weaviate

Vector DBs: a deep dive with Weaviate

JP Hwang
Senior developer educator

Study guide

Key topics covered:

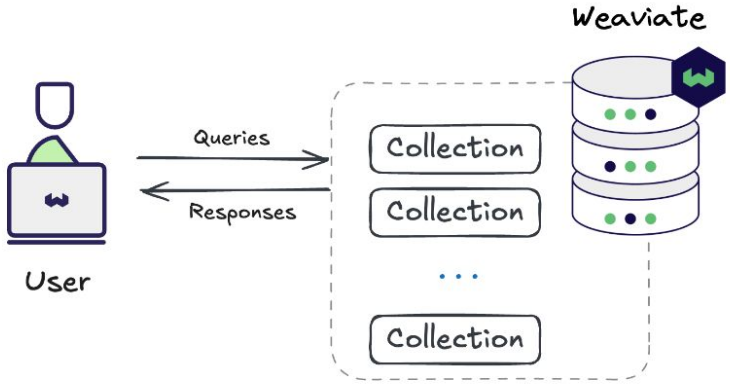
- What vectors are
- Differences between search types
- Role of embedding models
- Role of generative models
- Overview of HNSW
- Agentic AI

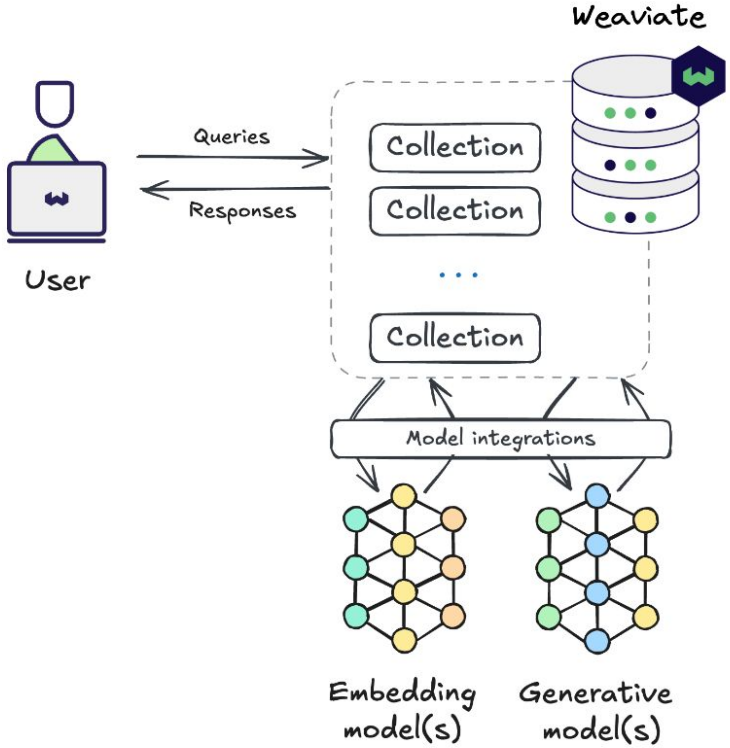
Example questions:

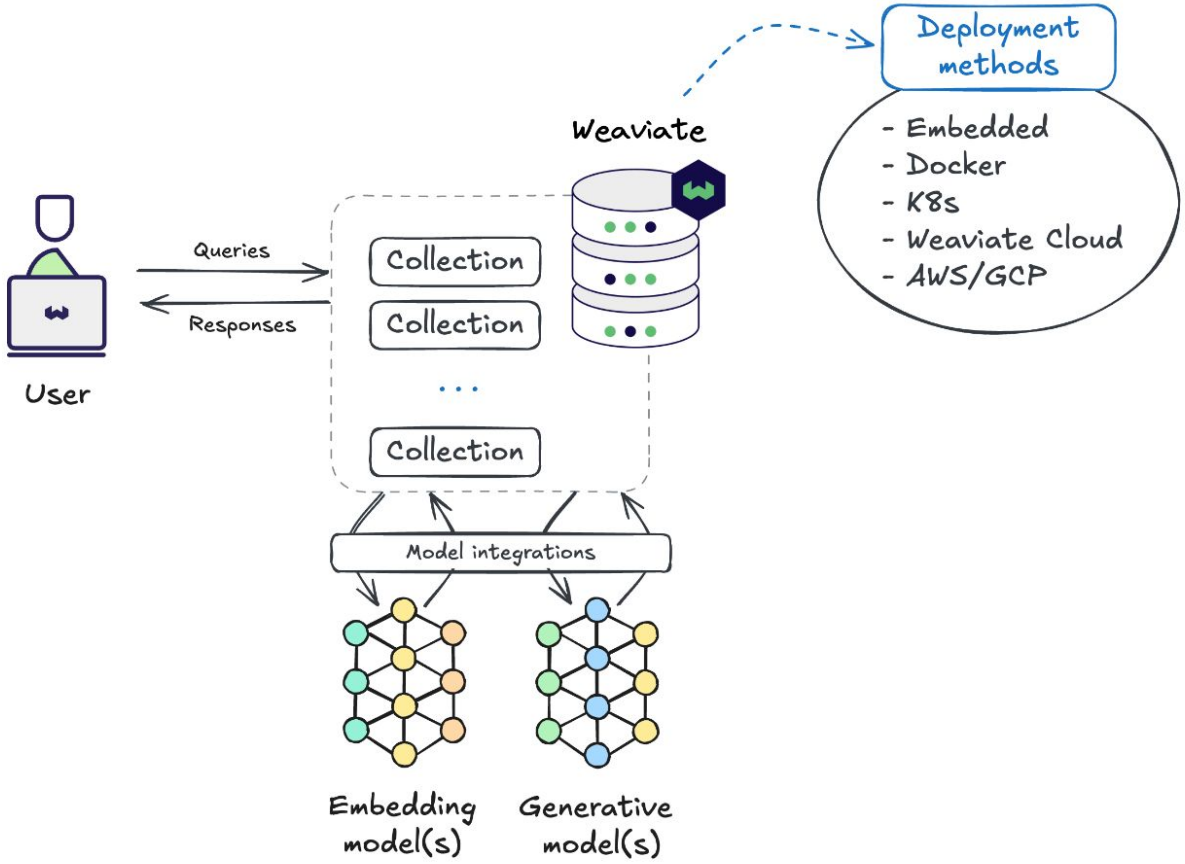
- What does a vector embedding capture?
- Which of hybrid, vector or keyword searches are based on a comparison of meanings?
- What does an embedding model output?
- What does a generative AI model output?
- In the G step of RAG, which type (embedding or generative) of model is used?
- In an HNSW index, does search begin on a higher (sparse) or a lower (dense) layer?
- What makes an application “agentic”?

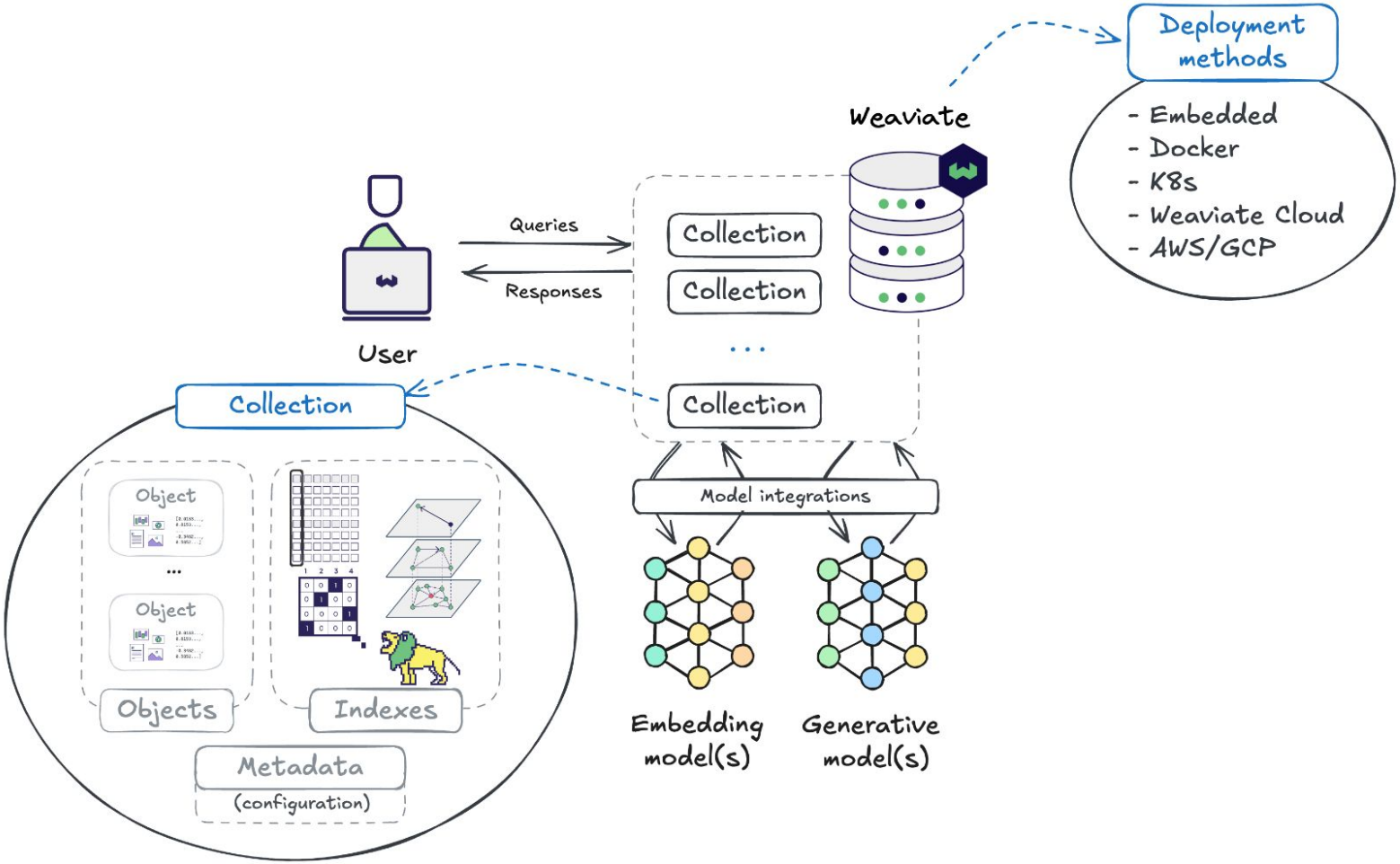
What is Weaviate ?

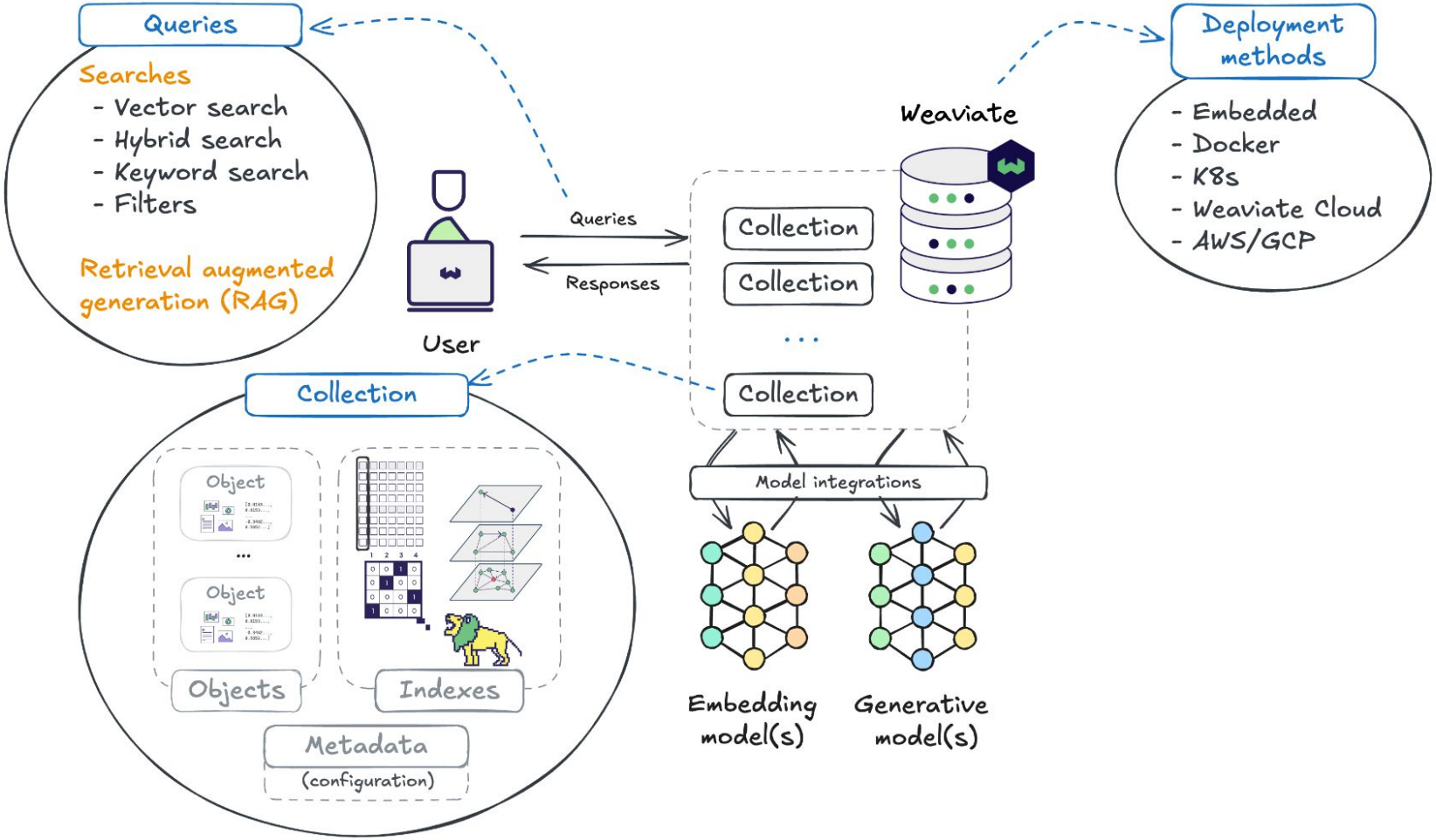
An introduction to vectors and vector DBs

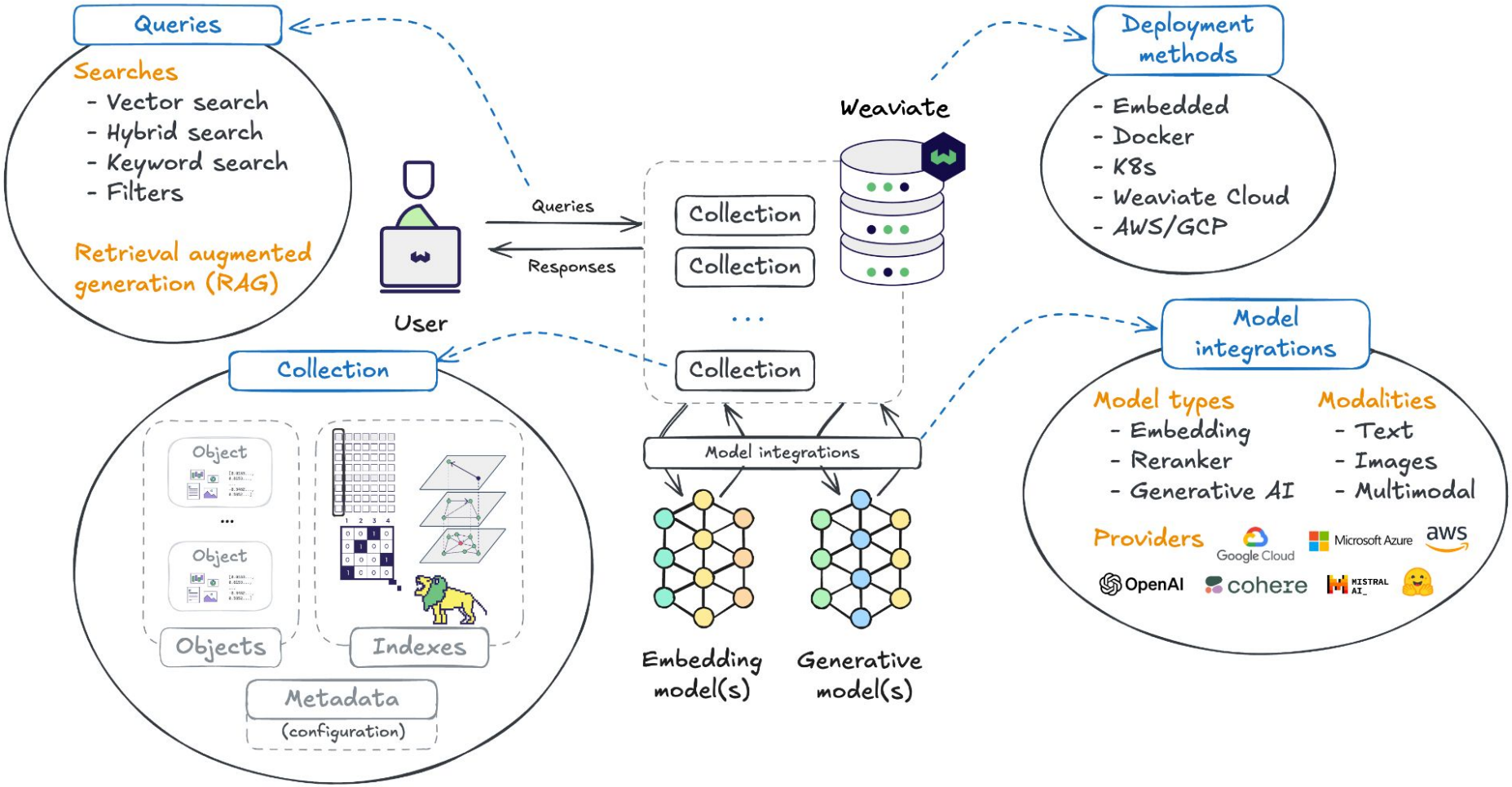

Weaviate
Queries
Responses
User
Collection
Collection
Collection


Weaviate
Queries
Responses
User
Collection
Collection
Collection
Model integrations
Embedding model(s)
Generative model(s)


Deployment methods
- Embedded
- Docker
- K8s
- Weaviate Cloud
- AWS/GCP
Weaviate
Queries
Responses
User
Collection
Collection
...
Collection
Model integrations
Embedding model(s)
Generative model(s)


Deployment methods
- Embedded
- Docker
- K8s
- Weaviate Cloud
- AWS/GCP
Weaviate
Queries
Responses
User
Collection
Collection
...
Collection
Collection
Object
...
Object
Objects
Indexes
Metadata
(configuration)
Model integrations
Embedding model(s)
Generative model(s)


Queries
Searches
- Vector search
- Hybrid search
- Keyword search
- Filters
Retrieval augmented generation (RAG)
User
Queries
Responses
Weaviate
Collection
Collection
...
Collection
Deployment methods
- Embedded
- Docker
- K8s
- Weaviate Cloud
- AWS/GCP
Collection
Object
...
Object
Objects
Indexes
Metadata
(configuration)
Model integrations
Embedding model(s)
Generative model(s)


Queries
Searches
- Vector search
- Hybrid search
- Keyword search
- Filters
Retrieval augmented generation (RAG)
User
Queries
Responses
Weaviate
Collection
Collection
...
Collection
Deployment methods
- Embedded
- Docker
- K8s
- Weaviate Cloud
- AWS/GCP
Collection
Object
...
Object
Objects
Indexes
Metadata
(configuration)
Model integrations
Embedding model(s)
Generative model(s)
Model integrations
Model types
- Embedding
- Reranker
- Generative AI
Modalities
- Text
- Images
- Multimodal
Providers
Google Cloud
Microsoft Azure
aws
OpenAI
cohere
MISTRAL AI_

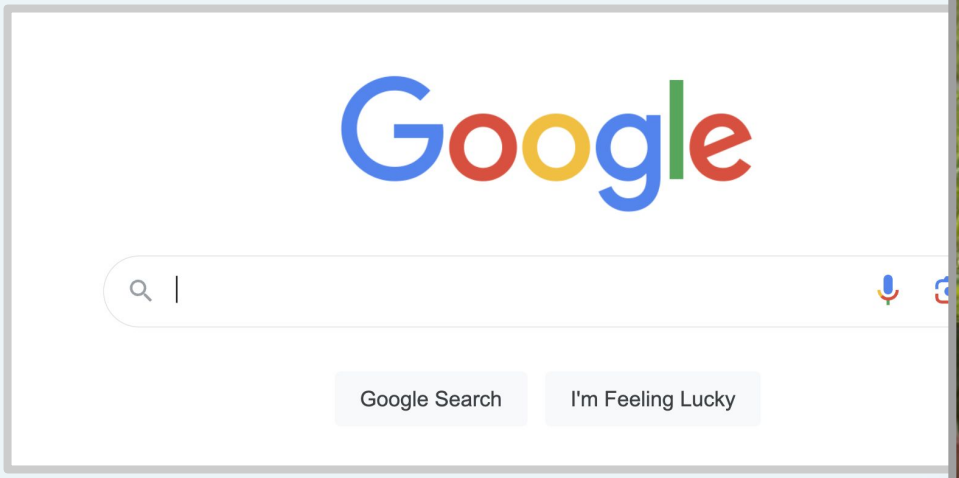
A quick demo

Where we are headed

Everything we do
starts with search

Why is search so ubiquitous?

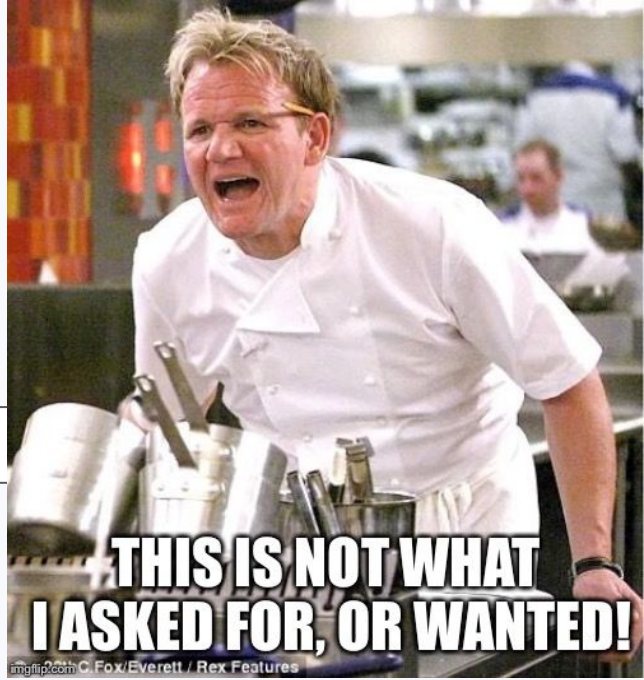
We are spoiled



The problem of *effective search*

Traditional search has shortcomings

How do airplanes fly?

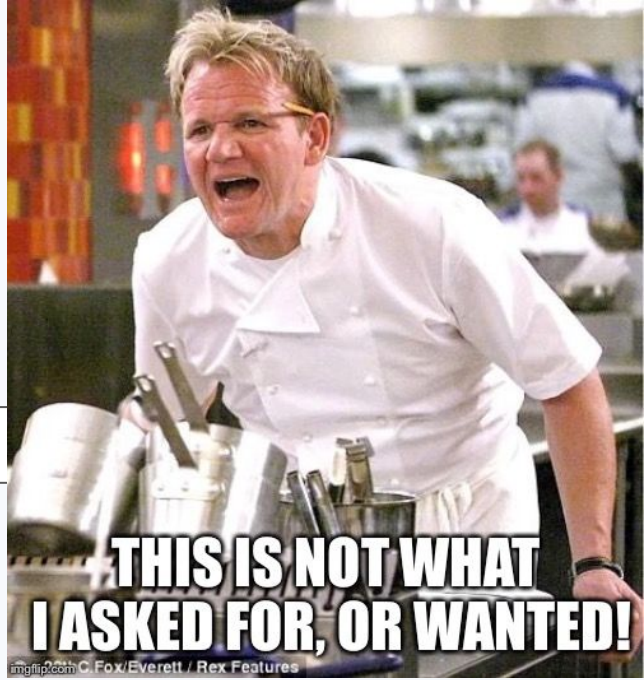


Why you should **fly** with ExpensiveAir's **airplanes**!

...fly with ExpensiveAir's modern fleet of **airplanes** to over 4000 destinations. **How** to book: ...

Traditional search has shortcomings

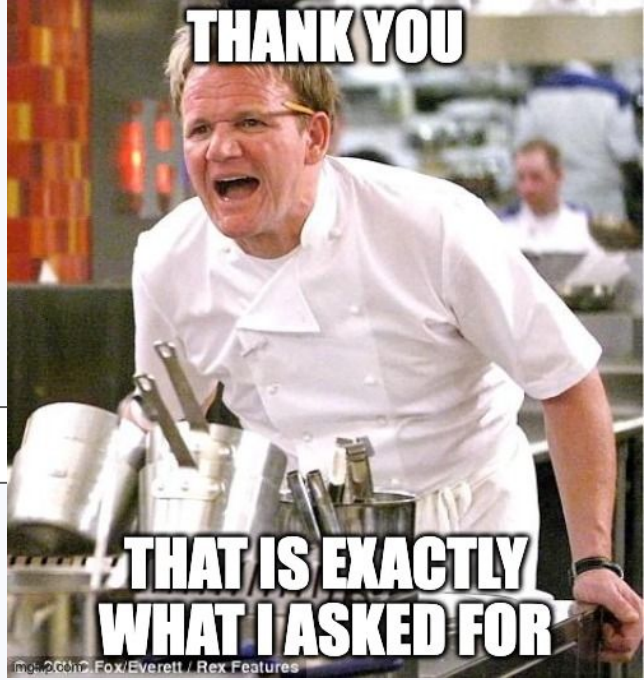
How do airplanes fly?



Why you should fly with ExpensiveAir's airplanes!
...fly with ExpensiveAir's modern fleet of airplanes to over 4000 destinations. How to book: ...

Semantic search to the rescue!

How do airplanes fly?



[Dynamics of Flight - NASA]

Airplane wings are shaped **to make air move faster over the top of the wing**. When air moves faster, the pressure of the air decreases.

How does this work?

Vectors!

What is a vector?

A vector is a set of numbers

That represents some meaning

Like

```
[1, 0]
```

or

```
[0.513, 0.155, 0.983, 0.001, 0.932]
```

or

```
[0.0009420722, 0.020158706, -0.03939992, 0.018441677, …, -0.025270471, -0.056622636, …]
```

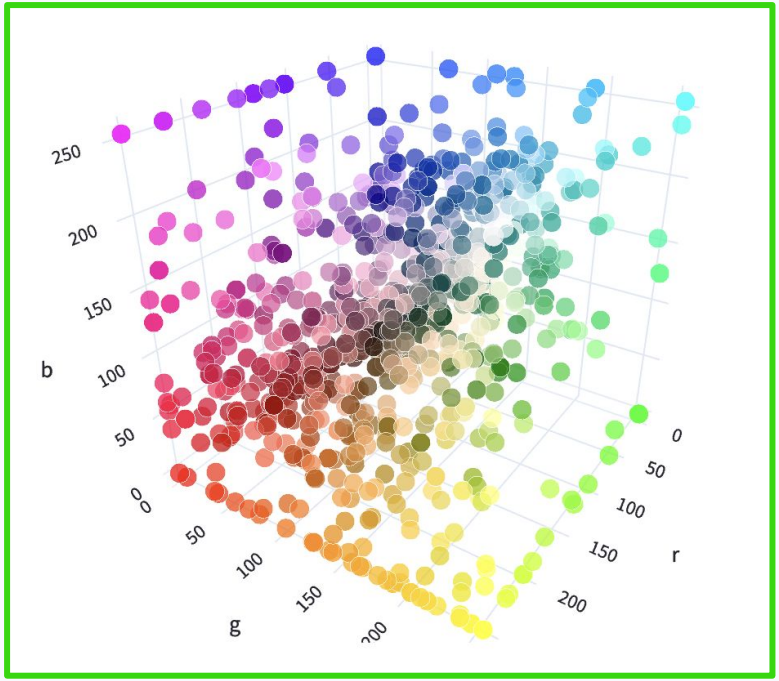
Numbers represent meaning?

Here's an example:

RGB numbers represent colors, like:

(255, 0, 0) = red
(80, 200, 120) = emerald

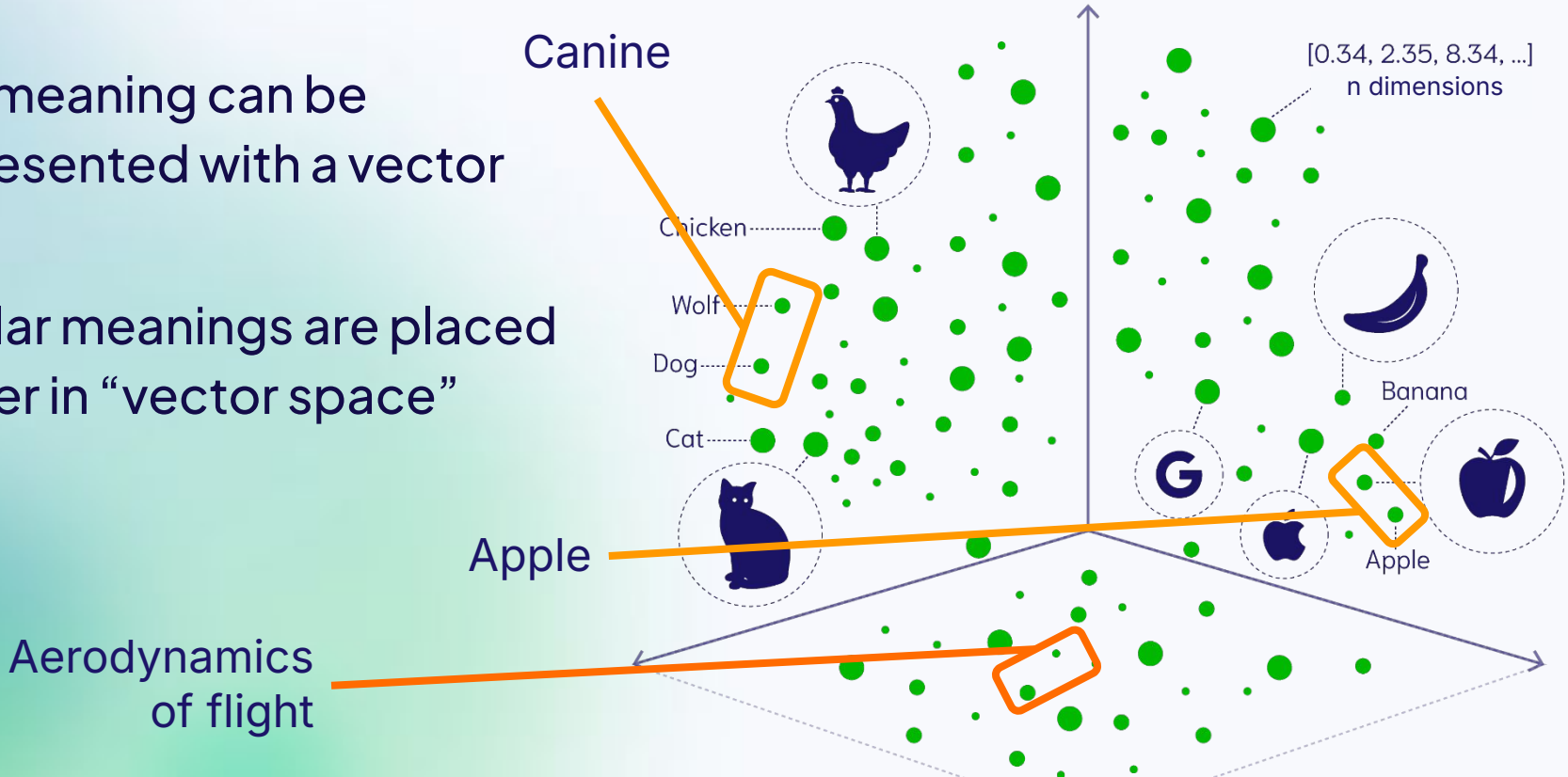
Each number is a dial
for (red, green, blue) ness.



Numbers represent meaning

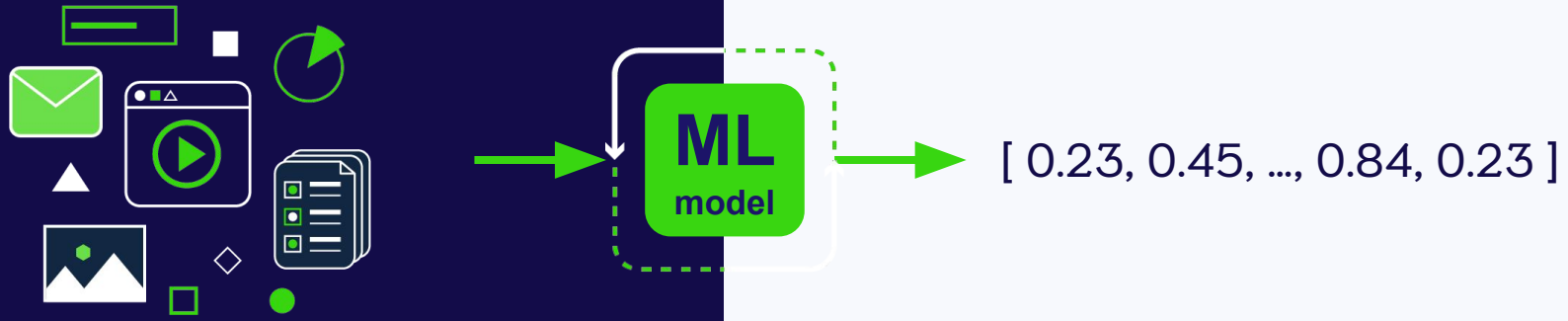
Any meaning can be represented with a vector

Similar meanings are placed closer in “vector space”



Where do vectors come from?

Machine Learning models



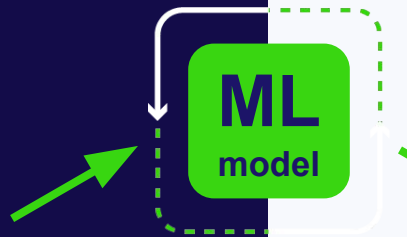
Vector Embeddings

"Fly with ExpensiveAir's modern fleet of airplanes to over 4000 destinations. How to book..."

Vector Embeddings

"Fly with ExpensiveAir's modern fleet of airplanes to over 4000 destinations. How to book..."

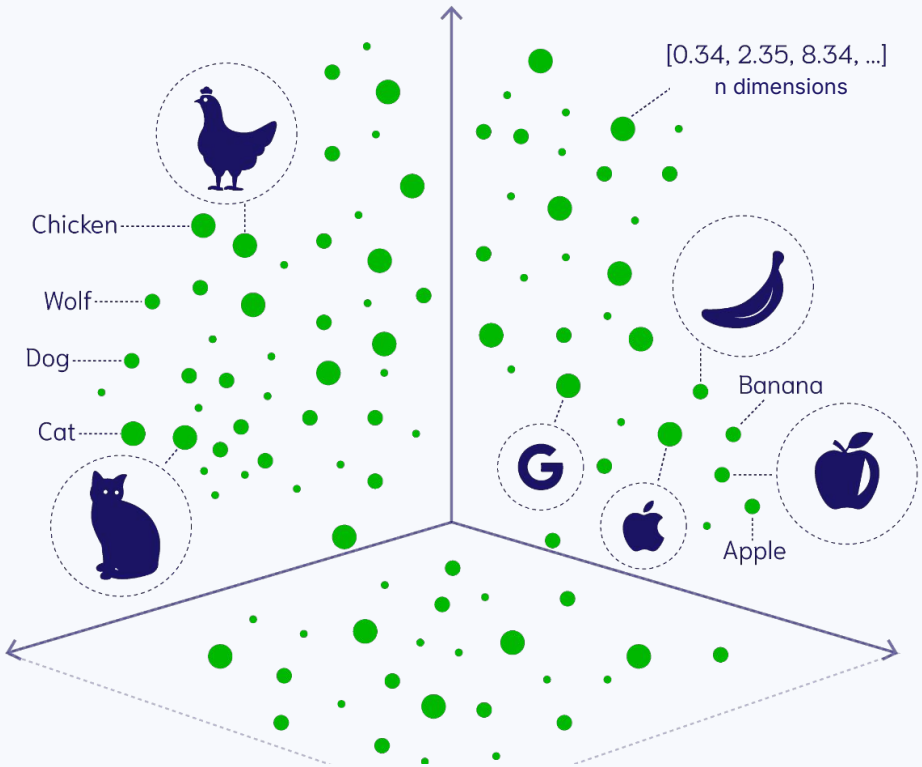
Airplane wings are shaped to make air move faster over the top of the wing. When air moves faster, the pressure of the air decreases.



[0.23, 0.45, ..., 0.84, 0.23]

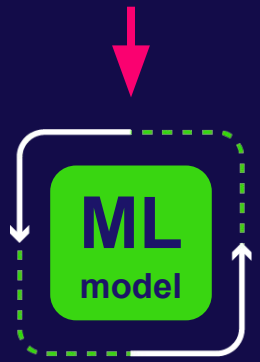
[0.26, 0.31, ..., 0.12, 0.44]

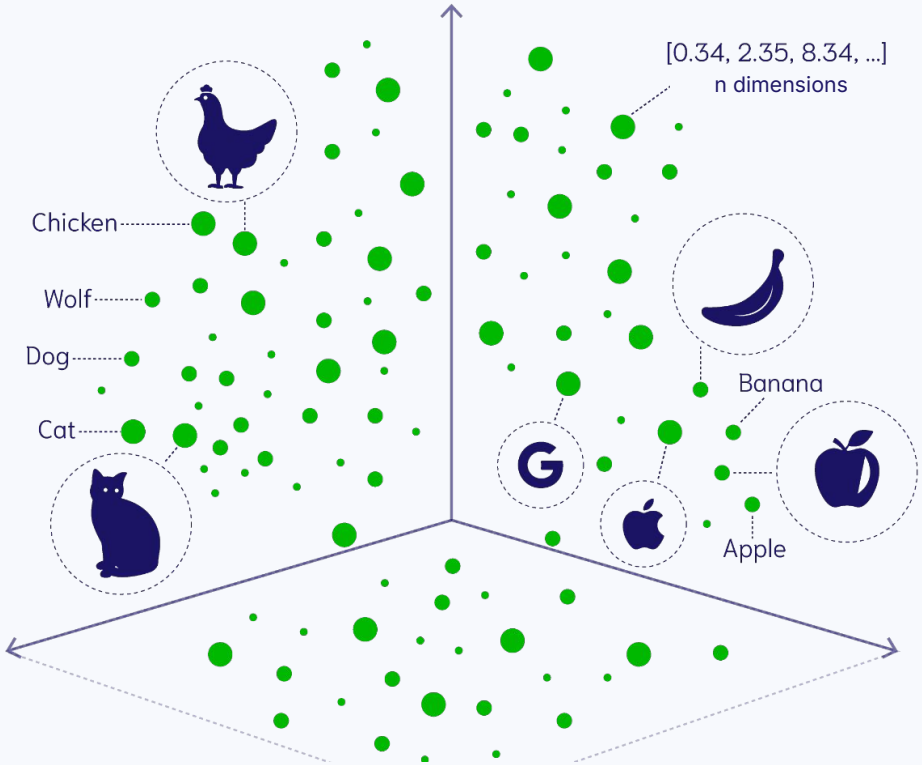
Vector Space



Vector Search

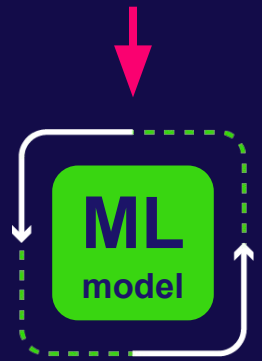
How do airplanes fly?



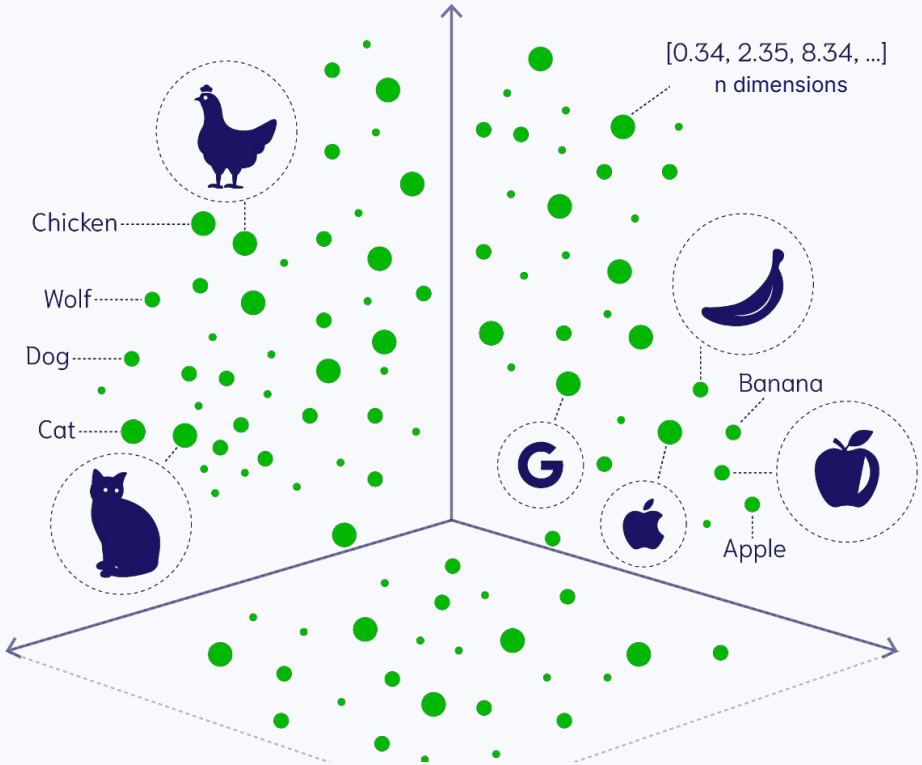


Vector Search

How do airplanes fly?



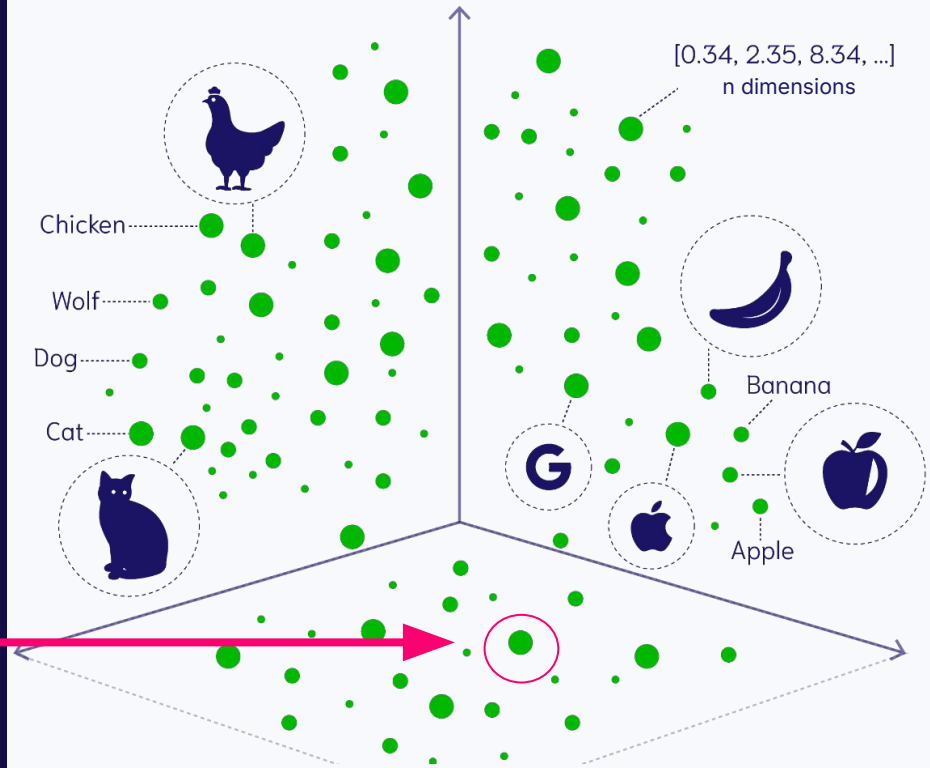
[0.24, 0.36, ..., 0.16, 0.46]



Vector Search

How do airplanes fly?

[0.24, 0.36, ..., 0.16, 0.46]



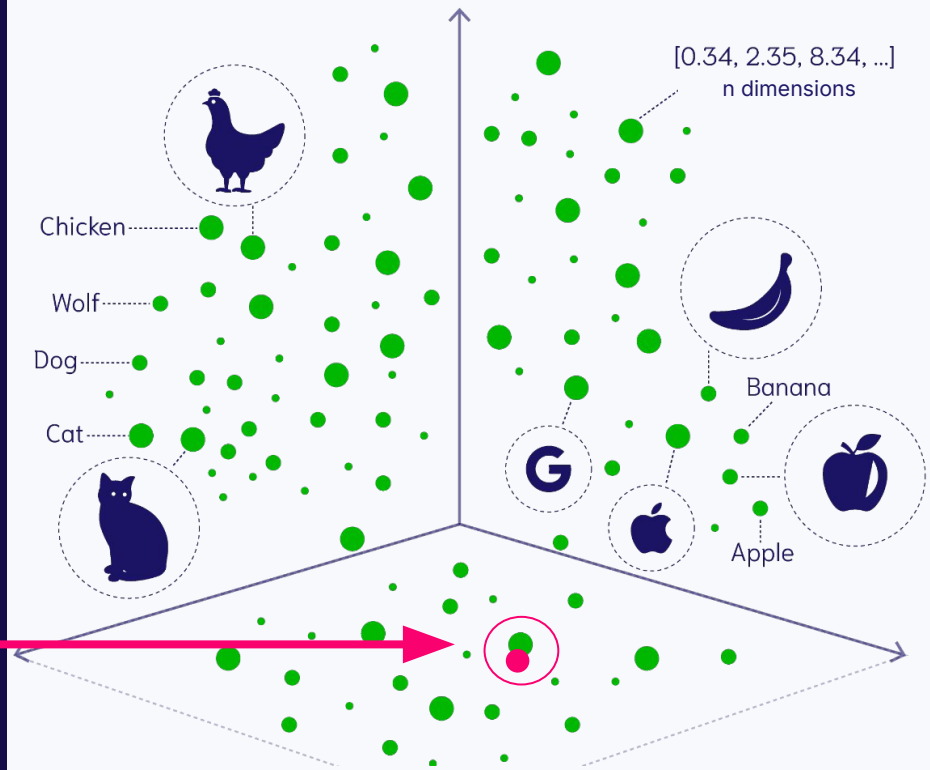
Vector Search

How do airplanes fly?

[0.24, 0.36, ..., 0.16, 0.46]

[0.26, 0.31, ..., 0.12, 0.44]

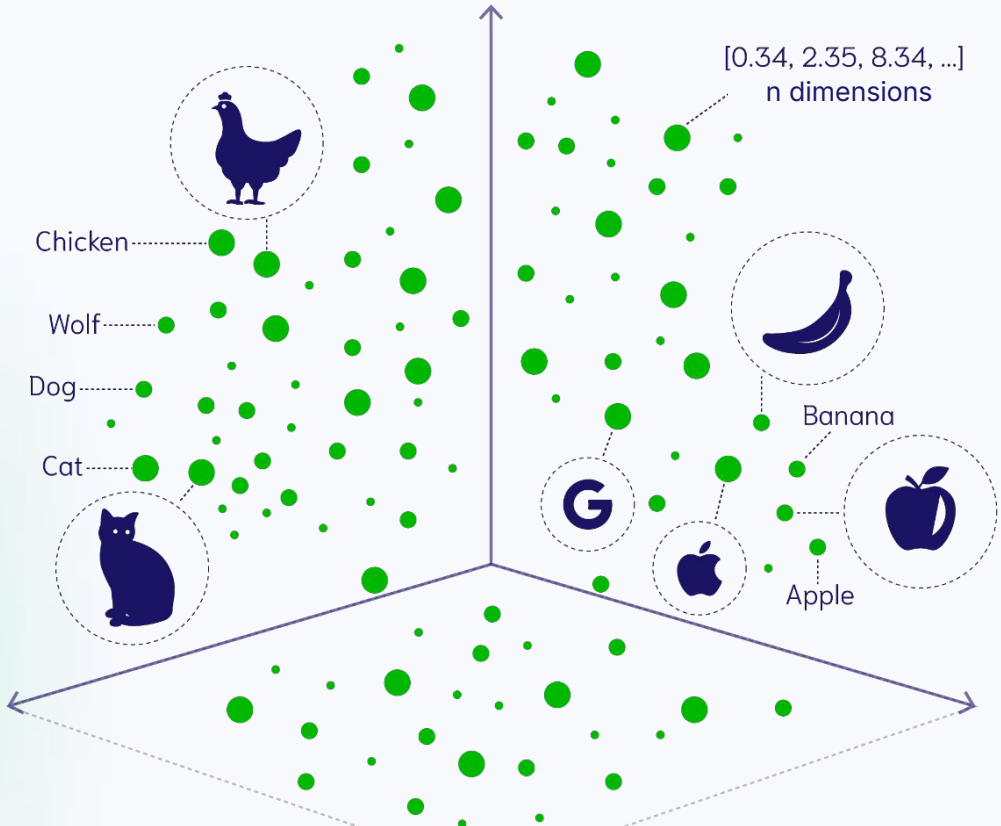
The NASA article



What is a vector database?

A database whose primary **data organization** is by meaning (vectors)

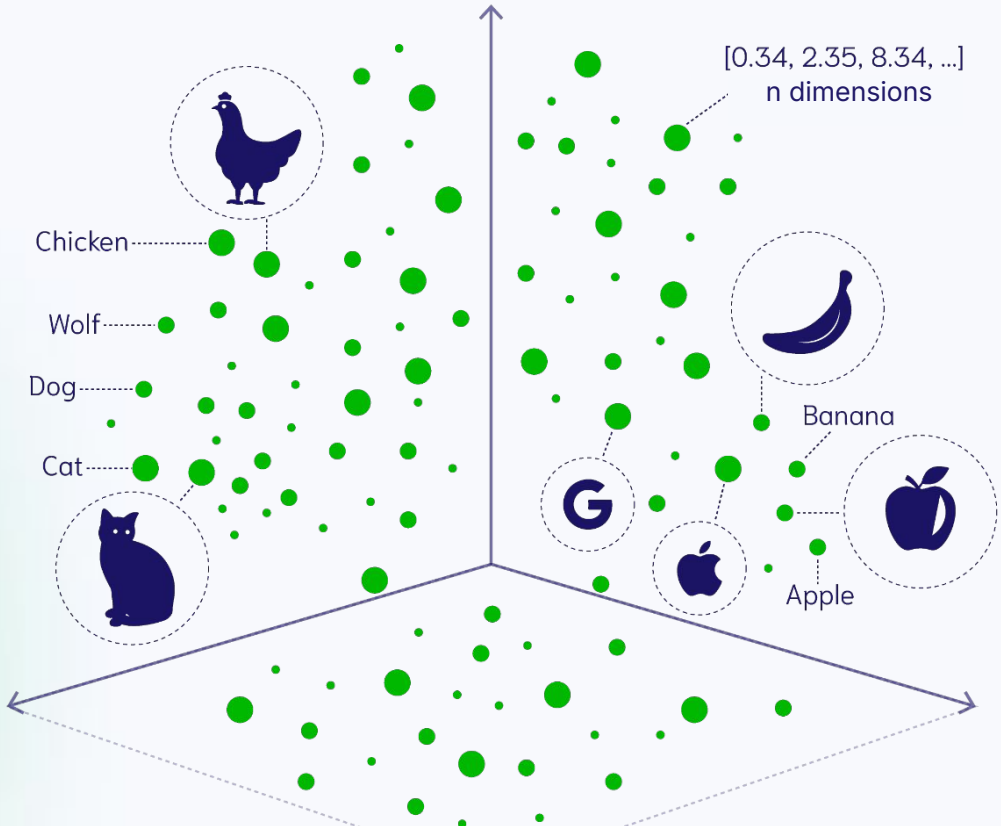
And optimized for data management and queries for using vectors **at scale.** (billions)



What is an AI-native database?

Vectors and AI-native workflow as first-class citizens

- AI model integrations
- Hybrid search
- Scalability (multi-tenancy, quantization)
- Developer experience



Better search

Use semantic, keyword and hybrid searches
to get the right data, faster

Keyword, Vector or Hybrid?

Differences between search types

Search: example

Source documents

Ad for an airline

"Come and fly with ExpensiveAir's modern fleet of airplanes to over 4000 destinations. How to book a flight with ExpensiveAir?..."

Explains aerodynamics

"Airplane wings are shaped to make air move faster over the top of the wing. When air moves faster, the pressure of the air decreases."

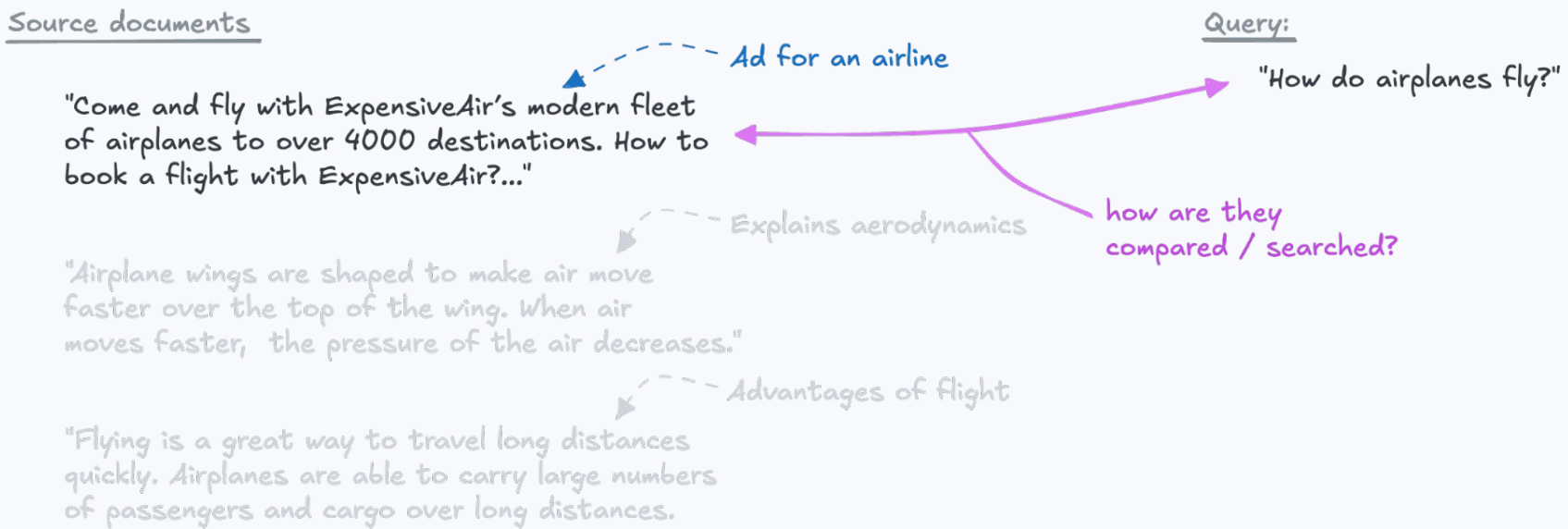
Advantages of flight

"Flying is a great way to travel long distances quickly. Airplanes are able to carry large numbers of passengers and cargo over long distances.

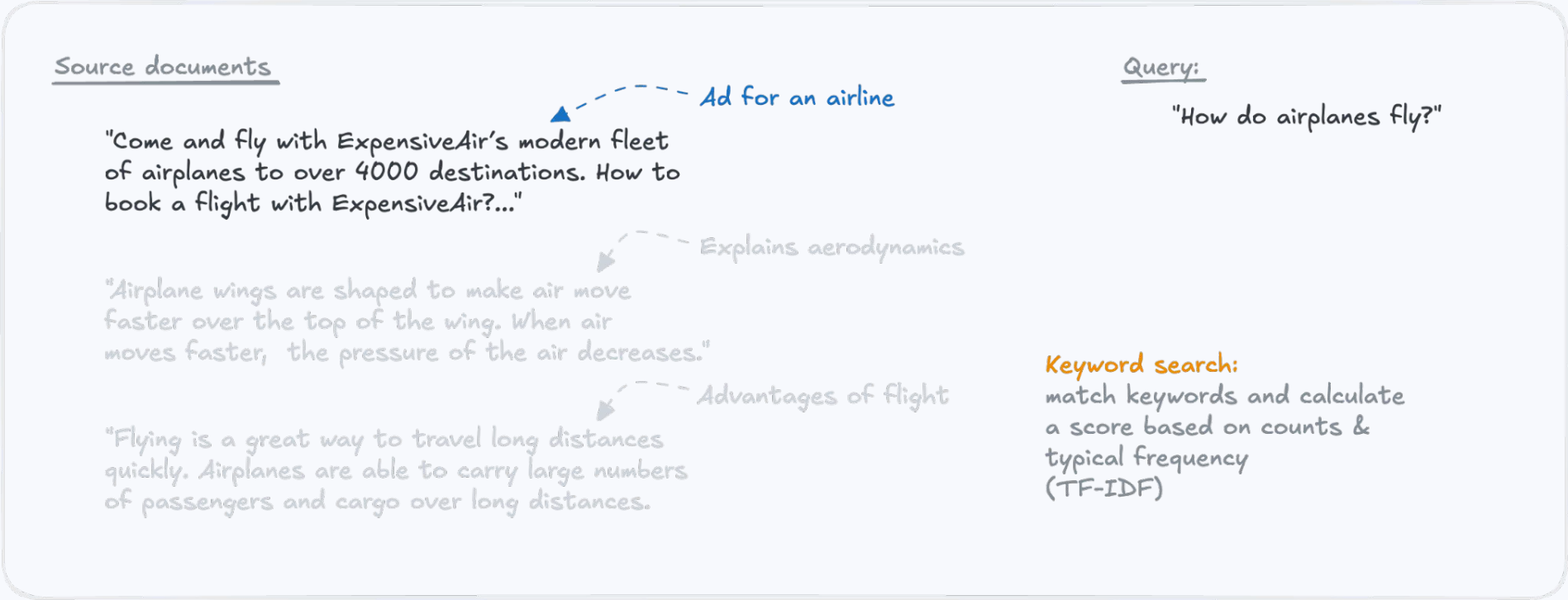
Query:

"How do airplanes fly?"

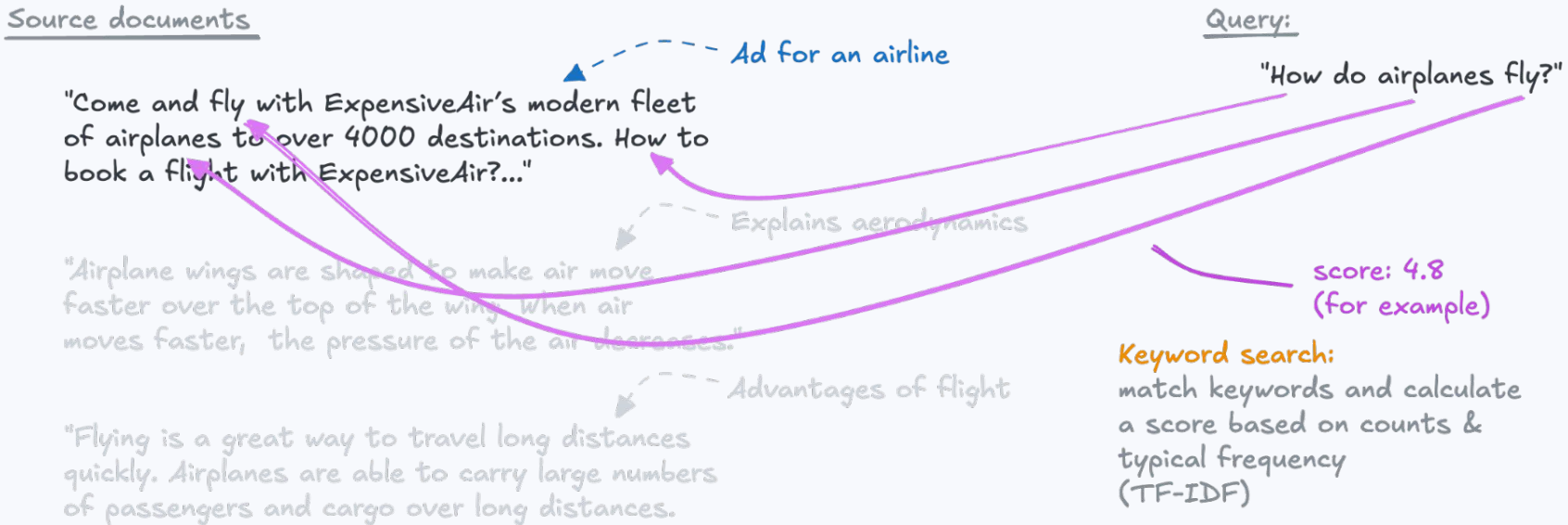
Search: example



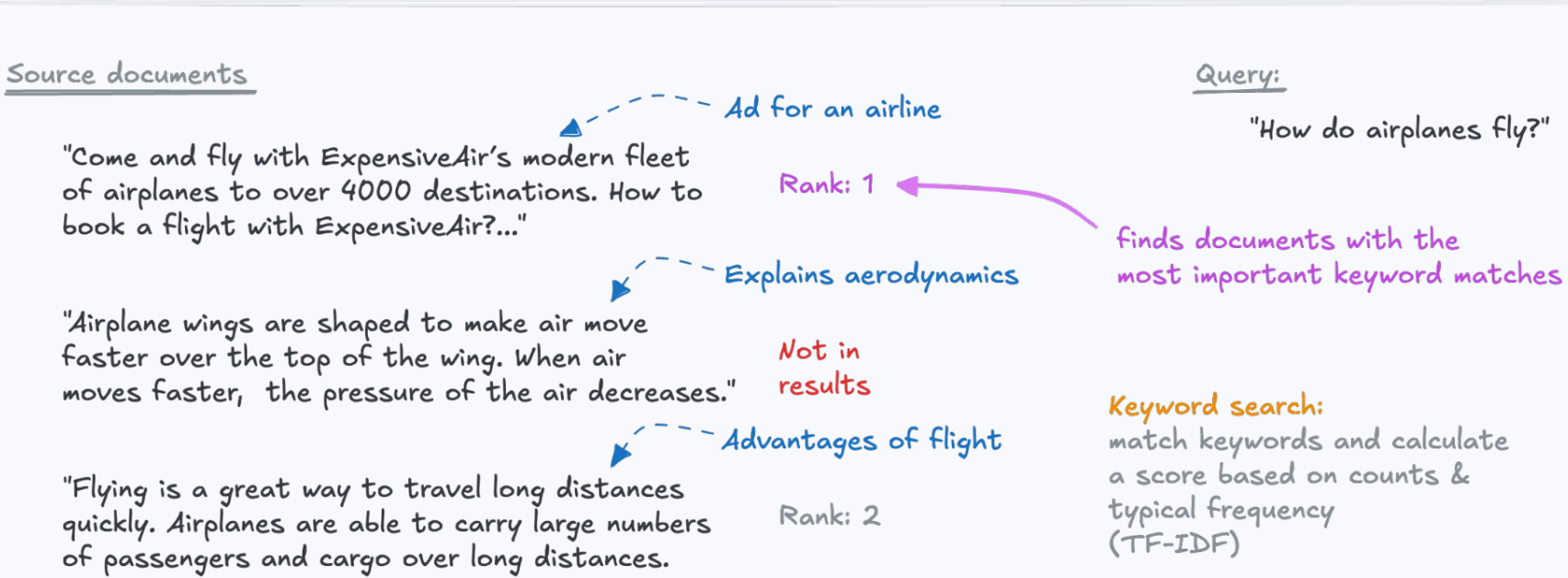
Keyword search (BM25)



Keyword search (BM25)



Keyword search results



When to use...

Keyword search

When exact matches important.

Examples:

- Domain-specific data (jargon)
- Product / part numbers
- URL matches

Keyword search syntax

```
products = client.collections.get("Product")

response = products.query.bm25(
    "How do airplanes fly?",
    limit=2
)
```

Vector search

Source documents

Ad for an airline

"Come and fly with ExpensiveAir's modern fleet of airplanes to over 4000 destinations. How to book a flight with ExpensiveAir?..."

Explains aerodynamics

"Airplane wings are shaped to make air move faster over the top of the wing. When air moves faster, the pressure of the air decreases."

Advantages of flight

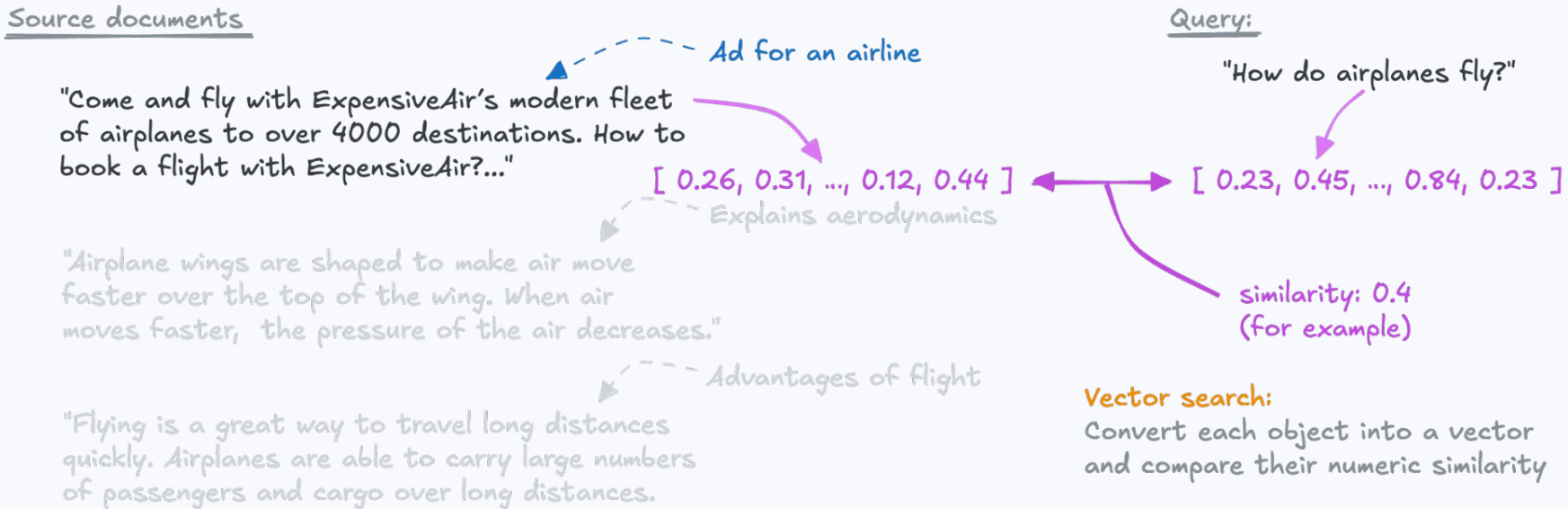
"Flying is a great way to travel long distances quickly. Airplanes are able to carry large numbers of passengers and cargo over long distances.

Query:

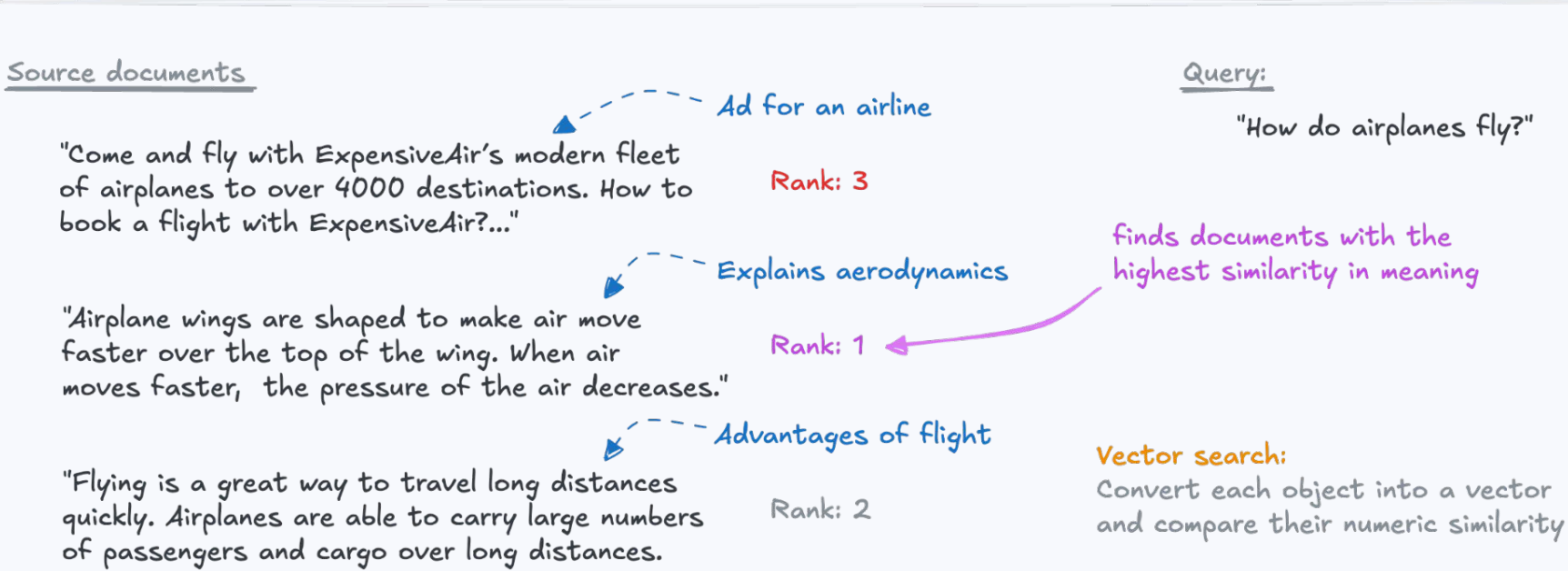
"How do airplanes fly?"

Vector search:
Convert each object into a vector and compare their numeric similarity

Vector search



Vector search results



When to use...

Vector search

When robustness is important, or cross-lingual or cross-modal search is desired.

Examples:

- Descriptions
- Image to text / image to image
- Product reviews

Vector search syntax

```python
products = client.collections.get("Product")

response = products.query.near_text(
    "How do airplanes fly?",
    limit=2
)
```

Hybrid search

Source documents

Ad for an airline

"Come and fly with ExpensiveAir's modern fleet of airplanes to over 4000 destinations. How to book a flight with ExpensiveAir?..."

Explains aerodynamics

"Airplane wings are shaped to make air move faster over the top of the wing. When air moves faster, the pressure of the air decreases."

Advantages of flight

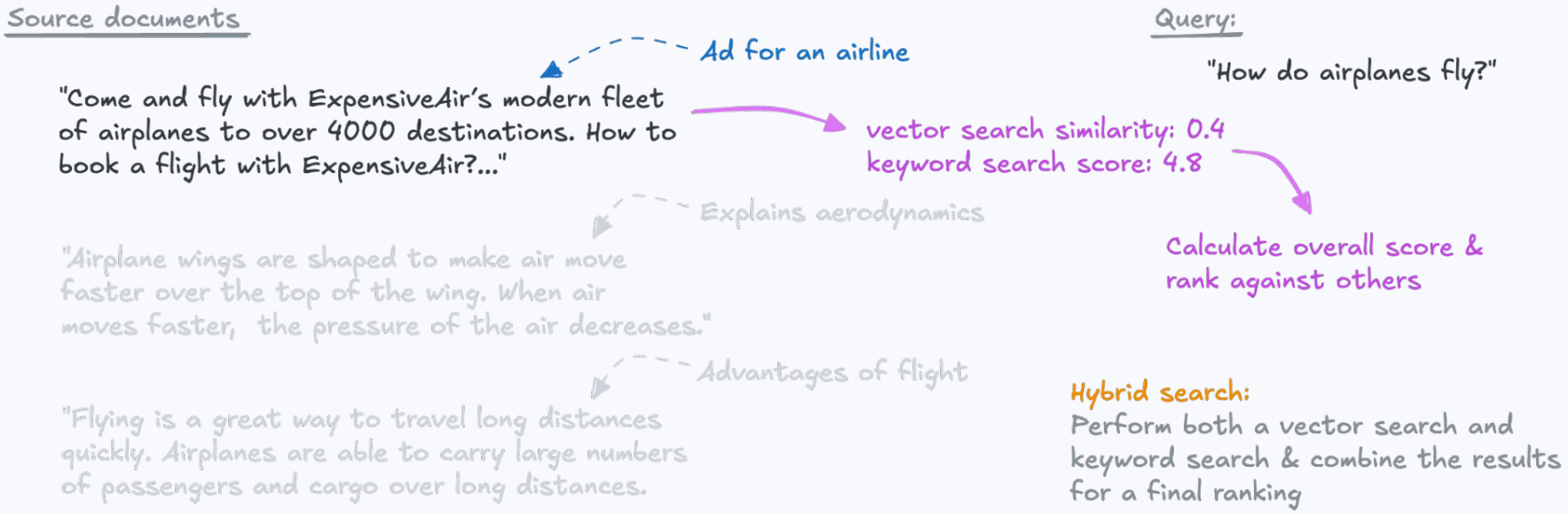
"Flying is a great way to travel long distances quickly. Airplanes are able to carry large numbers of passengers and cargo over long distances.

Query:

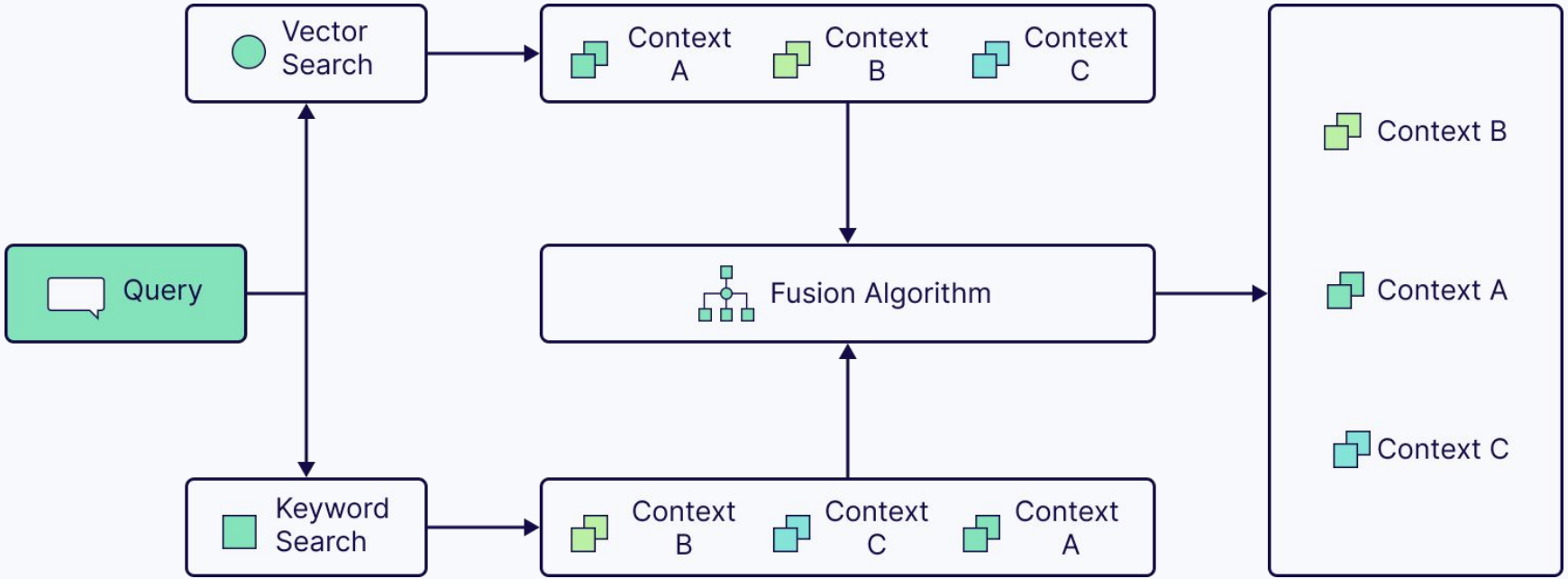
"How do airplanes fly?"

Hybrid search:
Perform both a vector search and keyword search & combine the results for a final ranking

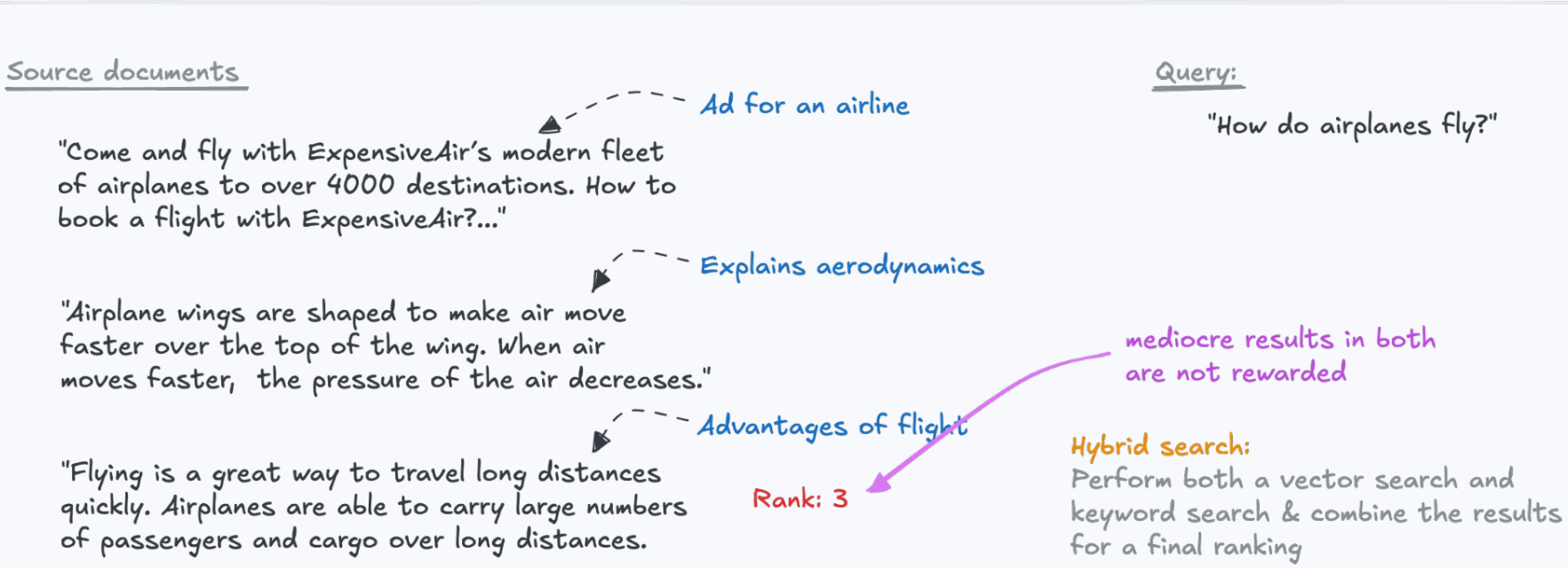
Hybrid search



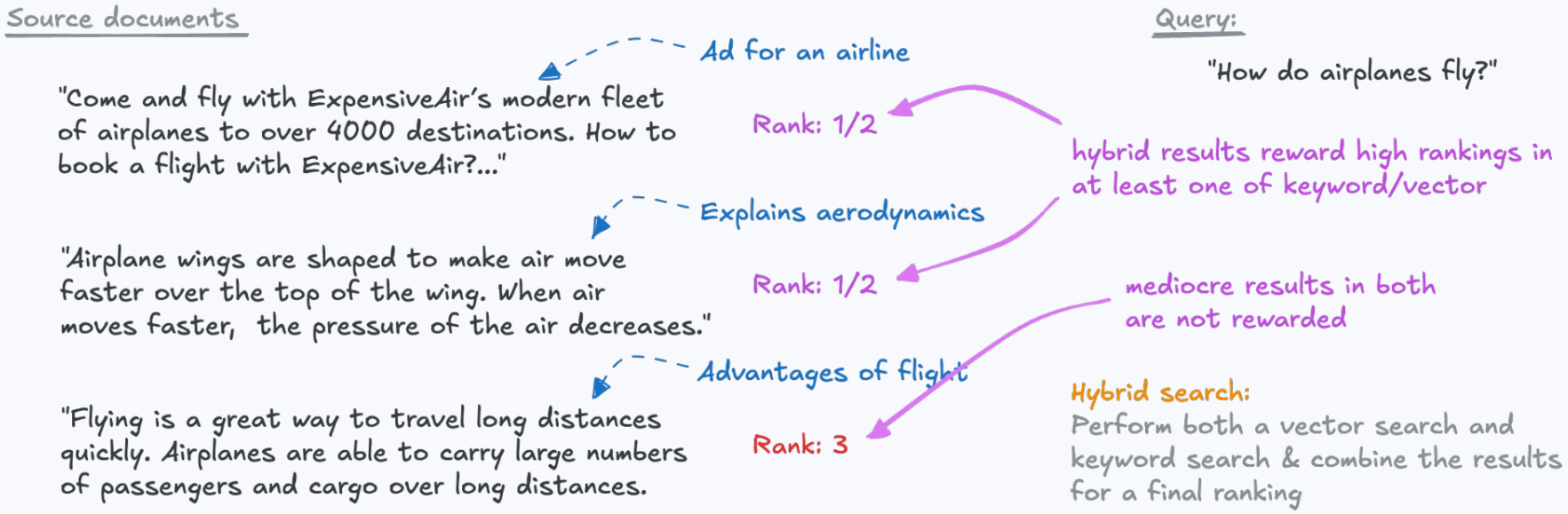
Hybrid Search



Hybrid search results



Hybrid search results



When to use...

Hybrid search

Best-of-both-worlds search. Surfaces top results from both keyword & vector searches

Good results reported “in the wild” from users due to resilience.

Hybrid search syntax

```python
products = client.collections.get("Product")

response = products.query.hybrid(
    "How do airplanes fly?",
    limit=2,
    alpha=0.5
)
```

Quick demos

Vector search: in depth

A deeper dive

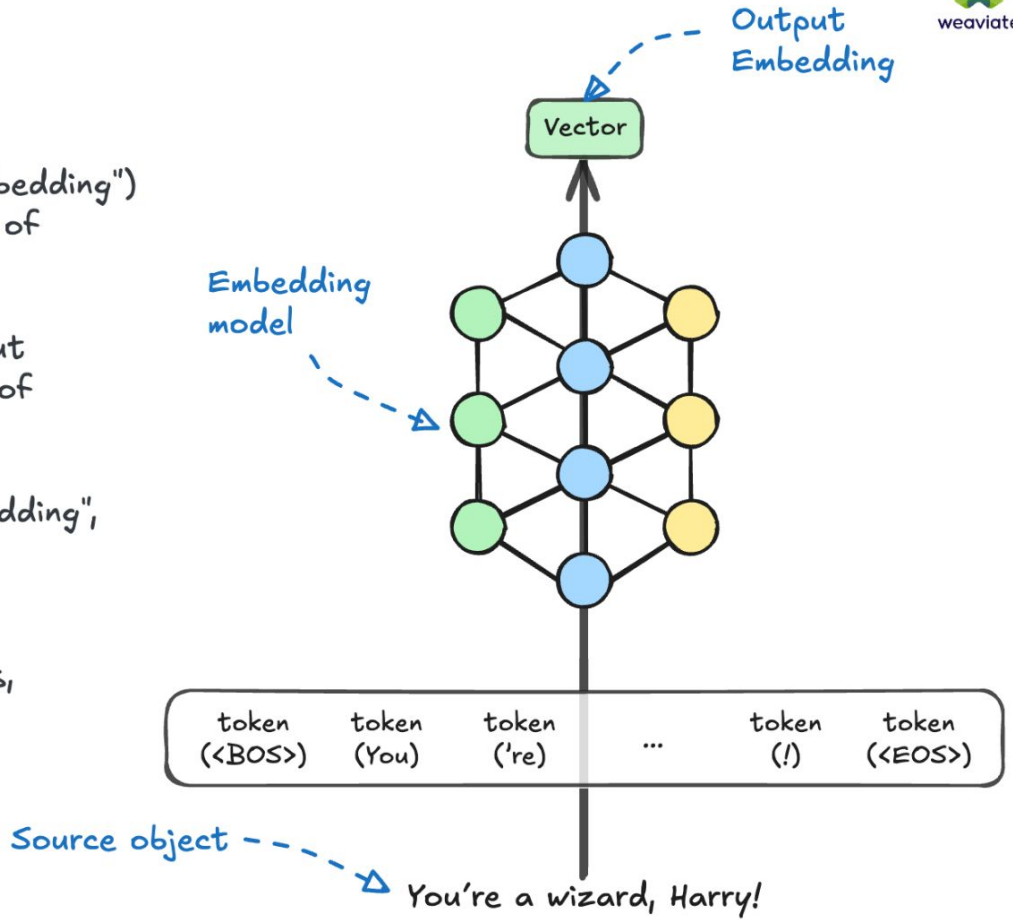
How embeddings are created

A vector embedding (a.k.a. "vector" or "embedding") captures meaning of an object as a series of numbers.

An embedding is created by passing an input into an "embedding model", a specific form of a deep learning or AI model.

The model then produces an output "embedding", which looks like this:
[0.0134, 0.8723, -0.4532, ..., 0.5892]

The more similar the meaning of the inputs, the more similar these embeddings will be.



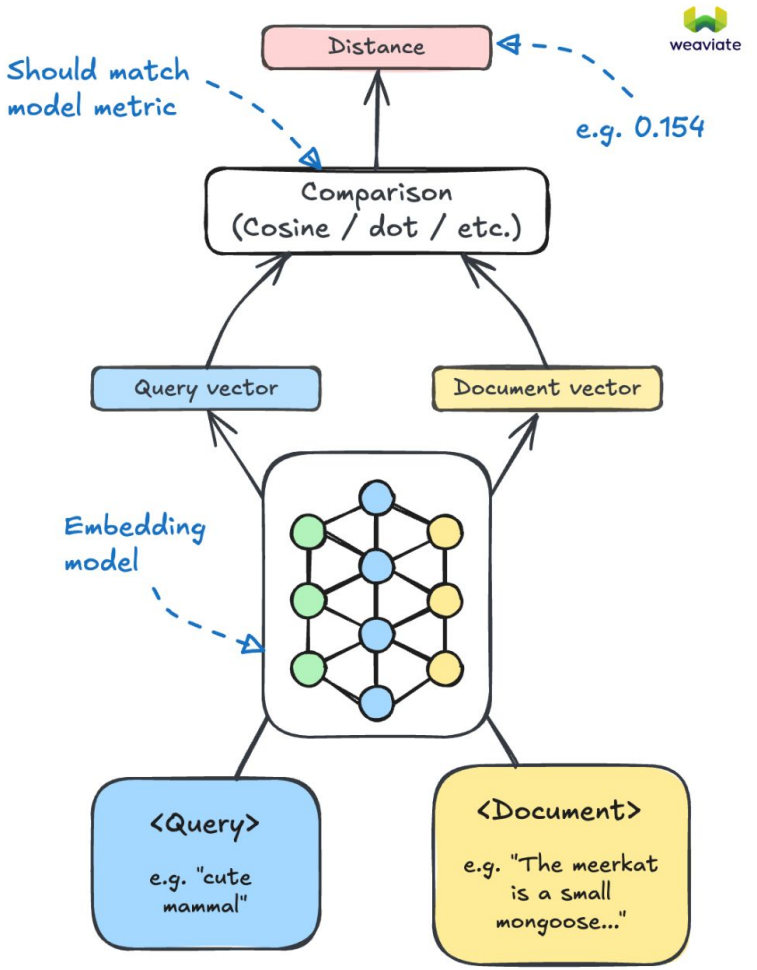
How embeddings are compared

Vector embeddings capture meanings numerically. As a result, two embeddings can be compared to produce a measure of similarity.

The result is often measured as a "distance", measured using the same distance metric that the embedding model is trained on.

A "distance" indicates dissimilarity in meaning.

Two objects with a larger distance are more dissimilar than two objects with a smaller distance.



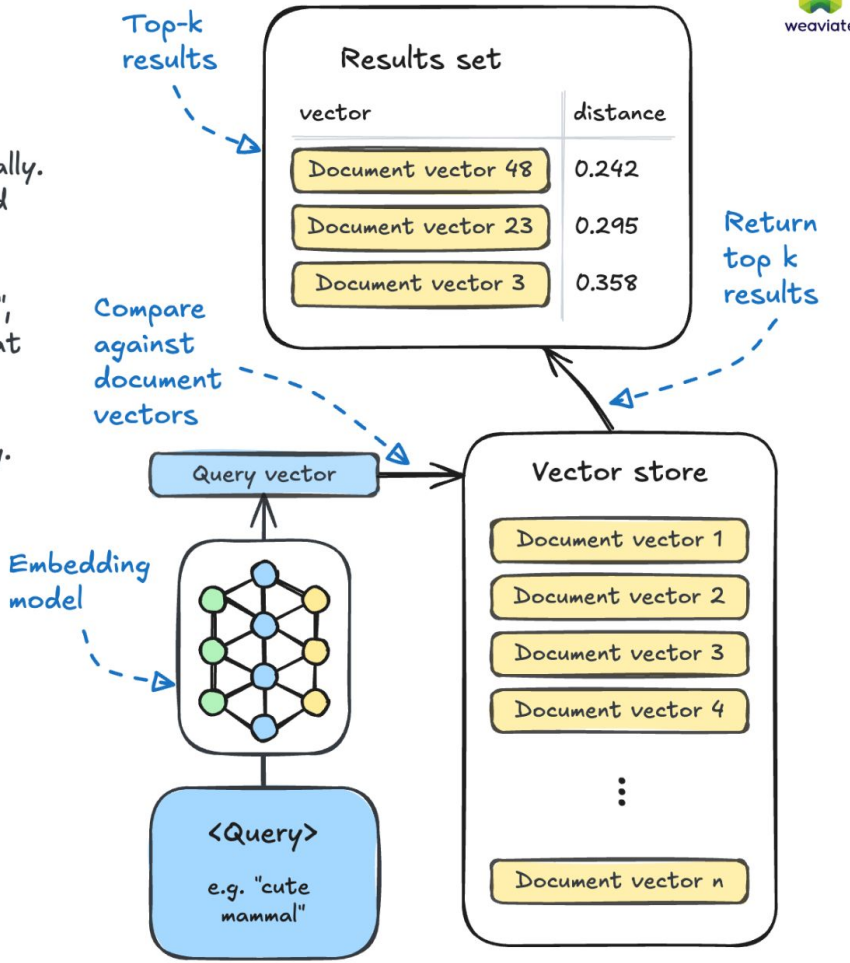
How vector search works

Vector embeddings capture meanings numerically. As a result, two embeddings can be compared to produce a measure of similarity.

The result is often measured as a "distance", measured using the same distance metric that the embedding model is trained on.

A "distance" indicates dissimilarity in meaning.

Two objects with a larger distance are more dissimilar than two objects with a smaller distance.



How do we search at scale?

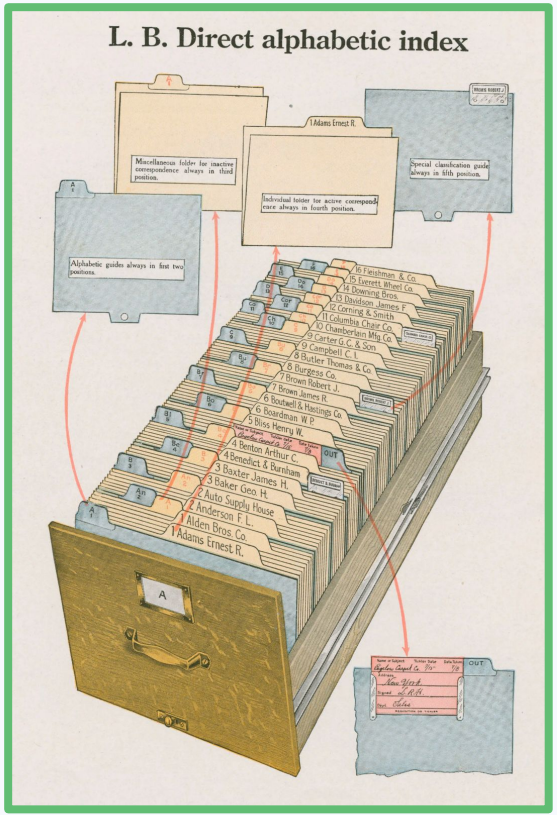
What happens as our dataset grows larger?

- How long does vector search take at:
  - 10,000 objects?
  - 10,000,000 objects?
  - 10,000,000,000 objects?
- And what are the typical bottlenecks?

Indexes

Used by databases for speed

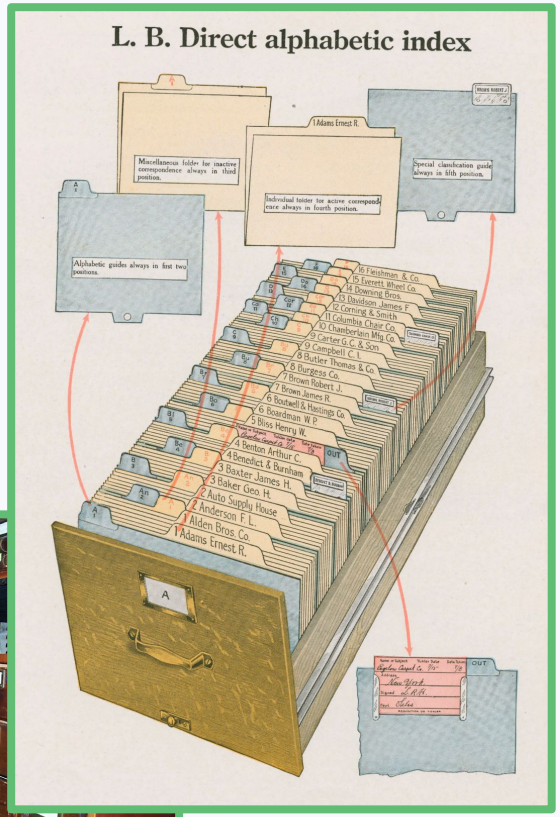
- e.g. Library catalogues



Indexes

In databases:

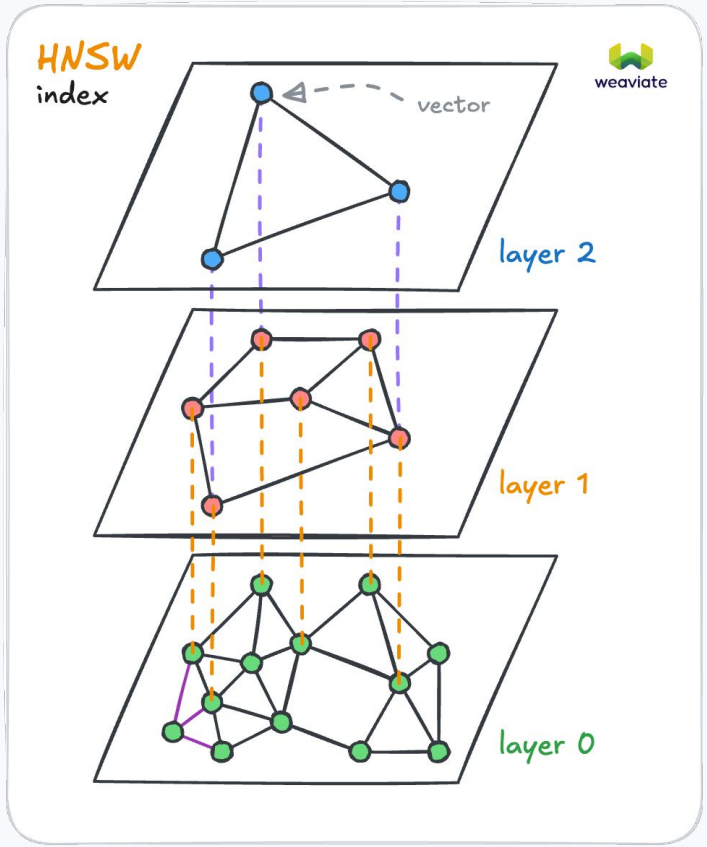
- **Catalogue data**
- **Speed up search & filtering**



Vector indexing (ANN)

A way to scale search
up to billions of vectors.

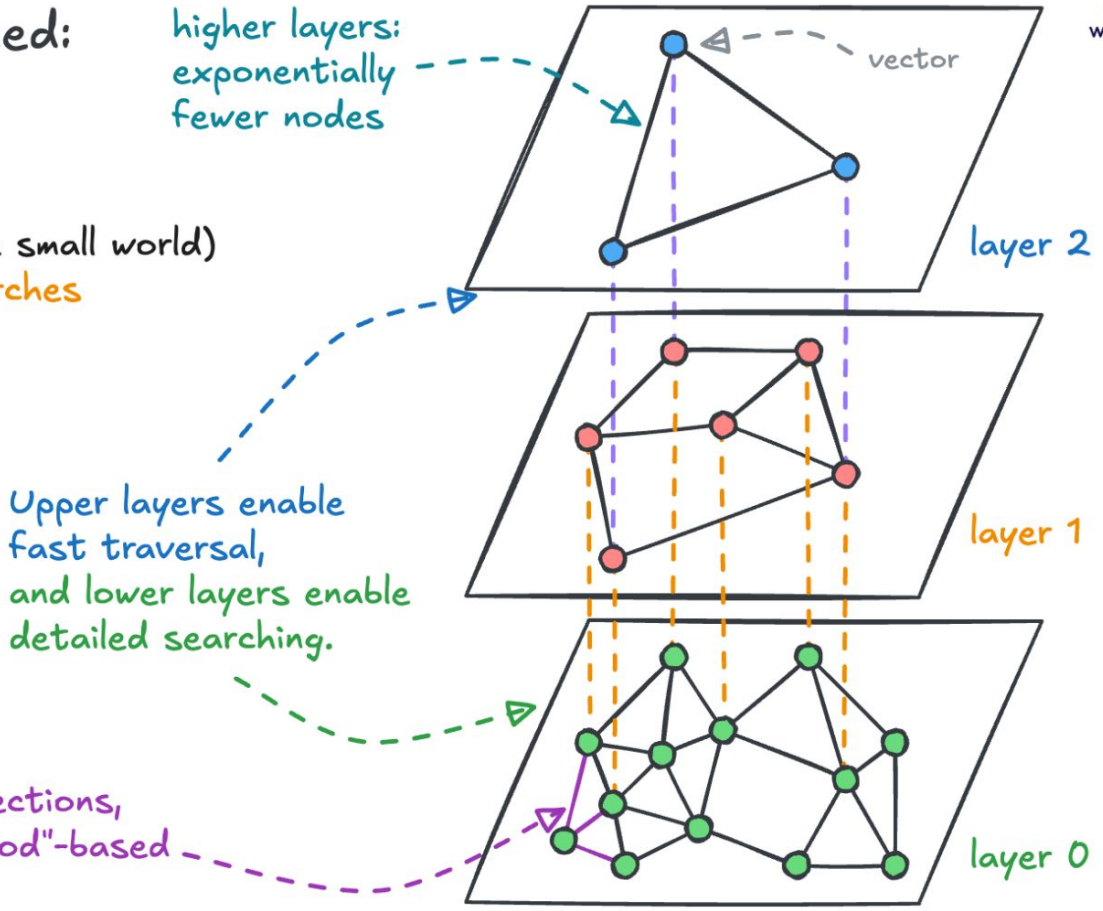
Most common index type:



Vector indexes explained:
HNSW

An HNSW (hierarchical navigable small world) index enables fast vector searches even at large sizes.

The HNSW algorithm builds a multi-layer graph that balances speed & search quality.



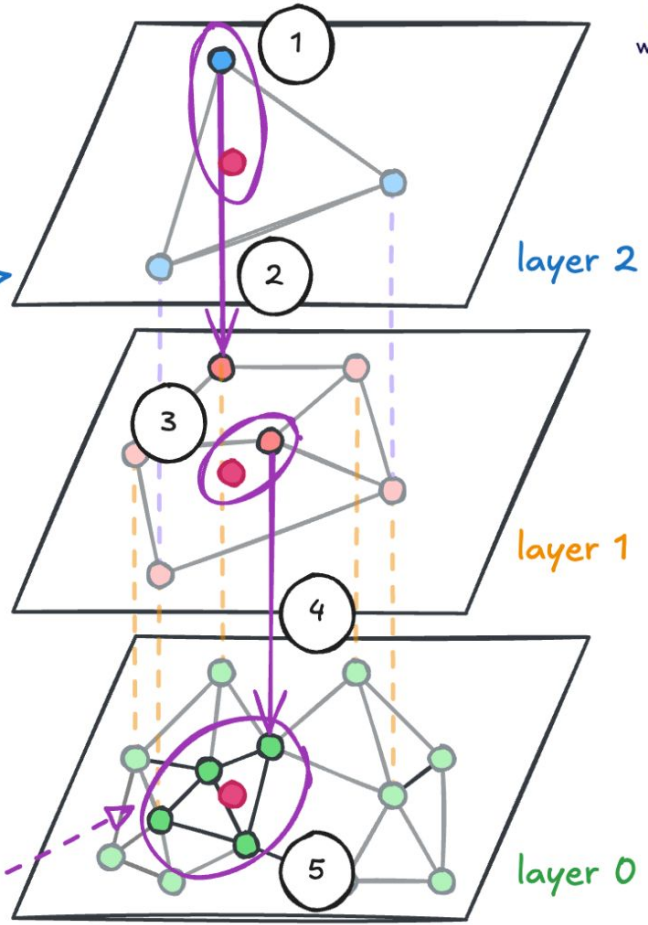
Vector indexes explained:
HNSW search

Vector search in HNSW proceeds by starting at the top layer, and proceeding down.

How would HNSW search for this vector: ● work?

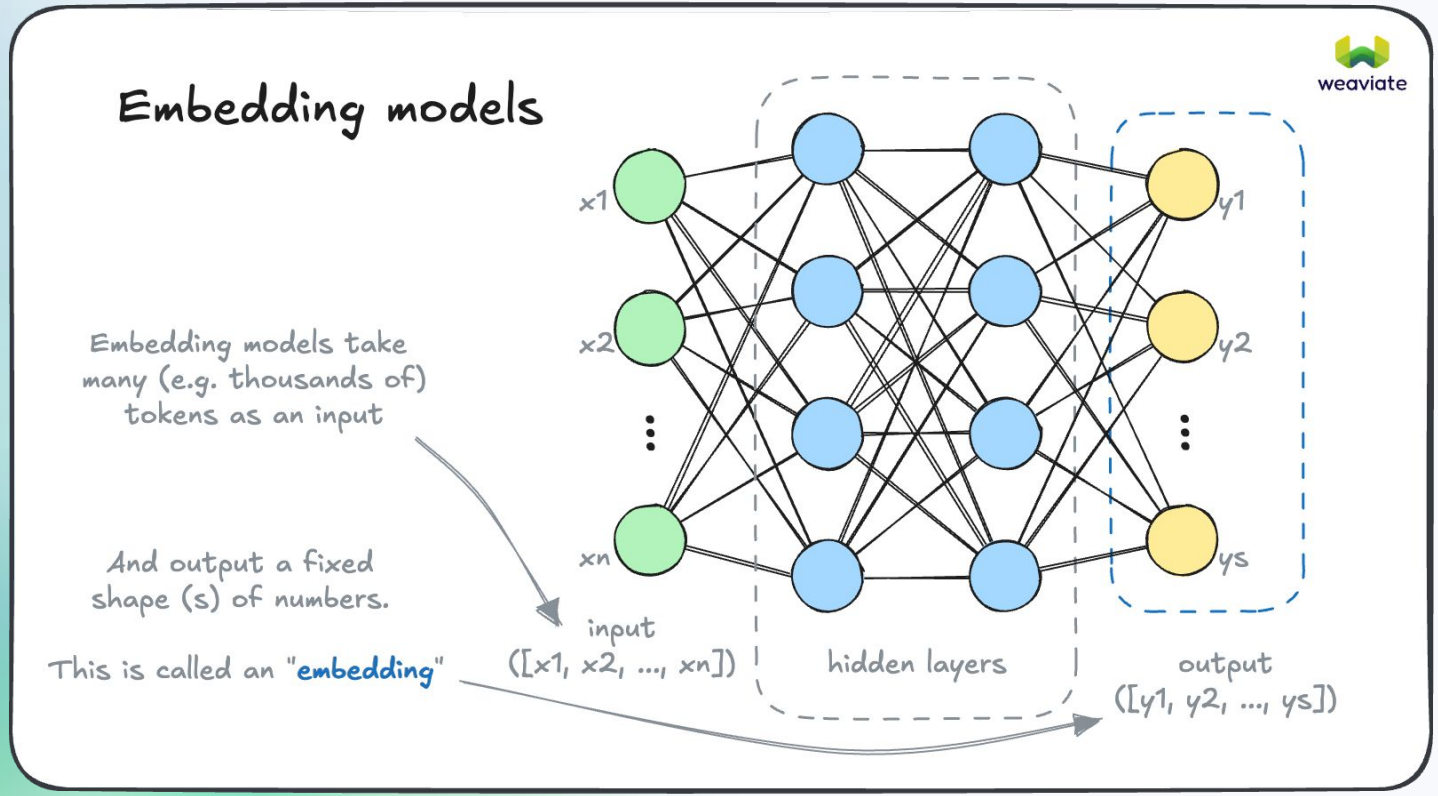
Starting from the top layer,

1: find the most similar node,
2: travel down
3-4: repeat until bottom layer
5: then, find the closest k nodes

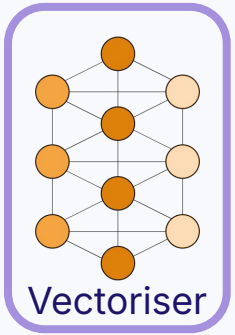


Where embeddings come from

Vectoriser: input ➡ vectors



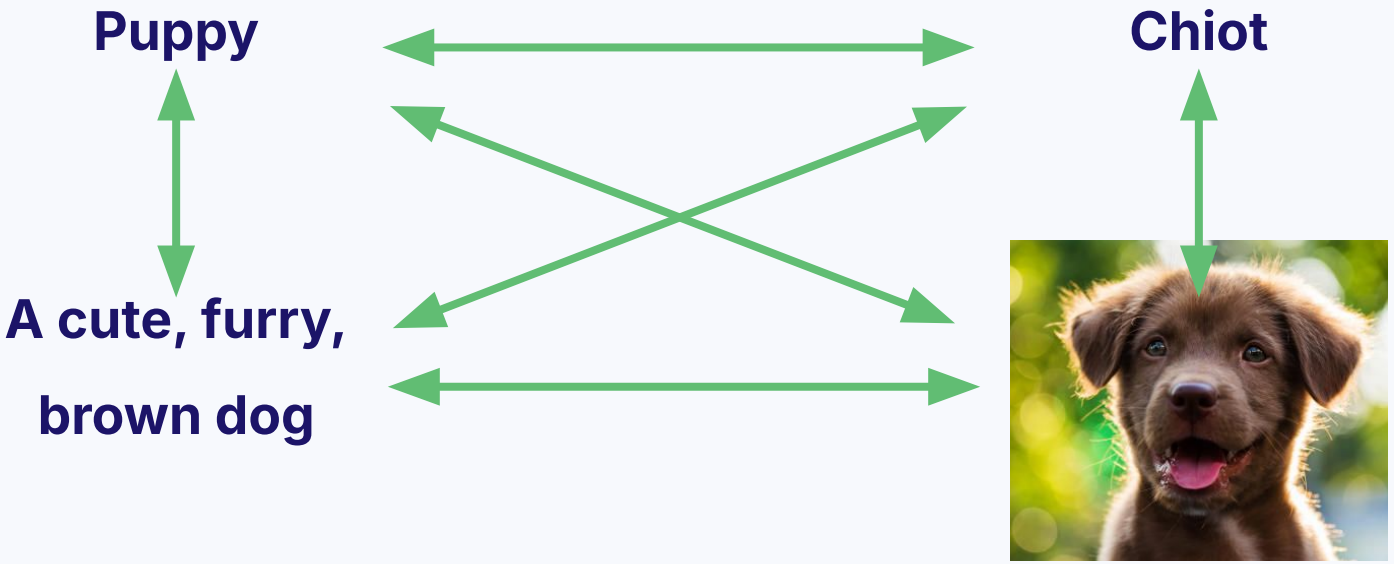
Vectoriser: input ➡ vectors



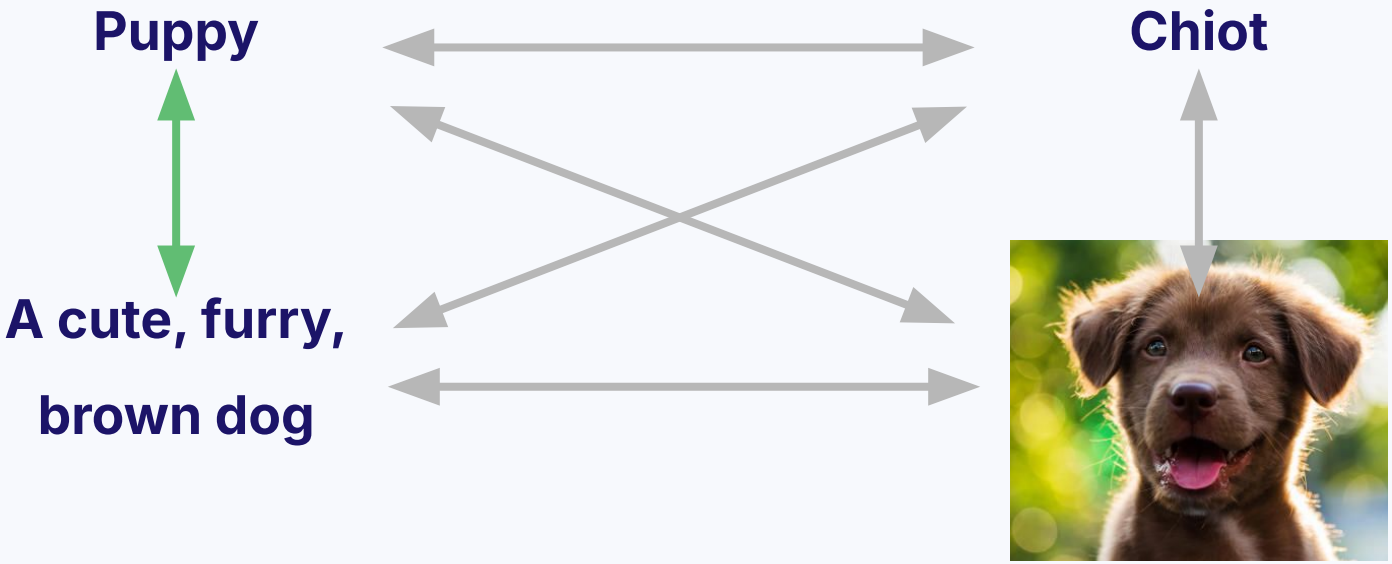
Hundreds of models are available:

- Proprietary models
  - Via Cohere, OpenAI, Google, AWS, etc.
- Open-source models
  - Via Hugging Face
- Why so many?

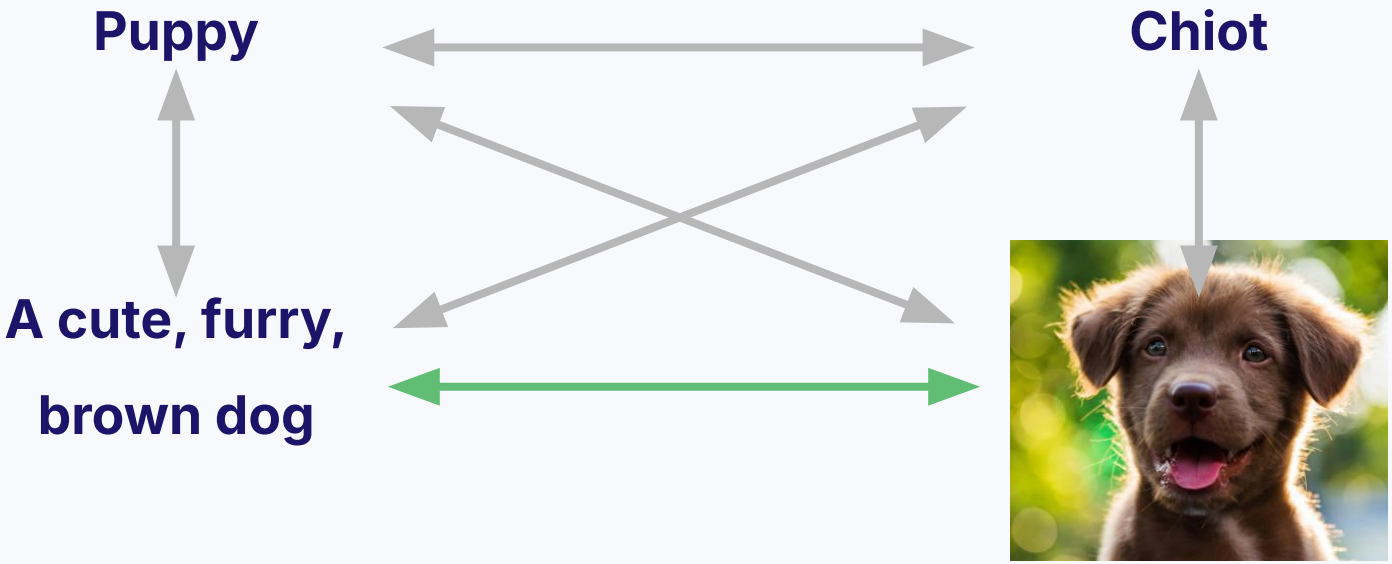
Which two are the *most* similar?



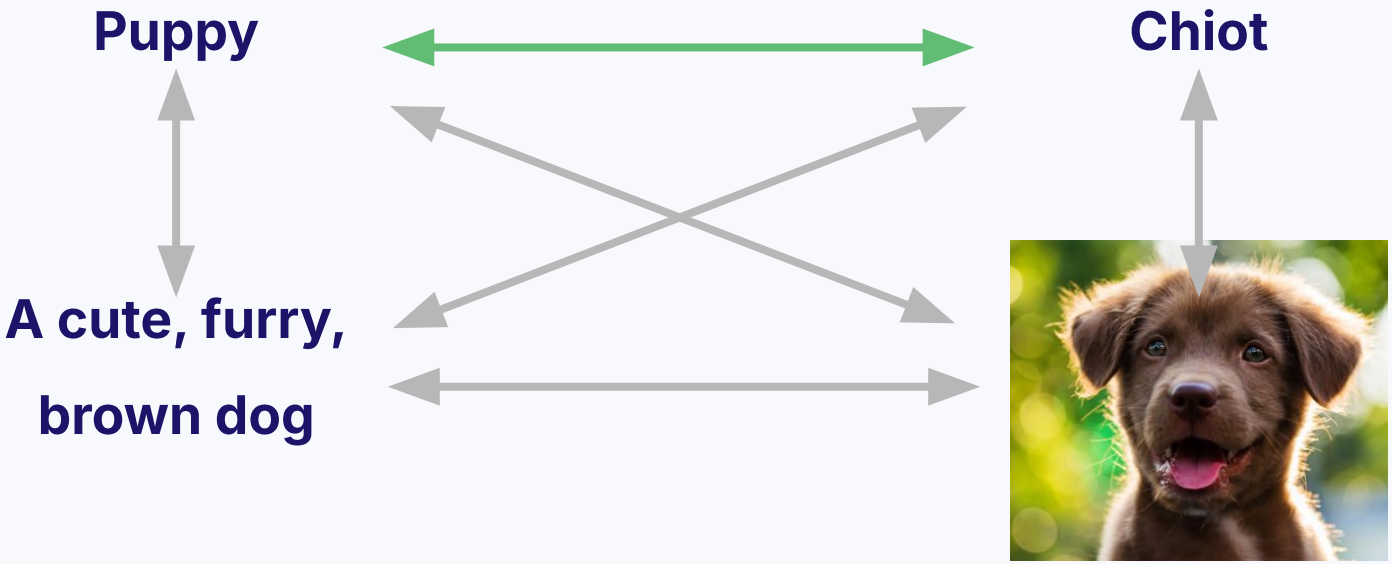
The only English descriptions



Best matching image to text



If you speak English & French



Which two are the *most* similar?

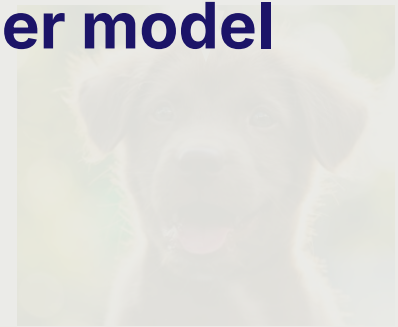
Which two are the *most* similar?

Puppy

Chiot

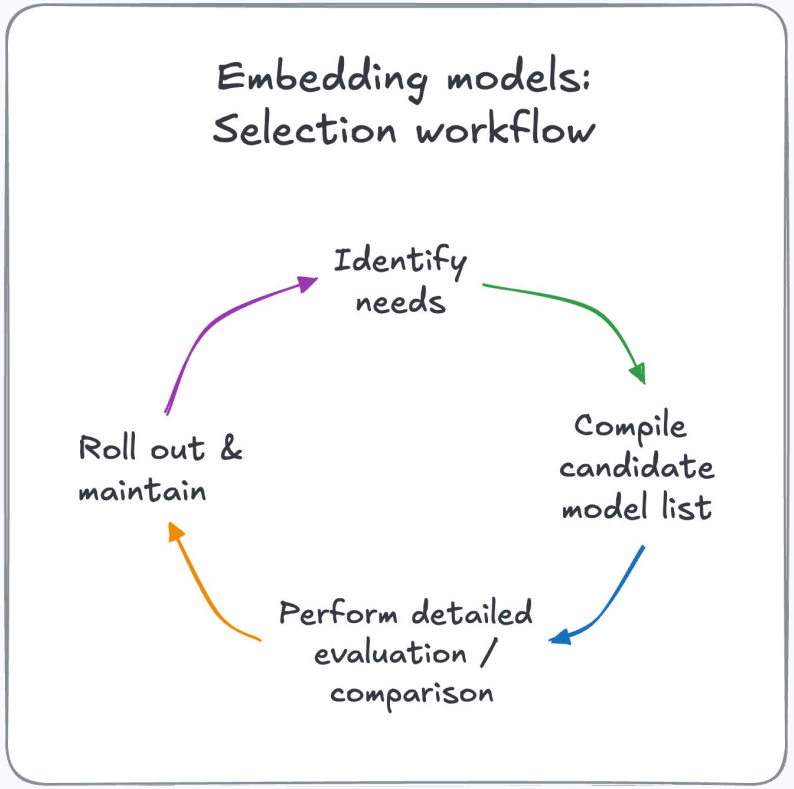
A cute, furry,

brown dog

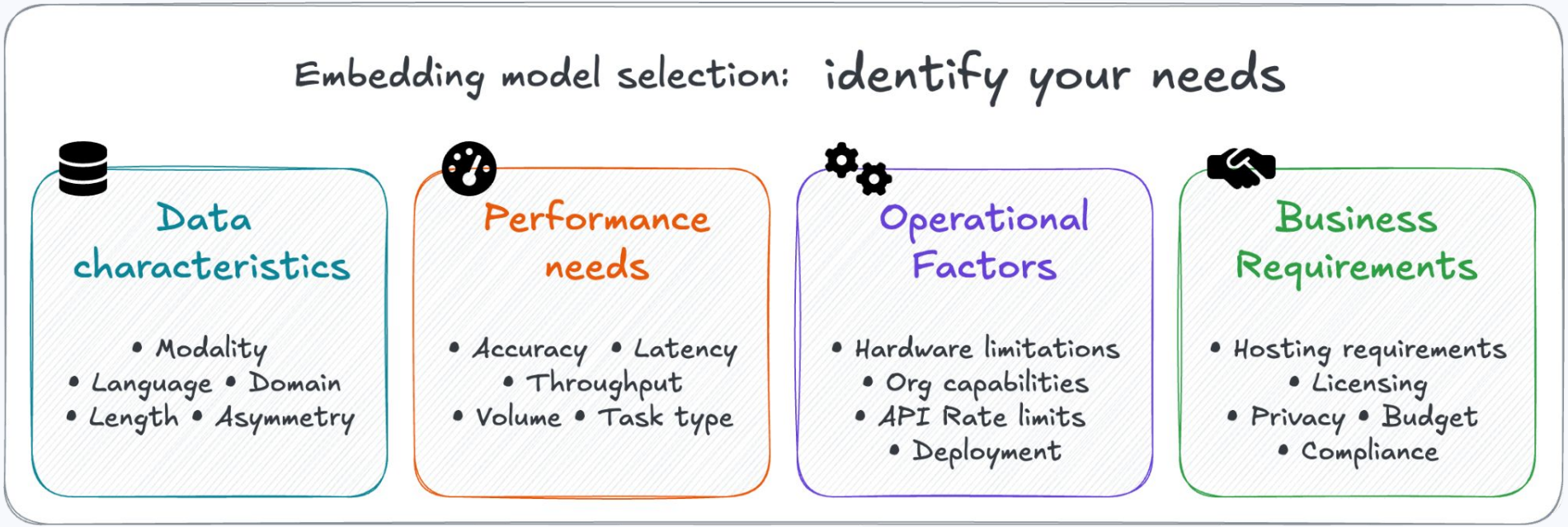


Is determined by the **vectoriser model**

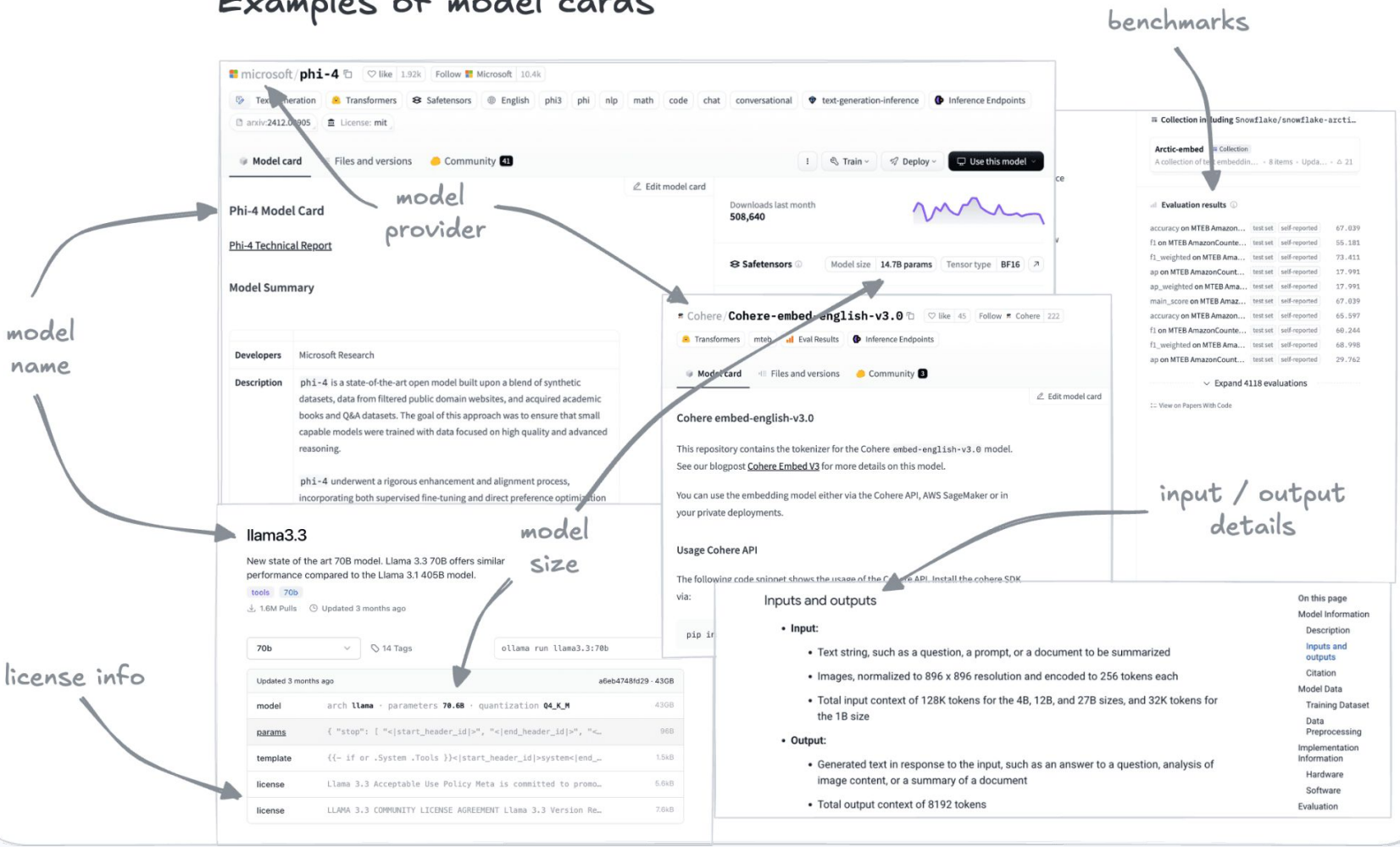
Vectorizer model selection



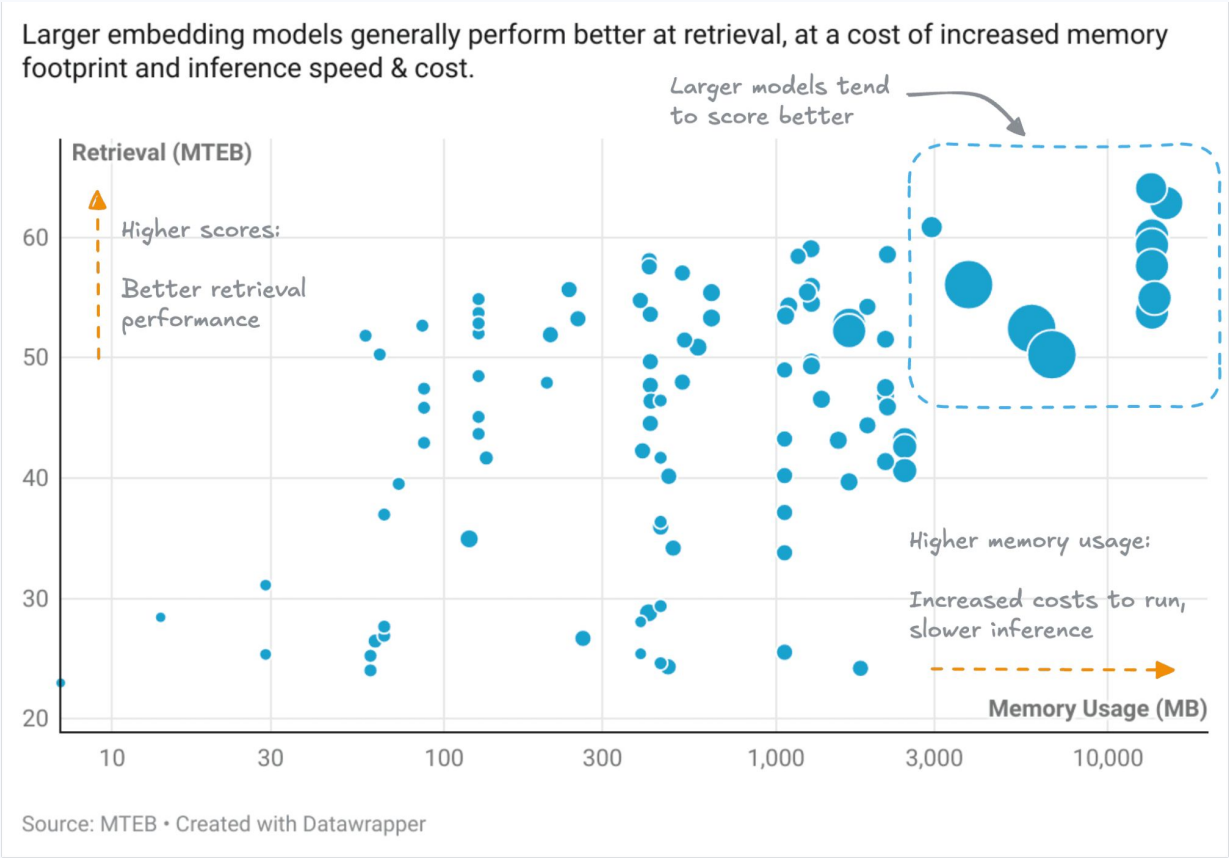
Vectorizer model selection



Examples of model cards



Vectorizer model selection



Evaluate benchmarks

MTEB Benchmarks by task (general)

A selection of embedding models' benchmark results, showing general domain tasks only.

MTEB Benchmarks by task (specialized)

A selection of embedding models' benchmark results, showing more specialized domain tasks only.

	AILAStatutes	ArguAna	LEMBPasskeyRetrieval	LegalBenchCorporateLobbying	SCIDOCS	StackOverflowQA	StatcanDialogueDatasetRetrieval	TRECCOVID
gemini-embedding-exp-03-07	48.8	86.4	38.5	96.0	25.2	96.7	51.1	86.3
NV-Embed-v2	31.6	70.1	42.8	95.2	21.9	92.9	19.5	88.9
text-embedding-3-large	42.0	58.0	69.8	95.2	23.1	92.4	31.1	79.6
e5-mistral-7b-instruct	34.5	61.7	30.8	94.6	16.3	91.0	44.8	87.0
Cohere-embed-multilingual-v3.0	29.7	55.1	38.3	93.8	19.3	89.4	24.2	83.4
snowflake-arctic-embed-l-v2.0	22.8	59.1	77.3	93.8	20.3	86.9	19.1	83.6
snowflake-arctic-embed-m-v2.0	28.1	57.9	58.0	93.5	20.3	90.7	21.6	80.3
text-embedding-3-small	28.2	55.5	38.5	93.9	20.8	88.7	24.1	77.9
Cohere-embed-multilingual-light-v3.0	29.1	53.4	38.0	92.6	17.1	81.5	22.6	78.9

Created for educational purposes to demonstrate embedding model selection.

Chart: JP Hwang @ Weaviate • Source: MTEB @ 2025.Apr.16 • Created with Datawrapper

Detailed evaluation

Evaluate at system-level, based on your own

- Custom benchmarks
  - Curate/create own datasets
  - Define specific metrics
- Quantitative analysis
- Qualitative analysis
  - Apply domain expertise
  - Look for patterns
- End-to-end system evaluation
- Periodic re-evaluation

AI-native workflows

About Weaviate's AI-native developer experience, integrating embedding and genAI models directly

Model provider integrations

API-based

Model provider	Embeddings	Generative AI	Others
Anthropic	-	Text	-
Anyscale	-	Text	-
AWS	Text	Text	
Cohere	Text, Multimodal	Text	Reranker
Databricks	Text	Text	-
FriendliAI	-	Text	-
Google	Text, Multimodal	Text	-
Hugging Face	Text	-	-
Jina AI	Text, Multimodal	-	Reranker
Mistral	Text	Text	-
NVIDIA	Text, Multimodal	Text	Reranker
OctoAI (Deprecated)	Text	Text	-
OpenAI	Text	Text	-
Azure OpenAI	Text	Text	-
Voyage AI	Text, Multimodal	-	Reranker
Weaviate	Text	-	-
xAI	-	Text	-

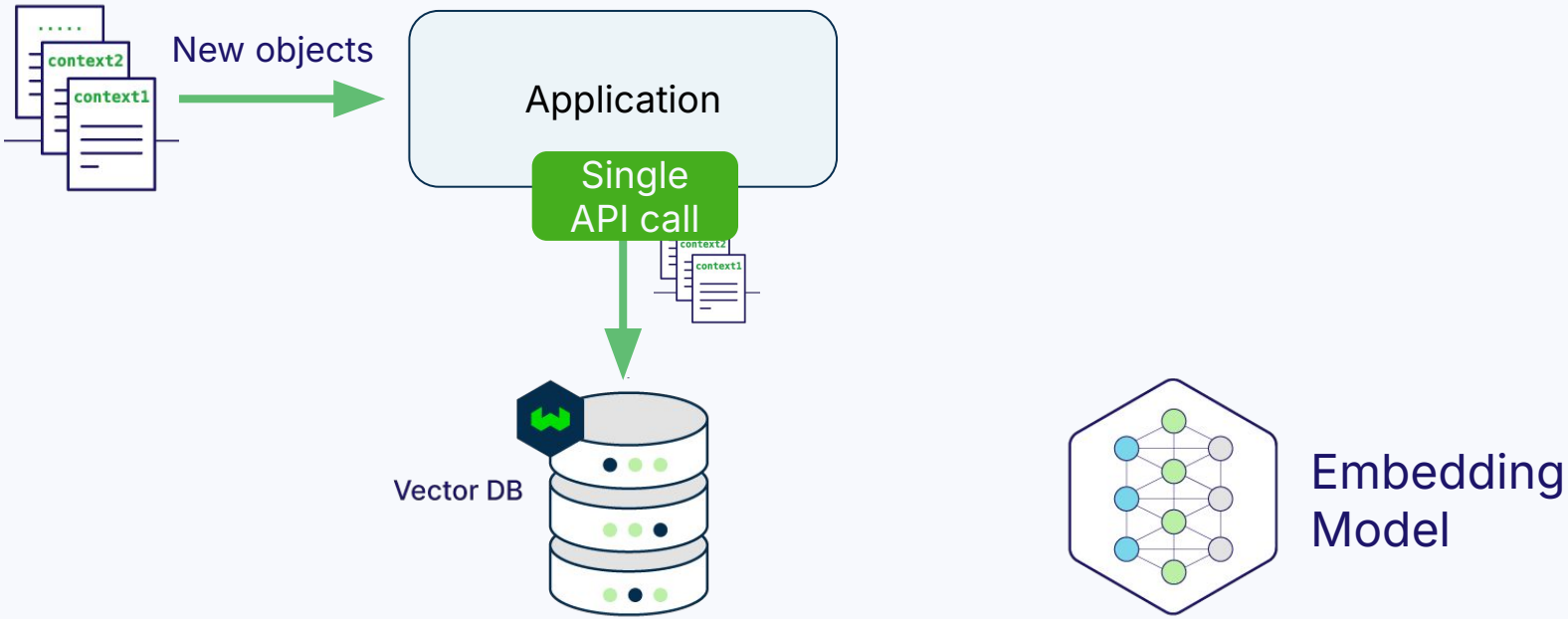
Out-of-the-box integrations

Locally hosted

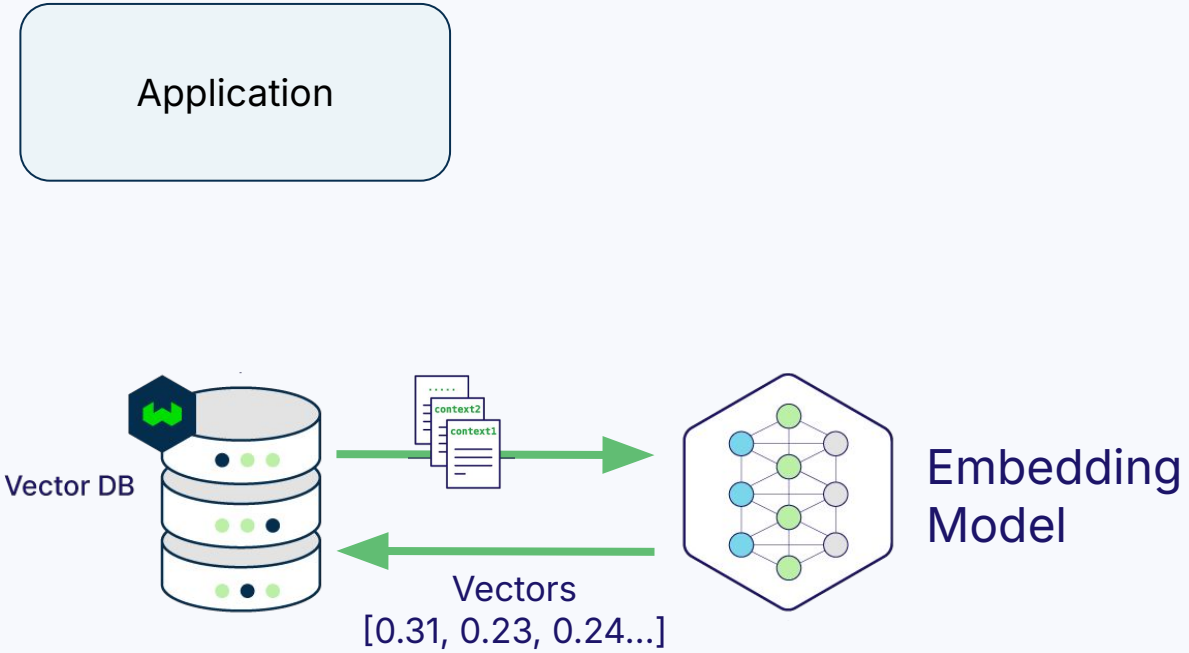
Model provider	Embeddings	Generative AI	Others
GPT4All	Text	-	-
Hugging Face	Text, Multimodal (CLIP)	-	Reranker
Meta ImageBind	Multimodal	-	-
Ollama	Text	Text	-

Embedding model integration

Data operations (i.e. data import)



Data operations (i.e. data import)



Data operations (i.e. data import)

New objects

Application

Single API call

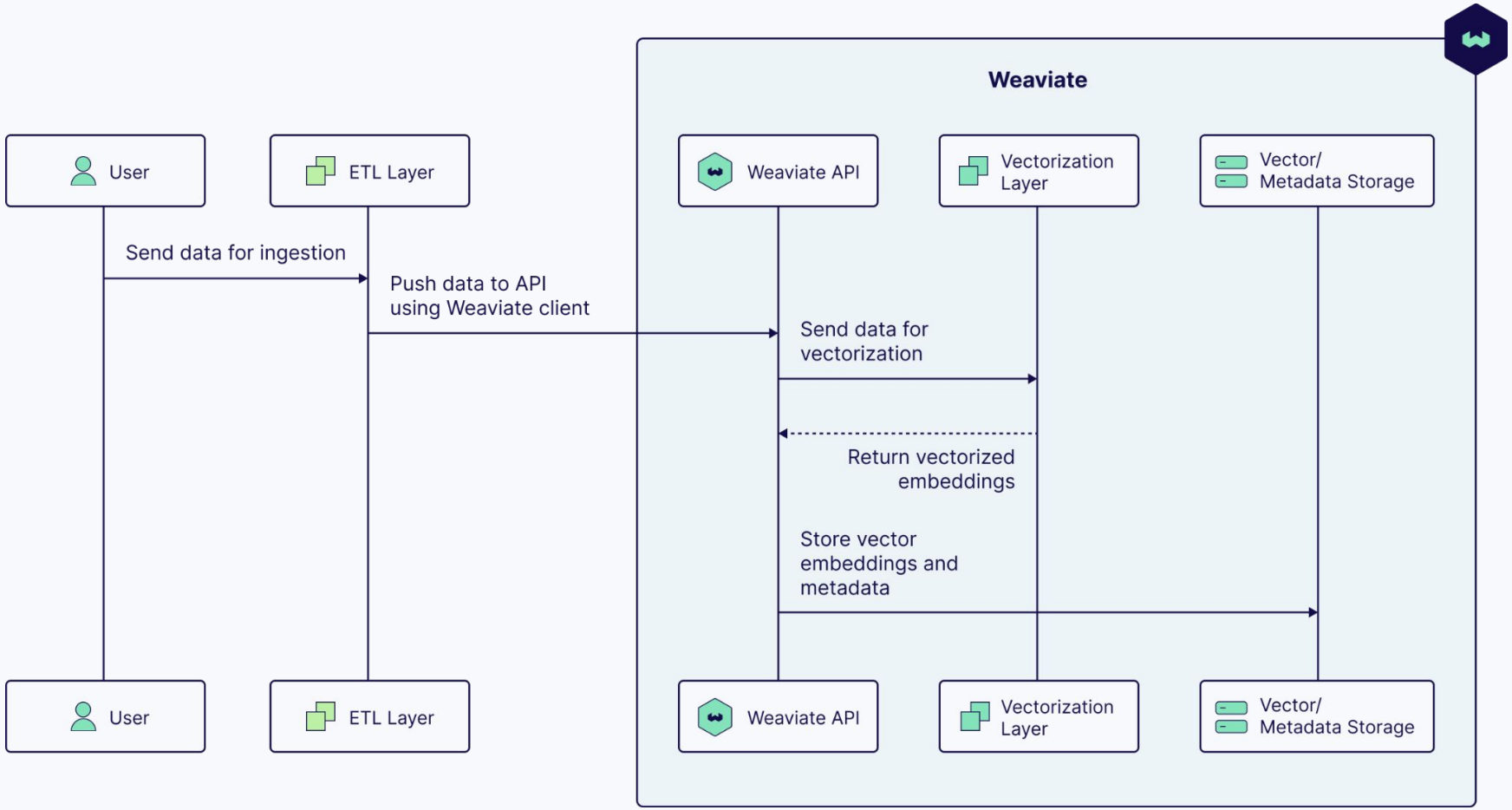

```python
collection = client.collections.get("MyCollection")

with collection.batch.fixed_size(200) as batch:
    for data_row in data_rows:
        batch.add_object(
            properties=data_row,
        )
```

Vector DB

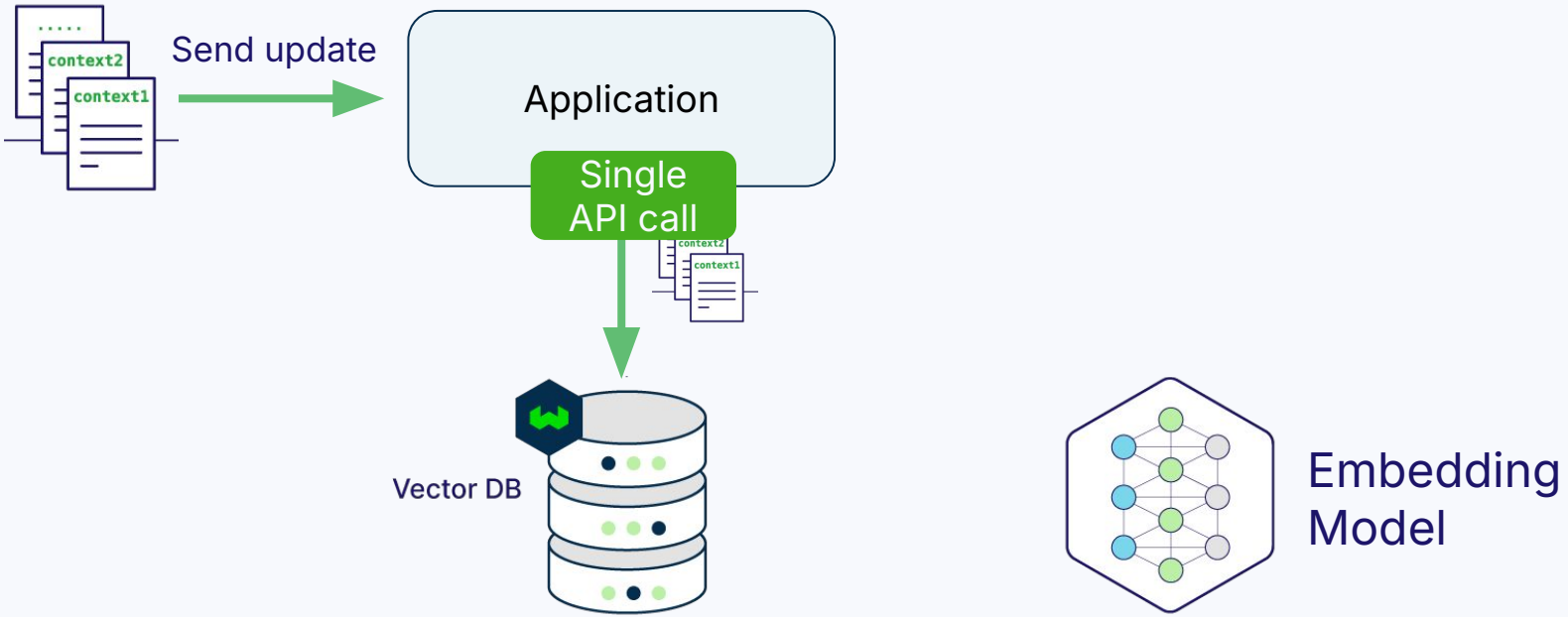
Embedding Model

Vectors
[0.31, 0.23, 0.24...]

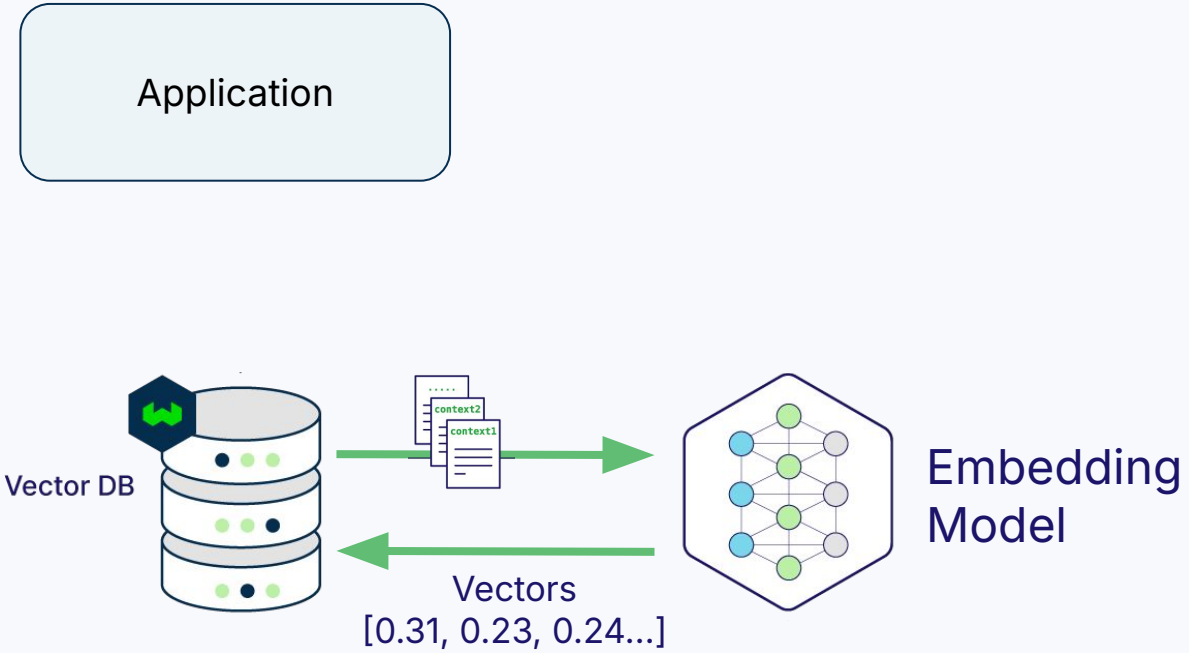
Data Ingestion



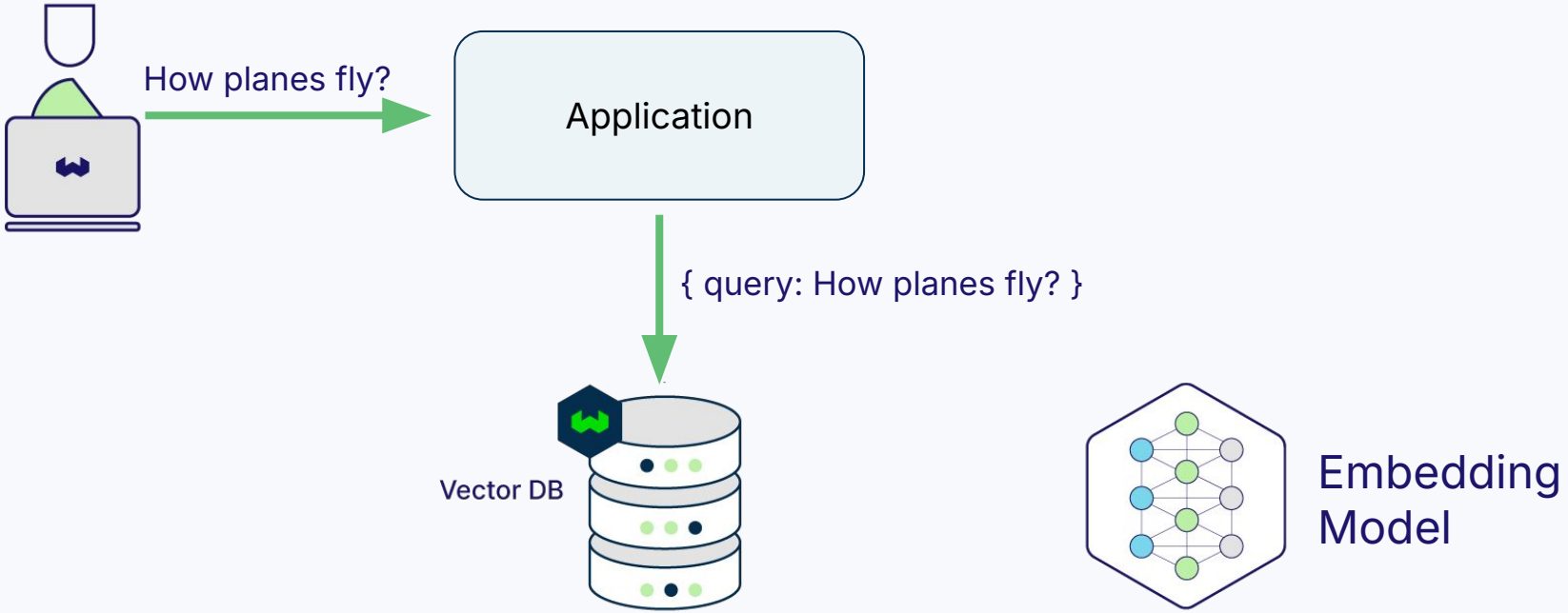
Data operations (i.e. update)



Data operations (i.e. update)

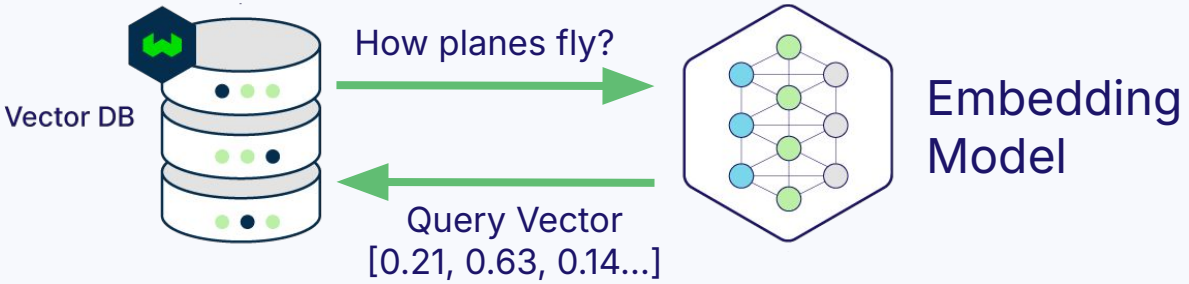


Query

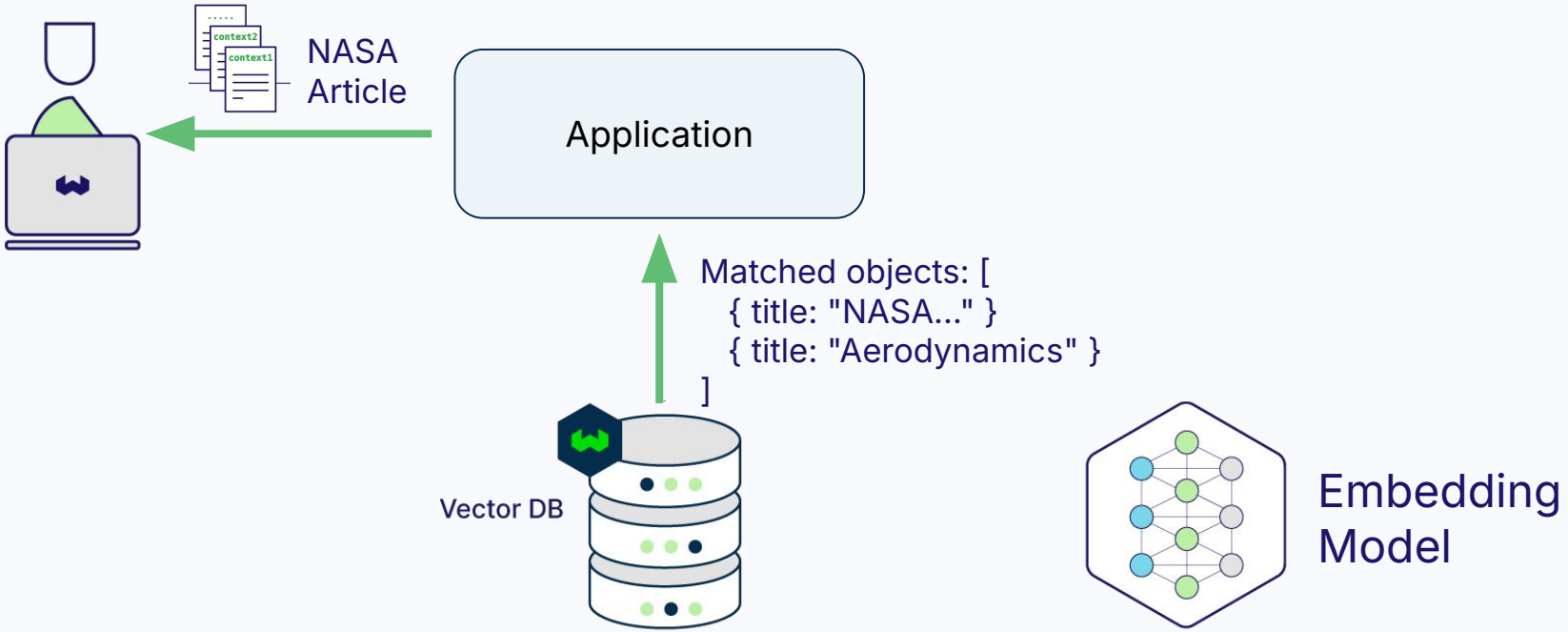

How planes fly?
Application
{ query: How planes fly? }
Vector DB
Embedding Model

Query

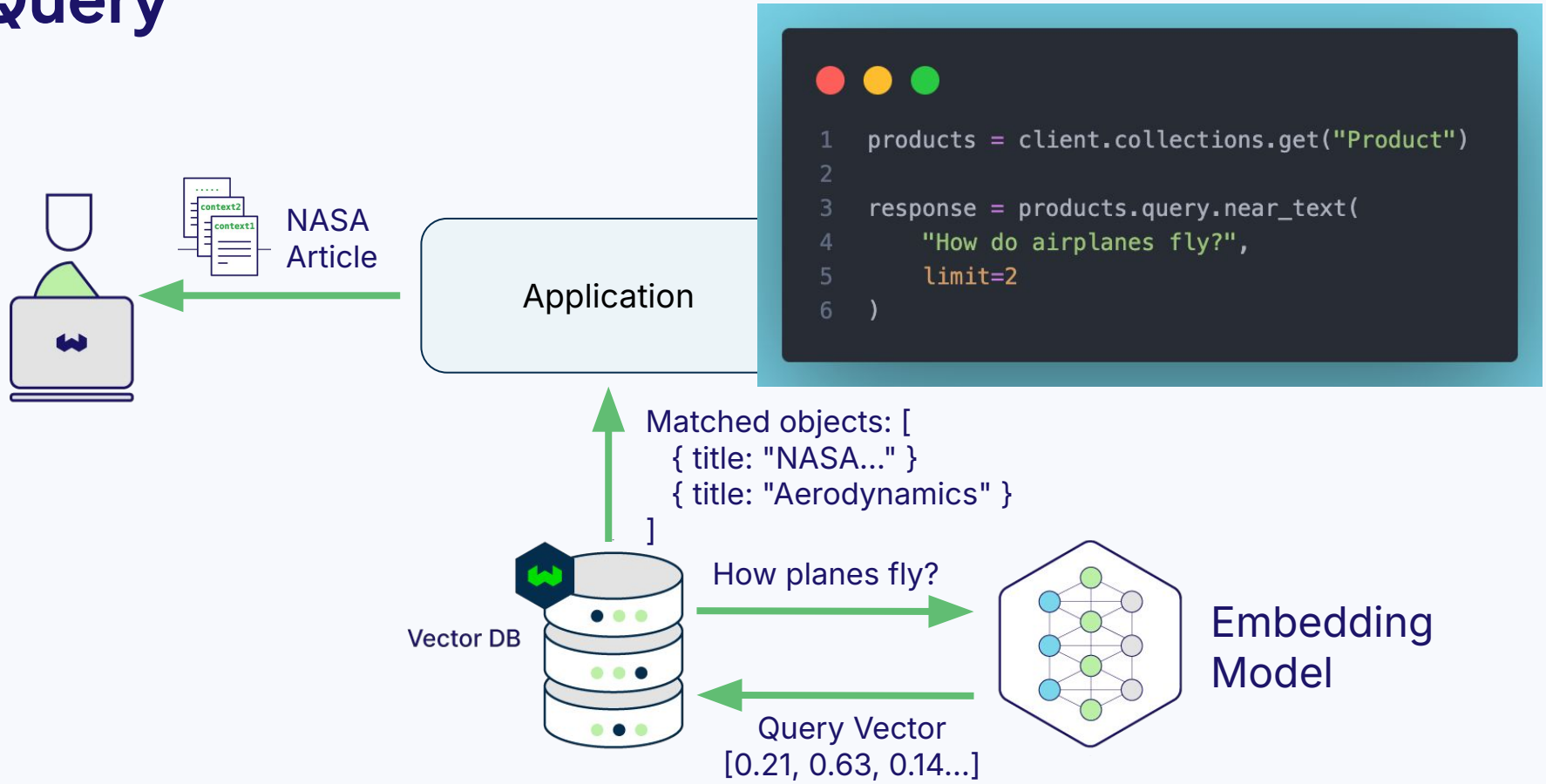
Application



Query


context2
context1
NASA
Article
Application
Matched objects: [
{ title: "NASA..." }
{ title: "Aerodynamics" }
]
Vector DB
Embedding
Model

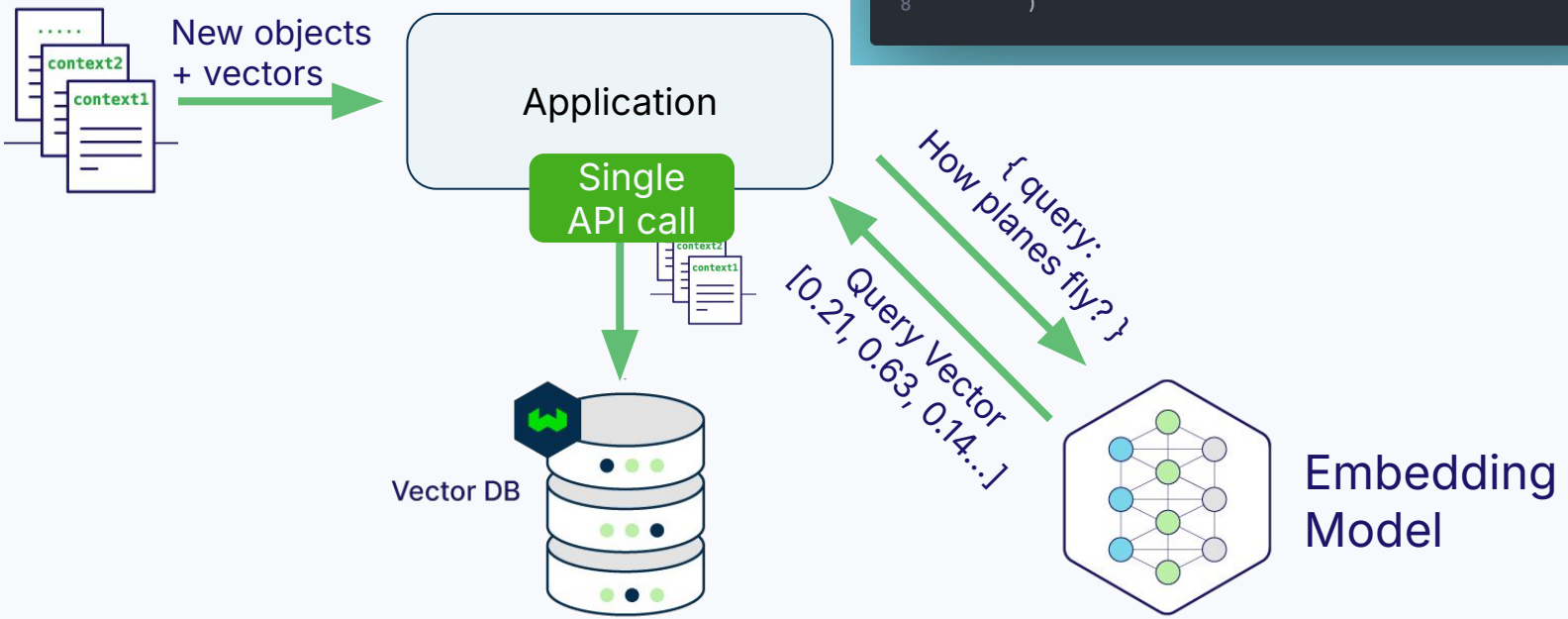
Query


products = client.collections.get("Product")
response = products.query.near_text(
"How do airplanes fly?",
limit=2
)
context2
context1
NASA Article
Application
Matched objects: [
{ title: "NASA..." }
{ title: "Aerodynamics" }
]
Vector DB
How planes fly?
Query Vector
[0.21, 0.63, 0.14...]
Embedding Model

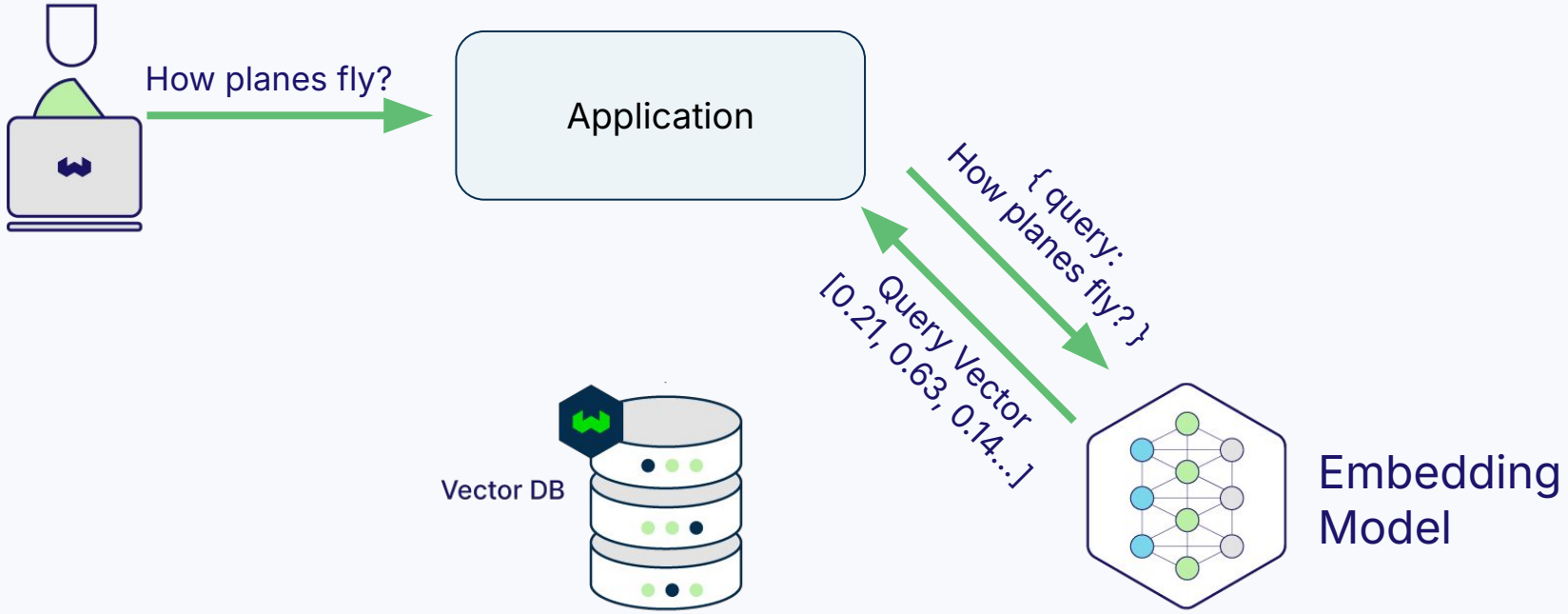
Bring your own vectors

Import Data with Vectors (no vectorizer)

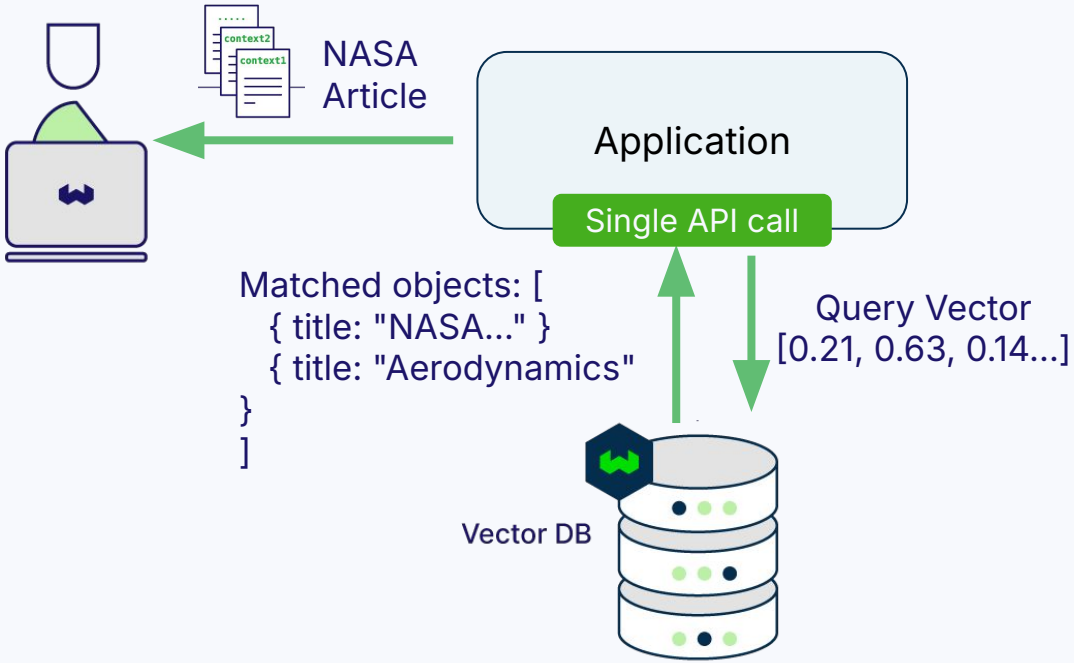
```python
collection = client.collections.get("MyCollection")

with collection.batch.fixed_size(200) as batch:
    for i, data_row in enumerate(data_rows):
        batch.add_object(
            properties=data_row,
            vector=vector_rows[i],  # Manually specify the vector
        )
```

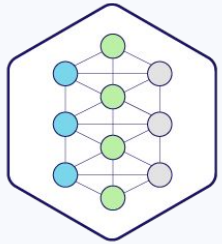


Query
(no vectorizer)



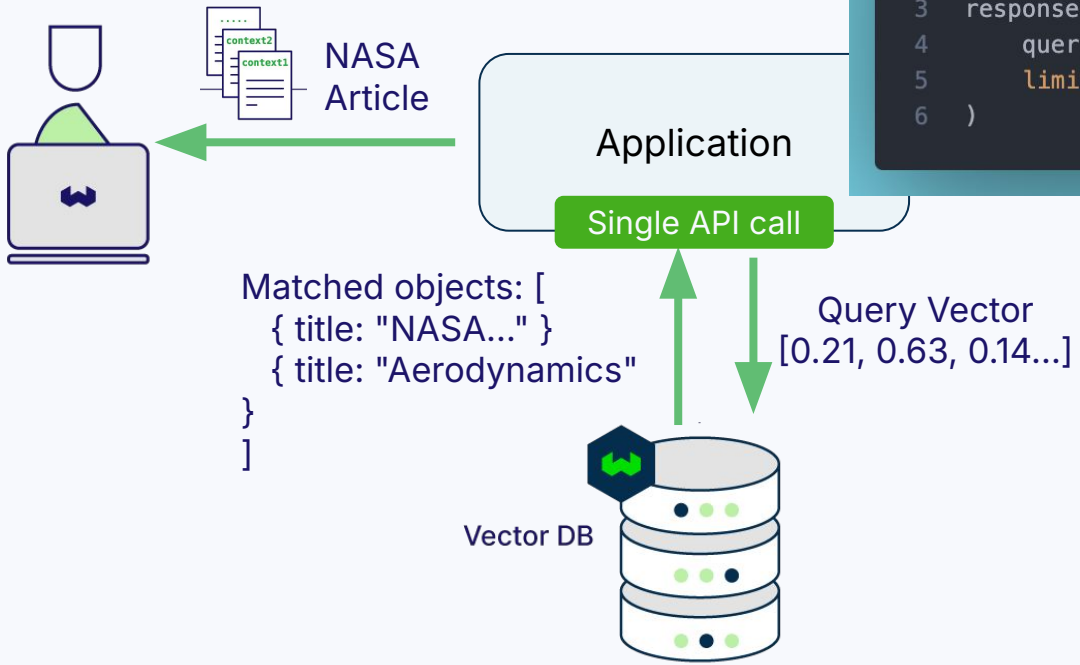
Query
(no vectorizer)





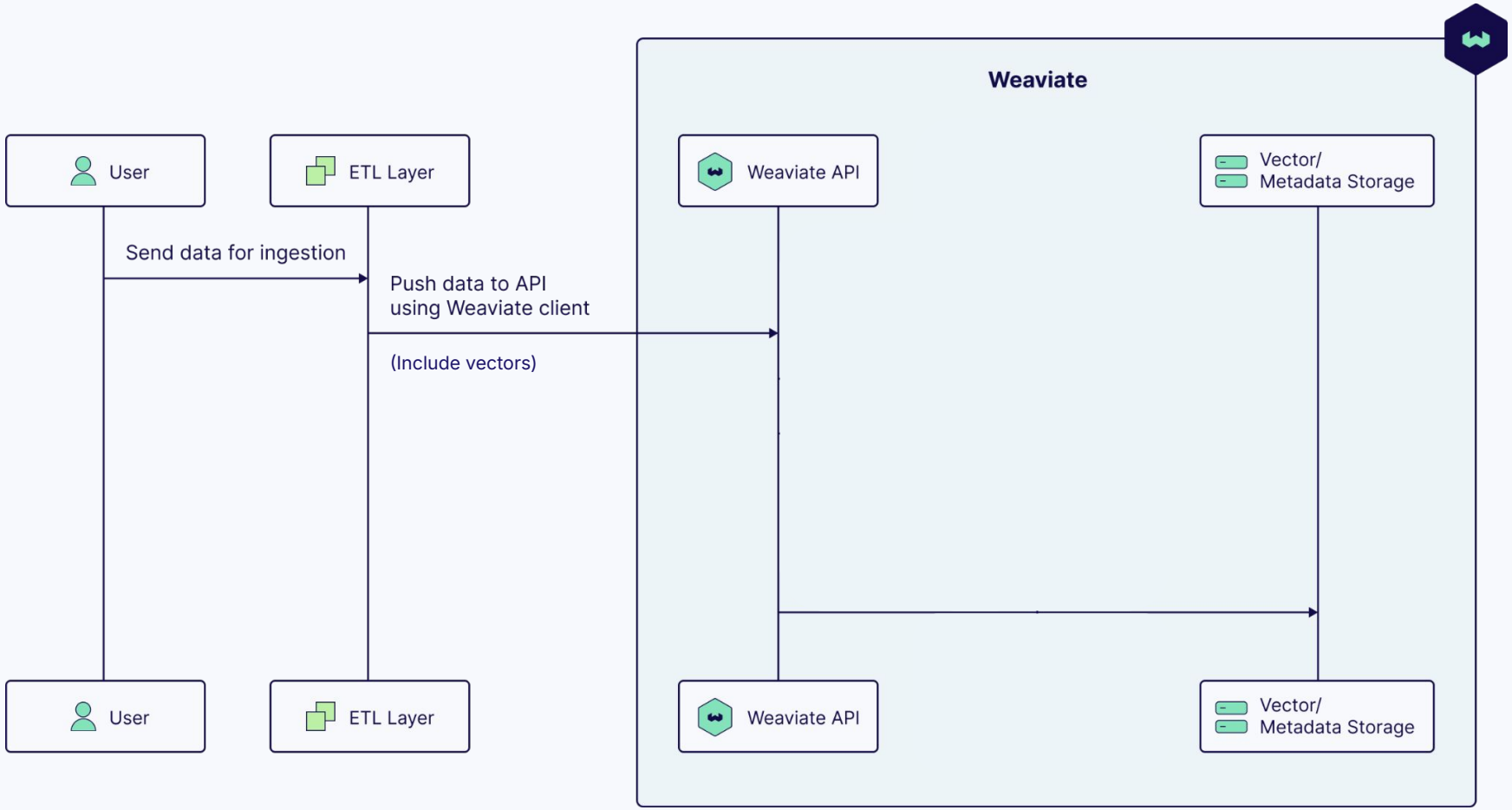
Embedding
Model

Query
(no vectorizer)




```python
products = client.collections.get("Product")

response = products.query.near_vector(
    query_vector,  # Manually obtained query vector
    limit=2
)
```

Data Ingestion



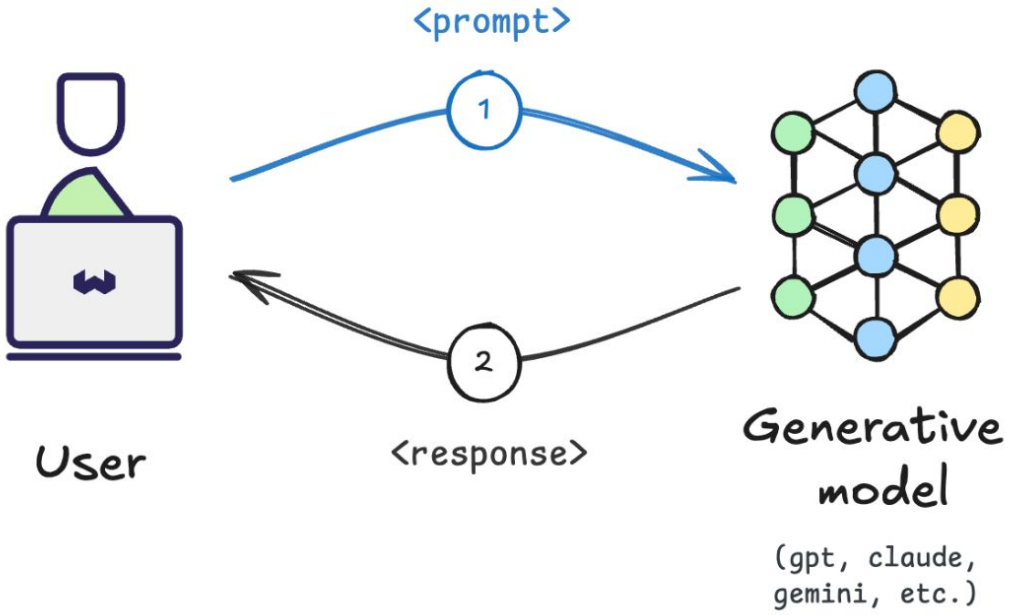
GenAI model integration

Generative AI - basic usage

A generative AI model (e.g. an LLM) can generate a response based on a prompt.
The prompt can be a single "question", or simulate a conversation.

Example prompt:
'What is the capital of France?'

Example pesponse:
'The capital of France is Paris.'



Vector Database

Large Language Model

One Key Difference

Vector Database (Stateful)

Large Language Model (Stateless)

Vector Database (Stateful)

- Insert data into the database, and generate vectors.

Large Language Model (Stateless)

- The knowledge base is trained into the model.

Vector Database (Stateful)

- Insert data into the database, and generate vectors.
- Data changes on the fly.

Large Language Model (Stateless)

- The knowledge base is trained into the model.
- The model doesn’t learn on the fly.

Vector Database (Stateful)

- Insert data into the database, and generate vectors.
- Data changes on the fly.
- CRUD operations to update the state.

Large Language Model (Stateless)

- The knowledge base is trained into the model.
- The model doesn’t learn on the fly.
- Requires fine-tuning to update its worldview.

This is not a story of VDBs vs LLMs

This is a story of VDBs + LLMs

This is a story of VDBs + LLMs

Focus on content:

- Keep the data up to date
- Provide accurate context to LLMs

Focus on response:

- Speak with your brand voice
- Boost understanding of the underlying context

Retrieval augmented generation

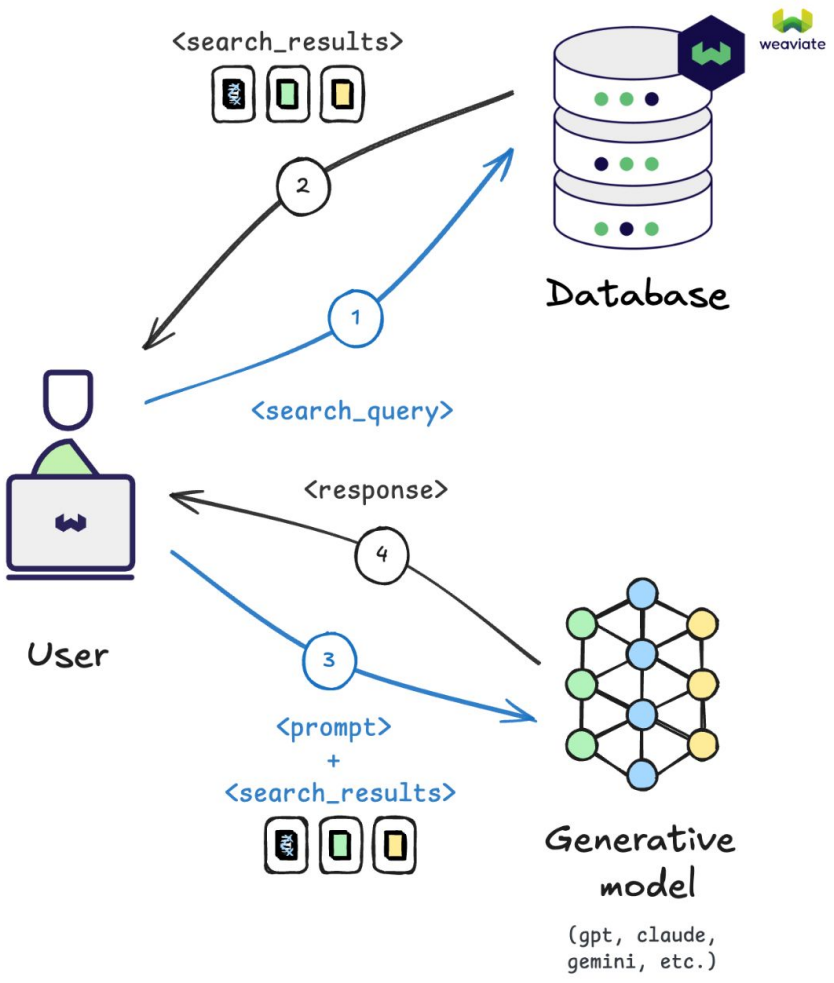
In retrieval augmented generation (RAG), additional information, or "context" is passed to the generative AI model along with a task.

The context is retrieved from a trusted data source, so that the generated text can be based on accurate and up-to-date information.

Example query:
`wiki.query.hybrid(<search_string>)`

Example prompt:
```
Please answer <question>',
using the included information:

===== INFORMATION =====
<search_results>
=====================
'
```

Example response:
`'Based on the provided information, ...'`



Retrieval augmented generation with Weaviate

Weaviate simplifies the RAG workflow by combining the retrieval and generation steps into one.

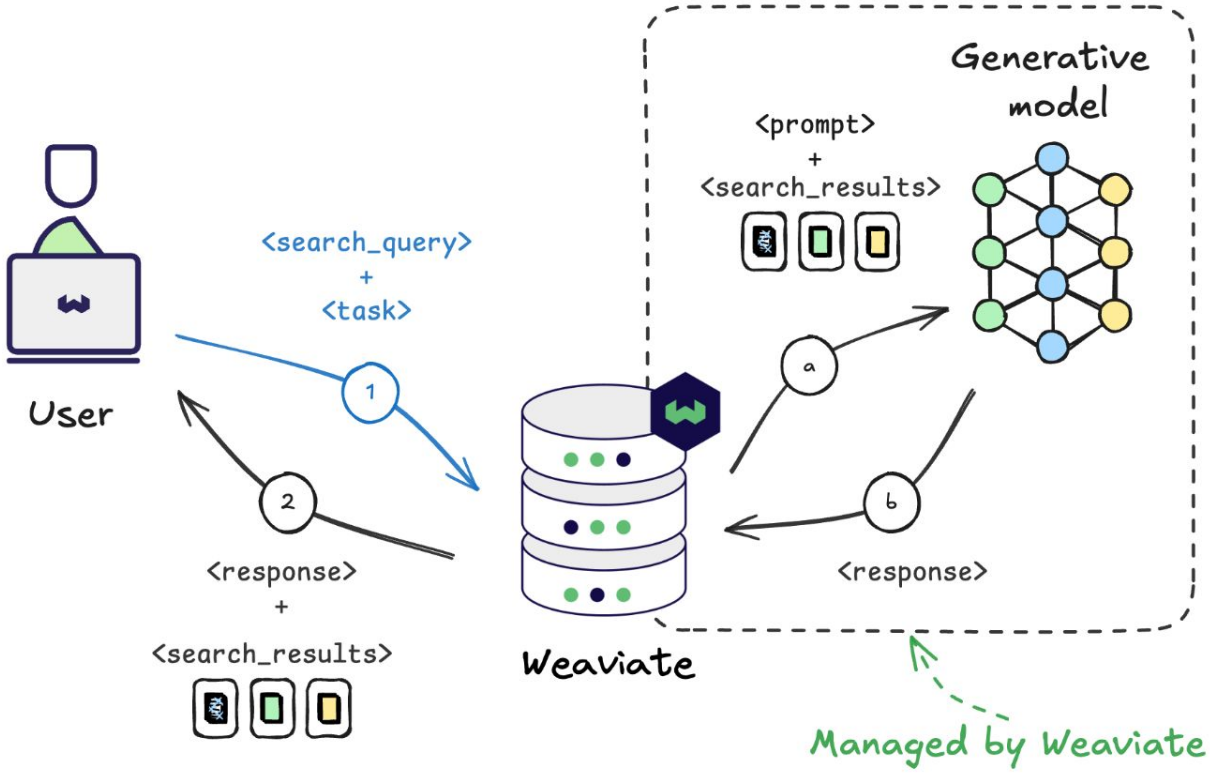
Define and send the search query to Weaviate along with the task, and Weaviate manages the rest.

Example query:

```
wiki.generate.hybrid(
    <search_string>,
    grouped_task=<question>
)
```

Example response:

'Based on the provided information, ...'



RAG – grouped task

```python
reviews = client.collections.get("WineReview")
response = reviews.generate.near_text(
    query="a sweet German white wine",
    limit=2,
    target_vector="description",
    grouped_task="Summarize these reviews",
    generative_provider=GenerativeConfig.openai(),
)

```

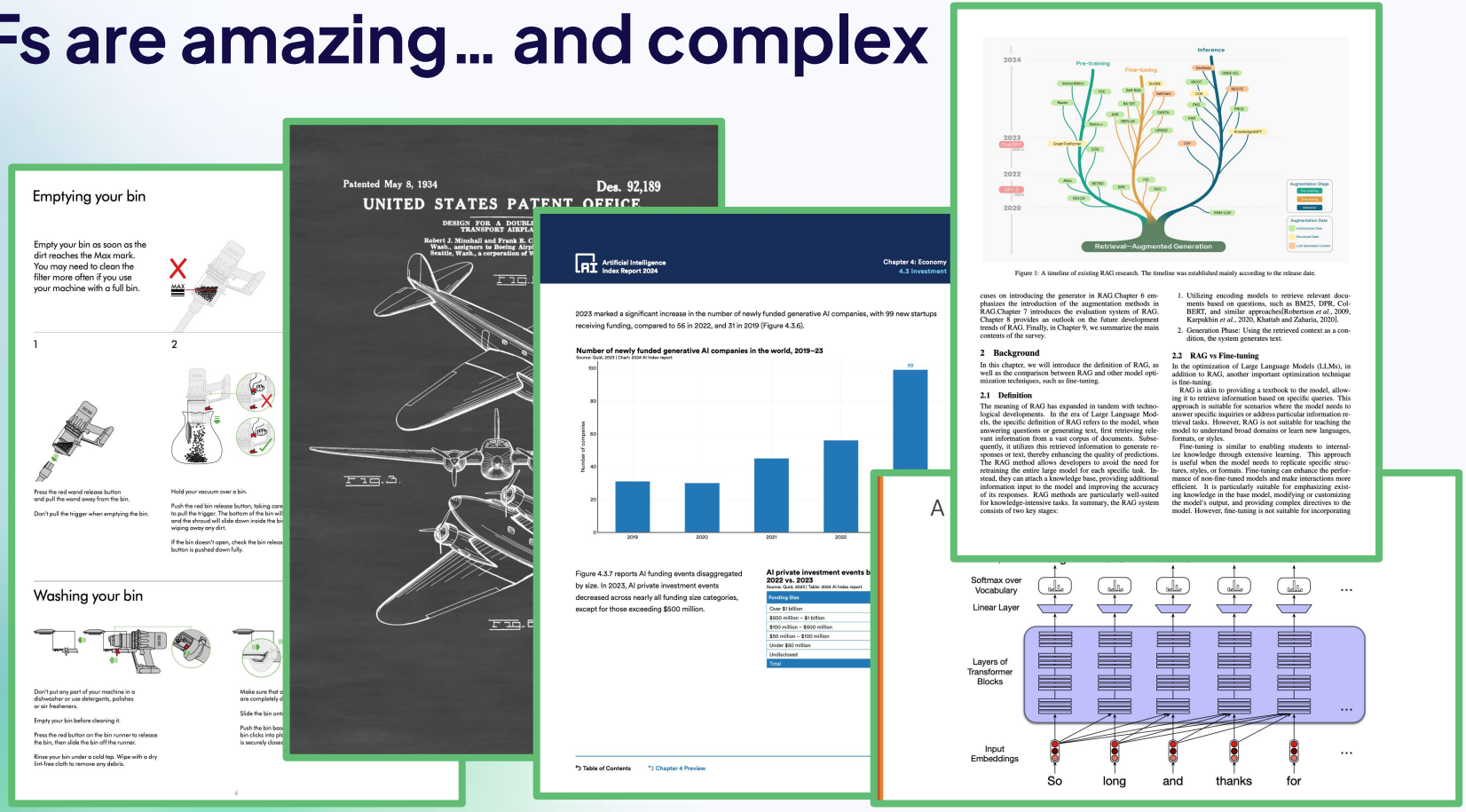
RAG – provider options

```python
response = collection.generate.near_text(
    query="A holiday film",
    limit=2,
    grouped_task="Write a tweet promoting these two movies",
    # highlight-start
    generative_provider=GenerativeConfig.anthropic(
        # # These parameters are optional
        # base_url="https://api.anthropic.com",
        # model="claude-3-opus-20240229",
        # max_tokens=512,
        # temperature=0.7,
        # stop_sequences=["\n\n"],
        # top_p=0.9,
        # top_k=5,
    ),
    # highlight-end
)
```

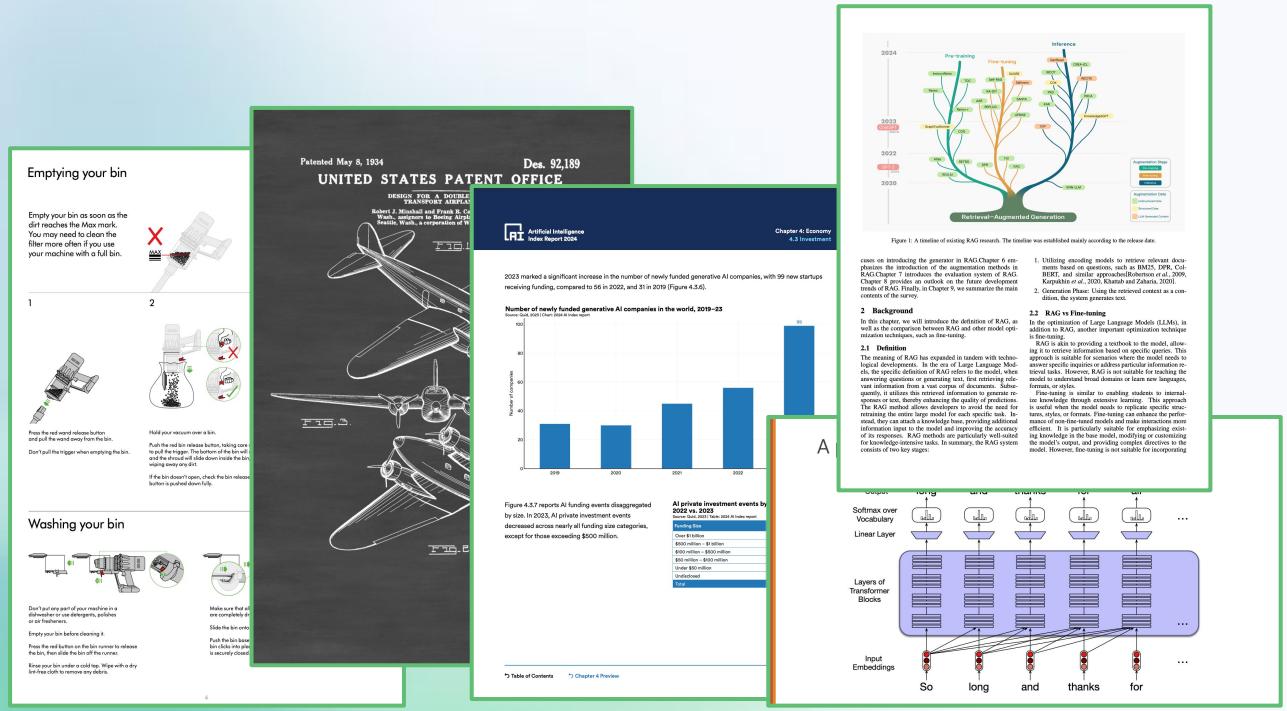
Multi- modality

Why PDFs?

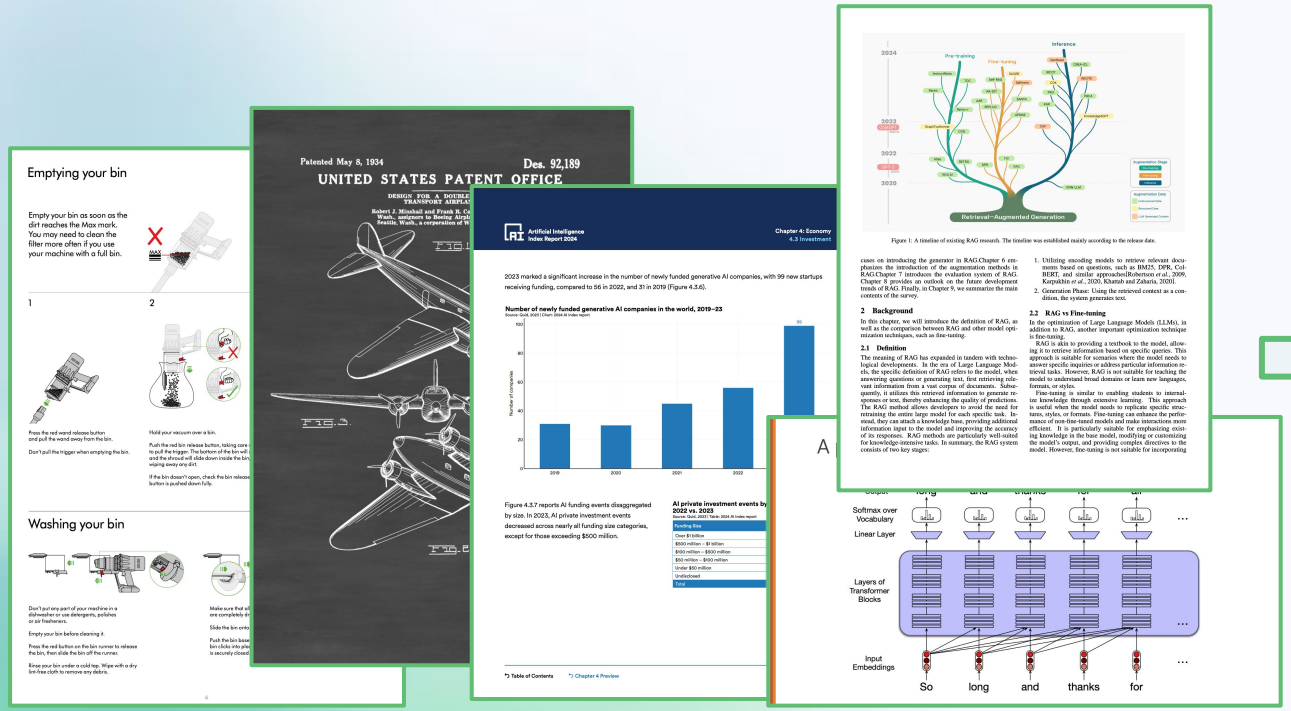
PDFs are amazing… and complex



How to go from here…



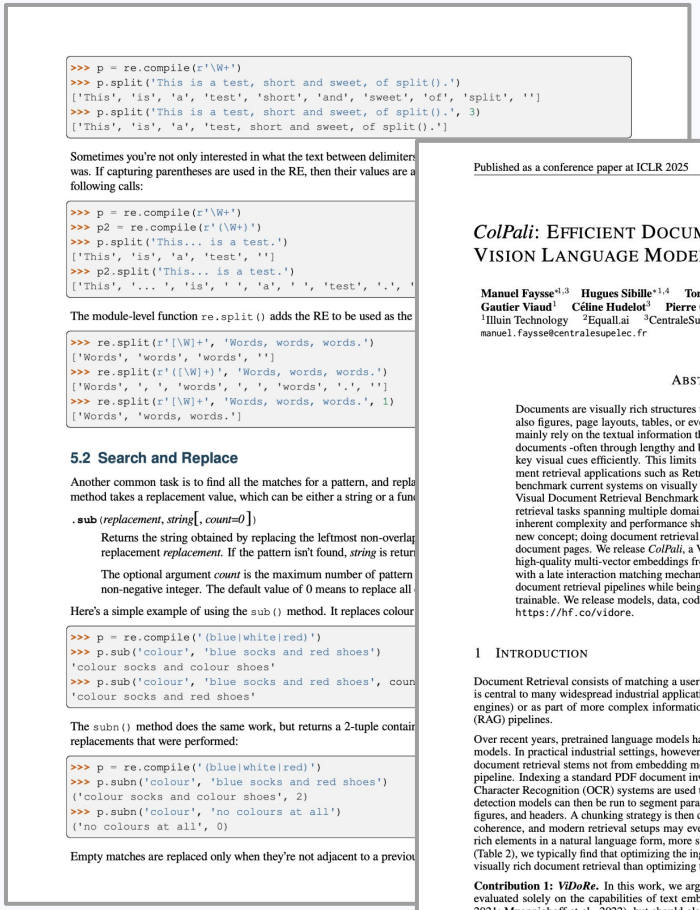
How to go from here… to here?



Solutions & challenges

With existing tooling

- Text-only examples:
  - Simple layout
  - Easy to parse & convert



```
>>> p = re.compile(r'\W+')
>>> p.split('This is a test, short and sweet, of split().')
['This', 'is', 'a', 'test', 'short', 'and', 'sweet', 'of', 'split', '']
>>> p.split('This is a test, short and sweet, of split().', 3)
['This', 'is', 'a', 'test, short and sweet, of split().']
```

Sometimes you're not only interested in what the text between delimiters
was. If capturing parentheses are used in the RE, then their values are a
following calls:

```
>>> p = re.compile(r'\W+')
>>> p2 = re.compile(r'(\W+)')
>>> p.split('This... is a test.')
['This', 'is', 'a', 'test', '']
>>> p2.split('This... is a test.')
['This', '... ', 'is', ' ', 'a', ' ', 'test', '.', '
```

The module-level function `re.split()` adds the RE to be used as the

```
>>> re.split(r'[\W]+', 'Words, words, words.')
['Words', 'words', 'words', '']
>>> re.split(r'([\W]+)', 'Words, words, words.')
['Words', ', ', 'words', ', ', 'words', '.', '']
>>> re.split(r'[\W]+', 'Words, words, words.', 1)
['Words', 'words, words.']
```

5.2 Search and Replace

Another common task is to find all the matches for a pattern, and repla
method takes a replacement value, which can be either a string or a fun

.sub (*replacement*, *string*[, *count=0*])

Returns the string obtained by replacing the leftmost non-overlap
replacement *replacement*. If the pattern isn't found, *string* is retur

The optional argument *count* is the maximum number of pattern
non-negative integer. The default value of 0 means to replace all

Here's a simple example of using the `sub()` method. It replaces colour

```
>>> p = re.compile('(blue|white|red)')
>>> p.sub('colour', 'blue socks and red shoes')
'colour socks and colour shoes'
>>> p.sub('colour', 'blue socks and red shoes', coun
'colour socks and red shoes'
```

The `subn()` method does the same work, but returns a 2-tuple contain
replacements that were performed:

```
>>> p = re.compile('(blue|white|red)')
>>> p.subn('colour', 'blue socks and red shoes')
('colour socks and colour shoes', 2)
>>> p.subn('colour', 'no colours at all')
('no colours at all', 0)
```

Empty matches are replaced only when they're not adjacent to a previou

Published as a conference paper at ICLR 2025

ColPali: Efficient Document Retrieval with Vision Language Models

Manuel Faysse[*1,3] **Hugues Sibille**[*1,4] **Tony Wu**[*1] **Bilel Omrani**[1]
Gautier Viaud[1] **Céline Hudelot**[3] **Pierre Colombo**[2,3]
[1]Illuin Technology [2]Equall.ai [3]CentraleSupélec, Paris-Saclay [4]ETH Zürich
manuel.faysse@centralesupelec.fr

Abstract

Documents are visually rich structures that convey information through text, but also figures, page layouts, tables, or even fonts. Since modern retrieval systems mainly rely on the textual information they extract from document pages to index documents -often through lengthy and brittle processes-, they struggle to exploit key visual cues efficiently. This limits their capabilities in many practical document retrieval applications such as Retrieval Augmented Generation (RAG). To benchmark current systems on visually rich document retrieval, we introduce the Visual Document Retrieval Benchmark *ViDoRe*, composed of various page-level retrieval tasks spanning multiple domains, languages, and practical settings. The inherent complexity and performance shortcomings of modern systems motivate a new concept; doing document retrieval by directly embedding the images of the document pages. We release *ColPali*, a Vision Language Model trained to produce high-quality multi-vector embeddings from images of document pages. Combined with a late interaction matching mechanism, *ColPali* largely outperforms modern document retrieval pipelines while being drastically simpler, faster and end-to-end trainable. We release models, data, code and benchmarks under open licenses at `https://hf.co/vidore`.

1 Introduction

Document Retrieval consists of matching a user query to relevant documents in a given corpus. It is central to many widespread industrial applications, either as a standalone ranking system (search engines) or as part of more complex information extraction or Retrieval Augmented Generation (RAG) pipelines.

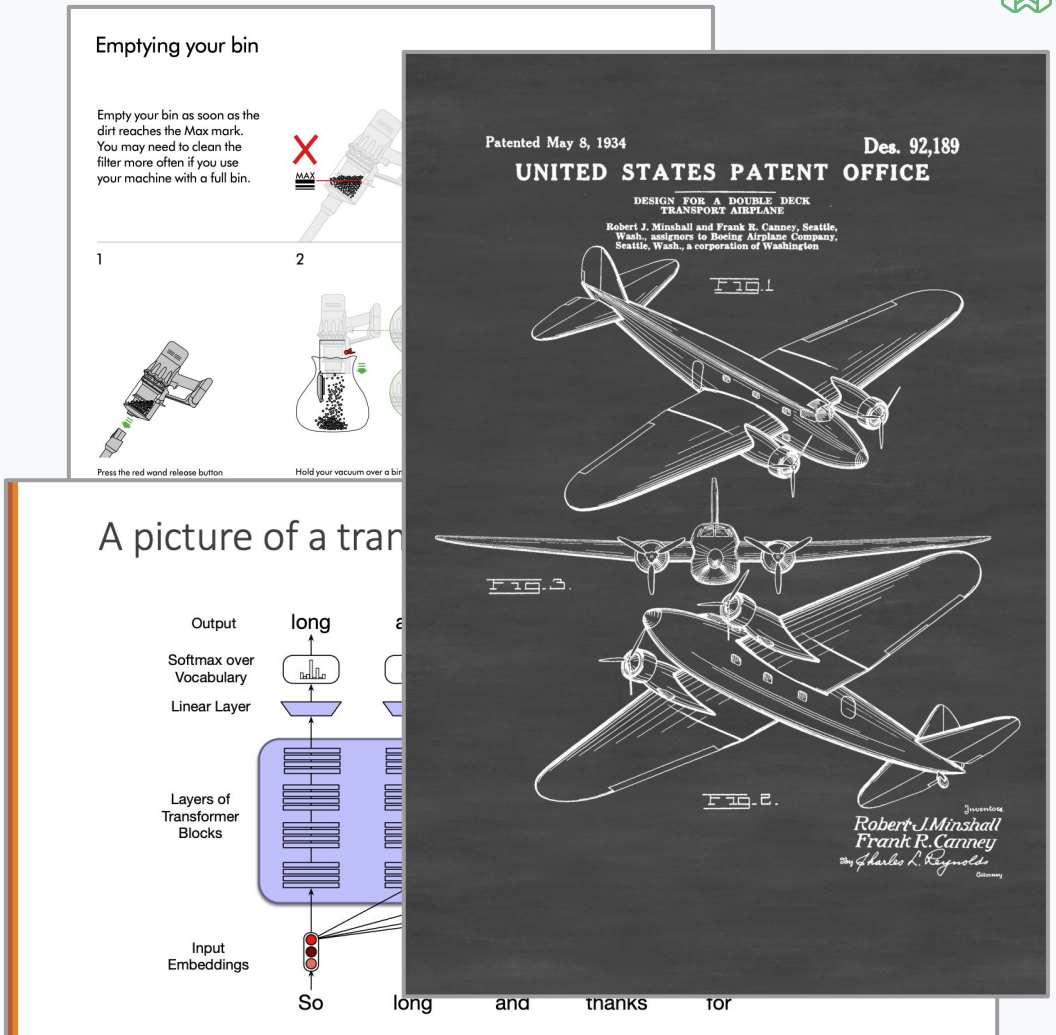
Over recent years, pretrained language models have enabled large improvements in text embedding models. In practical industrial settings, however, the primary performance bottleneck for efficient document retrieval stems not from embedding model performance but from the prior data ingestion pipeline. Indexing a standard PDF document involves several steps. First, PDF parsers or Optical Character Recognition (OCR) systems are used to extract words from the pages. Document layout detection models can then be run to segment paragraphs, titles, and other page objects such as tables, figures, and headers. A chunking strategy is then defined to group text passages with some semantical coherence, and modern retrieval setups may even integrate a captioning step to describe visually rich elements in a natural language form, more suitable for embedding models. In our experiments (Table 2), we typically find that optimizing the ingestion pipeline yields much better performance on visually rich document retrieval than optimizing the text embedding model.

Contribution 1: *ViDoRe*. In this work, we argue that document retrieval systems should not be evaluated solely on the capabilities of text embedding models (Bajaj et al., 2016; Thakur et al., 2021; Muennighoff et al., 2022), but should also consider the context and visual elements of the documents to be retrieved. To this end, we create and openly release *ViDoRe*, a comprehensive benchmark to evaluate systems on page-level document retrieval with a wide coverage of domains, visual elements, and languages. *ViDoRe* addresses practical document retrieval scenarios, where

[*]Equal Contribution

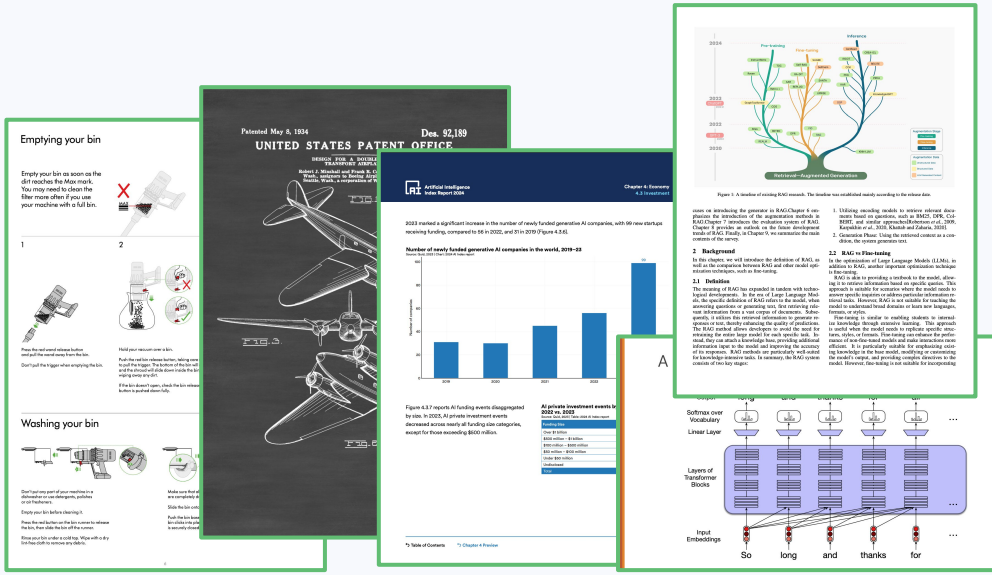
With existing tooling

- Mixed media, different layouts
- More challenging
  - How to decide boundaries?
  - What direction does information flow?
  - Do you know what the user will be looking for?

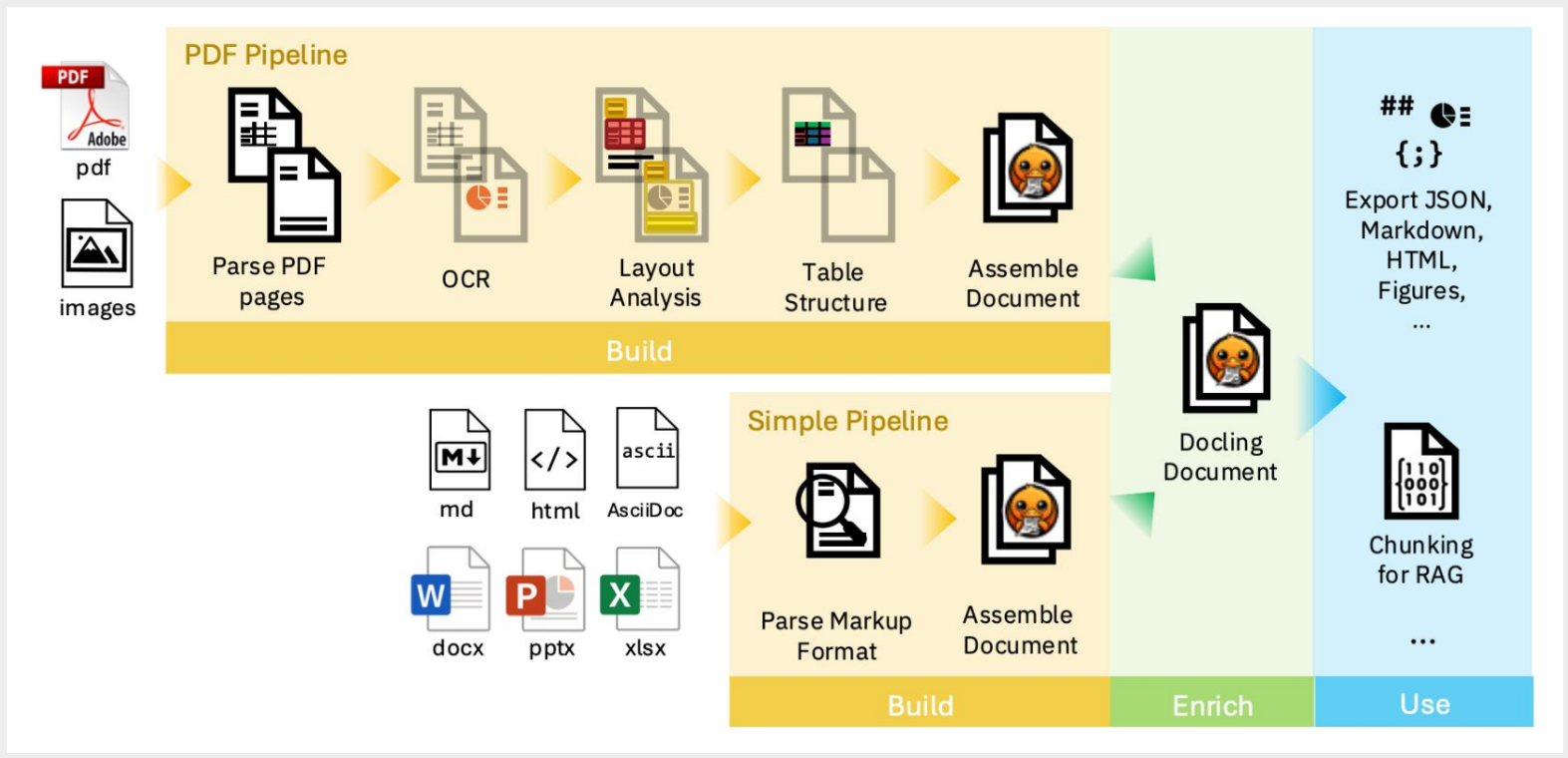


With existing tooling – example workflow

- Document parsing
- Image & text extraction
- Layout analysis
- Chunking
- Rebuild content (tables / images)
- Text searches (keyword / semantic)



Example – Docling

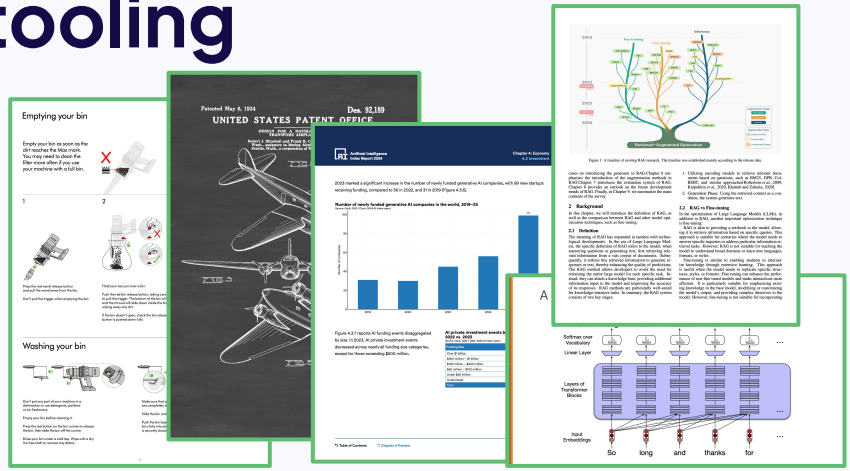


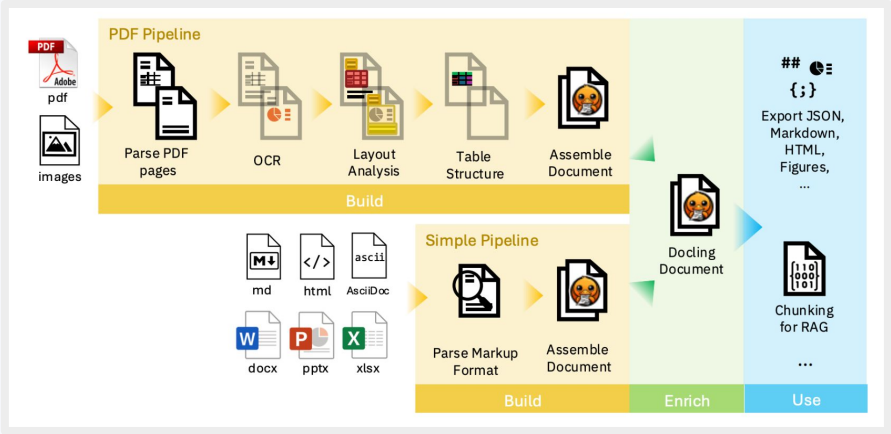
Demo – Docling

A quick demo 🚀

Challenges with existing tooling

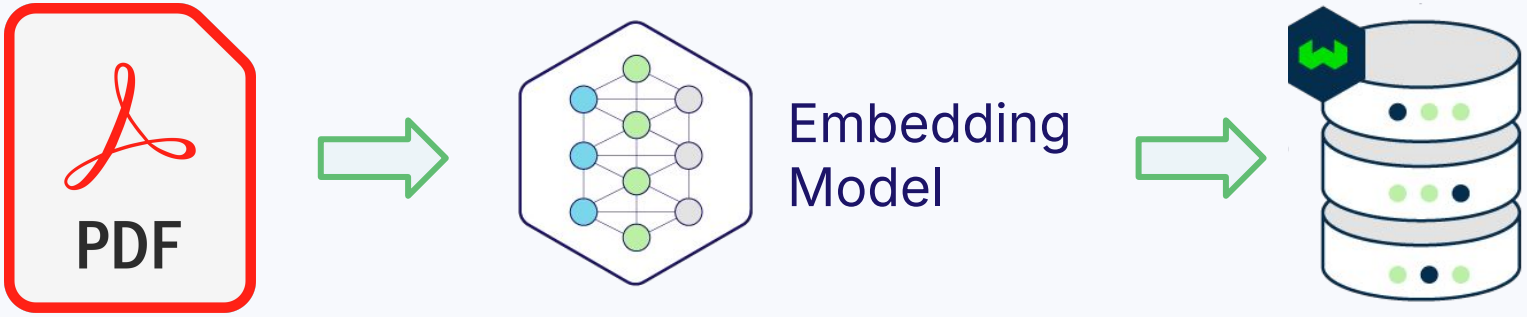
- Complicated pipeline
  - Image/text borders
  - Relationships between elements
  - Context management
  - Chunking
- Judgement calls required
- One-size-fits-all solution impossible



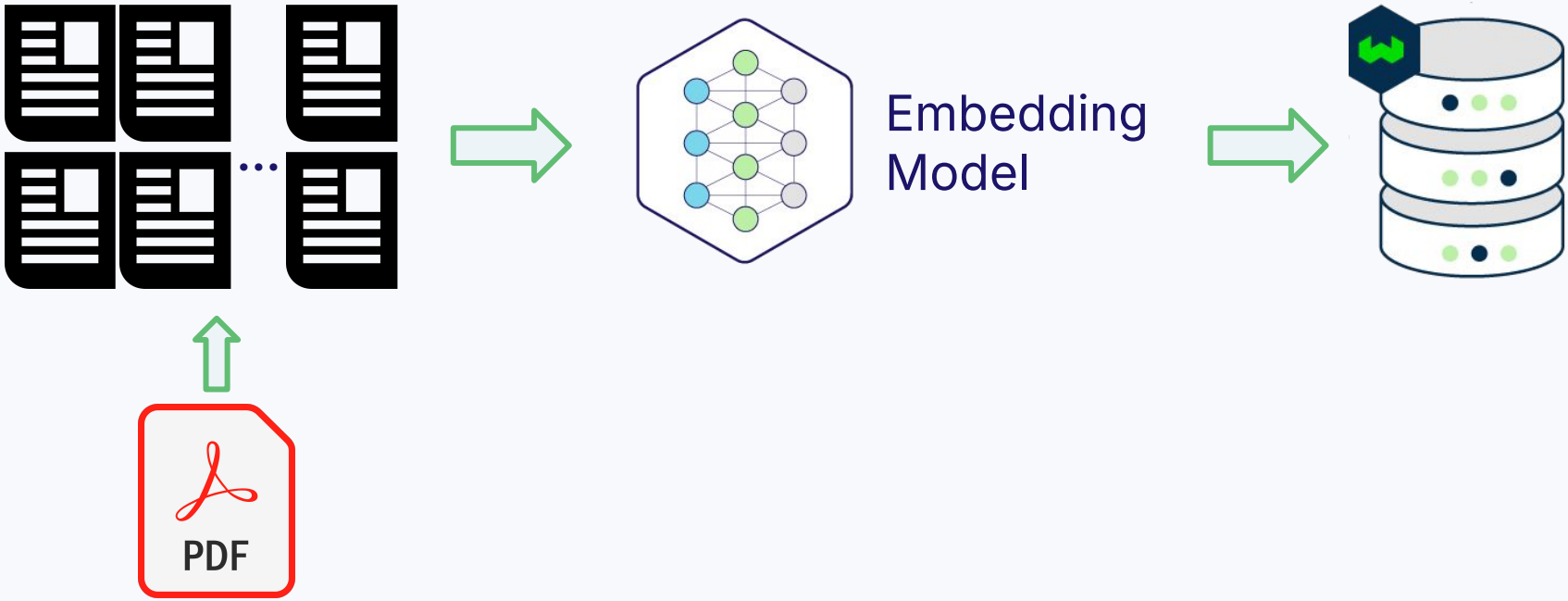


Another approach

Just embed the whole page!

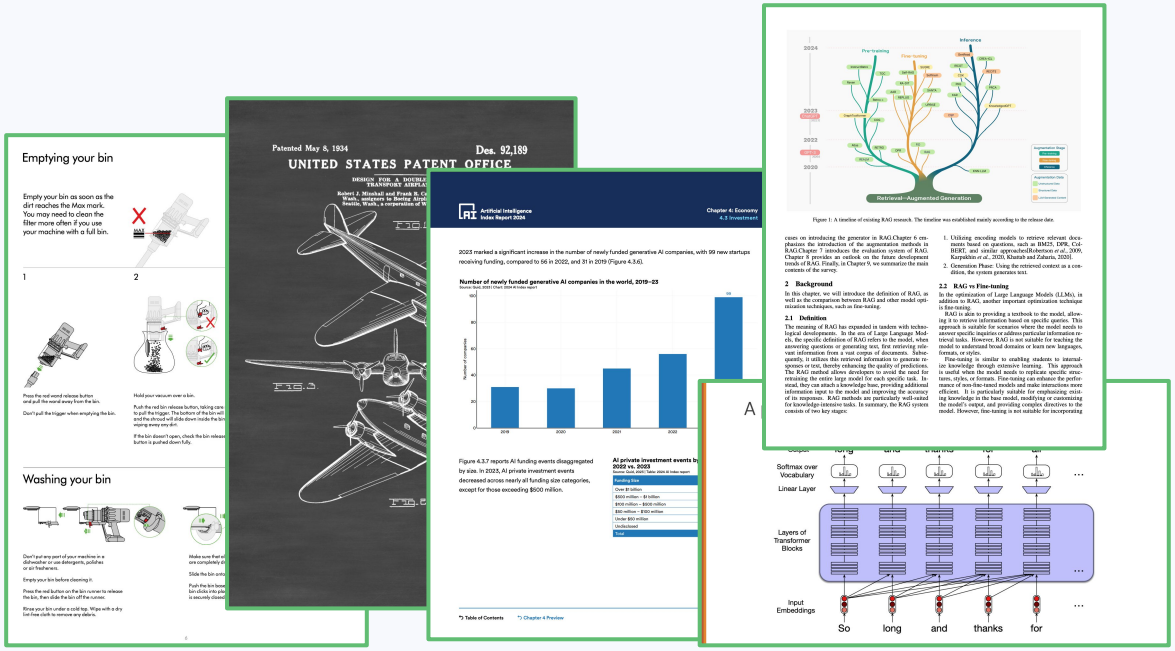


Just embed the whole page!



Why is this appealing?

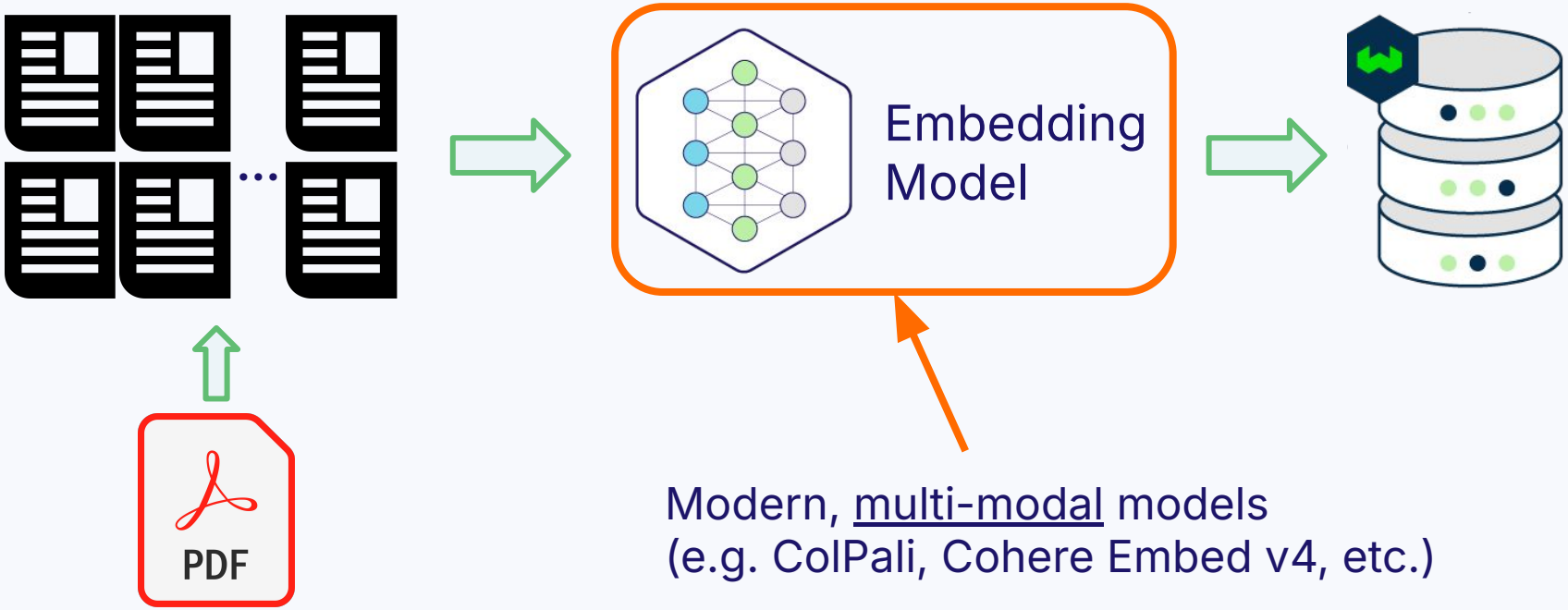
- Respects page layouts
- The data is already chunked (by page)
- Mimics how we read PDFs
- Is robust

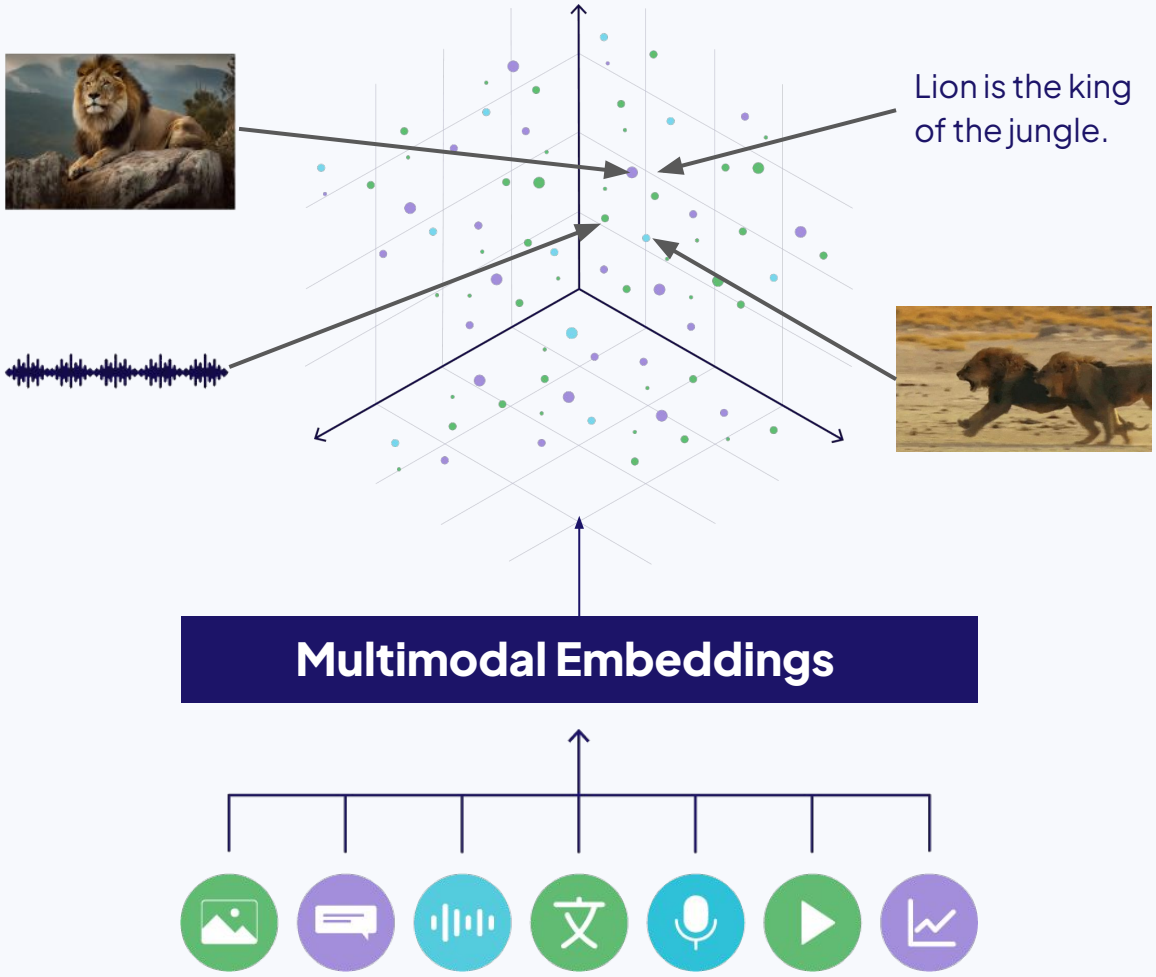


A Quick Demo 🚀

PDF search

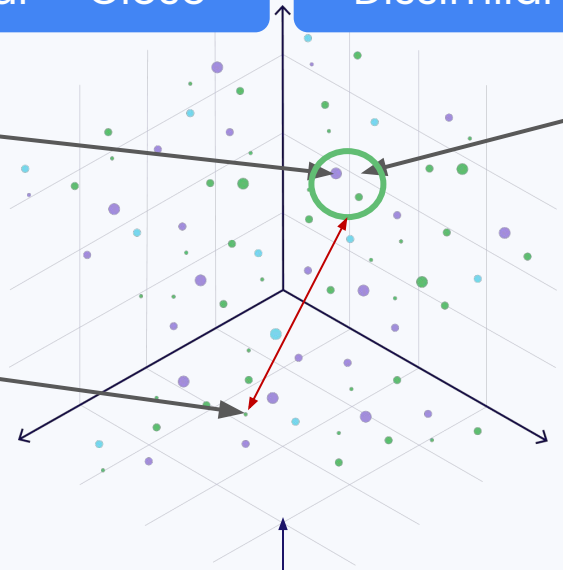
How does this work?




Lion is the king of the jungle.
Multimodal Embeddings

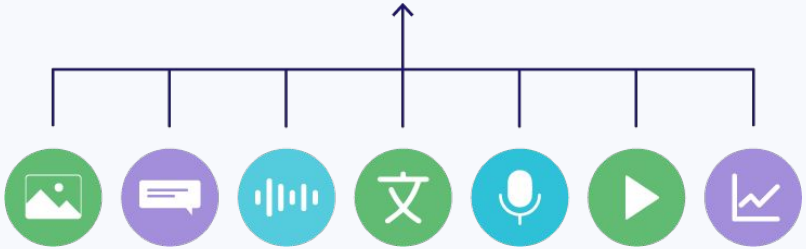
Similar = Close

Dissimilar = Far



Lion is the king of the jungle.

Multimodal Embeddings



Start with specialist models

The lion is the king of the savannas.

Text Encoder

Image Encoder

Audio Encoder

Video Encoder

[-0.10, 0.02, 0.14, ..., 0.48]

[0.66, 0.73, -0.42, ..., 0.52]

[-0.61, 0.31, 0.88, ..., 0.22]

[-0.21, 0.15, =0.27, ..., -0.59]

The lion is the king of the savannas.

Text Encoder

Image Encoder

Audio Encoder

Video Encoder

[-0.10, 0.02, 0.14, ..., 0.48]

[0.66, 0.03, 0.14, ..., 0.52]

[-0.11, 0.01, 0.15, ..., 0.25]

[-0.21, 0.05, 0.17, ..., -0.59]

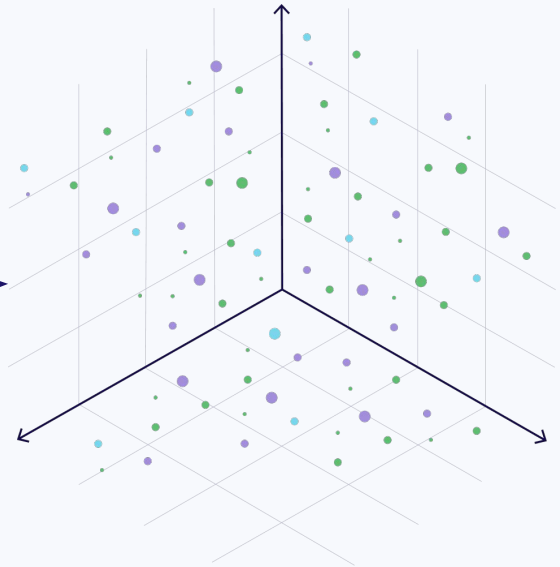
similar concepts = similar vectors

Text to Any Search

QUERY

RETRIEVED

Lions roam the savannas



King of the jungle.

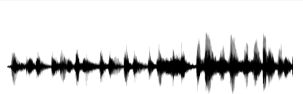
Any to Any Search

QUERY

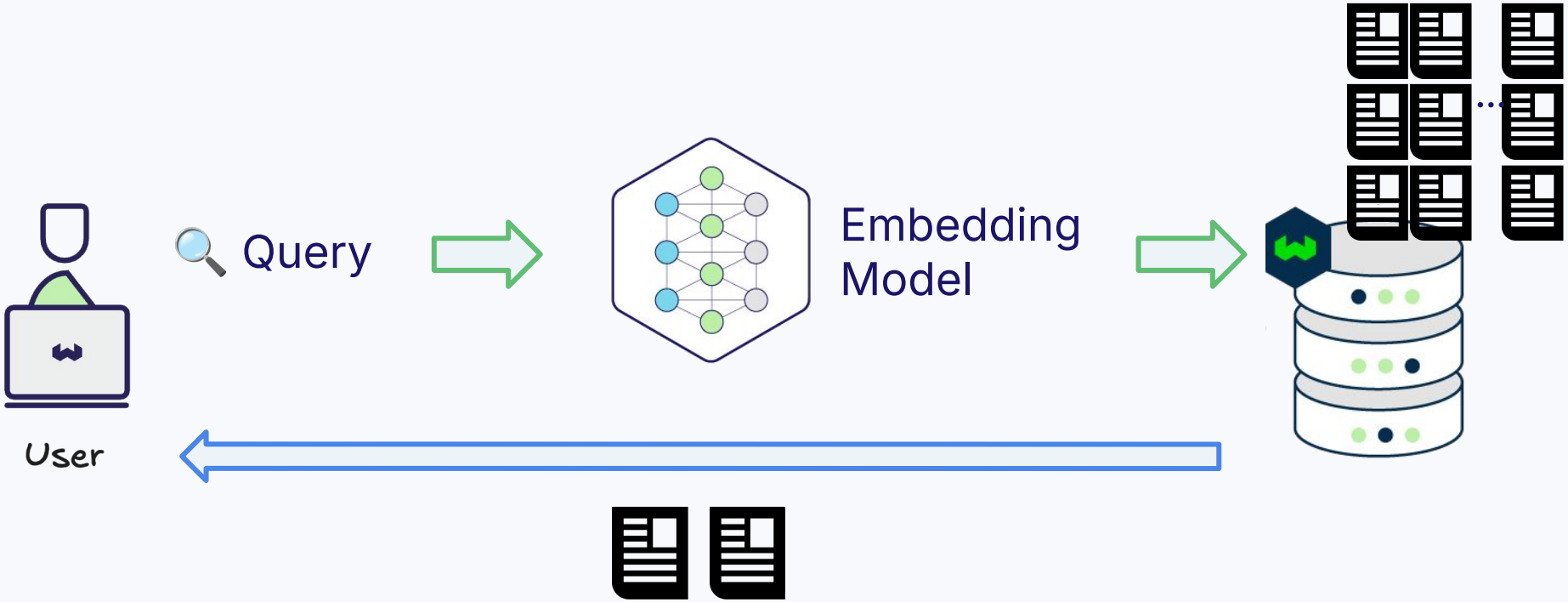
Lions roam the savannas.

RETRIEVED

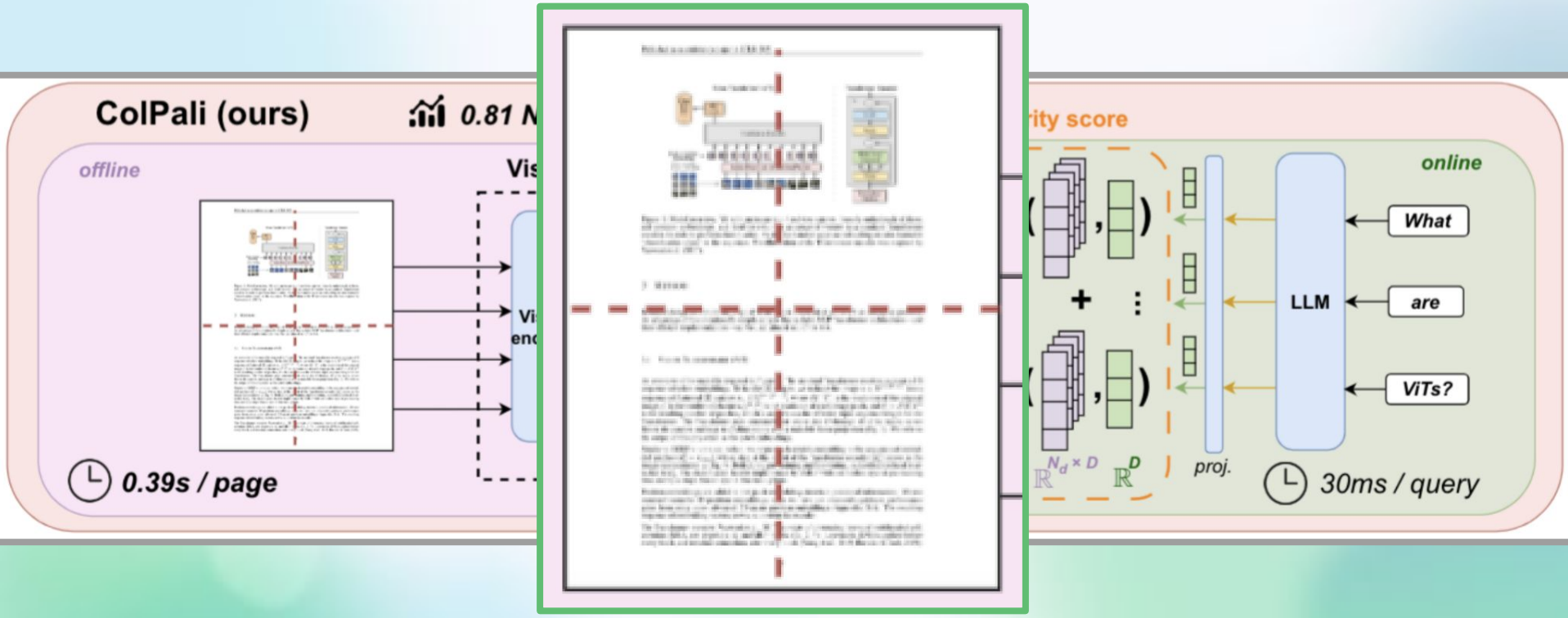
King of the jungle.



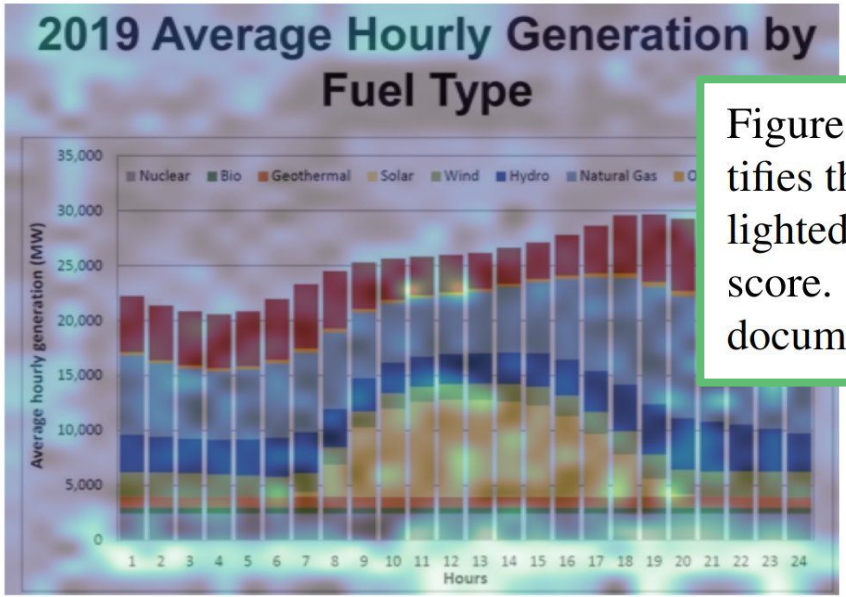
Multimodal retrieval



ColPali model



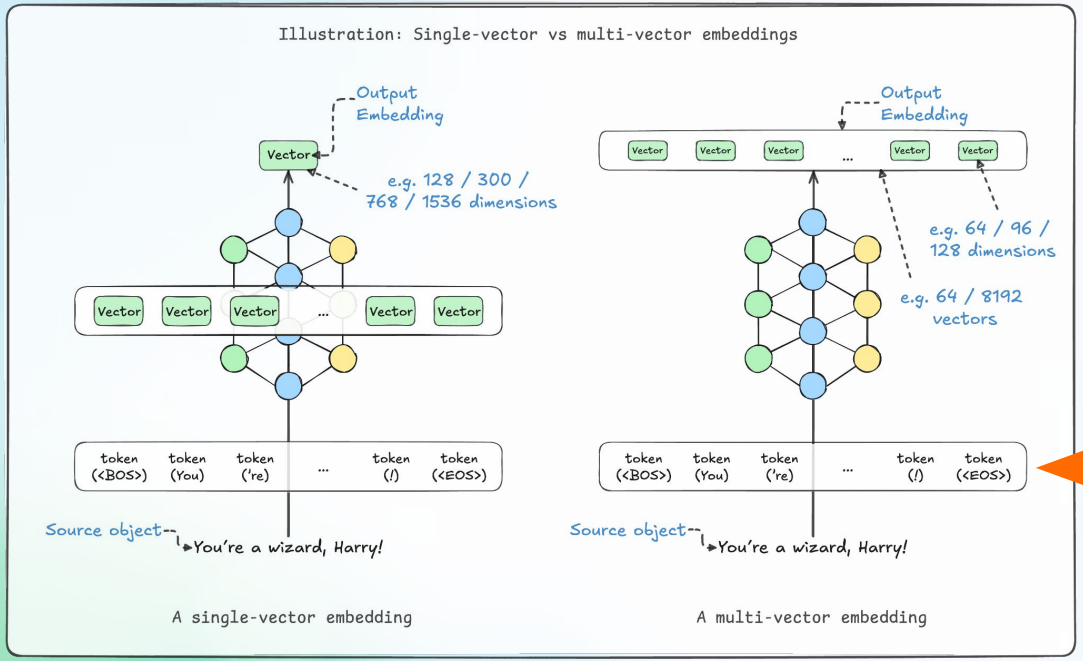
ColPali model: patches



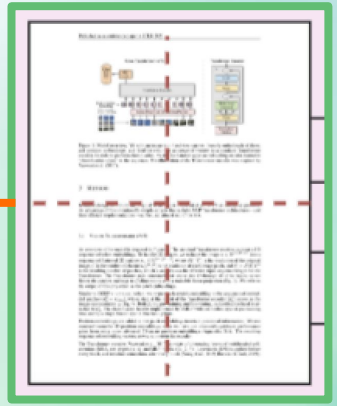
Query: "Which hour of the day had the highest overall eletricity generation in 2019?"

Figure 1: For each term in a user query, **ColPali** identifies the most relevant document image patches (highlighted zones) and computes a query-to-page matching score. We can then swiftly retrieve the most relevant documents from a large pre-indexed corpus.

Multi-vector embeddings

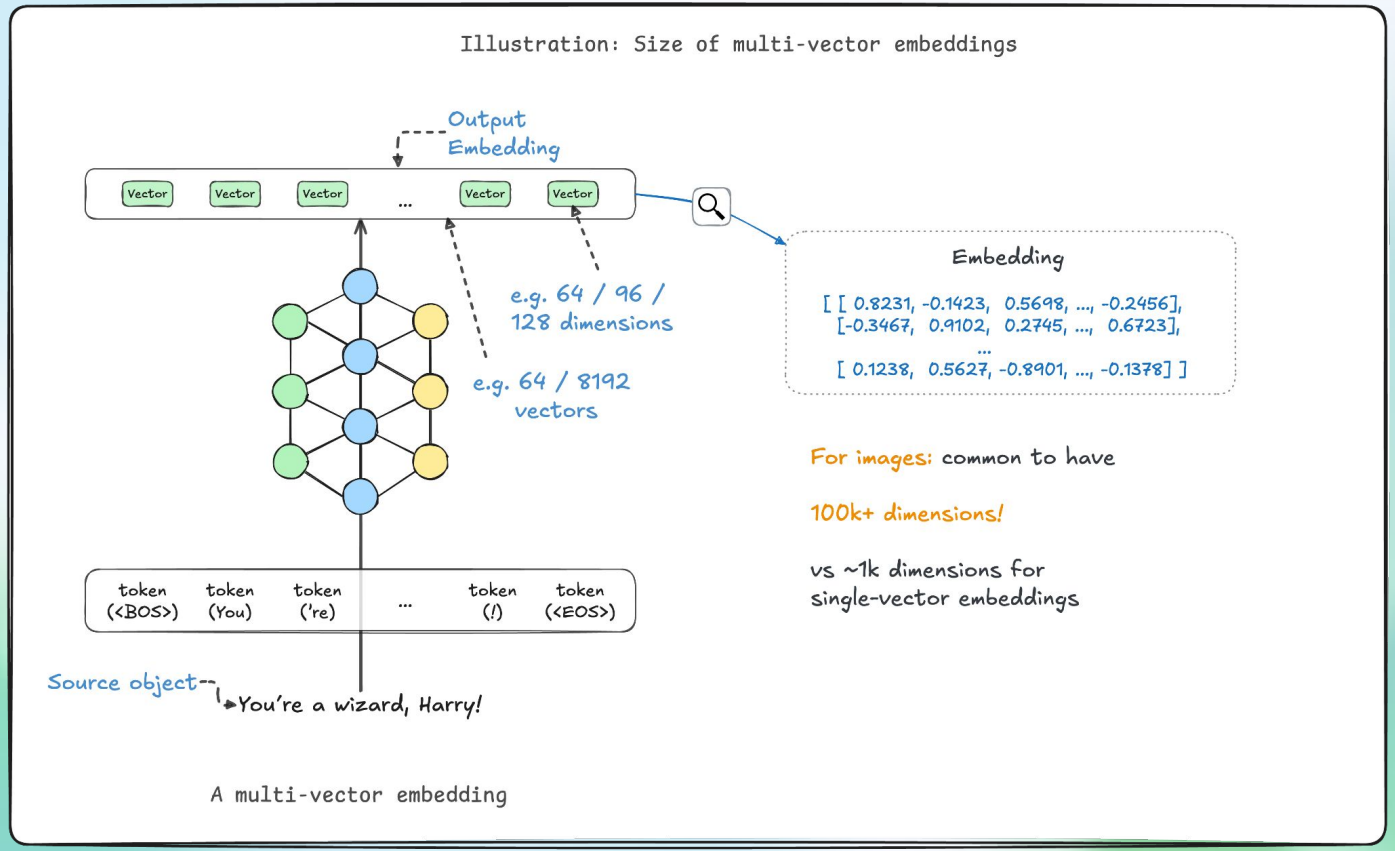


Each patch → token

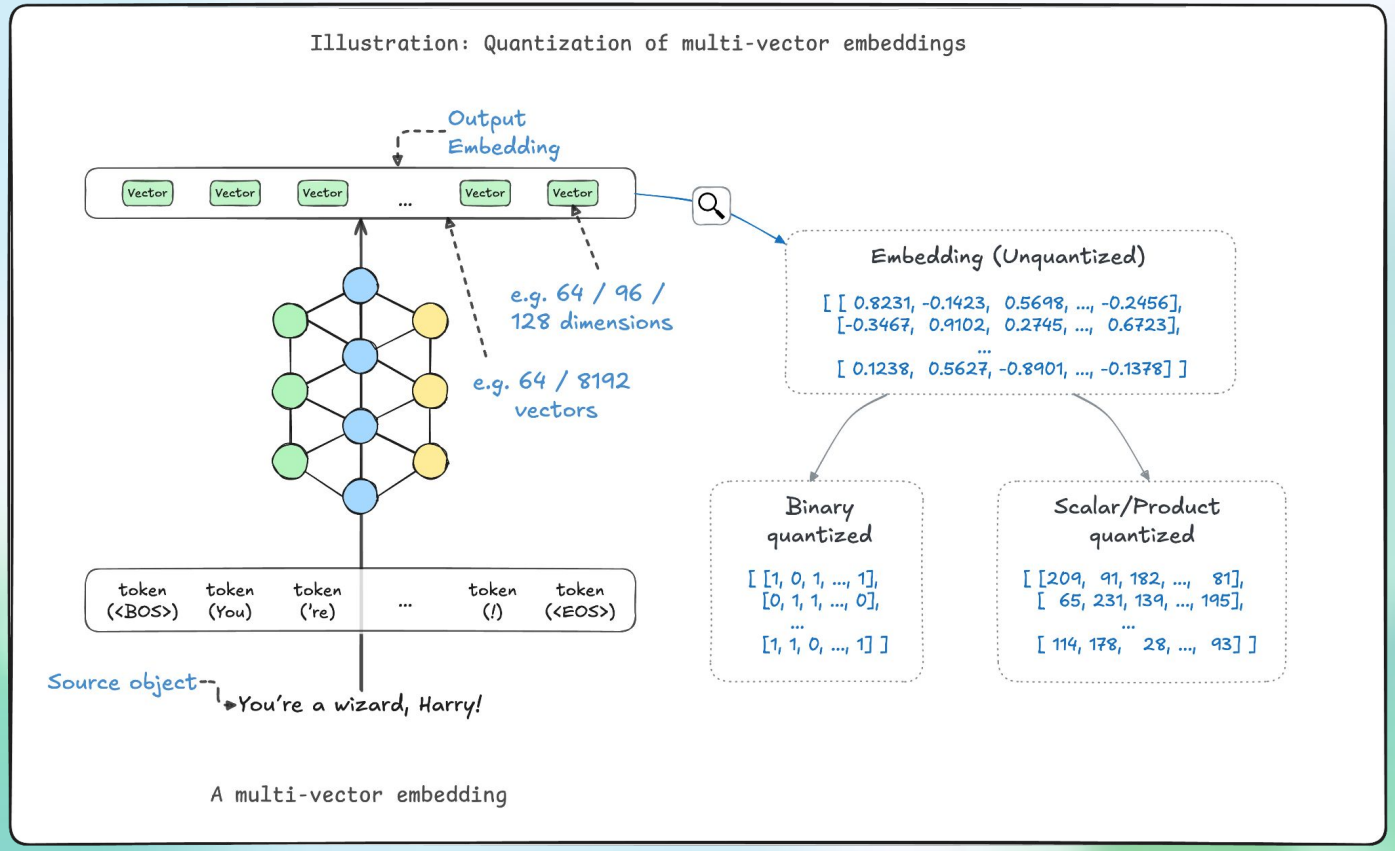


Scaling multi-vector models

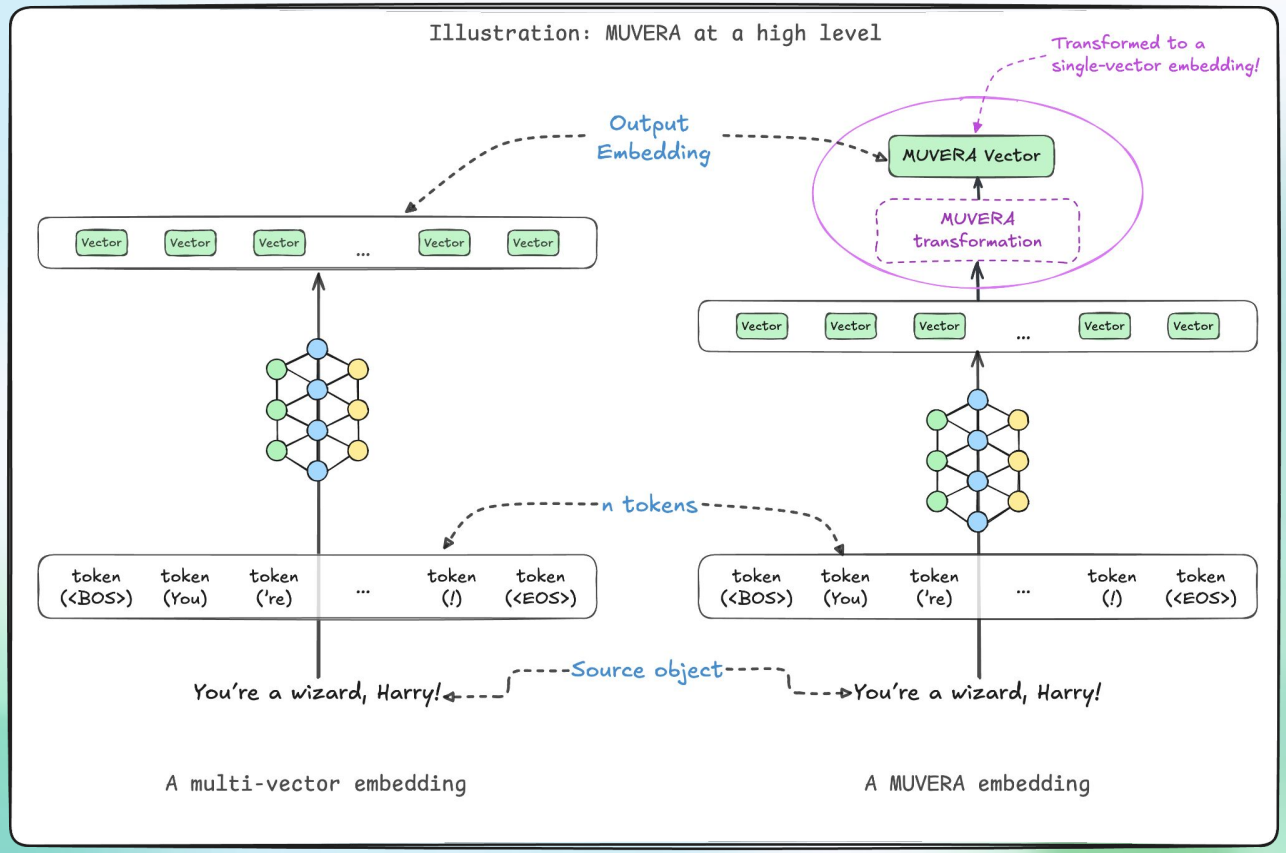
Multi-vector embeddings



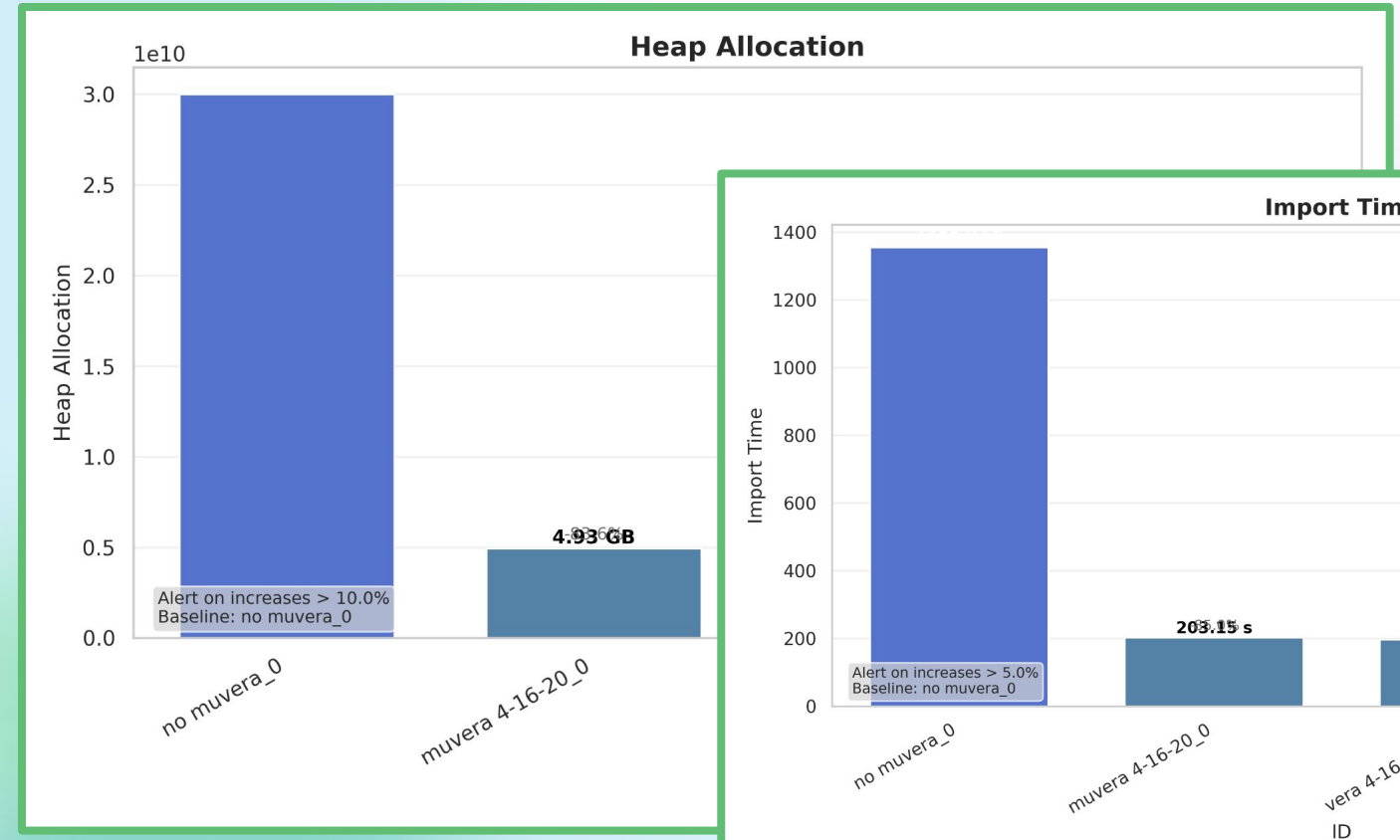
Multi-vector embeddings: Quantization



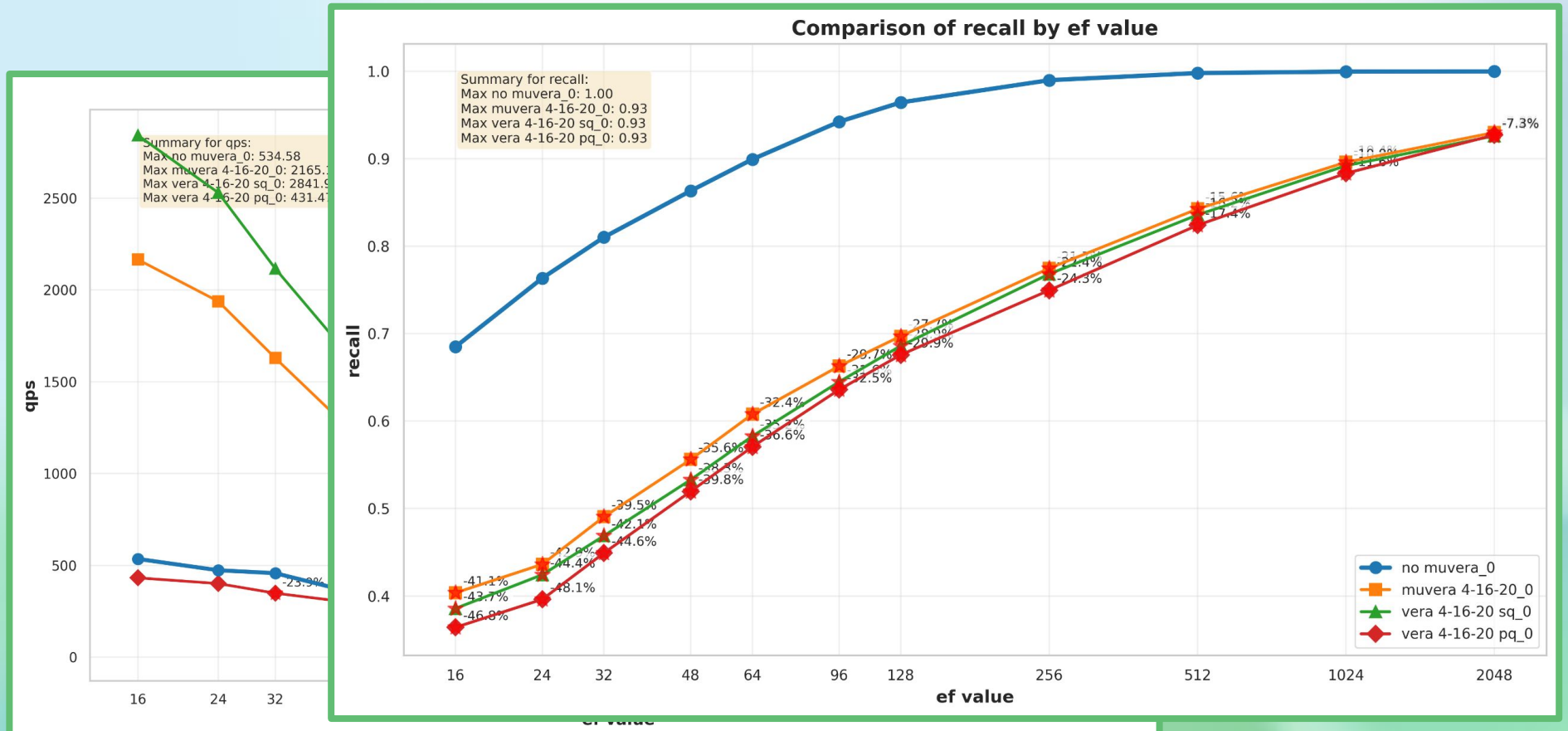
Multi-vector embeddings: MUVERA



Multi-vector embeddings: Resources



Multi-vector embeddings: Quality



Expanding to RAG

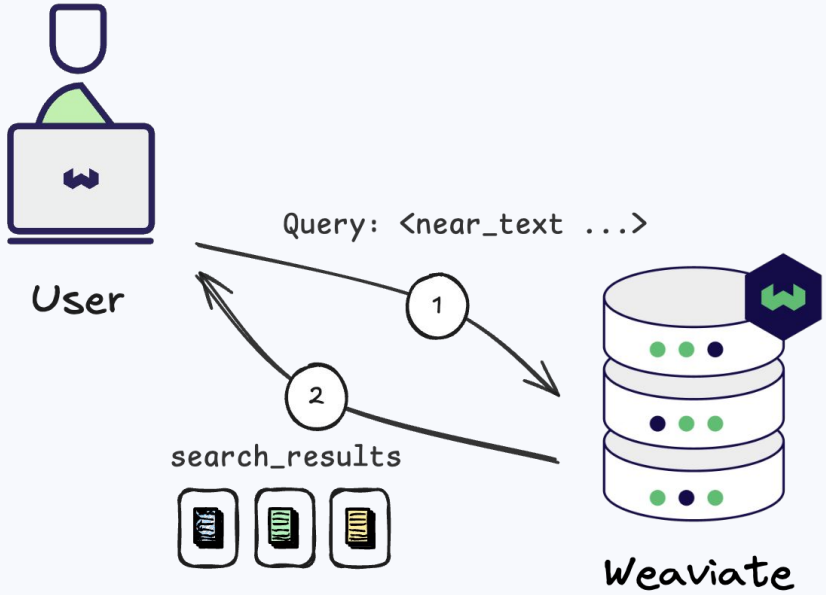
Another quick demo 🚀

What’s happening here?

Multimodal RAG

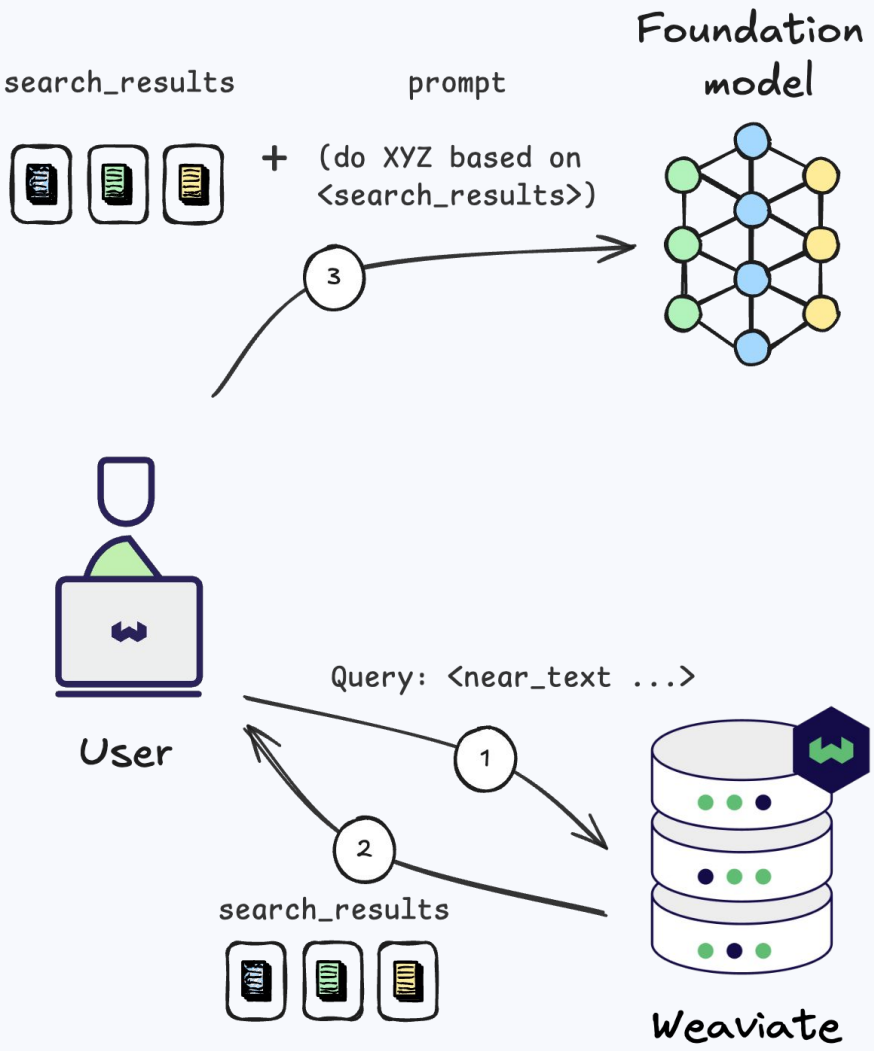
RAG

(Retrieval augmented generation)



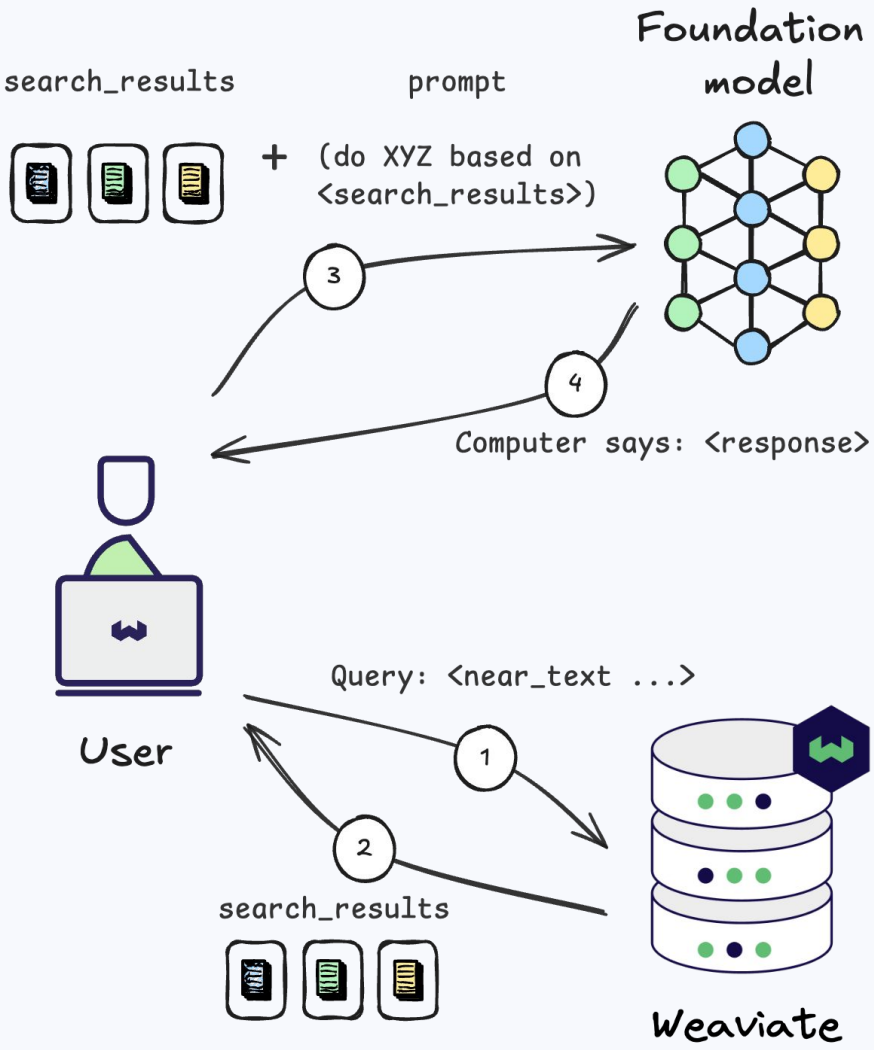
RAG

(Retrieval augmented generation)



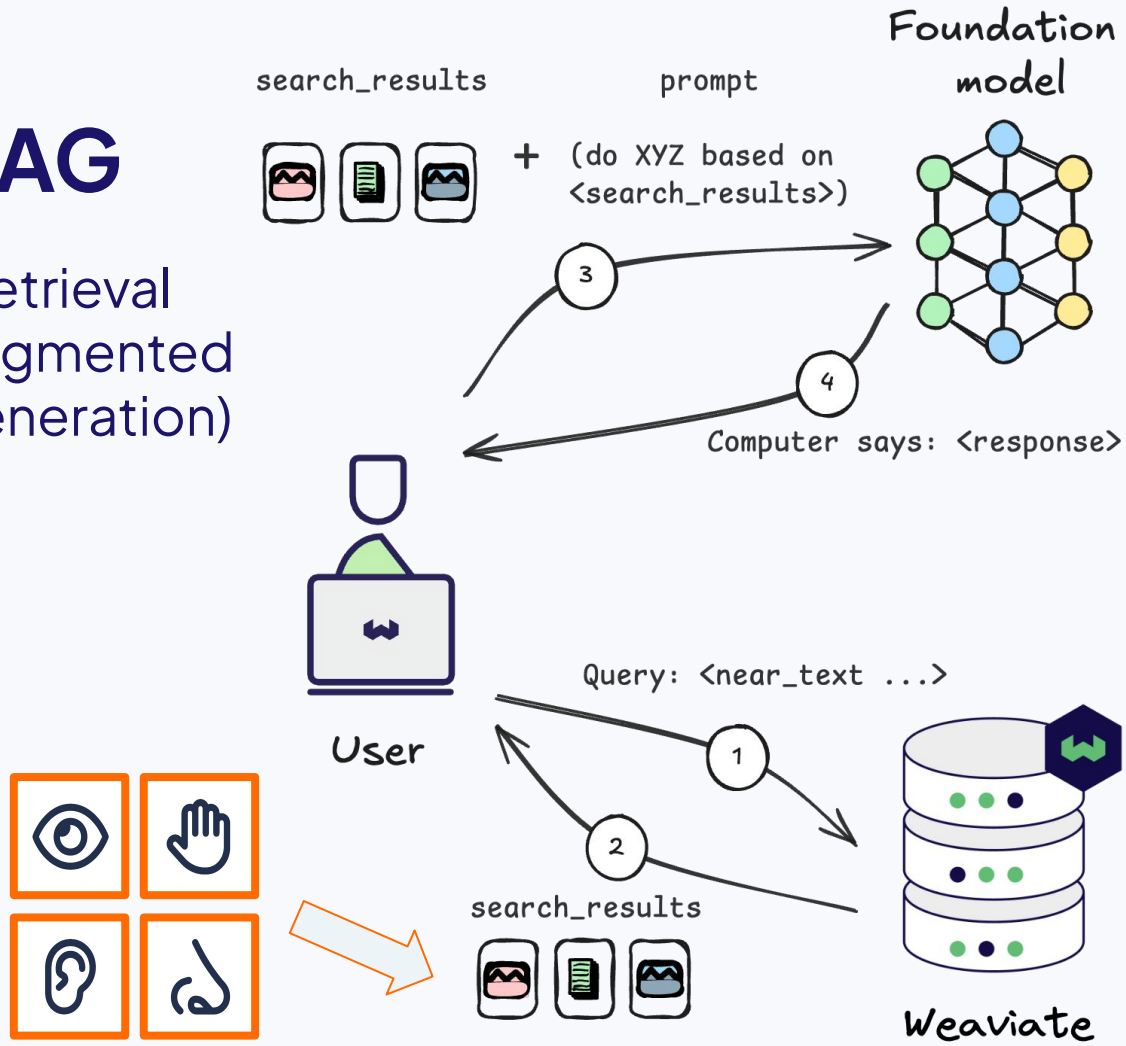
RAG

(Retrieval augmented generation)



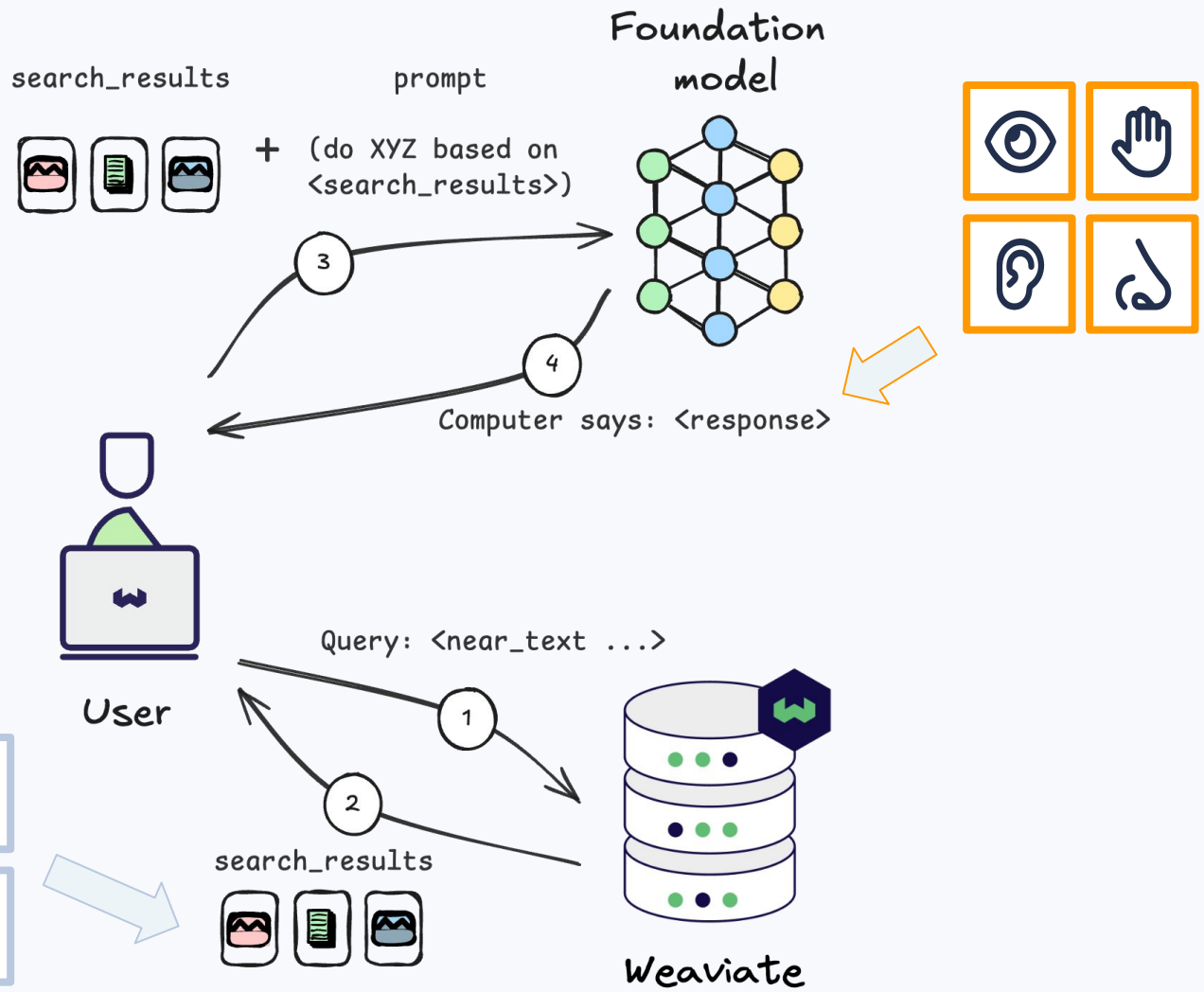
RAG

(Retrieval augmented generation)



RAG

(Retrieval augmented generation)



Suggested solution

search_results

prompt

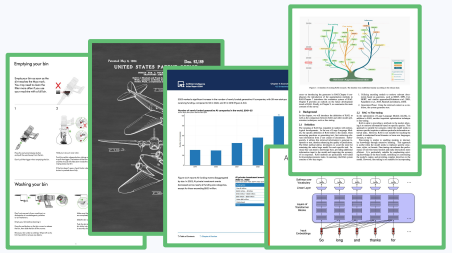
Foundation model

(do XYZ based on <search_results>)

3

4

Easier insights from data



User

Query: <near_text ...>

1

2

search_results

Weaviate

Embedding Model

Agentic AI Databases

1 A little bit of history of how we got to ‘Agentic AI’

Let’s define “Agents”

- A system that includes LLM capable of making decisions on the best course of action. (an LLM we’ve offloaded the burden of planning to)
- A system that can execute and interact with its environment based on generated course of action.

1 A little bit of history of how we got to ‘Agentic AI’

Let’s remember

- Language models do not know ✨everything✨..BUT..
- They work remarkably well with good instructions and the right context
- The instruction defines the task: QA, summarization, planning…?

1 A little bit of history of how we got to ‘Agentic AI’

ReAct: Reason & Act

★ Language models do not know ✨how✨ to do everything

REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS

Shunyu Yao*[,1], Jeffrey Zhao[2], Dian Yu[2], Nan Du[2], Izhak Shafran[2], Karthik Narasimhan[1], Yuan Cao[2]

[1]Department of Computer Science, Princeton University
[2]Google Research, Brain team
[1]{shunyuy,karthikn}@princeton.edu
[2]{jeffreyzhao,dianyu,dunan,izhak,yuancao}@google.com

ABSTRACT

While large language models (LLMs) have demonstrated impressive performance across tasks in language understanding and interactive decision making, their abilities for reasoning (e.g. chain-of-thought prompting) and acting (e.g. action plan generation) have primarily been studied as separate topics. In this paper, we explore the use of LLMs to generate both reasoning traces and task-specific actions in an interleaved manner, allowing for greater synergy between the two: reasoning traces help the model induce, track, and update action plans as well as handle exceptions, while actions allow it to interface with and gather additional information from external sources such as knowledge bases or environments. We apply our approach, named ReAct, to a diverse set of language and decision making tasks and demonstrate its effectiveness over state-of-the-art baselines in addition to improved human interpretability and trustworthiness. Concretely, on question answering (HotpotQA) and fact verification (Fever), ReAct overcomes prevalent issues of hallucination and error propagation in chain-of-thought reasoning by interacting with a simple Wikipedia API, and generating human-like task-solving trajectories that are more interpretable than baselines without reasoning traces. Furthermore, on two interactive decision making benchmarks (ALFWorld and WebShop), ReAct outperforms imitation and reinforcement learning methods by an absolute success rate of 34% and 10% respectively, while being prompted with only one or two in-context examples.

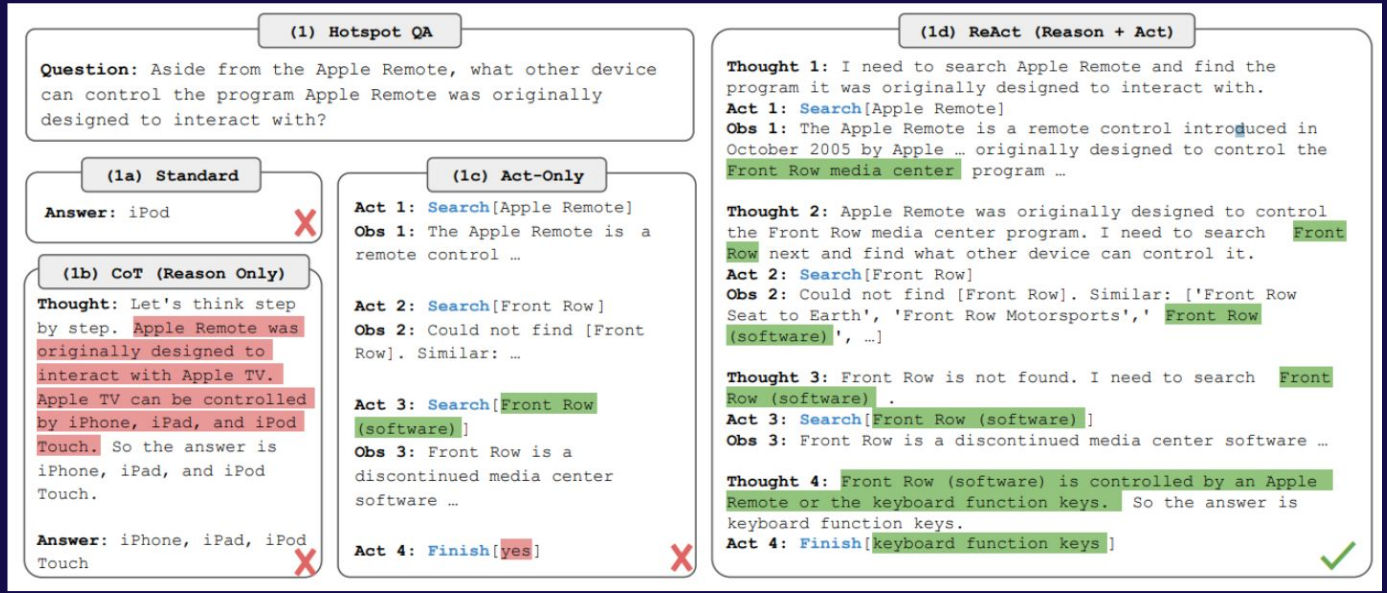
1 INTRODUCTION

A unique feature of human intelligence is the ability to seamlessly combine task-oriented actions with verbal reasoning (or inner speech, Alderson-Day & Fernyhough, 2015), which has been theorized to play an important role in human cognition for enabling self-regulation or strategization (Vygotsky, 1987; Luria, 1965; Fernyhough, 2010) and maintaining a working memory (Baddeley, 1992). Con-

1 A little bit of history of how we got to ‘Agentic AI’

ReAct: Reason & Act

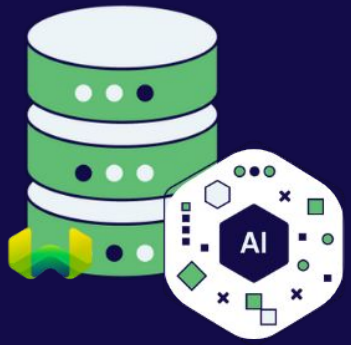
★ Given the right instruction, and access to the right ✨tools✨ – they may reason about what action to take to solve a task



2 Functions/Tool Calling

- Models can “reason” about which tool to run, and..
- They can produce an output about *how* to run the tool

2 Functions/Tool Calling



Name: weaviate_docs_rag
Description: Useful for finding answers in Weaviate documentation
Inputs: {query: str}
Outputs: {answer:str}

Name: weather_tool
Description: Useful for weather information
Inputs: {city: str, country: str, date: datetime}
Outputs: {temp: int, unit: str, location: str}

2 Functions/Tool Calling

Name: weaviate_docs_rag
Description: Useful for finding answers in Weaviate documentation
Inputs: {query: str}
Outputs: {answer:str}

Name: weather_tool
Description: Useful for weather information
Inputs: {city: str, country: str, date: datetime}
Outputs: {temp: int, unit: str, location: str}

2 Functions/Tool Calling

Name: weaviate_docs_rag
Description: Useful for finding answers in Weaviate documentation
Inputs: {query: str}
Outputs: {answer:str}

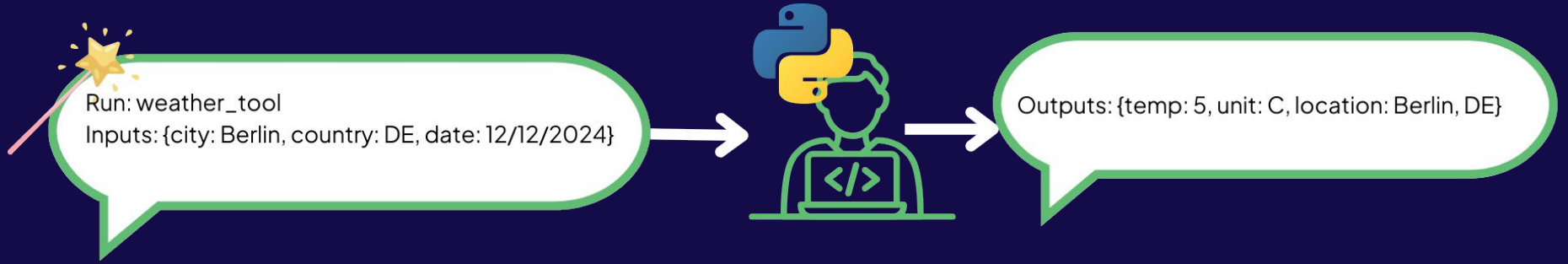
Name: weather_tool
Description: Useful for weather information
Inputs: {city: str, country: str, date: datetime}
Outputs: {temp: int, unit: str, location: str}

Run: weather_tool
Inputs: {city: Berlin, country: DE, date: 12/12/2024}

The execution is up to us

2 Functions/Tool Calling

Tool Calling Step



2 Functions/Tool Calling

Observation Step

Outputs: {temp: 5, unit: C, location: Berlin, DE}

2 Functions/Tool Calling

Thought & Answer Step

The question was about what the weather is like in Berlin and it seems and I have the answer: 5C

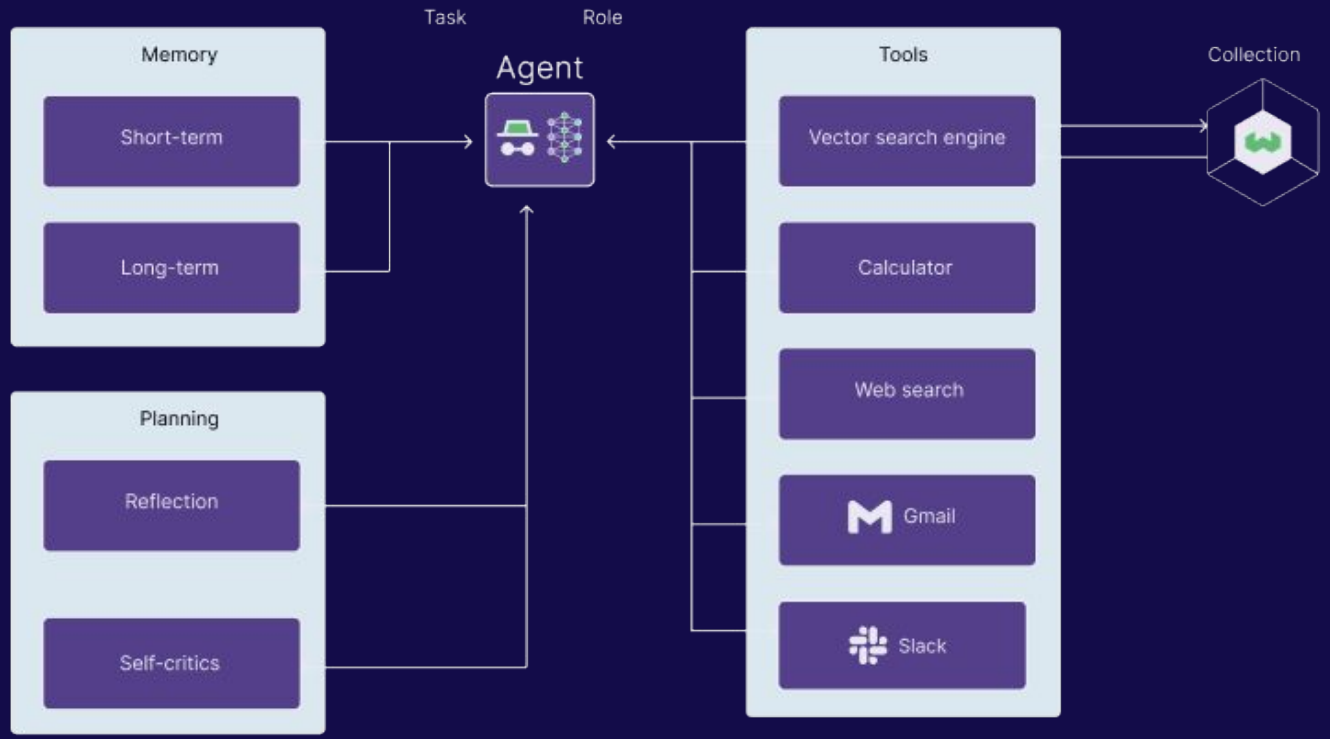
The weather in Berlin is 5 degrees Celcius

2 Functions/Tool Calling

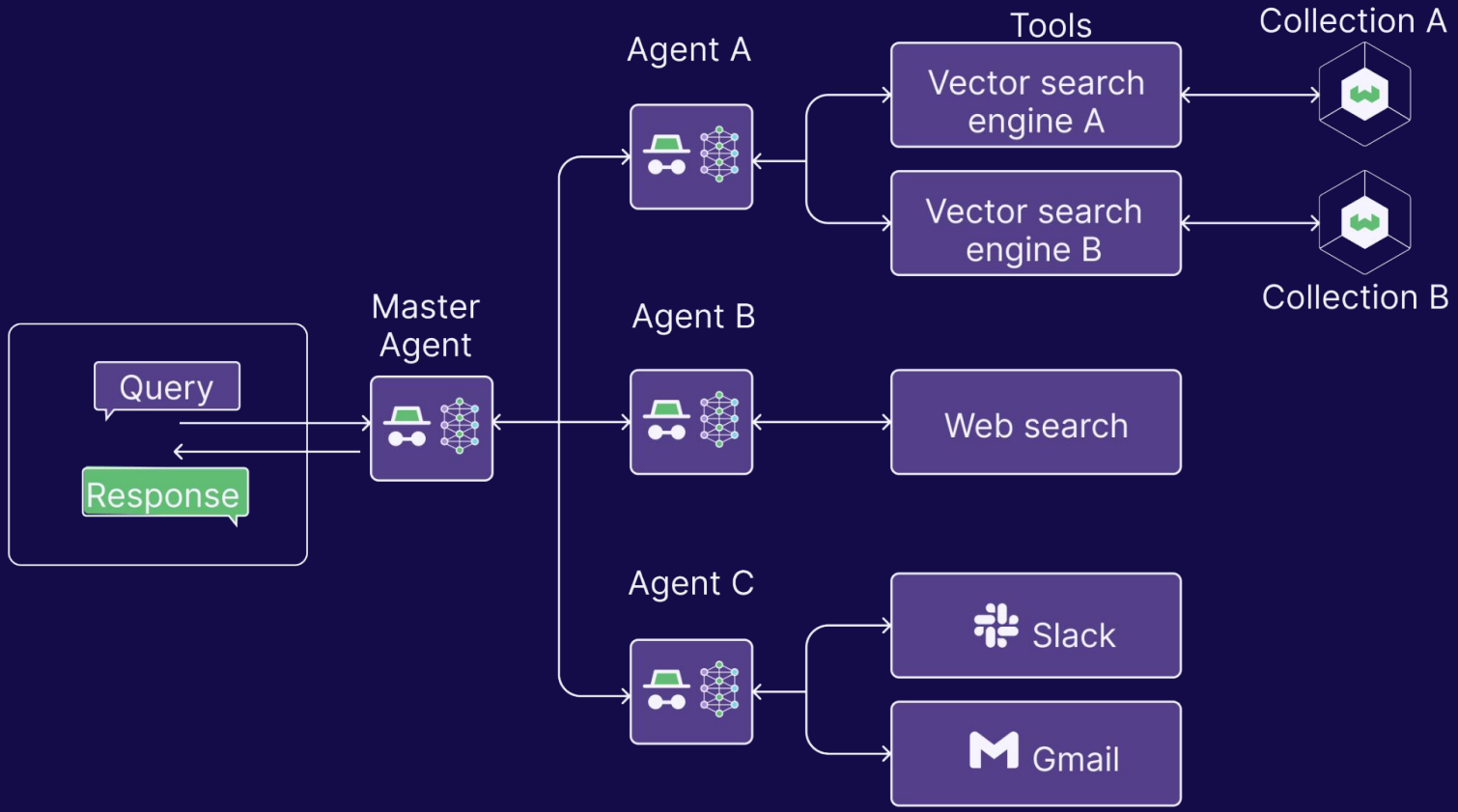
```python
from openai import OpenAI

client = OpenAI()

tools = [
    {
        "type": "function",
        "function": {
            "name": "get_delivery_date",
            "description": "Get the delivery date for a customer's order. Call this whenever you
                            need to know the delivery date, for example when a customer asks
                            'Where is my package'",
            "parameters": {
                "type": "object",
                "properties": {
                    "order_id": {
                        "type": "string",
                        "description": "The customer's order ID.",
                    },
                },
                "required": ["order_id"],
                "additionalProperties": False,
            },
        }
    }
]
```

OpenAI

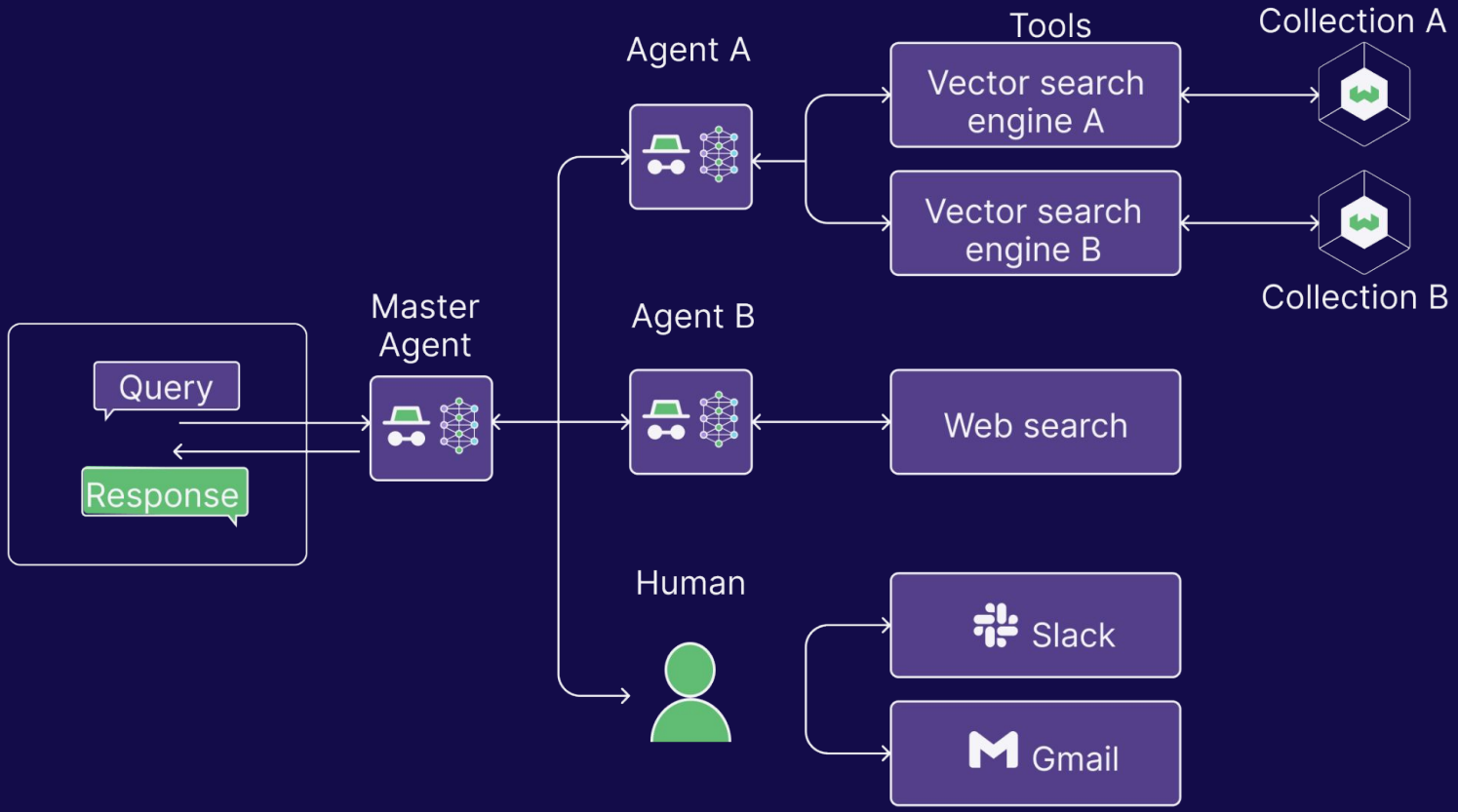
✨ Agents ✨


Task
Role
Agent
Memory
Short-term
Long-term
Planning
Reflection
Self-critics
Tools
Vector search engine
Calculator
Web search
Gmail
Slack
Collection

✨ Agents ✨


Tools
Collection A
Agent A
Vector search engine A
Vector search engine B
Collection B
Master Agent
Agent B
Query
Response
Web search
Agent C
Slack
Gmail

✨ Agents ✨


Agent A
Tools
Collection A
Vector search engine A
Vector search engine B
Collection B
Master Agent
Agent B
Query
Response
Web search
Human
Slack
Gmail

Quick notes

Weaviate at a high level +
Features we didn’t cover

(Multi-) Tenancy

Single vs multi-tenancy architectures. Use cases for each, and how to manage resources

Unique Support of Single & Multi-Tenant Use Cases

Single tenant

When your use case requires all data in a single collection

- Store and search through **Multi-billion scale** collections
- Process millions of updates on a daily basis
- Smart (re) vectorization techniques
- Combine keyword, and vector search for powerful **hybrid search**
- Sharding and replication allows for tuning throughput
- Fast filtering capabilities to narrow down results

Multi tenant

When your use case needs to isolate data per user

- Scales to **millions** of tenants
- Full isolation
- Tenants can be **Cold, Warm**, or **Hot**, depending on activity
- Control tenant status through APIs or automatically
- Store Frozen tenants on cloud storage to reduce cost
- Allows for on-disk tenants and in-memory depending on use case

Split Data per (Domain)

User
Each user with their own data

Customer
White labelled-solutions

Team / Department
Separate HR docs from legal docs

Split Data per (Resource)

PDF
Tenant for PDF chunks

Video
Tenant for Video chunks

Group of Documents
Tenant per topic

Videos
Tenant per set of videos
(i.e. course lessons)

Split Data per (Time)

Daily
Tenant per day

Monthly
Tenant per month

Built-in Multi-tenancy

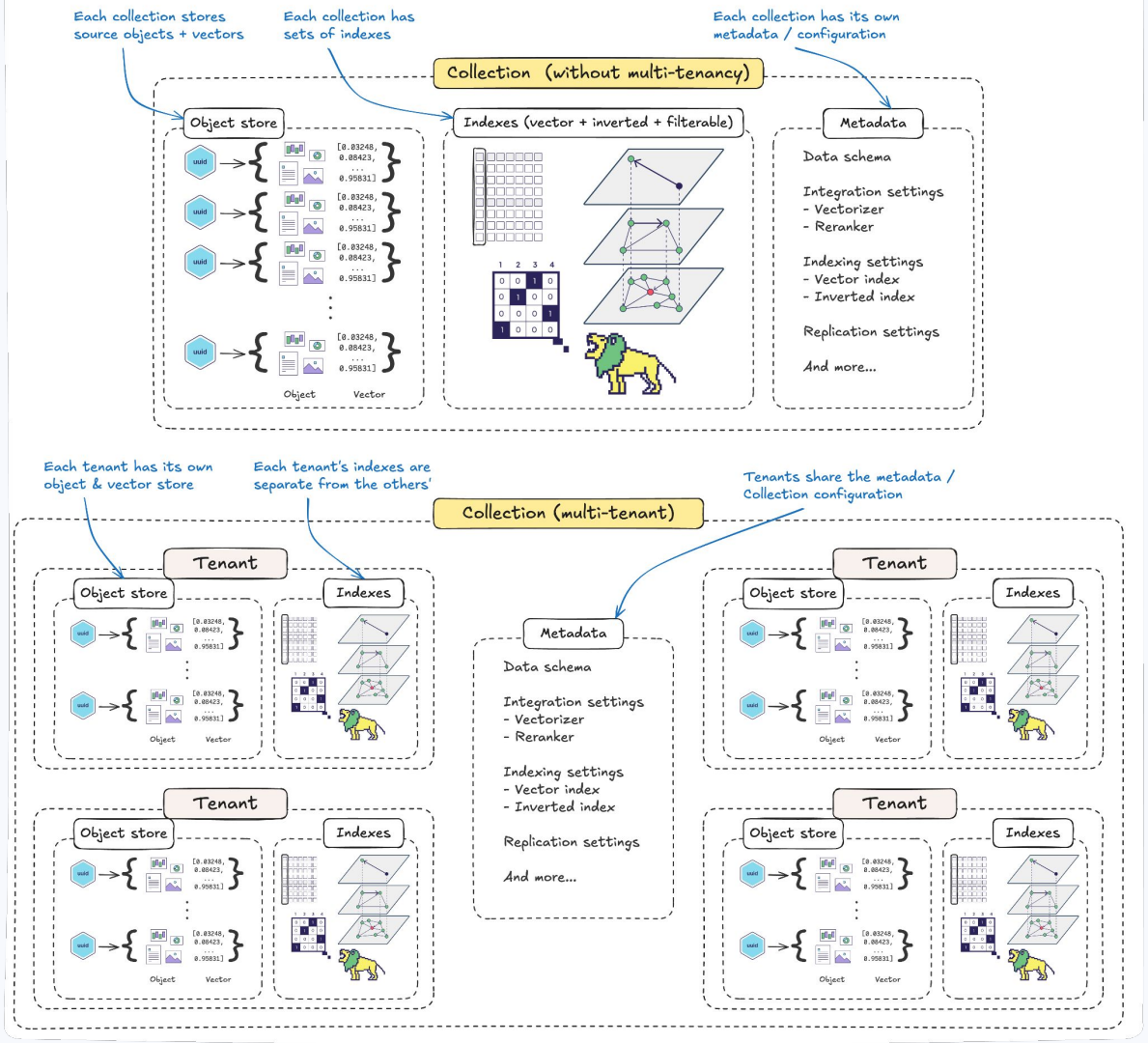
Smaller Indexes
Index per tenant

Faster queries
Less to search

GDPR compliant
For User tenants

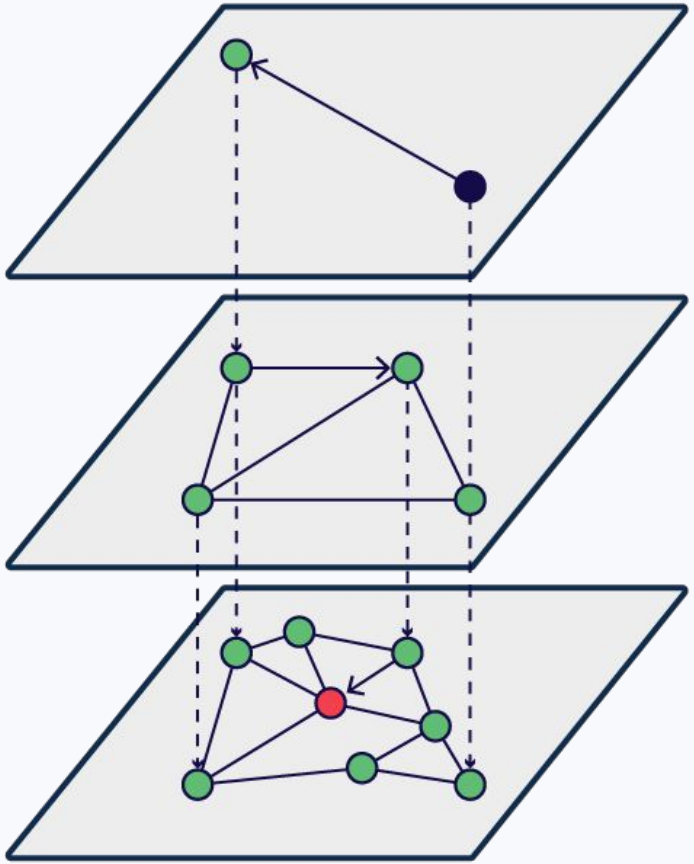
Tenant memory management
Hot and Cold

Flat Index - friendly
Save RAM


Each collection stores source objects + vectors
Each collection has sets of indexes
Each collection has its own metadata / configuration
Collection (without multi-tenancy)
Object store
Indexes (vector + inverted + filterable)
Metadata
Data schema
Integration settings
- Vectorizer
- Reranker
Indexing settings
- Vector index
- Inverted index
Replication settings
And more...
Object
Vector
Each tenant has its own object & vector store
Each tenant's indexes are separate from the others'
Tenants share the metadata / Collection configuration
Collection (multi-tenant)
Tenant
Object store
Indexes
Metadata
Data schema
Integration settings
- Vectorizer
- Reranker
Indexing settings
- Vector index
- Inverted index
Replication settings
And more...
Object
Vector

Indexing techniques

- **HNSW:** high-performance, in-memory. Scales well, even for very large data sets.
- **Flat:** disk-based indexes that perform brute-force vector searches. Best for small data sets.
- **Dynamic:** Automatically **s**witches from flat to HNSW when a collection (or tenant) reaches a threshold size.



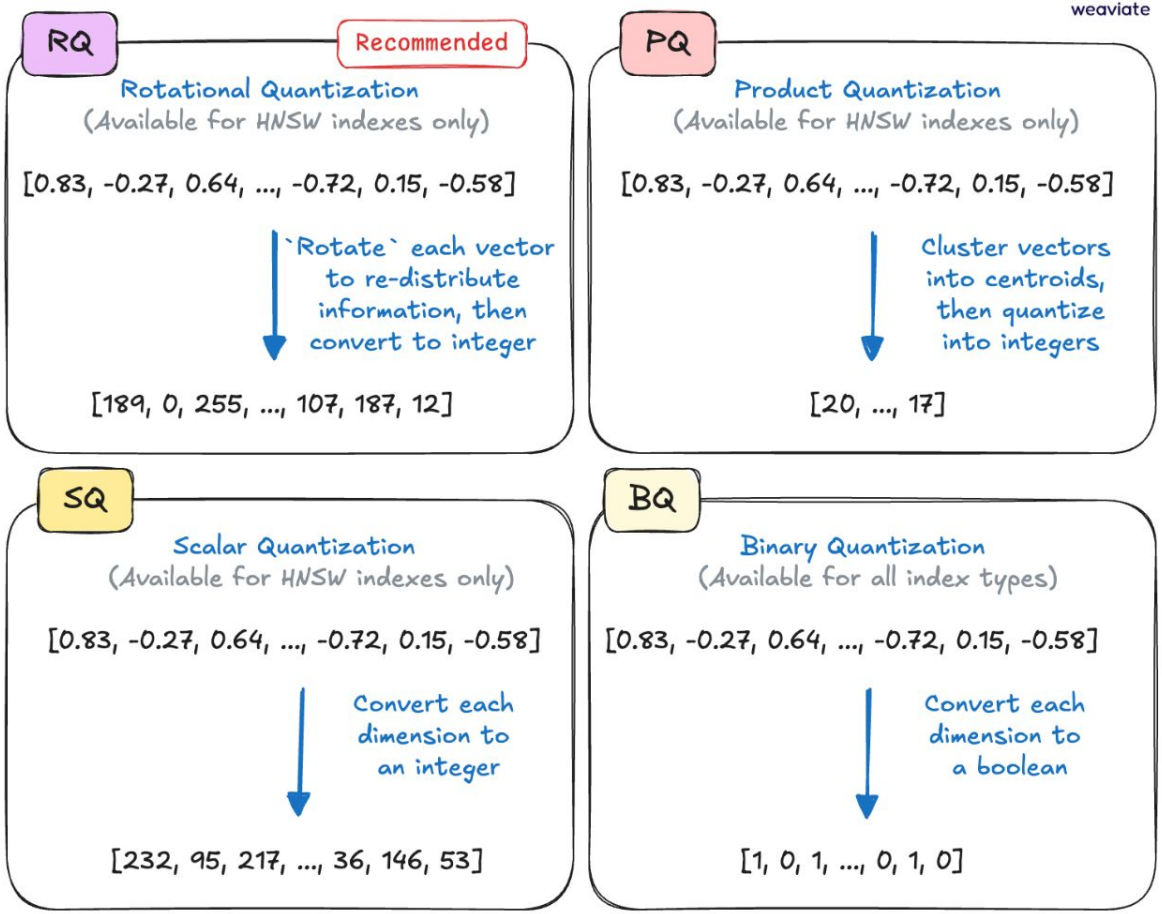
Quantization at a glance

Vectors can be quantized to reduce their size and improve search speeds.

Weaviate offers a choice of four quantization algorithms:

- Rotational
- Product
- Scalar
- Binary

Generally, RQ (or PQ) is the best choices for most users.



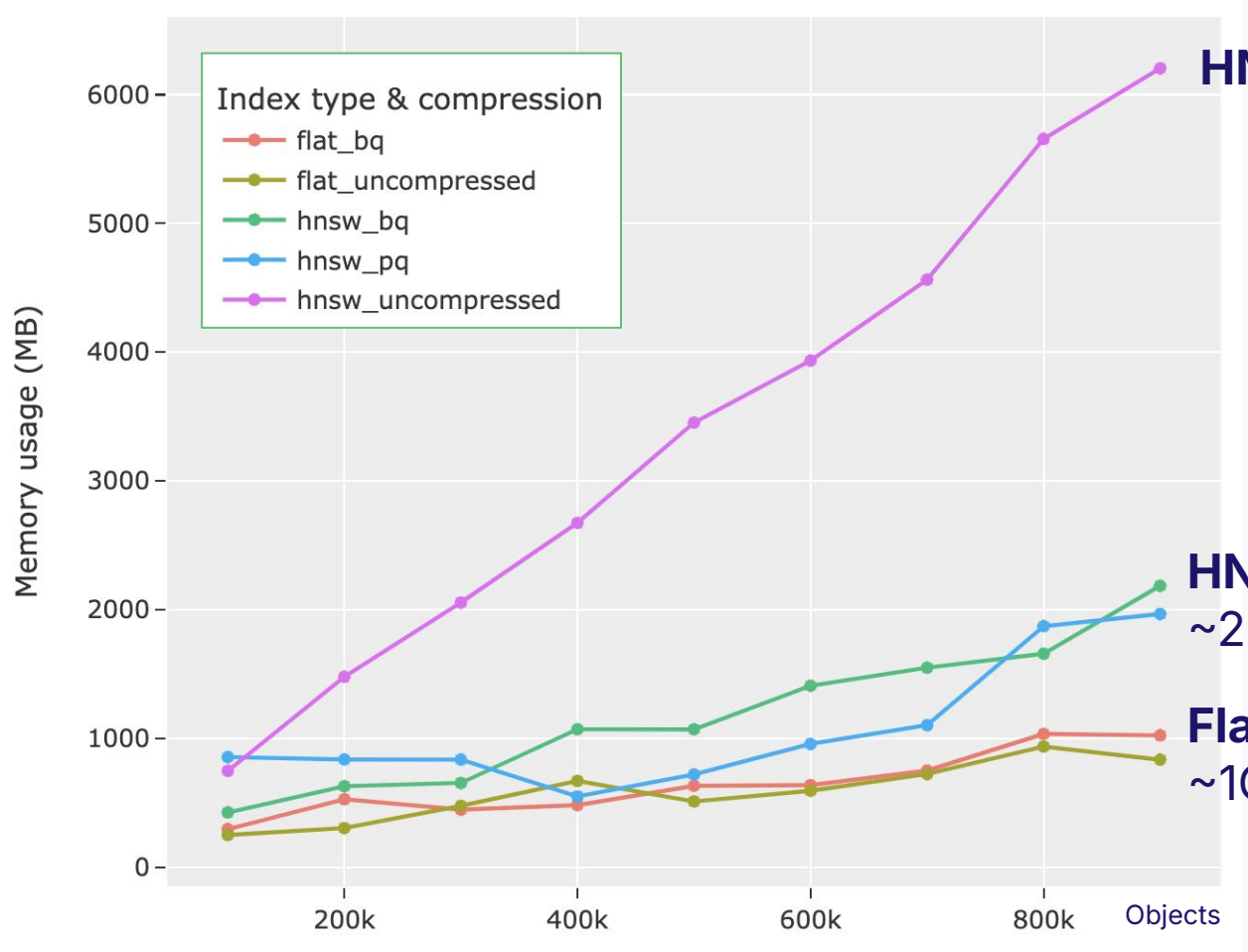
1024 gib

19 matches

Instance name	On-Demand hourly rate	vCPU	Memory	Storage
x2gd.metal	$5.344	64	1024 GiB	2 x 1900 SSD
x2gd.16xlarge	$5.344	64	1024 GiB	2 x 1900 SSD
hpc6id.32xlarge	$5.70	64	1024 GiB	4 x 3800 NVMe SSD
x2iedn.8xlarge	$6.669	32	1024 GiB	1 x 950 NVMe SSD
x2idn.16xlarge	$6.669	64	1024 GiB	1 x 1900 NVMe SSD

AWS Spot pricing, Nov 2024

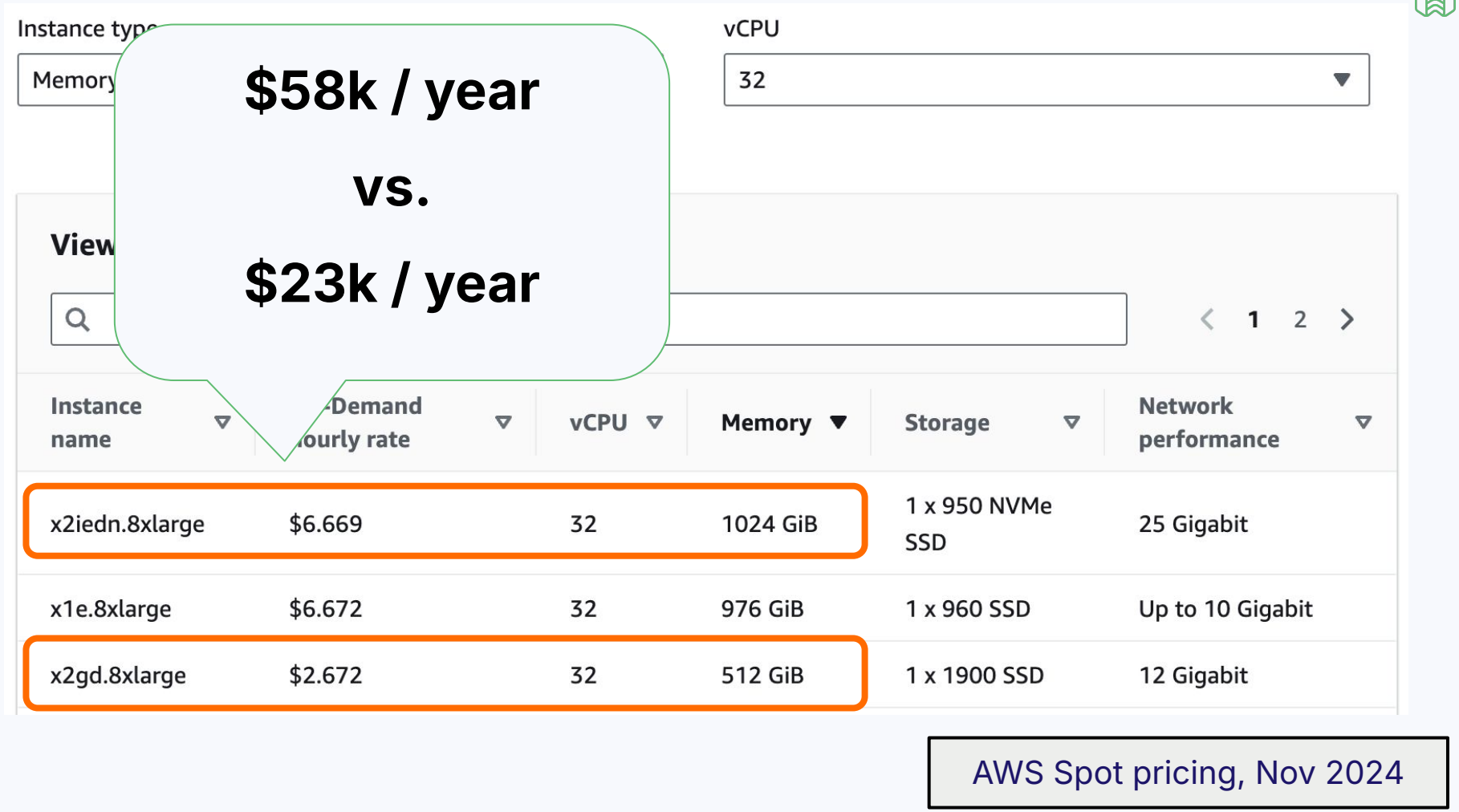
Example: Index type & quantization vs memory footprint



HNSW: ~6GB

HNSW + BQ/PQ: ~2GB

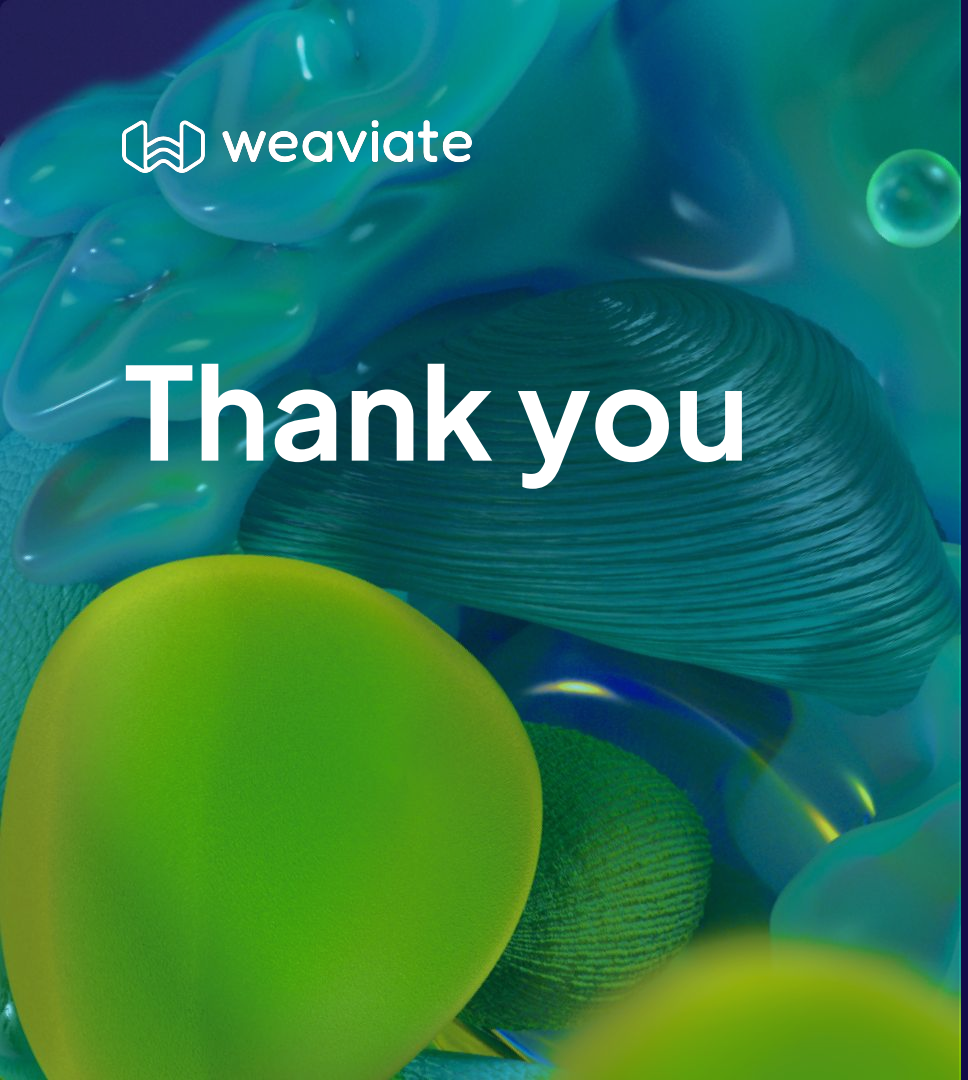
Flat (+BQ): ~1GB



AWS Spot pricing, Nov 2024

Q&A

Ask us anything!



weaviate.io

weaviate/weaviate

linkedin

@weaviate_io