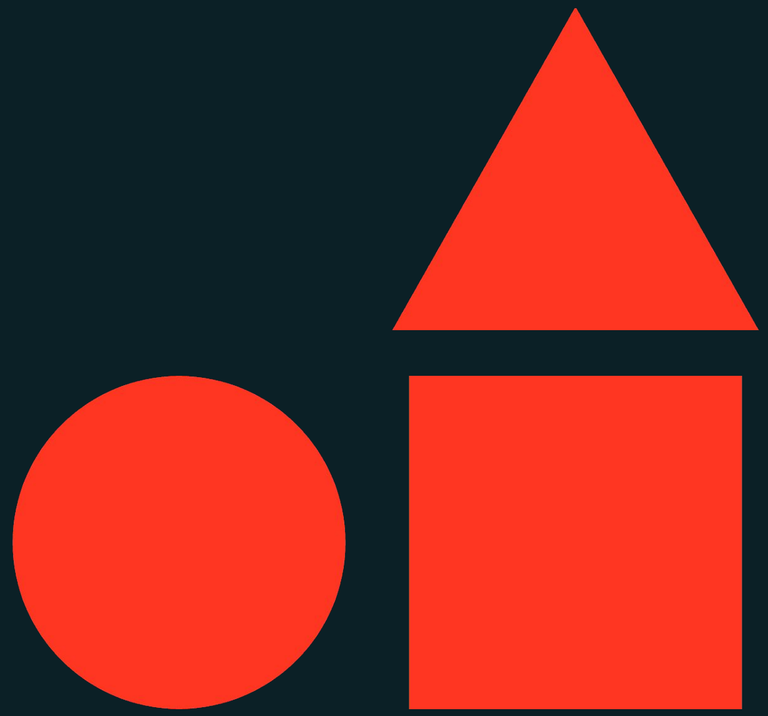
databricks

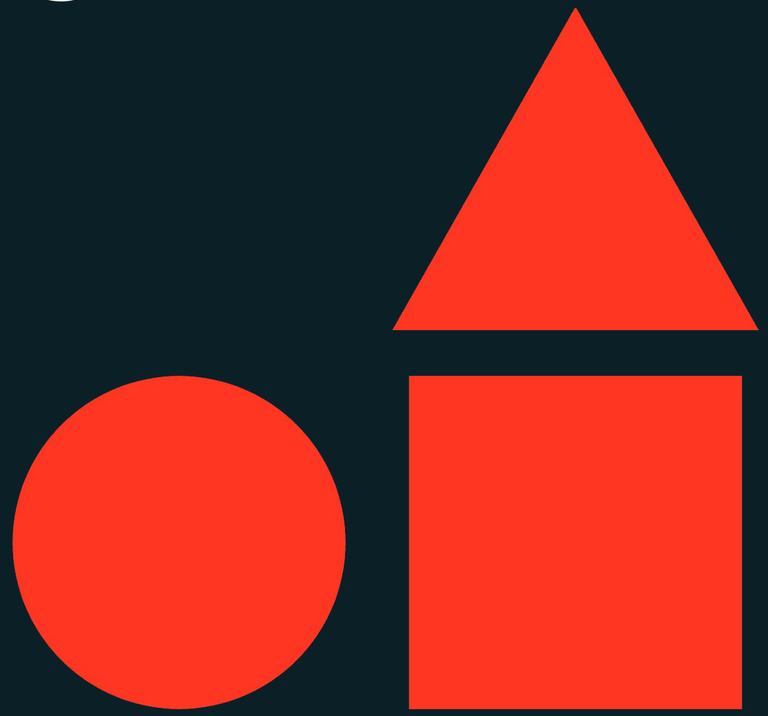
Generative AI Solution Development



Databricks Academy

databricks

From Prompt Engineering to RAG



Generative AI Solution Development

Learning Objectives

- Describe prompts and prompt engineering.
- Discuss various techniques and best practices of prompt engineering.
- Describe nuances: RAG, prompt engineering, fine-tuning, and pre-training.
- Identify use cases where RAG can be used to improve the quality, reliability, and accuracy of LLM completions.
- Describe the core components of the RAG architecture.
- Connect Databricks capabilities with the various components of RAG.

databricks

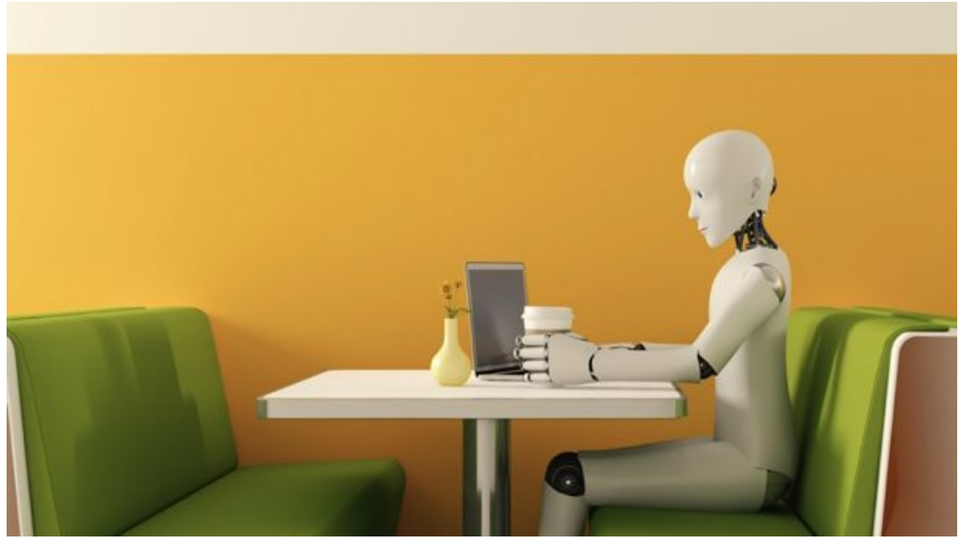
LECTURE

Prompt Engineering Primer

Prompt Basics

Definitions

- **Prompt**: An AI prompt is an **input or query** given to a large language model to elicit a specific response or output.
- **Prompt Engineering**: Prompt engineering is the practice of **designing and refining prompts** to optimize the responses generated by AI models



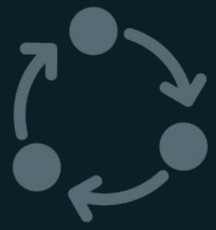
Prompt Basics

Prompt components

A good prompt usually consists of:

- **Instruction**: A clear directive that specifies what the model should do.
- **Context**: Background/additional information that provides the necessary details to understand the task.
- **Input / question**: The specific query or data that the model needs to process.
- **Output type / format**: The desired structure or style of the response generated by the model.

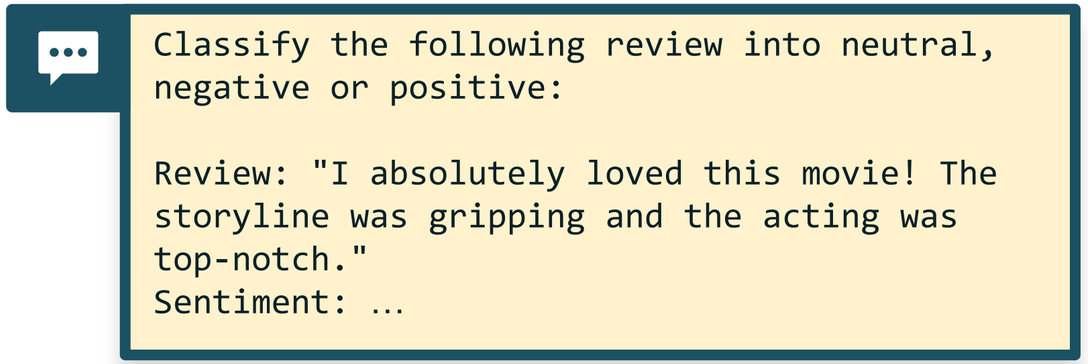
```
Summarize the following
article and write a list of
top 3 important points in
markdown format that answer
the following question;
“How is climate change
affecting polar bear habitats
and their ability to find
food?”

Article:
<The article discusses the
impact of climate change on
polar bear populations in the
Arctic.>
```

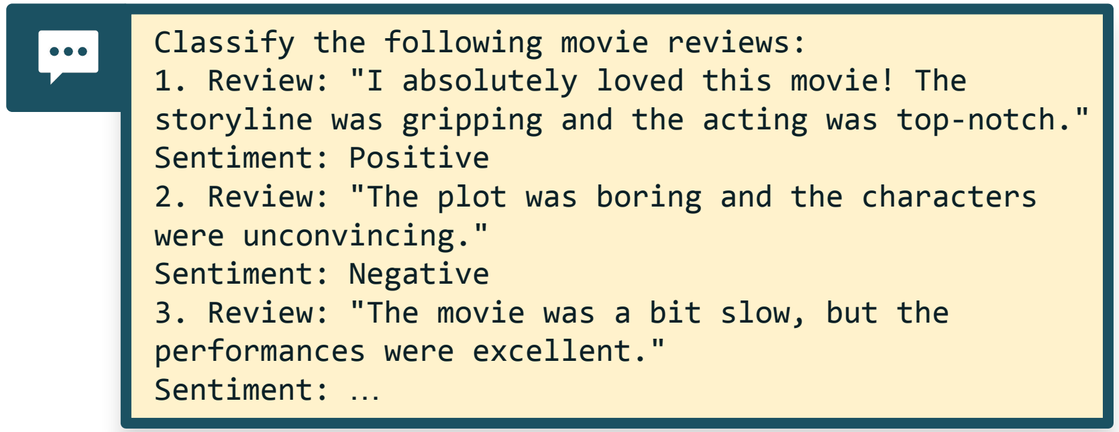


Prompt Engineering
Techniques

Zero-shot / Few-shot Prompting

Using or not-using an example





Zero-shot Prompting

- Prompt that generates text or perform a task **without providing any examples** or additional training specific to that task.

Few-shot Prompting

- Prompt provides with **a few input-output examples** to guide the model for generating the desired output.

Prompting Chaining

Break tasks into subtasks

- Multiple tasks are linked together, with the output of one prompt serving as the input for the next.
- This method allows for more complex tasks to be broken down into manageable steps.

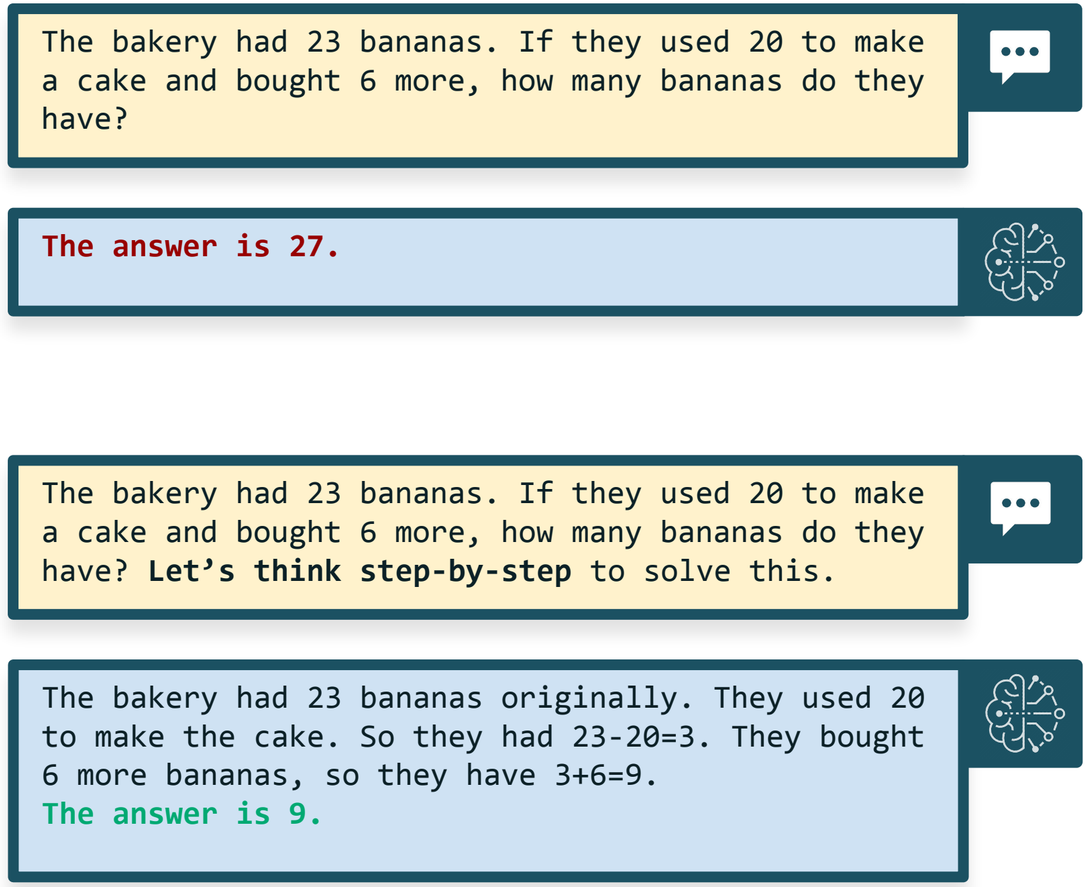
```
You are a helpful assistant. Your task is to
help answer a question given in a document. The
first step is to extract quotes relevant to the
question from the document, delimited by ####.
Please output the list of quotes using
<quotes></quotes>. Respond with "No relevant
quotes found!" if no relevant quotes were found.
####
Document: "The Amazon rainforest produces 20% of
the world's oxygen..."

Question: "What is the importance of the Amazon
rainforest?"
####
```

```
<quotes>
"The Amazon rainforest produces 20% of the
world's oxygen."
"The rainforest regulates the global climate and
has diverse wildlife."
</quotes>
```

Chain-of-Thought Prompting

"Let's think step-by-step"

- Chain-of-thought (CoT) prompting enhances the reasoning capabilities of LLMs by guiding them to **articulate their thought processes step-by-step**, similar to human reasoning.
- Research in this area has **mixed findings**;
  - Some research showed that simply asking the model to think step-by-step helps to solve reasoning questions.
  - Others showed that models don't follow CoT faithfully (Paper 1, Paper 2, Paper 3).



Prompt Engineering
Tips & Tricks

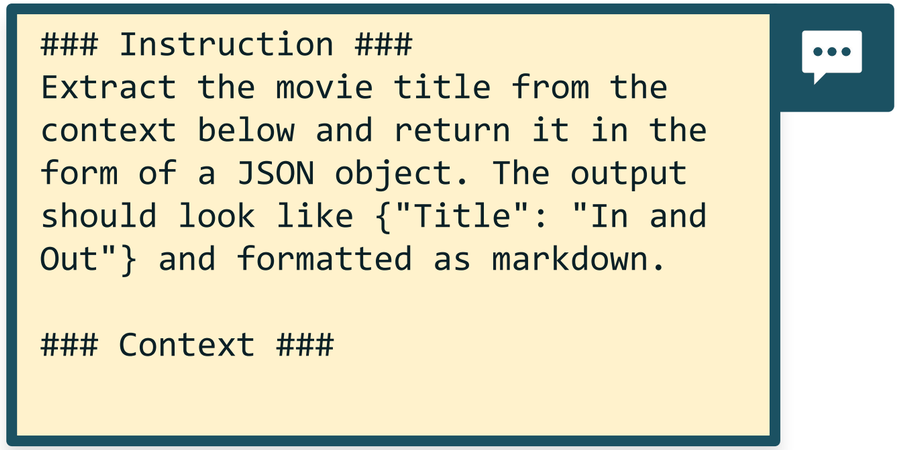
Tip: **Prompts are model-specific**

A prompt guides the model to complete task(s)

- Different models may require **different prompts**.
- Provide **examples and cues** to guide model's response generation.
- Different **use cases** may require different prompts.
- **Iterative development** is key.
  - Iterate by adjusting the **temperature parameter**—higher for more creative outputs, lower for focused responses.
- Be aware of **bias and hallucination**.

Tip: **Format prompts**

- Use delimiters to distinguish between instruction and context
  - Pound sign ###
  - Backticks ```
  - Braces / brackets {} / []
  - Dashes ---
- Ask the model to return structured output
  - HTML, json, table, markdown, etc.
- Provide a correct example
  - `"Return the movie name mentioned in the form of a Python dictionary. The output should look like {'Title': 'In and Out'}"`



Tip: **Guide the model for better responses**

- Ask the model not to make things up/*hallucinate*
  - `"Do not make things up if you do not know. Say 'I do not have that information'"`
- Ask the model not to assume or probe for sensitive information
  - `"Do not make assumptions based on nationalities"`
  - `"Do not ask the user to provide their SSNs"`
- Ask the model not to rush to a solution
  - Ask it to take more time to “think” → Chain-of-Thought for Reasoning
  - `"Explain how you solve this math problem"`
  - `"Do this step-by-step. Step 1: Summarize into 100 words. Step 2: Translate from English to French..."`

Benefits and Limitations

Benefits

- **Simple and efficient**: The time taken to generate the ideal results is significantly low.
- **Predictable results**: Consistently generate results that meet predefined standards for accuracy.
- **Tailored outputs**: Customization of AI responses to fit specific needs or styles.

Limitations

- The output depends on used model.
- Limited by pre-trained model's internal knowledge. **For external knowledge we need to use RAG**.

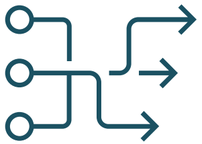
databricks

LECTURE

Introduction to RAG

How do Language Models Learn Knowledge?

This will be the focus of this course



Model Pre-Training

- Training an LLM from scratch
- Requires large datasets (billions to trillions of tokens)

Model Fine Tuning

- Adapting a pre-trained LLM to specific data sets or domains
- Requires thousands of domain-specific or instruction examples

Passing Contextual Information

- Combining an LLM with external knowledge retrieval
- Requires external knowledge base
- How do we use vectors to **search** and provide **relevant context** to LLMs?

LESS Complexity and Compute-intensiveness

Passing Context to LMs Helps Factual Recall

- **Passing context** as model inputs improves factual recall
  - Analogy: Take an exam with **open notes**

- LLMs are evolving to accept a larger/infinite input token window size

Downsides of "Long" context:

- Higher API costs (# input token)
- Longer completion/inference times
- Content/documents in the middle may be overlooked (Lost in the middle and needle in haystack test)

⇒ Ongoing research to address "context limitation"

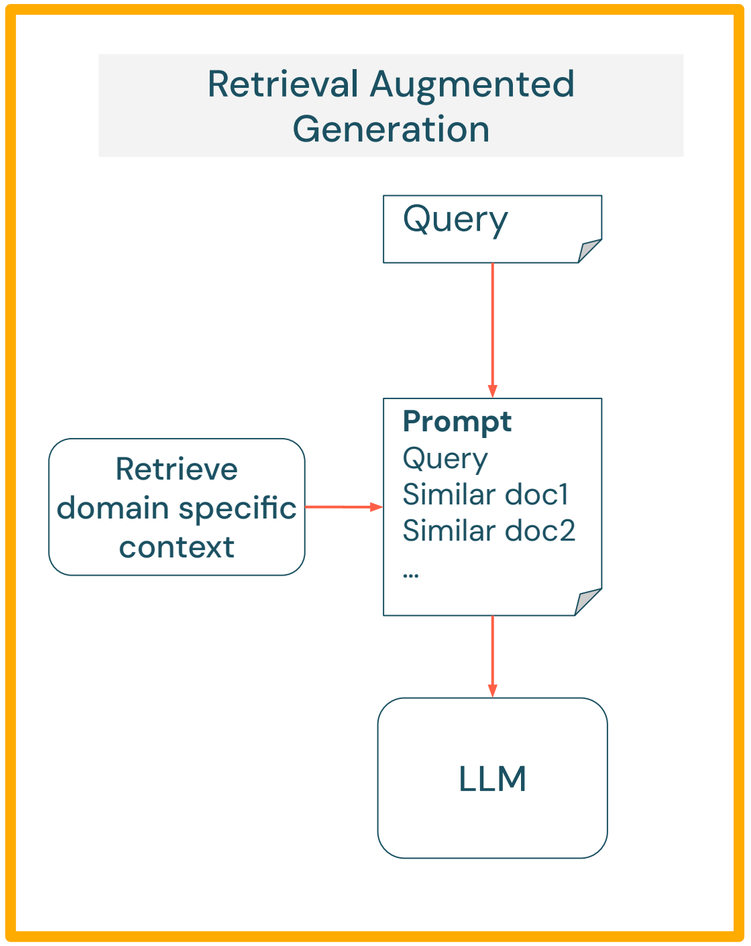
So what is RAG?

RAG is...

Retrieval **A**ugmented **G**eneration, or RAG;

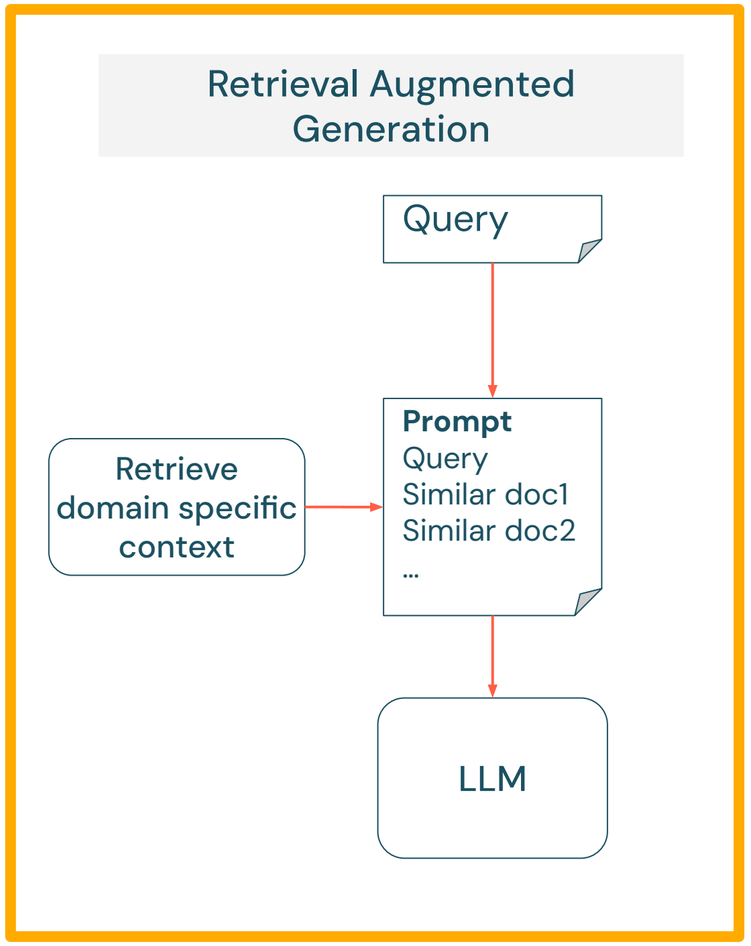
- Is a pattern (Lewis et al. 2020) that can improve the **efficacy** of large language model (LLM) applications by **leveraging custom data**.

- Is done by **retrieving** data/documents relevant to a question or task and providing them as context to **augment** the prompts to an LLM to improve **generation**.



RAG is...

- The main problem that is solved with RAG architecture is the **knowledge gap**. This approach enhances the accuracy and relevance of responses.



RAG Use Cases

Some practical use cases

Q&A Chatbots

- Incorporate LLMs with chatbots to automatically derive more accurate answers.
- Used to automate customer support and website lead follow-up to answer questions and resolve issues quickly.

Search Augmentation

- Incorporating LLMs with search engines that augment search results with LLM-generated answers.
- Makes it easier for users to find the information they need.

Content Creation and Summarization

- Facilitate the development of high-quality articles, reports, and summaries using additional context.
- Example; generation of news articles or summarization of lengthy reports.

Main Concepts of RAG Workflow

Components and concepts in search and RAG workflow

Index & Embed:

An embedding model used for creating **vector representation** of the documents and user queries.

Vector Store:

Specialized to store unstructured data indexed by vectors. Vectors can be stored with a **vector DB, library, or plugin**.

Retrieval:

Search stored vectors using similarity search to efficiently **retrieve relevant information**.

Filtering & Reranking:

The process of selecting or ranking retrieved documents before passing as context.

Main Concepts of RAG Workflow

Components and concepts in search and RAG workflow

Prompt Augmentation:

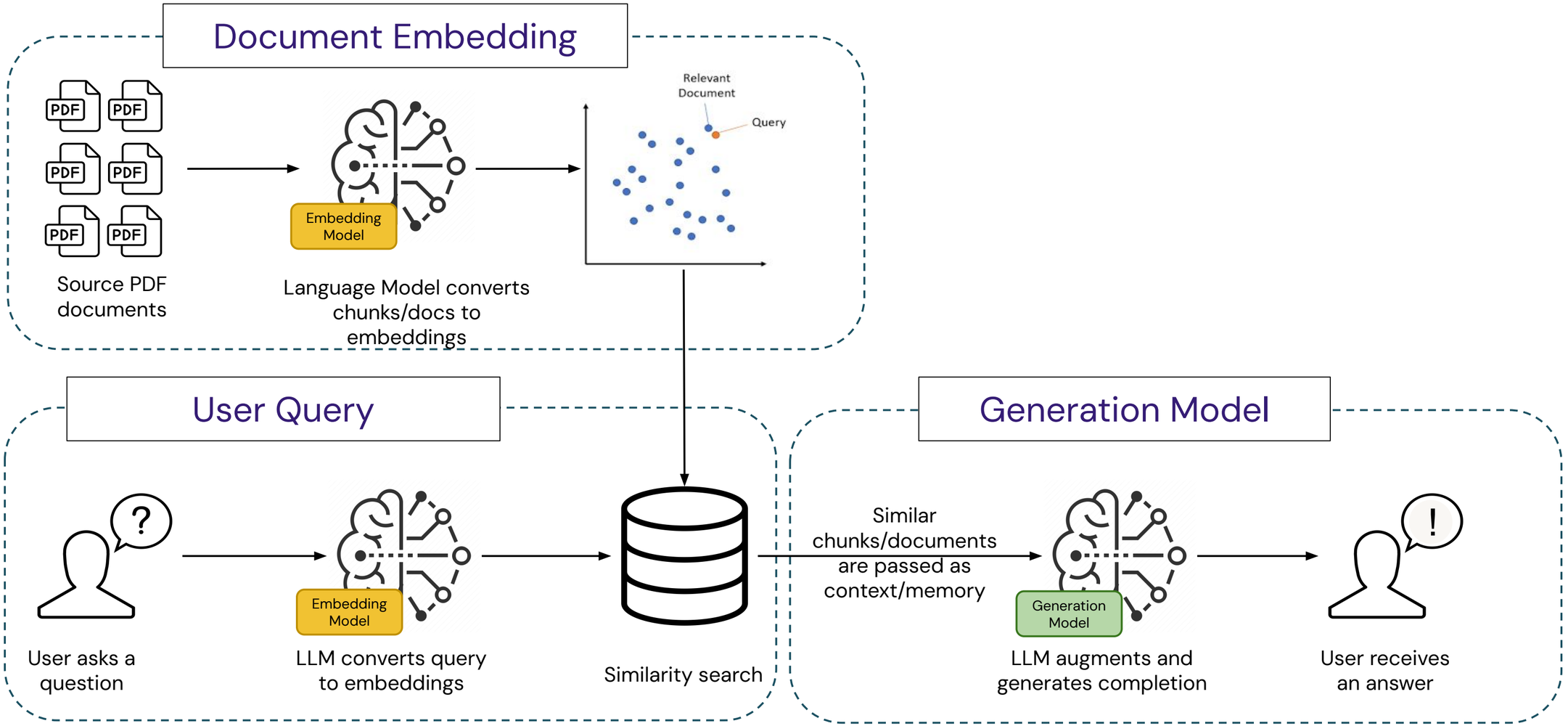
Prompt engineering workflow to **enhance context via injection of data retrieved** from Vector store

Generation:

A large language model used for generating a response for the user's request.

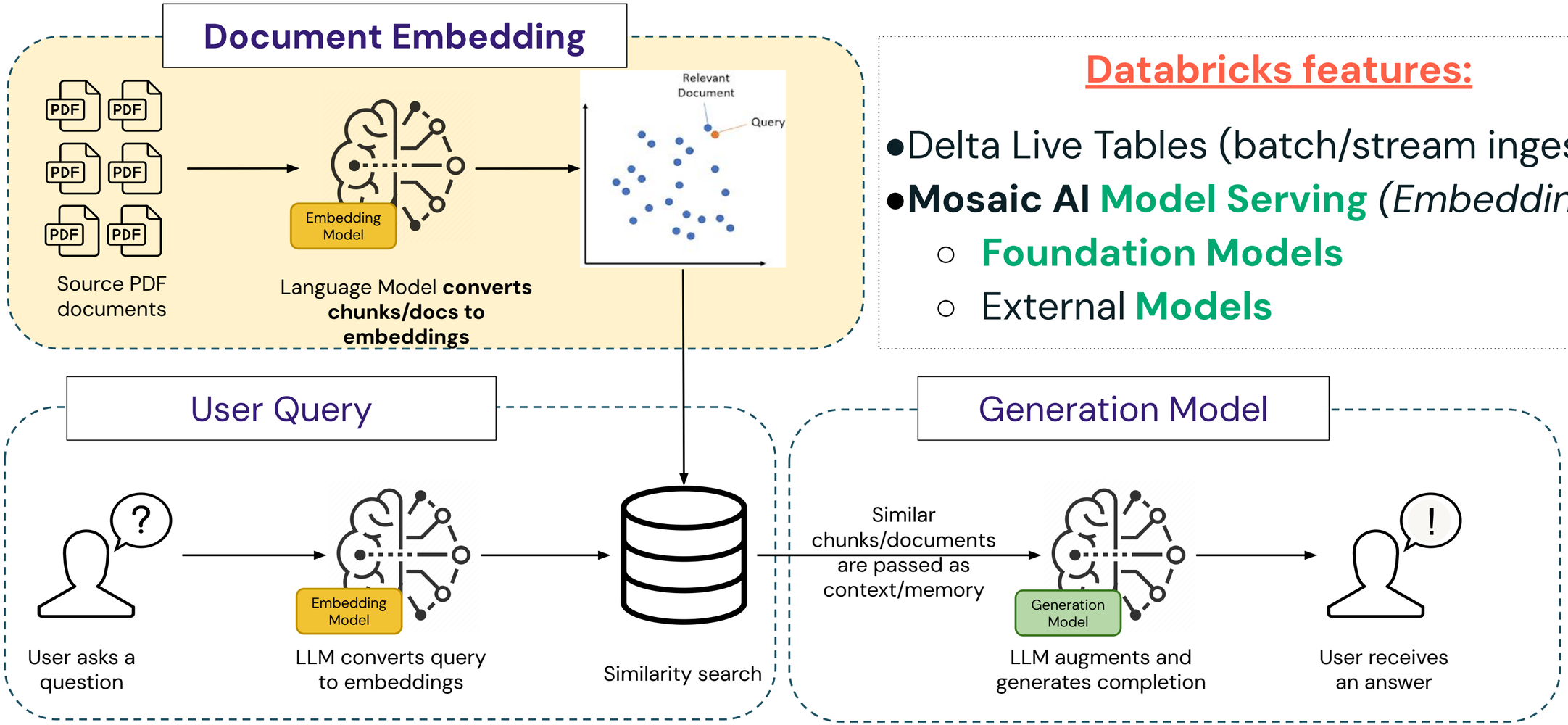
Retrieval Augmented Generation (RAG)

A sample architecture



Retrieval Augmented Generation (RAG)

A sample architecture

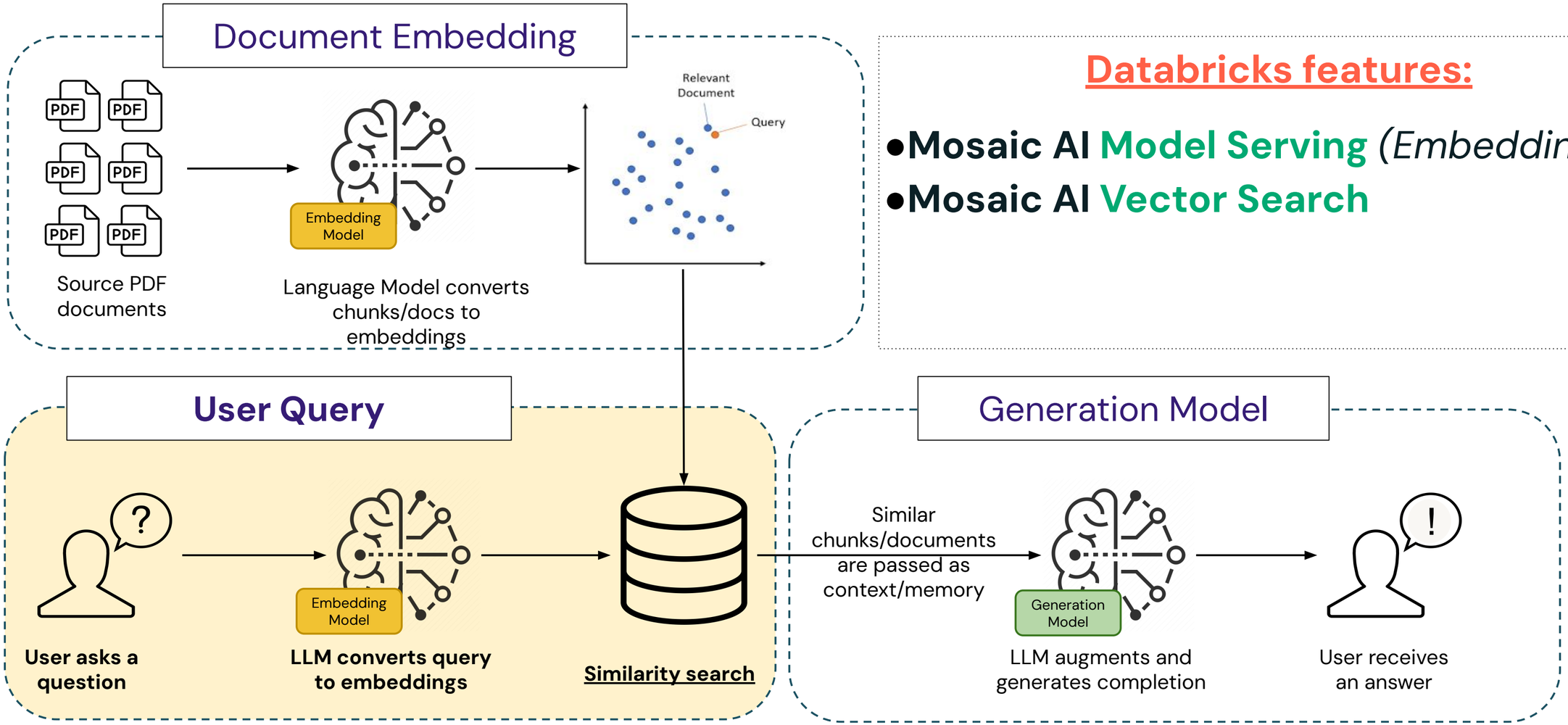


Databricks features:

- Delta Live Tables (batch/stream ingestion)
- **Mosaic AI Model Serving** *(Embeddings)*:
  - **Foundation Models**
  - External **Models**

Retrieval Augmented Generation (RAG)

A sample architecture

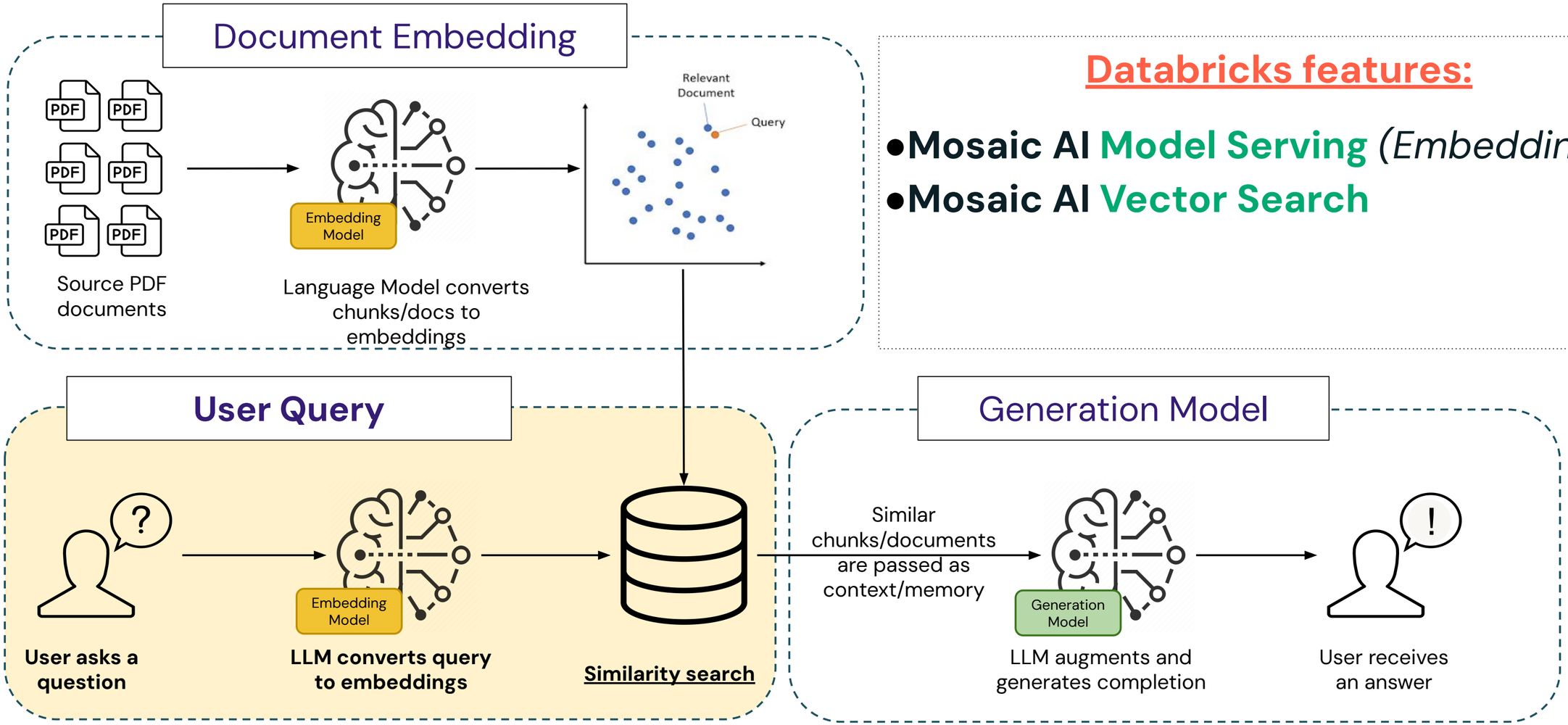


Databricks features:

- **Mosaic AI Model Serving** *(Embeddings)*
- **Mosaic AI Vector Search**

Retrieval Augmented Generation (RAG)

A sample architecture

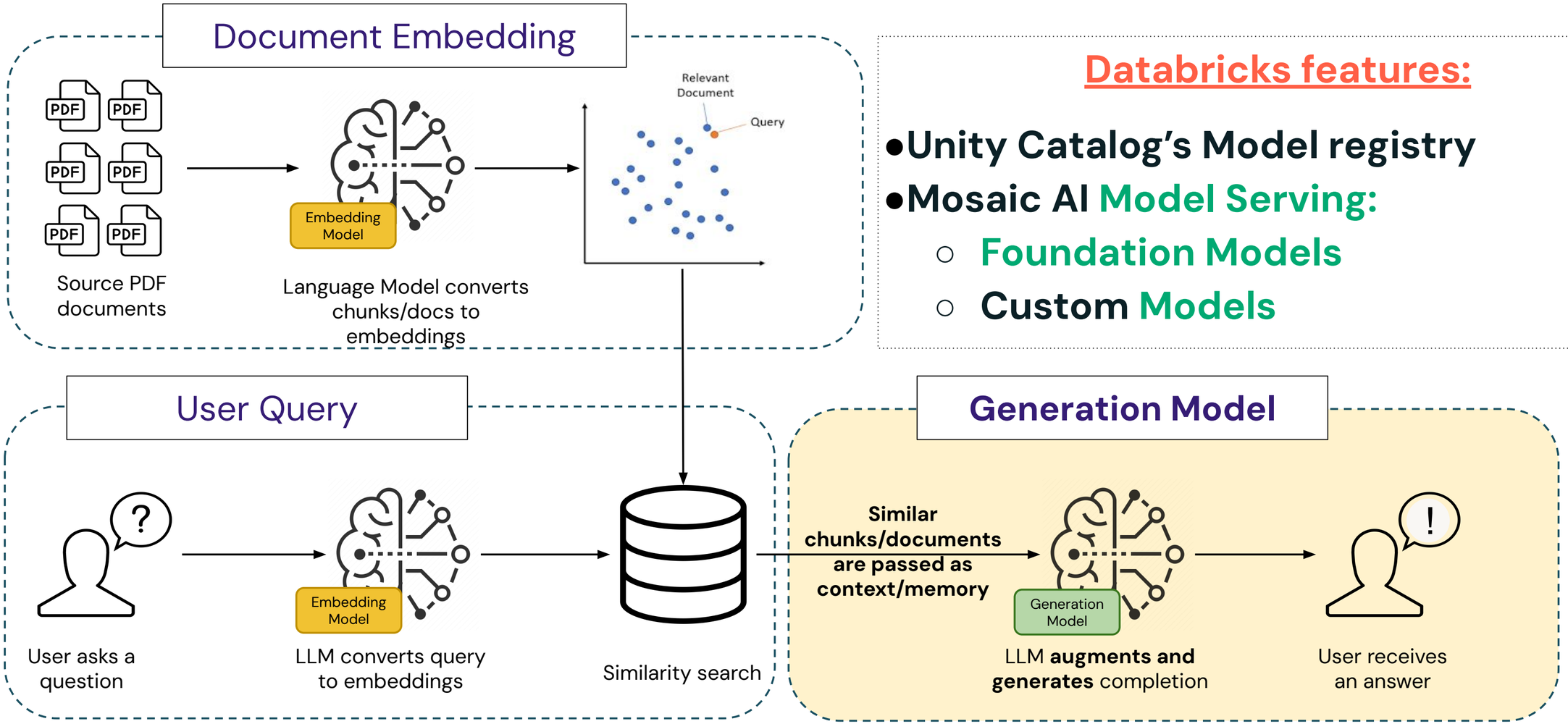


Databricks features:

- **Mosaic AI Model Serving** *(Embeddings)*
- **Mosaic AI Vector Search**

Retrieval Augmented Generation (RAG)

A sample architecture



Databricks features:

- **Unity Catalog's Model registry**
- **Mosaic AI Model Serving:**
  - **Foundation Models**
  - **Custom Models**

Benefits of RAG Architecture

Key benefits

Up-to-date and accurate responses

- LLM responses are not based solely on static training data.
- The model uses **up-to-date external data** sources to provide responses.

Reducing inaccurate responses, or hallucinations

- RAG attempts to mitigate the risk of producing "**hallucinations**" or incorrect information.
- Outputs can include citations of **original sources**, allowing human verification.

Benefits of RAG Architecture

Key benefits

Domain-specific contextualization

- Can be tailored to interface with **proprietary or domain-specific** data.
- Improves the accuracy of the response by using contextually relevant information.

Efficiency and cost-effectiveness

- Offers an alternative to fine-tuning LLMs by enabling in-context learning **without the up-front overhead of fine-tuning**.
- Beneficial where models need to be frequently updated with new data.

Mapping the RAG Workflow to Databricks Features

Find Relevant Context →

Mosaic AI Vector Search

- Built into the Databricks Intelligence Platform.
- Integrated with governance and productivity tools.
- Create and automatically sync with the underlying Delta table.

Generate Response →

Mosaic AI Model Serving

- State-of-the-art **foundation models** (e.g. Llama-3) and **external models**(e.g. GPT-4) made available by **Foundation Model APIs**.
- **Mosaic AI Playground:** Interact with supported models.

Serve RAG Chain →

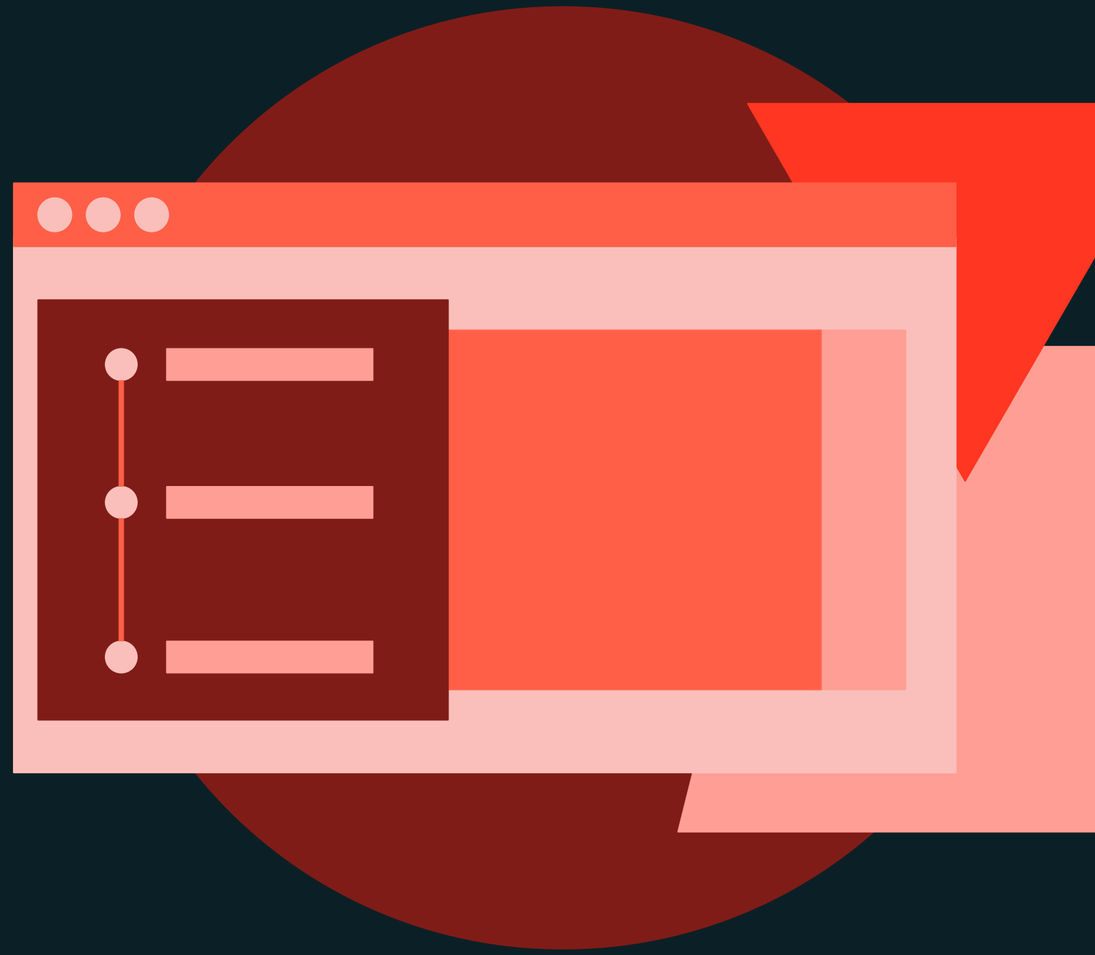
Mosaic AI Model Serving

- Unified interface for deploying and querying models.
- Low-latency, high-availability model deployment service.
- Scales up or down to meet demand changes.

databricks

DEMONSTRATION

In Context Learning with AI Playground



Demo Outline

In-Context Learning with AI Playground

What we'll cover:

- Accessing the AI Playground
- Simple Prompt Which Hallucinates:
  - Prompt the model to provide a response to a query that you know has no knowledge.
  - Review the response for hallucinations, incorrect information, or lack of detail.
- Simple Prompt Which Does Not Hallucinate:
  - Prompt the model to avoid hallucinations by providing clear instructions.
- Augment Prompt with Additional Context
  - Provide the prompt with a document or reference containing correct information as supplemental information.
  - Review the new response for changes in detail given the additional information.

databricks

LAB EXERCISE

In Context Learning with AI Playground

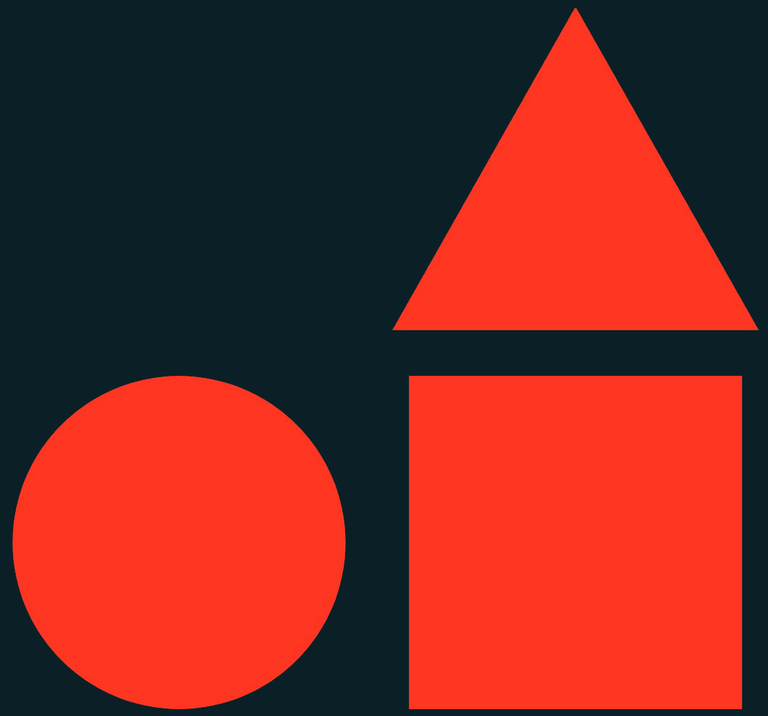
Lab Outline

What you'll do:

- **Task 1 :** Access the Mosaic AI Playground
- **Task 2 :** Prompt Which Hallucinates
- **Task 3 :** Prompt Which Does Not Hallucinate
- **Task 4 :** Augmenting the Prompt with Additional Context and Analyzing the Impact of Additional Context

databricks

Preparing Data for RAG Solutions



Generative AI Solution Development

Learning Objectives

- List potential consequences of improperly prepared data for RAG solutions.
- Describe the importance of the chunking strategy and embedding model.
- Describe strategies for preparing data for RAG solutions.
- Understand how Delta Lake and Unity Catalog support RAG patterns.
- Map Databricks products for structured and unstructured data preparation.

databricks

LECTURE

Preparing Data for RAG Solutions

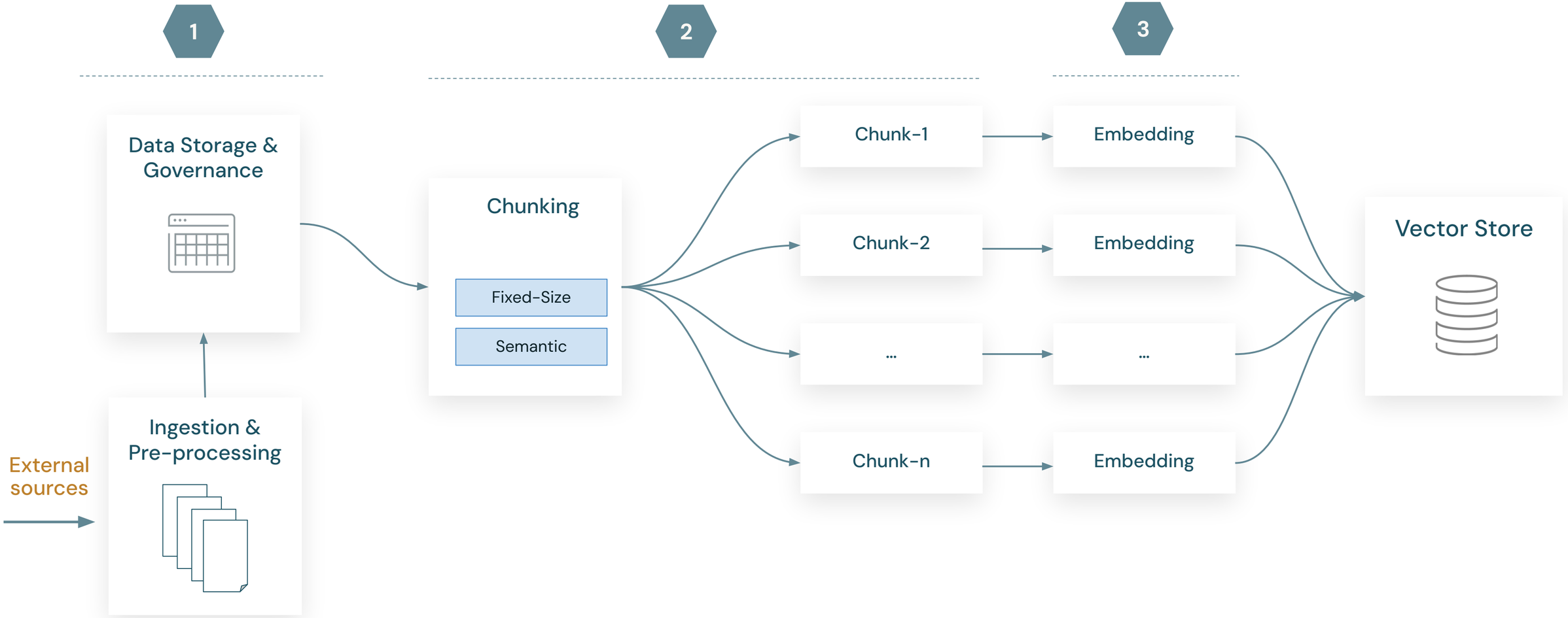
Why is Data Prep Important for RAG?

Potential **issues** when data is prepared improperly

- **Poor quality model output:** If data is inaccurate, incomplete, or biased, the RAG system is more likely to produce misleading or incorrect responses.
- **“Lost in the middle”**: In long context, LLMs tend to overlook the documents placed in the middle (Related Research Paper and needle in haystack test repo).
- **Inefficient retrieval**: Poorly prepared data would decrease the accuracy and precision of retrieving relevant information from knowledge base.
- **Exposing data**: Poor data governance could lead to exposing data during the retrieval process.
- **Wrong embedding model**: Wrong embedding model would decrease the quality of embeddings and retrieval accuracy.

Data Prep Process Overview

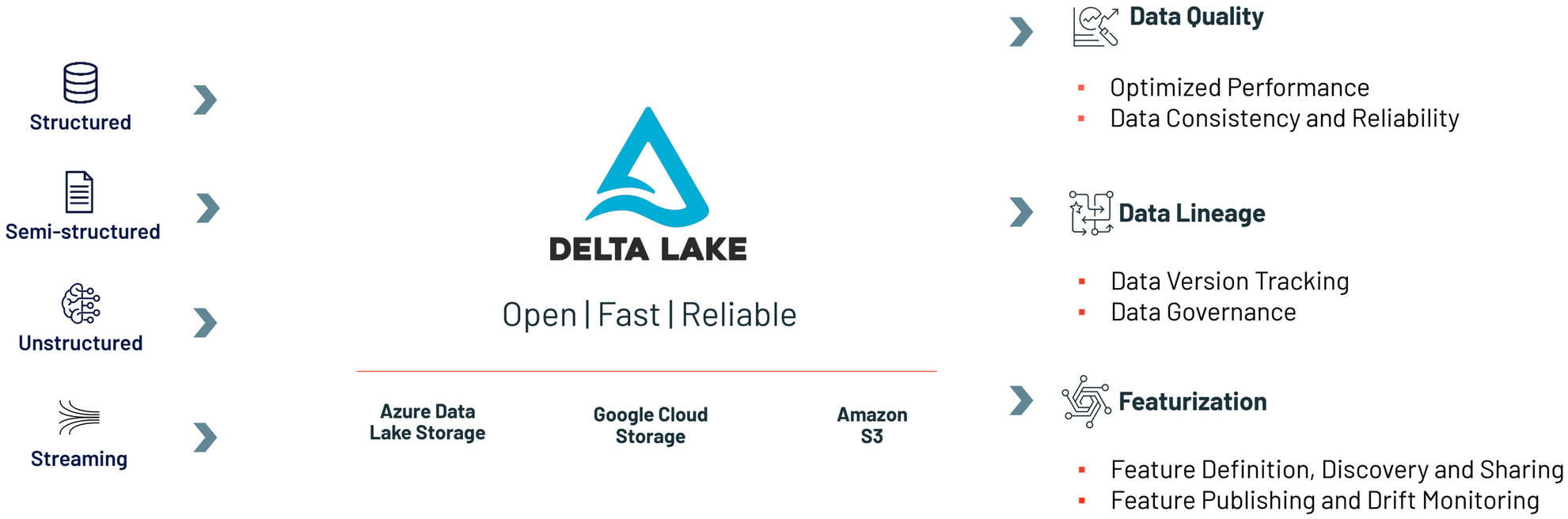
A simple data prep process



1. Data Storage and Governance

What is Delta Lake?

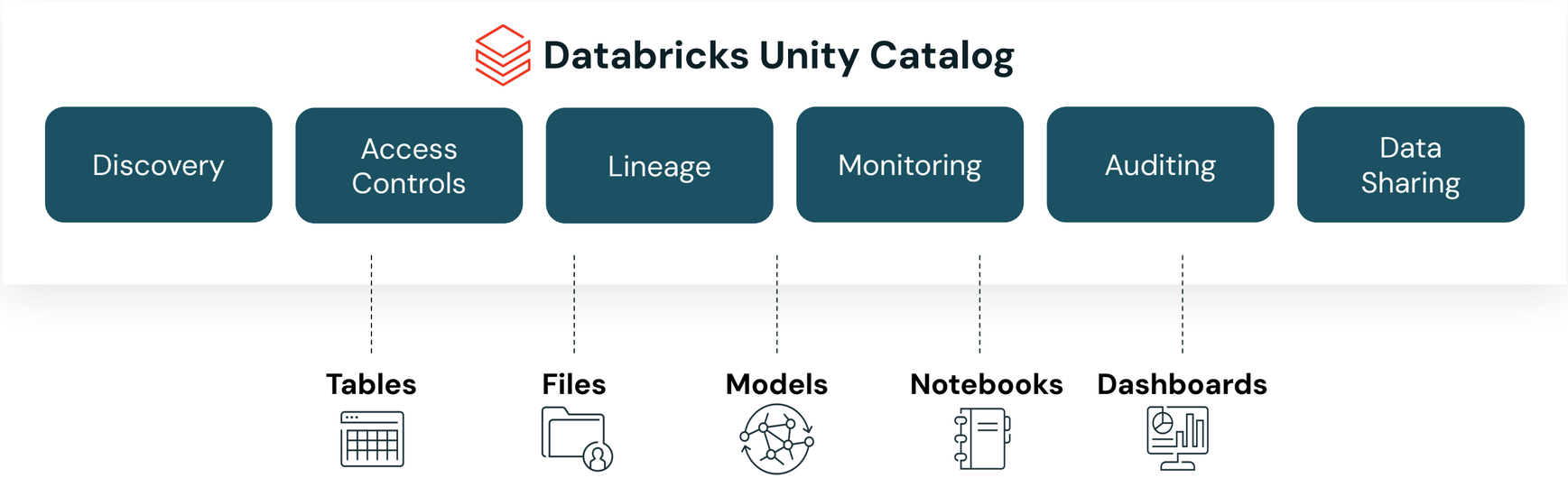
- A **unified data management layer** that brings data reliability and fast analytics to cloud data lakes. It is the optimized storage layer that provides the foundation for storing data and tables in the Databricks DI Platform.



Unity Catalog (UC)

Governance, discovery, access control for RAG applications

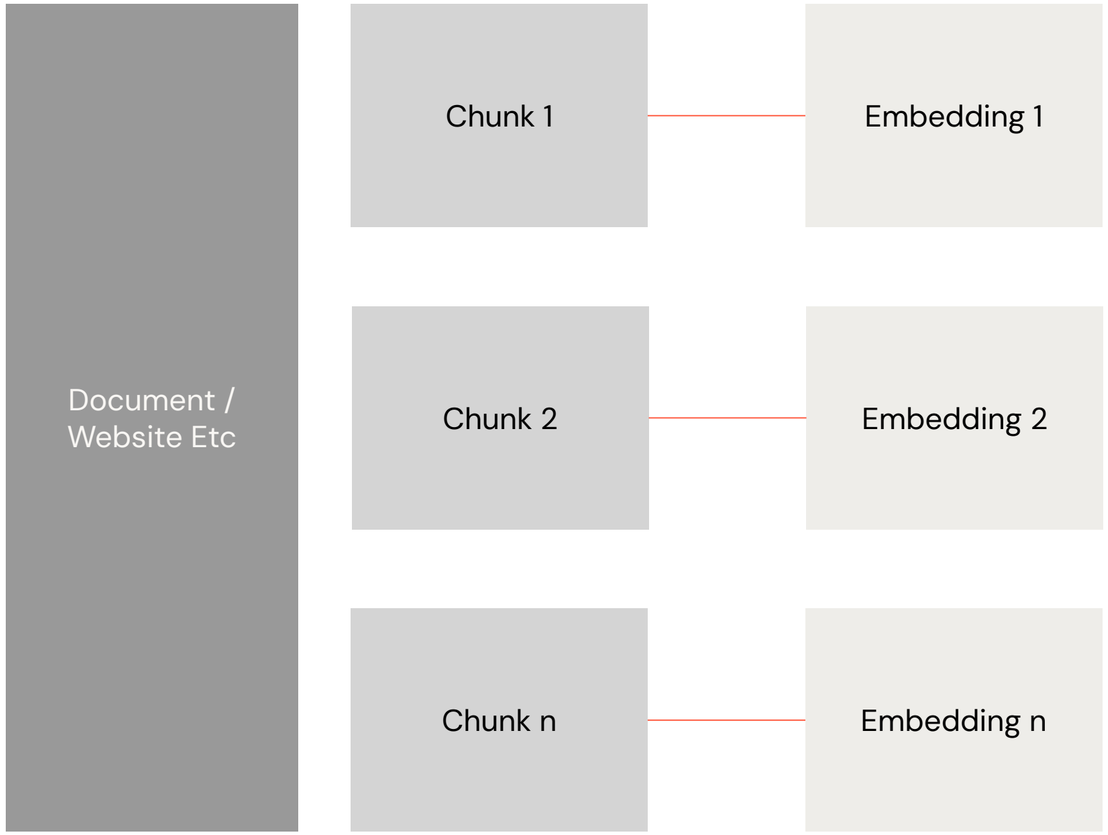
- Unified visibility into data and AI
- Single permission model for data and AI
- Open data sharing



02. Data Extraction & Chunking

Data/Document Extraction

Ingesting (text) documents and making them searchable



Typical Process:

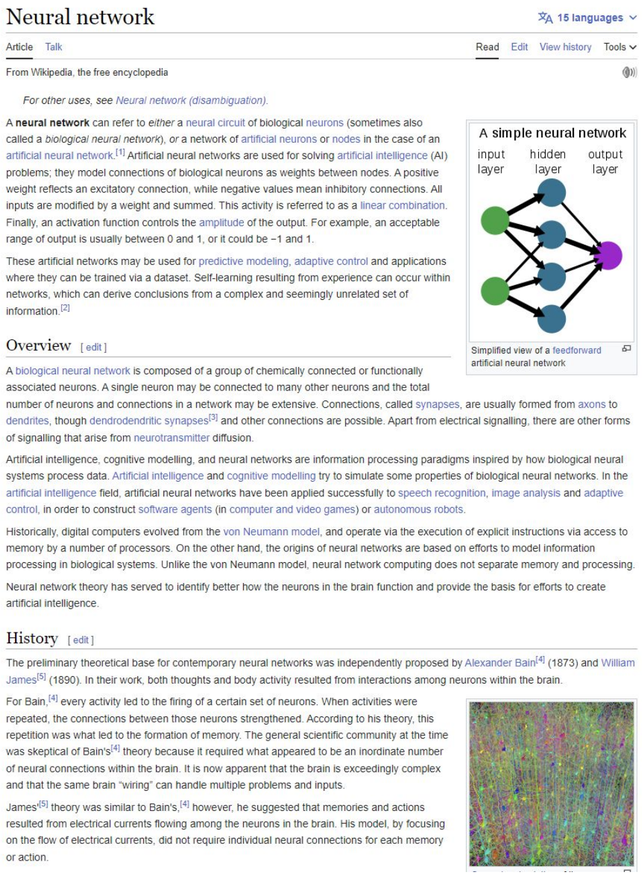
1. Split documents into chunks
2. Embed the chunks with a model
3. Store them in a vector store

Constraints/Risks:

- **Chunks could be out of Context**
- **No overall logic/rigor**

How to Chunk Data?

How should we organise it?



Neural network

15 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

For other uses, see Neural network (disambiguation).

A **neural network** can refer to *either* a neural circuit of biological neurons (sometimes also called a *biological neural network*), *or* a network of artificial neurons or nodes in the case of an artificial neural network.[1] Artificial neural networks are used for solving artificial intelligence (AI) problems; they model connections of biological neurons as weights between nodes. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed. This activity is referred to as a linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be −1 and 1.

These artificial networks may be used for predictive modeling, adaptive control and applications where they can be trained via a dataset. Self-learning resulting from experience can occur within networks, which can derive conclusions from a complex and seemingly unrelated set of information.[2]

A simple neural network

input layer hidden layer output layer

Simplified view of a feedforward artificial neural network

Overview [edit]

A biological neural network is composed of a group of chemically connected or functionally associated neurons. A single neuron may be connected to many other neurons and the total number of neurons and connections in a network may be extensive. Connections, called synapses, are usually formed from axons to dendrites, though dendrodendritic synapses[3] and other connections are possible. Apart from electrical signalling, there are other forms of signalling that arise from neurotransmitter diffusion.

Artificial intelligence, cognitive modelling, and neural networks are information processing paradigms inspired by how biological neural systems process data. Artificial intelligence and cognitive modelling try to simulate some properties of biological neural networks. In the artificial intelligence field, artificial neural networks have been applied successfully to speech recognition, image analysis and adaptive control, in order to construct software agents (in computer and video games) or autonomous robots.

Historically, digital computers evolved from the von Neumann model, and operate via the execution of explicit instructions via access to memory by a number of processors. On the other hand, the origins of neural networks are based on efforts to model information processing in biological systems. Unlike the von Neumann model, neural network computing does not separate memory and processing.

Neural network theory has served to identify better how the neurons in the brain function and provide the basis for efforts to create artificial intelligence.

History [edit]

The preliminary theoretical base for contemporary neural networks was independently proposed by Alexander Bain[4] (1873) and William James[5] (1890). In their work, both thoughts and body activity resulted from interactions among neurons within the brain.

For Bain,[4] every activity led to the firing of a certain set of neurons. When activities were repeated, the connections between those neurons strengthened. According to his theory, this repetition was what led to the formation of memory. The general scientific community at the time was skeptical of Bain's[4] theory because it required what appeared to be an inordinate number of neural connections within the brain. It is now apparent that the brain is exceedingly complex and that the same brain "wiring" can handle multiple problems and inputs.

James[5] theory was similar to Bain's,[4] however, he suggested that memories and actions resulted from electrical currents flowing among the neurons in the brain. His model, by focusing on the flow of electrical currents, did not require individual neural connections for each memory or action.

Title

Section

Diagram

Semantic Chunking:

- Chunk by sentence/paragraph/section
- Leverage special punctuation (i.e. '.', '\n')
- Include/Inject metadata/tags/title(s)

&/OR

Fixed-size Chunking:

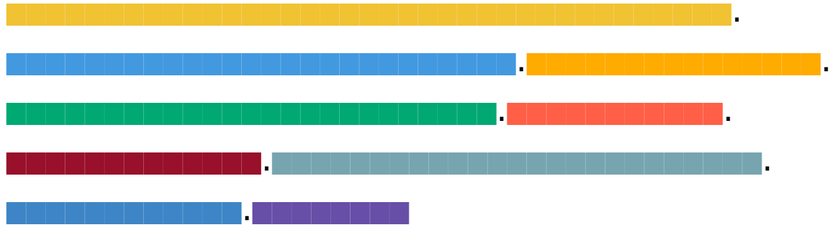
- Divide by a specific number of tokens
- Simple and computationally cheap method

Chunking Strategy is Use-Case Specific

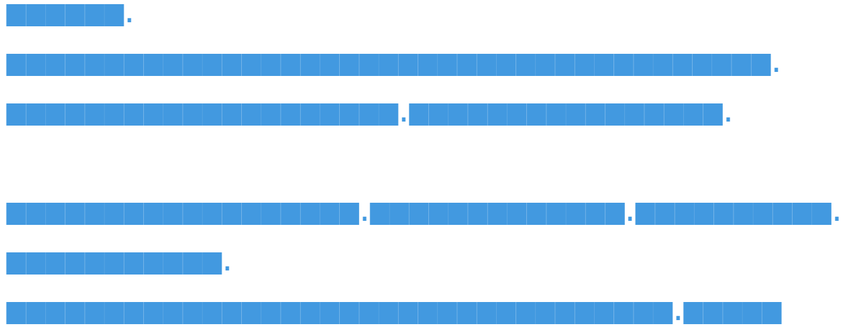
Another iterative step! Experiment with different chunk sizes and approaches

- How long are our documents?
  - 1 sentence?
  - N sentences?
- If 1 chunk = **1 sentence**, embeddings focus on specific meaning
- If 1 chunk = **multiple paragraphs**, embeddings capture broader theme
  - How about splitting by headers?

Chunking by sentence:



Chunking by Paragraph:



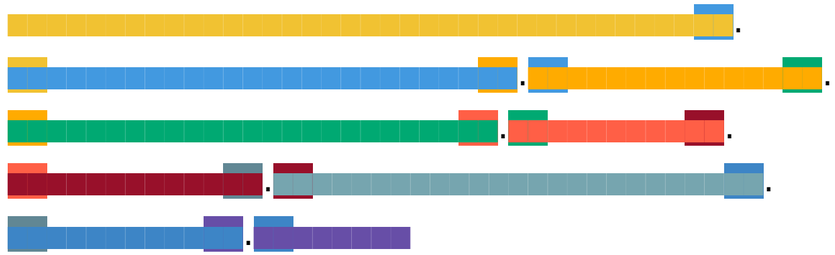
Use ChunkViz helper tool

Chunking Strategy is Use-Case Specific

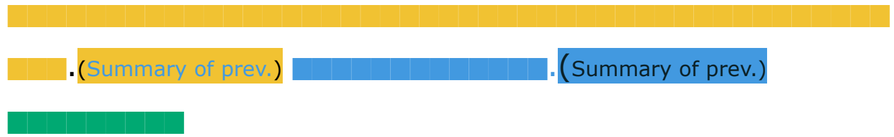
Another iterative step! Experiment with different chunk sizes and approaches

- Chunk **overlap** defines the amount of overlap between consecutive chunks, ensuring that no contextual information is lost between them.
- **Windowed summarization** is a 'context-enriching' chunking method where each chunk includes a 'windowed summary' of previous few chunks.

Chunk overlap:



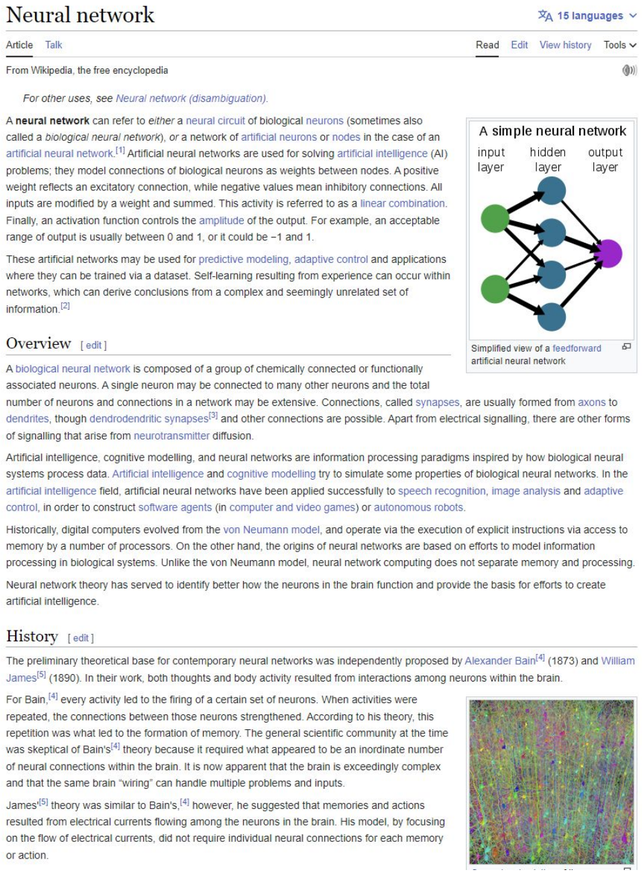
Windowed summarization:



- Prior knowledge of user's query patterns can be helpful *(i.e. query length?)*
  - While long queries may have better aligned embeddings to returned chunks, shorter queries could be more precise

Advanced Chunking Strategies

Summarization


Neural network

15 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

For other uses, see Neural network (disambiguation).

A **neural network** can refer to *either* a neural circuit of biological neurons (sometimes also called a *biological neural network*), *or* a network of artificial neurons or nodes in the case of an artificial neural network.[1] Artificial neural networks are used for solving artificial intelligence (AI) problems; they model connections of biological neurons as weights between nodes. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed. This activity is referred to as a linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be −1 and 1.

These artificial networks may be used for predictive modeling, adaptive control and applications where they can be trained via a dataset. Self-learning resulting from experience can occur within networks, which can derive conclusions from a complex and seemingly unrelated set of information.[2]

A simple neural network

input layer hidden layer output layer

Simplified view of a feedforward artificial neural network

Overview [edit]

A biological neural network is composed of a group of chemically connected or functionally associated neurons. A single neuron may be connected to many other neurons and the total number of neurons and connections in a network may be extensive. Connections, called synapses, are usually formed from axons to dendrites, though dendrodendritic synapses[3] and other connections are possible. Apart from electrical signalling, there are other forms of signalling that arise from neurotransmitter diffusion.

Artificial intelligence, cognitive modelling, and neural networks are information processing paradigms inspired by how biological neural systems process data. Artificial intelligence and cognitive modelling try to simulate some properties of biological neural networks. In the artificial intelligence field, artificial neural networks have been applied successfully to speech recognition, image analysis and adaptive control, in order to construct software agents (in computer and video games) or autonomous robots.

Historically, digital computers evolved from the von Neumann model, and operate via the execution of explicit instructions via access to memory by a number of processors. On the other hand, the origins of neural networks are based on efforts to model information processing in biological systems. Unlike the von Neumann model, neural network computing does not separate memory and processing.

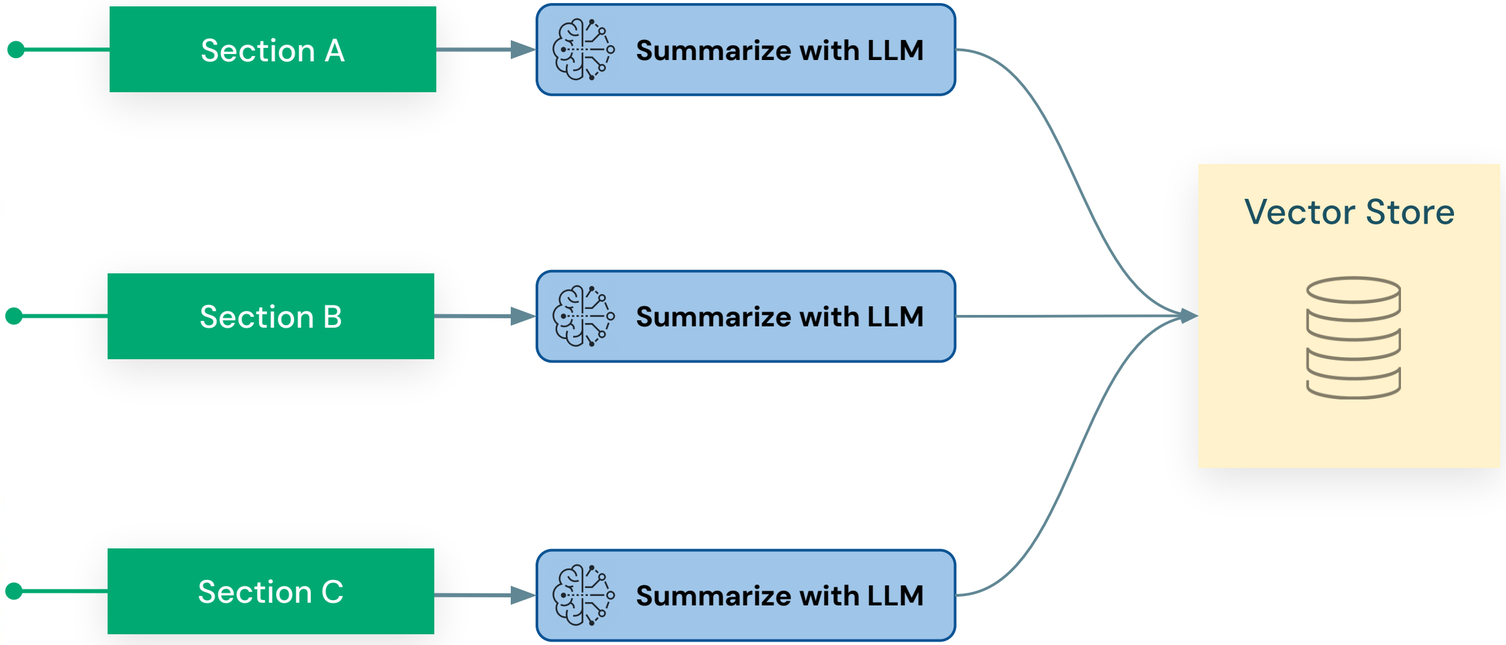
Neural network theory has served to identify better how the neurons in the brain function and provide the basis for efforts to create artificial intelligence.

History [edit]

The preliminary theoretical base for contemporary neural networks was independently proposed by Alexander Bain[4] (1873) and William James[5] (1890). In their work, both thoughts and body activity resulted from interactions among neurons within the brain.

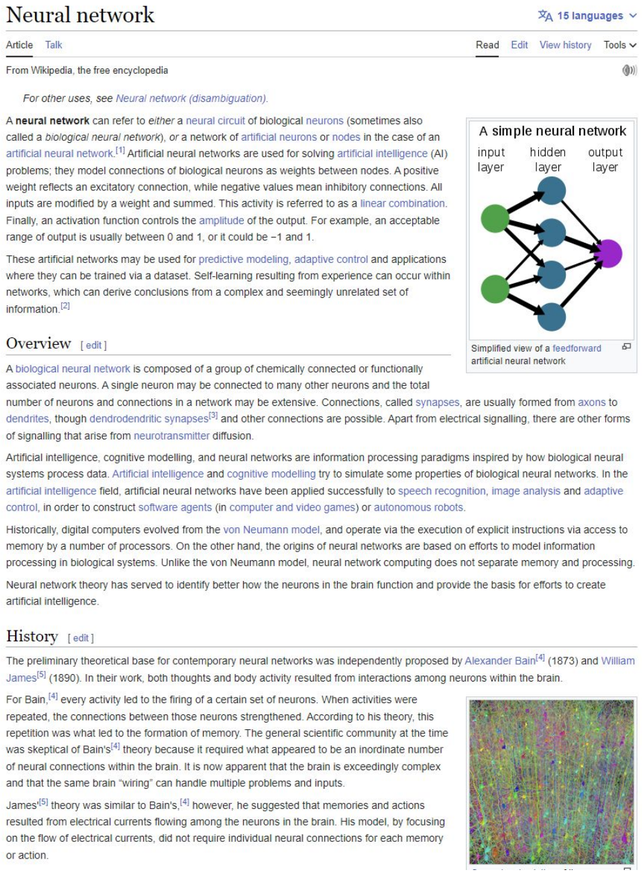
For Bain,[4] every activity led to the firing of a certain set of neurons. When activities were repeated, the connections between those neurons strengthened. According to his theory, this repetition was what led to the formation of memory. The general scientific community at the time was skeptical of Bain's[4] theory because it required what appeared to be an inordinate number of neural connections within the brain. It is now apparent that the brain is exceedingly complex and that the same brain "wiring" can handle multiple problems and inputs.

James'[5] theory was similar to Bain's,[4] however, he suggested that memories and actions resulted from electrical currents flowing among the neurons in the brain. His model, by focusing on the flow of electrical currents, did not require individual neural connections for each memory or action.



Advanced Chunking Strategies

Summarization with metadata


Neural network

15 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

For other uses, see Neural network (disambiguation).

A **neural network** can refer to *either* a neural circuit of biological neurons (sometimes also called a *biological neural network*), *or* a network of artificial neurons or nodes in the case of an artificial neural network.[1] Artificial neural networks are used for solving artificial intelligence (AI) problems; they model connections of biological neurons as weights between nodes. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by a weight and summed. This activity is referred to as a linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be −1 and 1.

These artificial networks may be used for predictive modeling, adaptive control and applications where they can be trained via a dataset. Self-learning resulting from experience can occur within networks, which can derive conclusions from a complex and seemingly unrelated set of information.[2]

A simple neural network

input layer hidden layer output layer

Simplified view of a feedforward artificial neural network

Overview [edit]

A biological neural network is composed of a group of chemically connected or functionally associated neurons. A single neuron may be connected to many other neurons and the total number of neurons and connections in a network may be extensive. Connections, called synapses, are usually formed from axons to dendrites, though dendrodendritic synapses[3] and other connections are possible. Apart from electrical signalling, there are other forms of signalling that arise from neurotransmitter diffusion.

Artificial intelligence, cognitive modelling, and neural networks are information processing paradigms inspired by how biological neural systems process data. Artificial intelligence and cognitive modelling try to simulate some properties of biological neural networks. In the artificial intelligence field, artificial neural networks have been applied successfully to speech recognition, image analysis and adaptive control, in order to construct software agents (in computer and video games) or autonomous robots.

Historically, digital computers evolved from the von Neumann model, and operate via the execution of explicit instructions via access to memory by a number of processors. On the other hand, the origins of neural networks are based on efforts to model information processing in biological systems. Unlike the von Neumann model, neural network computing does not separate memory and processing.

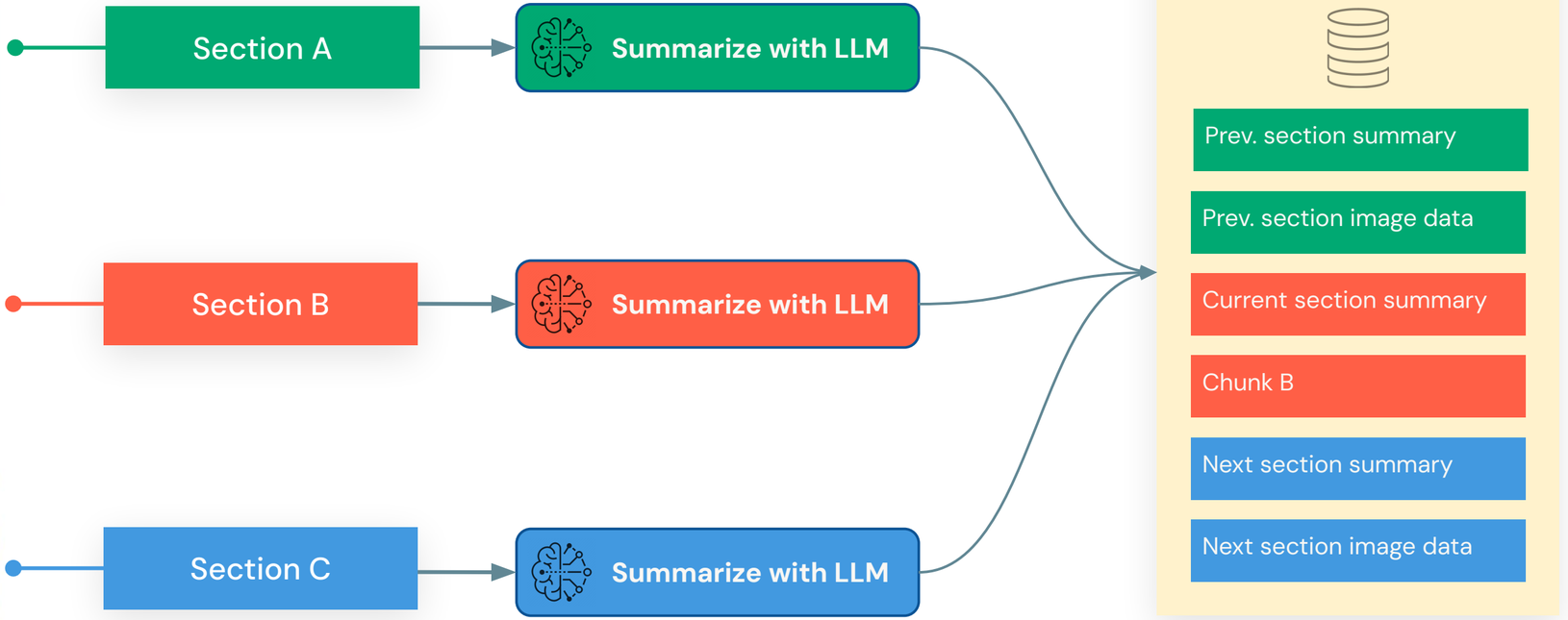
Neural network theory has served to identify better how the neurons in the brain function and provide the basis for efforts to create artificial intelligence.

History [edit]

The preliminary theoretical base for contemporary neural networks was independently proposed by Alexander Bain[4] (1873) and William James[5] (1890). In their work, both thoughts and body activity resulted from interactions among neurons within the brain.

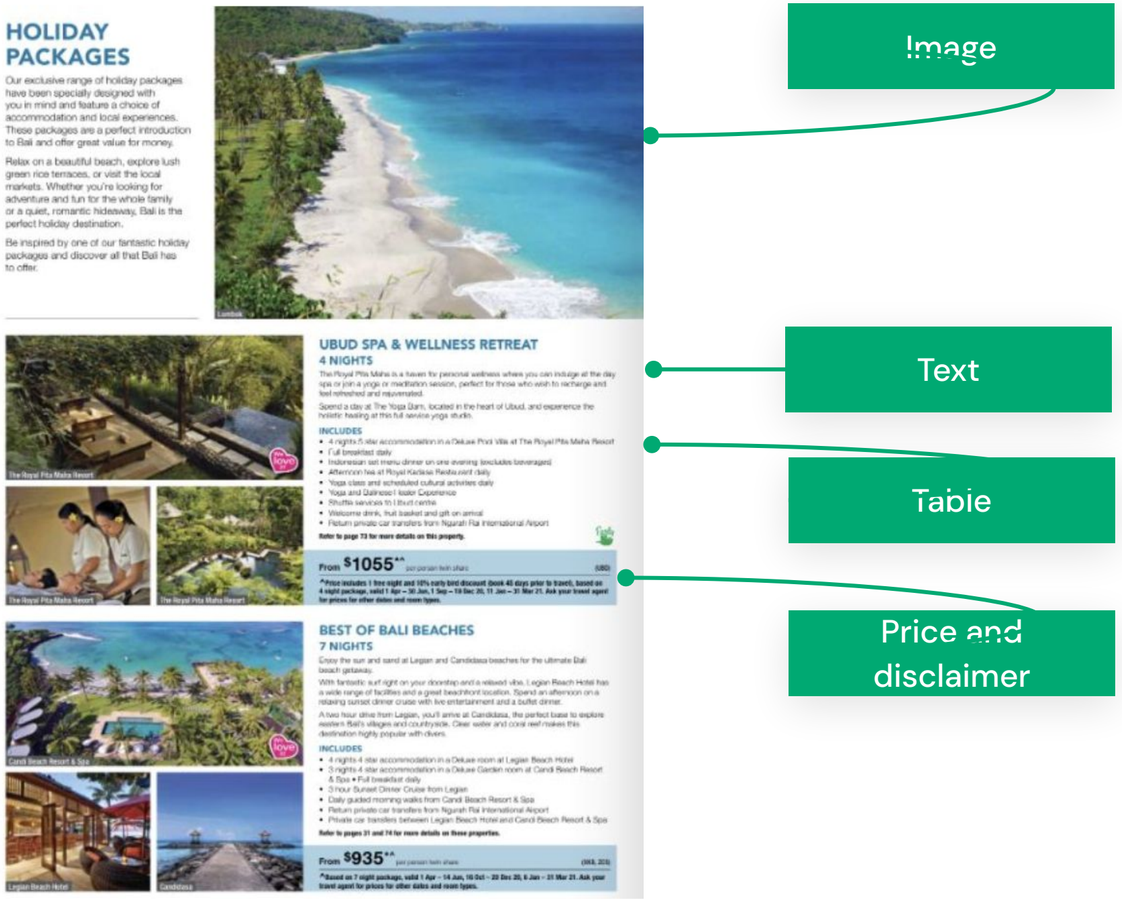
For Bain,[4] every activity led to the firing of a certain set of neurons. When activities were repeated, the connections between those neurons strengthened. According to his theory, this repetition was what led to the formation of memory. The general scientific community at the time was skeptical of Bain's[4] theory because it required what appeared to be an inordinate number of neural connections within the brain. It is now apparent that the brain is exceedingly complex and that the same brain "wiring" can handle multiple problems and inputs.

James'[5] theory was similar to Bain's,[4] however, he suggested that memories and actions resulted from electrical currents flowing among the neurons in the brain. His model, by focusing on the flow of electrical currents, did not require individual neural connections for each memory or action.



Data Extraction and Chunking Challenges

Working with complex documents



Image

Text

Table

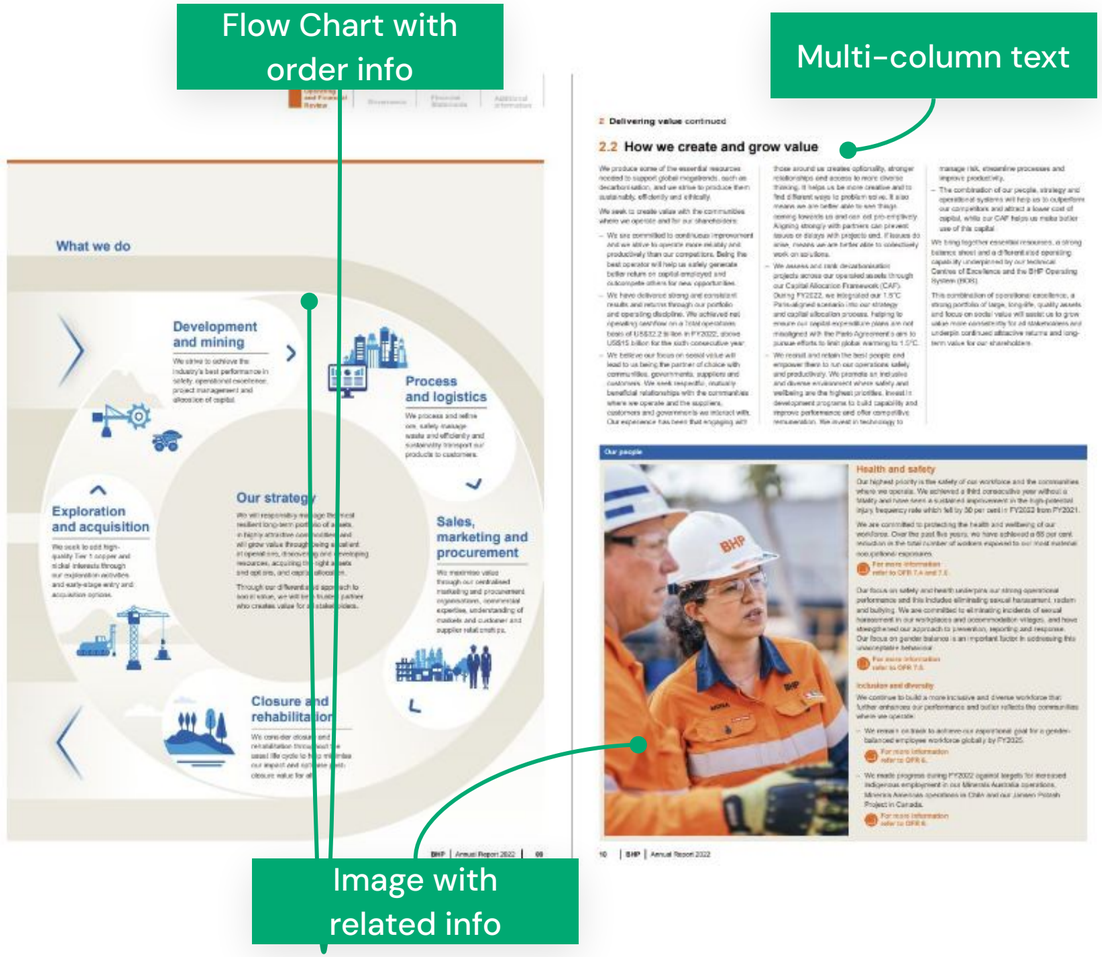
Price and disclaimer

Other challenges:

- Text mixed with image
- Irregular placement of text
- Color encoded focus *(Important for context)*

Data Extraction and Chunking Challenges

Working with complex documents



Other challenges:

- Chart with hierarchical information. Keeping the order of the information is critical.
- Multi-column text and the order of columns if crucial.
- Keeping images with related information is crucial.

Data Extraction and Chunking Challenges

Working with different document types

- Preserving logical sections of the content might be difficult for some document types.
  - HTML requires tag based chunking to preserve logical structure of the content.
- HTML Tables are token heavy.
- Example: 3x3 small table;
  - Plain text: 11 tokens
  - JSONL: 29 tokens
  - HTML: 132 tokens (‼️)

```html
<table style="border-collapse: collapse; width: 50%;">
    <tr>
        <th style="background-color: #f2f2f2; padding: 8px;">Header 1</th>
        <th style="background-color: #f2f2f2; padding: 8px;">Header 2</th>
        <th style="background-color: #f2f2f2; padding: 8px;">Header 3</th>
    </tr>
    <tr>
        <td style="border: 1px solid #ddd; padding: 8px;">Row 1, Cell 1</td>
        <td style="border: 1px solid #ddd; padding: 8px;">Row 1, Cell 2</td>
        <td style="border: 1px solid #ddd; padding: 8px;">Row 1, Cell 3</td>
    </tr>
    <tr>
        <td style="border: 1px solid #ddd; padding: 8px;">Row 2, Cell 1</td>
        <td style="border: 1px solid #ddd; padding: 8px;">Row 2, Cell 2</td>
        <td style="border: 1px solid #ddd; padding: 8px;">Row 2, Cell 3</td>
    </tr>
    <tr>
        <td style="border: 1
        <td style="border: 1
        <td style="border: 1
    </tr>
</table>
```

```text
Header 1    Header 2    Header 3
Row 1, Cell 1    Row 1, Cell 2    Row 1, Cell 3
Row 2, Cell 1    Row 2, Cell 2    Row 2, Cell 3
Row 3, Cell 1    Row 3, Cell 2    Row 3, Cell 3
```

```json
{
    "table": [
        {
            "Header 1": "Row 1, Cell 1",
            "Header 2": "Row 1, Cell 2",
            "Header 3": "Row 1, Cell 3"
        },
        {
            "Header 1": "Row 2, Cell 1",
            "Header 2": "Row 2, Cell 2",
            "Header 3": "Row 2, Cell 3"
        },
        {
            "Header 1": "Row 3, Cell 1",
            "Header 2": "Row 3, Cell 2",
            "Header 3": "Row 3, Cell 3"
        }
    ]
}
```

General Approaches

Approaches to address unstructured/complex raw text documents

Traditional Approach

Libraries:

- PyMuPDF

Features:

- Breaks down text to into raw constructs
- Very low level requires hard coding rules

Use a layout model(*)

Libraries:

- Hugging Face's LayoutLMv3
- doctr
- PyPDF
- Unstructured

Features:

- Apply Deep learning models built to do text extraction and context extraction

Multi-Modal Models

Models:

- OpenAI's GPT-4o (and beyond)
- Alphabet's Gemini1.5 (and beyond)
- Other OSS models (i.e. Dolphin's Series, OpenFlamingo, Llava, OLMo)

Features:

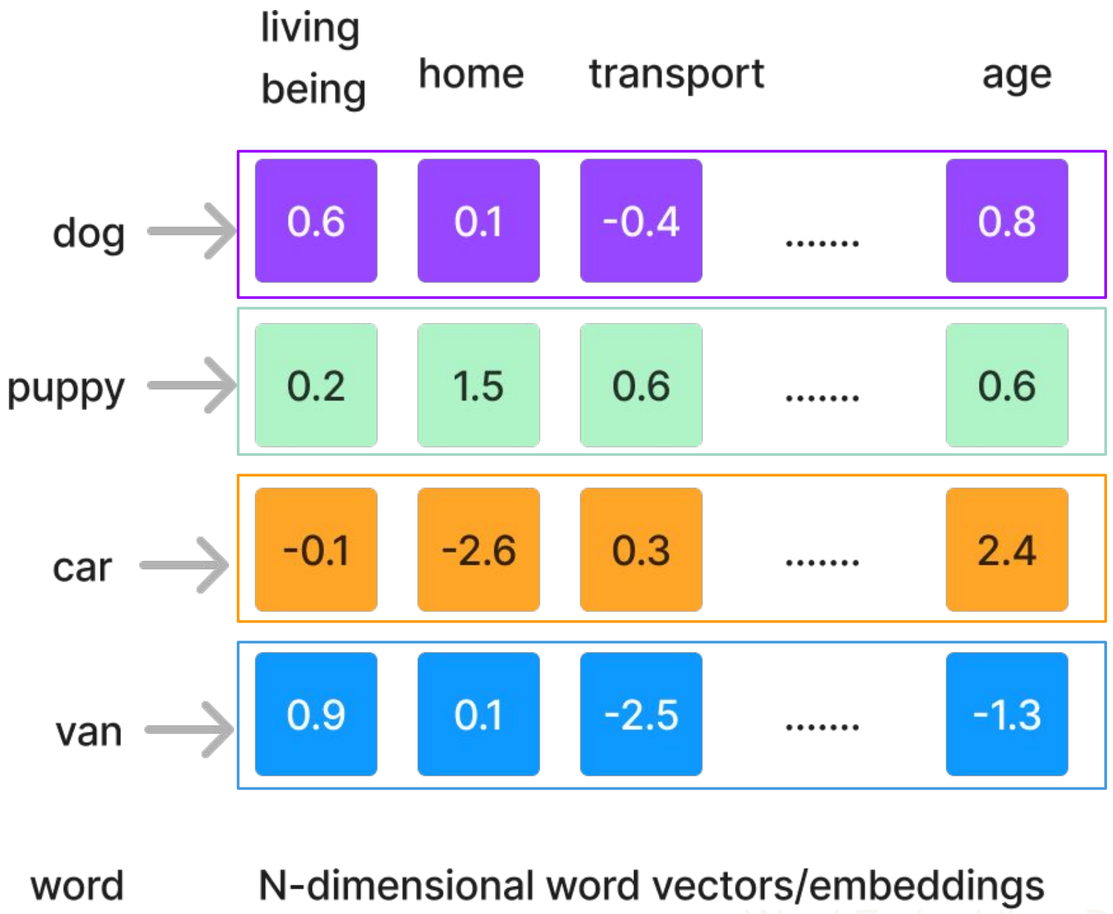
- Multimodal LLMs intrinsically understand images but are still more experimental at this stage

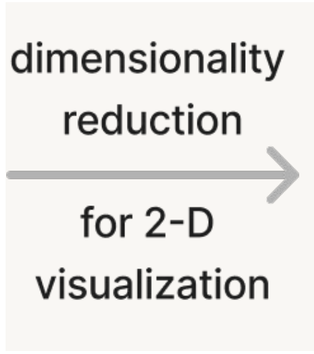
(*) the spatial arrangement and organization of elements within a document

03. Embedding Model

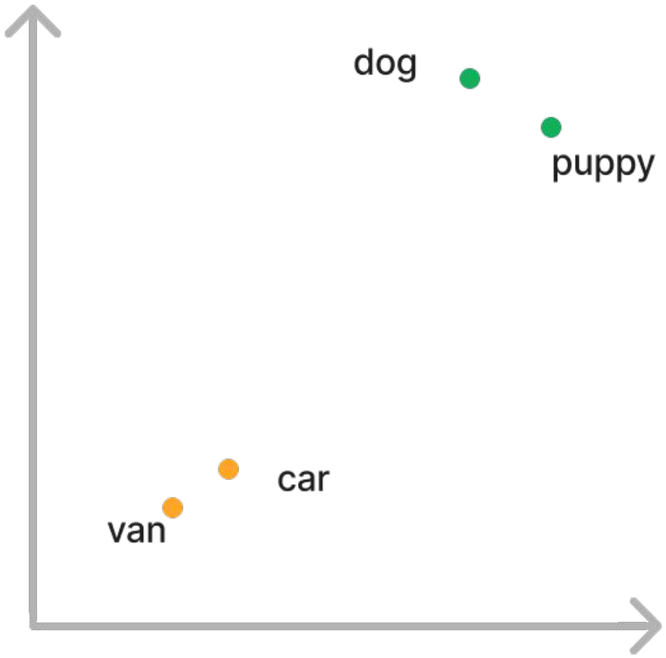
Refresher: Representing Words with Vectors

Embedding: A numerical representation of content





We can project these vectors onto 2D to see how they relate graphically



Word Embedding: Basics, Create a vector from a word | by Hariom Gautam | Medium

Embedding Models

Choosing the **right model for your application**

- Data/Text properties:
  - Vocabulary size in your text/documents (some models handles more diverse words)
  - Domain/Topic (i.e. finance, medical, news etc.)
  - Text length: typical length of chunks/docs to be embedded
- Model capabilities:
  - Multi-Language support
  - Embedding dimensions/size: more storage cost for higher dimensions

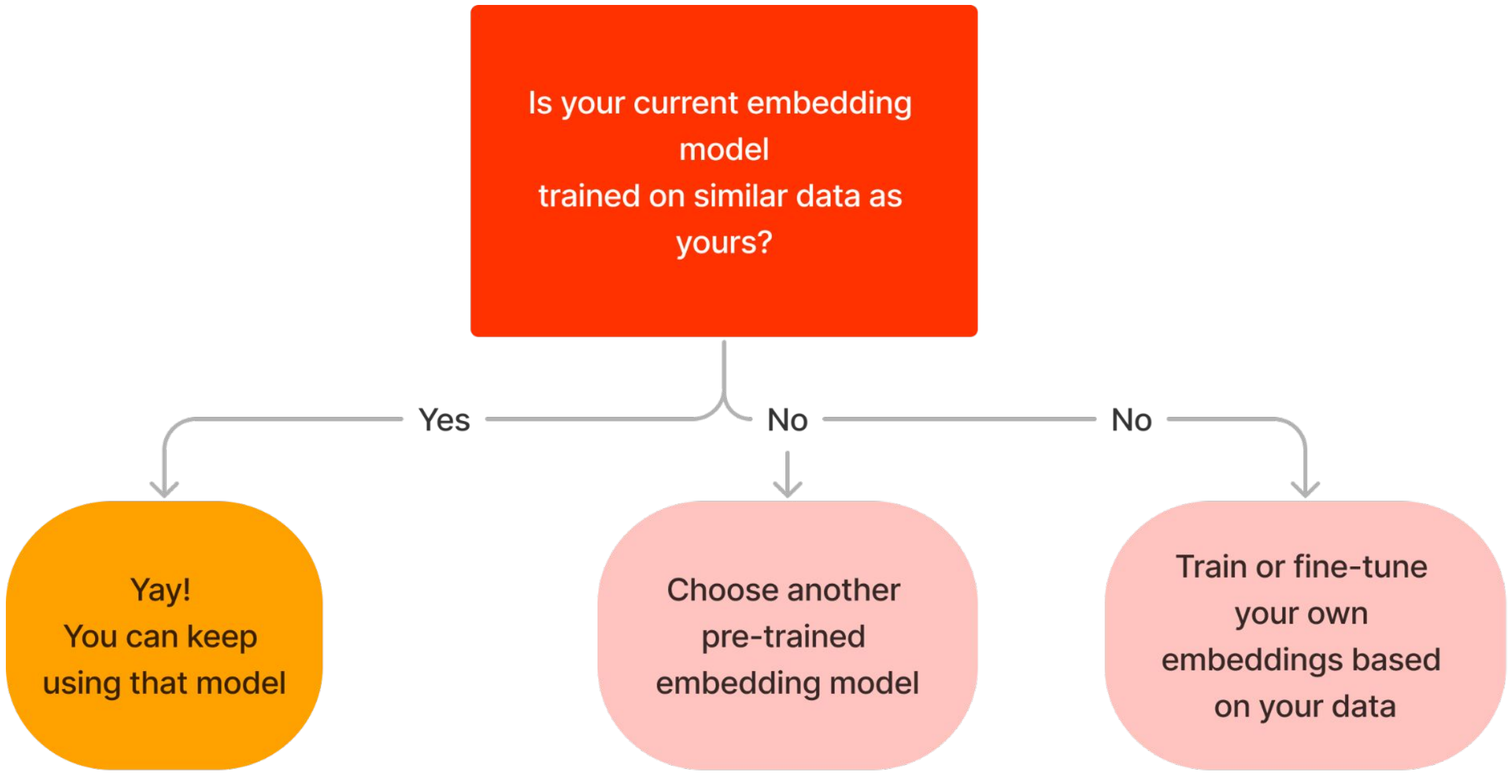
Practical considerations:

- Be aware of context window limitations. Many embedding models will **ignore text beyond their context window limits**.
- Privacy and cost/licensing when using proprietary API-based models.

⇒ benchmark multiple models & choose the one that strikes the best balance.

Tip 1: Choose Your Embedding Model Wisely

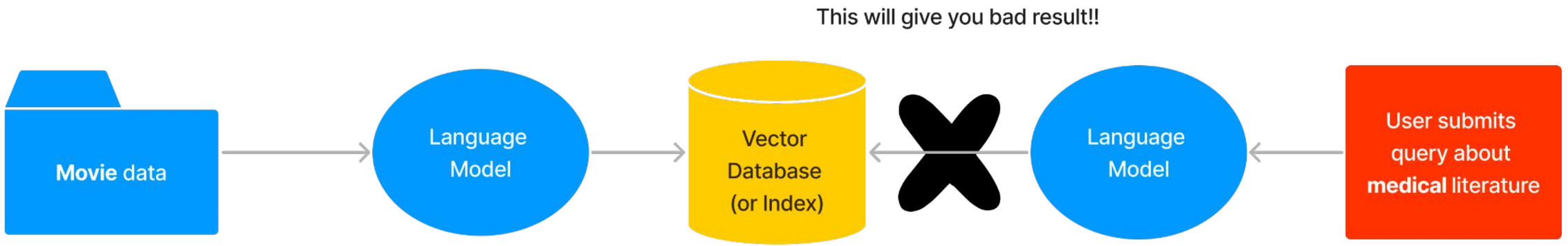
The embedding model should represent BOTH queries and documents



This practice has been around for years in NLP.
Example: Fine-tune BERT embeddings

Tip 2: Ensure similar Embedding Space for both Queries and Documents

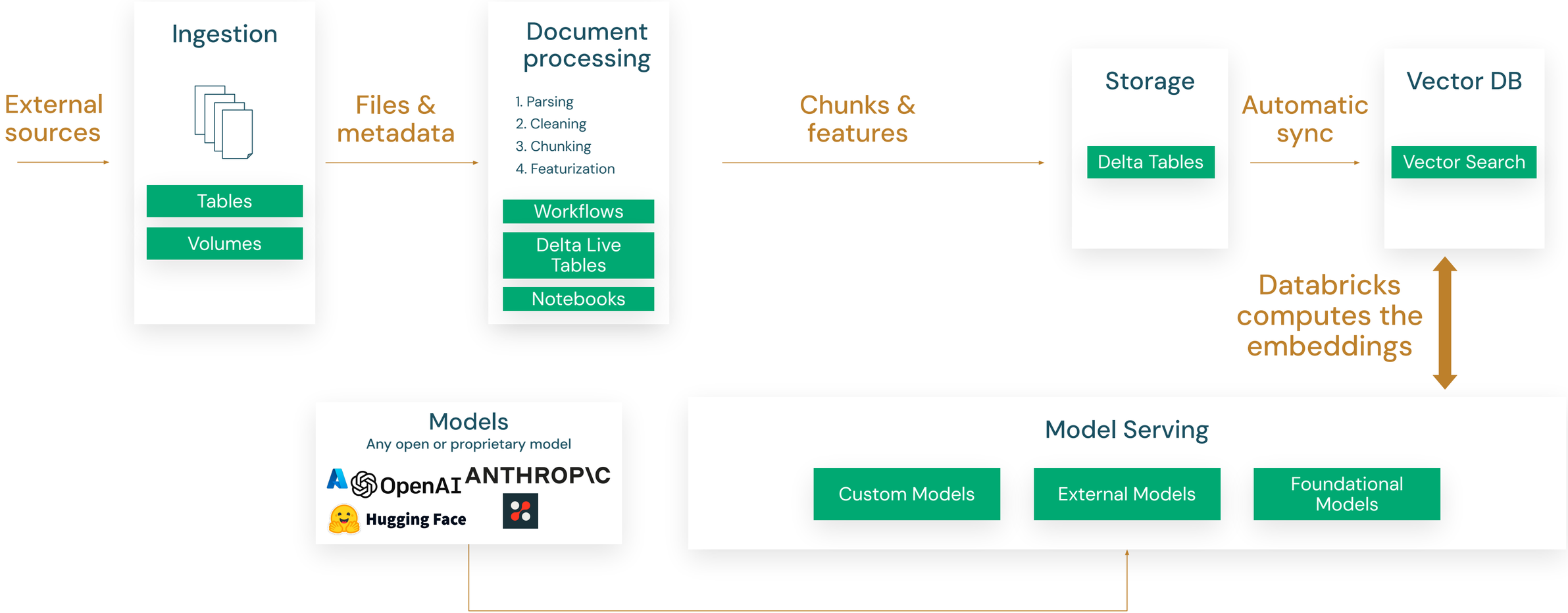
- Use the same embedding model for indexing and querying
- OR – if you use different embedding models, make sure they are trained on similar data (therefore produce the same embedding space!)



Data Preparation in Databricks

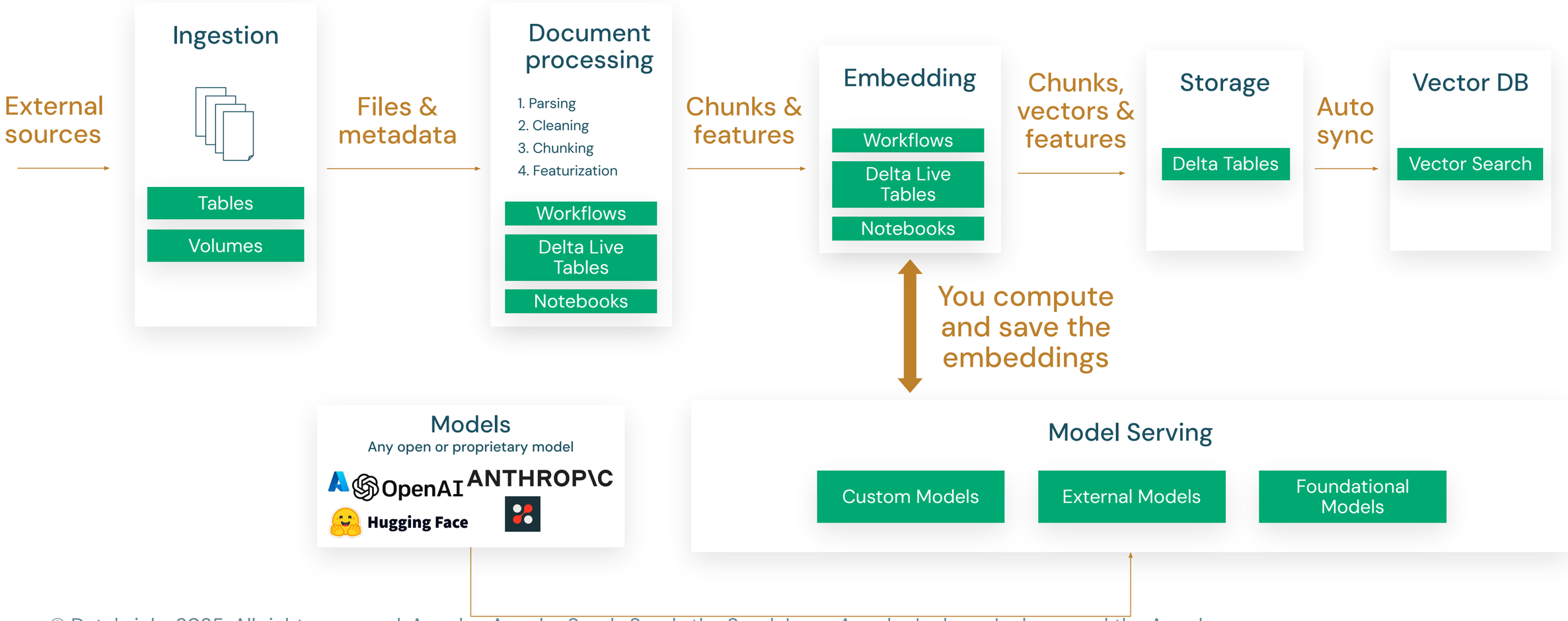
Unstructured Data Prep

Vector Search with Databricks-managed embeddings



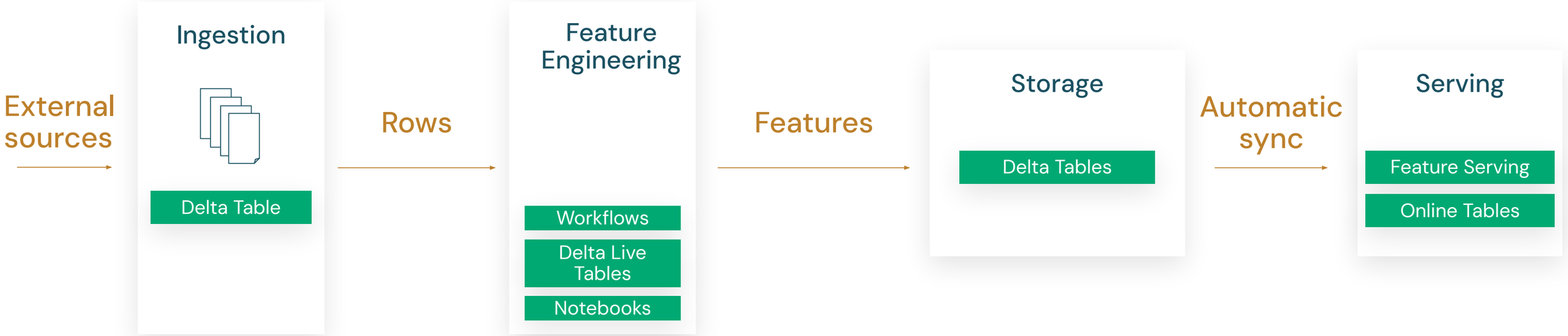
Unstructured Data Prep

Vector Search with user-managed embeddings



Structured Data Prep

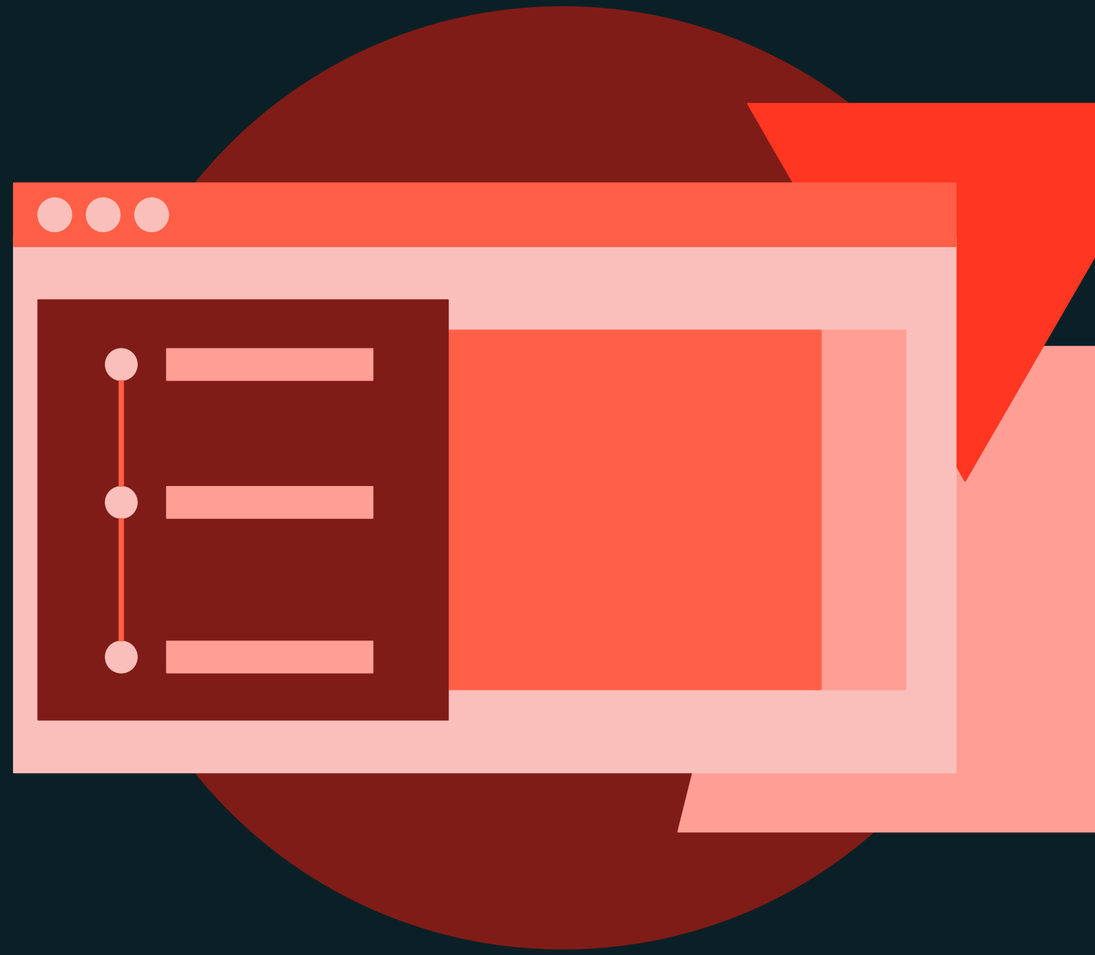
Feature Serving and Online Tables



databricks

DEMONSTRATION

Preparing Data for RAG



Demo Outline

Preparing data for RAG

What we'll cover:

- Extract PDF content as text chunks
- Creating embeddings with foundation model API endpoints
  - How to use a foundation model
  - Compute chunks' embeddings
- Save computed embeddings to a Delta Table

databricks

LAB EXERCISE

Preparing Data for RAG

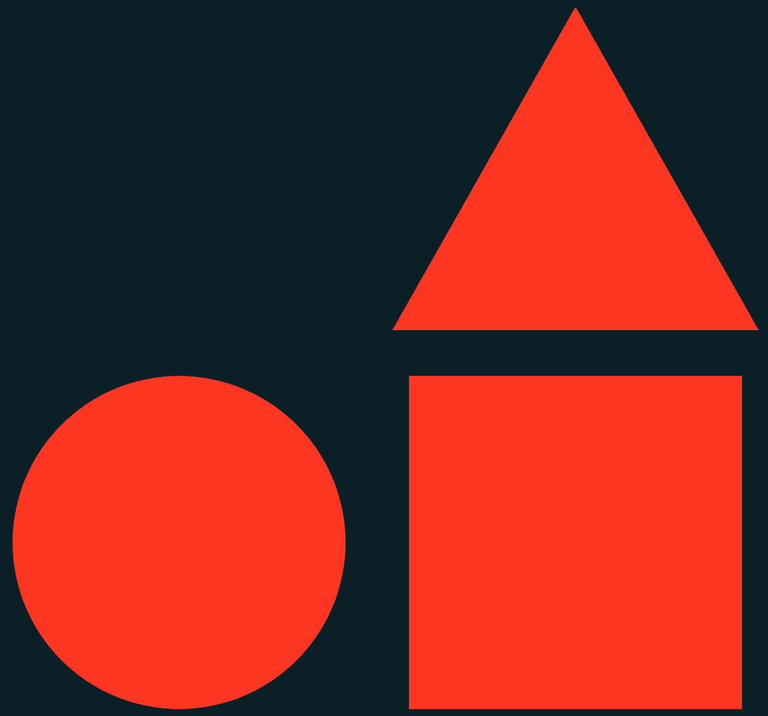
Lab Outline

What you'll do:

- **Task 1**: Read the PDF files
- **Task 2**: Extract text content and load to a dataframe
- **Task 3**: Compute embeddings for text chunks
- **Task 4**: Store text embeddings to a Delta table

databricks

Mosaic AI Vector Search



Generative AI Solution Development

Learning Objectives

- Describe characteristics of an embedding vector and a vector database.
- Identify use cases where vector databases are valuable.
- Understand how vector databases support generative AI applications.
- Describe the process of executing a search using a vector database.
- Describe the features and benefits of Mosaic AI Vector Search.

databricks

LECTURE

Introduction to Vector Stores

What are Vector Databases?

- A vector database is a database that is optimized to **store and retrieve high-dimensional vectors such as embeddings**.
- In RAG architecture **contextual information** is stored in vectors.
- Vector databases are designed for **efficient storage of vectors** utilized in generative AI applications, which rely on identifying documents or images with similarities.
- Vector databases provide a query interface that retrieves vectors most similar to a specified query vector.

Why are Vector Databases So Hot?

Query time and scalability

- Specialized, full-fledged databases for **unstructured data**
  - Inherit database properties, i.e. Create-Read-Update-Delete (CRUD)

- Speed up query search for the closest vectors
  - Uses vector search algorithms such as Approximate Nearest Neighbor (ANN)
  - Organize embeddings into indices

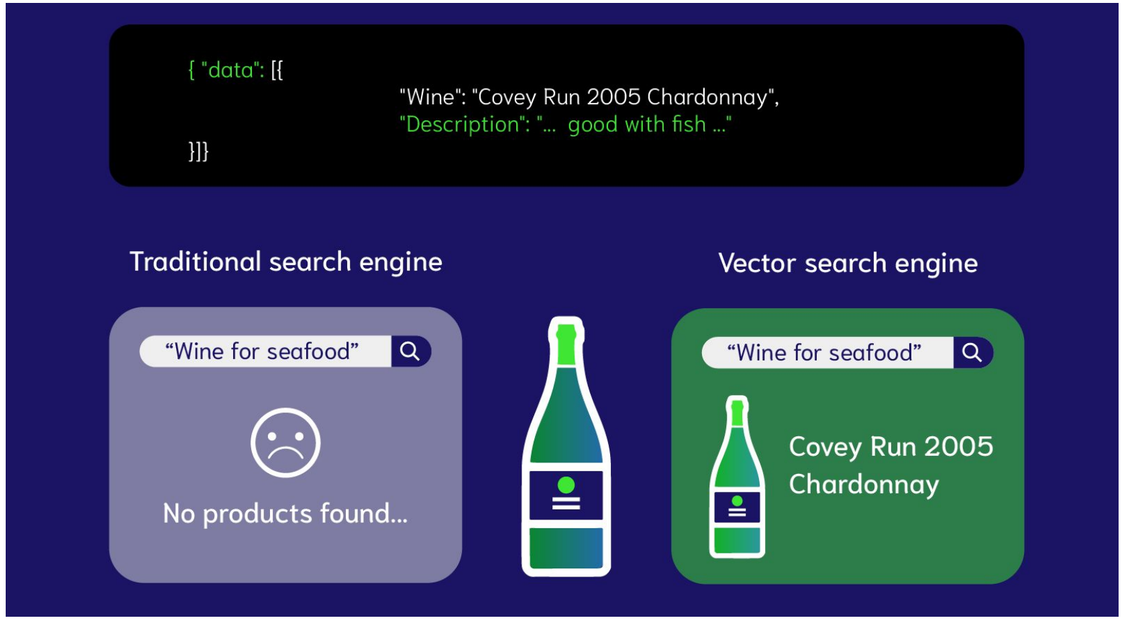


Image Source: Weaviate

Common Use Cases for Vector Databases

RAG

- Delivering relevant unstructured documents to help a RAG application answer user's questions.

Recommendation Engines

- Personalized, context aware recommendations to users.
- More efficient similarity search than traditional approaches. Example; recommending movies with similar genres, actors, or directors to those a user has previously enjoyed.

Similarity Search

- Semantic match, rather than keyword match!
- Enabling plain language search queries that deliver relevant results.
- Text, images, audio.
- Understand similarities and differences between data.

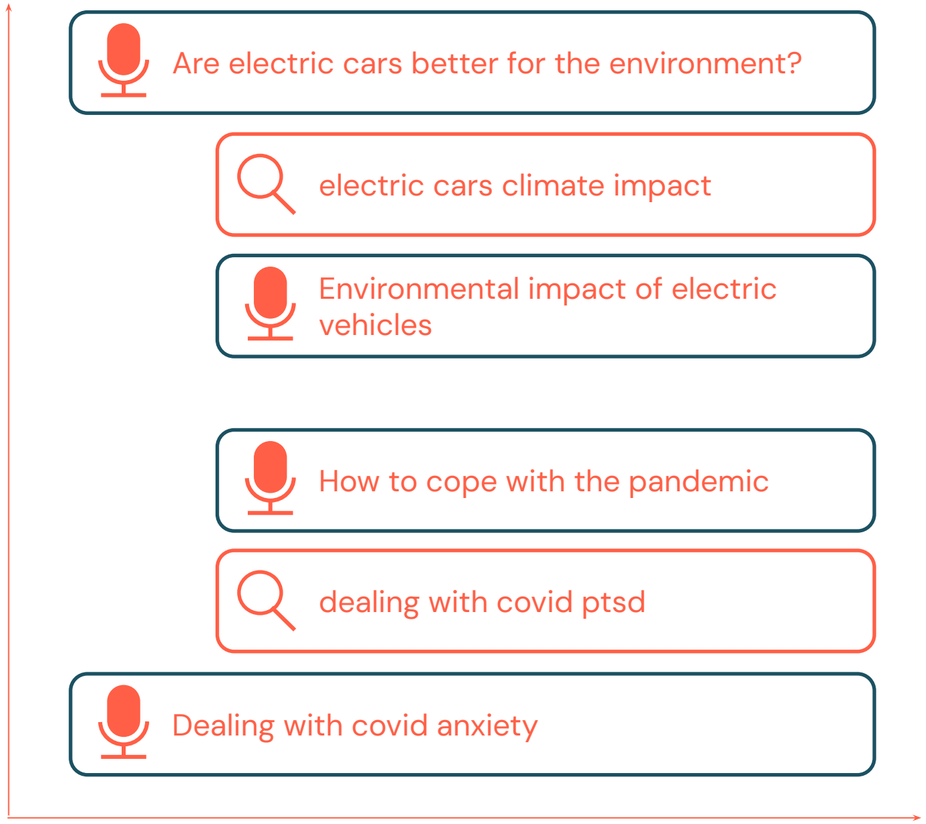
A Sample Use Case

Spotify uses natural language search for their recommendation engine

Problem: Fuzzy matching, normalization, and manual aliases cannot capture all variations of user queries.

Solution:

- Match user queries with content that is semantically correlated instead of exact word matching.
- Used vector search techniques like Approximate Nearest Neighbor (ANN).



Shared embedding space for queries and podcast episodes

Source: Spotify

Vector Search Process and Performance

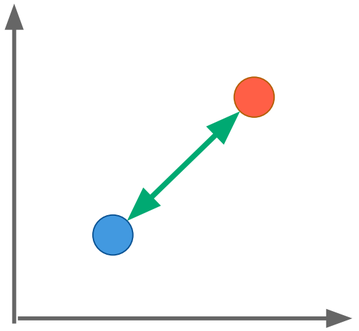
Vector Similarity

How to measure if 2 vectors are similar?

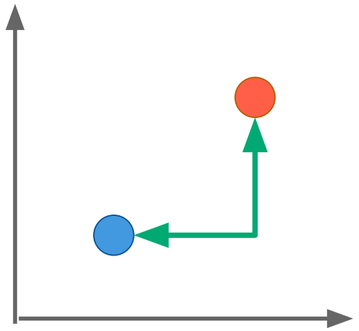
Distance Metrics

- The **higher** the metric, the **less similar**
- L2 (Euclidean) is the most popular

Euclidean (L2) Distance

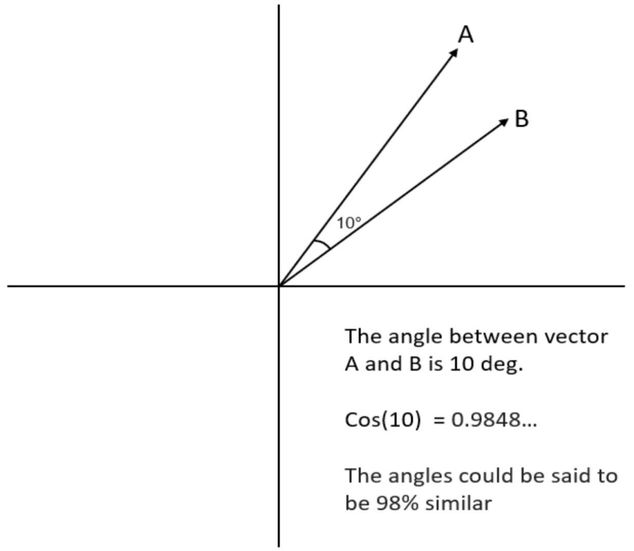


Manhattan (L1) Distance



Similarity Metrics

- The **higher** the metric, the *more similar*
- Cosine similarity is the most popular

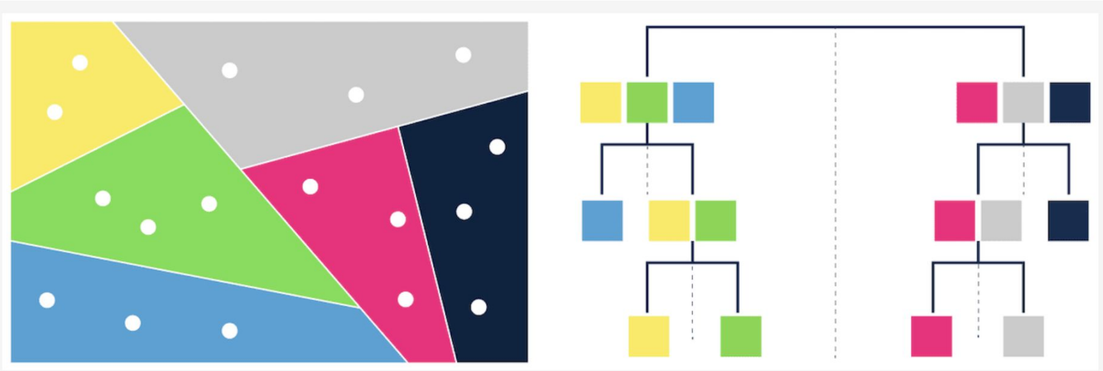


Source: buildin.com

Vector Search Strategies

- K-nearest neighbors (KNN)

- Approximate nearest neighbors (ANN)
  - Trade accuracy for speed gains
  - Examples of indexing algorithms:
    - Tree-based: ANNOY by Spotify
    - Proximity graphs: HNSW
    - Clustering: FAISS by Facebook
    - Hashing: LSH
    - Vector compression: SCaNN by Google, Product Quantization (PQ)



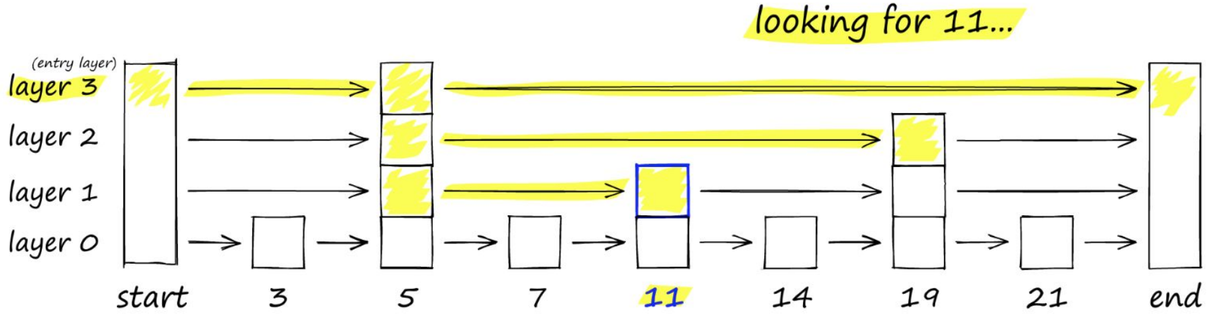
[Figure 3 - Tree-based ANN search]

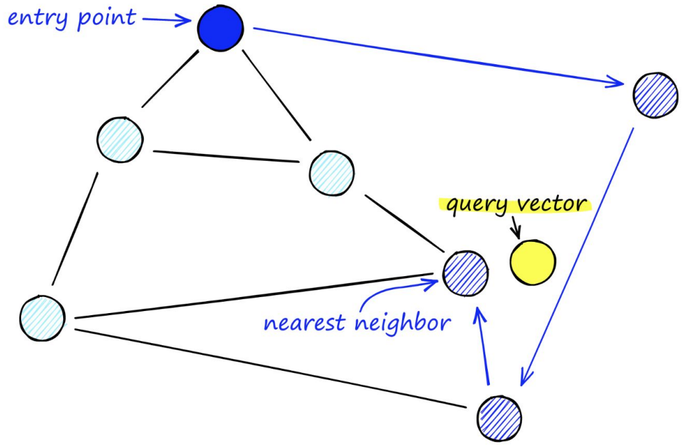
Source: Weaviate

HNSW: Hierarchical Navigable Small Worlds

Builds proximity graphs based on Euclidean (L2) distance

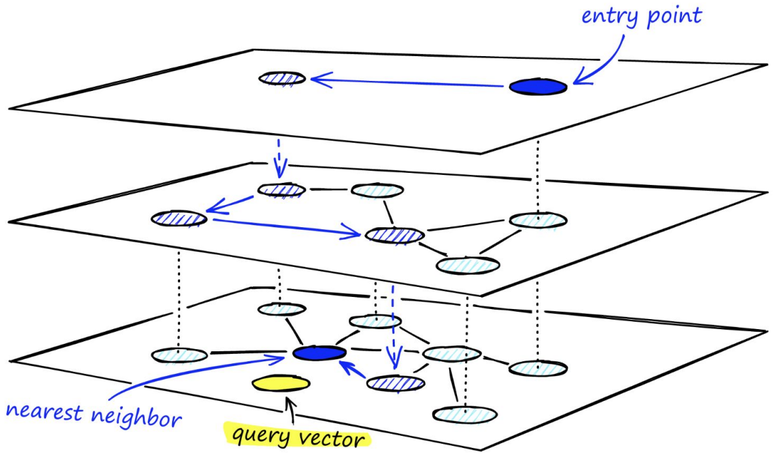
Uses linked list to find the element x: “11”





Traverses from query vector node to find the neare

- What happens if too many nodes?
  Use hierarchy!

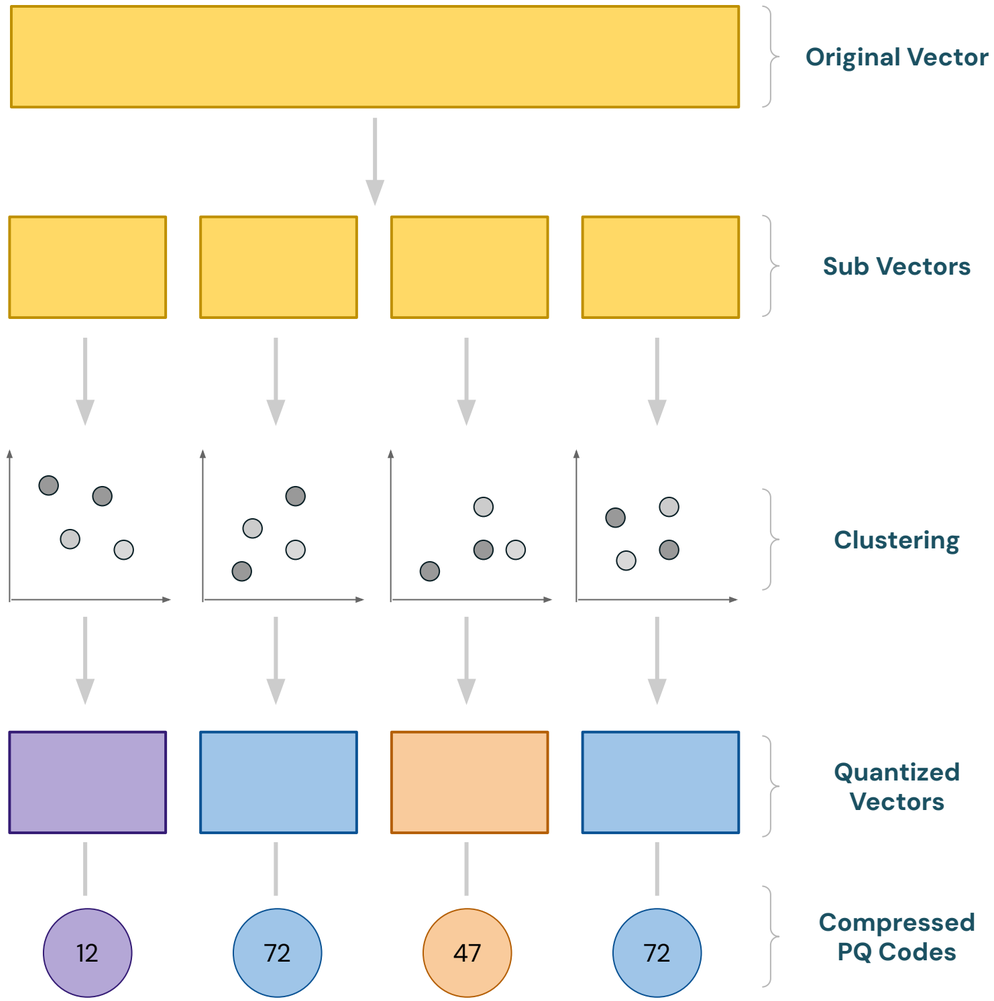


Source: Pinecone

Product Quantization (PQ)

Definition

- Product Quantization (PQ) is a technique used to **compress high-dimensional vectors** and perform efficient approximate nearest neighbor (ANN) searches.
- Useful in systems where memory usage and search speed are critical.



Product Quantization (PQ)

What is its purpose?

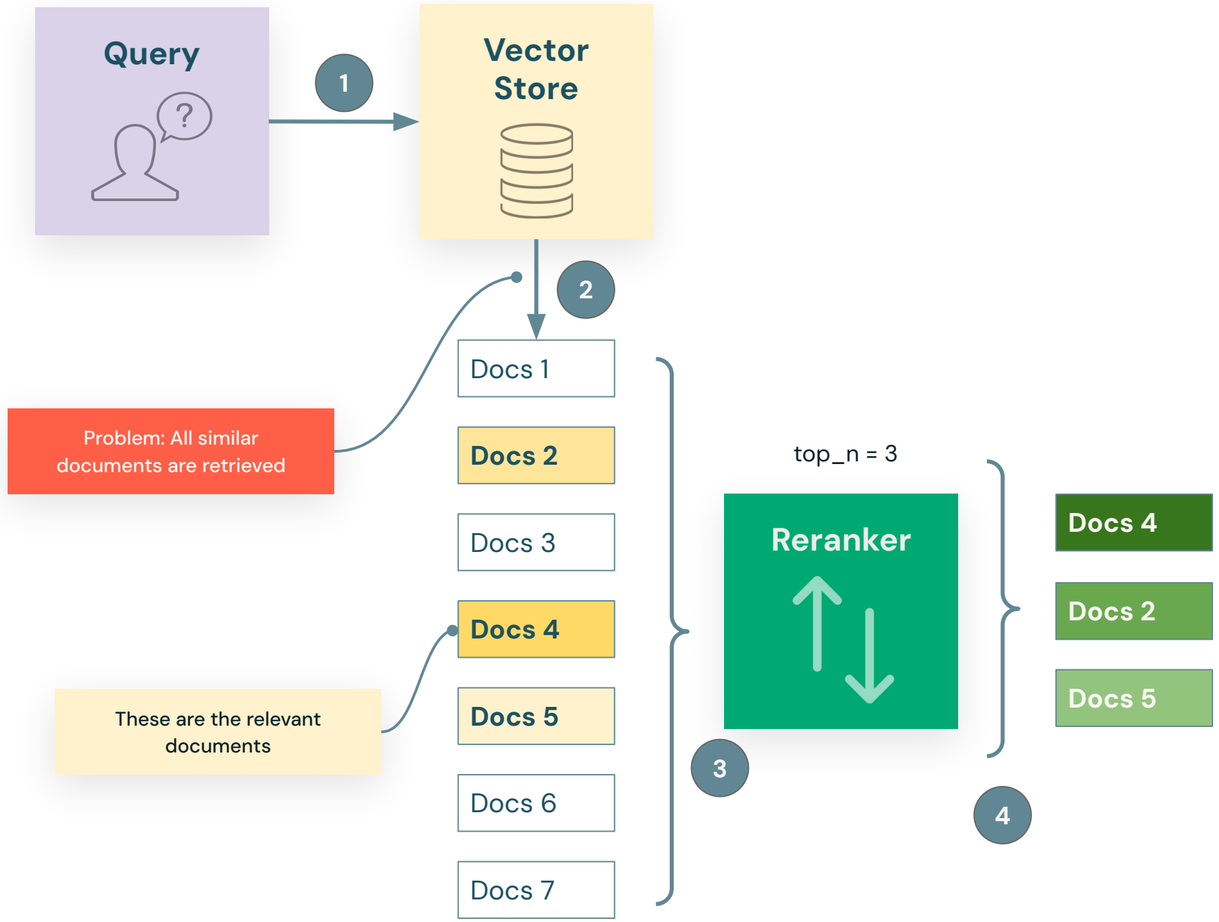
- **Vector compression**:
  - Convert vectors to compact codes which reduces memory footprint.
  - Instead of storing the full detail of every vector, centroid of the clusters are indexed.
- **Efficient Similarity Search**:
  - The compressed vectors (PQ codes) enable efficient similarity searches.
  - Input query doesn’t compared with all vectores. Instead, it is compared with centroids of each cluster (the compressed representations instead of the full high-dimensional vectors).
- **Scalability**:
  - Suitable for dynamic databases where new data is continuously added.

How to Filter Only Highly Relevant Documents?

Reranking

A method of prioritizing documents most relevant to user's query

- **Initial retrieval**
  - Not all documents are equally important.
  - We should use only the relevant documents.
- **Reranker**
  - Reorder documents based on the relevance scores.
  - The goal is to **place most relevant documents at the top** of the list.

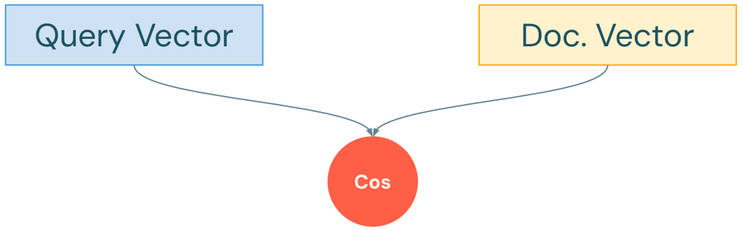


Reranking

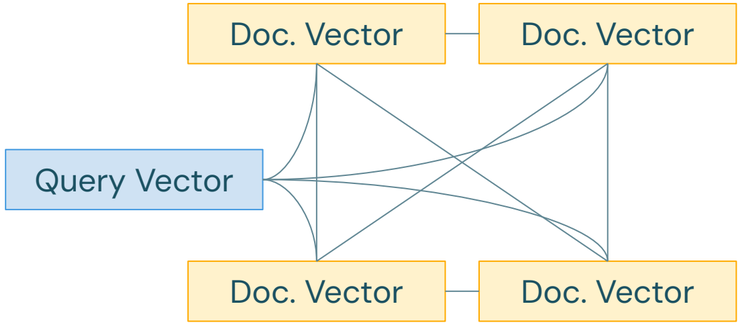
A method of prioritizing documents most relevant to user's query

- Reranking adjusts the initial ranking of retrieved documents to **enhance the precision and relevance** of search results.
- Reranking supports **deeper semantic understanding** based on documents' actual relevance to the query.
- Rerankers:
  - Private APIs: Cohere ReRank, Jina Rerank
  - Open-source: Cross-encoders, bge-reranker-base, **FlashRank**

Representation-based Comparison:



Reranker



Using Reranking

Benefits and challenges

Benefits

- Select more **relevant documents**.
- Improve the **accuracy** of the response.
- Reduce hallucinations.

Challenges

- The LLM must be called repeatedly, increasing the **cost and latency** of the RAG chain.
- Implementing rerankers adds **complexity** to the RAG pipeline.

databricks

LECTURE

Introduction to Mosaic AI Vector Search

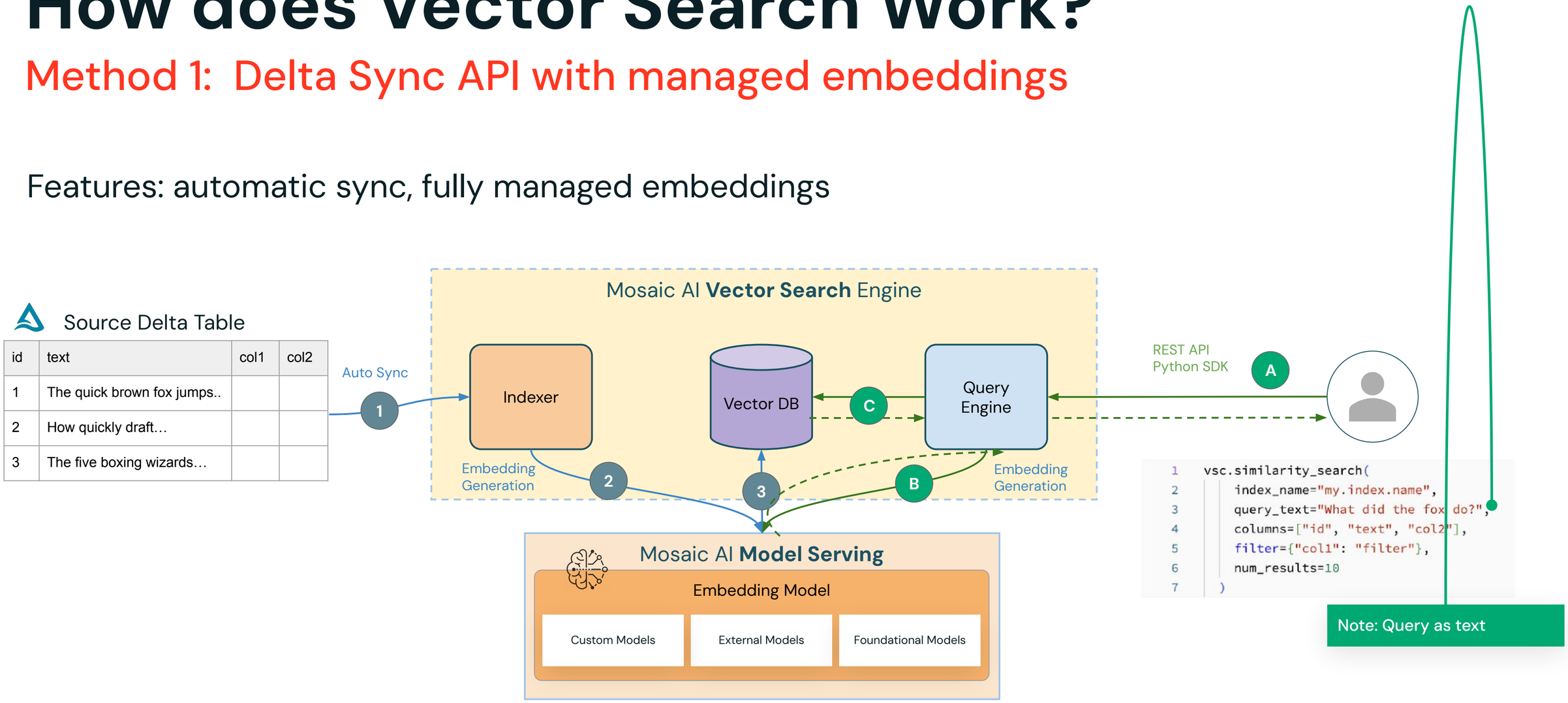
Mosaic AI Vector Search

- Stores vector representation of your data, plus metadata
- Tightly integrated with your Lakehouse
- Scalable, low latency production service with zero operational overhead
- Supports ACLs using Unity Catalog integration
- API for real-time similarity search
  - Query can include filters on metadata
  - REST API and Python client

How does Vector Search Work?

Method 1: Delta Sync API with managed embeddings

Features: automatic sync, fully managed embeddings

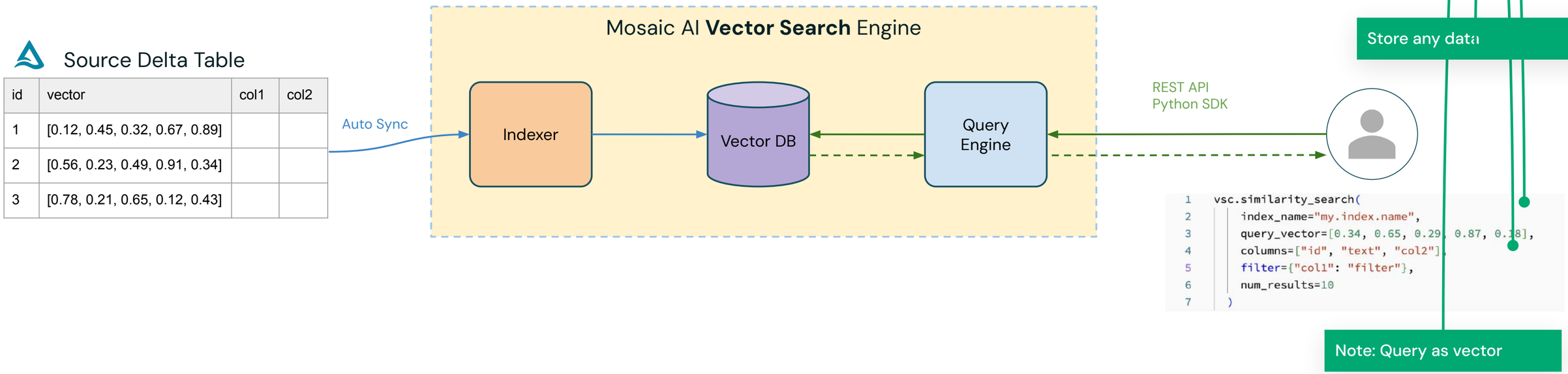


How does Vector Search Work?

Method 2: Delta Sync API with self-managed embeddings

Features:

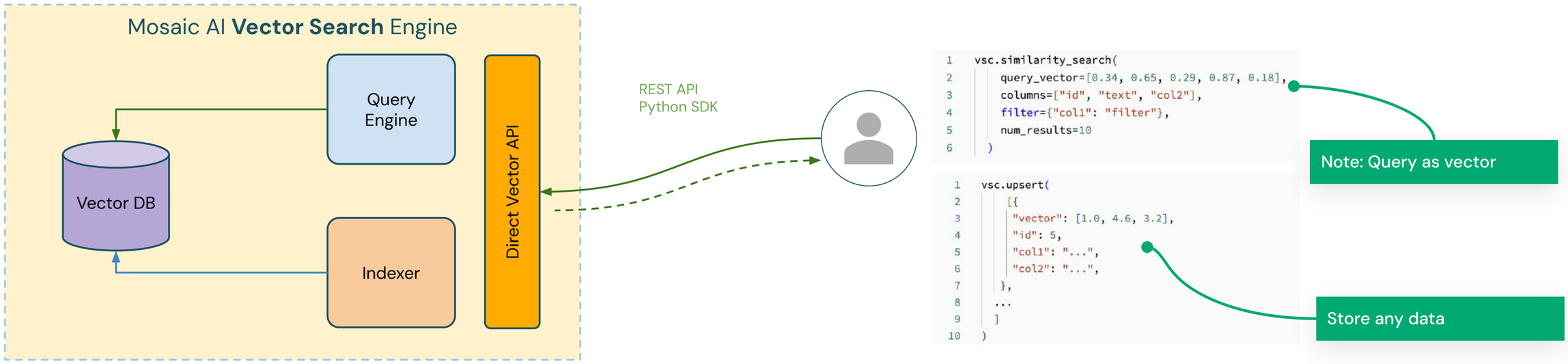
- automatic sync
- self-managed embeddings



How does Vector Search Work?

Method 3: Direct access CRUD API

Features: manual sync via API, with self-managed embeddings



Set up Vector Search

Key concepts/components for Vector Search

1. Create a Vector Search Endpoint

- This is the **compute resource** associated with vector search.
- Endpoints scale automatically to support the size of the index or the number of concurrent requests.
- Support for multiple compute types.

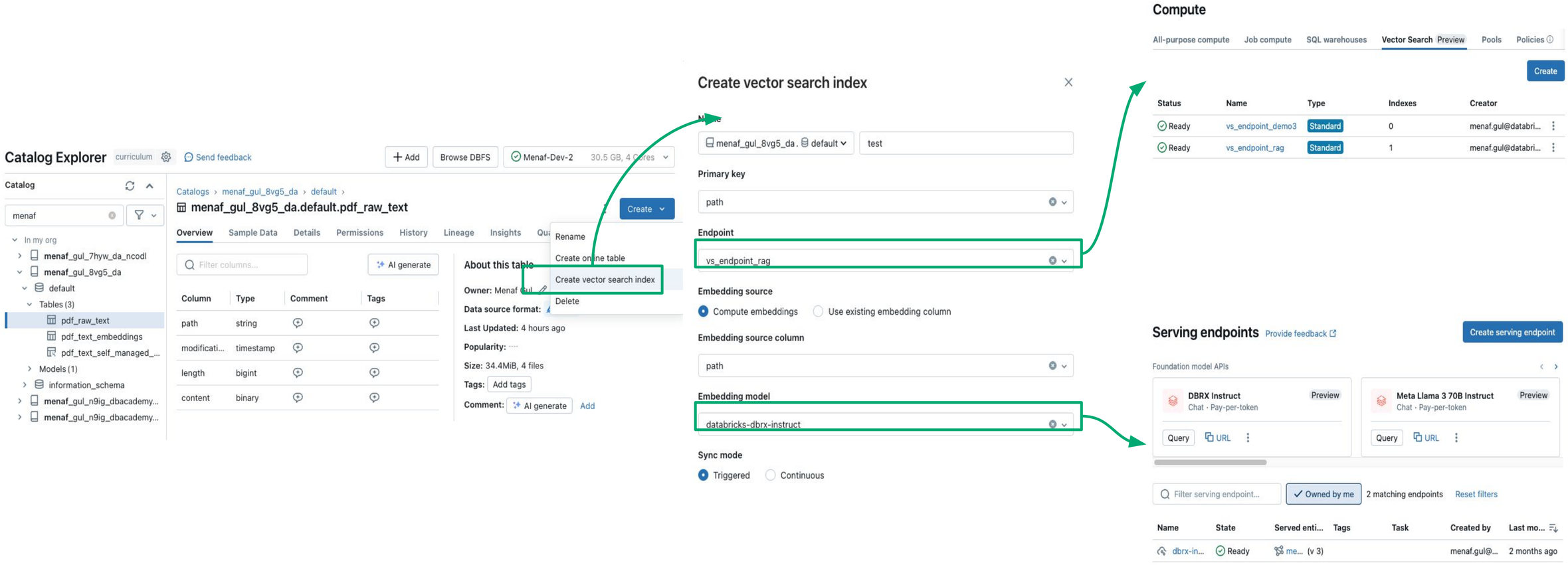
2. Create a Model Serving Endpoint

- Create if you choose to have Databricks compute the embeddings.
- Model Serving supports embeddings via **Foundation Models APIs** (e.g., BGE), **external models** (e.g., OpenAI's ada-002), and **custom models**.

3. Create a Vector Search Index

- Created and **auto-synced** from a Delta table.
- Optimized to provide real-time approximate nearest neighbor searches.
- Indexes appear in and are **governed by Unity Catalog**.
- Index level ACLs.

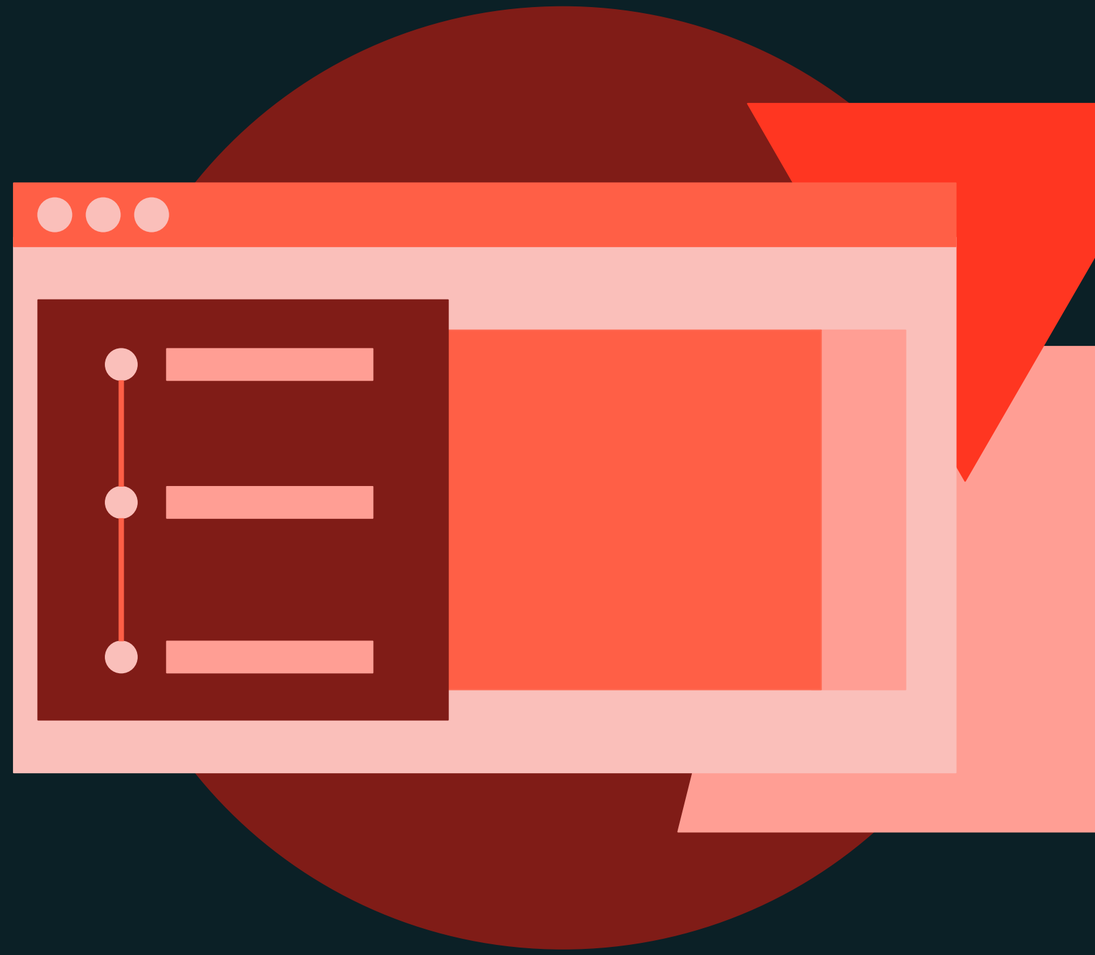
Set up Vector Search



databricks

DEMONSTRATION

Create Self-managed Vector Search Index



Demo Outline

Create Vector Search Index

What we'll cover:

- Demo overview
- Create a Vector Search index
  - Setup a Vector Search endpoint
  - View the endpoint
  - Connect Delta table with the Vector Search index
- Search for similar content using Vector Search
- Re-ranking search results

databricks

LAB EXERCISE

Create Managed Vector Search Index

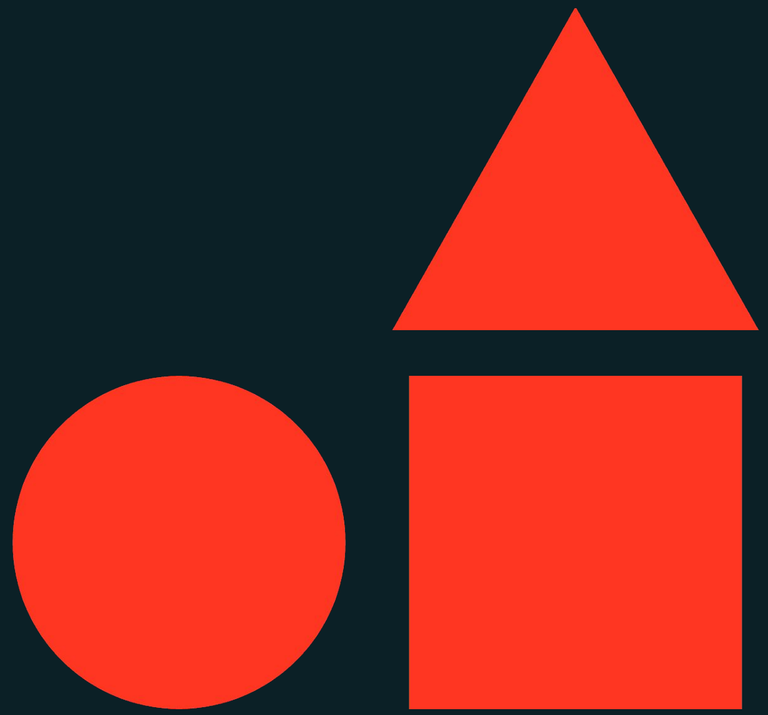
Lab Outline

What you'll do:

- **Task 1**: Create a vector search endpoint
- **Task 2**: Create a managed vector search index
- **Task 3**: Search documents similar to the query
- **Task 4**: Re-rank search results

databricks

Assembling a RAG Application



Generative AI Solution Development

Learning Objectives

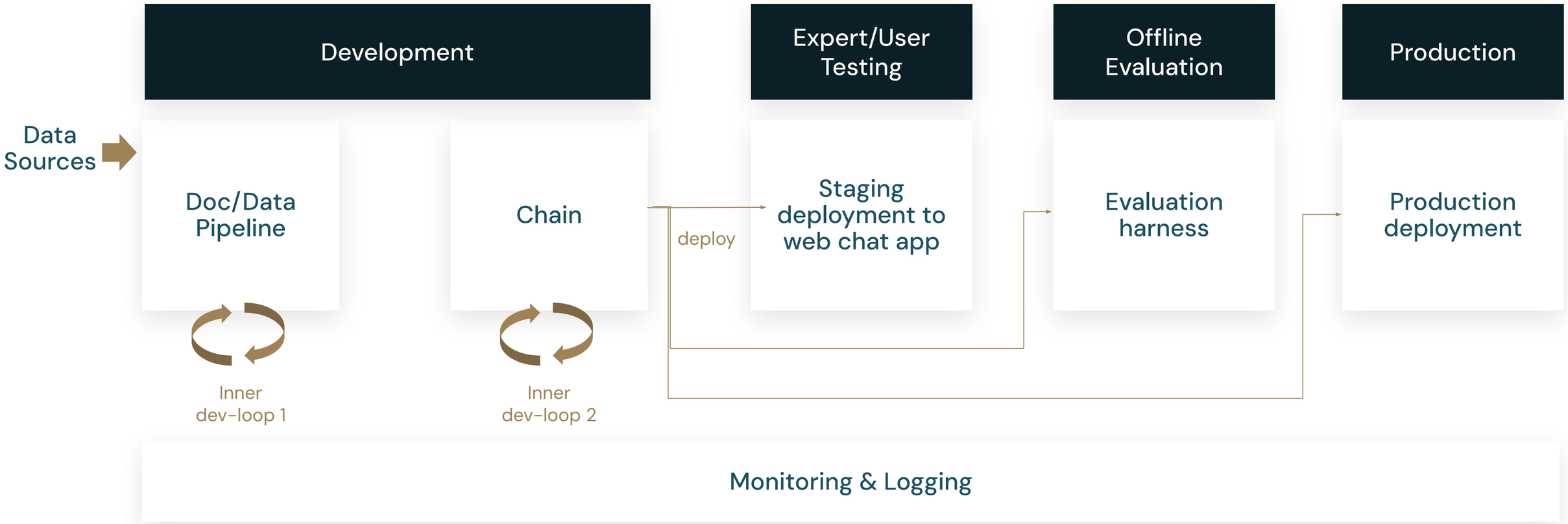
- Describe how RAG components are assembled into a single chain
- Describe how MLflow's LLM supports enable RAG workflows on Databricks

databricks

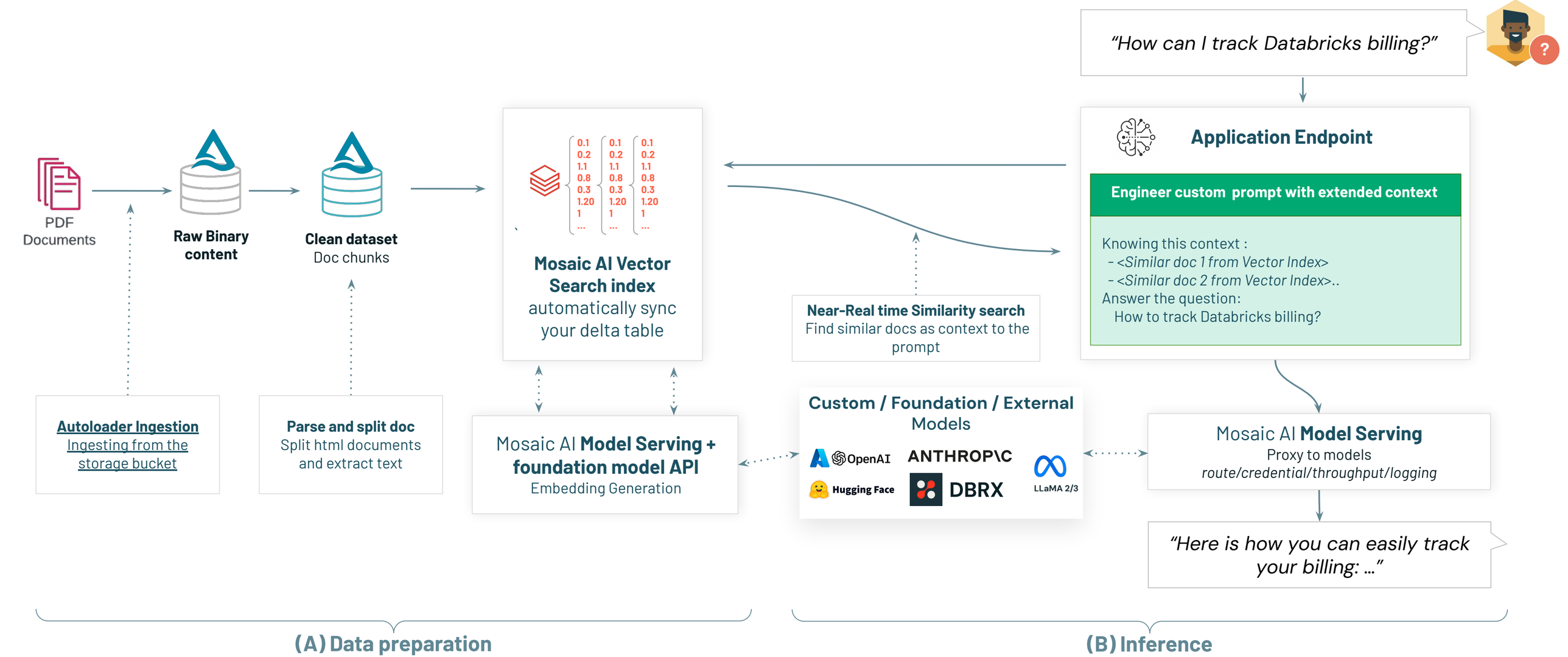
LECTURE

MLflow for RAG Applications

RAG Application Workflow



Assembling a RAG Application



MLflow

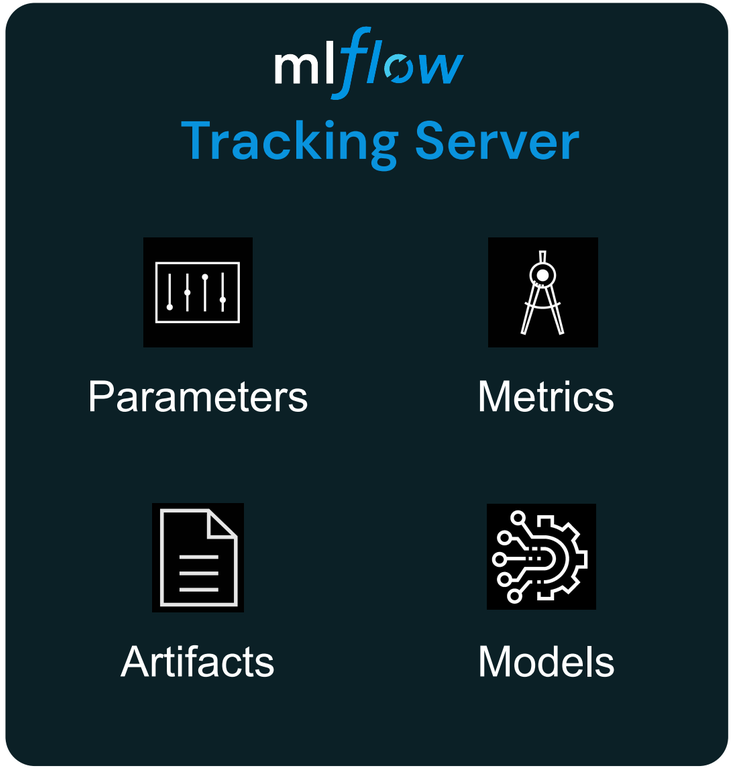
Using MLflow for RAG solutions

- **Open-source** platform for machine learning lifecycle
- Co-developed by Databricks and the ML community
- Pre-installed on the Databricks Runtime for ML
- Operationalizing Generative AI development lifecycle

MLflow Model Tracking

Make your GenAI workflow more manageable and transparent

- Record LLM **parameters** such as temperature and model configurations.
- Log **metrics** and compare them to get insights about the performance and accuracy LLMs.
- Store and manage output **artifacts** such as visualization images and serialized models.
- Store model's **source code** from the run.



MLflow – Model ("Flavor")

Make your GenAI workflow more manageable and transparent

- Each **MLflow Model** is a directory containing arbitrary files, together with an `MLModel` file.
- `MLModel` file can define **multiple flavors** that the model can be viewed in.
- With MLflow Models deployment tools can understand the model.
- Model file can contain **additional metadata** such as `signature`, `input example` etc.

mlflow LangChain Flavor

```
# Directory written by
mlflow.langchain.log_model(model, "chain",...)
chain/
├──model
    └── steps
    └── steps.yaml
├── MLmodel
├── lc_model.py
├── conda.yaml
├── python_env.yaml
├── requirements.txt
```

```
# MLModelfile
artifact_path: chain
flavors:
  langchain:
    code: null
    langchain_version: 0.1.5
    model_data: model
  python_function:
    loader_module: mlflow.langchain
...
```

MLflow – Model ("Flavor")

Built-in model flavors

Python Function (`mlflow.pyfunc`):

- Serves as a **default model interface** for MLflow Python models.
- Any MLflow Python model is expected to be loadable as a python function.
- Allows you to deploy models as Python functions.
- It includes all the information necessary to **load and use a model**.
- Some functions: **`log_model`, `save_model`, `load_model`, `predict`**

mlflow Model Flavors

Example built-in model flavors:

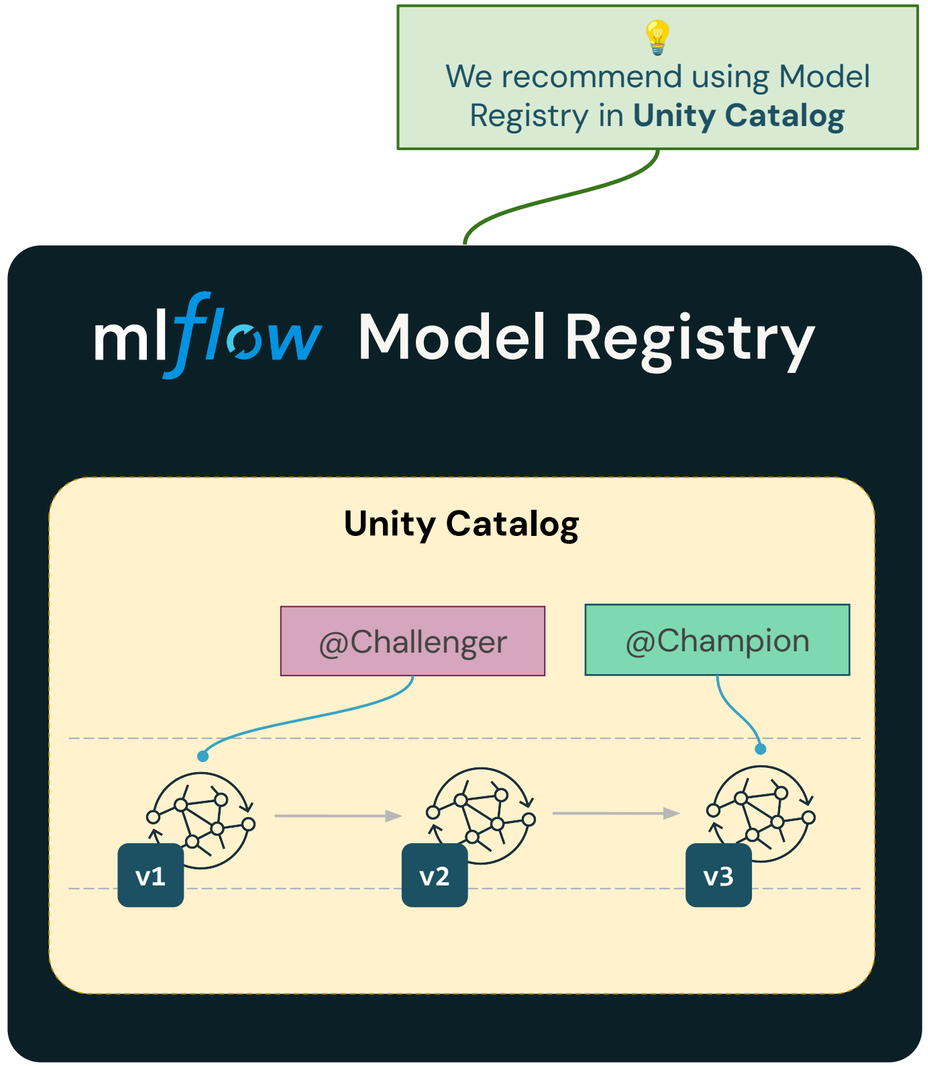
- LangChain
- OpenAI
- HuggingFace
- PyTorch
- TensorFlow
- ONNX
- **Python function**

...

MLflow Model Registry

A centralized model store

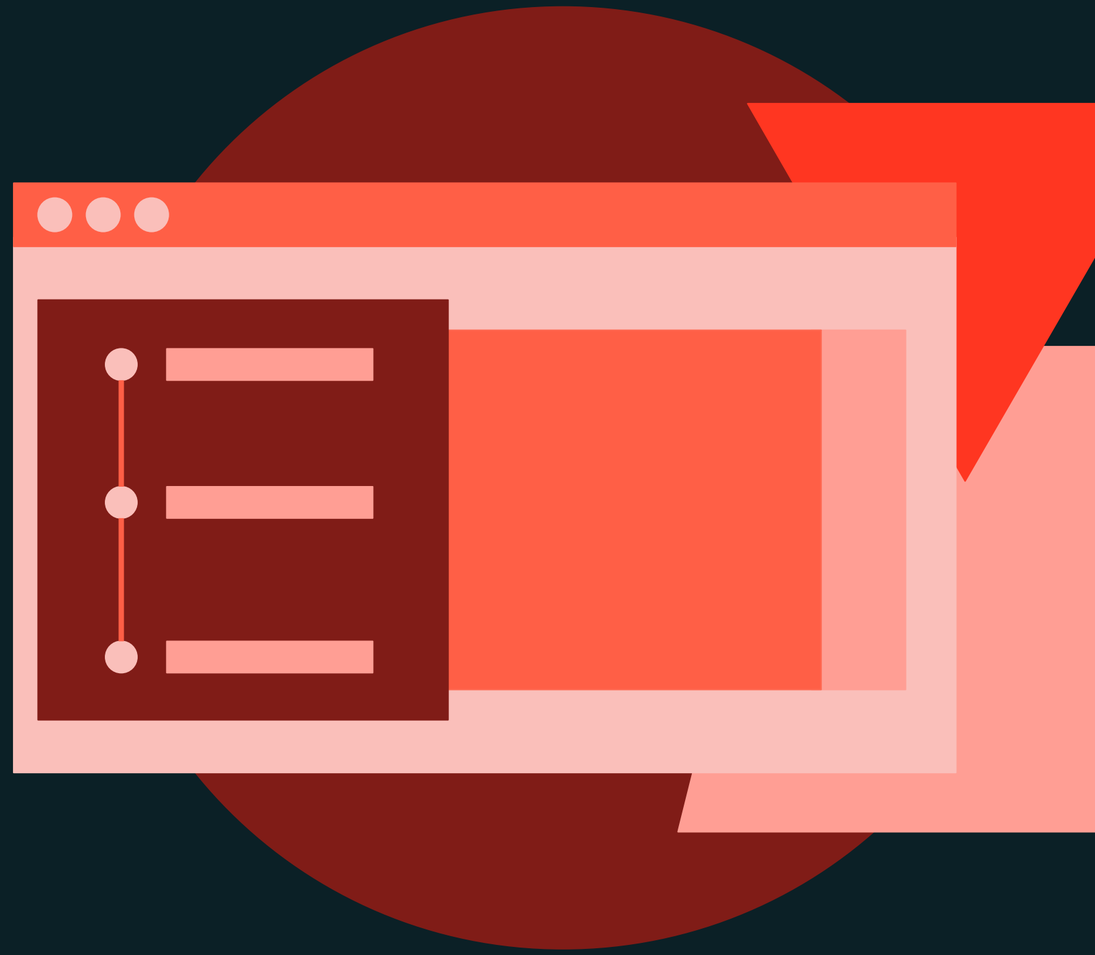
- Deploy and organize models.
- Model **versioning.**
- Model lifecycle management with **aliases**. Example: **`champion`** for the **`Production`** stage.
- Collaboration and permission management.
- Full model lineage.
- Tagging and annotations.



databricks

DEMONSTRATION

Assembling a RAG Application



Demo Outline

Assembling a RAG Application

What we'll cover:

- Set up the RAG components
  - Setup the retriever
  - Setup the foundation model
- Assembling the complete RAG pipeline
- Evaluating the RAG pipeline
- Save the model to Model Registry in Unity Catalog

databricks

LAB EXERCISE

Assembling a RAG Application

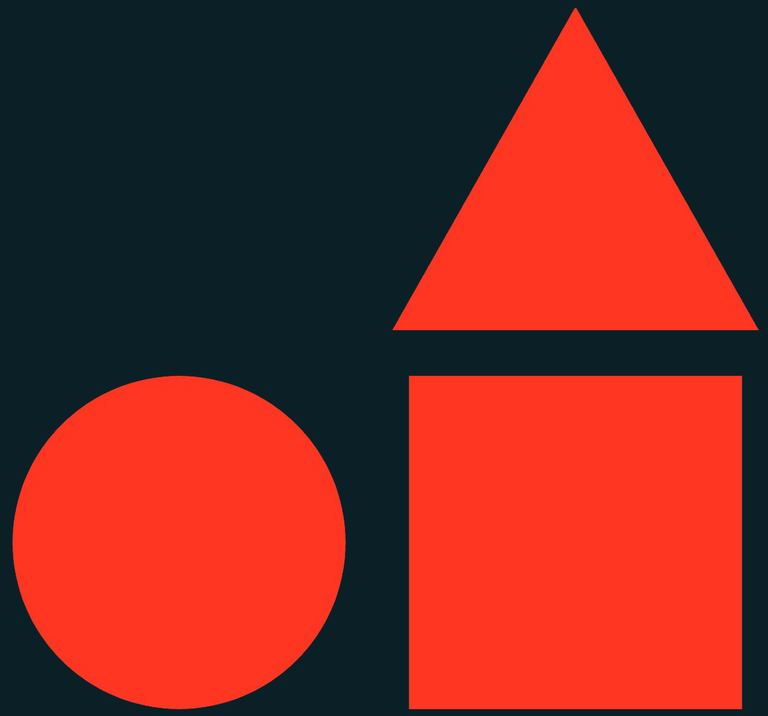
Lab Outline

What you'll do:

- **Task 1**: Setup the retrieval component
- **Task 2**: Setup the foundation model
- **Task 3**: Assemble the complete RAG pipeline
- **Task 4**: Save the model to model registry in Unity Catalog

databricks

Course Summary and Next Steps



Generative AI Solution Development

databricks

LECTURE

Evaluating a RAG Application and Continual Learning

Evaluating RAG Pipeline

RAG is complex

- When evaluating RAG solutions, we need to **evaluate each component separately and together**.
- Components to evaluate
  - Chunking: method, size
  - Embedding model
  - Vector store
    - Retrieval and re-ranker
  - Generator

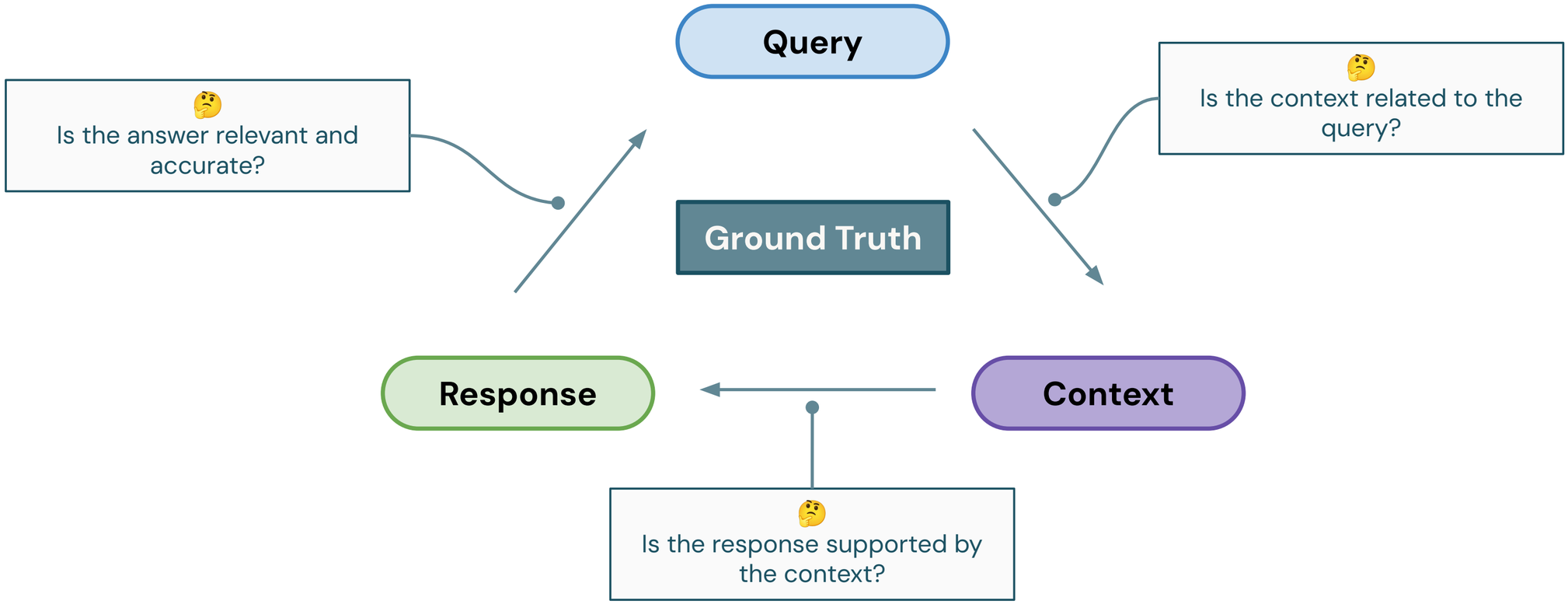
Chunking Performance

Retrieval Performance

Generator Performance

Evaluation Metrics

Retrieval and generation related metrics

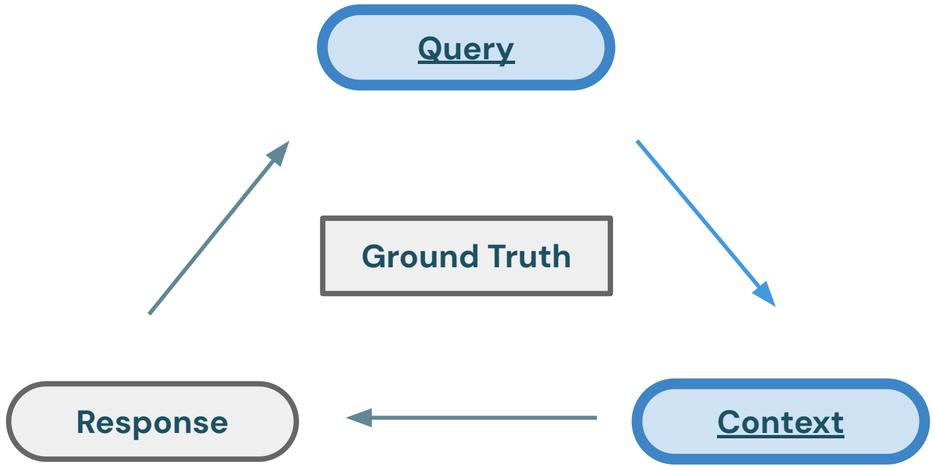


Context Precision

Retrieval related metrics

Context Precision:

- Signal-to-noise ratio for the retrieved context.
- Based on **Query** and **Context(s)**.
- It assesses whether the chunks/nodes in the retrieval context ranked higher than irrelevant ones.



Example:

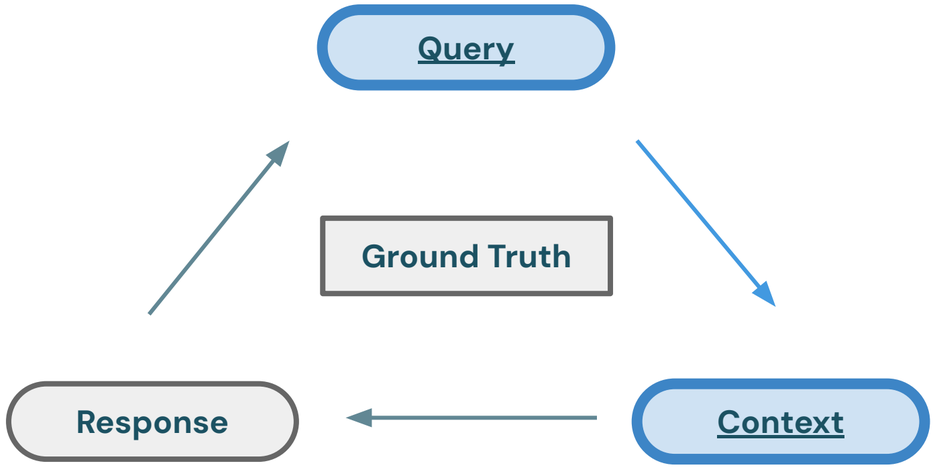
- **Query:** What was Einstein's role in the development of quantum mechanics?
- **Ground Truth:** Einstein contributed to the development of quantum theory, including his early skepticism and later contributions to quantum mechanics.
- **High Context Precision:** [The contexts specifically mention Einstein's contributions to quantum theory]
- **Low Context Precision:** [The contexts broadly discuss Einstein's life and achievements without specifically addressing his contributions to quantum mechanics.]

Context Relevancy

Retrieval related metrics

Context Relevancy:

- Measure the relevancy of the retrieved context.
- Based on both the **Query** and **Context(s)**.
- It does not necessarily consider the factual accuracy but focuses on how well the answer addresses the posed question



Example:

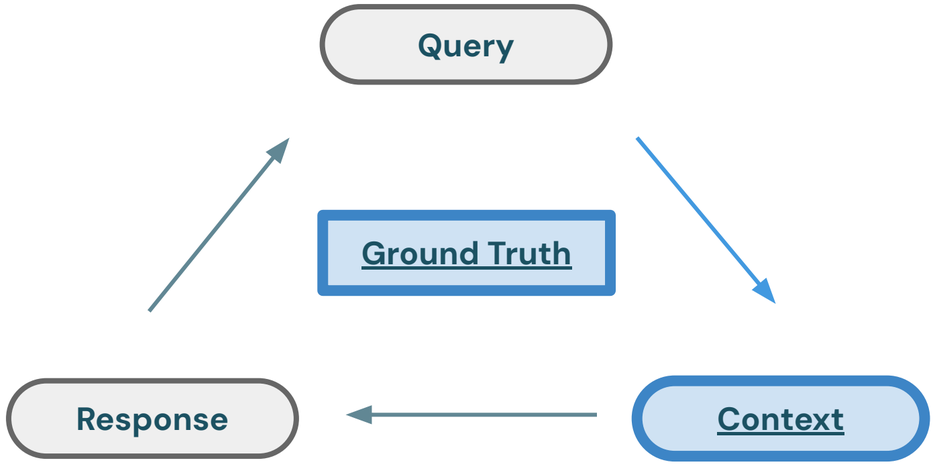
- **Query:** What was Einstein's role in the development of quantum mechanics?
- **High context relevancy:** Einstein initially challenged the quantum theory but later contributed foundational ideas to quantum mechanics.
- **Low context relevancy:** Einstein was known for his pacifist views during the early 20th century and became a U.S. citizen in 1940.

Context Recall

Retrieval related metrics

Context Recall:

- Measures the extent to which all relevant entities and information are retrieved and mentioned in the context provided.
- Based on **Ground Truth** and retrieved **Context(s)**.



Example:

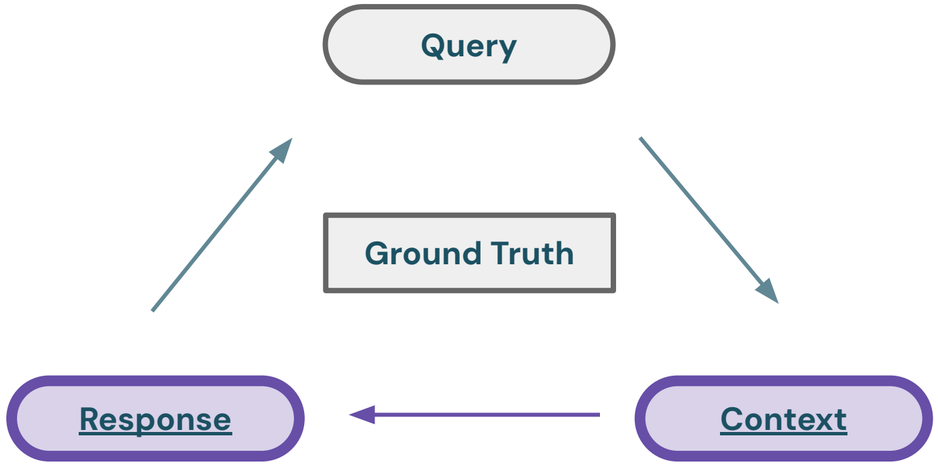
- **Query:** What significant scientific theories did Einstein contribute to?
- **Ground truth:** Einstein contributed to the theories of relativity and had insights into quantum mechanics.
- **High-recall context:** Einstein contributed to relativity and quantum mechanics.
- **Low-recall context:** Einstein contributed to relativity.

Faithfulness

Generation related metrics

Faithfulness:

- Measures the factual accuracy of the generated answer in relation to the provided context.
- Based on the **Response** and **retrieved Context(s)**.



Example:

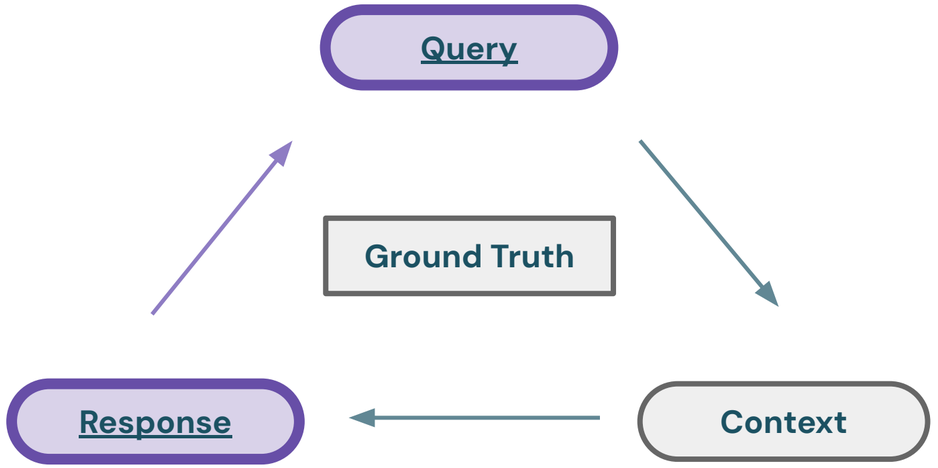
- **Query:** Where and when was Einstein born?
- **Context:** Albert Einstein (born 14 March 1879) was a German-born theoretical physicist, widely held to be one of the greatest and most influential scientists of all time.
- **High faithfulness answer:** Einstein was born in *Germany on 14th March 1879*.
- **Low faithfulness answer:** Einstein was born in *Germany on 20th March 1879*.

Answer Relevancy

Generation related metrics

Answer Relevancy:

- Assesses how pertinent and applicable the generated response is to the user's initial query.
- Based on the alignment of the **Response** with the user's intent or **Query** specifics.



Example:

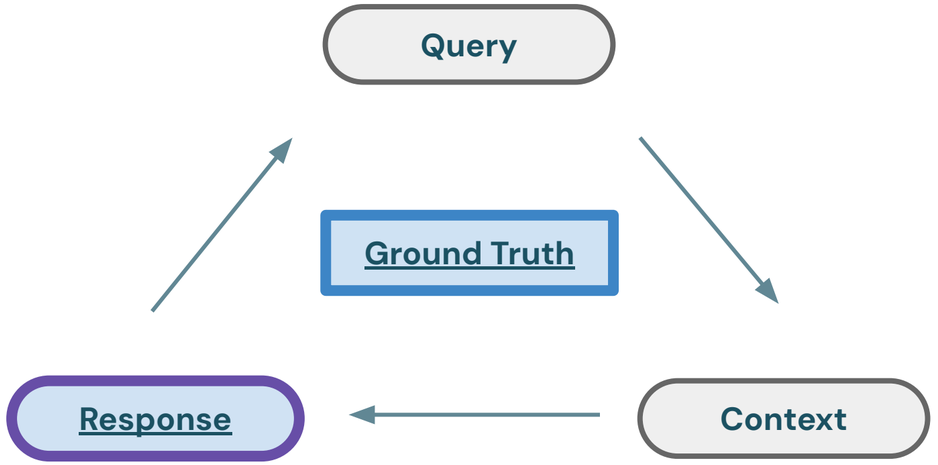
- **Query:** What is Einstein known for?
- **High relevancy answer:** Einstein is known for developing the theory of relativity.
- **Low relevancy answer:** Einstein was a scientist.

Answer Correctness

Generation related metrics

Answer Correctness:

- Measures the accuracy of the generated answer when compared to the ground truth.
- Based on the **Ground Truth** and the **Response**.
- Encompasses both semantic and factual similarity with the ground truth.



Example:

- **Ground truth:** Albert Einstein was awarded the Nobel Prize in Physics in 1921 for his explanation of the photoelectric effect.
- **High answer correctness:** Einstein received the Nobel Prize in Physics in 1921 for his work on the photoelectric effect.
- **Low answer correctness:** Einstein won the Nobel Prize in Physics in the 1930s for his theory of relativity.

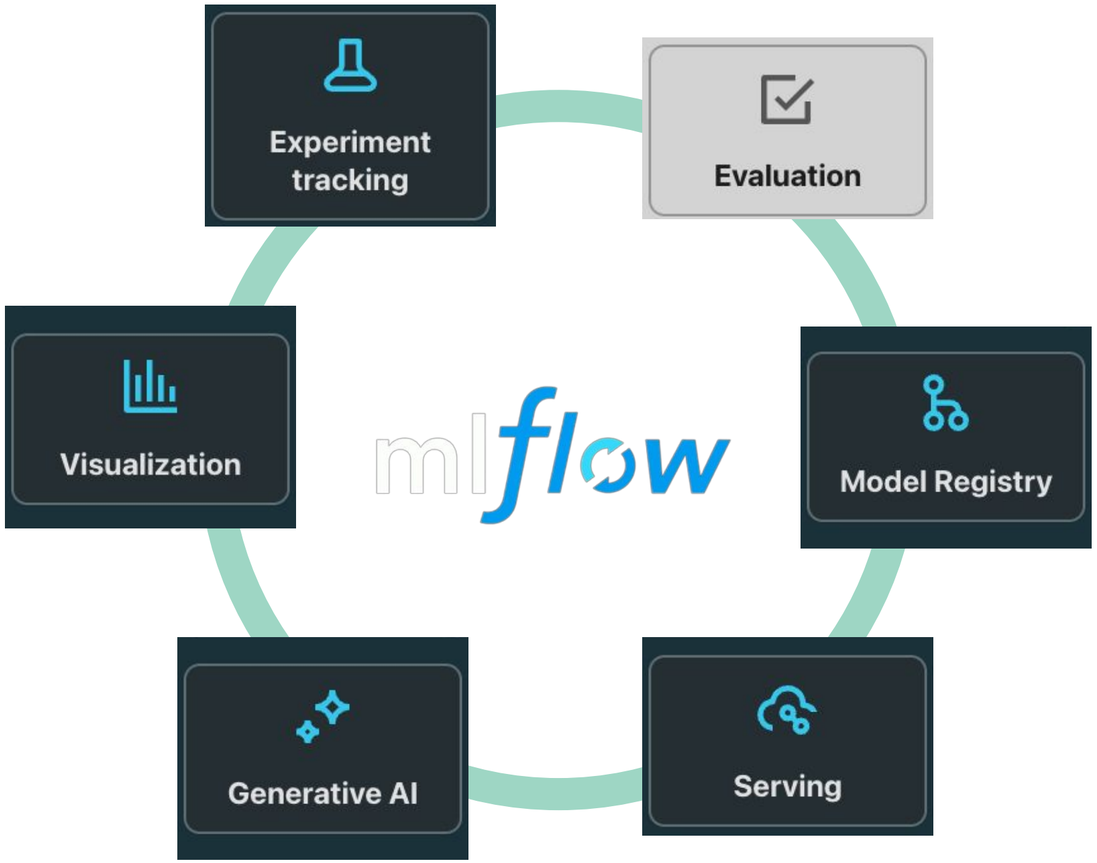
MLflow (LLM) Evaluation

Efficiently evaluate retrievers and LLMs

Batch comparisons:
Compare Foundational Models with fine-tuned models on many questions

Rapid and scalable experimentation:
MLflow can evaluate unstructured outputs automatically, rapidly, and at low-cost.

Cost-Effective:
Automating evaluations with LLMs, can save time on human evaluation



MLflow (LLM) Evaluation

Efficiently evaluate retrievers and LLMs

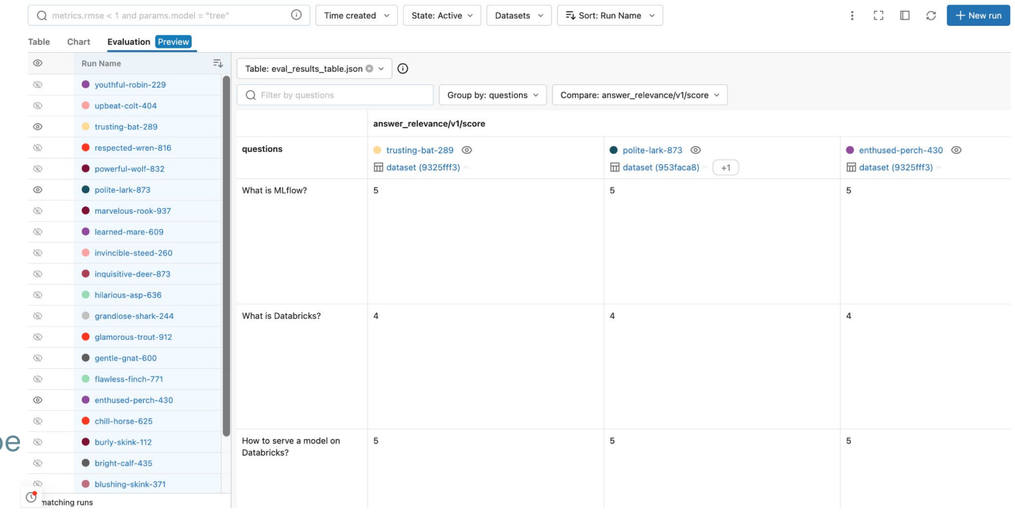
Batch evaluation in code

- **LLM-as-a-judge**: Evaluate using foundation models, to scale beyond human evaluation
- Ground truth: Efficiently evaluate using large curated datasets

```python
from mlflow.metrics.genai.metric_definitions import answer_relevance

answer_relevance_metric = answer_relevance(model="endpoints:/gpt-4")

results = mlflow.evaluate(
 model,
 eval_df,
 model_type="question-answering",
 evaluators="default",
 predictions="result",
 extra_metrics=[answer_relevance_metric, mlflow.metrics.latency()],
 evaluator_config={
  "col_mapping": {
   "inputs": "questions",
   "context": "source_documents",
  }
 }
)
print(results.metrics)


results.tables["eval_results_table"]
```

Interactive evaluation in UI

- Compare multiple models and prompts visually
- Iteratively test new queries during development



databricks