



# AI 3-in-1: Agents, RAG, and Local Models



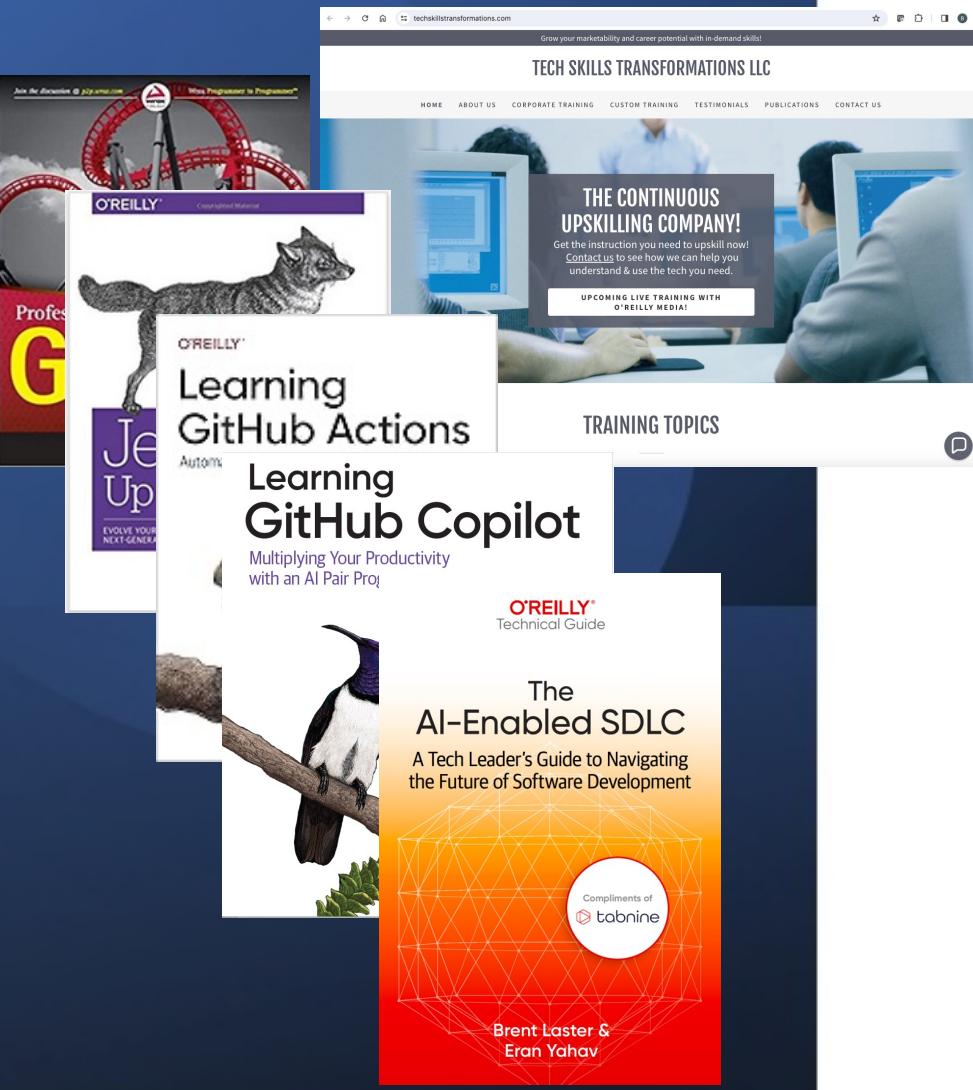
Presented by Brent Laster &  
Tech Skills Transformations LLC

© 2025 Brent C. Laster & Tech Skills Transformations LLC



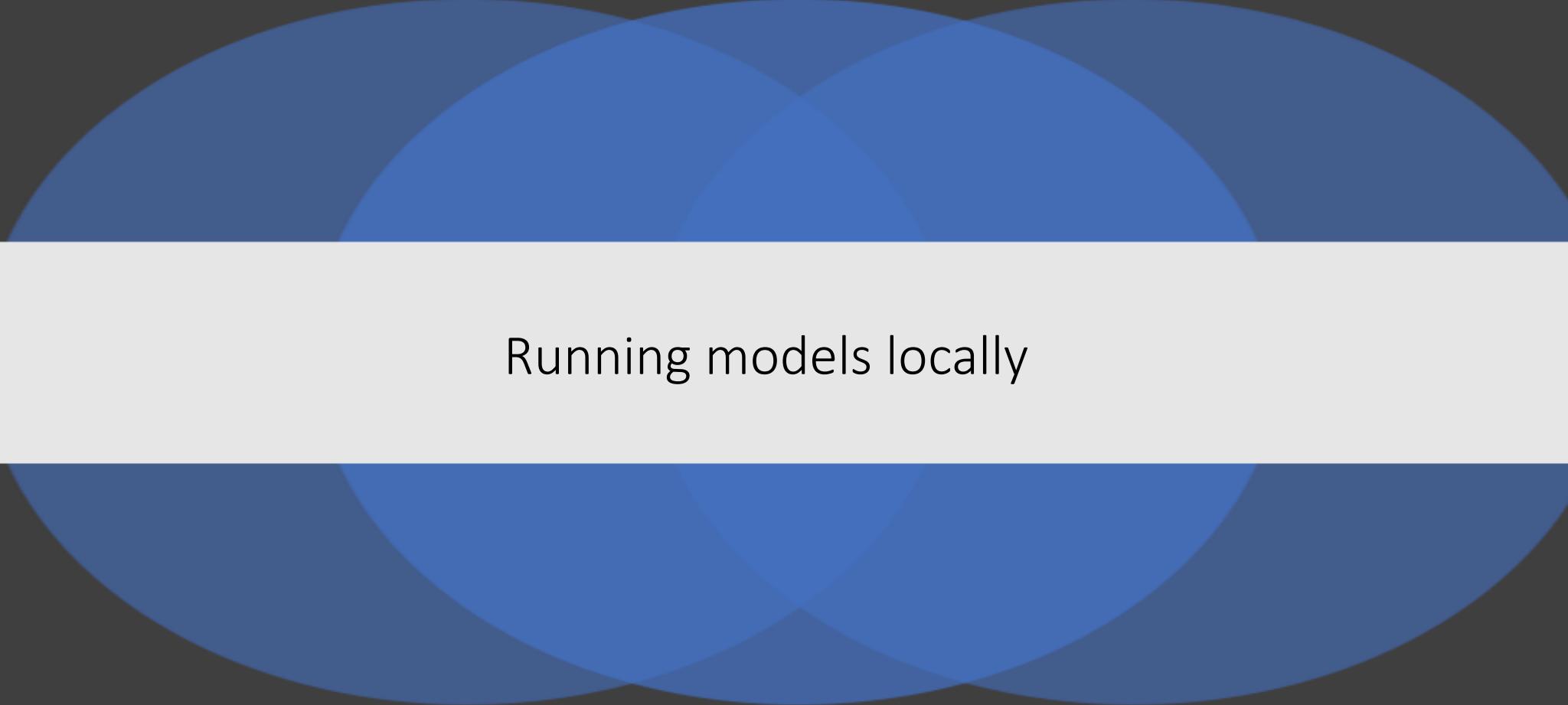
All rights reserved

# About me



- Founder, Tech Skills Transformations LLC
- <https://getskillsnow.com>
- [info@getskillsnow.com](mailto:info@getskillsnow.com)
- Long career in corporate as dev, manager, and director in DevOps and other areas
- Author
  - O'Reilly "reports"
  - Books
    - Professional Git
    - Jenkins 2 – Up and Running
    - Learning GitHub Actions
    - Learning GitHub Copilot
    - AI-Enabled SDLC
- Speaker
- Social media
  - LinkedIn: [brentlaster](#)
  - X: [@BrentCLaster](#)
  - Bluesky: [brentclaster.bsky.social](#)
  - GitHub: [brentlaster](#)





Running models locally



# Why run models locally?

- Privacy - no need to share data
- Gives you control over setup, configuration, and customization options
  - Can tailor LLM to your needs, experiment with settings, integrate into your infra
- Can easily swap between different models for different tasks
- Work in offline mode
- Cost savings
  - No charges for subscriptions or API calls
- No censoring of results





# Where to get models +

<https://huggingface.co/models>

The Hugging Face website interface shows a search bar, navigation links for Models, Datasets, Spaces, Posts, and Docs. On the left, there's a sidebar for Tasks (Libraries, Datasets, Languages, Licenses), Multimodal tasks (Audio-Text-to-Text, Image-Text-to-Text, Visual Question Answering, Document Question Answering, Video-Text-to-Text, Visual Document Retrieval, Any-to-Any), and Computer Vision tasks (Depth Estimation, Image Classification, Object Detection, Image Segmentation). The main content area displays a list of models, with the first few listed:

- Qwen/Qwen3-235B-A22B
- deepseek-ai/DeepSeek-Prover-V2-671B
- nari-labs/Dia-1.6B
- Qwen/Qwen3-30B-A3B
- Qwen/Qwen3-32B

<https://www.kaggle.com/models>

The Kaggle website interface shows a search bar, navigation links for Create, Home, Competitions, Datasets, Models (which is highlighted), Code, Discussions, Learn, and More. The main content area features a "Models" section with a "New Model" button and a "Featured Models" section listing Qwen 3, Gemini 2.5 Flash API, and deepcogito.



# Options for running LLMs locally

- GPT4All - <https://github.com/nomic-ai/gpt4all>
- LM Studio - <https://lmstudio.ai>
- Jan AI - <https://jan.ai>
- llama.cpp - <https://github.com/ggerganov/llama.cpp>
- LlamaFile - <https://github.com/Mozilla-Ocho/llamafile>
- Ollama - <https://ollama.com/>
- HuggingFace Transformers - <https://huggingface.co/docs/transformers>
- More!





# Ollama

- Command line tool for downloading, exploring and using LLMs on local machine
- open source
- supports most of Hugging Face's popular models
- allows uploading new ones
- Links:
  - main site: <https://ollama.com>
  - GitHub: <https://github.com/ollama/>
- Advantages
  - speeds up and simplifies
    - » model selection and download
    - » configuring endpoints
    - » integration with Python or JavaScript codebase

Left side of the browser window showing the Ollama website at [ollama.com](https://ollama.com). The page features a large llama icon, navigation links for Blog, Discord, GitHub, Models, Sign in, and a prominent Download button. The main content area includes a sub-headline "Get up and running with large language models.", a callout for running Llama 2 and Code Llama, a "Customize and create your own." section, a large "Download ↓" button, and a note about availability for macOS, Linux, and Windows (preview).



# Working with Ollama #1

The screenshot shows the Ollama Search extension in a browser window. The URL is ollama.com/search. The main content is a card for the "Llama 3.2" model. The card features a yellow header with the text "Llama3.2" and a stylized stack of colored layers (yellow, orange, red, blue, purple) below it. Below the card, there's a section for "Embedding" and "Vision" tools. The "Llama 3.2" logo is prominently displayed with the text "The current, most capable model that runs on a single GPU." Below the logo, there are buttons for "vision", "1b", "4b", "12b", and "27b". At the bottom, it says "4.1M Pulls", "21 Tags", and "Updated 2 weeks ago". A large blue arrow points upwards from the "Ollama" logo at the bottom to the "ollama pull" text above the model card.



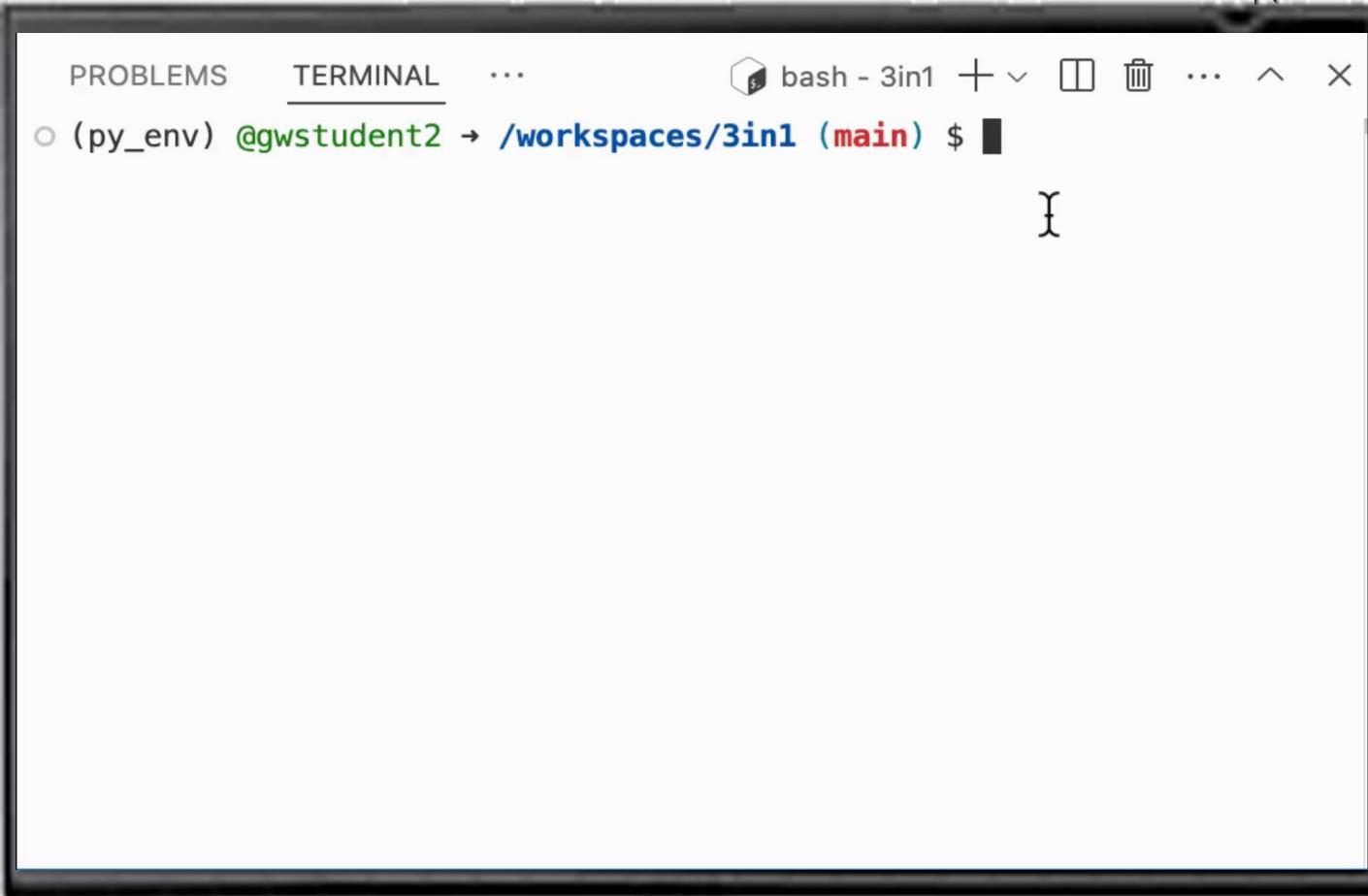
A terminal window is shown with the command "(py\_env) @gwstudent2 → /workspaces/3in1 (main) \$ █" being typed. The terminal has a dark background with light-colored text. The cursor is at the end of the command line, indicated by a small arrow pointing to the right.



# Working with Ollama #2

10

*>>> Briefly explain what  
an AI model is*



```
PROBLEMS TERMINAL ...
(py_env) @gwstudent2 → /workspaces/3in1 (main) $
```

*ollama run*



*>>> query*

llama3.2



# Working with Ollama #3

```
from openai import OpenAI

# Initialize the client to talk to your local Ollama server
client = OpenAI(
    base_url="http://localhost:11434/v1",
    api_key="n-a"
)

# Send a chat completion request
completion = client.chat.completions.create(
    model="llama3.2",
    messages=[
        {"role": "system", "content": "Always answer in 3 bullet points."},
        {"role": "user", "content": "Tell me what AI is."}
    ],
    temperature=0.7,
)

# Print out the assistant's reply
print(completion.choices[0].message.content)
```

The terminal window shows the following details:

- PROBLEMS
- TERMINAL (selected tab)
- ... (dropdown menu)
- bash - 3in1 (main) \$ (cursor)

The terminal command history shows:

```
(py_env) @gwstudent2 → /workspaces/3in1 (main) $
```

*ollama serve*



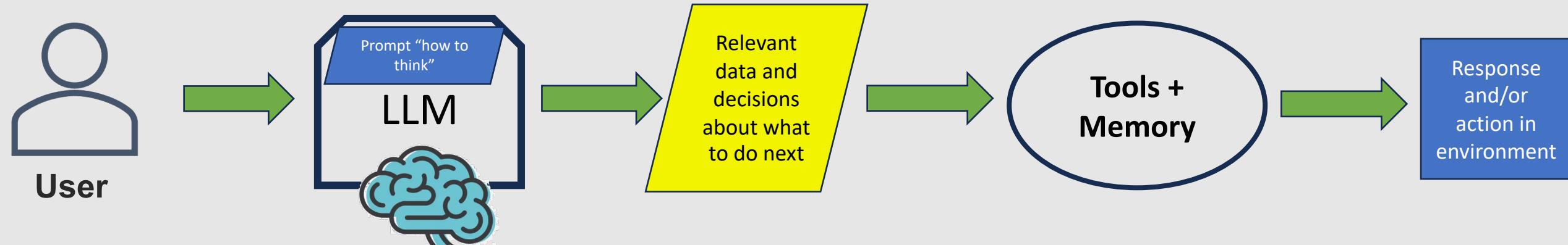
Demo #1 – Simple program to work with local model

# Agents



# What is an AI Agent?

- A **system** that operates within an **environment** by using **sensors** to **perceive** information, a **decision-making mechanism** to process and **reason about the data**, and **actuators** to **take actions that influence or update/respond to the environment**
- This interaction enables the agent to achieve specific goals autonomously while continuously learning and adapting over time
- Agents use LLMs to identify key data, drive decisions, and communicate naturally





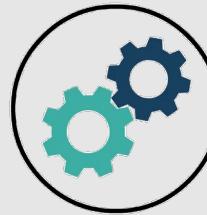
# Architectural Features of AI Agents

## Planning



- AI autonomously outlines and executes a logical series of steps for accomplishing a given objective.
- Provides the AI with a way to dynamically adapt its approach based on real-time data and feedback..
- Might employ reflection to evaluate and improve responses
- **Example:** A research agent plans search → summarize → generate report.

## Tool Use



- AI agents interact with external APIs, databases, and functions.
- Enhances LLMs by providing access to real-world knowledge.
- Reduces hallucinations by using retrieval-augmented generation (RAG).
- **Example:** Calling a Python function to perform complex calculations.

## Memory



- Short-term handles tasks; long term stores knowledge and experience
- Memory ensures consistency and efficiency in multi-step decisions
- Memory recalls preferences to enhance personalization and user experience
- **Example:** Storing user preferences for future reference or personalized responses



A system\_message=""You are an AI assistant designed to help users accurately and efficiently. Your primary goal is to provide precise, helpful, and clear responses.

You have access to the following tools:

**Tool Name: find\_weather, Description: Get weather for a location., Arguments: latitude: float, longitude: float, Outputs: string**

You should think step by step in order to fulfill the objective with a reasoning process divided into Thought/Action/Observation. This cycle can repeat multiple times if needed.

You should first reflect with “Thought: {your\_thoughts}” on the current query, then (if necessary), call a tool with the proper JSON formatting “Action: {JSON\_BLOB}”, or else print your final answer starting with the prefix “Final Answer:”“””



# Agent Example

18

**system\_message="”You are an AI assistant designed to help users accurately and efficiently. Your primary goal is to provide precise, helpful, and clear responses.**

**You have access to the following tools:**

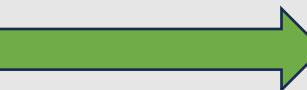
**Tool Name: find\_weather, Description: Get weather for a location., Arguments: latitude: float, longitude: float, Outputs: string**

**You should think step by step in order to fulfill the objective with a reasoning process divided into Thought/Action/Observation. This cycle can repeat multiple times if needed.**

**You should first reflect with “Thought: {your\_thoughts}” on the current query, then (if necessary), call a tool with the proper JSON formatting “Action: {JSON\_BLOB}”, or else print your final answer starting with the prefix “Final Answer:”“””**



What's the weather in Paris?



User

## Chain of Thought – Step 1: Interpret User Query

Thought: "The user is asking about the weather in Paris. I need to extract 'Paris' as the location.

Action: Extracted location = "Paris"



AI Agent



# Agent Example

19

**system\_message="”You are an AI assistant designed to help users accurately and efficiently. Your primary goal is to provide precise, helpful, and clear responses.**

**You have access to the following tools:**

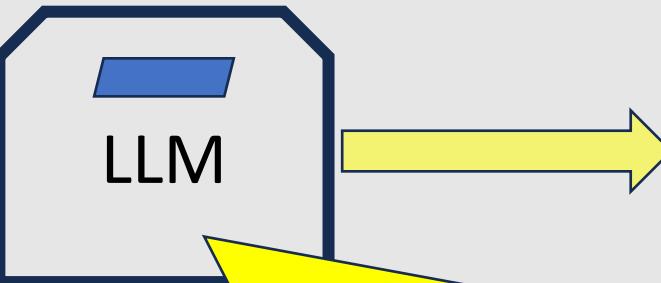
**Tool Name:** find\_weather, **Description:** Get weather for a location., **Arguments:** latitude: float, longitude: float, **Outputs:** string

**You should think step by step in order to fulfill the objective with a reasoning process divided into Thought/Action/Observation. This cycle can repeat multiple times if needed.**

**You should first reflect with “Thought: {your\_thoughts}” on the current query, then (if necessary), call a tool with the proper JSON formatting “Action: {JSON\_BLOB}”, or else print your final answer starting with the prefix “Final Answer:”“””**



What's the weather in Paris?



**Chain of Thought – Step 2: Decide to use tool**  
Thought: "I need real-time data, so I will call the '**find\_weather**' tool. First, I need to get the latitude and longitude for the tool call."

```
AIResponse(  
    tool_calls=[  
        {  
            name: "find_weather"  
            parameters: {  
                latitude: "48.8566",  
                longitude: "2.3522",  
            },  
            id: "call_tool123",  
            type: "tool_invoke"  
        }  
    ]  
)
```



AI Agent



# Agent Example



What's the weather in Paris?

User



`system_message=""You are an AI assistant designed to help users accurately and efficiently. Your primary goal is to provide precise, helpful, and clear responses.`

`You have access to the following tools:`

`Tool Name: find_weather, Description: Get weather for a location., Arguments: latitude: float, longitude: float, Outputs: string`

`You should think step by step in order to fulfill the objective with a reasoning process divided into Thought/Action/Observation. This cycle can repeat multiple times if needed.`

`You should first reflect with "Thought: {your_thoughts}" on the current query, then (if necessary), call a tool with the proper JSON formatting "Action: {JSON_BLOB}", or else print your final answer starting with the prefix "Final Answer:"""`

20

`AIResponse(  
 tool_calls=[  
 name: "find_weather"  
 parameters: {  
 latitude: "48.8566",  
 longitude: "2.3522",  
 },  
 id: "call_tool123",  
 type: "tool_invoke"  
 ]  
)`



AI Agent

Agent parses LLM output  
identifies JSON tool call,  
parses it, forms it into  
actual tool call

{  
 name: "find\_weather"  
 parameters: {  
 latitude: "48.8566",  
 longitude: "2.3522",  
 },  
 id: "call\_tool123",  
 type: "tool\_invoke"  
}



# Agent Example

21

**system\_message=""""**You are an AI assistant designed to help users accurately and efficiently. Your primary goal is to provide precise, helpful, and clear responses.

You have access to the following tools:

Tool Name: `find_weather`, Description: Get weather for a location., Arguments: `latitude: float, longitude: float`, Outputs: `string`

You should think step by step in order to fulfill the objective with a reasoning process divided into Thought/Action/Observation. This cycle can repeat multiple times if needed.

You should first reflect with “Thought: {your\_thoughts}” on the current query, then (if necessary), call a tool with the proper JSON formatting “Action: {JSON\_BLOB}”, or else print your final answer starting with the prefix “Final Answer:”“””



What's the weather in Paris?



User

```
AIResponse(  
    tool_calls=[  
        name: "find_weather"  
        parameters: {  
            latitude: "48.8566",  
            longitude: "2.3522",  
        },  
        id: "call_tool123",  
        type: "tool_invoke"  
    ]  
)
```

Agent executes tool call



AI Agent

```
{  
    name: "find_weather"  
    parameters: {  
        latitude: "48.8566",  
        longitude: "2.3522",  
    },  
    id: "call_tool123",  
    type: "tool_invoke"  
}
```



# Agent Example

22



What's the weather in Paris?

User



```
ToolResponse(  
    content="53 and rainy",  
    name="find_weather",  
    tool_invoke_id:  
    "call_tool123"  
)
```

system\_message="" You are an AI assistant designed to help users accurately and efficiently. Your primary goal is to provide precise, helpful, and clear responses.  
You have access to the following tools:  
Tool Name: find\_weather, Description: Get weather for a location., Arguments: latitude: float, longitude: float, Outputs: string  
You should think step by step in order to fulfill the objective with a reasoning process divided into Thought/Action/Observation. This cycle can repeat multiple times if needed.  
You should first reflect with "Thought: {your\_thoughts}" on the current query, then (if necessary), call a tool with the proper JSON formatting "Action: {JSON\_BLOB}", or else print your final answer starting with the prefix "Final Answer:"""

```
AIResponse(  
    tool_calls=[  
        name: "find_weather"  
        parameters: {  
            latitude: "48.8566",  
            longitude: "2.3522",  
        },  
        id: "call_tool123",  
        type: "tool_invoke"  
    ]  
)
```

Weather tool returns result

Weather Search Tool

AI Agent

```
{  
    name: "find_weather"  
    parameters: {  
        latitude: "48.8566",  
        longitude: "2.3522",  
    },  
    id: "call_tool123",  
    type: "tool_invoke"  
}
```



# Agent Example

23

`system_message=""`You are an AI assistant designed to help users accurately and efficiently. Your primary goal is to provide precise, helpful, and clear responses.

You have access to the following tools:

Tool Name: `find_weather`, Description: Get weather for a location., Arguments: `latitude: float, longitude: float`, Outputs: string

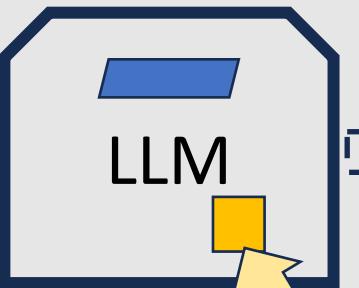
You should think step by step in order to fulfill the objective with a reasoning process divided into Thought/Action/Observation. This cycle can repeat multiple times if needed.

You should first reflect with “Thought: {your\_thoughts}” on the current query, then (if necessary), call a tool with the proper JSON formatting “Action: {JSON\_BLOB}”, or else print your final answer starting with the prefix “Final Answer:”“””

What's the weather in Paris?



User



Agent includes tool output in message/prompt back to model

`ToolResponse(`  
  `content="53 and rainy",`  
  `name="find_weather",`  
  `tool_invoke_id:`  
  `"call_tool123"`  
`)`

`AIResponse(`  
  `tool_calls=[{`  
    `name: "find_weather"`  
    `parameters: {`  
      `latitude: "48.8566",`  
      `longitude: "2.3522",`  
    `},`

`"call_tool123",`  
  `name: "tool_invoke"`

{  
  `name: "find_weather"`  
  `parameters: {`  
    `latitude: "48.8566",`  
    `longitude: "2.3522",`  
  `},`  
  `id: "call_tool123",`  
  `type: "tool_invoke"`

Weather Search Tool

AI Agent



# Agent Example

24

**system\_message="”You are an AI assistant designed to help users accurately and efficiently. Your primary goal is to provide precise, helpful, and clear responses.**

**You have access to the following tools:**

**Tool Name:** find\_weather, **Description:** Get weather for a location., **Arguments:** latitude: float, longitude: float, **Outputs:** string

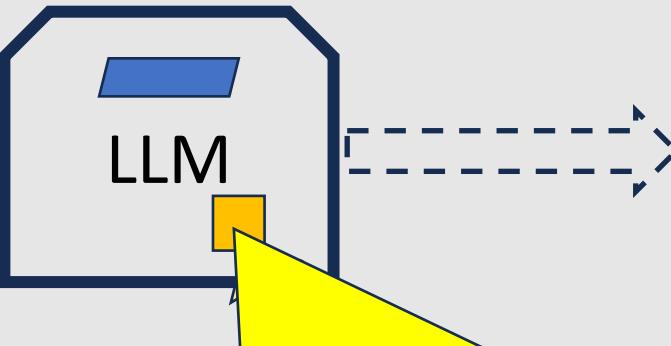
**You should think step by step in order to fulfill the objective with a reasoning process divided into Thought/Action/Observation. This cycle can repeat multiple times if needed.**

**You should first reflect with “Thought: {your\_thoughts}” on the current query, then (if necessary), call a tool with the proper JSON formatting “Action: {JSON\_BLOB}”, or else print your final answer starting with the prefix “Final Answer:”“””**



What's the weather in Paris?

User



**Chain of Thought – Step 3 : Interpret JSON Response**  
Thought: "The tool returned weather data for Paris. I will summarize the information concisely."

```
name="find_weather",
tool_invoke_id:
"call_tool123"
)
```

**AIResponse**  
tool\_calls=[  
  name: "find\_weather"  
  parameters: {  
    latitude: "48.8566",  
    longitude: "2.3522",  
  },  
  id: "call\_tool123",  
  type: "tool\_invoke"  
]  
)



AI Agent

```
{
  name: "find_weather"
  parameters: {
    latitude: "48.8566",
    longitude: "2.3522",
  },
  id: "call_tool123",
  type: "tool_invoke"
}
```



# Agent Example

25

**system\_message="”You are an AI assistant designed to help users accurately and efficiently. Your primary goal is to provide precise, helpful, and clear responses.**

**You have access to the following tools:**

**Tool Name: find\_weather, Description: Get weather for a location., Arguments: latitude: float, longitude: float, Outputs: string**

**You should think step by step in order to fulfill the objective with a reasoning process divided into Thought/Action/Observation. This cycle can repeat multiple times if needed.**

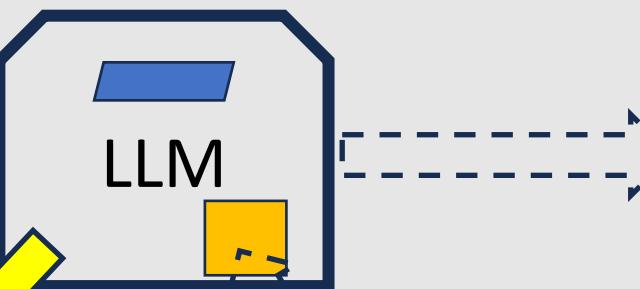
**You should first reflect with “Thought: {your\_thoughts}” on the current query, then (if necessary), call a tool with the proper JSON formatting “Action: {JSON\_BLOB}”, or else print your final answer starting with the prefix “Final Answer:”“””**



What's the weather in Paris?

User

AIFinalResponse(  
content="The current  
weather in Paris is 53  
degrees Celsius with  
light rain."  
)



ToolResponse(  
content="53 and  
rainy",  
name="find\_weather",  
tool\_invoke\_id:  
"call\_tool123"  
)

AIResponse(  
tool\_calls=[  
    name: "find\_weather"  
    parameters: {  
        latitude: "48.8566",  
        longitude: "2.3522",  
    },  
    id: "call\_tool123",  
    type: "tool\_invoke"  
])



AI Agent

{  
    name: "find\_weather"  
    parameters: {  
        latitude: "48.8566",  
        longitude: "2.3522",  
    },  
    id: "call\_tool123",  
    type: "tool\_invoke"  
}



# Agent Example

26

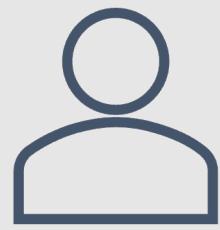
**system\_message=""""**You are an AI assistant designed to help users accurately and efficiently. Your primary goal is to provide precise, helpful, and clear responses.

You have access to the following tools:

Tool Name: `find_weather`, Description: Get weather for a location., Arguments: `latitude: float, longitude: float`, Outputs: string

You should think step by step in order to fulfill the objective with a reasoning process divided into Thought/Action/Observation. This cycle can repeat multiple times if needed.

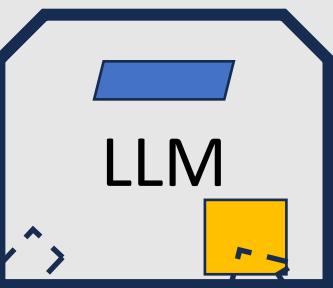
You should first reflect with “Thought: {your\_thoughts}” on the current query, then (if necessary), call a tool with the proper JSON formatting “Action: {JSON\_BLOB}”, or else print your final answer starting with the prefix “Final Answer:”“””



What's the weather in Paris?

User

```
AIFinalResponse(  
    content="The current  
    weather in Paris is 53  
    degrees Celsius with  
    light rain."  
)
```



```
ToolResponse(  
    content="53 and  
    rainy",  
    name="find_weather",  
    tool_invoke_id:  
    "call_tool123"  
)
```

```
AIResponse(  
    tool_calls=[  
        {  
            name: "find_weather"  
            parameters: {  
                location: "Paris",  
            },  
            id: "call_tool123",  
            type: "tool_invoke"  
        }]  
)
```



```
{  
    name: "find_weather"  
    parameters: {  
        latitude: "48.8566",  
        longitude: "2.3522",  
    },  
    id: "call_tool123",  
    type: "tool_invoke"  
}
```

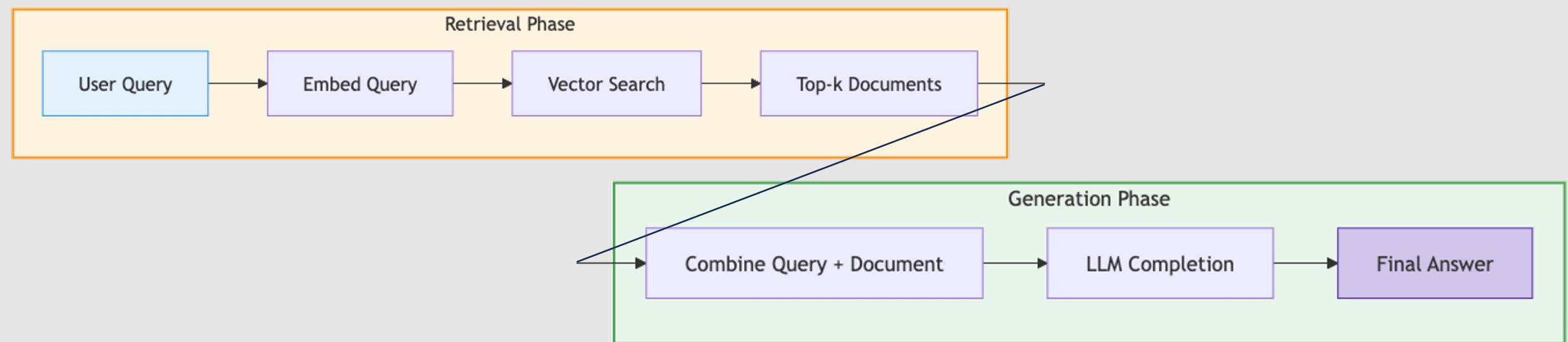
## Demo #2 – Adding agency to our code



RAG

# What is RAG and how does it work?

- **Combination of retrieval and generation:** RAG combines information retrieval (like a search engine) with text generation (like a language model).
- **Uses external knowledge:** Instead of relying solely on pre-trained knowledge, RAG retrieves relevant documents or data from an external source (like a database or private knowledge bases) to generate more accurate and up-to-date responses.
- **Improves factual accuracy:** By pulling in real-time data or documents, RAG reduces the risk of generating factually incorrect or outdated information.
- **Two-step process:**
  - **Retrieve:** The model searches for relevant information from a knowledge source.
  - **Generate:** It then uses the retrieved data to create a coherent, contextually accurate answer.

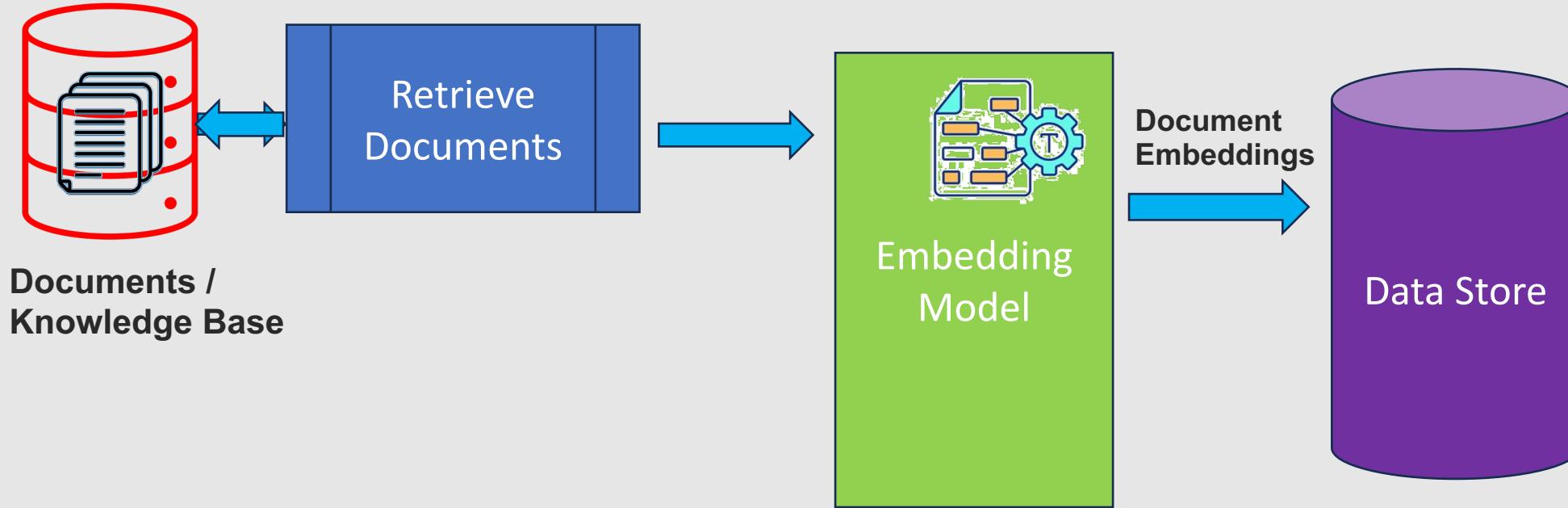


Source: <https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/>



# How is RAG setup?

*Doc Ingestion and Retrieval*

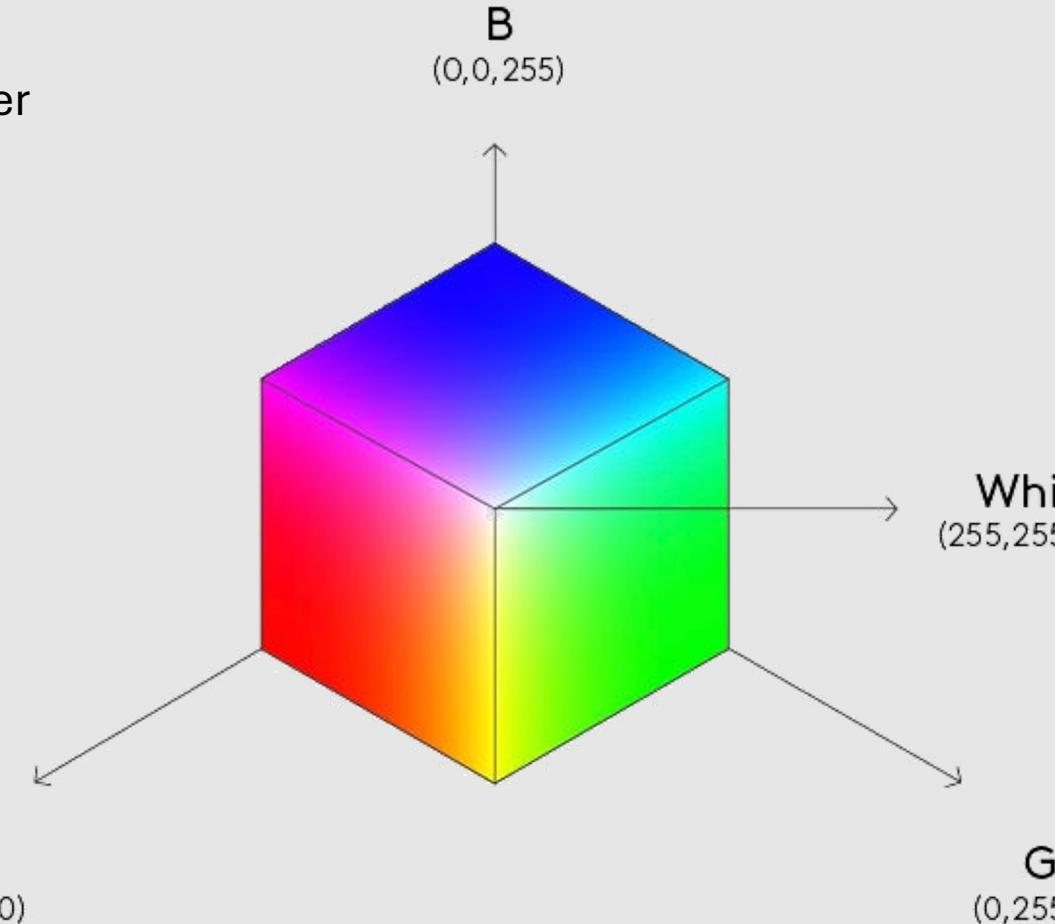


- You provide data sources and point application to them
- Info is retrieved from the data sources and tokenized, embedded and stored in a data store
- For queries/prompts, application gathers results (most relevant ones) from the vector database with your data



# Embeddings

- Embeddings represent text as sets of numeric data - tensors (lots of dimensions)
- Each dimension stores some info about the text's meaning, context, or syntactical aspects
- Words or sentences with similar meanings are stored closer together in the vector space
  - If two pieces of text are similar syntactically, they will have similar embeddings (smaller distance between their vectors)
- During training, models learn to place text with similar meanings closer together in the embedding space
- Common pre-trained models used for generating embeddings include BERT and variants (RoBERTa, DistilBERT)
- Once you have embeddings, you can use them for NLP tasks like semantic search, text classification, sentiment analysis

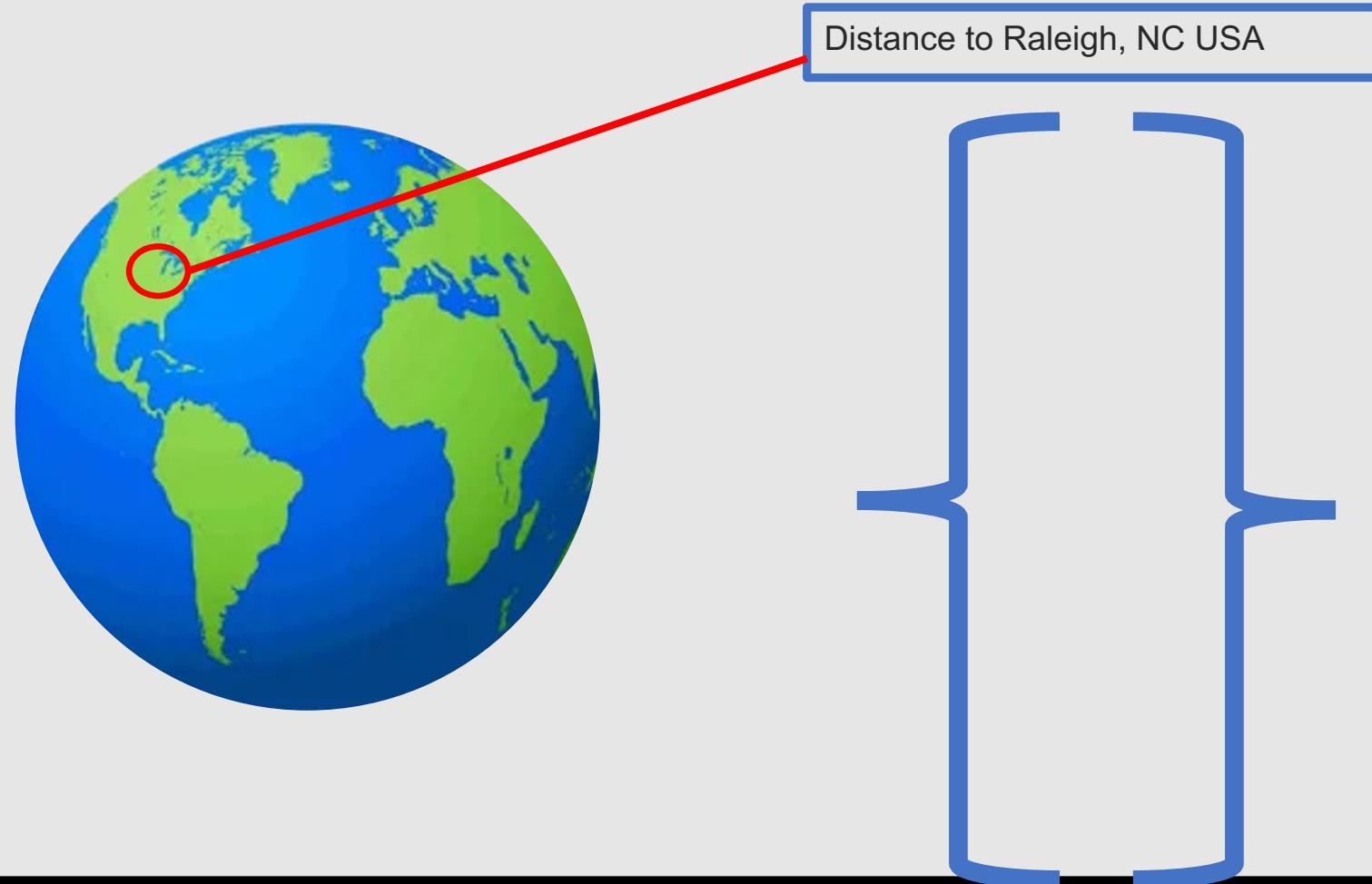




# Understanding vectors in AI

32

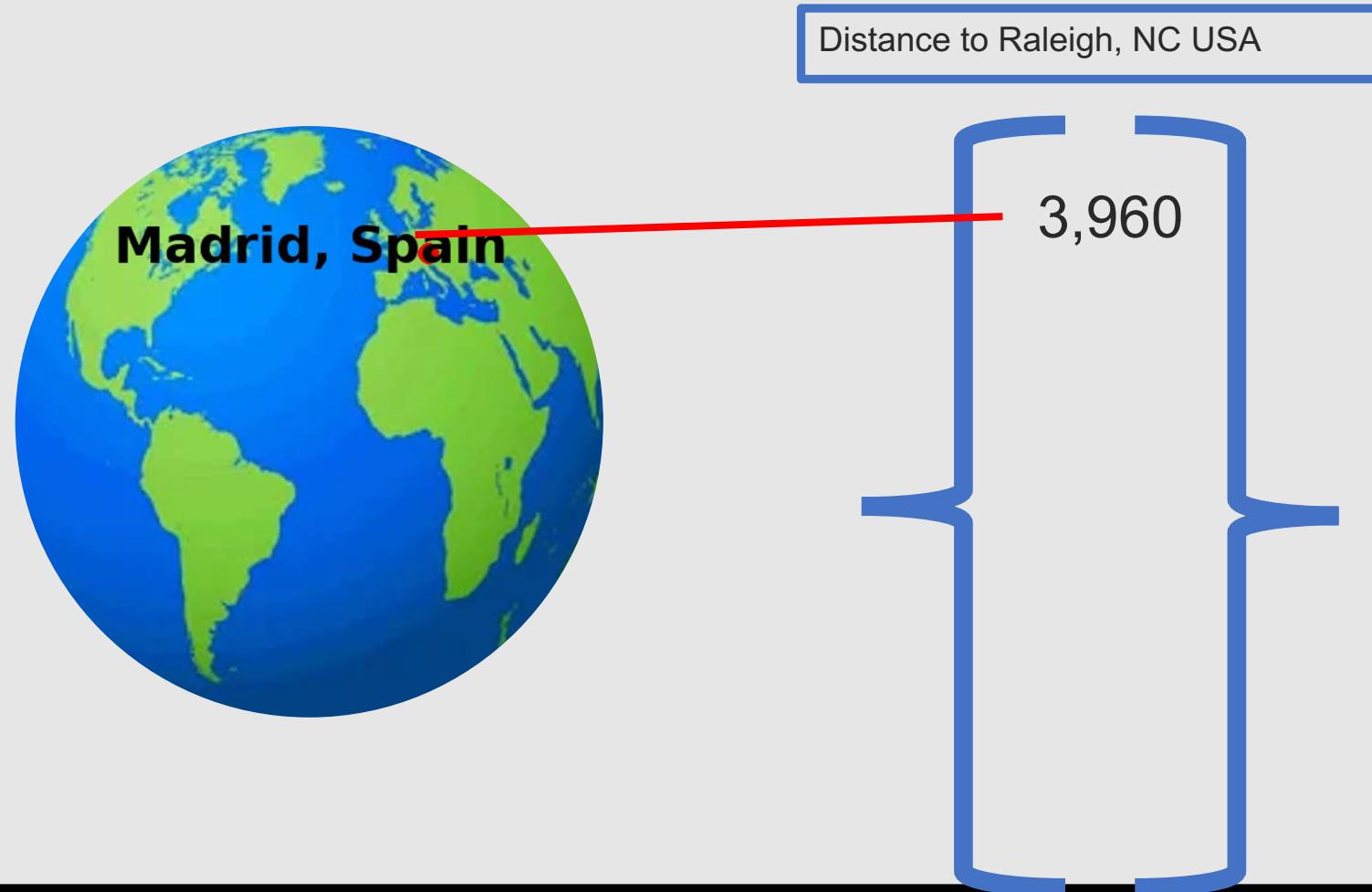
- Collection of data points that encapsulate an item's relationship to other items





# Understanding vectors in AI

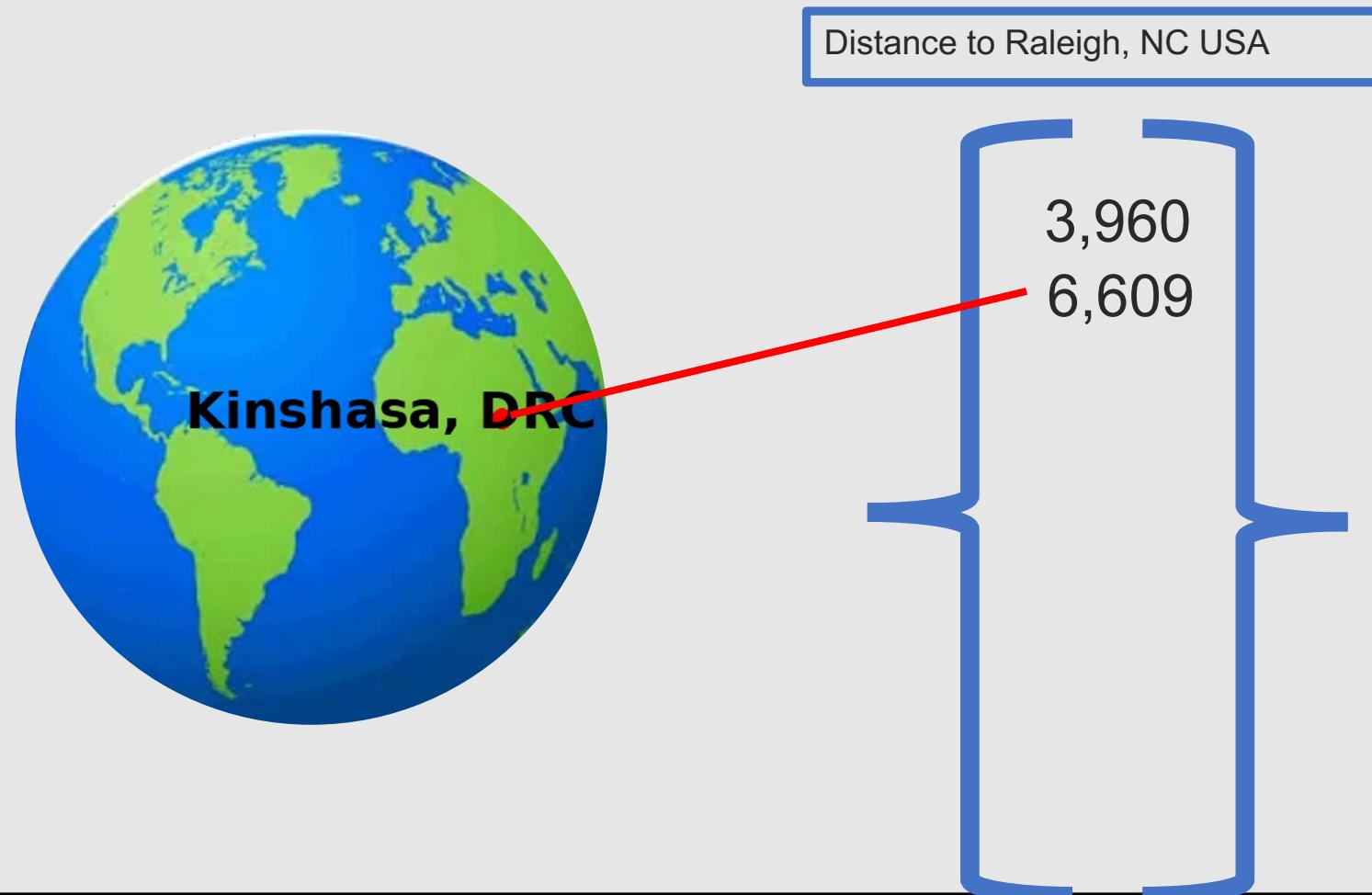
- Collection of data points that encapsulate an item's relationship to other items





# Understanding vectors in AI

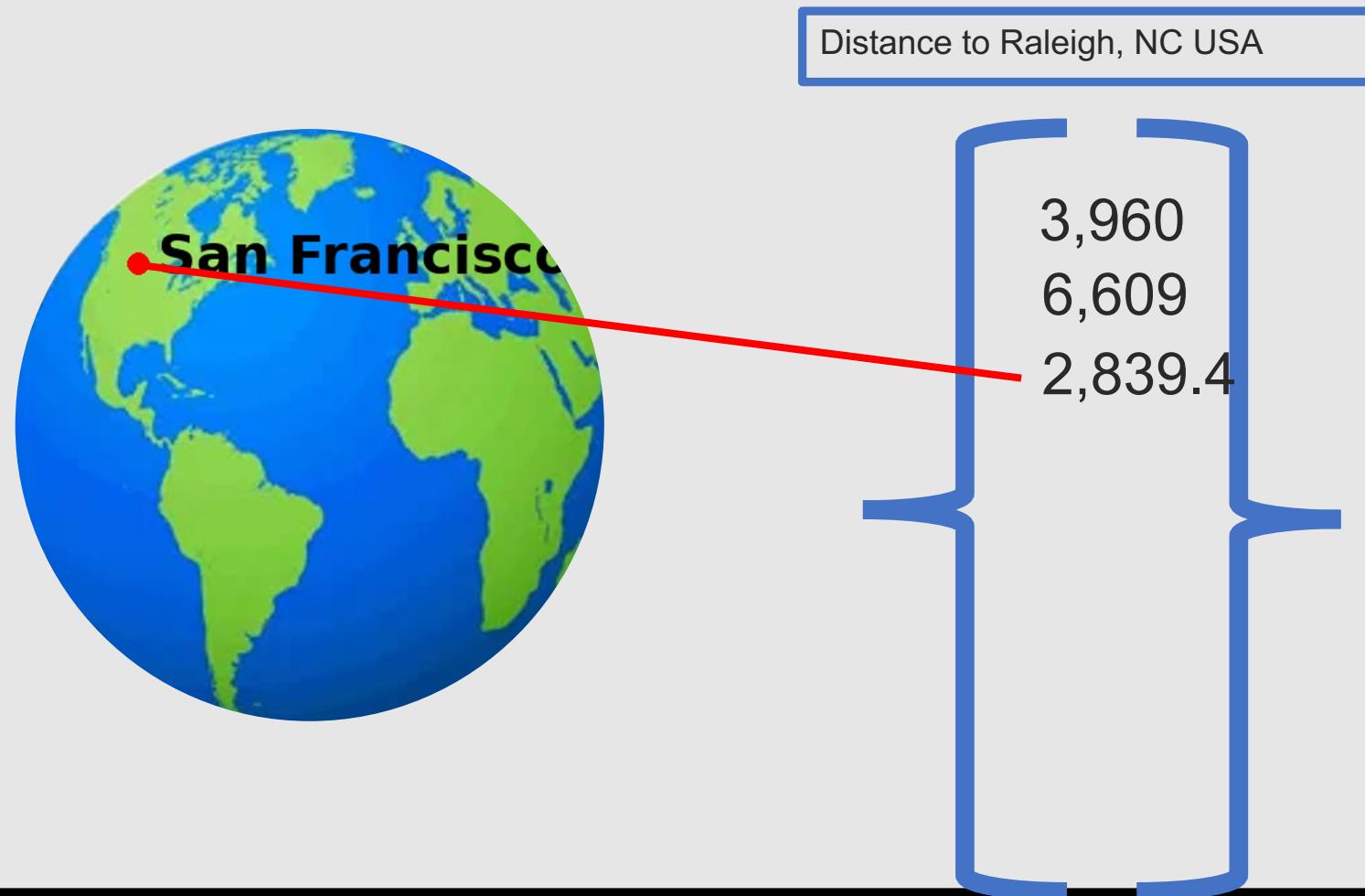
- Collection of data points that encapsulate an item's relationship to other items





# Understanding vectors in AI

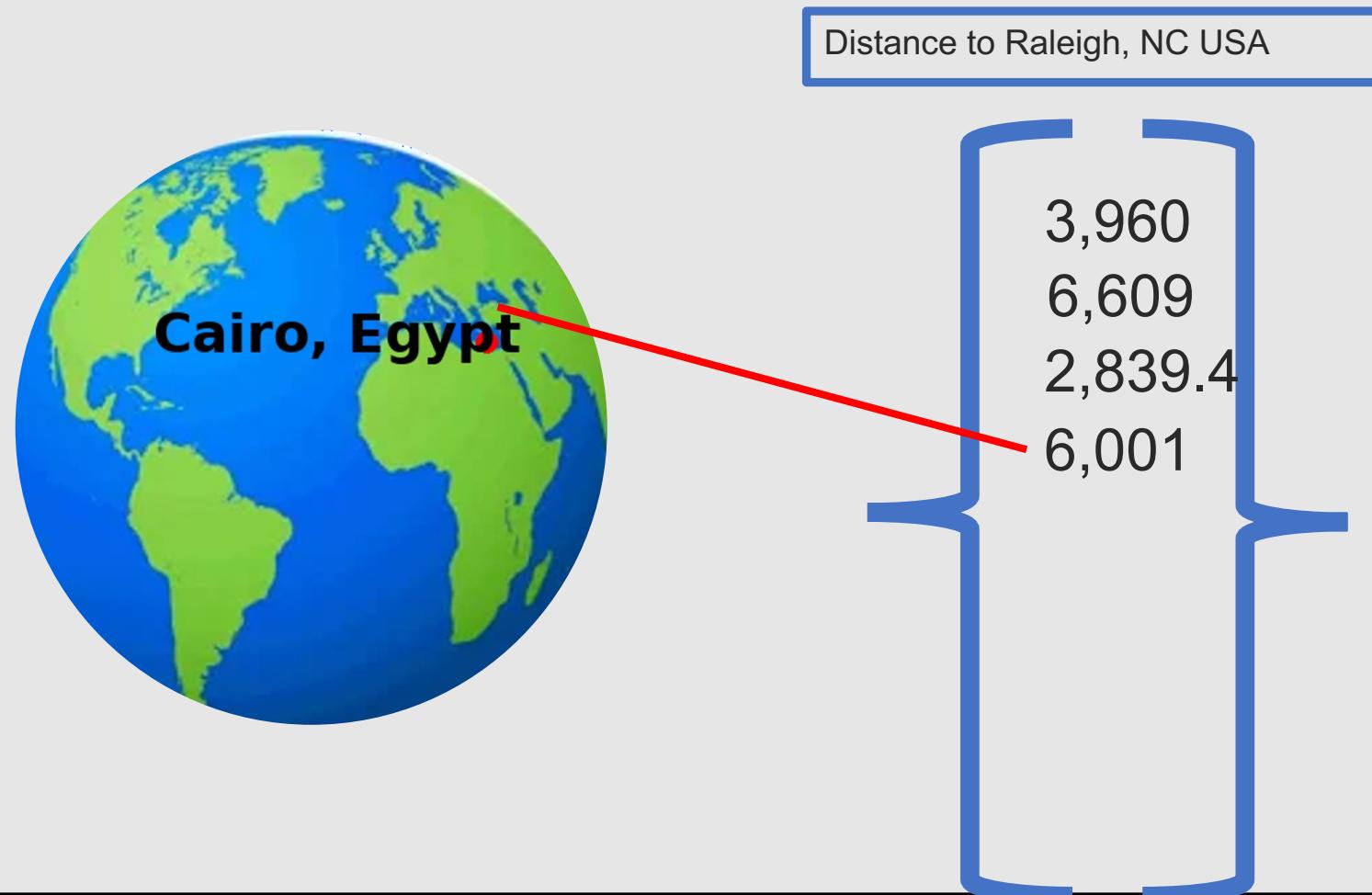
- Collection of data points that encapsulate an item's relationship to other items





# Understanding vectors in AI

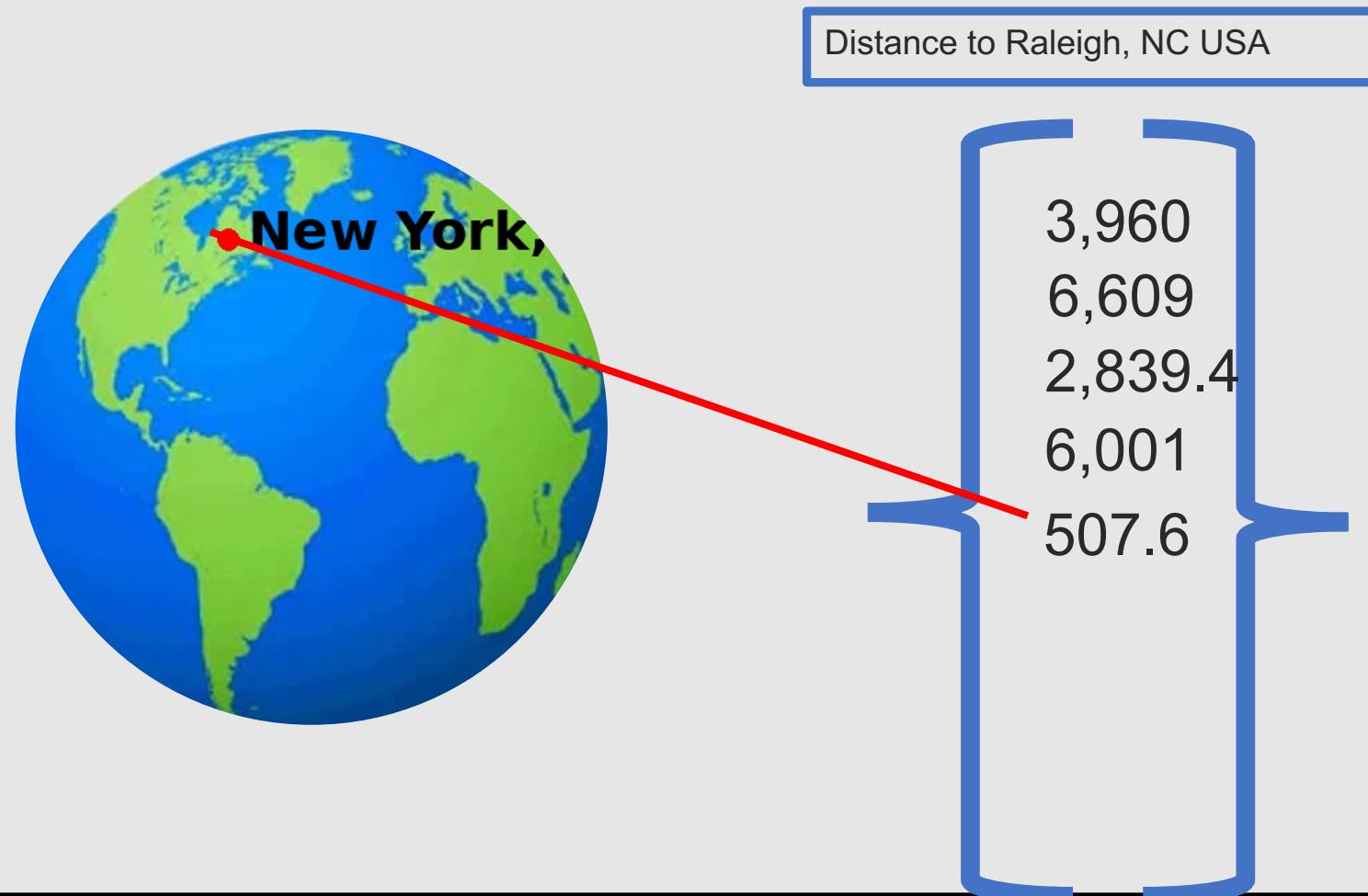
- Collection of data points that encapsulate an item's relationship to other items





# Understanding vectors in AI

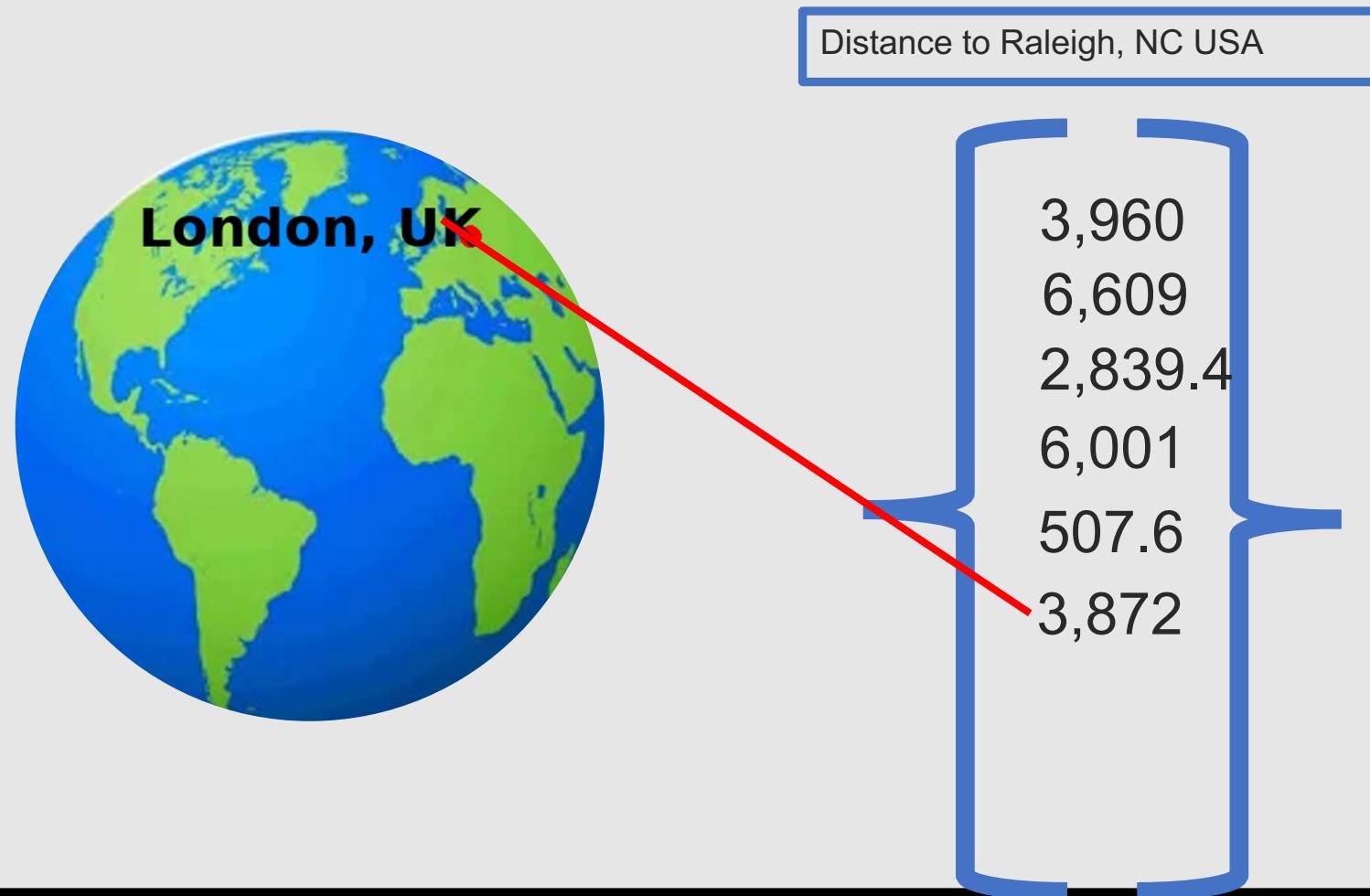
- Collection of data points that encapsulate an item's relationship to other items





# Understanding vectors in AI

- Collection of data points that encapsulate an item's relationship to other items





# Understanding vectors in AI

- Collection of data points that encapsulate an item's relationship to other items



Distance to Raleigh, NC USA

3,960  
6,609  
2,839.4  
6,001  
507.6  
3,872  
7,679



# Understanding vectors in AI

- Collection of data points that encapsulate an item's relationship to other items



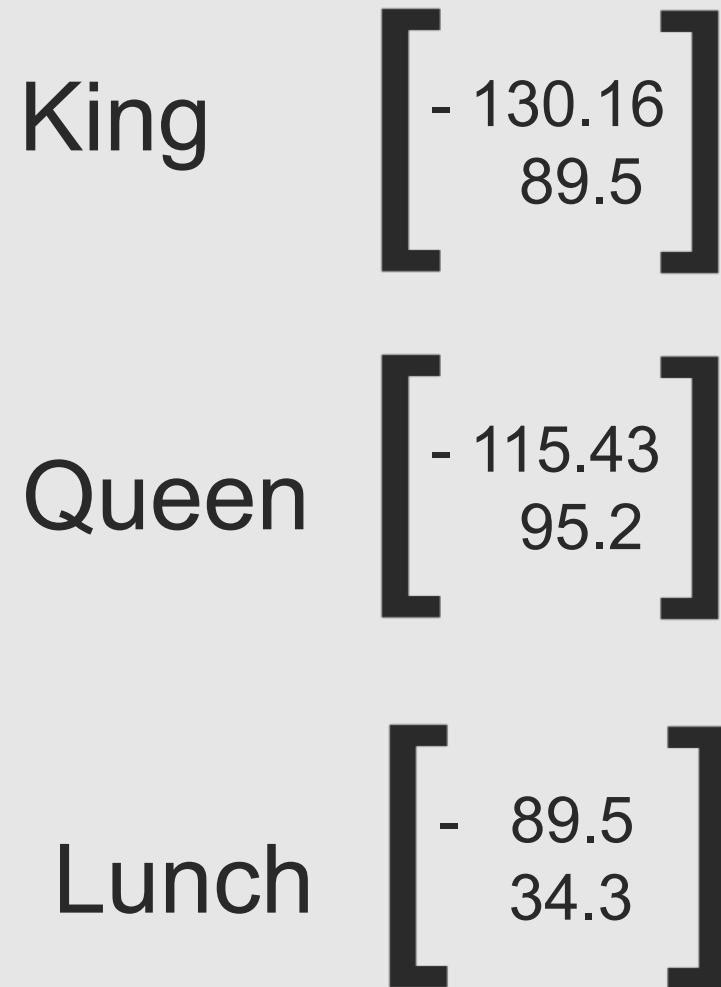
Distance to Raleigh, NC USA

3,960  
6,609  
2,839.4  
6,001  
507.6  
3,872  
7,679  
2,870.1



# Semantic meaning / relationships

- Suppose we have 3 words
- King and Queen are more similar to each other than they are to lunch
- In order for neural net to understand the relationships, each word needs to be represented as a vector
- Suppose each word is represented by a 2-dimensional vector

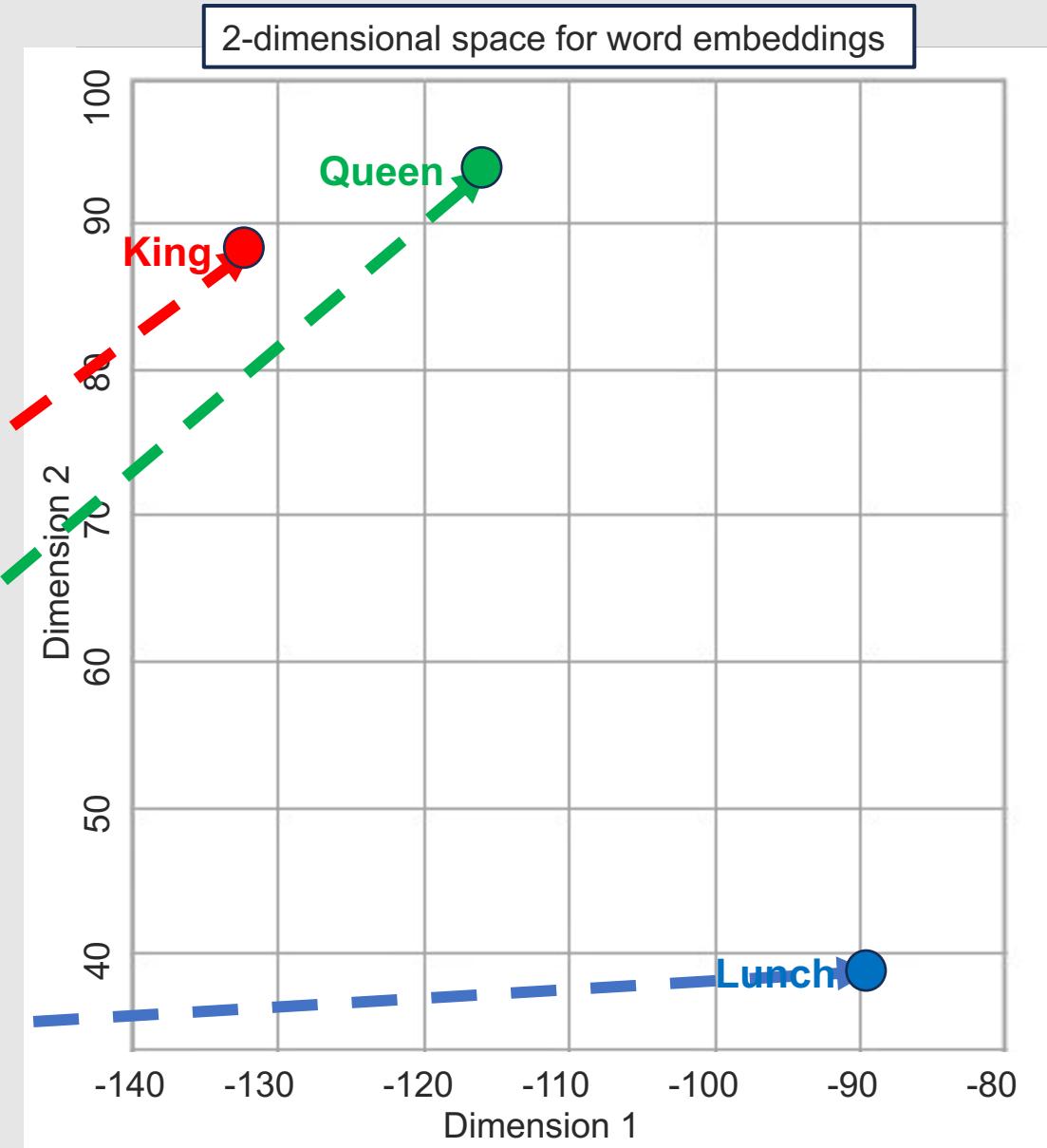




# Embedding space

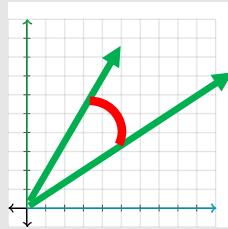
- Plotting in 2-dimensional embedding space shows relationships
- Way to let NN understand relationships between words
- We want the NN to learn that King and Queen are more similar to each other than they are to lunch

|       |   |
|-------|---|
| King  | $\begin{bmatrix} -130.16 \\ 89.5 \end{bmatrix}$ |
| Queen | $\begin{bmatrix} -115.43 \\ 95.2 \end{bmatrix}$ |
| Lunch | $\begin{bmatrix} -89.5 \\ 34.3 \end{bmatrix}$   |

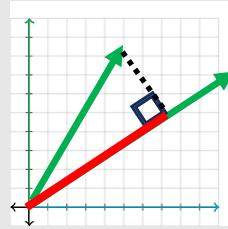


# Searching for Vectors - similarity metrics

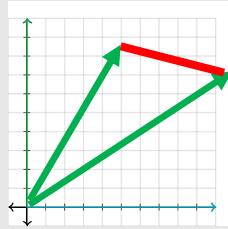
- 3 metrics commonly used to determine similarity of two vectors (2-dimensional representation)



**Cosine similarity** - measure the angle between two vectors; values from -1 to 1; 1 = both point in same direction; -1 point in opposite directions; 0 = orthogonal (perpendicular)



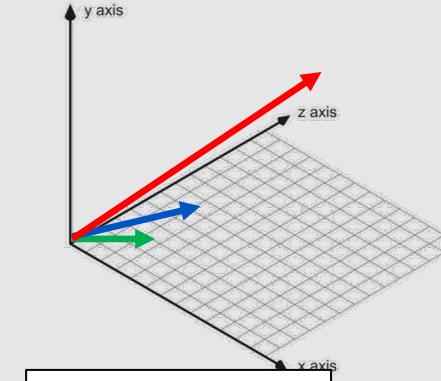
**Dot product / inner product** - measures how well 2 vectors align with each other; values from  $-\infty$  to  $\infty$ ; positive values indicate vectors are in same direction; negative values indicate opposite directions; 0 = orthogonal



**Euclidean distance** - measures the distance between two vectors; values from 0 to  $\infty$ ; 0 = identical; larger numbers farther apart

imagine 3 vectors -  $a, b, c$

$$a = \begin{bmatrix} .01 \\ .07 \\ .1 \end{bmatrix} \quad b = \begin{bmatrix} .01 \\ .08 \\ .11 \end{bmatrix} \quad c = \begin{bmatrix} .91 \\ .57 \\ .6 \end{bmatrix}$$



## Cosine similarity

$$\text{sim}(u, v) = \frac{u \cdot v}{\|u\| \|v\|} = \frac{\sum_{i=1}^n a_i b_i}{\sqrt{\sum_{i=1}^n u_i^2} \sqrt{\sum_{i=1}^n v_i^2}}$$

0.0141

$$\text{sim}(a, b) = \frac{(a_1 * b_1) + (a_2 * b_2) + (a_3 * b_3)}{\sqrt{a_1^2 + a_2^2 + a_3^2} \sqrt{b_1^2 + b_2^2 + b_3^2}}$$

$$= \frac{(0.01 * 0.01) + (0.07 * 0.08) + (0.1 * 0.11)}{\sqrt{0.01^2 + 0.07^2 + 0.1^2} \sqrt{0.01^2 + 0.08^2 + 0.11^2}}$$

## Dot product / inner product

$$u \cdot v = |u| |v| \cos\theta = \sum_{i=1}^n a_i b_i \quad a \cdot b = (a_1 b_1) + (a_2 b_2) + (a_3 b_3)$$

0.0167

Dot product formula

$$= (0.01 * 0.01) + (0.07 * 0.08) + (0.1 * 0.11)$$

## Euclidean distance

$$d(u, v) = \sqrt{\sum_{i=1}^n (u_i - v_i)^2}$$

0.9998

Euclidean distance formula

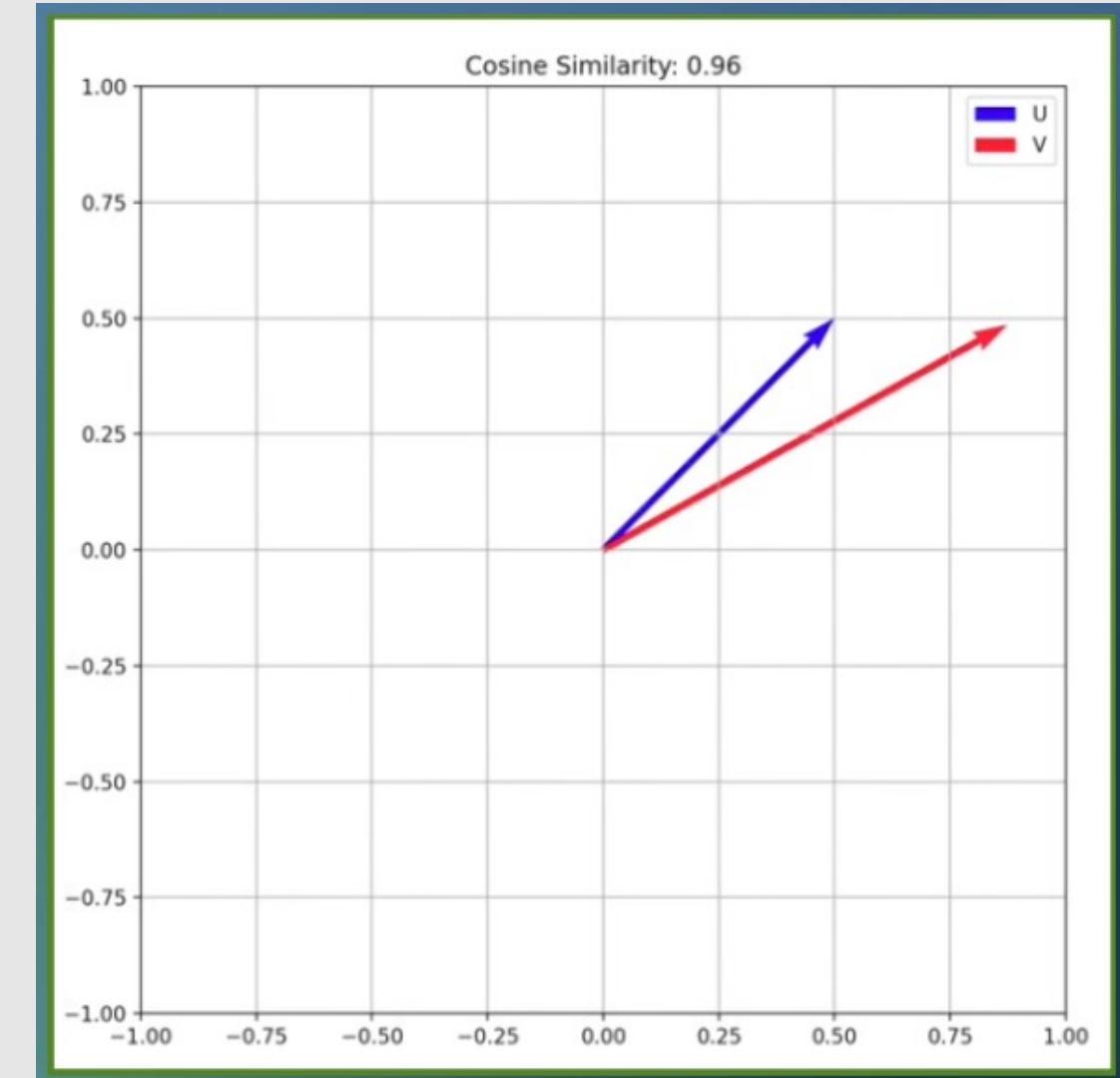
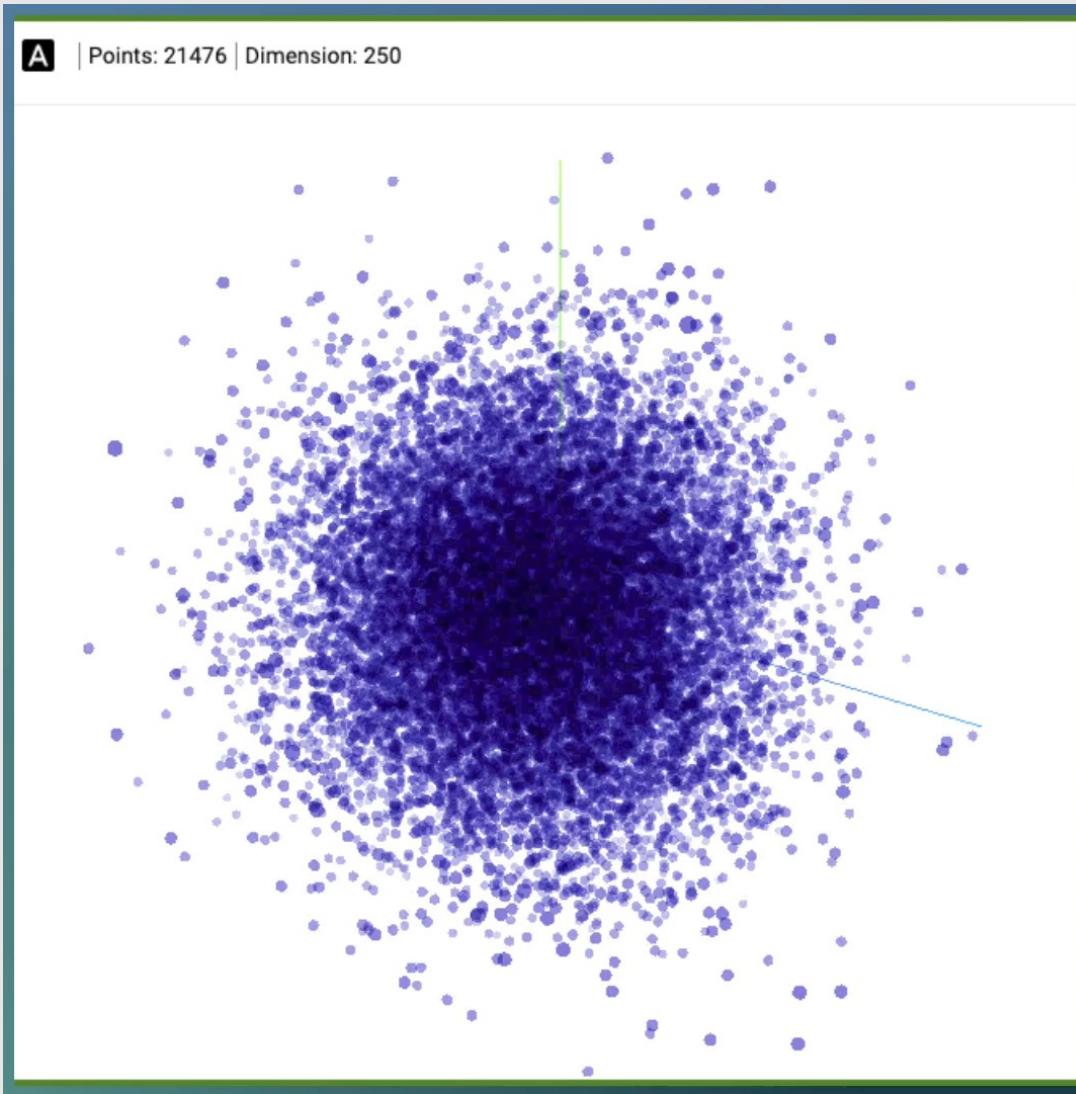
$$d(a, b) = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + (b_3 - a_3)^2}$$

$$= \sqrt{(0.01 - 0.01)^2 + (0.08 - 0.07)^2 + (0.11 - 0.1)^2}$$

edit: <https://towardsdatascience.com/similarity-metrics-in-nlp-acc0777e234c>



# Visualizing Embeddings and Vector Similarity

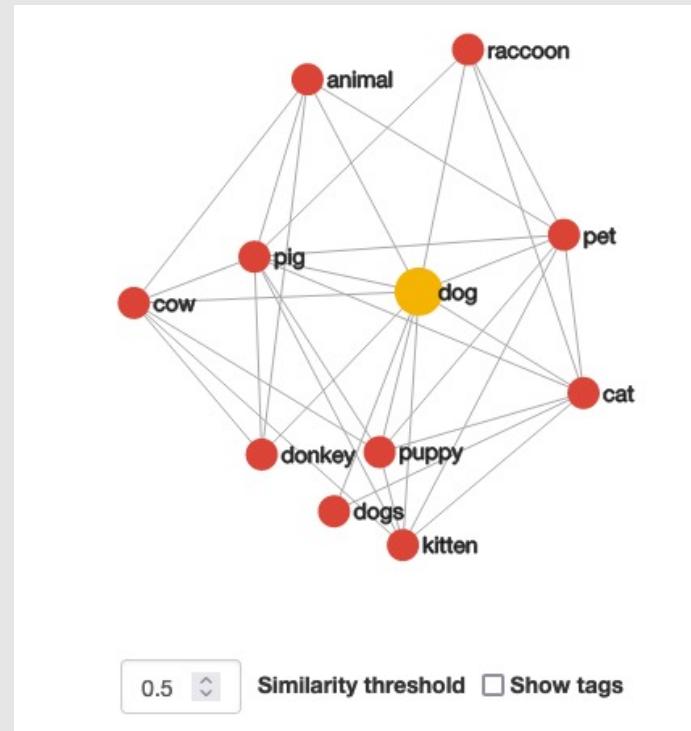
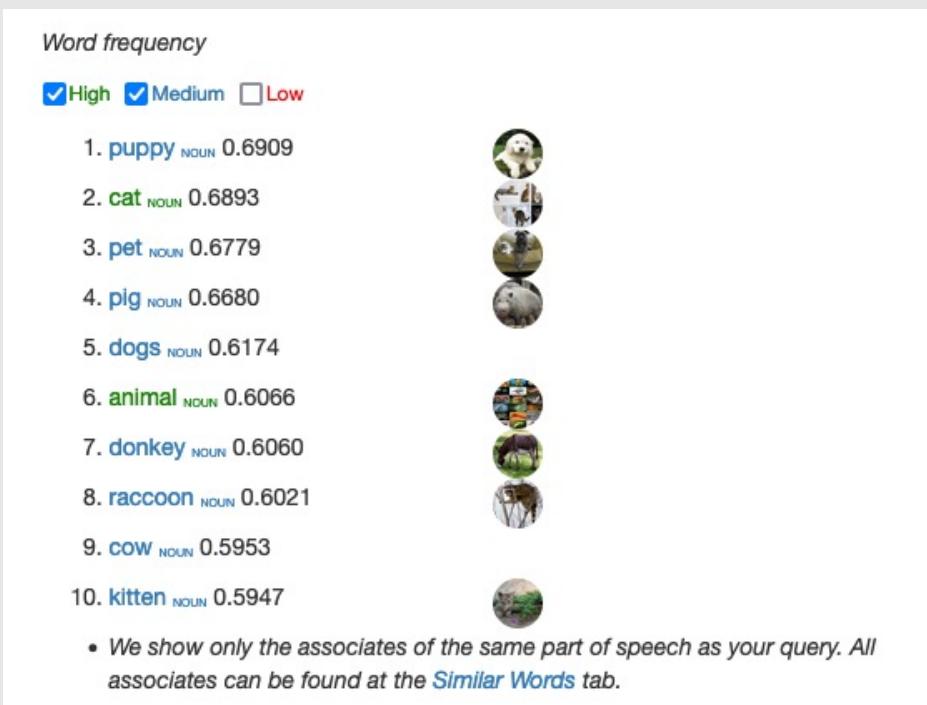


source: [https://projector.tensorflow.org/?config=https://gist.githubusercontent.com/martin-labrecque/4483ff5a104f0b56417585c3bc9a12f1/raw/57348e12a70c8d70c2c573d3dbc0122ac077556b/journaux\\_config.json](https://projector.tensorflow.org/?config=https://gist.githubusercontent.com/martin-labrecque/4483ff5a104f0b56417585c3bc9a12f1/raw/57348e12a70c8d70c2c573d3dbc0122ac077556b/journaux_config.json)



# Vectors and relationships example

- Query - what words are related to "dog" in model "English Wikipedia"?



Show the raw vector of «dog» in model

MOD\_enwiki\_upos\_skipgram\_300\_2\_2021:

```
[-0.03301828354597092, 0.05134638026356697, 0.0036009703762829304,
-0.04066073149442673, 0.10361430048942566, 0.013021323829889297,
0.028161464259028435, -0.0027567853685468435, 0.03388035297393799,
-0.044882044196128845, 0.005169689189642668, -0.05818631127476692,
0.0533536821603775, 0.016616210341453552, 0.0203078053816588,
-0.008570297621190548, -0.10925538837909698, -0.0708925873041153,
0.04675082117319107, -0.03091960959136486, -0.05172094330191612,
0.04471702128648758, 0.008674593642354012, -0.01816382259130478,
0.0590931884944389, 0.10409023612737656, 0.05633684620261192,
-0.024881813675165176, 0.01872968301177025, 0.007228093687444925,
-0.023127363994717598, 0.01528552919626236, -0.0643191784620285,
-0.010359424166381359, -0.06104437634348869, -0.13868044316768646,
-0.023004498332738876, 0.0038427673280239105, -0.021551262587308884,
-0.03467748314142227, 0.010687021538615227, -0.017304275184869766,
0.026886526495218277, -0.0030398862436413765, -0.03685504570603371,
-0.06017328053712845, 0.047442398965358734, -0.10714898258447647,
0.14808930456638336, -0.06579480320215225, -0.004342162515968084,
0.06226382404565811, 0.08031187951564789, -0.055930640548467636,
-0.07030591368675232, 0.015474628657102585, 0.05367768555879593,
0.0917837843298912, 0.031899698078632355, 0.055091146379709244,
-0.025078952312469482, -0.048126623034477234, -0.09730836749076843,
-0.07128141075372696, 0.019415033981204033, -0.025872433558106422,
-0.01761292852461338, 0.015608762390911579, -0.029876720160245895,
-0.008602319285273552, 0.049825914204120636, 0.06784739345312119,
0.005586292129009962, -0.07148509472608566, -0.03097137063741684,
-0.020296750590205193, 0.05099814385175705, 0.14920306205749512,
0.03855258508923683, -0.0818730816245079, -0.061504941142366814]
```

Source: [http://vectors.nlpl.eu/explore/embeddings/en/MOD\\_enwiki\\_upos\\_skipgram\\_300\\_2\\_2021/dog\\_NOUN/](http://vectors.nlpl.eu/explore/embeddings/en/MOD_enwiki_upos_skipgram_300_2_2021/dog_NOUN/)

# Vector Databases



# Vector Databases

- Specialized database that index and stores *vector embeddings*
- Useful for
  - fast retrieval
  - similarity search
- Offer comprehensive data management capabilities
  - metadata storage
  - filtering
  - dynamic querying based on associate metadata
- Scalable and can handle large volumes of vector data
- Support real-time updates
- Play key role in AI and ML applications

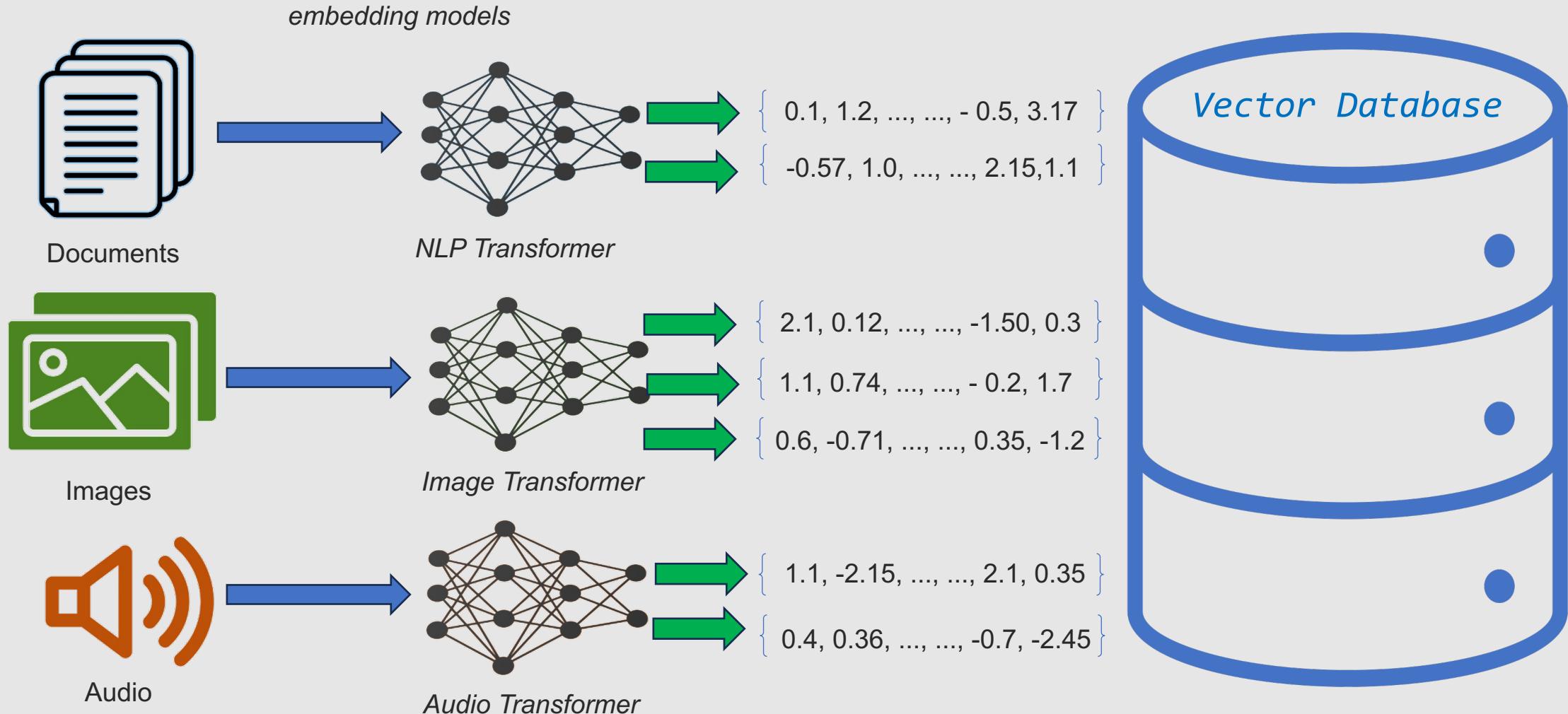




# How data gets into Vector Databases

48

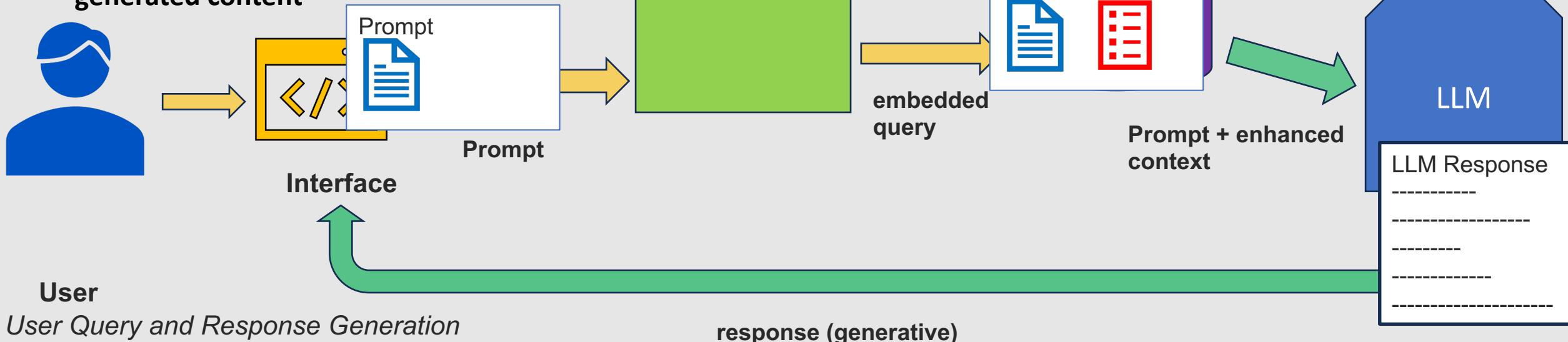
- Data is input, converted to embeddings (vectors) and stored
- Queries are input, converted to embeddings (vectors) and then **similarity metrics** are used to find results ("nearest neighbors")





# How does RAG work?

- For queries/prompts, application gathers results (most relevant ones) from the vector database with your data
- Adds results to your regular LLM query/prompt
- Asks the LLM to answer based on the augmented/enriched query/prompt
- NOTE: Items returned via RAG search are existing items from the data store, not generated content**



## Demo #3 – Adding RAG to our code



# DIY - [github.com/brentlaster/3in1](https://github.com/brentlaster/3in1)

- Fork if desired
- Click on button in README to start codespace
- Follow guide.md

[github.com/brentlaster/3in1](https://github.com/brentlaster/3in1)

[README](#) [MIT license](#)

## ATO Meetup: 3-in-1 - Agents, RAG and Local Models

Repository for ATO Meetup Examples

These instructions will guide you through configuring a GitHub Codespaces environment that you can use to run the code if you want.

1. Click on the button below to start a new codespace from this repository.

Click here [Open in GitHub Codespaces](#)

2. Then click on the option to create a new codespace.

[Codespaces](#)

[github.com/brentlaster/3in1](https://github.com/brentlaster/3in1)

brentlaster / 3in1

Type to search

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

3in1 Public Pin Unwatch 1 Fork 0 Star 0

main 1 Branch 0 Tags Go to file + <> Code About

brentlaster Update requirements.txt 20d7266 · 1 hour ago 36 Commits

.devcontainer Update devcontainer.json 4 days ago

charts Delete charts/flow.txt 3 hours ago

Code samples and DIY for ATO 3in1 Meetup

Readme MIT license

<https://literate-garbanzo-776p6qr45fr447.github.dev>

3in1 [Codespaces: literate garbanzo]

EXPLORER [Preview] README.md local.py

code > local.py

```

1 # Import necessary libraries
2 import json
3 import requests
4 import math
5 from openai import OpenAI
6
7 # ANSI color codes for terminal output
8 BLUE = "\u001b[94m"
9 GREEN = "\u001b[92m"
10 RED = "\u001b[91m"
11 RESET = "\u001b[0m"
12 BOLD = "\u001b[1m"

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1

(py\_env) @brentlaster → /workspaces/3in1 (main) \$

Spaces: 4 UTF-8 LF Python Layout: U.S.

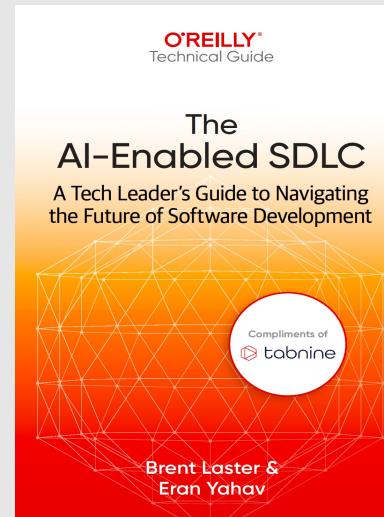
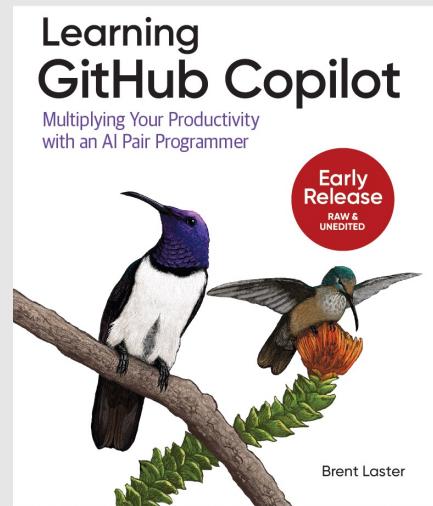
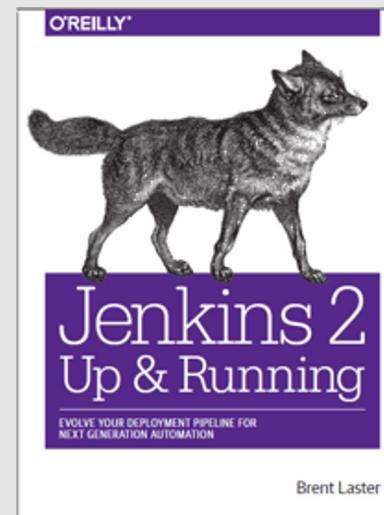
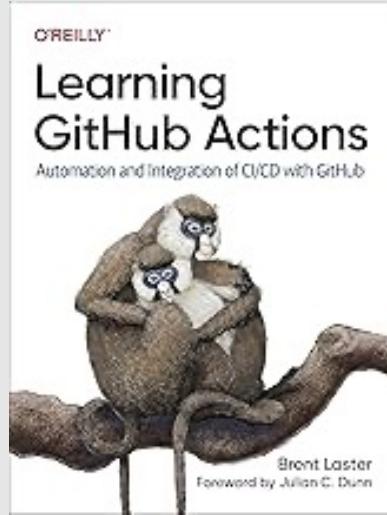
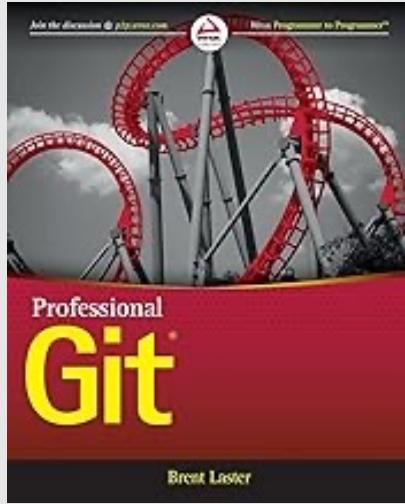


# That's all - thanks!

52

**Contact:** [training@getskillsnow.com](mailto:training@getskillsnow.com)

[techskillstransformations.com](http://techskillstransformations.com)  
[getskillsnow.com](http://getskillsnow.com)



- ❑ [LinkedIn: brentlaster](#)
- ❑ [X: @BrentCLaster](#)
- ❑ [Bluesky: brentclaster.bsky.social](#)
- ❑ [GitHub: brentlaster](#)