

# Vector Search Embedding in Databricks

## PoC Workflow Overview

---

- **Goal:** Identify usage and benefit of storing embeddings on Databricks to serve an LLM model.
- **Key Steps:**
  - Create Delta table with source data.
  - Create Vector Search Endpoint (UI or SDK).
  - Create the Index (target columns, embedding model).
  - Query the Index (semantic search).
  - Serve the Index via RAG to a Language Model.

# Databricks Setup: Vector Search & Index

---

## Vector Search Prerequisites

- **Unity Catalog** enabled workspace.
- **Serverless compute** available.

## Index Creation

- Can be done via **UI** or **Python SDK**.
- Requires: Primary ID, text columns, and an **Embedding Model**.
- *Note: Change Data Feed must be enabled on the source Delta table.*

[Databricks Documentation](#)

[Vector Search Modules](#)

# How Vector Index Works: Core Concepts

---

## 1. Embedding

- Text strings are converted into an array of floats (vector).
- Models (e.g., GTE, BERT) process tokens to capture context.

## 2. Vector Similarity

- Distance computation between two embedding vectors.
- **Smaller distance = Similar semantic items** (used for effective querying).

# Vector Similarity Visualized

---

## Semantic Clustering in 2D Space

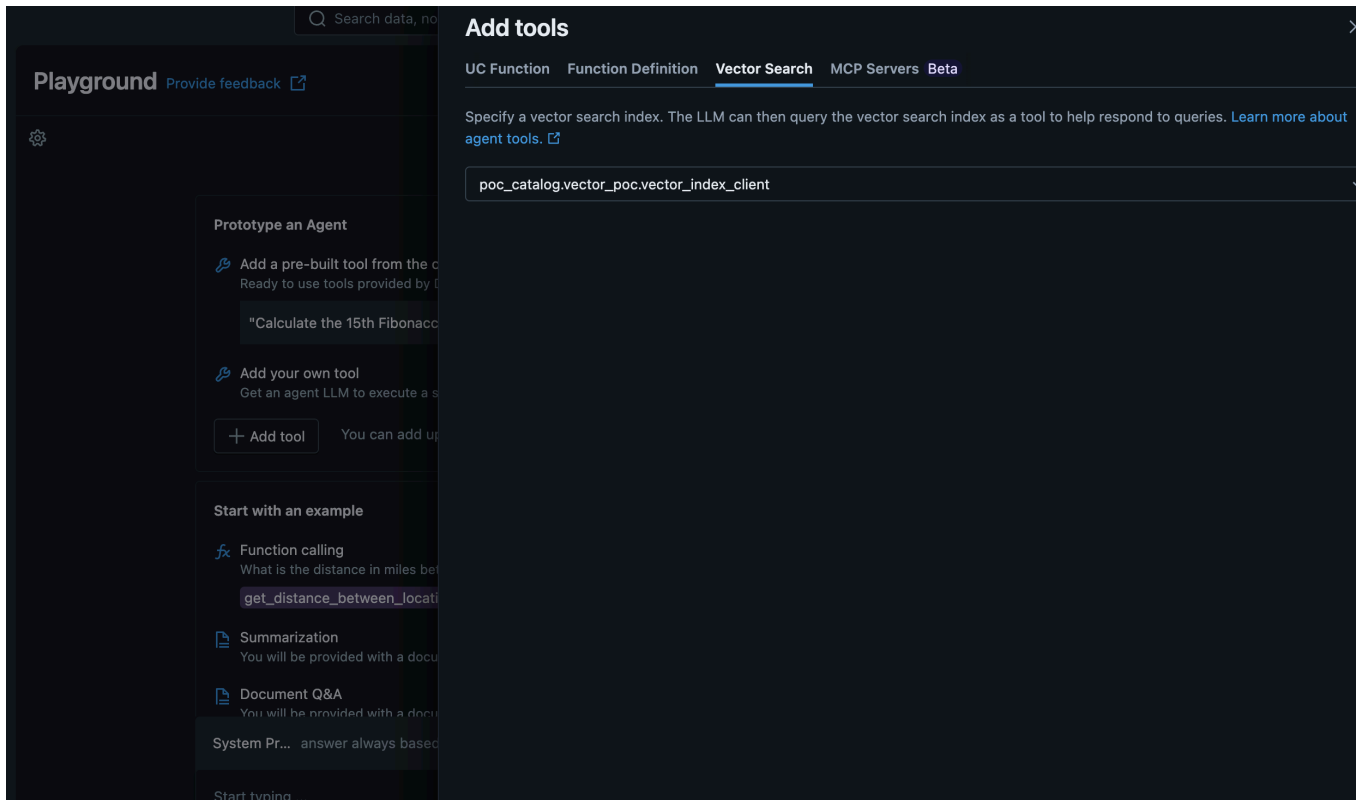
- A vector database holds the text and its embedded vector.
- Semantically similar words are grouped together.

# Databricks Agentbricks & RAG

---

## Using the Index in AI Context

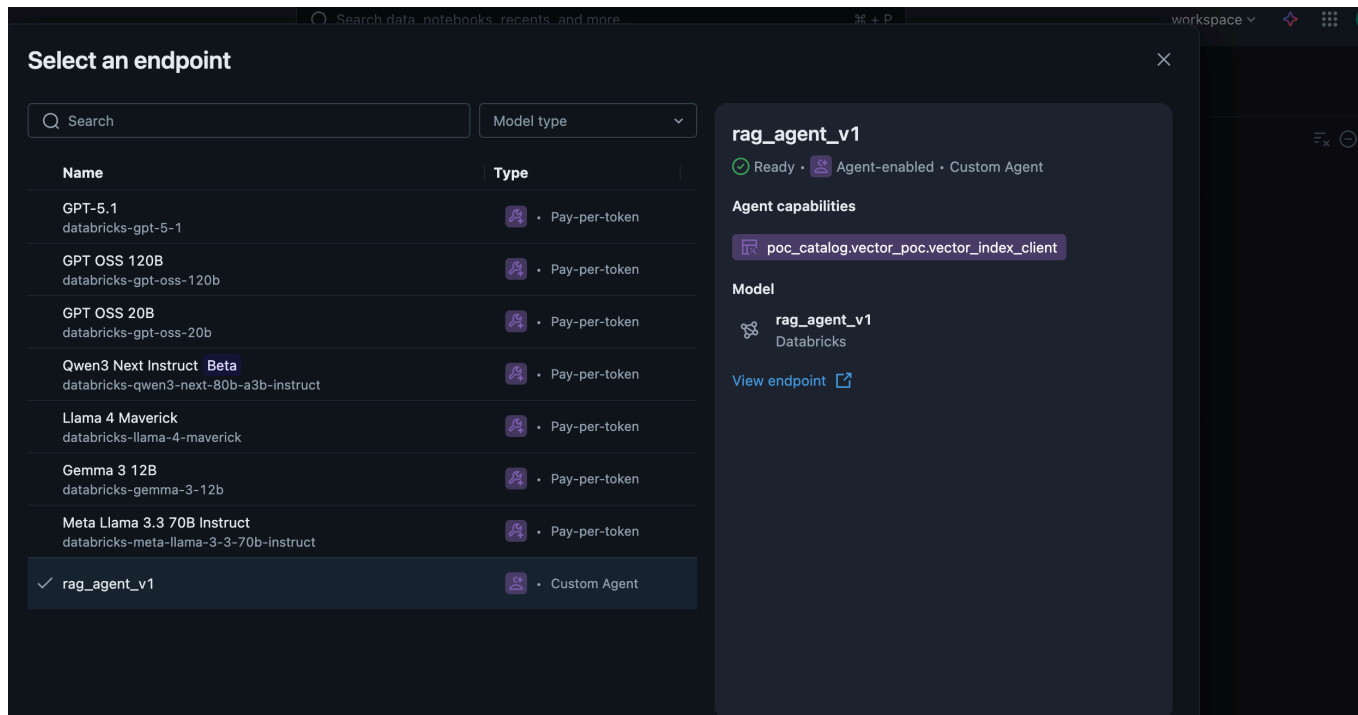
- Index can be used directly from Unity Catalog.
- Integrated as a **tool** for RAG in the Databricks Playground.



# Model Serving and MLflow

## Model Deployment

- Code generated in Playground ( `driver.ipynb` , `agent.py` ) can be used to **Register the Model**.
- Model is served via **Serving Endpoints** (UI or Python SDK).



# MLflow Functionality

---

## End-to-End ML Lifecycle Management

- **Experiment Tracking:** Log parameters, metrics, artifacts.
- **Model Registry:** Centralized store for versioning and management.
- **Model Deployment:** Deploy to batch, streaming, or real-time endpoints.
- Seamlessly integrated with Databricks workflows.

# Local Integration: RAG System

---

## Local LLM Setup

- Target Model: **Google Flan-T5-Base** (250M params).
- Use `huggingface_hub` to download and cache the model locally.

## Access Requirements

- **Databricks Token** (for accessing Vector Search API).
- Token and model cache path stored in `.env` file.

# Local Integration: Backend Query

---

## Running the Backend Server

The command to spin up the server is:

```
uvicorn src.backend.main:app --reload --host 0.0.0.0 --port 8000
```

## Testing (Untrained/No Context)

The cURL command is:

```
curl -X POST 'http://127.0.0.1:8000/query' \
-H 'Content-Type: application/json' \
-H 'X-API-Key: default_key' \
-d '{
    "query": "What is a delta lake",
    "top_k": 1,
    "query_context": false
}'
```

# Local Integration: RAG Success

---

## Testing (With Databricks Vector Context)

- Enables retrieval of context from the vector index.
- Confirms successful connection and RAG pipeline function.

The RAG-enabled cURL command is:

```
curl -X POST 'http://127.0.0.1:8000/query' \  
-H 'Content-Type: application/json' \  
-H 'X-API-Key: default_key' \  
-d '{  
  "query": "What is a delta lake",  
  "top_k": 1,  
  "query_context": true  
'
```