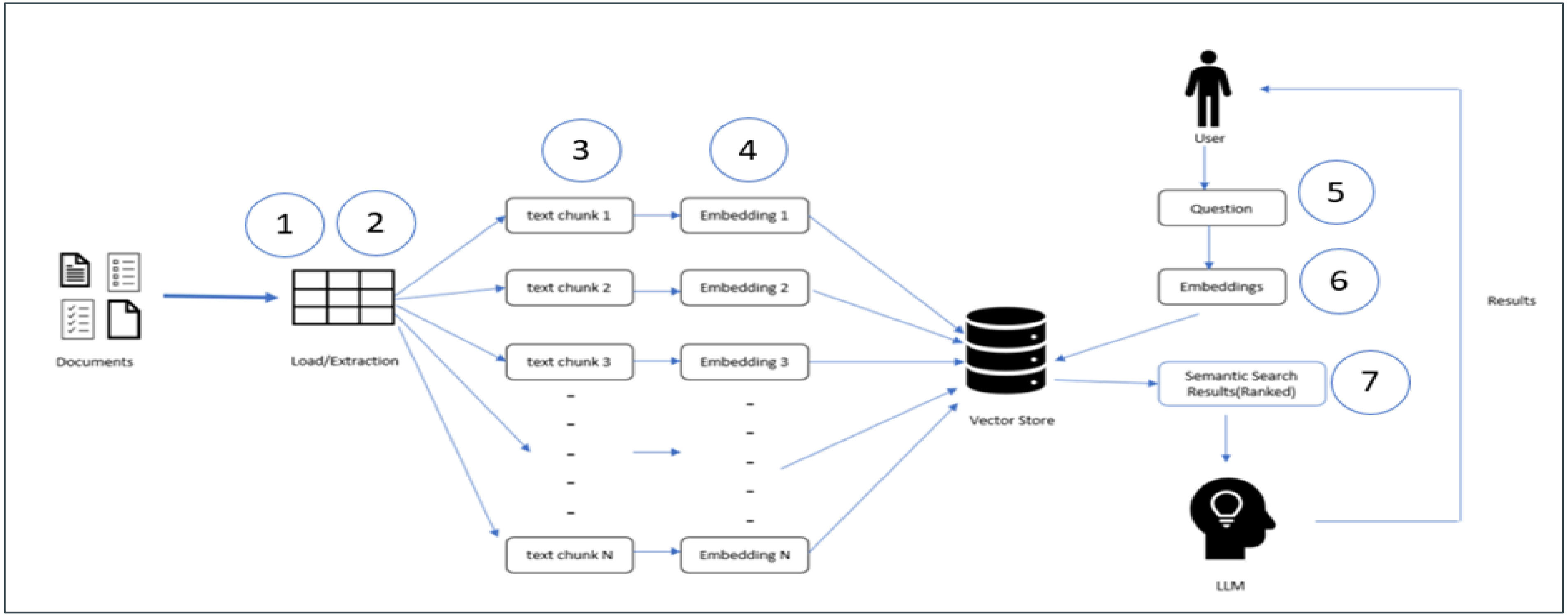
Building a RAG application with Vector Search and LangChain

. A simple RAG application using Oracle AI Vector Search and Langchain framework **
** Note: For the complete details of this sample application, refer to Oracle [AI Vector Search livelab]



Building a RAG Application with Vector Search and LangChain

1. Load the pdf document such as Oracle database 23ai user guild.pdf
2. Transform the pdf document to text.
3. Chunk the text document into smaller pieces
4. Embed the chunk as vectors to be stored in Oracle Database 23ai
5. Ask the question for the prompt. The question will be vectorized in the same embedding model
6. The question will be passed to Oracle Database 23ai to do a similarity search
7. The search results (context) are passed to the LLM to generate the response.



Building a RAG Application with Vector Search and LangChain

1. Load the pdf document

#Creating a pdf reader object pdf:

pdf = PdfReader('filename.pdf')

```
[4]:  # RAG Step 1 - Load the document

      # creating a pdf reader object
      pdf = PdfReader('oracle-database-23ai-new-features-guide.pdf')

      # print number of pages in pdf file
      print("The number of pages in this document is ",len(pdf.pages))
      # print the first page
      print(pdf.pages[0].extract_text())

      The number of pages in this document is  126

       Oracle Database®
      Oracle Database New Features



      Release 23ai
      F48428 -20
      May 03, 2024
```

2. Transform the pdf document to text

for page in pdf.pages
    text += page.extract_text()

```
# RAG Step 2 - Transform the document to text

if pdf is not None:
  print("Transforming the PDF document to text...")
text=""
for page in pdf.pages:
    text += page.extract_text()
    #print(text)
print("Your have transformed the PDF document to text format")

Transforming the PDF document to text...
Your have transformed the PDF document to text format
```

Building a RAG Application with Vector Search and LangChain

3. Split the text document into smaller chunks

text_splitter = CharacterTextSplitter(separator="\n",chunk_size=800,chunk_overlap=100,length_function=len)

chunks = text_splitter.split_text(text)

6. RAG Step 3 - Split the text document into smaller chunks

[6]:
```
# RAG Step 3 - Chunk the text document into smaller chunks

text_splitter = CharacterTextSplitter(separator="\n",chunk_size=800,chunk_overlap=100,length_function=len)
chunks = text_splitter.split_text(text)
print("Split successful, printing the first chunk")
print(chunks[0])
```

```
Split successful, printing the first chunk
Oracle Database®
Oracle Database New Features



Release 23ai
F48428 -20
May 03, 2024


 2 Oracle Database Oracle Database New F eatures, Release 23ai
F48428 -20
Copyright © 2022, 2024, Oracle and/or its affiliates.
This software and related documentation are provided under a license agreement containing
restrictions on use and disclosure and are protected by intellectual property la ws. Except as
expressly permitted in your license agreement or allowed by law, you may not use, copy,
reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or
display any part, in any form, or by any means. Reve rse engineering, disassembly, or
decompilation of this software, unless required by law for interoperability, is prohibited.
```

Building a RAG application with Vector Search and LangChain

4. Using an embedding model to embed the chunk as vectors to be stored in Oracle Database 23ai

The embedding model used here **all-MiniLM-L6-v2** from HuggingFace

model_4db = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")

knowledge_base = OracleVS.from_documents(docs, model_4db, client=conn23c,
table_name="MY_DEMO4",
distance_strategy="DistanceStrategy.DOT_PRODUCT")

```
# Initialize model
model_4db = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")

# Configure the vector store with the model, table name, and using the indicated distance strategy for the similarity search and vecto
s1time = time.time()
knowledge_base = OracleVS.from_documents(docs, model_4db, client=conn23c, table_name="MY_DEMO4", distance_strategy=DistanceStrategy.DO
#knowledge_base = OracleVS.from_documents(docs, model_4db, client=conn23c, table_name="MY_DEMOOCI", distance_strategy="DistanceStrateg
s2time =  time.time()
print( f"Vectorizing and inserting chunks duration: {round(s2time - s1time, 1)} sec.")
```

```
Table dropped successfully...
Table created successfully...
Vectorizing and inserting chunks duration: 14.7 sec.
```

Building a RAG Application with Vector Search and LangChain

Check the database table created by Langchain: Columns: id, metadata, chunk text and corresponding vector :

Select * from my_demo4 where rownum < 2

```
# loads the SQL magic extensions
%load_ext sql
# Connect to Oracle using oracledb library
# this is for legacy cx_Oracle %sql oracle+cx_oracle://scott:tiger@localhost:1521?service_name=FREEPDB1

%sql oracle+oracledb://vector:vector@localhost:1521?service_name=ORCLPDB1

%sql select * from my_demo4 where rownum < 2
```

[26]:

id	text	metadata	embedding
	Oracle Database 23ai is the next long - term support releas e of Oracle Database. It includes over 300 new features with a focus on artificial intelligence (AI) and developer productivity. Features such as AI Vector Search enable you to leverage a		array('f', [-0.03658159822225571, -0.01941094547510147, -0.010810347273945808, 0.00278968526981771, 0.03951172158122063, -0.009920173324644566, 0.023954592645168304, 0.029138969257473946, -0.10468441992998123, -0.006722270045429468, -0.025112343952059746, -0.016803069040179253, 0.06708680838346481, -0.02237202227115631, -0.018269244581460953, -0.0010180049575865269, -0.03190352022647858, 0.008001388050615788, -0.031326111406087875, -0.09920547157526016, -0.0300012044608593, 0.02320478856563568, -0.011083254590630531, -0.052901316434144974, 0.016076592728495598, 0.08270624279975891, 0.014443156309425583, -0.058435216546058655, 0.001321742543950677, -0.046375907957553864, 0.019295541569590057, 0.014564533717930317, 0.020035443827509888, 0.021781660616397858, -0.13767936825752258, -0.023051675409078598, -0.0814642533659935, -0.024693731218576431, 0.026278920471668243, -0.03430929780006409, -0.05079200491309166, -0.05630091205239296, -0.100071981549263, 0.012328796088695526, 0.0861680880188942, 0.021900791674852371, -0.06671038269996643, -0.027864944189786911, 0.029703803360462189, 0.08335181325674057, -0.11445116251707077, 0.0036093010567128660, -0.026479218155145645, 0.018975013867020607, 0.038737919181585310, 0.027217395603656770, 0.020795857533812523, 0.030421249568462372, -0.0032251004595309496, -0.011276564560830593, 0.06390005350112915, -0.07379689067602158, 0.09508409351110458, -0.011043175123631954, 0.021890377625823020, 0.0012366424780339003, -0.008321236819028854, -0.031668595969676970, 0.044548097997903824, -0.12191140651702881, -0.03850025311112404, 0.09607793390750885, -0.054113198071718216, 0.021668544039130210, -0.08652691543102264, 0.09547340869903564, 0.08389736711978912, 0.004319169558584690, 0.028444334864616394, -0.024005504325032234, -0.025050297379493713, 0.027645906433463097, -0.04765533283352852, 0.010380342602729797, 0.05332623422145843, -0.05691875144839287, 0.07027460634708405, -0.031938884407281876, 0.049787215888500214, 0.010874697938561440, 0.042229048907756805, -0.03480788320302963, 0.025079077109694480, -0.05916276201605797, 0.014800400473177433, 0.012537235394120216, -0.06082005426287651, -0.08434253185987473, 0.019186334684491158, 0.029068531468510628, -0.04787250980734825, 0.11926072835922241, 0.013148070313036442, -0.04680410400032997, -0.08488404005765915, 0.023632651194930077, 0.04484761133790016, -0.0371067076921463, 0.05358376353979111, -0.06544138491153717, -0.016474299132823944, 0.002346457913517952, -0.014560796320438385, -0.15739497542381287, 0.05741714686155319, 0.04043841361999512, -0.1364080011844635, 0.0117074279114604, 0.04871664196252823, 0.05271068960428238, 1.4189237845130265e-05, -0.001957066124305129, -0.04116304963827133, 0.014474596828222275, 0.022032873705029488, 0.021832393482327460, -0.10601190477609634, 3.872511487808496e-33, -0.06387393921613693, 0.004077907651662826, -0.006564603652805090, -0.06890714168548584, 0.07419261336326599, -0.04284325987100601, -0.01643178798258304,

Building a RAG Application with Vector Search and LangChain

5. Build the prompt to query the document

user_question = (‘Tell me more about AI Vector Search”)

```
# RAG Step 5 - Build the prompt to query the document

user_question = ("Tell me more about AI Vector Search")
print ("The prompt to the LLM will be:",user_question)
```

```
The prompt to the LLM will be: Tell me more about AI Vector Search
```

Start the Vector search based on the prompt , Set up time to measure performance:

result_chunks=knowledge_base.similarity_search(user_question)

```
# Set up time to measure performance

# code not needed, used to measure time for search and return only, used for measuring performance
if user_question:
 s3time =  time.time()
 result_chunks=knowledge_base.similarity_search(user_question)
 s4time = time.time()
 print(f"Search for the user question in the Oracle Database 23ai and return similar chunks duration: {
 print("")
```

```
Search for the user question in the Oracle Database 23ai and return similar chunks duration: 0.0 sec.
```

Building a RAG Application with Vector Search and LangChain

Choose an LLM to generate your response

Choose 1: Use Meta Llama LLM through OCI GenAI service

Sets up the OCI GenAI Service LLM to use Meta Llama

```
ENDPOINT = "https://inference.generativeai.us-chicago-1.oci.oraclecloud.com"
COMPARTMENT_OCID = COMPARTMENT_OCID
EMBED_MODEL="meta.llama-2-70b-chat"
print(ENDPOINT)

# set the LLM to get response
llm = OCIGenAI(
  model_id="meta.llama-2-70b-chat",
  service_endpoint="https://inference.generativeai.us-chicago-1.oci.oraclecloud.com",
  compartment_id=COMPARTMENT_OCID,
  model_kwargs={"temperature": 0.7, "top_p": 0.75, "max_tokens": 2000},
  auth_type="API_KEY",
)
print("The LLM model you will use is meta.llama-2-70b-chat from OCI GenAI Service")
```

```
https://inference.generativeai.us-chicago-1.oci.oraclecloud.com
The LLM model you will use is meta.llama-2-70b-chat from OCI GenAI Service
```

Building a RAG Application with Vector Search and LangChain

Choose 2: use Cohere LLM through OCI GenAI LLM

Set up OCI GenAI service to use the Cohere LLM

ENDPOINT = "https://inference.generativeai.us-chicago-1.oci.oraclecloud.com"

llm = OCIGenAI(model_id="cohere.command",
service_endpoint="https://inference.generativeai.us-chicago-1.oci.oraclecloud.com",
compartment_id=COMPARTMENT_OCID,
model_kwargs={"temperature": 0.7, "top_p": 0.75, "max_tokens": 2000},
auth_type="API_KEY",)

```
ENDPOINT = "https://inference.generativeai.us-chicago-1.oci.oraclecloud.com"
COMPARTMENT_OCID = COMPARTMENT_OCID
print(ENDPOINT)

# set the LLM to get response
llm = OCIGenAI(
  model_id="cohere.command",
  service_endpoint="https://inference.generativeai.us-chicago-1.oci.oraclecloud.com",
  compartment_id=COMPARTMENT_OCID,
  model_kwargs={"temperature": 0.7, "top_p": 0.75, "max_tokens": 2000},
  auth_type="API_KEY",
)
print("The LLM model you will use is cohere.command from OCI GenAI Service")
```

```
https://inference.generativeai.us-chicago-1.oci.oraclecloud.com
The LLM model you will use is cohere.command from OCI GenAI Service
```

Building a RAG application with Vector Search and LangChain

Set up a template for the question and context, and instantiate the database retriever object to use the retriever to retrieve context from Oracle Database 23ai

template = """Answer the question based only on the following context: {context} Question: {question} """

prompt = PromptTemplate.from_template(template)

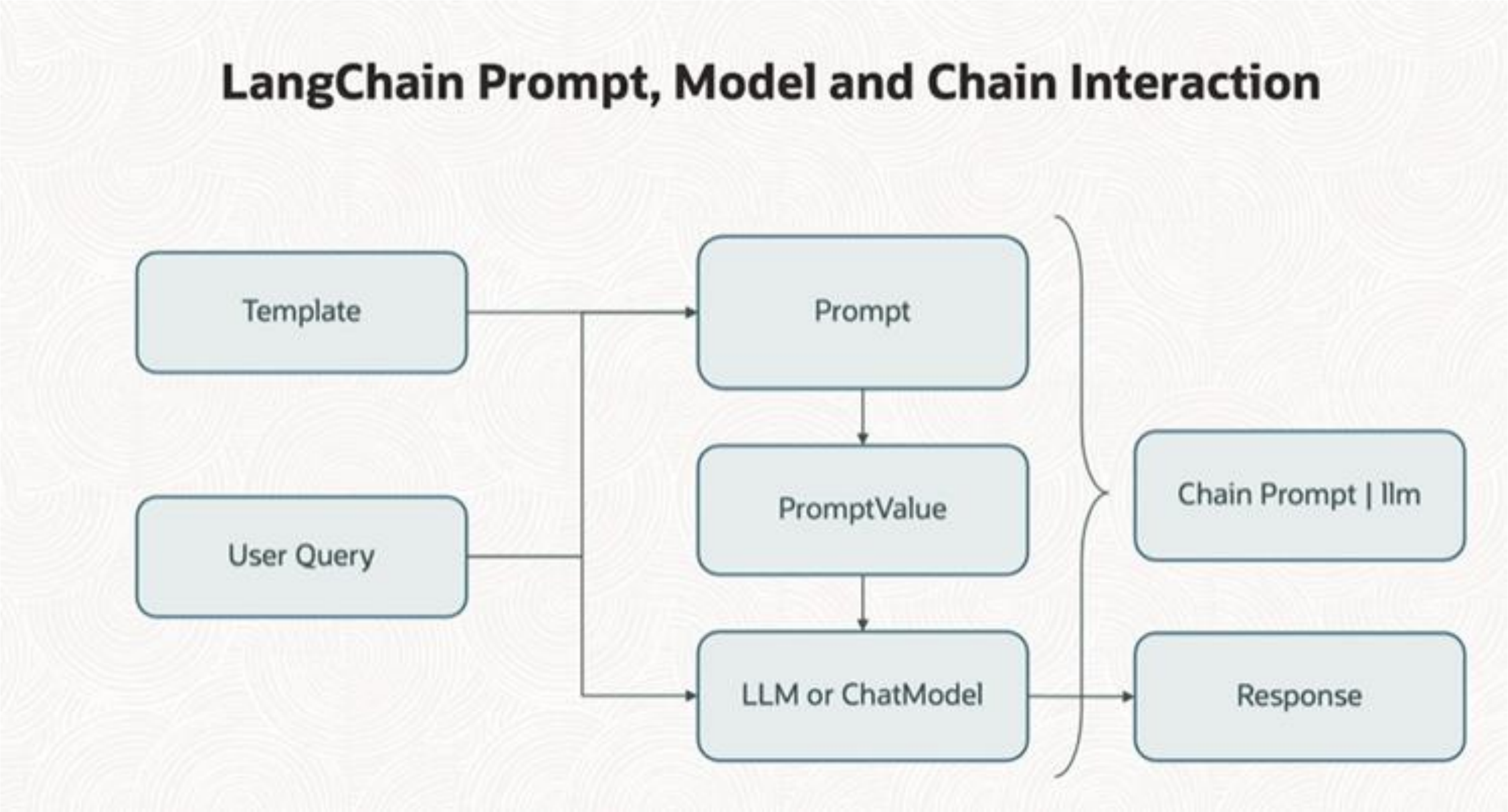
retriever = knowledge_base.as_retriever()

```
# Set up a template for the question and context, and instantiate the database retriever object

template = """Answer the question based only on the following context:
           {context} Question: {question} """
prompt = PromptTemplate.from_template(template)
retriever = knowledge_base.as_retriever()
print("The template is:",template)
```

```
The template is: Answer the question based only on the following context:
           {context} Question: {question}
```

Building a RAG application with Vector Search and LangChain

LangChain provides framework for creating chains of component , including LLMs and other type of components.



Building a RAG application with Vector Search and LangChain

the LangChain pipeline : chains all the components together to produce an LLM response:

with context: **Retrieve the context**, construct the prompt with the **question** and context

Pass to LLM for the response

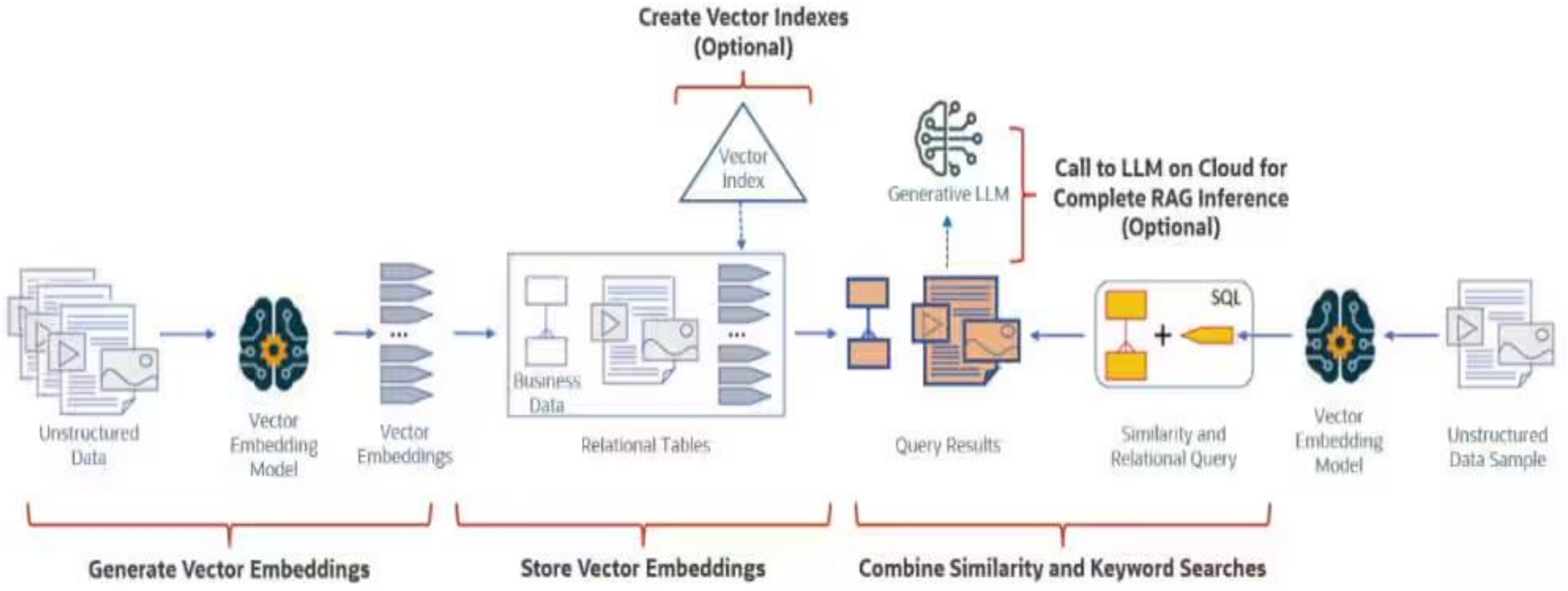
chain = **({"context": retriever, "question": RunnablePassthrough()} | prompt| llm | StrOutputParser())**

```python
# RAG Step 6. and 7 - Chain the entire process together, retrieve the context, construct the prompt wi

s5time = time.time()
print("We are sending the prompt and RAG context to the LLM, wait a few seconds for the response...")
chain = (
  {"context": retriever, "question": RunnablePassthrough()}
     | prompt
     | llm
     | StrOutputParser()
)
response = chain.invoke(user_question)
print(user_question)
print(prompt)
print(response)
# Print timings for the RAG execution steps

s6time = time.time()
print("")
print( f"Send user question and ranked chunks to LLM and get answer duration: {round(s6time - s5time,
```

```
We are sending the prompt and RAG context to the LLM, wait a few seconds for the response...
Tell me more about AI Vector Search
input_variables=['context', 'question'] template='Answer the question based only on the following context:\n
{context} Question: {question} '
AI Vector Search is a feature introduced in Oracle Database 23 AI that allows you to run AI-powered vector si
milarity searches within the database. With this feature, you can leverage AI models to generate vectors from
documents, images, sound, and more, and then index and search for similarity based on those vectors. This eli
minates the need to move business data to a separate vector database, reducing complexity and improving secur
ity.

Vector Indexes are a crucial component of the AI Vector Search, they are used to efficiently store and search
high-dimensional vector data by organizing similar items together, which makes the search process efficient.

With AI Vector Search, you can combine sophisticated business data searches with AI vector similarity searche
s using simple SQL queries, enabling the rapid development of AI-driven applications.

Would you like to know more about the advantages of using AI Vector Search?

Send user question and ranked chunks to LLM and get answer duration: 5.6 sec.
```

Finish

GenAI RAG Architecture with Vector Search



Summary

- Vector and Vector search
- Vector Search in Oracle Database 23ai
- Vector Embedding Methods
- AI Vector Search for RAG in GenAI

Analytics and Data
ORACLE USER COMMUNITY

Save the Date

Analytics and Data
Summit 2025

*Registration is now **open**!*

April 8-10, 2025
Oracle Conference Center
Redwood Shores, California

https://andouc.org/andsummit2025/

Helpful Links –

ORACLE ANALYTICS VIDEOS:
https://www.youtube.com/@OracleAnalytics/videos

OAC SEPTEMBER NEW FEATURES VIDEOS BY ORACLE: https://bit.ly/OACSept24Features

OAC NEW FEATURES DOCUMENTATION BY ORACLE:
https://docs.oracle.com/en/cloud/paas/analytics-cloud/acswn/index.html#GUID-CFF90F44-BCEB-49EE-B40B-8D040F02D476

ORACLE ANALYTICS COMMUNITY:
https://community.oracle.com/products/oracleanalytics

ORACLE ANALYTICS LIBRARY/EXAMPLES:
https://www.oracle.com/business-analytics/data-visualization/examples/

ORACLE ANALYTICS LIVE DEMOS:
https://www.oracle.com/business-analytics/data-visualization/demos/