

Vector search

George Panchuk
HSE AI Fall 2025



github.com/joein/vector-search-course

George Panchuk



ML Team Lead @ Qdrant 2022 - ...
qdrant-client and fastembed maintainer



HSE MLDS (AI) Alumni 2023
Thesis supervisor 2023 - ...



@jmzzomg



github.com/joein



Course structure

10 lectures, optional seminars (on demand)

Formulae:

$$0.8 * HW + 0.2 * (\text{Bonus HW} \parallel \text{vector search talk})$$

HW:

$$0.1 * HW1 + 0.2 * HW2 + 0.2 * HW3 + 0.1 * HW4 + 0.2 * HW5 + 0.1 * HW6$$

Late submission multiplier:

≤ 1 week late: 0.75

> 1 week late: 0.5

HW1: Brute-force search (13.11, 23.11)

HW2: ANN Search (26.11, 7.12)

HW3: Vector search engines (3.12, 14.12)

HW4: Payload filters (4.12, 14.12)

HW5: Sparse models, late interaction models (10.12, 18.12)

HW6: Hybrid search (11.12, 18.12)

Bonus HW: Solve an embedding-based game (1.12, 20.12)

Course assistant:
Arsenii Korol



@senyafeelsgood

How to submit a task?

- Create a private repository at GitHub.com
- Invite GitHub.com/joein and GitHub.com/senyafeelsgood to become a contributor
- Each hw should be done in its own branch, e.g. hw-1
- Once a task is ready for review:
 - Create a pull request to main
 - Request a review from both of us*
 - Once a PR is approved by 1 of us - merge
- If the task was sent 5 days before the first deadline, you can receive comments to address in order to improve the final mark**

* Counted as a date of submission

** Time to address the comments is max(2 days, deadline date), comments can be received only 1 time per hw

Task requirements

- Complete the task
- Have a clean repository
- Write clean well-structured code
(pre-commit hooks, ruff, type hints, etc)
- Python version: 3.10+
- Package manager: uv
- Each hw has to have a dependency group,
common dependencies can be put without a
group
- Readme with installation instructions

Repository structure:

The screenshot shows a GitHub repository named "course-template". The repository is private and has 1 branch and 0 tags. It contains 2 commits from user "joein" made "now". The commits are:

- new: keep directories (commit 62ff185)
- new: keep directories (commit 62ff185)

The repository structure includes:

- bonus_hw
- hw_1
- hw_2
- hw_3
- hw_4
- hw_5
- hw_6
- .python-version
- README.md
- pyproject.toml
- uv.lock

The README file contains:

How to install dependencies:

```
uv install # the actual command will include groups
```

What this course is about?

What this course is about

- Applications of vector search in different scenarios and modalities
- Various vector search algorithms, their advantages and disadvantages
- Understanding, building and evaluating retrieval pipelines (including multistep ones)
- Optimizations for real world use cases
- Vector search beyond just k-nn

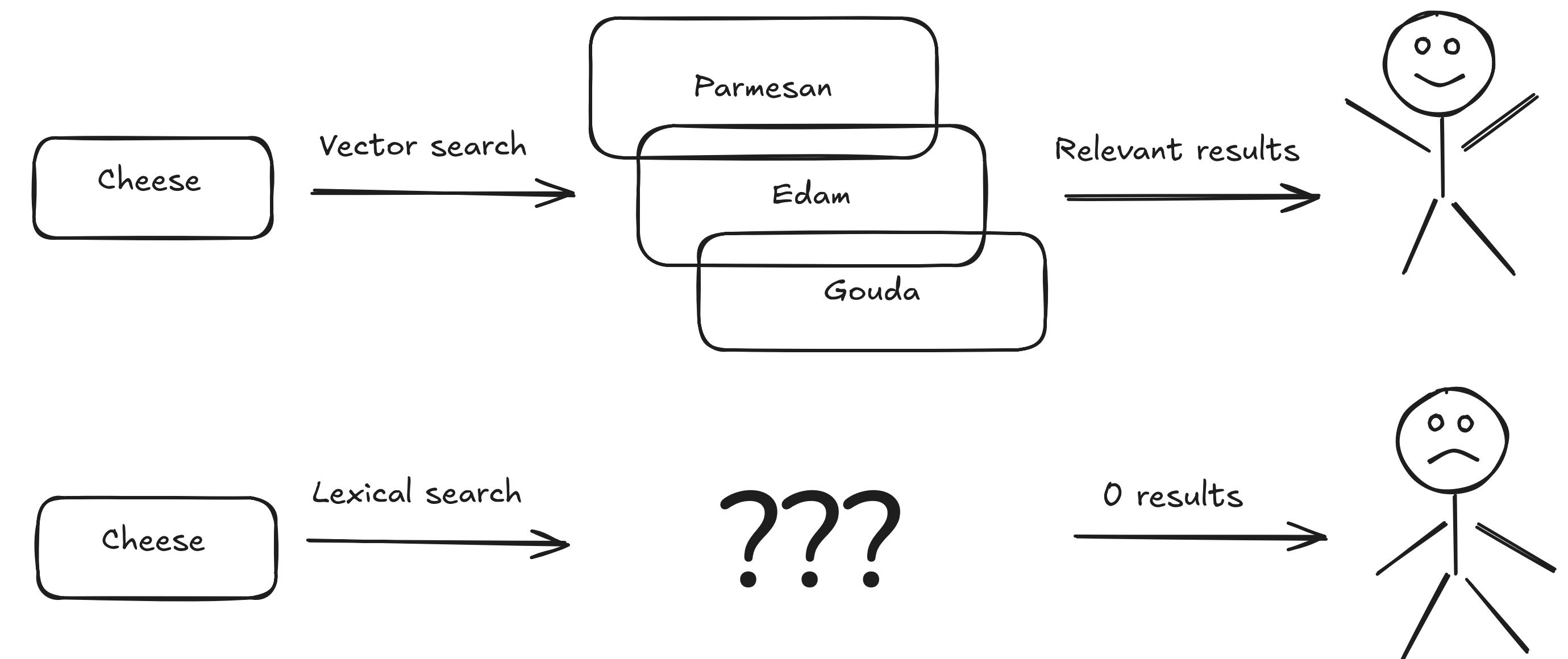
What this course is not about

- Training your own models
- Deep dive into LLMs
- Building e2e platforms
- Covering unrelated topics (e.g. image generation)
- Memorizing math formulas of proofs

What is vector search?

Vector search - is a type of search which captures *semantic meanings* of objects

- Does not require an exact match
- Multimodal (text, audio, video, etc.)
- Unlocks or simplifies advanced search scenarios (e.g. exploratory search)



Examples

- Video recommendation
- Image search
- Music recommendation
- RAG
- ...
- Your ideas?

DAILYMOTION

The fastest motorcycles in the...
Videoo.tv (English) 6 years ago

The best electric motorcycles of 2019
Videoo.tv (English) 6 years ago

World's Safest Motorcycle - Powerf...
Seeker Land 2 weeks ago

Electric Cruiser Motorcycles For 2020
Motorcycle Cruiser 5 years ago

2021 Sondors Metacycle Electric...
Motorcyclist 5 years ago

Alta Motors Electric Redshift MX...
Motorcyclist 9 years ago

Zero SR/F Electric Motorcycle First Look
Motorcyclist 7 years ago

2025 CFMoto Ibex 450 Dyno Test
Cycle World 5 months ago

What's It Like to Live with an Electric...
Cycle World 1 year ago

Gameplay directo Asus ROG Xbox Ally X
xataka 3 days ago

Like Bookmark Share More

WMC250EV – The Worlds Fastest Electric Motorcycle

xataka Follow

4 years ago

Category Motor

Examples

“When you’re dealing with over a billion plus user-generated, multi-modal pieces of content from hundreds of millions of monthly active users across 21 countries, 11M businesses and all the complex user interactions that come with it, you need a way to bring it all together.

Now, we can represent everything from hotel preferences to restaurant choices to user behavior in a unified way.”

Rahul Todkar - VP, head of data and AI @ Tripadvisor

<https://qdrant.tech/blog/case-study-tripadvisor/>

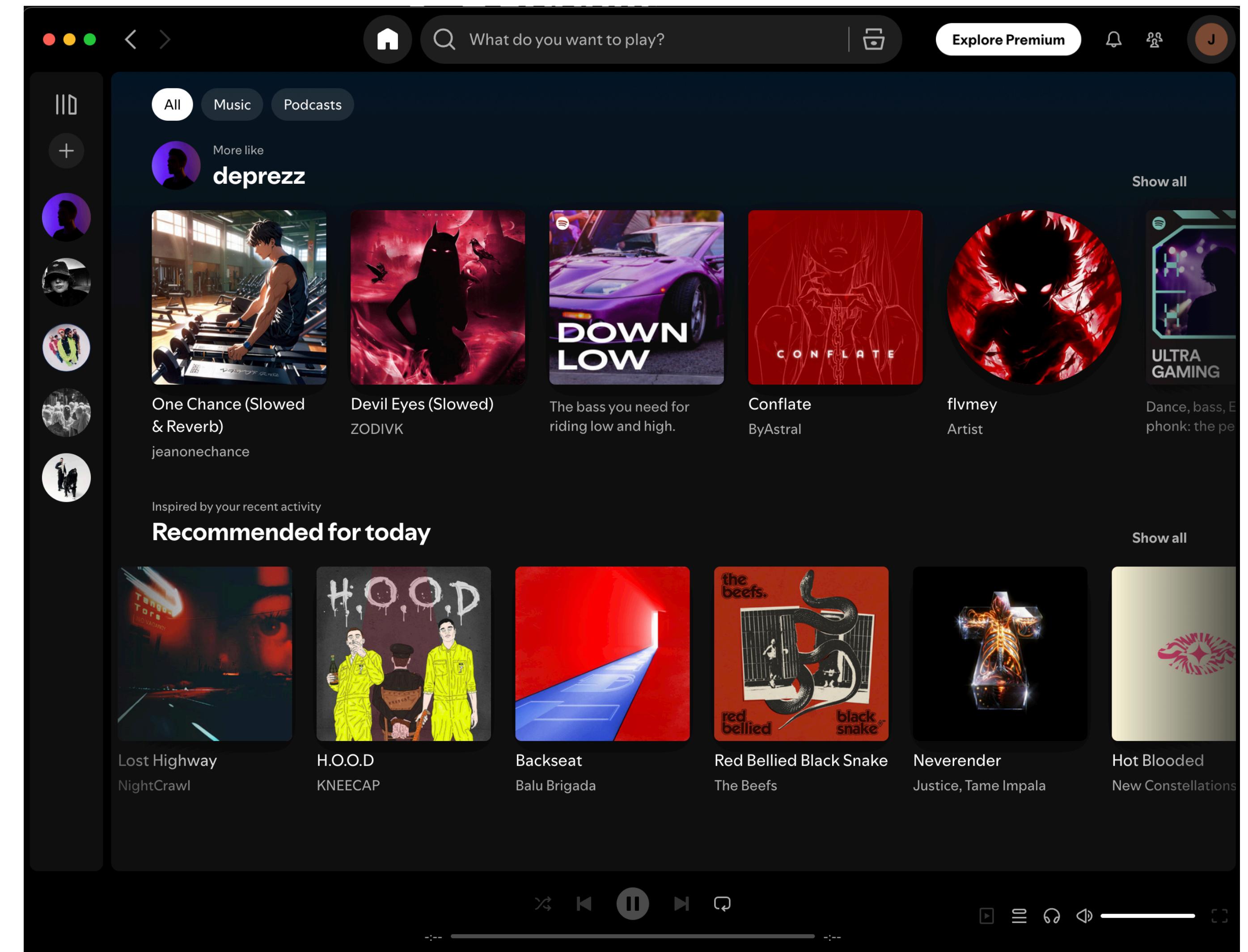
The screenshot shows the Tripadvisor homepage for Phuket. At the top, there's a search bar with the placeholder "Search" and a "Tripadvisor" logo. Below the search bar are navigation links for "Rewards", "Discover", "Trips", "Review", and a currency selector set to "USD". A user profile icon is also visible. The main content area features a large button with the text "Need guidance? Ask Ollie." and a sub-section titled "Ask Ollie about Phuket". Below this are several green rounded rectangles containing AI-generated travel questions and answers, such as "Build a trip to Phuket with AI", "What are the top attractions in Phuket?", and "Where can I try authentic Thai cuisine?". Further down, there's a section titled "Essential Phuket" with a "Things to do" heading. It includes three cards: "Big Buddha Phuket" (a large white seated Buddha statue), "Chaitharam Temple (Wat Chalong)" (a traditional golden-roofed temple entrance), and "Patong Boxing Stadium" (two people in a boxing ring). Each card has a small "2025" badge in the bottom left corner.

Examples

“For the past decade, Spotify has used approximate nearest-neighbor search technology to power our personalization, recommendation, and search systems.

These technologies allow engineers and researchers to build systems that recommend similar items (like similar tracks, artists, or albums) without needing to run slow and expensive machine learning algorithms in real time.”

Peter Sobot - Staff ML Engineer @ Spotify



Examples

I have no proof that it uses vector search, but it definitely does something that could be done with vector search.

Reversely.ai

Home All Tools Blog Faq's Pricing Login

Filter

Similar People Duplicates

New Hero MotoCorp Karizma XMR la... Click to Open

All New 2024 Hero Karizma XMR 210... Click to Open

Get Ready to Ride in Style with the 'Xt... Click to Open

Team-BHP - Hero MotoCorp Karizma ... Click to Open

Hero Karizma XMR 210 Price In Bangl... Click to Open

Karizma new model bike new arrivals Click to Open

Edit

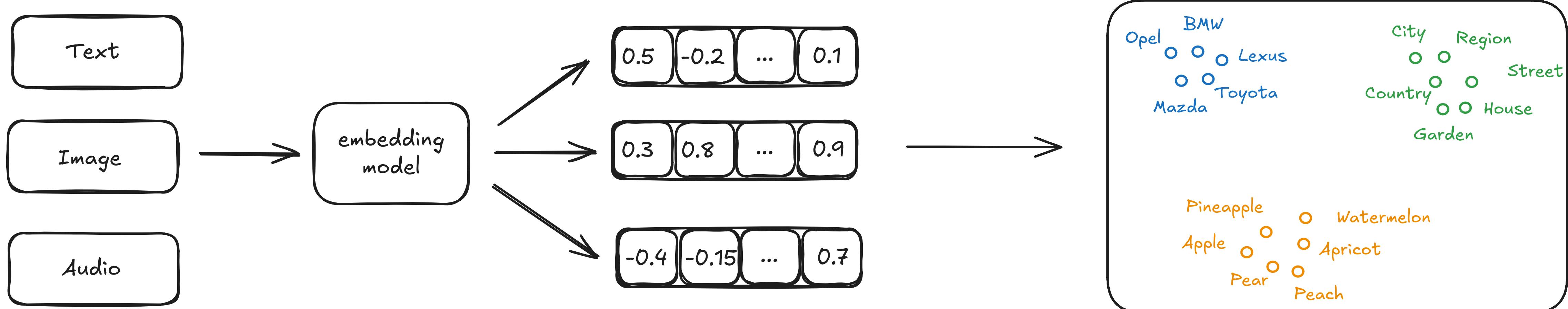
Drop, paste your image anywhere or Click here to upload an image.

Only JPG, JPEG, PNG, WEBP, and HEIC formats allowed.

The screenshot shows the Reversely.ai website interface. At the top, there is a navigation bar with links for Home, All Tools, Blog, Faq's, Pricing, and Login. Below the navigation bar, there is a search bar with a filter icon and a crown icon. The main content area displays a grid of images related to a yellow motorcycle. The first image in the grid is a larger thumbnail of the same yellow motorcycle. To its right are several smaller cards, each featuring a different image of the same motorcycle from a different angle. Each card includes a title, a 'Click to Open' button, and a small crown icon. At the bottom left of the grid, there is a purple button labeled 'Edit' and a dashed box containing instructions to drop or upload an image, along with a note about file formats. A small icon of a camera with a plus sign is also present. The overall layout is clean and modern, typical of a search engine interface.

What are embeddings?

Embeddings are numerical representations of the semantic of the input data.

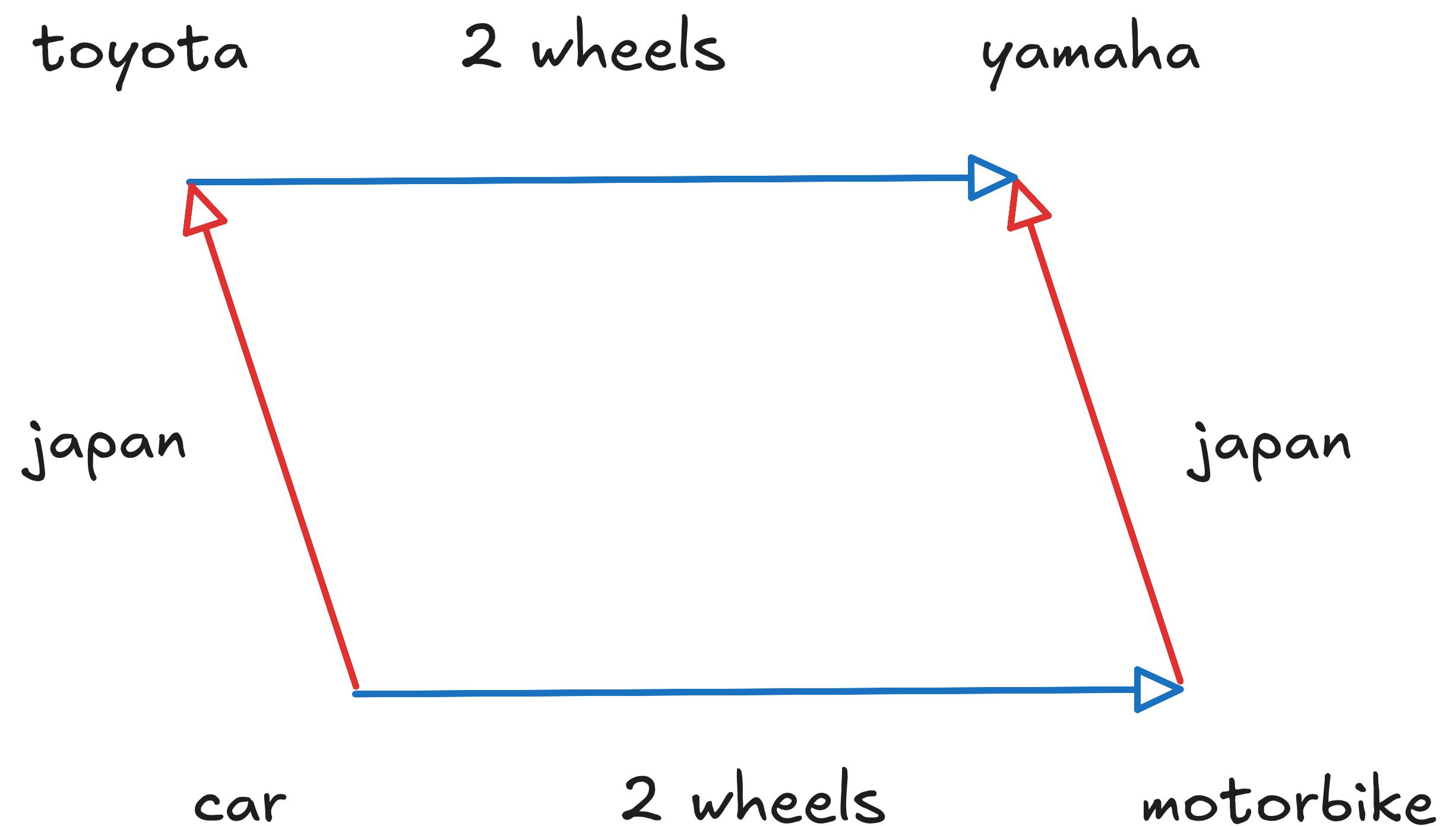


How to get embeddings?

Word2vec

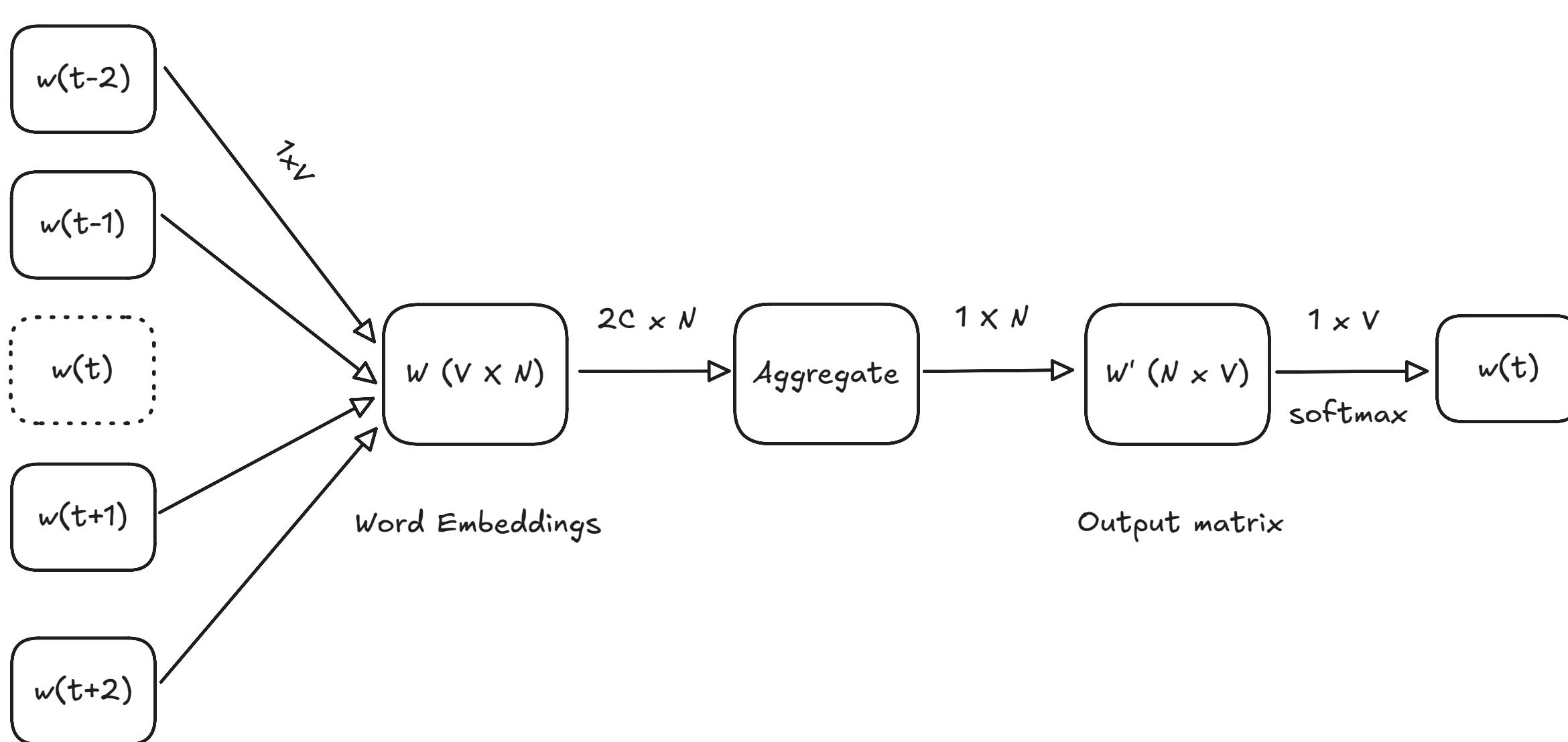
Word2Vec is a method to represent words as vectors in a continuous vector space so that similar words have similar vectors.

- Captures semantic meaning
- Fast inference -> just a dictionary lookup $O(1)$ time.
- Context-independent
- No builtin out-of-vocabulary and typos handling
- Limited to textual data
- Requires preprocessing

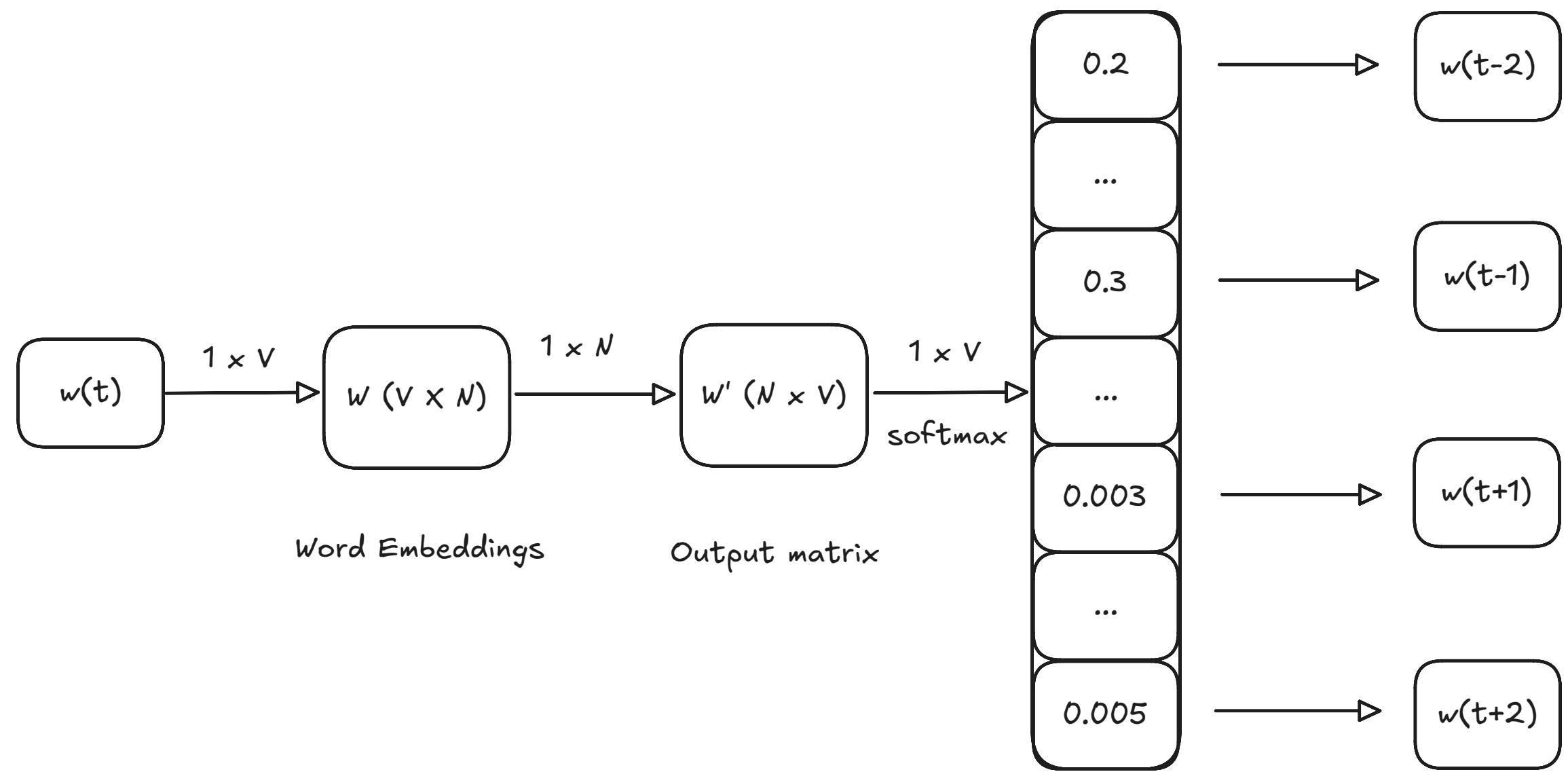


Word2vec

CBOW



Skip-gram



fastText

- Operates over n-grams
- Understands words morphology
- Better at handling out-of-vocabulary and typos
- Limited to textual data
- Does not require stemming or lemmatisation

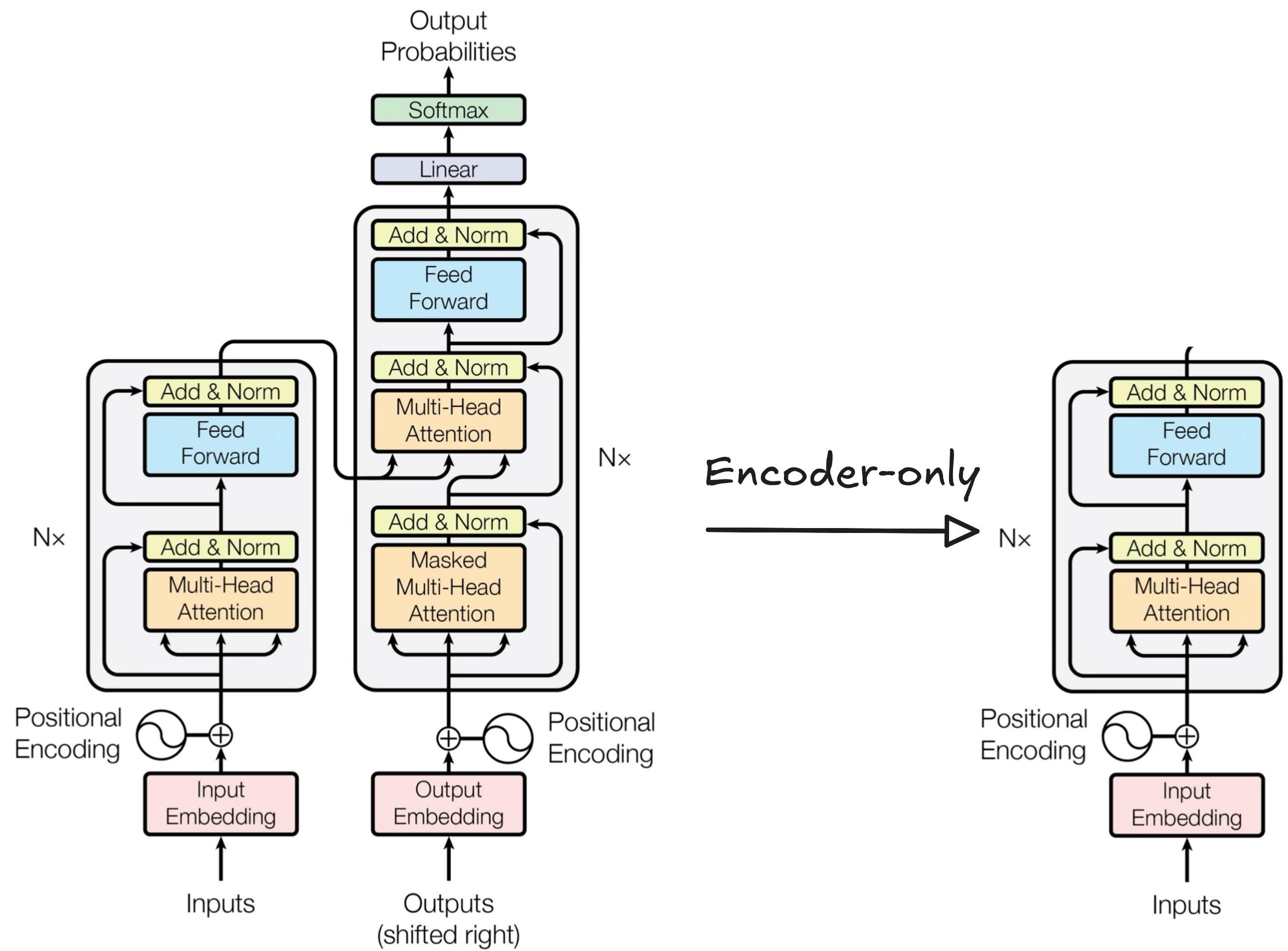
I am using **fastText**

3 grams <fa fas ast stT tTe Tex ext xt>

4 grams <fas fast astT stTe Text ext>

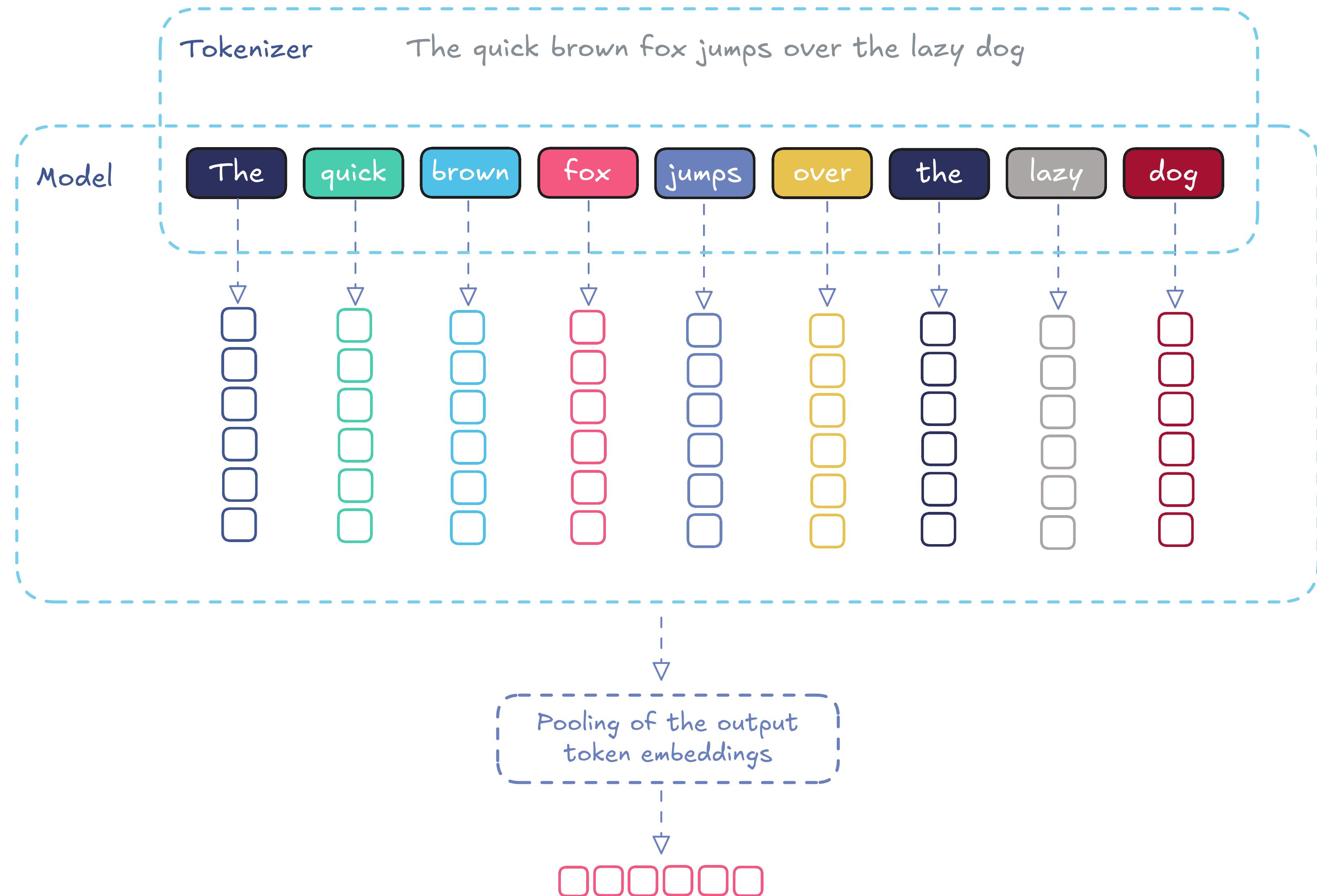
Attention, transformers, etc.

- Context aware
- Better at sentence-level semantics
- Easier extended to multimodal capabilities (CLIP)
- More semantically aligned -> better recall



Transformer embeddings

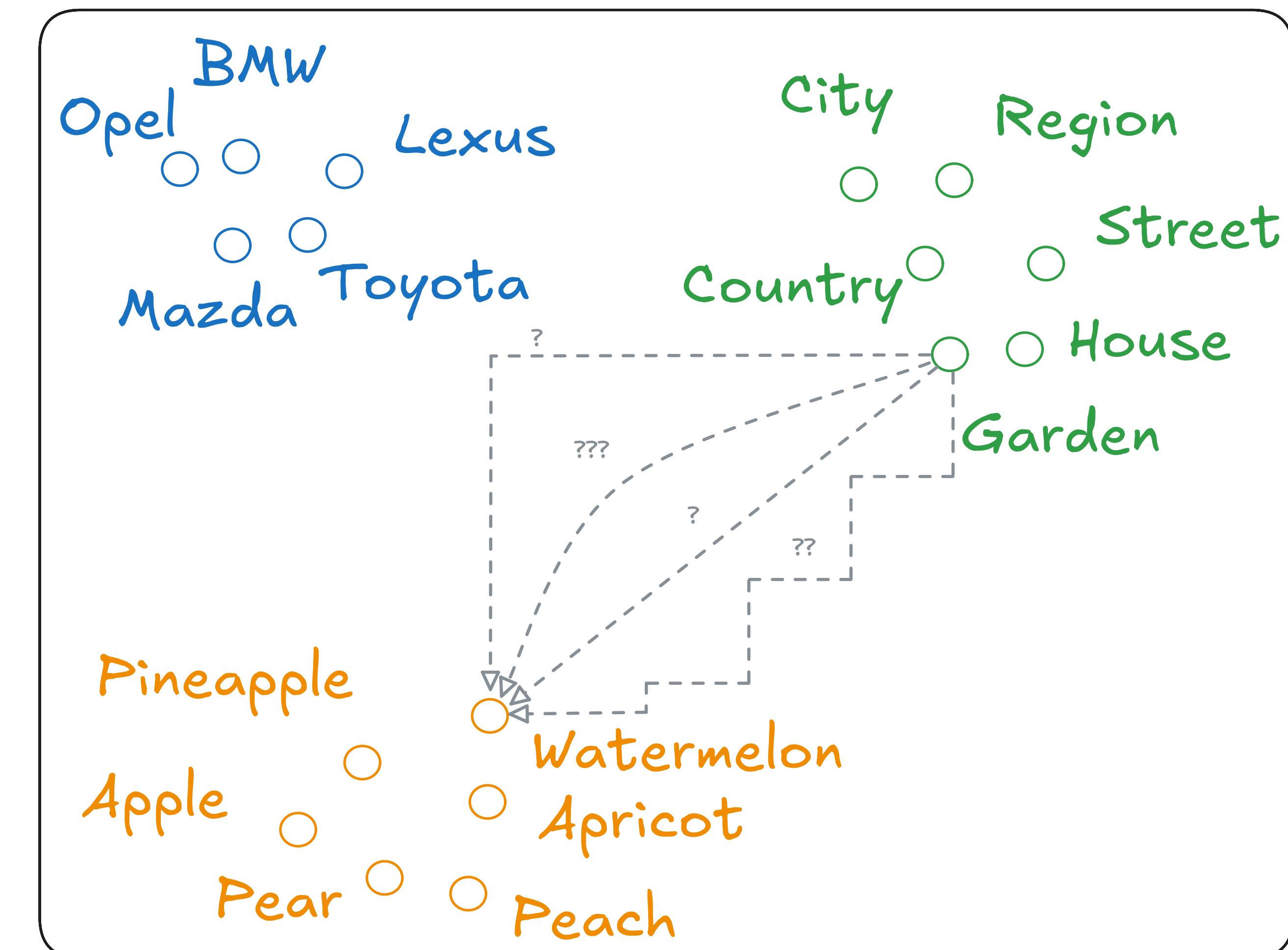
- Split input into tokens
- Compute embedding per token
- Pool or take [CLS] embedding



How do we compare?

Distance metrics

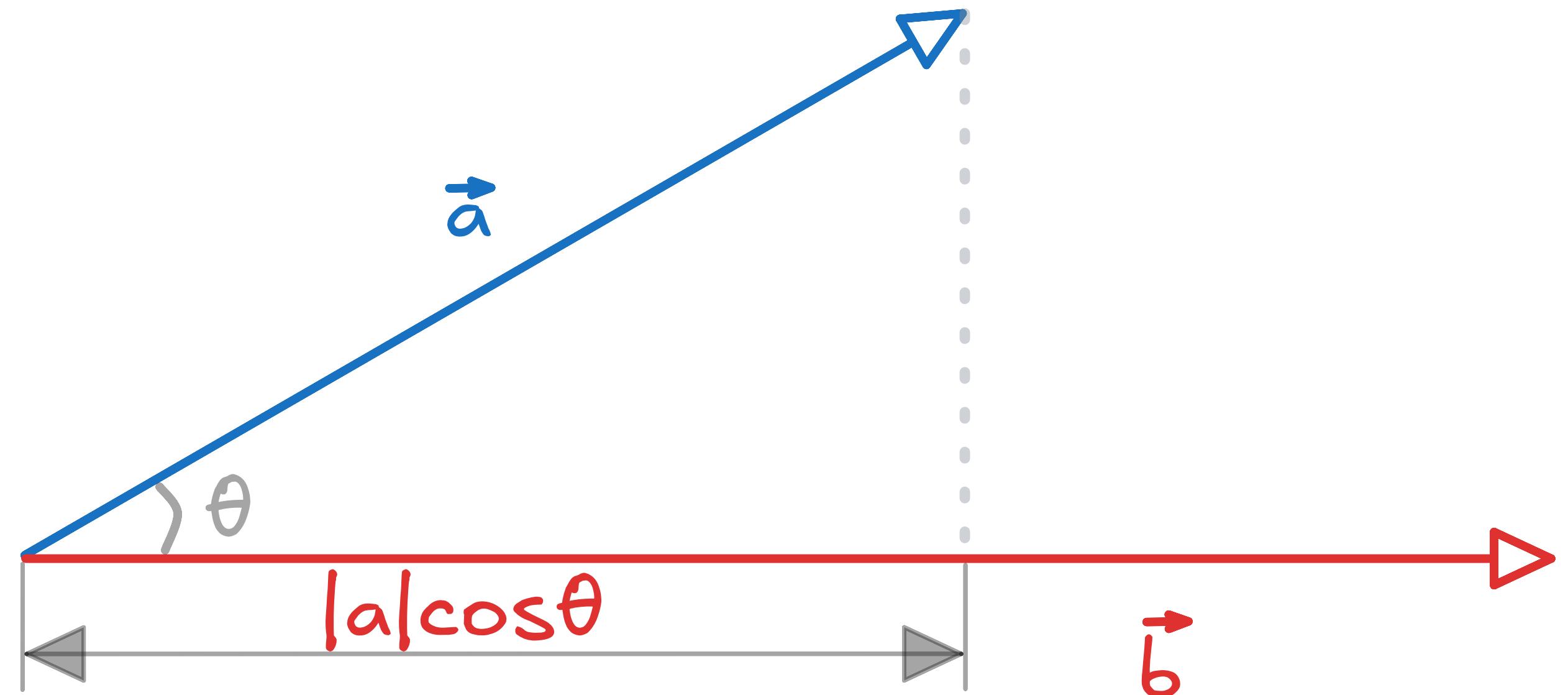
- Dot product
- Cosine
- Euclidean
- Manhattan
- Custom



Dot product

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^n a_i b_i = |\mathbf{a}| |\mathbf{b}| \cos\theta$$

- Depends on both magnitude and direction
- $\mathbf{a} \cdot \mathbf{b} = 0 \rightarrow$ Vectors are orthogonal (no similarity)
- $\mathbf{a} \cdot \mathbf{b} > 0 \rightarrow$ vectors point in the same direction
- $\mathbf{a} \cdot \mathbf{b} < 0 \rightarrow$ vectors point in the opposite directions
- Scaling a vector increases the dot product proportionally to its magnitude
- If embeddings have different norms, a vector with larger magnitude can appear more “similar” just because it’s longer even if directions differ



Cosine

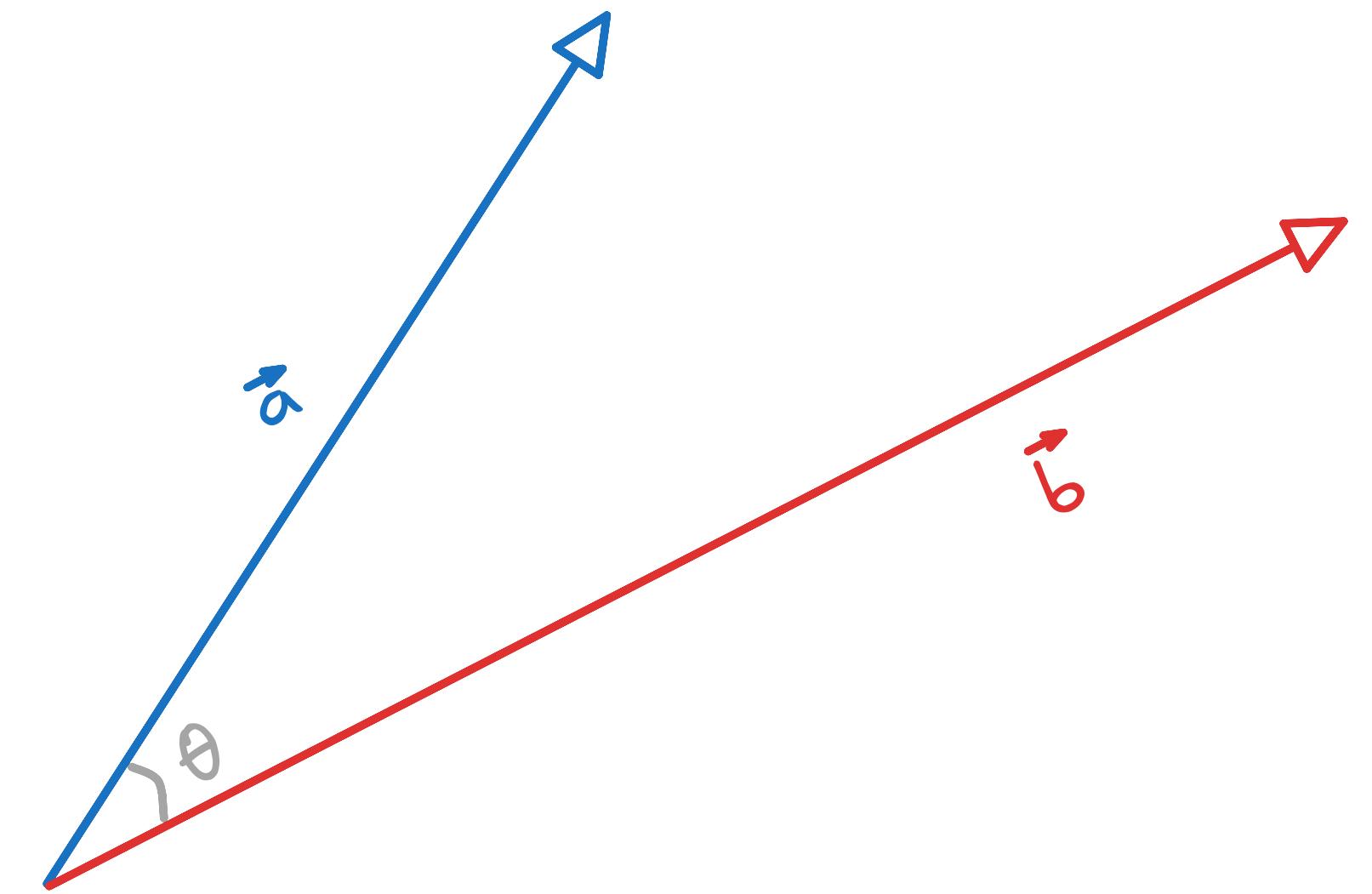
- Depends only on direction
- Normalising vectors makes cosine similarity equivalent to the dot product
- Robust to differences in vector norms, so longer embeddings don't bias similarity
- Usually scaled to [0; 1]
- The most popular choice

$$sim(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|}$$

$$\cos\theta \in [-1; 1]$$

$$scaled_cos\theta = \frac{\cos\theta + 1}{2}$$

$$scaled_cos\theta \in [0; 1]$$



Euclidean and Manhattan

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_i (a_i - b_i)^2}$$

- Measures the “straight-line” distance between vectors
- Sensitive to both magnitude and direction
- $d(\mathbf{a}, \mathbf{b}) = 0 \rightarrow$ vectors are identical

$$d(\mathbf{a}, \mathbf{b}) = \sum_i |a_i - b_i|$$

- Sensitive to both magnitude and direction
- $d(\mathbf{a}, \mathbf{b}) = 0 \rightarrow$ vectors are identical
- More robust to outliers in some cases

How to choose the correct metric?

- Benchmark on your data
- Choose whatever written in a model card or the distance it was trained with

Training procedure

🔗 Pre-training

We use the pretrained [nreimers/MiniLM-L6-H384-uncased](#) model. Please refer to the model card for more detailed information about the pre-training procedure.

Fine-tuning

We fine-tune the model using a contrastive objective. Formally, we compute the cosine similarity from each possible sentence pairs from the batch. We then apply the cross entropy loss by comparing with true pairs.

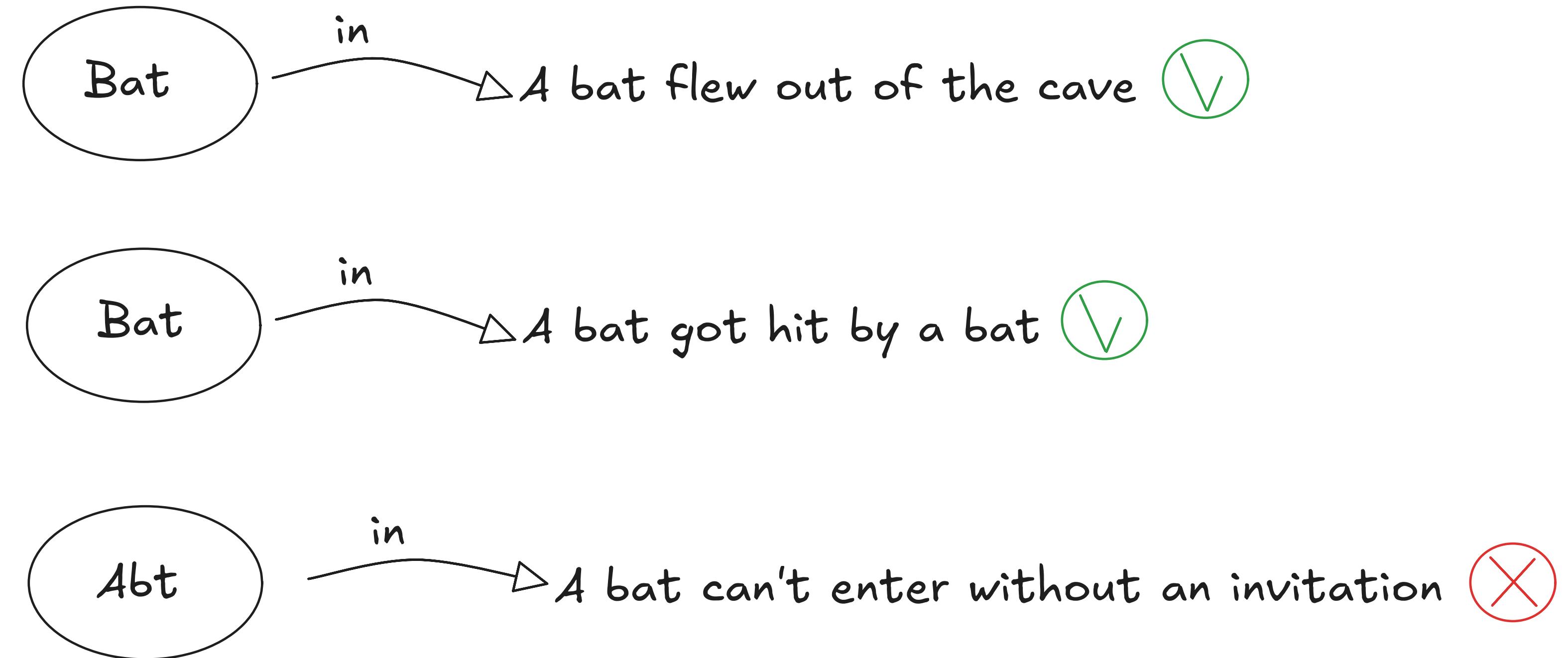
Hyper parameters

We trained our model on a TPU v3-8. We train the model during 100k steps using a batch size of 1024 (128 per TPU core). We use a learning rate warm up of 500. The sequence length was limited to 128 tokens. We used the AdamW optimizer with a 2e-5 learning rate. The full training script is accessible in this current repository: [train_script.py](#).

Lexical search

Exact keyword match

- Retrieves documents containing exact query terms
- Fast for small datasets
- Easy to implement and interpret
- Works well for rare or unique keywords
- No term weighting
- Cannot handle partial matches or typos
- No semantic understanding



Bm25

$$BM25(q, d) = \sum_{t \in q} \log\left[\frac{N}{df(t)}\right] \cdot \frac{(k_1 + 1) \cdot tf(t, d)}{k_1 \cdot [(1 - b) + b \cdot \frac{dl(d)}{dl_{avg}}] + tf(t, d)}$$

- Retrieves documents containing exact query terms
- Cheap and fast
- Strong baseline
- Can be extended with synonyms
- Takes into account frequency of term in document and in the dataset
- Cannot handle typos out-of-the-box
- No semantic understanding

k_1 - term frequency saturation; Typical range: [1.2; 2.0]

b - length normalisation. $b \in [0; 1]$; Typical value 0.75

$dl(d)$ - document length

dl_{avg} - average document length in the corpus

N - number of documents in the corpus

$df(t)$ - the number of documents containing the term

$tf(t, d)$ - term frequency

Vector search vs lexical search

Algorithms result comparison

Neither of the algorithms works perfect.

Sometimes Bm25 is better, sometimes - semantic.

Examples based on a e-commerce dataset WANDS.

Query	Bm25	Semantic
Cybersport desk	Desk	Gaming desk
Plates for ice cream	"Eat" plates on wood wall decor	Alicyn 8.5 " melamine dessert plate
Kitchen table with a thick board	Craft kitchen acacia work cutting board	Industrial solid wood dining table
Wooden bedside table	30" bedside table lamp	Portable bedside end table
Computer chair	Vibrant computer task chair	Office chair
64.2 inch console table	Cervantez 64.2 " console table	69.5 " console table

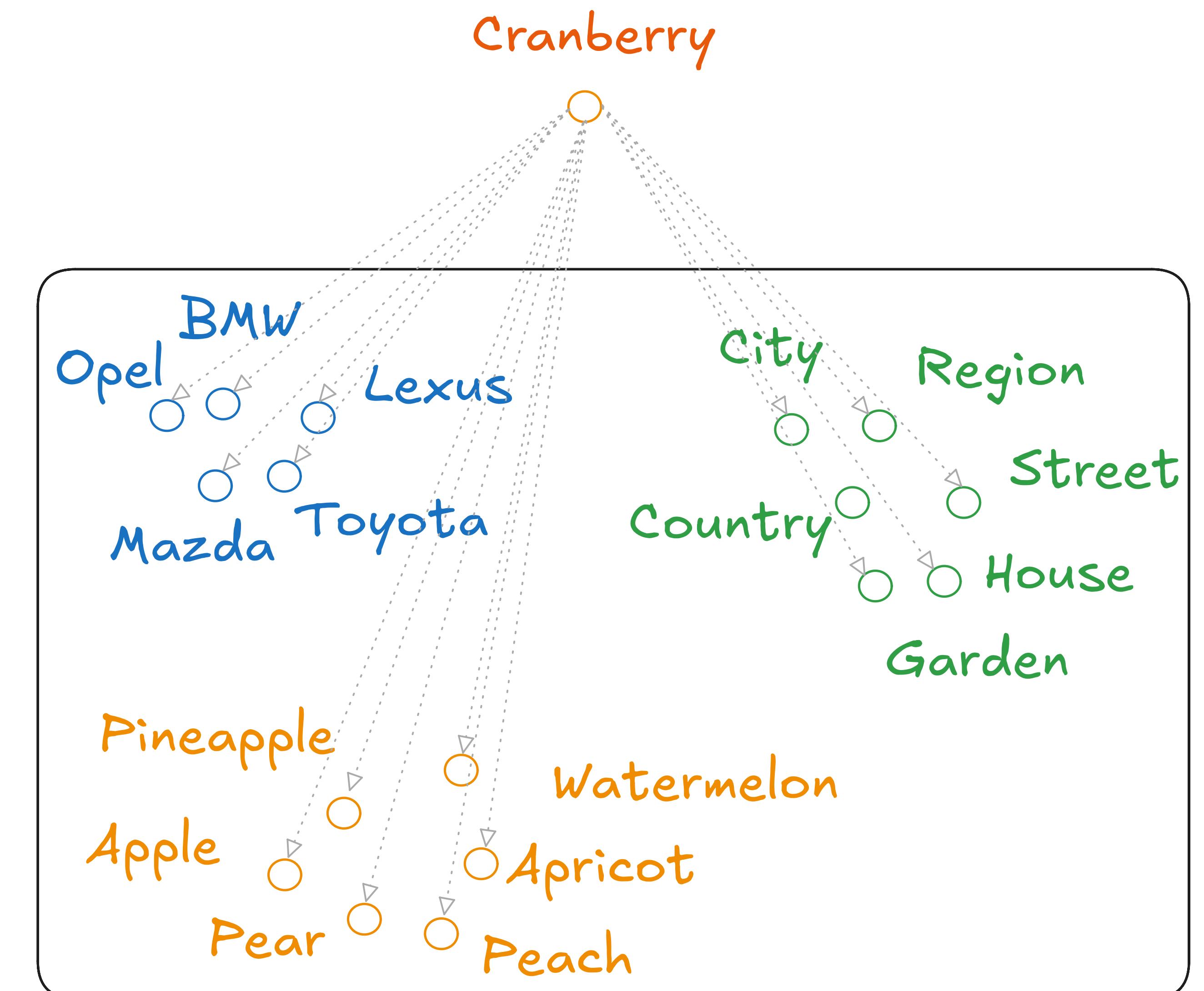
Vector search vs lexical search

Pros and cons

	Lexical search	Vector search
Matching type	Matches exact keywords in the text	Matches meaning or intent
Handling synonyms	Requires expansion	Naturally capture synonyms
Handling typos / variations	Fails unless fuzzy search is implemented	Often robust to typos / variations, depends embeddings
Explainability	High, formula is straightforward	Hard to interpret
Number and dates	Supports if formatting is well defined	Might struggle even if with a defined formatting
Speed and implementation	Fast for low-frequency terms, slower for high-frequency	Fast (might be significantly slower if embedding generation is included)
Memory usage	Low-to-moderate	High

Brute force approach

- Compares query vector to all the vectors in a collection
- Computes exact similarity
- Guarantees 100% vector recall
- $O(N \cdot D)$ complexity
- Stores only vectors, no index overhead
- Performs well on small datasets (20-100k vectors), slow and costly for large



Brute force

Code snippet example



```
import numpy as np
import time

# Generate random vectors
N, D = 50_000, 512 # 50k vectors, 512 dimensions
data = np.random.randn(N, D).astype(np.float32)
query = np.random.randn(D).astype(np.float32)

# Normalize for cosine similarity
data /= np.linalg.norm(data, axis=1, keepdims=True)
query /= np.linalg.norm(query)

# Benchmark brute-force search
start = time.perf_counter()
scores = data @ query # dot product = cosine similarity
top_k = np.argpartition(-scores, 5)[:5] # top 5 nearest neighbors
elapsed = time.perf_counter() - start

print(f"Top-5 indices: {top_k}")
print(f"Search time: {elapsed*1000:.2f} ms")
```

Scales linear with vector number and dimensionality:

- $50k, 512d$ – $3.56ms$
- $100k, 256d$ – $2.54ms$
- $100k, 512d$ – $7.20ms$
- $500k, 512d$ – $29.86ms$

ANN



```
import time
import numpy as np
from qdrant_client import QdrantClient, models

COLLECTION_NAME = 'brute-force-comparison'
DIM = 512
NUM_VECTORS = 500_000
cl = QdrantClient()
cl.create_collection(
    COLLECTION_NAME,
    vectors_config=models.VectorParams(size=DIM, distance=models.Distance.COSINE)
)

data = np.random.randn(NUM_VECTORS, DIM).astype(np.float32)
query = np.random.randn(DIM).astype(np.float32)

cl.upload_collection(COLLECTION_NAME, vectors=data)

start = time.perf_counter()
cl.query_points(COLLECTION_NAME, query=query, limit=5)
elapsed = time.perf_counter() - start

print(f"Search time: {elapsed*1000:.2f} ms")
```

Scales sub-linearly (usually logarithmically)

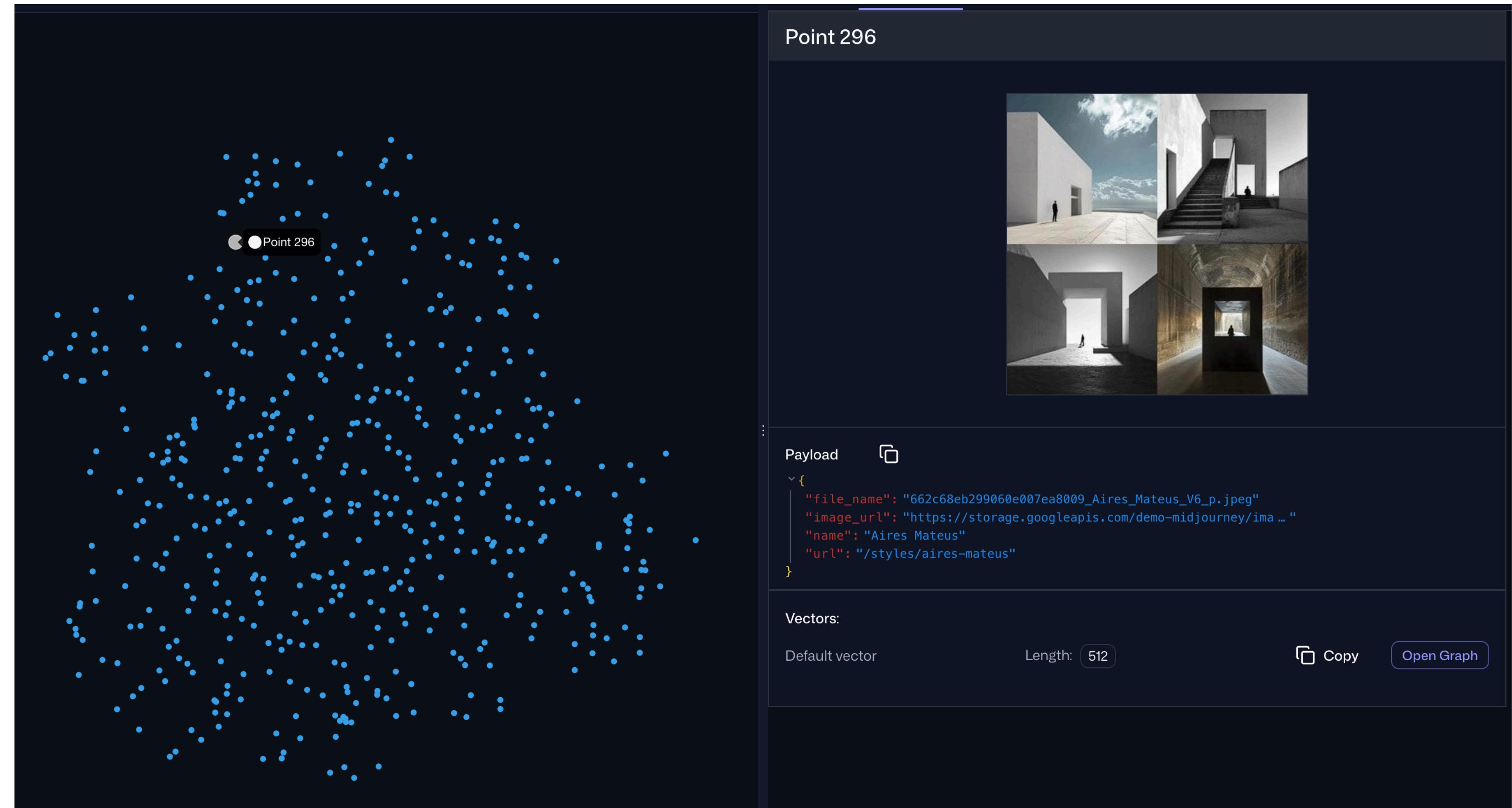
- $100k, 512d$ – $11.26ms$
- $500k, 512d$ – $12.2ms$
- $1kk, 512d$ – $17ms$

Visualisation

It is important to know your data.

Points visualisation is one of the ways to understand the nature of the data.

It can be done with various algorithms depending on your need. Such as PCA, T-SNE, U-Map, etc.



Visualisation

Graph exploration

Graph based visualisation let users explore the data and see nearest neighbours to the chosen points.

The image shows a graph visualization interface. On the left, a large network of nodes is displayed, consisting of numerous small circles connected by thin lines. The nodes are colored in various shades of orange and cyan. On the right, a detailed view of a specific node is shown. The title of this view is "Point 2619". Below the title, there are two photographs of a stone-arched corridor or staircase. At the bottom of the right panel, there is a JSON object labeled "Payload" with the following content:

```
Payload □  
{  
  "file_name": "662a2f8f94cf4bc70a238456_Carlo_Scarpa_V6_p.jpeg",  
  "image_url": "https://storage.googleapis.com/demo-midjourney/ima ... ",  
  "name": "Carlo Scarpa",  
  "url": "/styles/carlo-scarpa"  
}
```

Questions?