ORACLE

Graphs, Graph-RAGs and LLMs

An Introduction

Damien Hilloulin, Melli Annamalai, Marouane Maatouk

Oracle

Who Are We?

Damien Hilloulin,
Research Manager

Zurich, Switzerland

Melli Annamalai
Distinguished Product Manager

Nashua, NH

Marouane Maatouk
Senior Member of Technical Staff

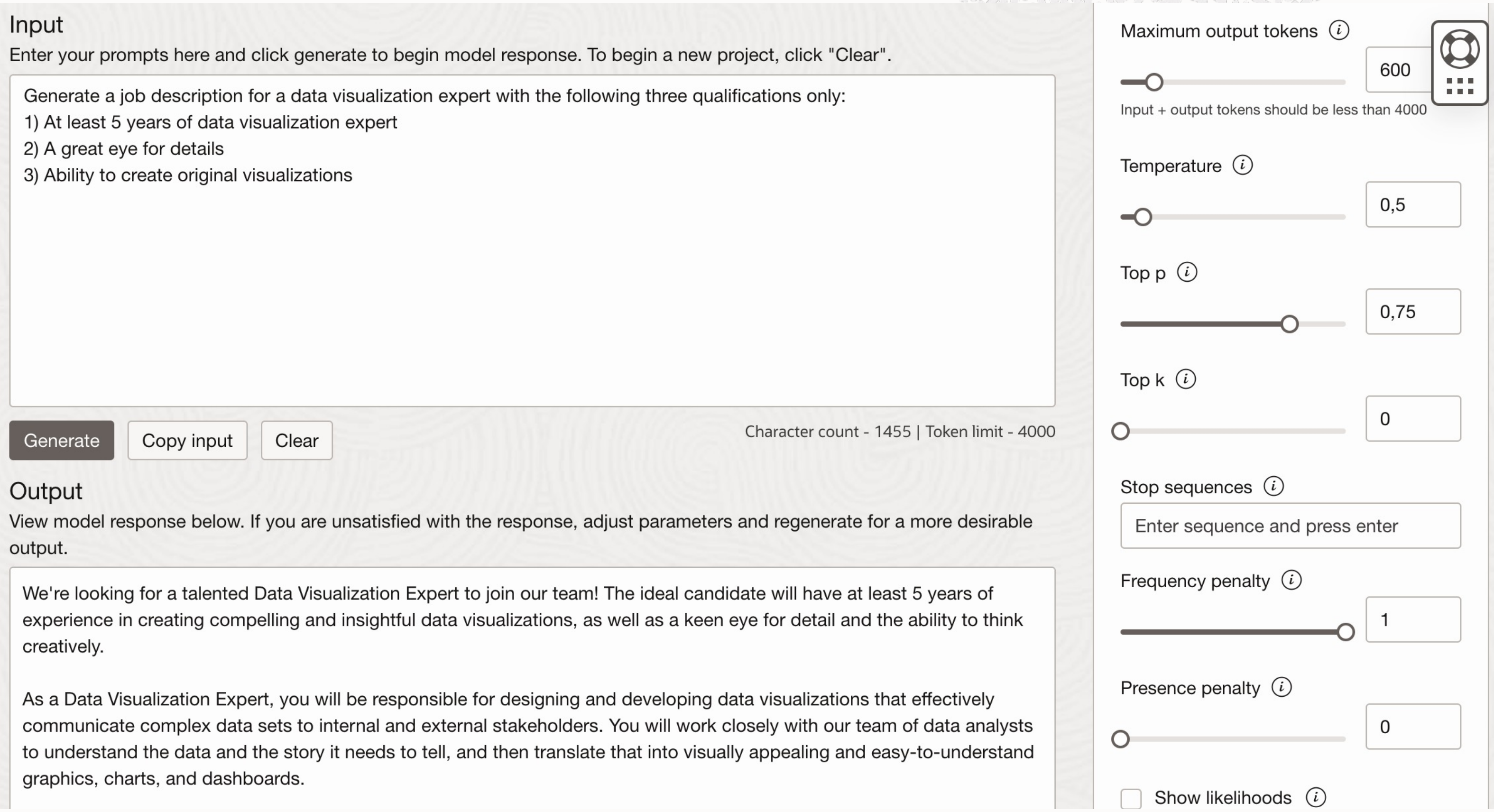
Casablanca, Morocco

Safe harbor statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Large Language Models (LLMs)

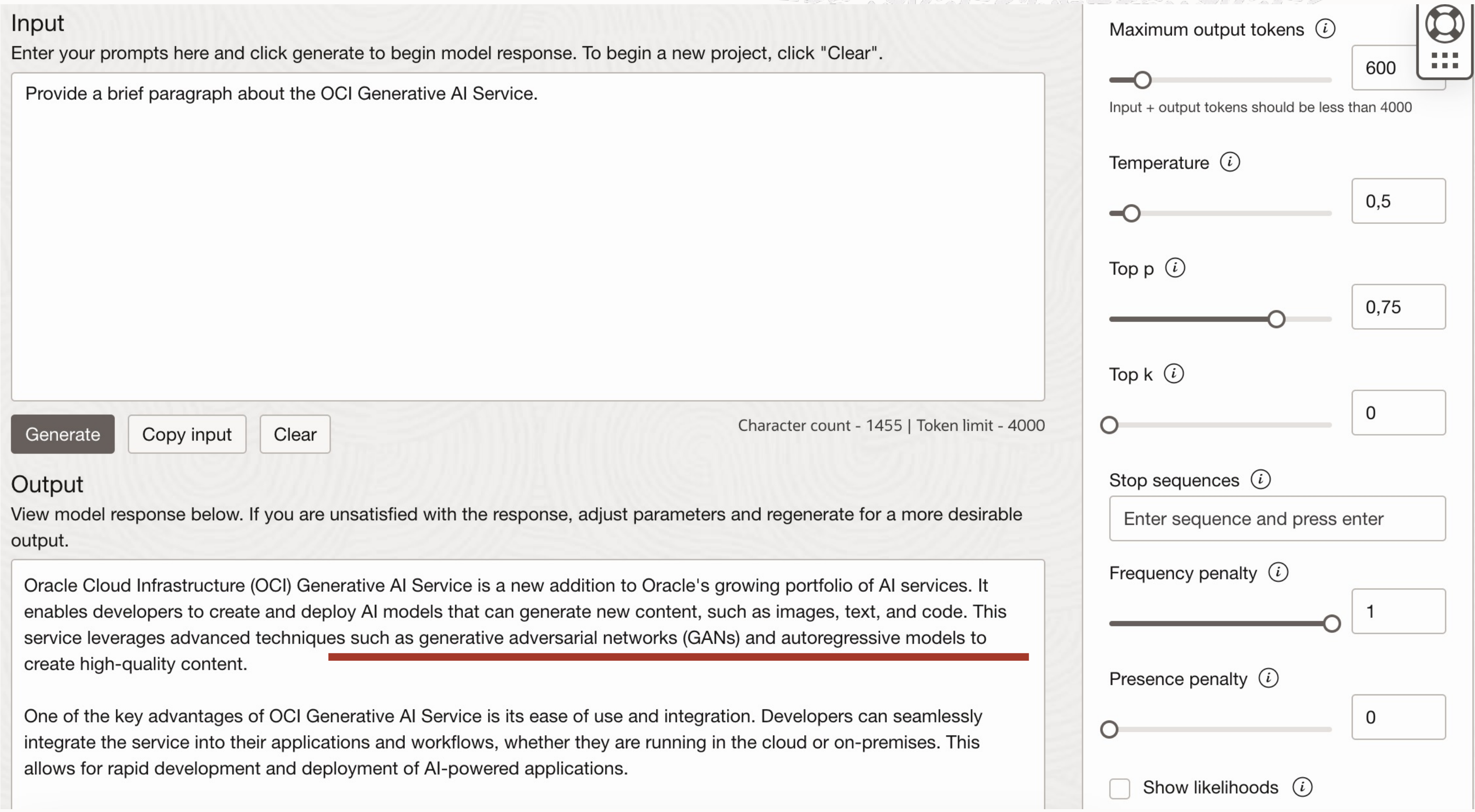
- Powerful AI models that can process and generate human language text
- Model created by training on massive volumes of data
- Researchers and practitioners are racing to learn how LLMs can help their products and their business



Some Challenges Faced by LLMs

- LLMs only as good as the data they are trained on
  - They can get out-of-date
  - They can have bias
  - They are expensive to re-train

- Sometimes answers are misleading and incorrect – “hallucination”
  - Because they generate output based on patterns in data *they have been trained on* rather than from a deep understanding of facts

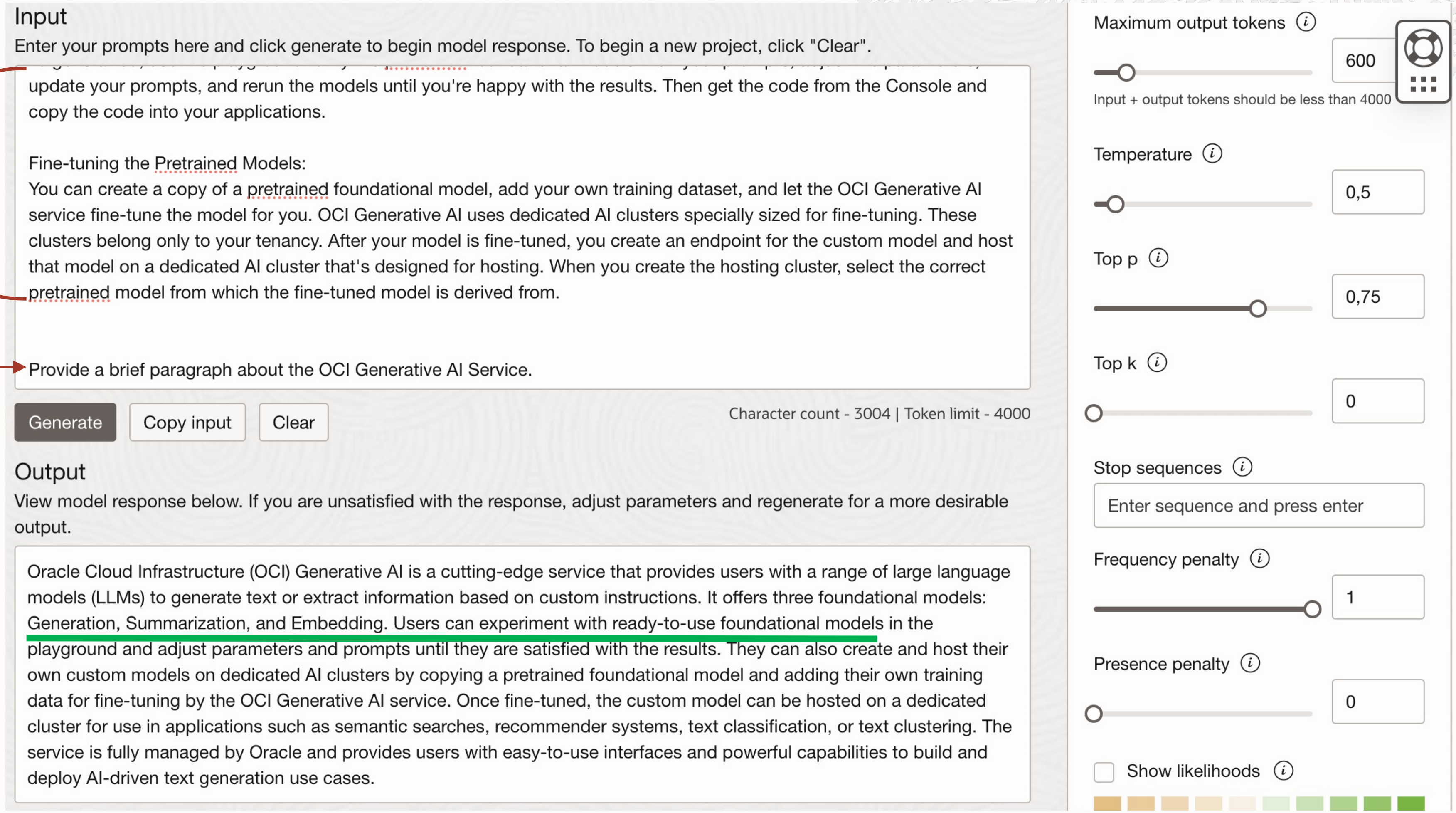


Not True

Improved Responses by Adding Context in the Prompt

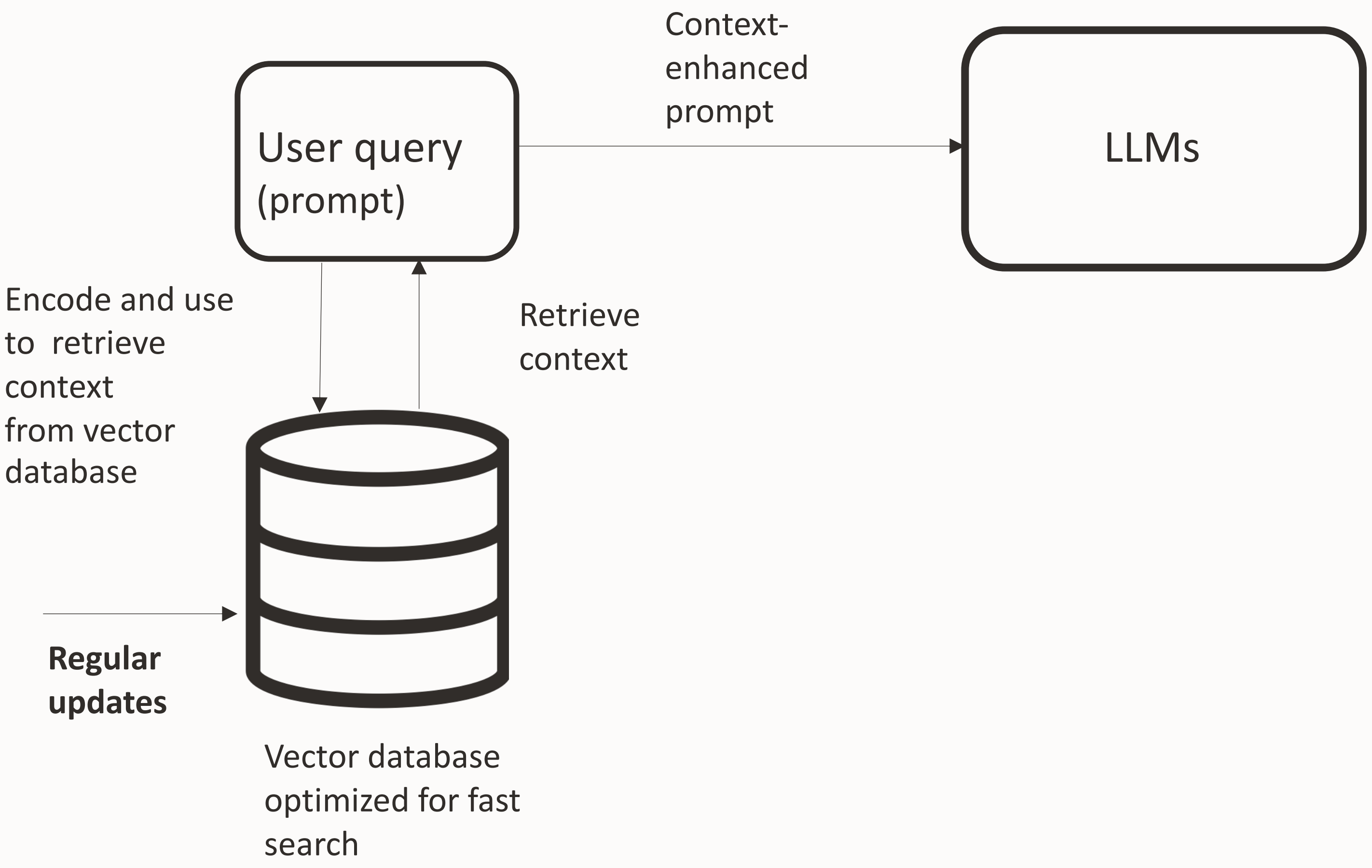
context

query



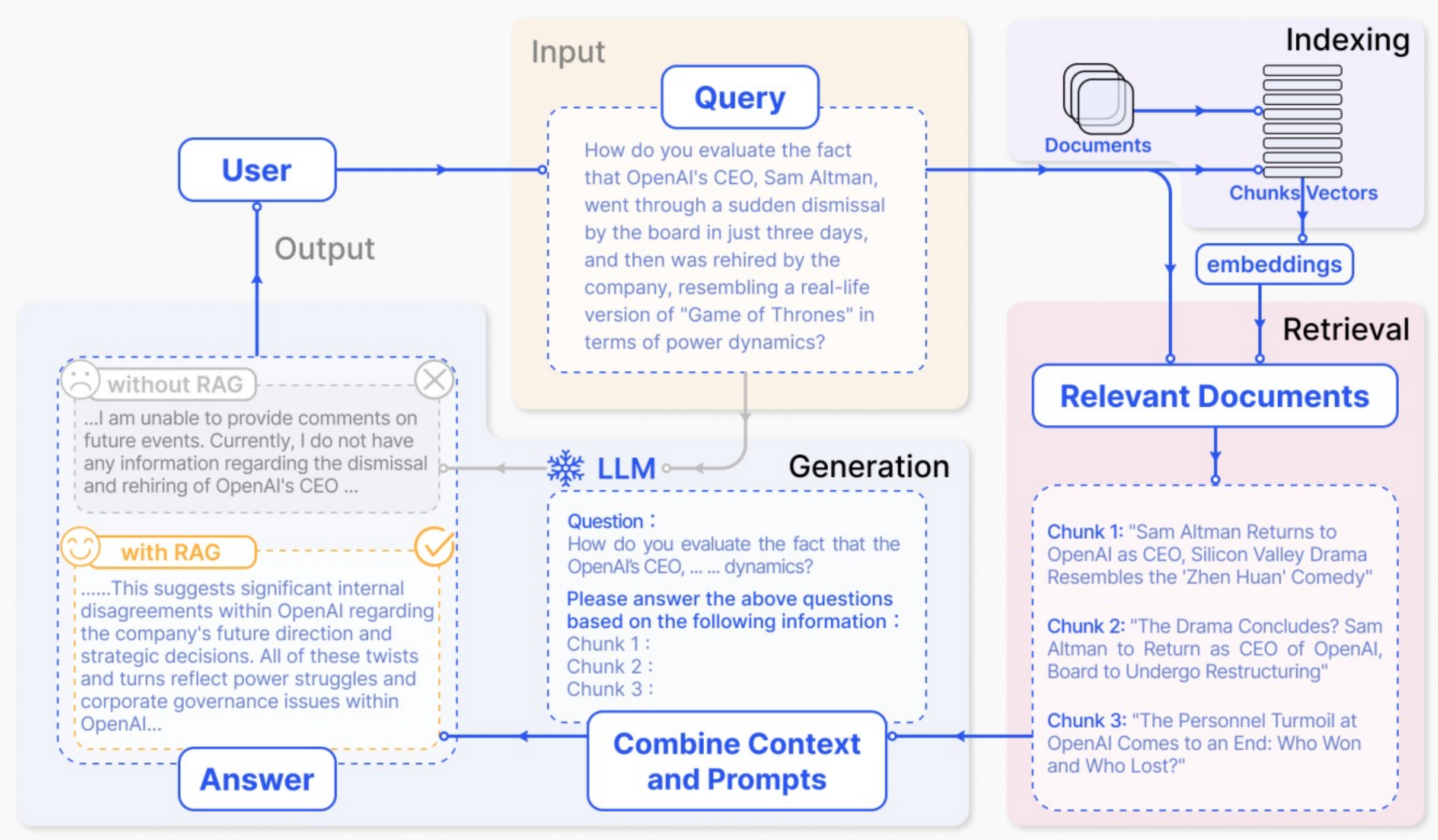
This is correct

RAG: Retrieval-Augmented Generation



- Use latest data to provide context to LLM
- Create encodings (referred to as embeddings) that are stored as vectors in a vector database
- User query is encoded and matched with stored vectors
- Top matches are retrieved and provided as context with the prompt

Example



Flowchart from: Retrieval-Augmented Generation for Large Language Models: A Survey. Gao et al

Retrieval-Augmented Generation with Langchain and OCI GenAI Service

```python
import os
from langchain.document_loaders import PyPDFLoader
from langchain.vectorstores.faiss import FAISS
from langchain_community.embeddings import OCIGenAIEmbeddings
from langchain import PromptTemplate
from langchain_community.llms import OCIGenAI
from langchain.chains import RetrievalQA


_OCI_AUTH_ARGS = {
    "service_endpoint": os.environ["SERVICE_ENDPOINT"],
    "compartment_id": os.environ["COMPARTMENT_ID"],
    "auth_type": "INSTANCE_PRINCIPAL"
}

_LLM_KWARGS = {"temperature": 0.7, "max_tokens": 4000}

# ----------------------------------------
# 1. Indexing
# ----------------------------------------
loader = PyPDFLoader("mack_resume.pdf")
documents = loader.load()
# Set up the encoder
embeddings = OCIGenAIEmbeddings(
    model_id="cohere.embed-english-v3.0",
    **_OCI_AUTH_ARGS
)
# Index the documents in a vector database
vector_store = FAISS.from_documents(documents, embeddings)
```

```python
# ----------------------------------------
# 2. Retrieval
# ----------------------------------------
user_query = """
Does Mack have a direct experience with Java and MapReduce?
"""
top_k = 5 # Number of top matches to return
# Create a retriever from the vector store
retriever = vector_store.as_retriever(k=top_k)
# Fetch the relevant documents to the user query
documents = retriever.get_relevant_documents(user_query)
```

```python
# ----------------------------------------
# 3. Generation
# ----------------------------------------
# Set up a prompt template
PROMPT = PromptTemplate.from_template(
"""You are an assistant for question-answering tasks.
Use the following pieces of retrieved context to answer the question in a concise manner.
If you don't know the answer, just say that you don't know.
Question: {question}
Context: {context}
Answer:"""
)

# Set up OCIGenAI client
llm = OCIGenAI(
    model_id="cohere.command",
     model_kwargs=_LLM_KWARGS,
    **_OCI_AUTH_ARGS,
)

# Setup the RAG pipeline
pipeline = RetrievalQA.from_chain_type(
    llm=llm,
    retriever=retriever,
    chain_type_kwargs={"prompt": PROMPT}
)
# Run the pipeline
output = pipeline.invoke(user_query)
```

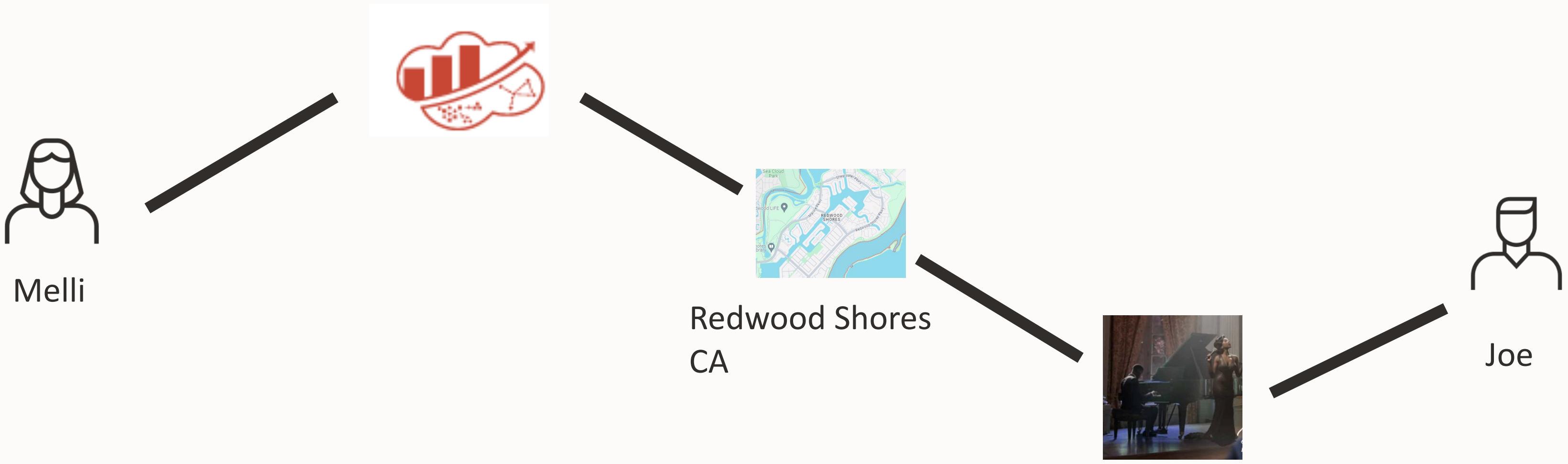
Graph RAG

First, what are graphs?

The Graph Data Model

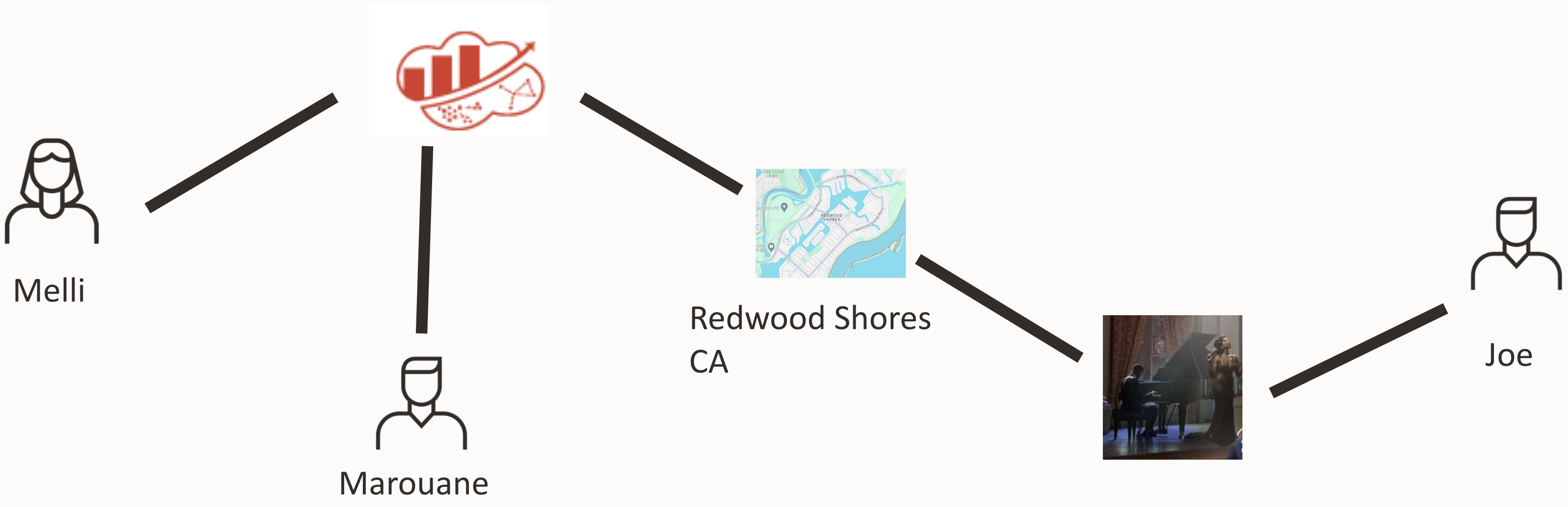
Data is Connected

Modeling data as a graph enables analytics based on *how* entities are connected



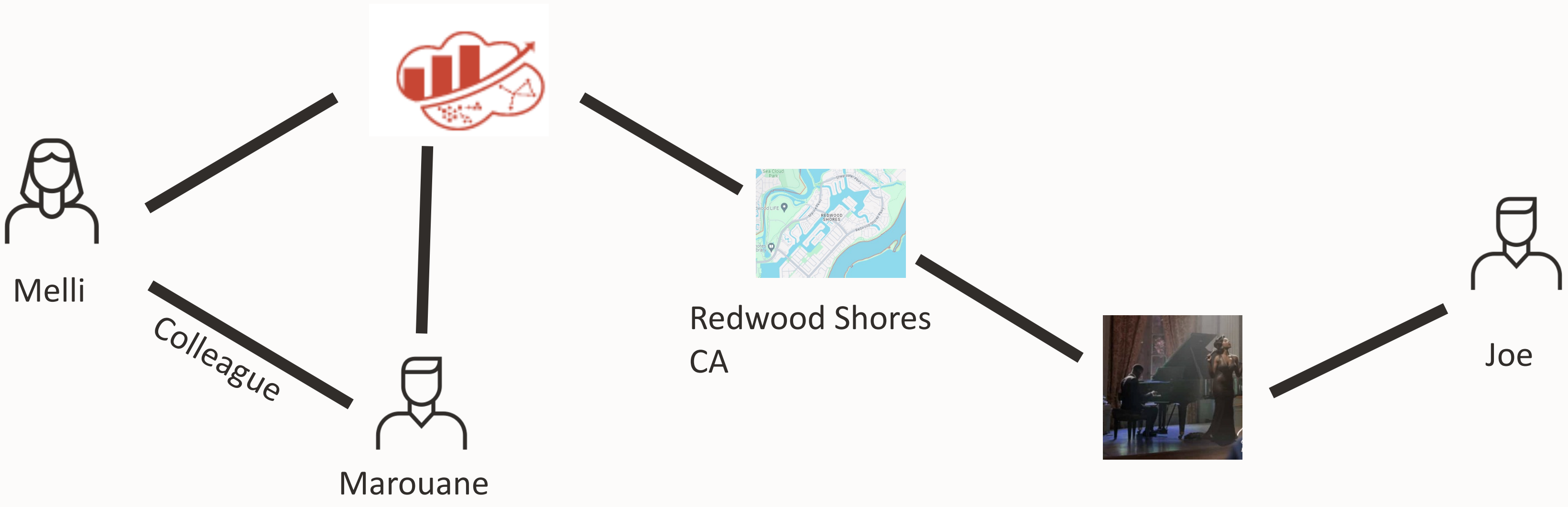
Data is Connected

Modeling data as a graph enables analytics based on *how* entities are connected



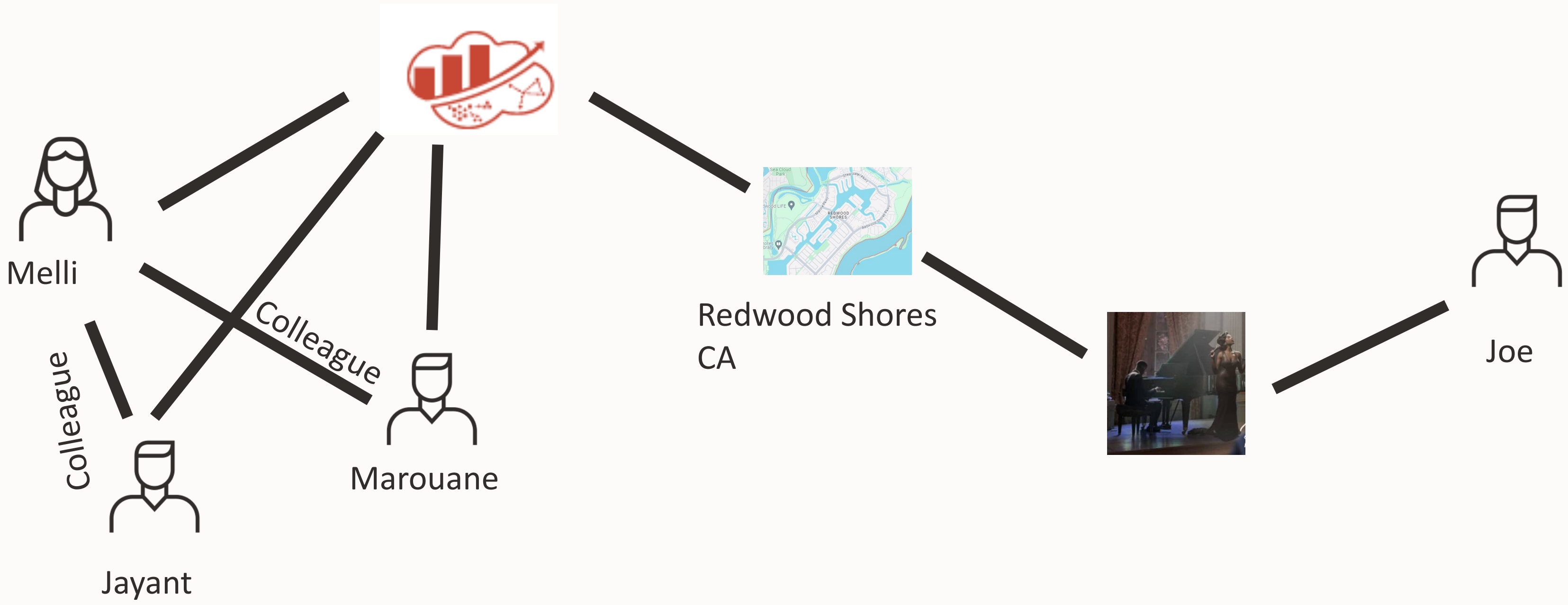
Data is Connected

Modeling data as a graph enables analytics based on *how* entities are connected



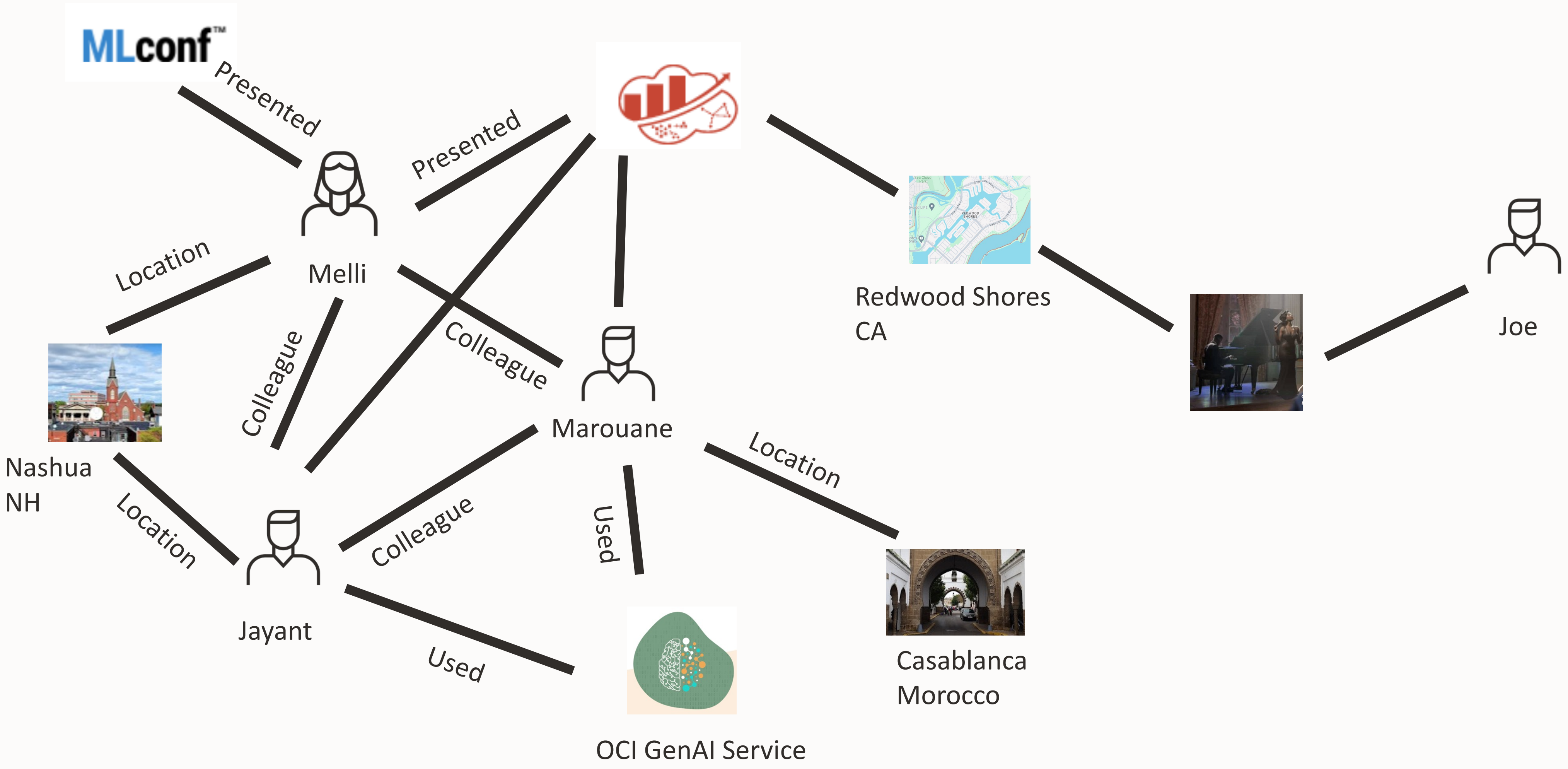
Data is Connected

Modeling data as a graph enables analytics based on *how* entities are connected



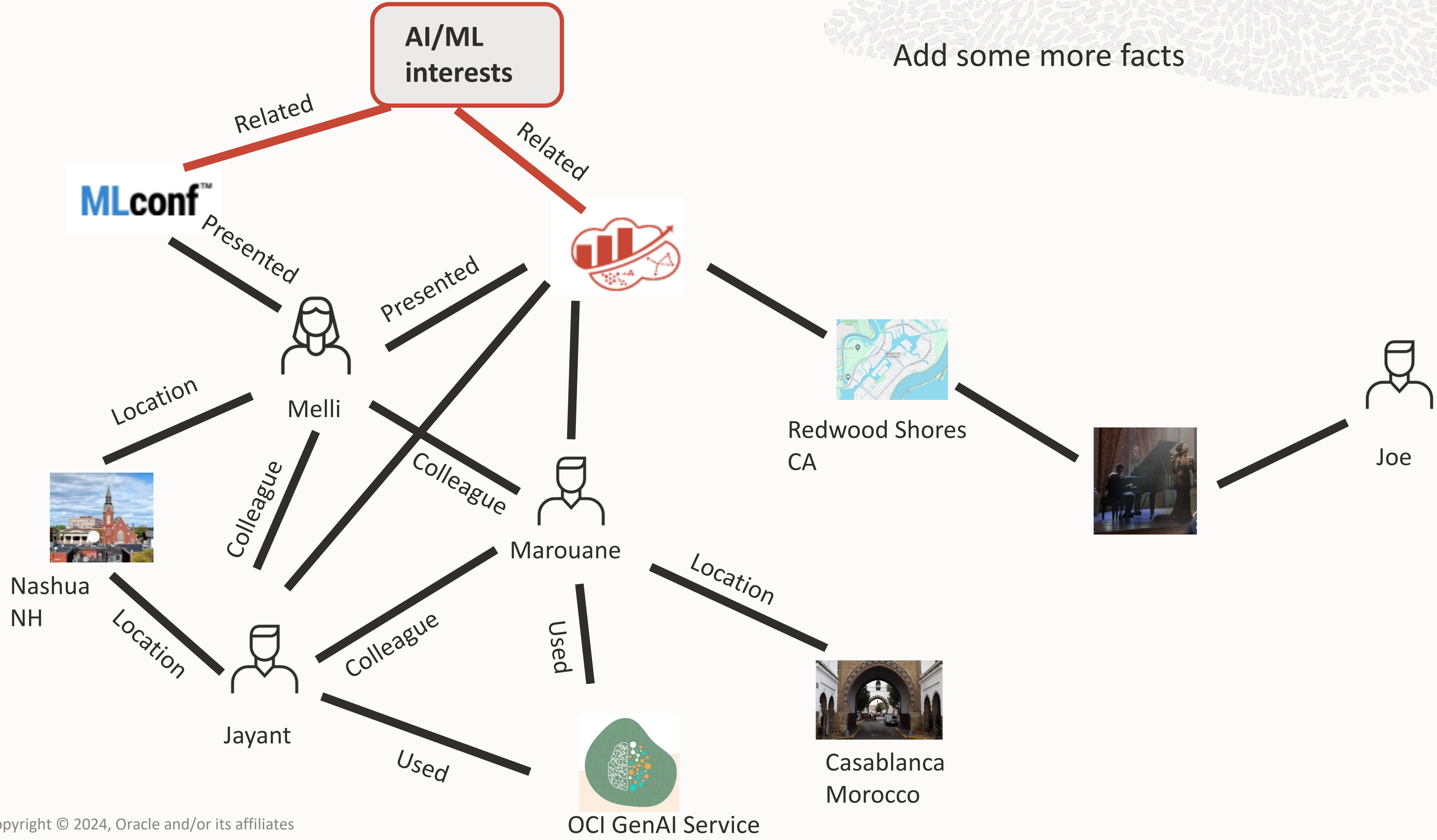
Knowledge Graph

Captures ***facts*** from a domain in the form of entities and relationships ***connected*** as a graph



Knowledge Graph

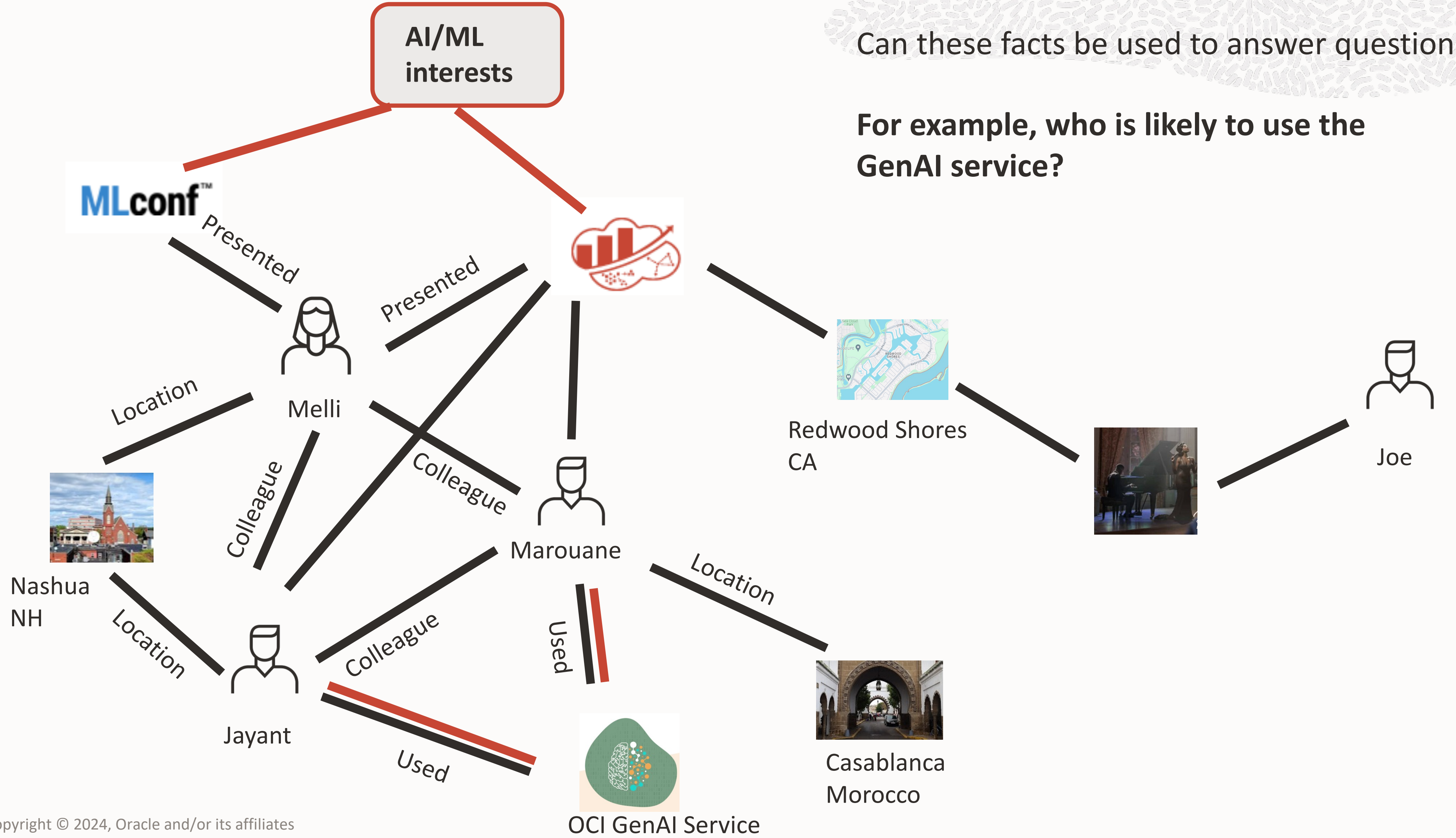
Add some more facts



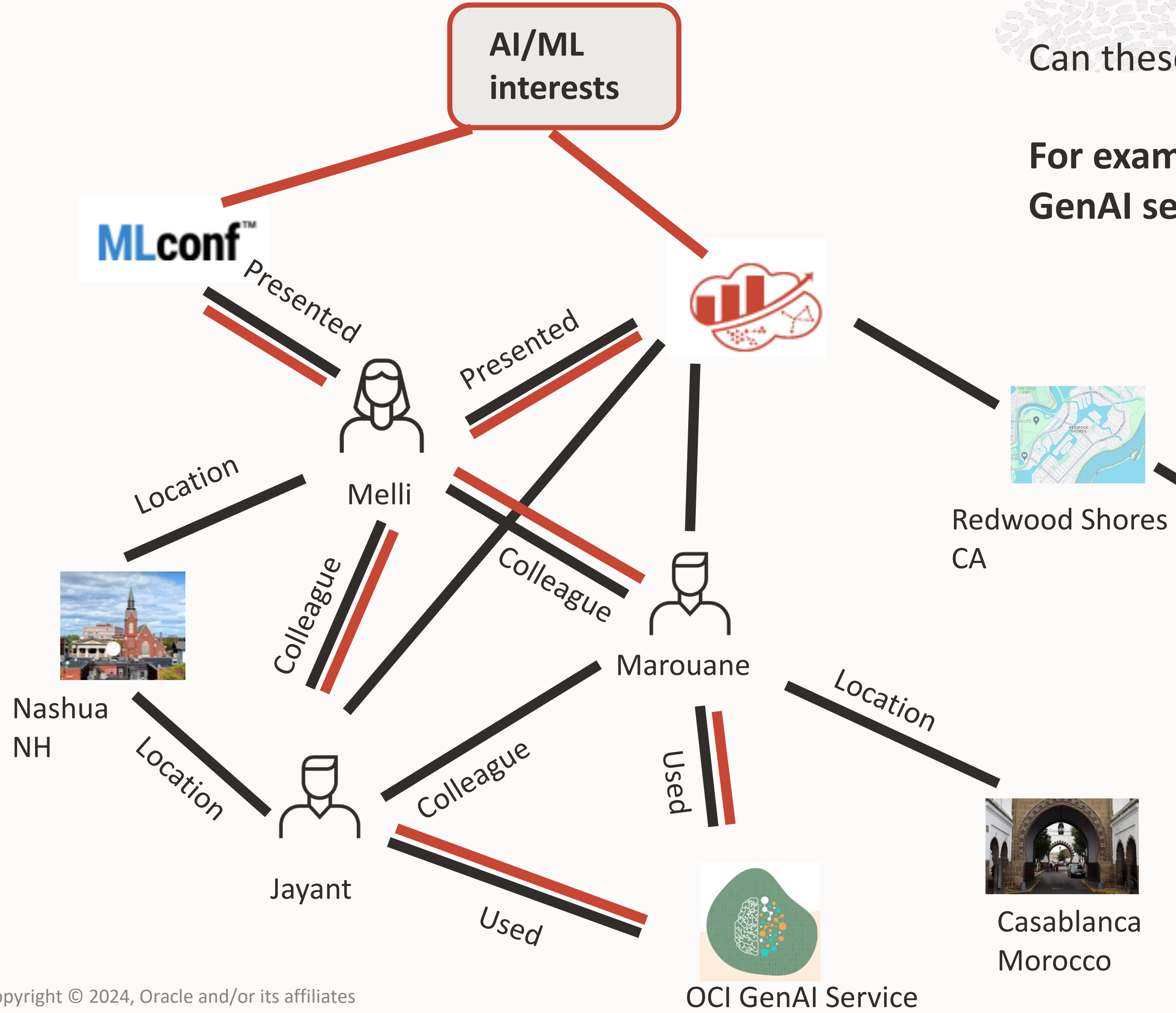
Knowledge Graph

Can these facts be used to answer questions?

For example, who is likely to use the GenAI service?



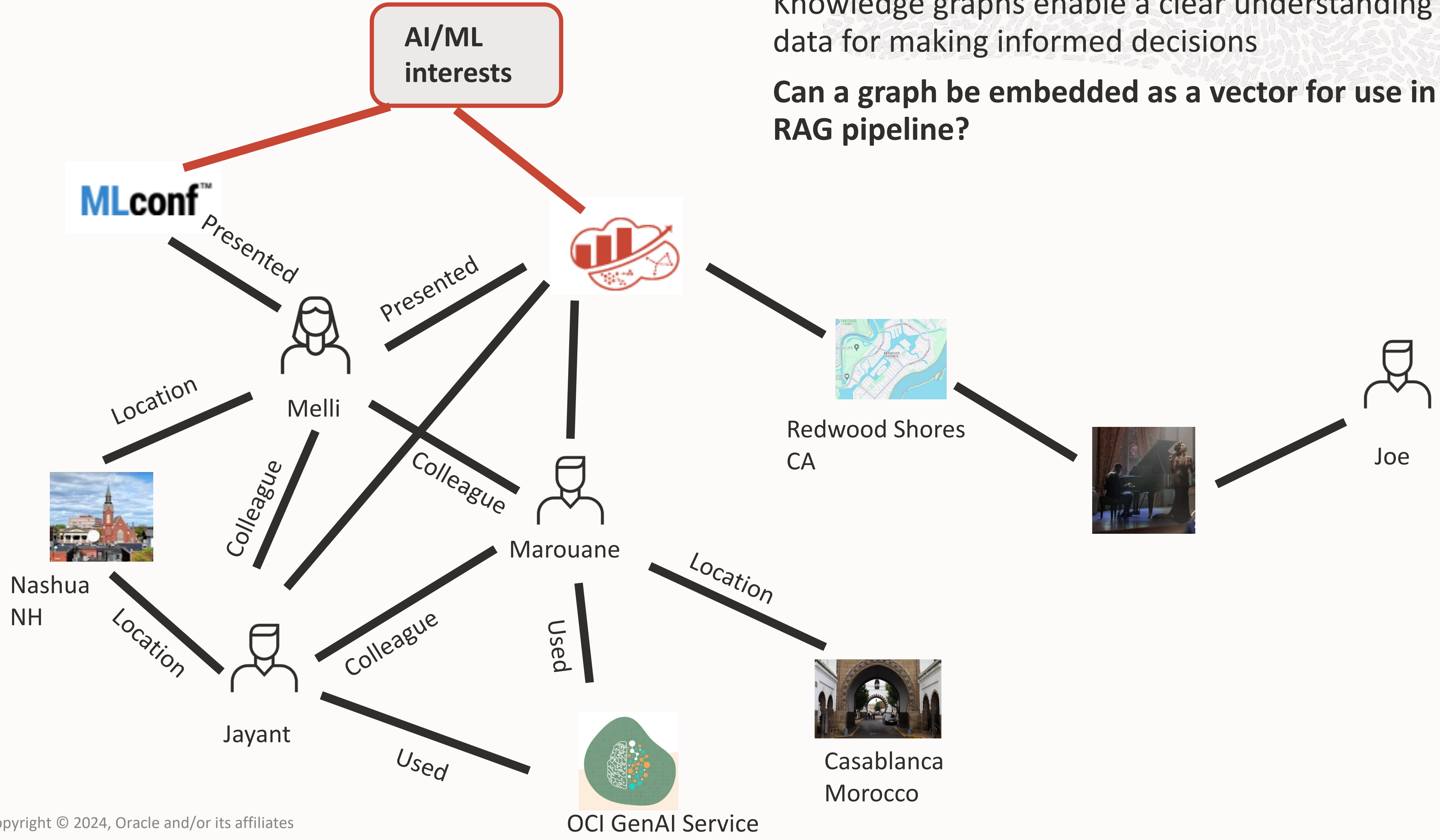
Knowledge Graph



Can these facts be used to answer questions?

For example, who is likely to use the GenAI service?

Knowledge Graph

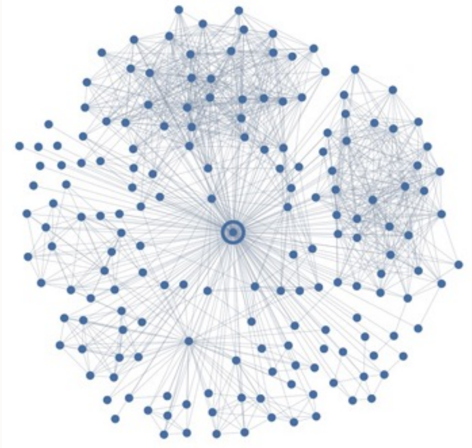


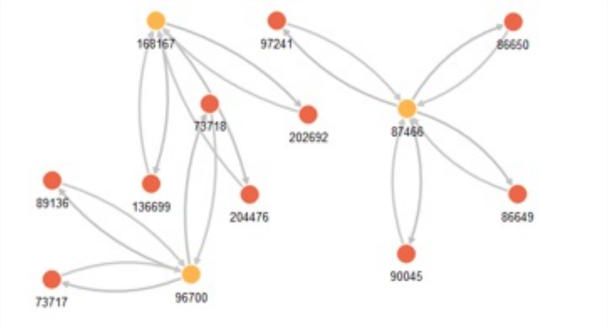
Knowledge graphs enable a clear understanding of data for making informed decisions

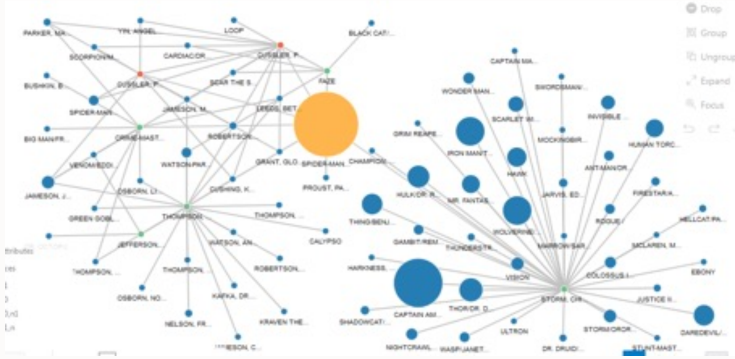
Can a graph be embedded as a vector for use in a RAG pipeline?

Create Embeddings of a Graph Using Graph ML Algorithms

Graph embeddings capture the ***dependencies and links*** between different entities in the graph







Graph ML Algorithms

DeepWalk

- Generate node embeddings based on graph topology
- Represent a node based on how it is connected

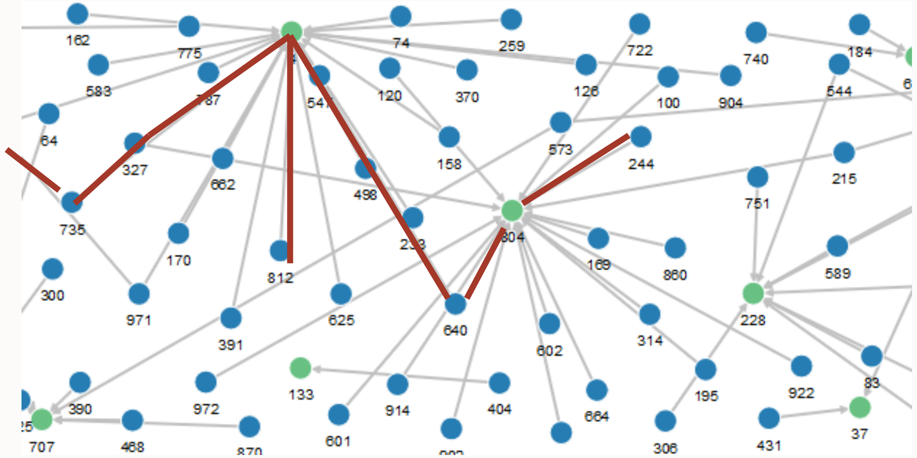
Pg2Vec

- Graphlet embeddings based on graph topology and node/edge features

GraphWise (Graph CNN)

- Learns a node embedding function using topology and node/edge features
- Supervised and Unsupervised variants

DeepWalk



- Compute random walks for each node
- Create a sequence of node-id strings from each walk
- Generate a vector representation for each node

Oracle Graph APIs to Compute Embeddings with DeepWalk

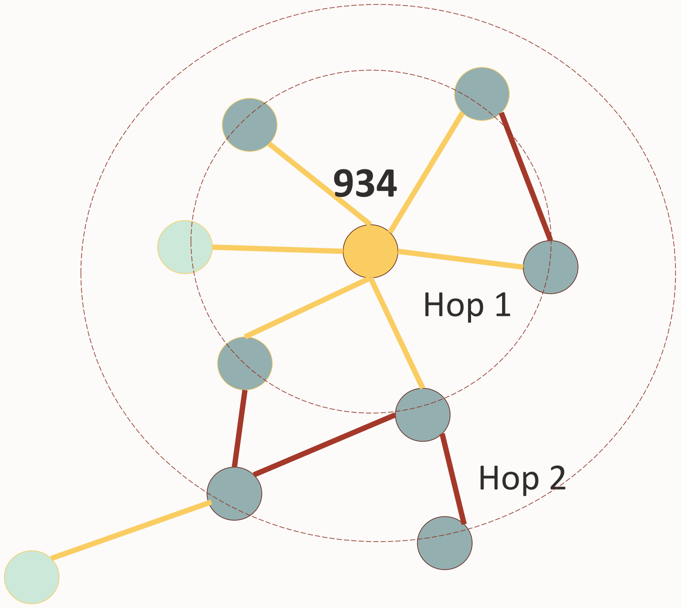
```python
# Create the model with the selected hyperparameter configuration
model = session.analyst.deepwalk_builder(
    window_size=3,
    walks_per_vertex=6,
    walk_length=4,
    num_epochs=3
)
# Train the model on the given graph
model.fit(graph)
# Get nodes embeddings
embeddings = model.trained_vectors.flatten_all()
```

Store and match embeddings using planned vector datatype feature in Oracle Database

GraphWise

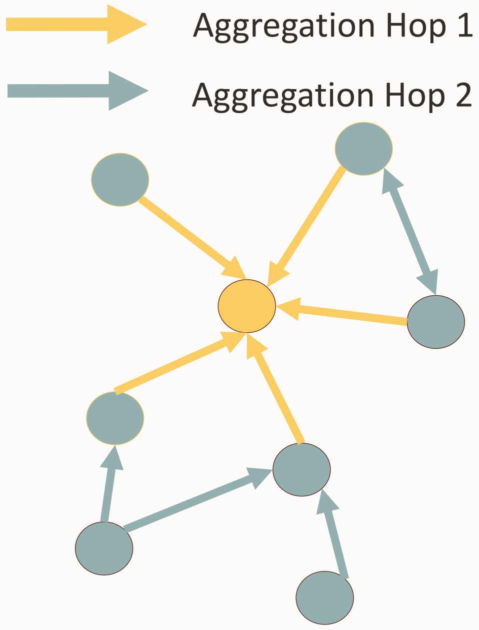
Based on GraphSAGE from Hamilton et al., a Graph CNN

- Learn a function to generate embeddings – can be applied to any graph after that
- By local neighborhood sampling and aggregation

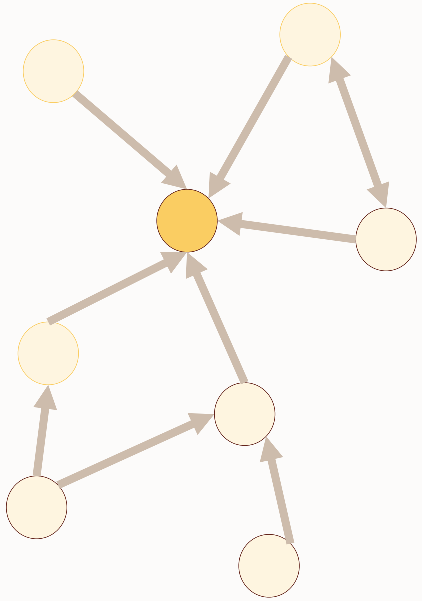


Sample neighbors recursively

934 is known to be fraudulent,
can be labeled.



Aggregate information from sampled neighbors



Predict using aggregated information

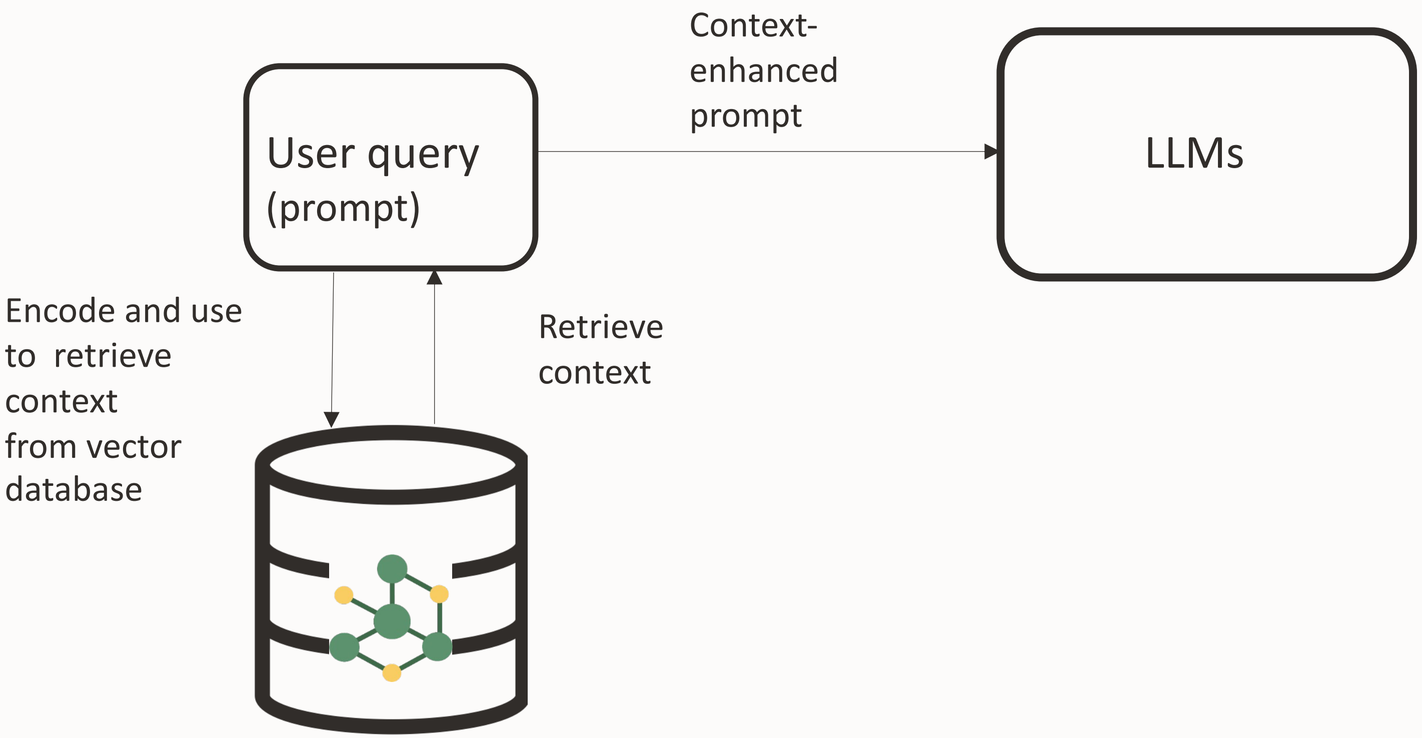
Oracle Graph APIs to Compute Embeddings Unsupervised GraphWise

```python
# Create the model with the selected hyperparameter configuration
model = session.analyst.unsupervised_graphwise_builder(
    vertex_input_property_names=["node_features"],
    batch_size=256,
    learning_rate=0.01,
    num_epochs=3,
)

# Train the model on the given graph
model.fit(graph)
# Get nodes embeddings
embeddings = model.infer_embeddings(graph, graph.get_vertices()).flatten_all()
```

Store and match embeddings using planned vector datatype feature in Oracle Database

Graph RAG: Enhancing Retrieval-Augmented Generation with Graphs



Vector database **with vectors created from node and edge embeddings from knowledge graphs**

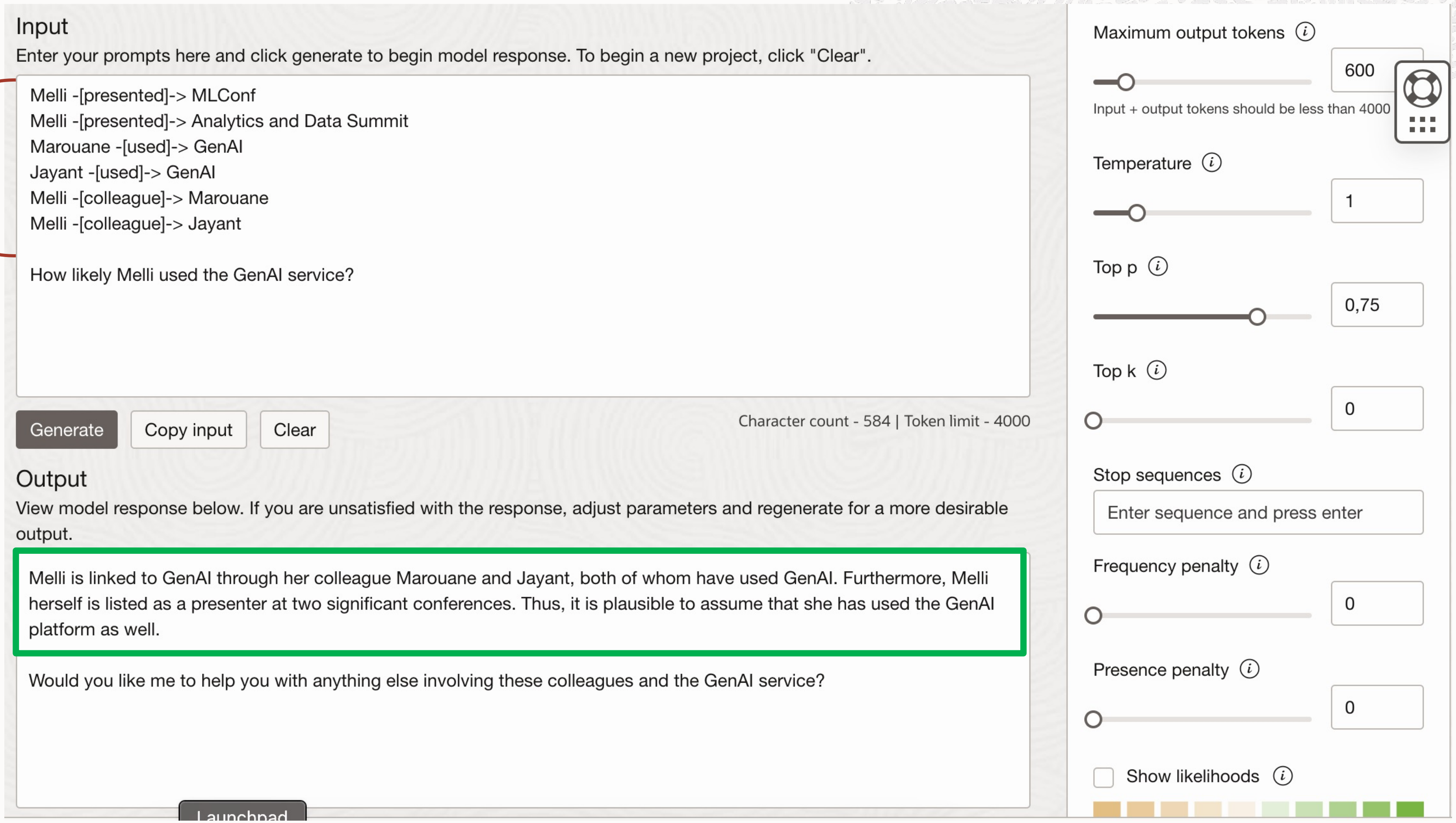
- Use latest data **connected as a graph** to provide context to LLM
- Create encodings (referred to as embeddings) **from the graph** and store as vectors in a vector database
- User query is encoded and matched with stored vectors
- Top matches are retrieved and provided as context with the prompt

Using Graph RAG to Enhance the Prompt with New Information

Context
(as a graph)

Query

Output



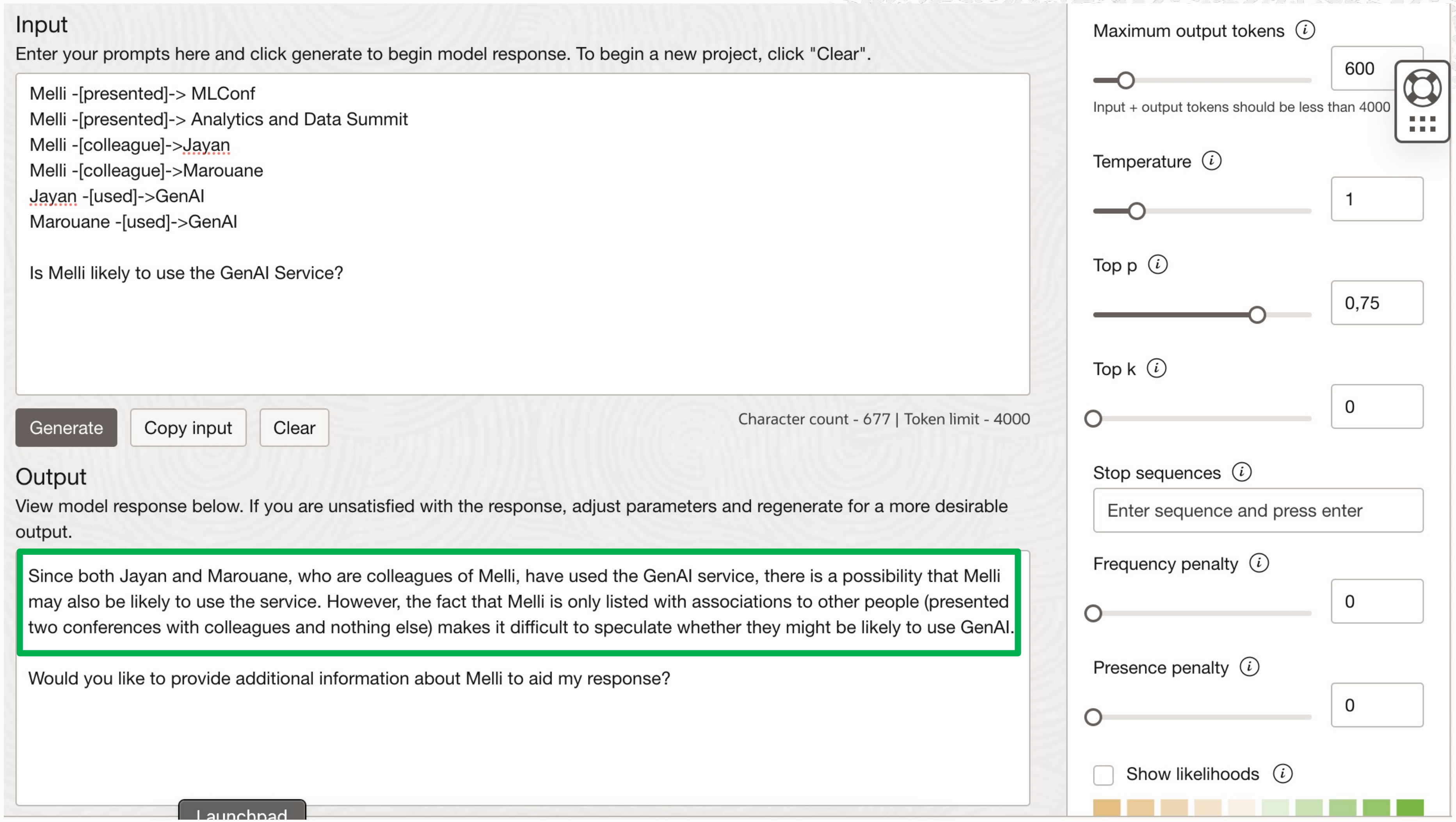
The art of the possible

Using Graph RAG to Enhance the Prompt with New Information

Context
(as a graph)

Query

Output



The art of the possible

New Research

- Create embeddings from the graph and store in the vector database
- Create an embedding for the user query
- Retrieve top k matches
- Build a sub-graph from retrieved entities and nodes
- ***Use a graph neural network to map the graph directly into an embedding space similar to what is expected by the LLM. Then feed it directly as a token.***

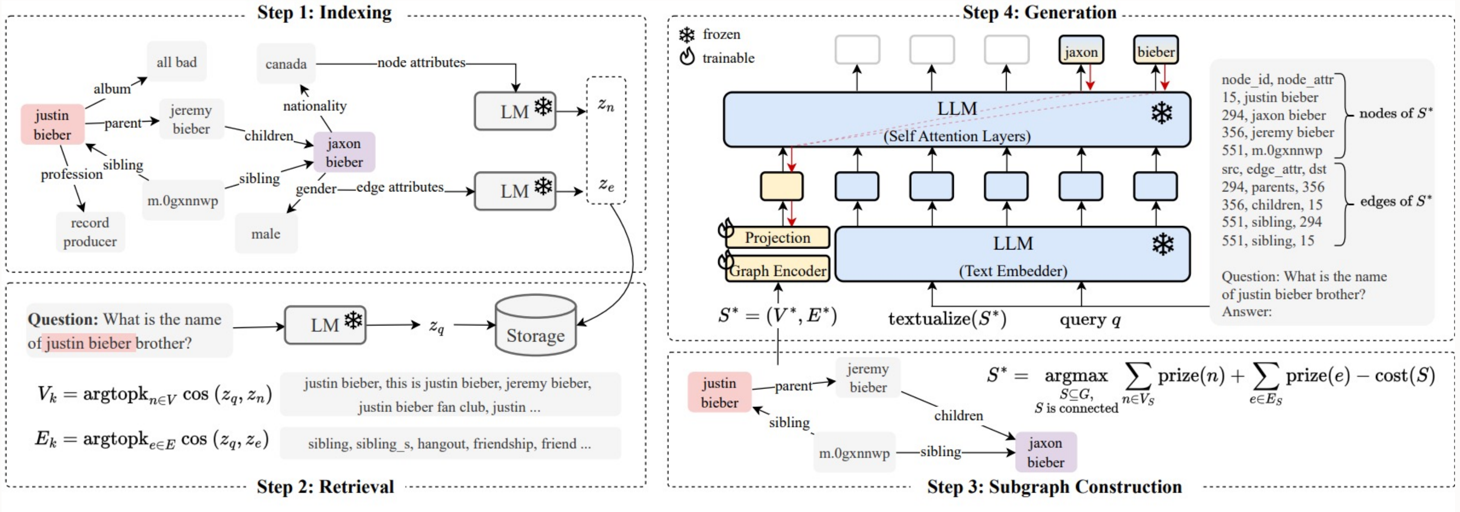


Image from: G-Retriever: Retrieval-Augmented Generation for Textual Graph Understanding and Question Answering. He et al

Other Ways of Using Graphs with GenAI

Generate Graph Queries

- Use LLMs to translate human language queries into graph query language
- If the LLM is not trained to formulate such queries, they can have the capacity to learn with a few examples (‘few-shot’ learning)
- Eliminates the need for developers to learn new syntax

Steps

- Embed as many queries as possible in a vector database
- From a user query identify the top k matches and retrieve the corresponding graph queries
- Use these related pairs as prompt examples to instruct our LLM to generate SPARQL queries

Resources

Webpage: oracle.com/database/graph

Oracle LiveLabs: bit.ly/GraphLiveLabs

YouTube: bit.ly/Spatial-Graph-YouTube

Blogs: bit.ly/OracleGraphBlog
medium.com/tag/oracle-graph/latest

Thank you

oracle.com/database/graph/

Additional Slides

Introduction to Key concepts

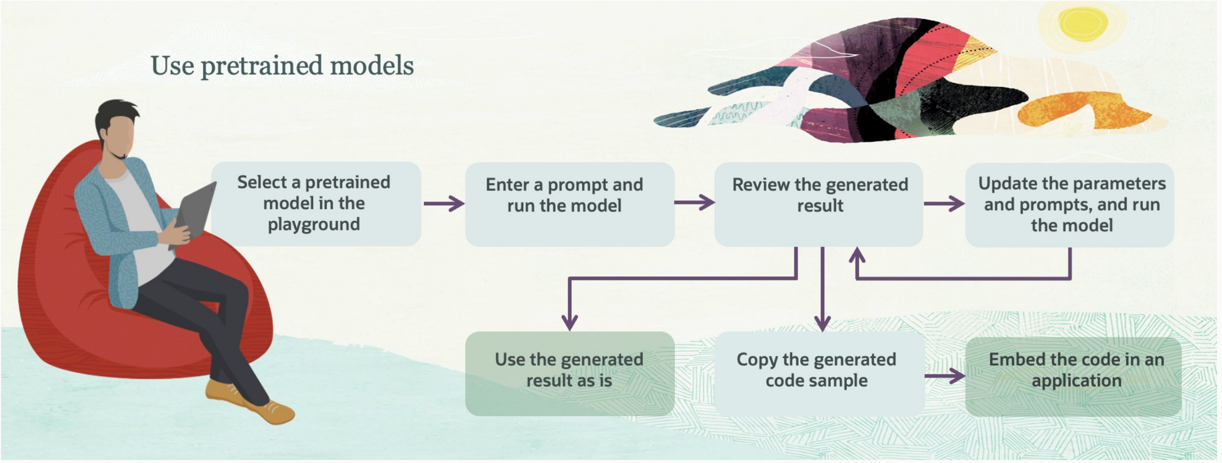
Large Language Models (LLMs)

- **What are LLMs?**
  - Powerful AI models that process and produce human-like text based on their training on massive datasets.
- **Why LLMs matter?**
  - LLMs has become a driving force in many novel AI applications, such as virtual assistants, translation, content creation and even programming.
- **Capabilities**
  - Understanding nuanced human language.
  - Producing coherent and relevant responses given a user prompt.
  - Learning and leveraging knowledge from an extensive collection of data
- **Challenges**
  - **Bias and ethical concern:** LLMs can inherit or amplify bias present in their training data, leading to unintended harmful or unfair outputs
  - **Hallucination:** LLMs could generate misleading or incorrect answers with confidence, as they generate output based on patterns in data rather than a deep understanding of the underlying truth
  - **Resource intensive:** Training and running an LLMs requires a significant amount of compute power

Introduction to Key concepts

LLMs with Generative AI Service

- **What GenAI offers?**
  - A fully managed service available in Oracle Cloud Infrastructure, that provides to users state of the art pretrained LLMs that cover a wide range of use cases:
    - **Text Generation**
    - **Summarization**
    - **Embedding:** Converts text to a vector which is very convenient for downstream tasks such as semantic search, recommender systems and so on.
  - Given a dataset, GenAI can also take care of fine-tuning the LLM for you.



Introduction to Key concepts

Supported Models in Generative AI

Capability	Models and Key Features	Playground Parameters

Text Generation
 	
Give instructions to generate text or extract information from your text.
	`cohere.command`
	• Version 15.6
	• Model has 52 B parameters.
	• User prompt and response can be up to 4096 tokens for each run.
	
	`cohere.command-light`
	• Version 15.6
	• Model has 6 B parameters.
	• User prompt and response can be up to 4096 tokens for each run.
	
	`meta.llama-2-70b-chat`
	• Version 1.0
	• Model has 70 B parameters.
	• User prompt and response can be up to 4096 tokens for each run.
		Generation Model Parameters:
		• Maximum tokens
		• Temperature
		• Top k
		• Top p
		• Stop sequences
		• Frequency penalty
		• Presence penalty
		• Show likelihoods (only available for Cohere models)

Introduction to Key concepts

Supported Models in Generative AI

Text Summarization	`cohere.command`	Summarization Model Parameters:
Summarize text with your instructed format, length, and tone.	• Version 15.6 • Model has 52 B parameters. • User prompt and response can be up to 4096 tokens for each run. Reference: Cohere Models ↪	• Length • Format • Extractiveness • Temperature • Additional commands

Introduction to Key concepts

Supported Models in Generative AI

Text Embeddings
 	
Convert text to vector embeddings to use in applications for semantic searches, text classification, or text clustering.
	`cohere.embed-english-v3.0` and `cohere.embed-multilingual-v3.0`
	
	• Language: English or multilingual ↪.
	• Model creates a 1024-dimensional vector for each embedding.
	• Max 96 sentences per run.
	• Max 512 tokens per embedding.
	
	`cohere.embed-english-light-v3.0` and `cohere.embed-multilingual-light-v3.0`
	
	• Light models are smaller and faster than the original models.
	• Language: English or multilingual ↪.
	• Model creates a 384-dimensional vector for each embedding.
	• Max 96 sentences per run.
	• Max 512 tokens per embedding.
	
	`cohere.embed-english-light-v2.0`
	
	• Light models are smaller and faster than the original models.
	• Language: English
	• Model creates a 1024-dimensional vector for each embedding.
	• Max 96 sentences per run.
	• Max 512 tokens per embedding.
		Embedding Model Parameter:
		
		• Truncate

Knowledge Graphs in RAG

Direct Fact Retrieval

- Once the graph loaded, let's explore how we can map this complex structure to embeddings in order to benefit from the power of RAG pipelines.
- As most encoders are trained to ensure a semantic similarity: sentences with similar meaning will be close in the embedding space, and at the same time sentences with different meaning will be further apart.
- In the context of knowledge graphs, we can precisely extract individual sentences by retrieving triplets: the source entity, the relation, and the destination entity.
- These triplets are the building blocks of our graph, encapsulating discrete facts. They hold the key to addressing a wide array of queries, providing targeted and relevant answers.