Dense embeddings

George Panchuk
HSE AI Fall 2025

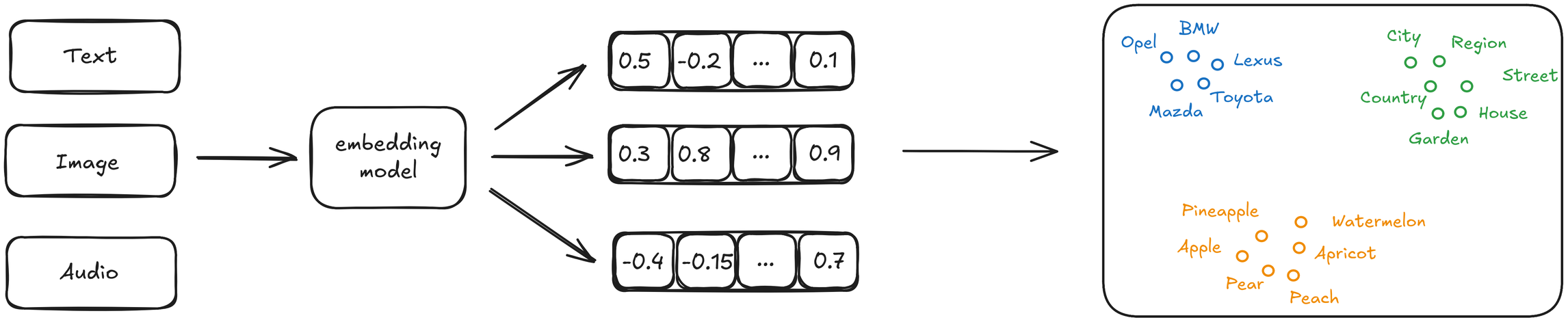
github.com/joein/vector-search-course

Dense embeddings

- Continuous, high-dimensional vectors (e.g. 256-4096 dims)
- Encode semantic meaning, similar vectors are close in space
- Quite heavy

Examples:

- *all-MiniLM-L6-v2 (text)*
- *text-embedding-3-small (text)*
- *ViT (image)*
- *CLIP (text-image)*



How to choose embedding size

- Larger embeddings might capture more semantics of the data
- Smaller embeddings are easier to compare and they weigh less
- Embeddings of size 512-1024 dims usually offer a good cost/quality ratio
- Embedding size alone is not enough to estimate speed, data distribution also matter for vector search algorithms
- Smaller embeddings are typically cheaper

Model	Dims	Pages per dollar	Performance on MTEB Eval	Max Input
Text-embedding-3-small	1536	62,500	62.3%	8192
Text-embedding-3-large	3072	9,615	64.6%	8192
Text-embedding-ada-002	1536	12,500	61.0%	8192

https://platform.openai.com/docs/guides/embeddings

How to choose embedding size

Embedding size memory calculation

$$Memory(bytes) = n \cdot d \cdot dtype\ \ size$$

n - number of embeddings,

d - dimensionality,

$dtype\ \ size$ - data type size,

e.g. float32 - 4b

Common data types and sizes:

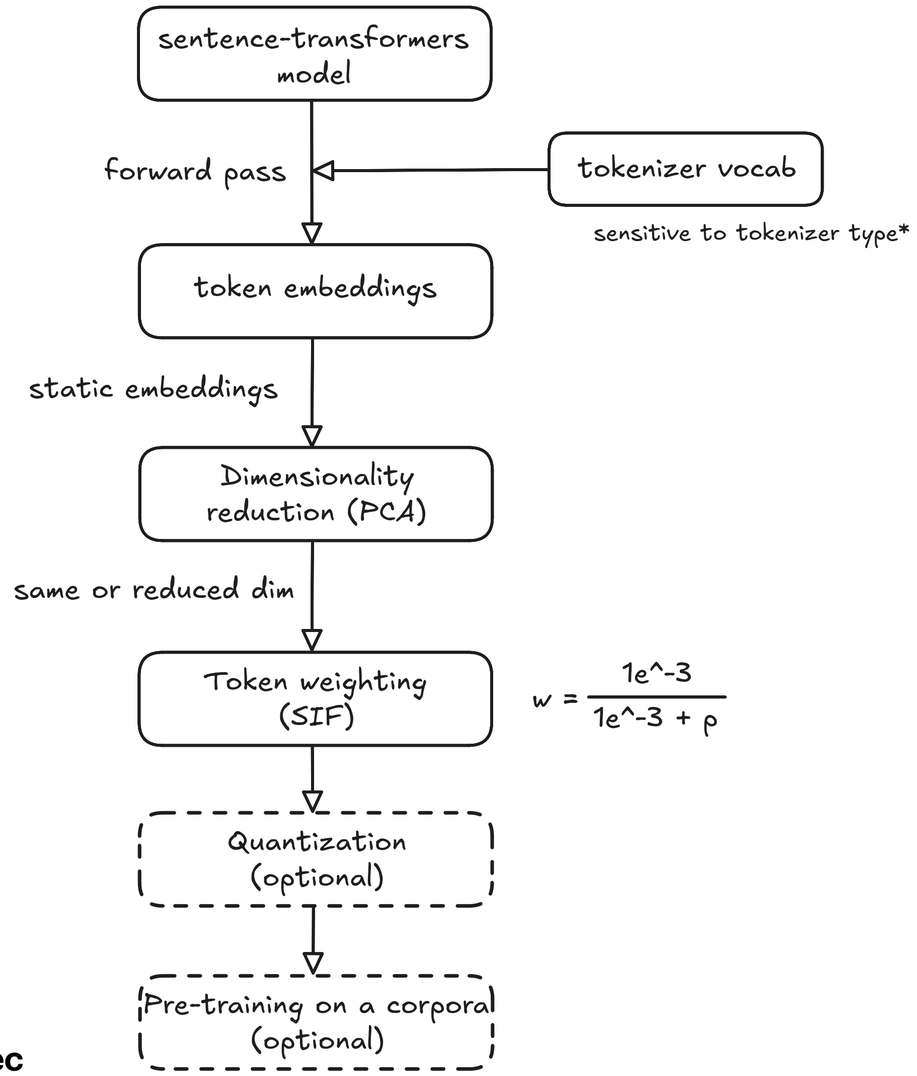
- float32 (f32) - 4 bytes,
- float16 (f16) - 2 bytes
- int8 - 1 byte
- binary - 1 bit

Model	Num embeddings	Data type	Formula	Size
All-minilm-l6-v2 (384)	*1*	*float32*	*1 * 384 * 4*	*1536b*
CLIP (512)	*100,000*	*float32*	*100,000 * 512 * 4*	*~195mb*
Text-embedding-small-3 (1536)	*1,000,000*	*float32*	*1,000,000 * 1536 * 4*	*~5.72gb*
Text-embedding-small-3 (1536)	*1,000,000*	*int8*	*1,000,000 * 1536 * 1*	*~1.43gb*

Modern static vectors

Model2vec

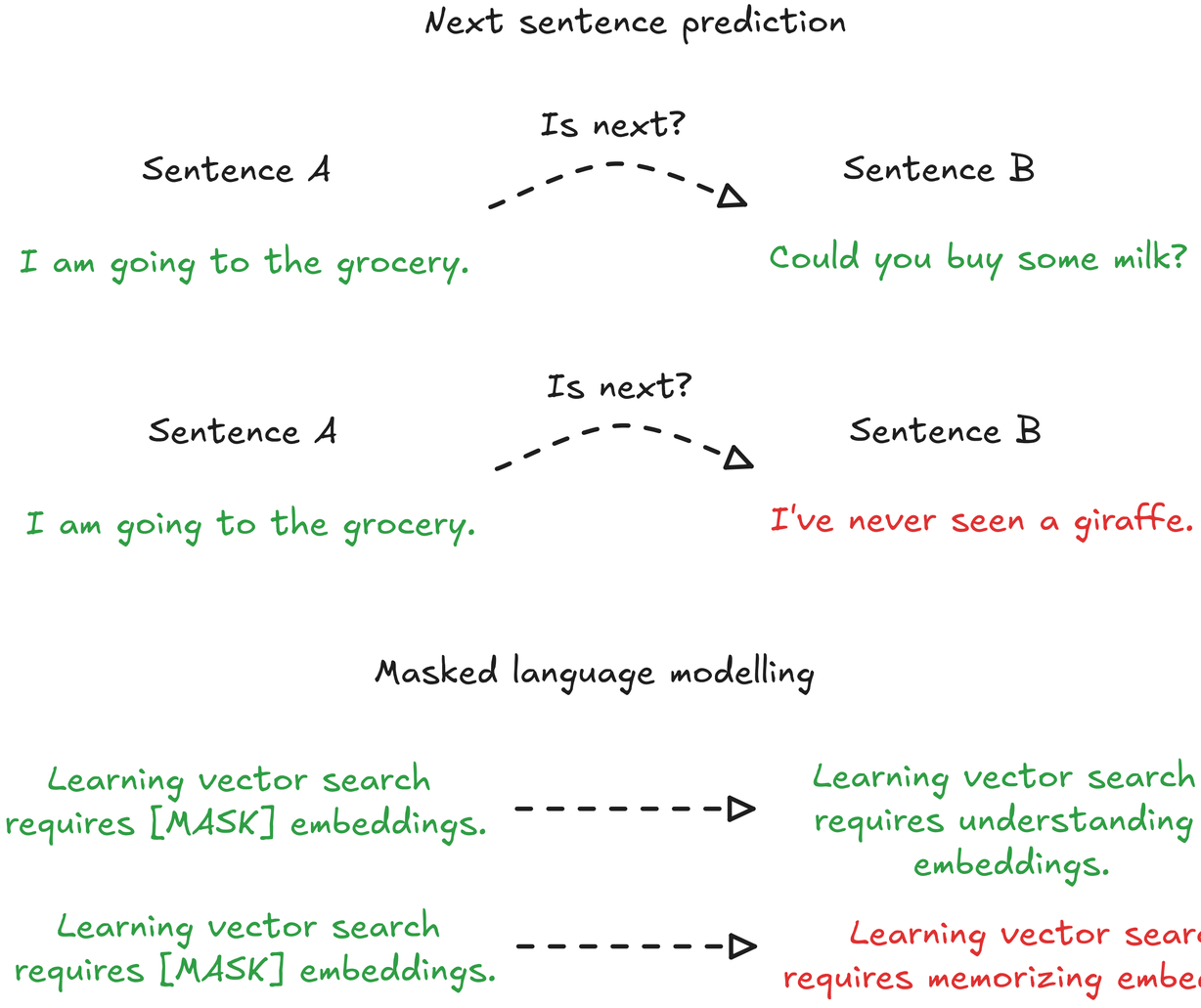
- Uses context aware models as base models
- Distillates to produce static embeddings
- Up to 500 times faster
- Up to 50 times smaller
- Acceptable quality loss
- Expandable vocabulary



MinishLab/model2vec

Why Bert might not be the best choice?

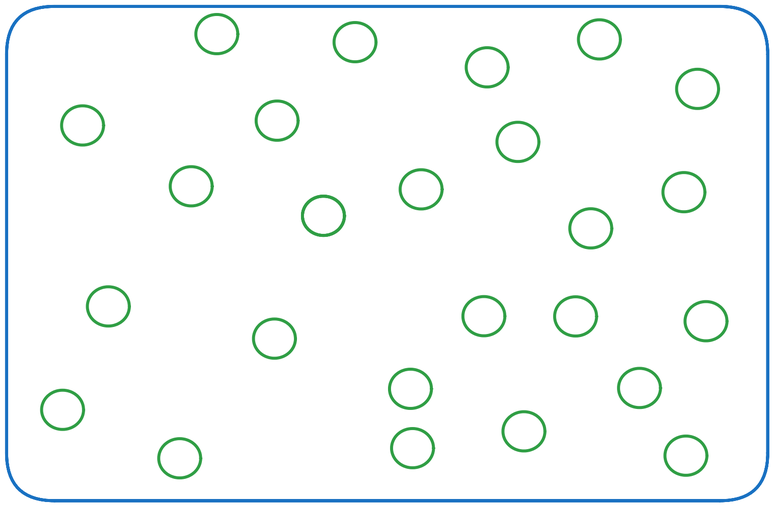
- Neither of the objectives teaches to:
  - Map semantically similar sentences near each other
  - Produce stable sentence embeddings
  - Optimize for vector-space retrieval tasks
- Embeddings are anisotropic (embeddings are concentrated in a narrow cone-like subspace) → brakes cosine, hard to cluster and search
- Relatively heavy (110m parameters)



Isotropic vs Anisotropic

Isotropic

The vectors are uniformly distributed and spread out across the entire vector space

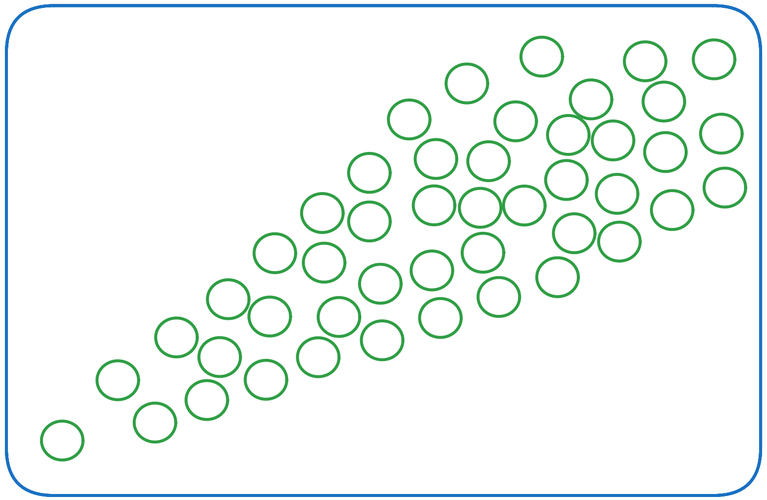


Utilise the full range of the distance metric

Truly similar embedding will have high similarity, while dissimilar ones will have lower similarity

Anisotropic

The vectors are concentrated in a narrow cone-like subspace



The cosine similarity values are all high (e.g. between 0.8 and 1.0)

Hard to distinguish between truly similar and mildly dissimilar sentences

Metric learning

Metric learning - is a way of training a model to create semantically meaningful and well-structured embedding spaces.

Goal:

*To l*earn an embedding function that maps data points...

(too many words you won't read this anyway)

BERT's Issue	Metric learning solution
Objectives don't teach to map semantically similar sentences near each other	Directly optimises for this goal using specifically designed loss functions
Does not produce stable sentence embeddings	Training with specific distance constraints forces the model to create a more consistent and stable embedding for similar inputs
Embeddings are anisotropic	Losses push dissimilar points apart while pulling similar points together

Contrastive loss

$$L_{pair} = y \cdot \frac{1}{2} D(x_1, x_2)^2 + (1 - y) \cdot \frac{1}{2} max(0, m - D(x_1, x_2))^2,$$

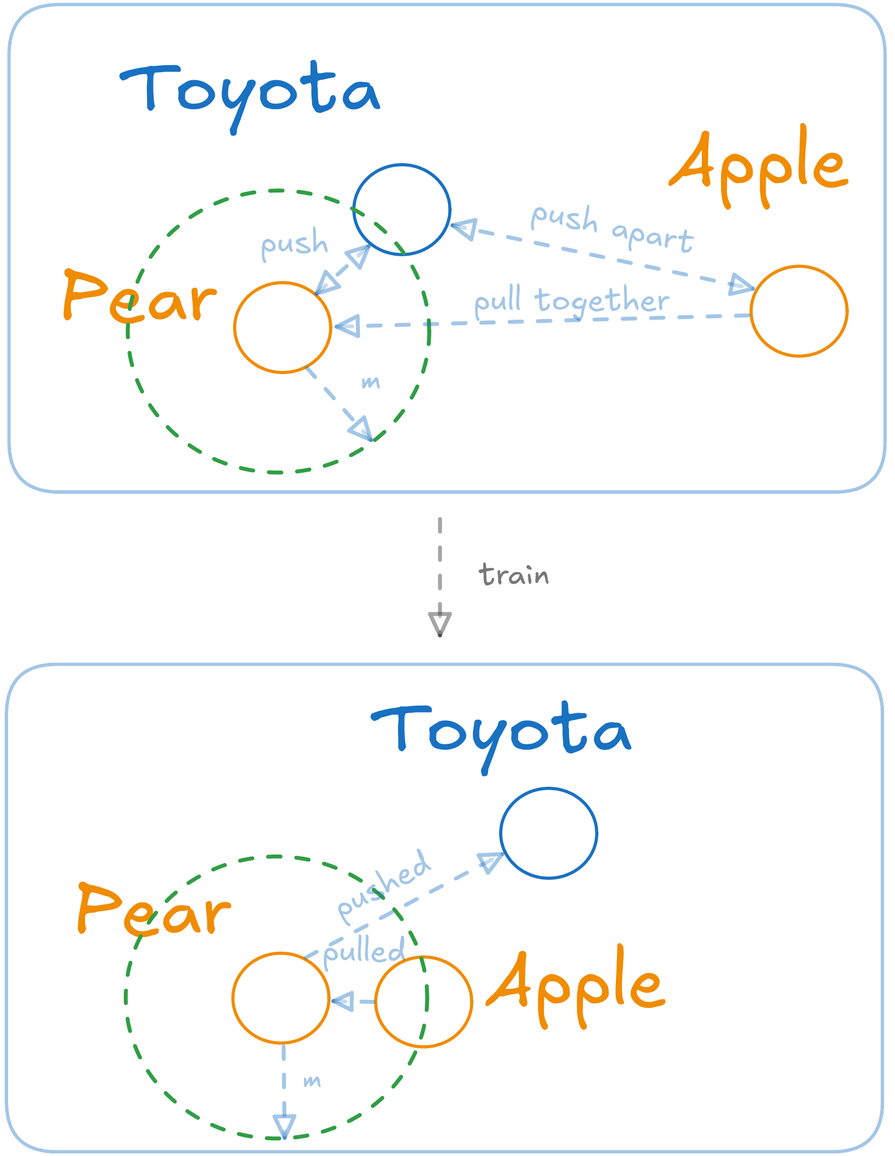
$D(x_1, x_2)$ - distance of choice (cosine or other), m - margin

$$L_{batch} = \frac{1}{N} \sum_{i=1}^{N} L(x_1^{(i)}, x_2^{(i)})$$

Goal: Learn an embedding space where similar items are close and dissimilar items are far

Input: Pairs of samples (x_1, x_2) and a similarity label $y \in \{0,1\}$

Output: Embeddings $f(x_1), f(x_2)$ such that distances reflect similarity



Contrastive loss

Caveats

Requires carefully sampled pairs

If pairs are sampled randomly:

- Most pairs are negative, leading to imbalanced training
- Many negatives are “easy” (already far apart contributing almost no gradient)

Training can be inefficient; hard-negative mining is used to mitigate the problem;

Limited batch efficiency

$O(N^2)$ - possible pairs

Often cases:

- Sample a small subset of pairs (lose information)
- Use a large batch (high memory cost)

Margin sensitivity

Choosing m is tricky:

- Too small → dissimilar points are not separated enough
- Too large →optimisation becomes harder; network might collapse.

Triplet loss

$$L = max(0,\ D(a,p) - D(a,n) + m),$$

a - anchor, p - positive, n - negative,

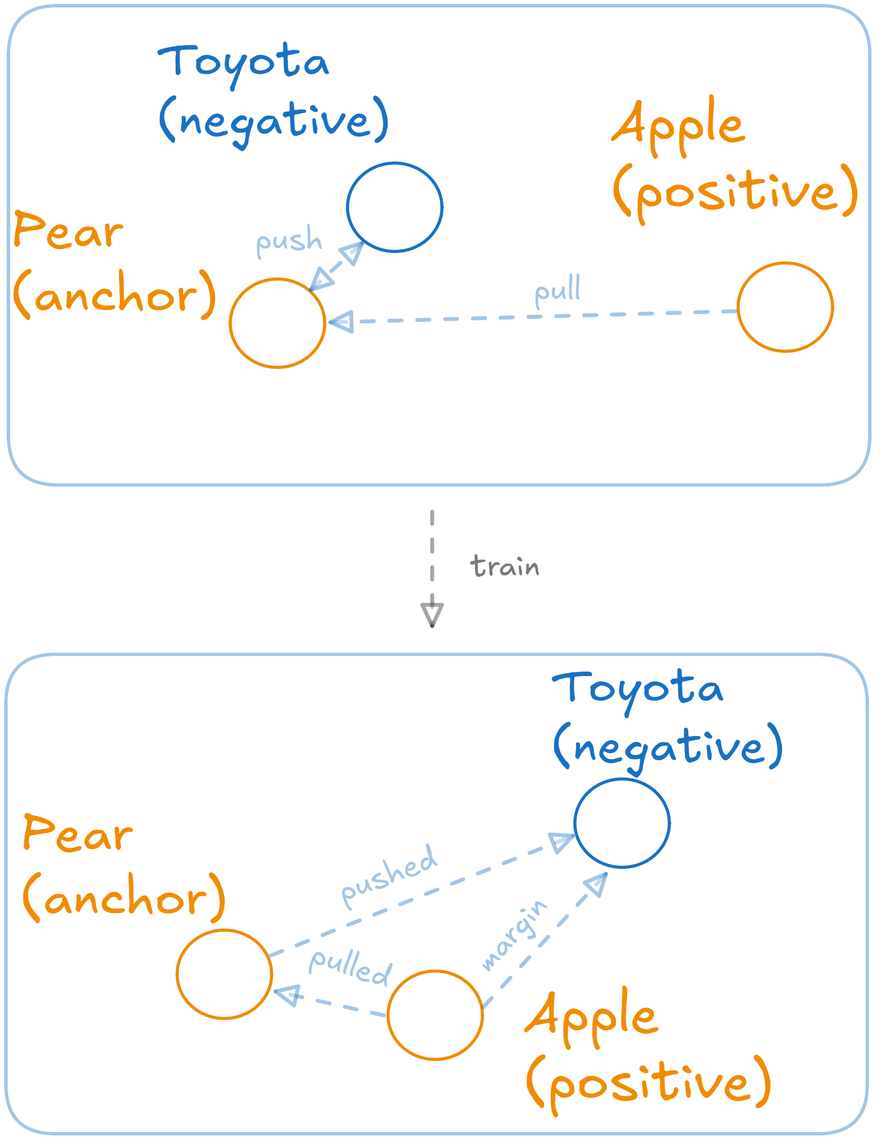
D - distance of choice, m - margin.

Goal: Learn an embedding space where:

- Anchor is closer to positive than to negative
- Encourages relative distances rather than absolute positions

Input: Triplets of samples (a, n, p)

- Needs triplet mining for efficiency
- Training can be slow for large datasets
- Sensitive to margin selection



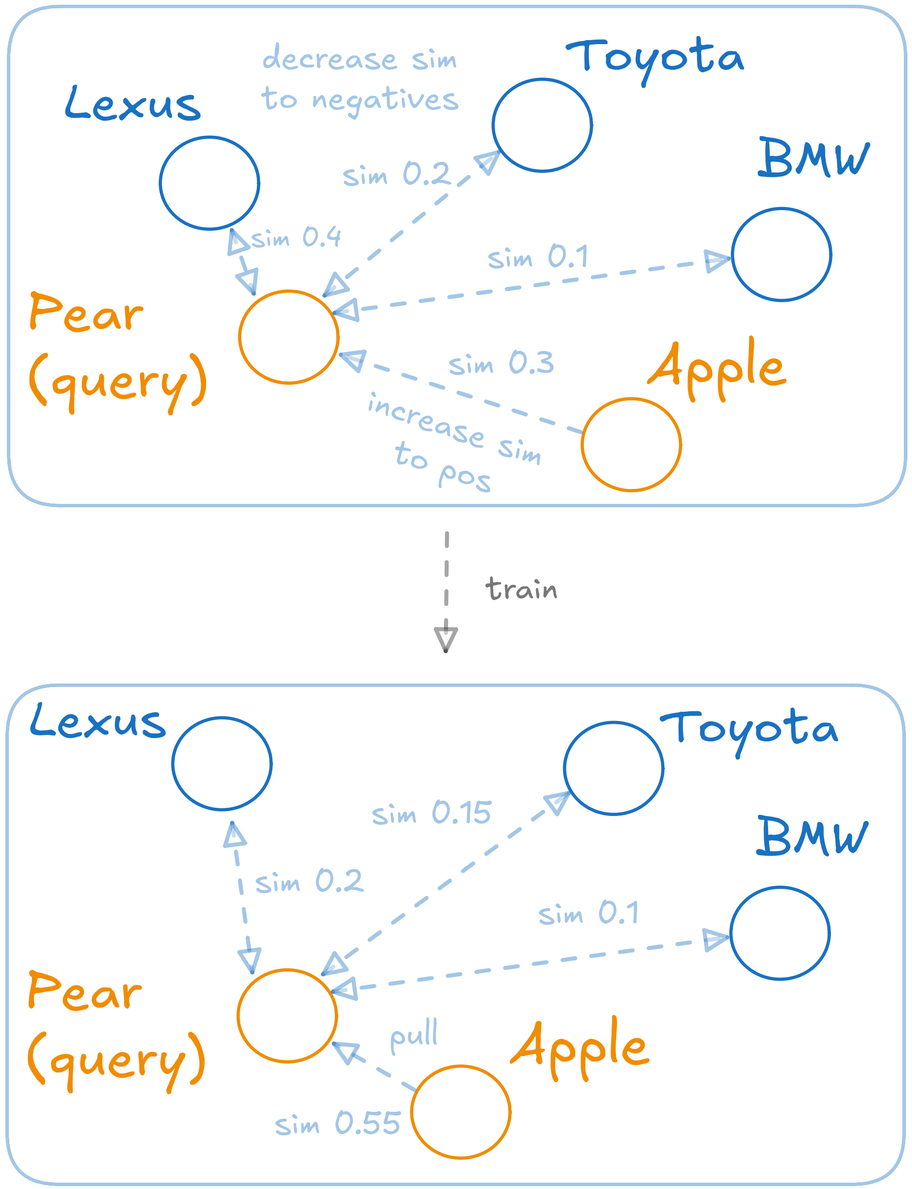
InfoNCE

$$L_i = -\log \frac{e^{D(x_i, x_i^+)/\tau}}{\sum_{j=1}^{n} e^{D(x_i, x_j)/\tau}}$$

Goal: Learn embeddings such that positive pairs are more similar than all other negatives in the batch using a softmax-based contrastive formulation.

Input: query x_i, positive key x_i^+, a set of negatives x_j

- Temperature $\tau > 0$ to scale similarities
- Training can be slow for large datasets



How to choose a model

Metrics

Binary relevance @ Fixed Cutoff (Top-K)

- Precision @ k
- Recall @ k
- Hit rate @ k

Ranking quality metrics

- MRR@k
- DCG@k
- nDCG@k
- mAP@k

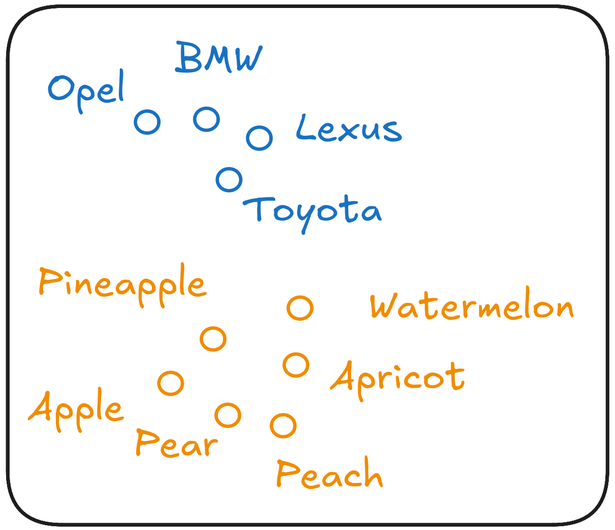
Binary relevance @ Fixed Cutoff metrics

Precision @ k

Precision @ k - measures how many items appeared in topK results are relevant

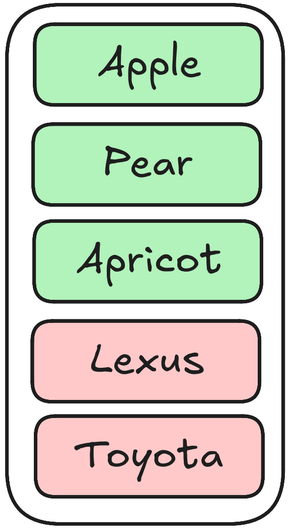
$$Precision@k = \frac{Number\ of\ relevant\ items\ in\ k}{Total\ number\ of\ items\ in\ k}$$

$$Precision@k \in [0; 1]\ *$$

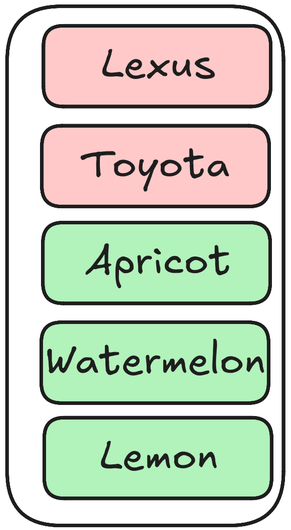


Query: mandarin
Precision@5 = 1.0

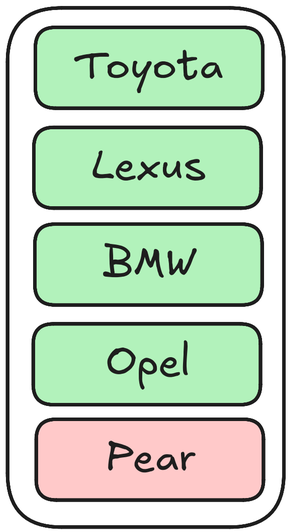
Query: mandarin
Precision@5 = 0.6



Query: mandarin
Precision@5 = 0.6



Query: Mazda
Precision@5 = 0.8



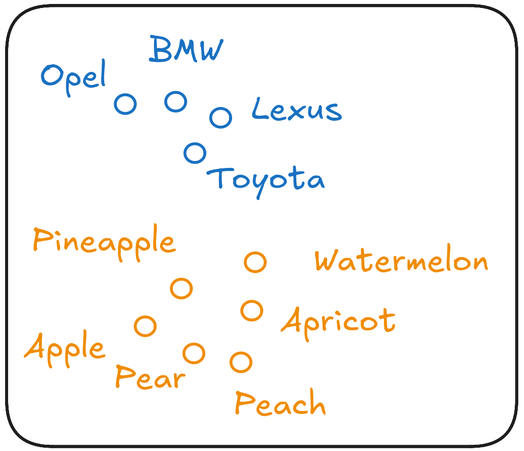
Binary relevance @ Fixed Cutoff metrics

Recall @ k

Recall @ k - measures the share of all relevant items in topK results

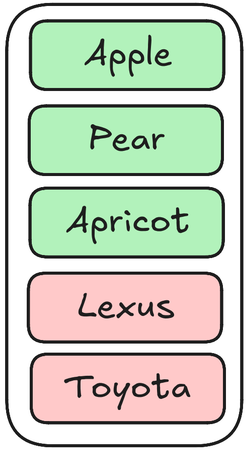
$$Precision@k = \frac{Number\ of\ relevant\ items\ in\ k}{Total\ number\ of\ relevant\ items}$$

$$Recall@k \in [0; 1]\ *$$

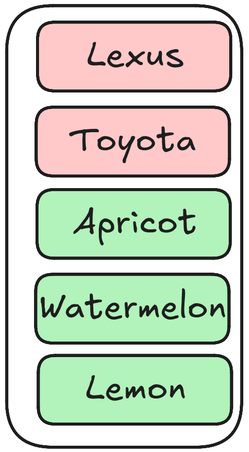


Query: mandarin
Recall@5 = 1.0

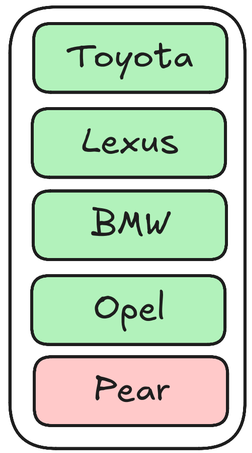
Query: mandarin
Recall @ 5 = 0.6



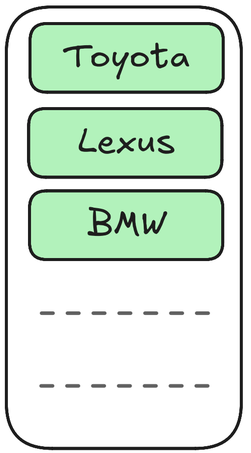
Query: mandarin
Recall @ 5 = 0.6



Query: Mazda
Recall@5 = 1.0



Query: Mazda
Recall@3 = 0.8



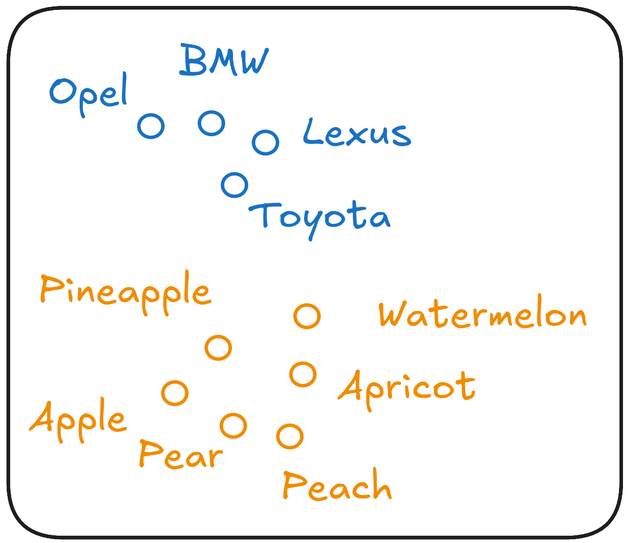
Binary relevance @ Fixed Cutoff metrics

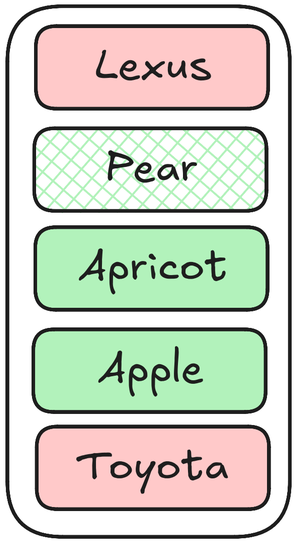
Hit rate @ k

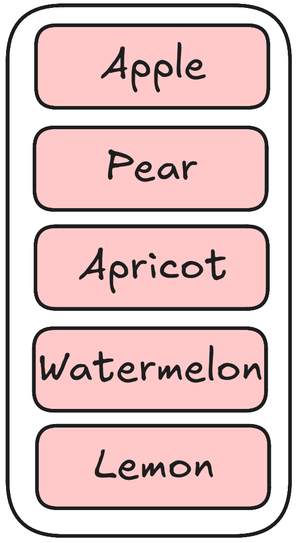
HitRate@k - checks whether there is at least one relevant item in topK results.

$$\text{HitRate@k} = \begin{cases} 1, & \text{if at least one relevant item is in the top } k, \\ 0, & \text{otherwise .} \end{cases}$$

$$HitRate@k \in \{0; 1\}$$







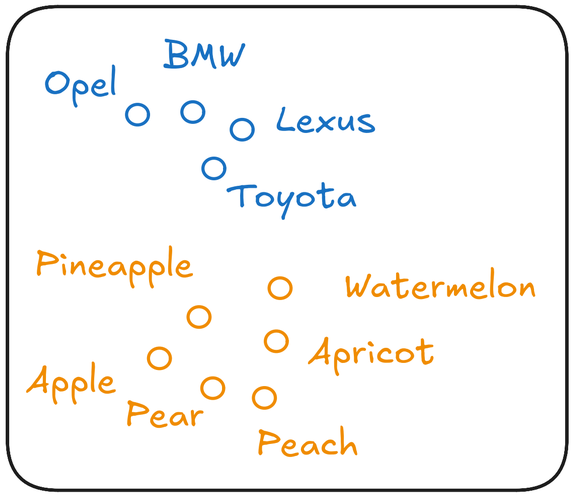
Ranking quality metrics

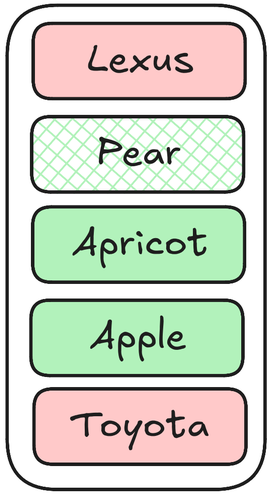
MRR

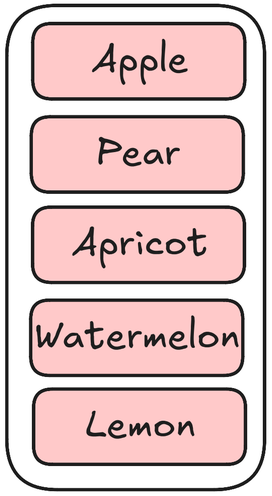
MRR (mean reciprocal rank) - shows how fast you can find the first relevant item.

$MRR = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{rank_i}$, $rank_i$ - the first relevant result for query i, N - number of queries.

$MRR \in [0; 1]$







MRR for 3 queries:

$$\frac{(\frac{1}{1}+\frac{1}{2}+0)}{3} = 0.25$$

Ranking quality metrics

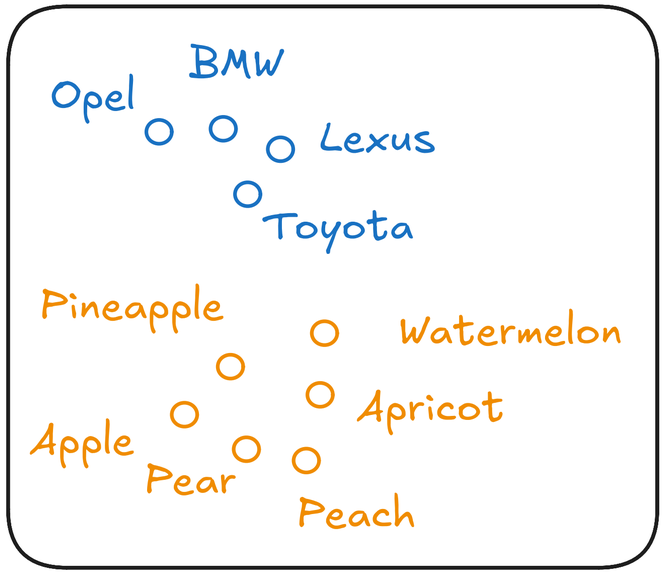
DCG

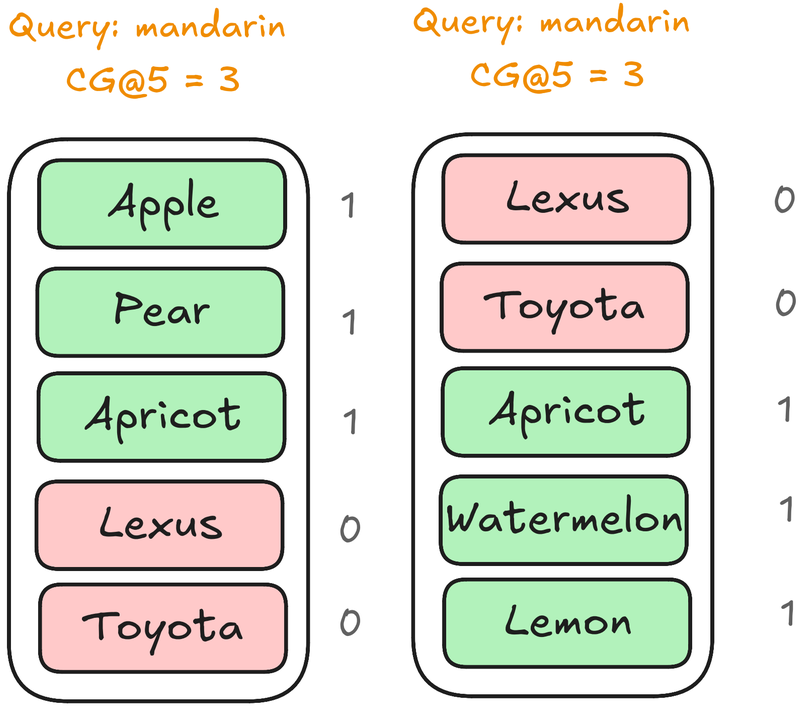
DCG - measures the total item relevance in a list with a discount that helps address the diminishing value of items further down the list

Cumulative gain is a sum of scores of all relevant items among the topK results

$$CG@k = \sum_{i=1}^{k} G_i$$

G_i - gain of a list item





Ranking quality metrics

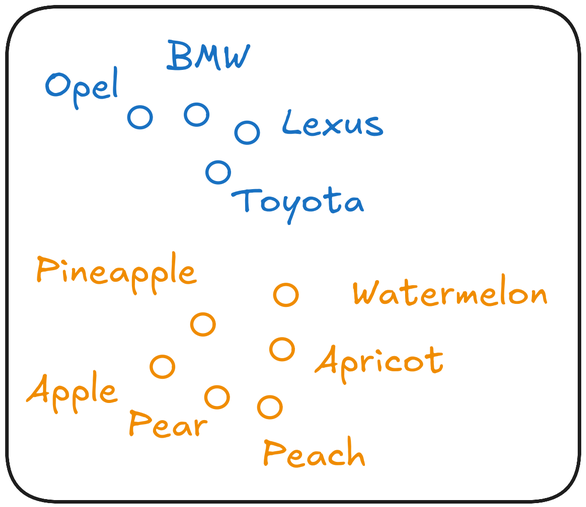
DCG

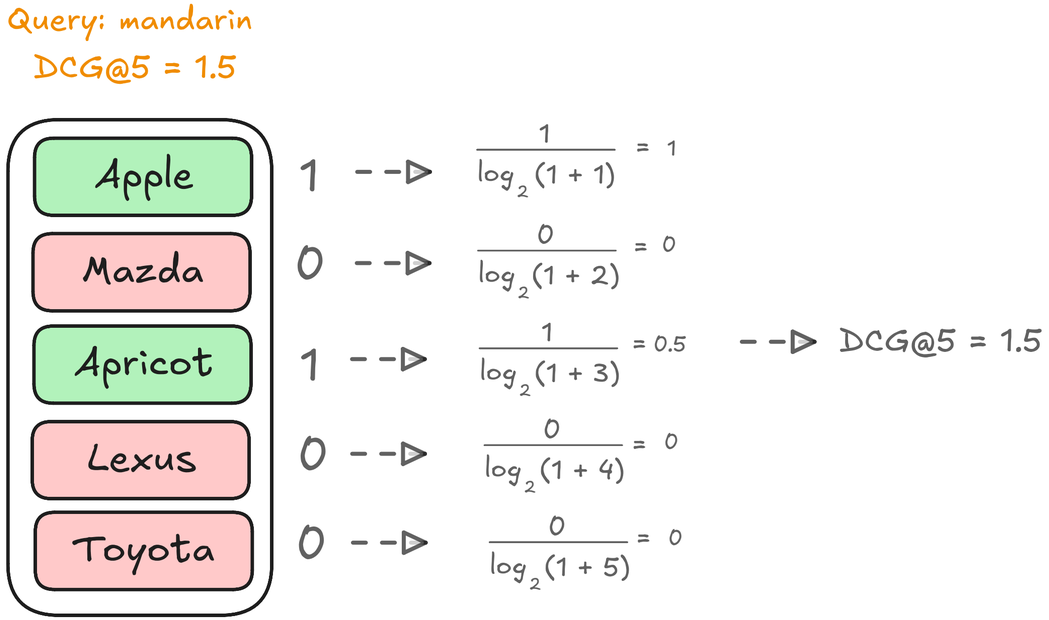
Discount - a logarithmic discount that helps assign lower gain when relevant items appear further down in the ranked list.

$$DCG@k = \sum_{i=1}^{k} \frac{rel_i}{log_2(i+1)}$$

rel_i - graded relevance of the item at position i.

Depends on the number of items in the list and the relevance scores.





Ranking quality metrics

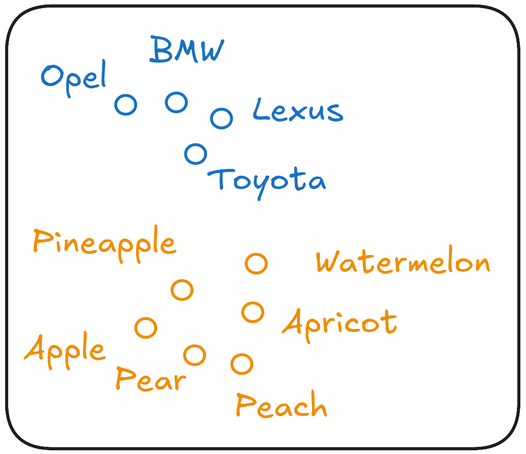
nDCG

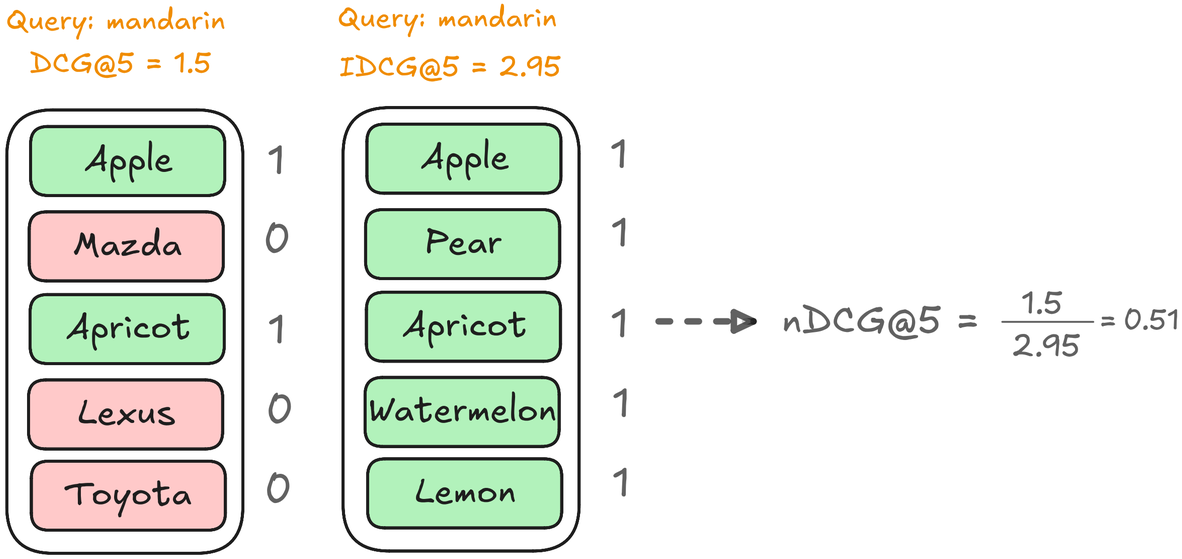
nDCG - normalised DCG, achieved by dividing DCG by the ideal ranking

IDCG@k - ideal ranking, where every item appears in the order of relevance.

$$nDCG@k = \frac{DCG@k}{IDCG@k}$$

$$nDCG@k \in [0; 1]$$





Ranking quality metrics

Mean Average Precision

mAP@k - evaluates the *overall quality of the entire ranks list*, rewarding systems that return *all relevant items* and place them *early* in the ranking.

mAP works only with binary relevancy scores.

$mAP \in [0; 1]$

$$mAP@k = \frac{1}{N} \sum_{i=1}^{N} AP@k_i,$$

k - cutoff point, N - total number of queries, AP - average Precision for a given ranking list.

Ranking quality metrics

Average Precision

$AP@K$- is computed as an average of Precision values at all the relevant positions within k.

Only Precision at ranks where the items are relevant are taken into account.

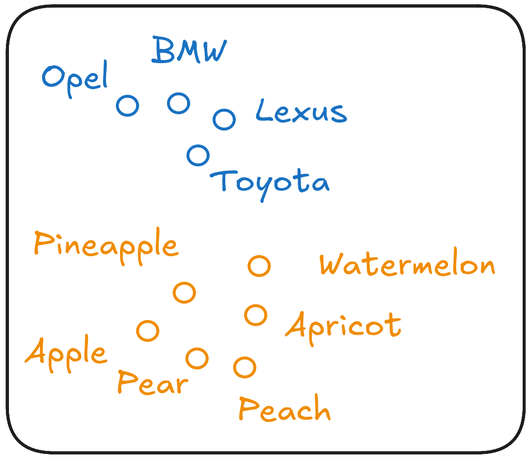
$AP@K \in [0; 1]$

$$AP@K = \frac{1}{N} \sum_{k=1}^{K} Precision(k) \cdot rel(k),$$

N - total number of relevant items for a particular query in the top-K results,

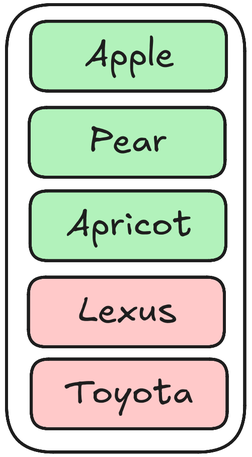
$Precision(k)$ - is the precision calculated at each position,

$rel(k)$ - equal 1 if the item at position k is relevant and 0 otherwise

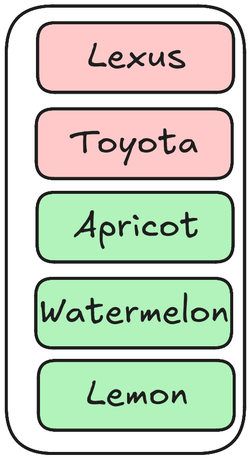


Query: mandarin
AP@5 = 1.0

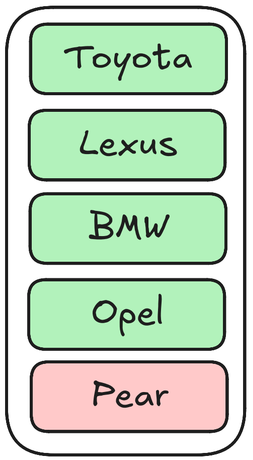
Query: mandarin
AP@5 = 0.6



Query: mandarin
AP@5 = 0.48



Query: Mazda
AP@5 = 1.0



How to find an embedding model?

MTEB, RTEB, BeIR

MTEB (Massive text embedding benchmark) - is the standard and most widely used benchmark suite for evaluating text embedding models across any tasks.

MTEB/RTEB

Contains 8+ task categories and 50+ datasets

The most relevant for search task is *Retrieval*

Beir

Benchmarking IR

RTEB (Retrieval text embedding benchmark) - is a separate benchmark inspired by MTEB.

Contains only retrieval tasks

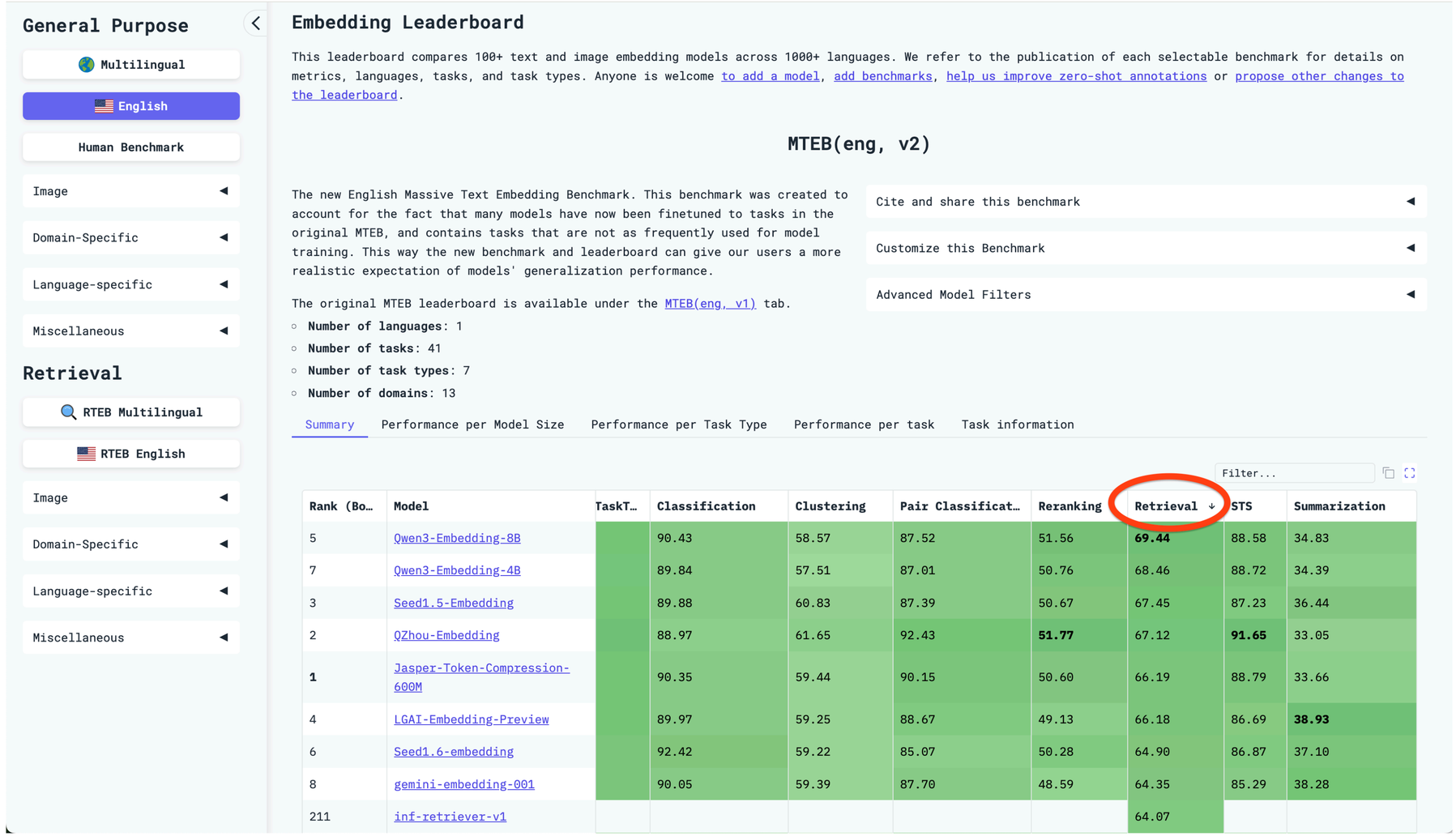
Designed to better evaluate real-world RAG retrieval

Includes only query → passage retrieval datasets

BeIR (Benchmarking Information Retrieval) - is a former standard of evaluating retrieval models, consists of 18 information-retrieval datasets. Now it is getting outdated cause it is not fully aligned with production retrieval.

How to find an embedding model?

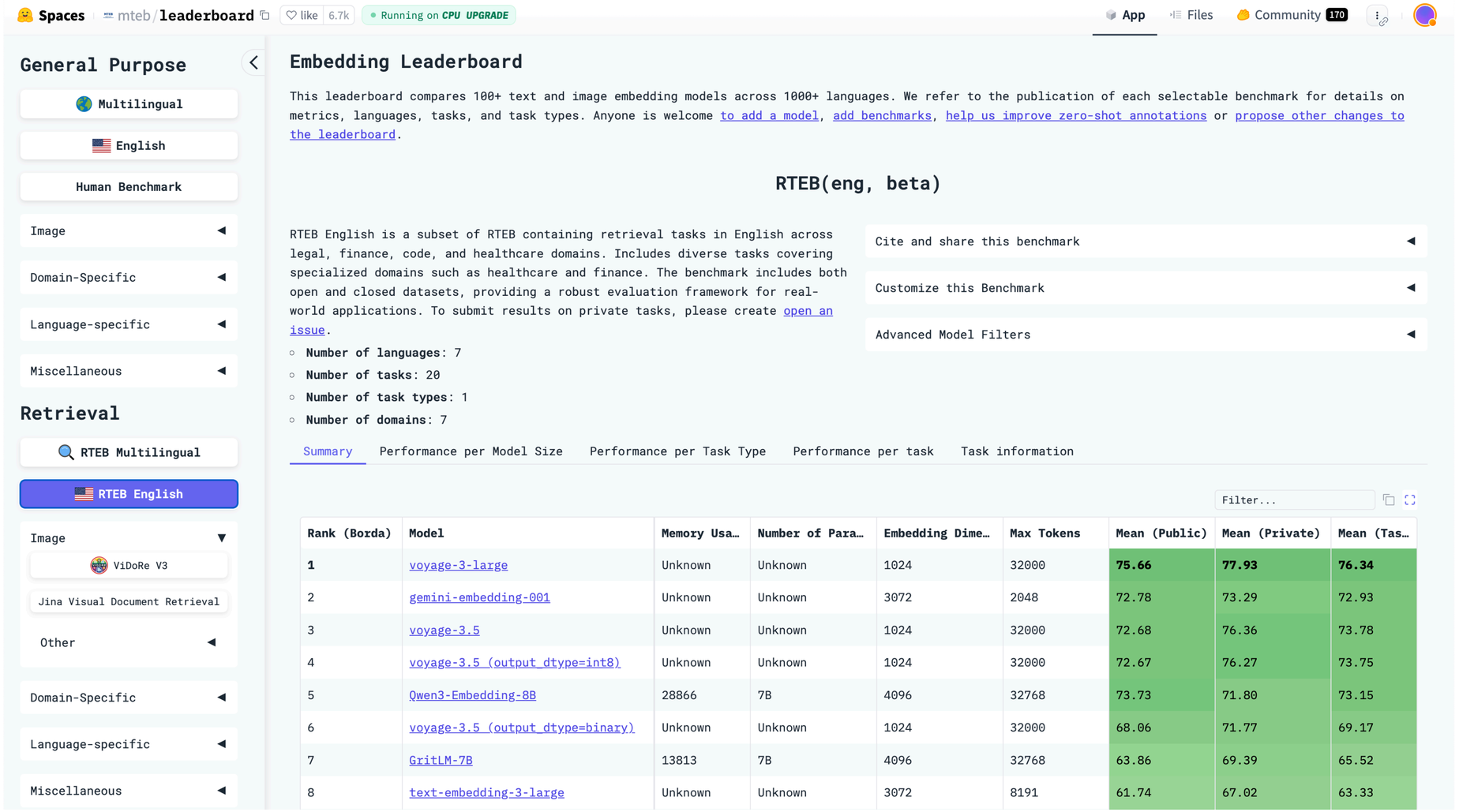
MTEB



https://huggingface.co/spaces/mteb/leaderboard

How to find an embedding model?

RTEB



https://huggingface.co/spaces/mteb/leaderboard

Image dense embeddings

- Continuous, high-dimensional vectors (e.g. 256-4096 dims)
- Capture high-level semantics: objects, style, context
- Robust to small variations (lighting, minor transforms)
- Quite heavy
- Requires preprocessing as any image model

Examples:

- *DINO / DINOv2*
- *CLIP visual*
- *ViT with contrastive learning*
- *CLIP (text-image)*

How it was done before

B.C. methods

Handcrafted feature-based retrieval with:

- Global features: colour histograms, texture descriptors, shape descriptors
- Local features / Bag-of-Visual-Words:

-Keypoints: SIFT, SURF, ORB

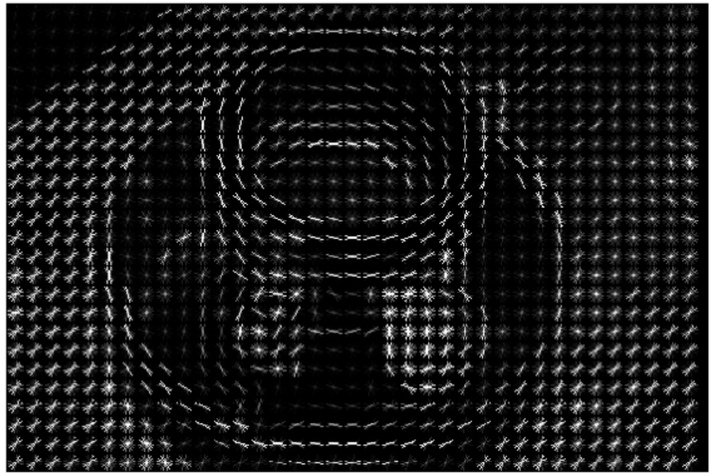
-Shape descriptors (HoG, SIFT, etc.)

Problems:

Sensitive to variations, no semantic understanding, might be heavy and inefficient, limited generalisation.

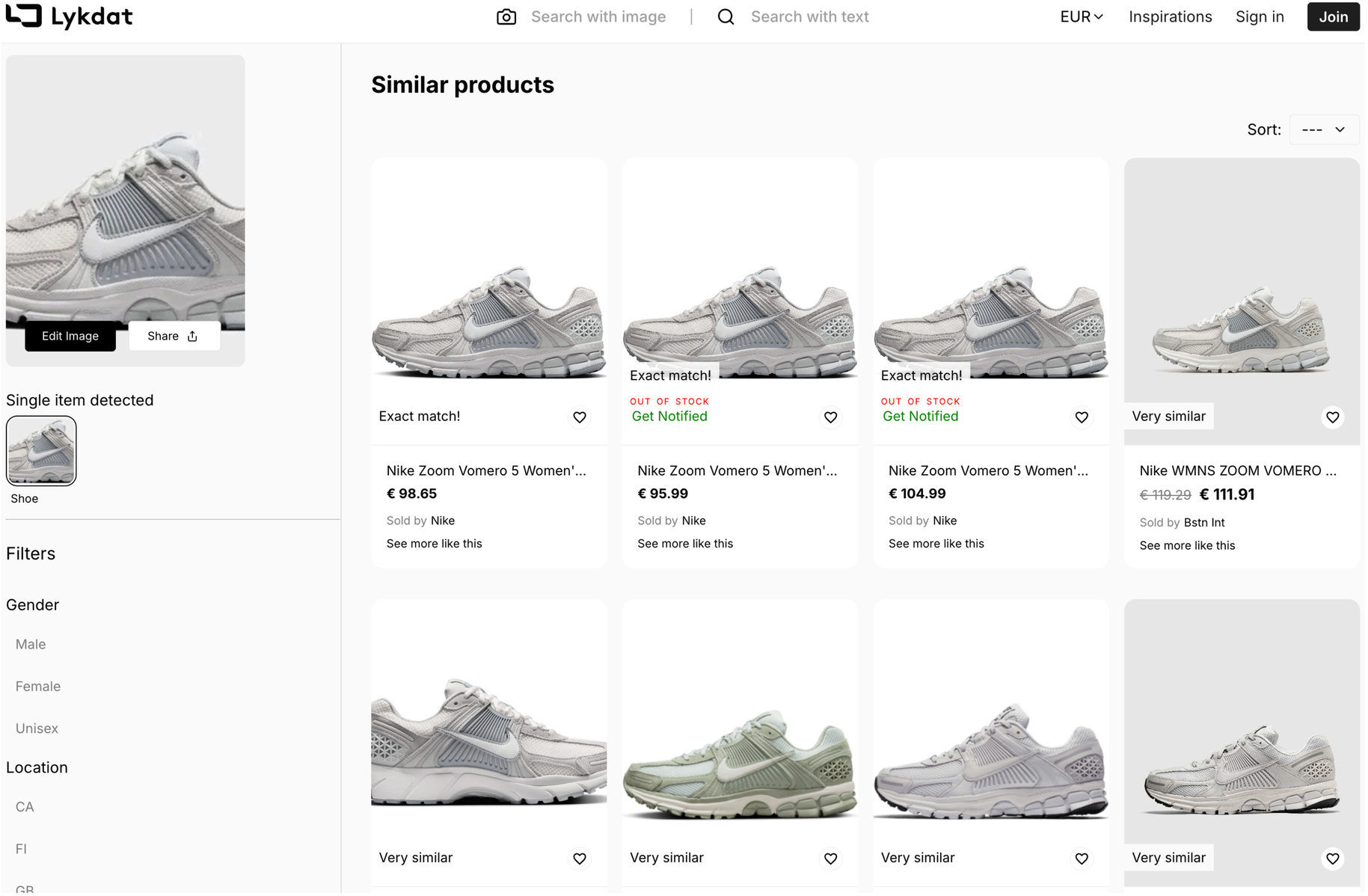
Test Image

Histogram of Oriented Gradients



Examples

- Fashion search platforms
- E-commerce product search
- Medical image search
- Reverse image search for arbitrary objects
- Defect detection
- Data cleaning



ResNet, ViT, DINO, etc

Pretrained classification models (ResNet, ViT)

Idea: Use models trained for classification as feature extractors

Remove the final classification head → take the embeddings from the last layer

Pros: Simple, widely available.

Cons: Not optimised for retrieval, embeddings may be less semantically aligned

Pretrained or fine-tuned for retrieval (DINO / DINOv2, CLIP)

Idea: Models explicitly trained to produce embeddings for similarity

Use contrastive or self-supervised objects to align semantically similar images, often produce more isotropic and well-structured embeddings.

Pros: embeddings are semantically meaningful and optimised for similarity search.

Cons: fewer models readily available than classification models; may require fine-tuning for domain-specific daa.

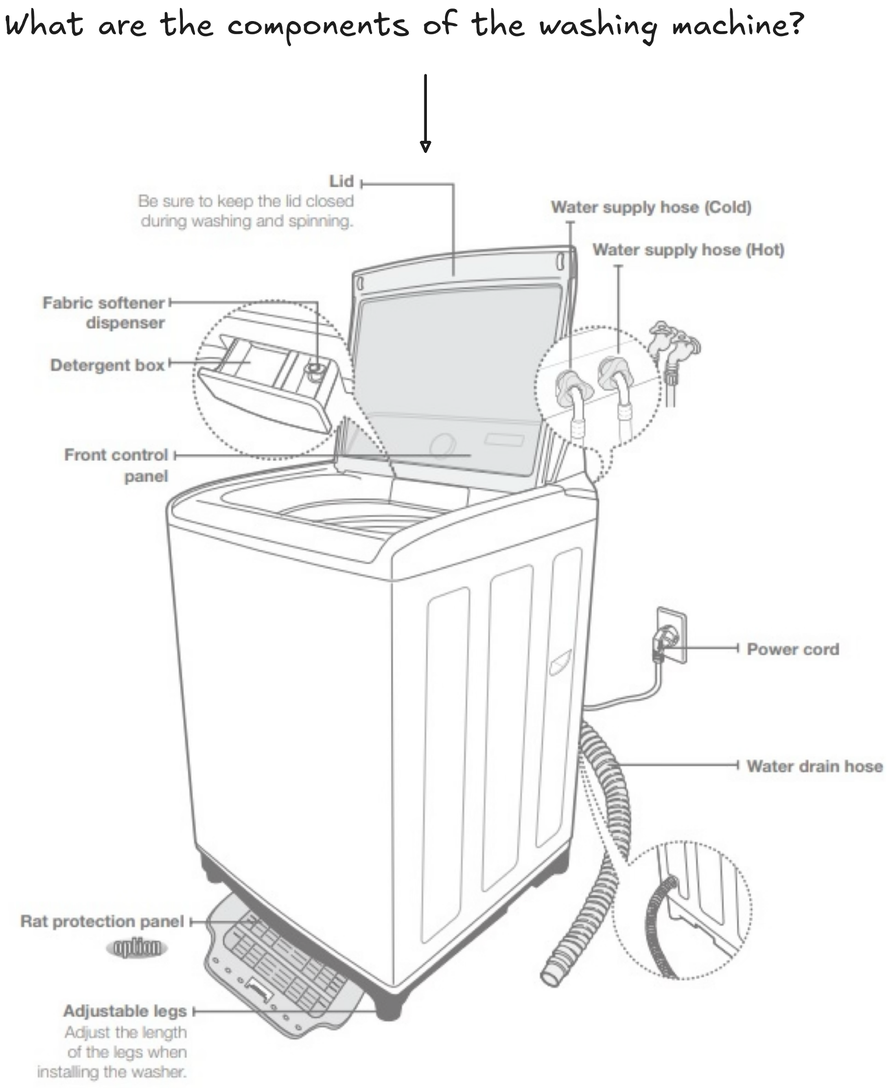
Multimodal embeddings

Goal:

Map diverse data types (e.g. text and image) into a shared high-dimensional space where their semantic similarity is directly proportional to their vector distance

Mechanism:

Models are trained to align the different modalities, e.g. with contrastive learning

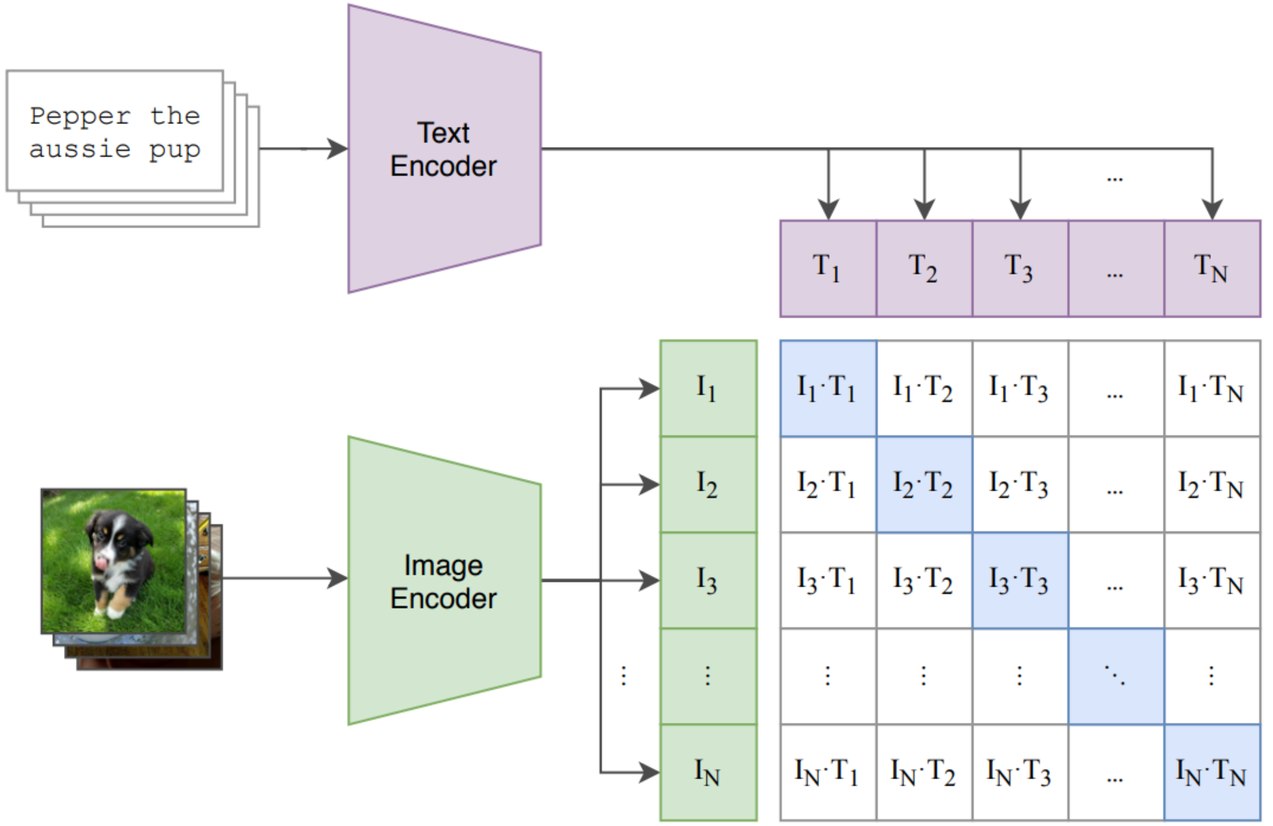


CLIP

Maps texts and images into a shared embedding space such that similar texts and images are located close to each other.

Constructed of a text encoder and image encoder trained from scratch* 400M+ image-text pairs collected from the internet.

Alignment is done via training with infoNCE loss.



*There is a vast amount of CLIP-based models, many of them were trained via pre-trained encoders.

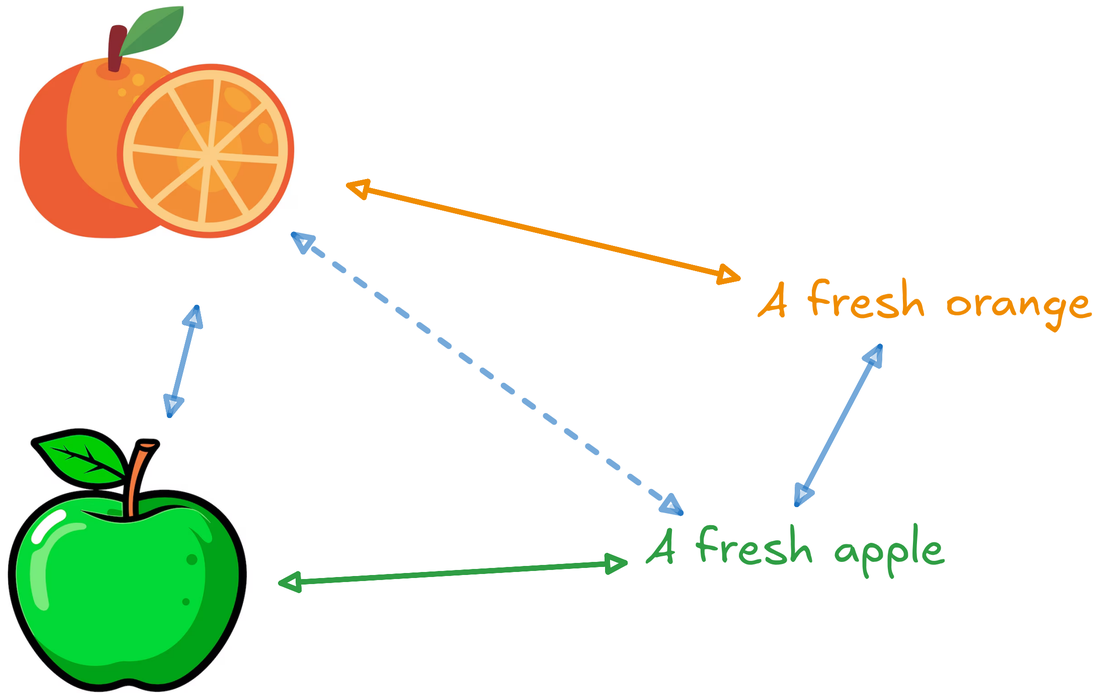
Learning Transferable Visual Models From Natural Language Supervision

CLIP

- Support text-to-image retrieval and any other combinations of the two
- Has a flexible architecture, e.g. can be trained with ResNet, ViT or other models
- Works well across styles (e.g. “sketch of a banana” and “photo of a banana”)
- As a popular model, have been fine-tuned for different type of applications by the community (e.g. *Marqo/marqo-fashionCLIP)*
- Might be deconstructed for inference, so image encoder and text encoder will be able to scale differently
- Struggles with counting (“3 cats vs 4 cats”), spatial relations (“cup on a table”), and very specific sub-categories (e.g., exact car models)
- Trained on captions, has 77-token limit
- Might show poor performance in specialised domains and require fine-tuning (medical images, manufacturing defects, etc.)
- Might not be the best model for retrieving images based on text written inside the image

Multimodality gap

- In CLIP-style contrastive models, text and image embeddings often occupy distinct regions in the shared space
- Contrastive learning reinforce separation: the loss pushes non-matching pairs apart, but doesn't explicitly force the two modalities to fully intermix
- False negatives during training: random negatives may still be semantically related, pushing embeddings further apart.
- Might affect applications using multiple search methods simultaneously (e.g., image2image + text2image)
- Relatively safe for text2image or image2text alone.

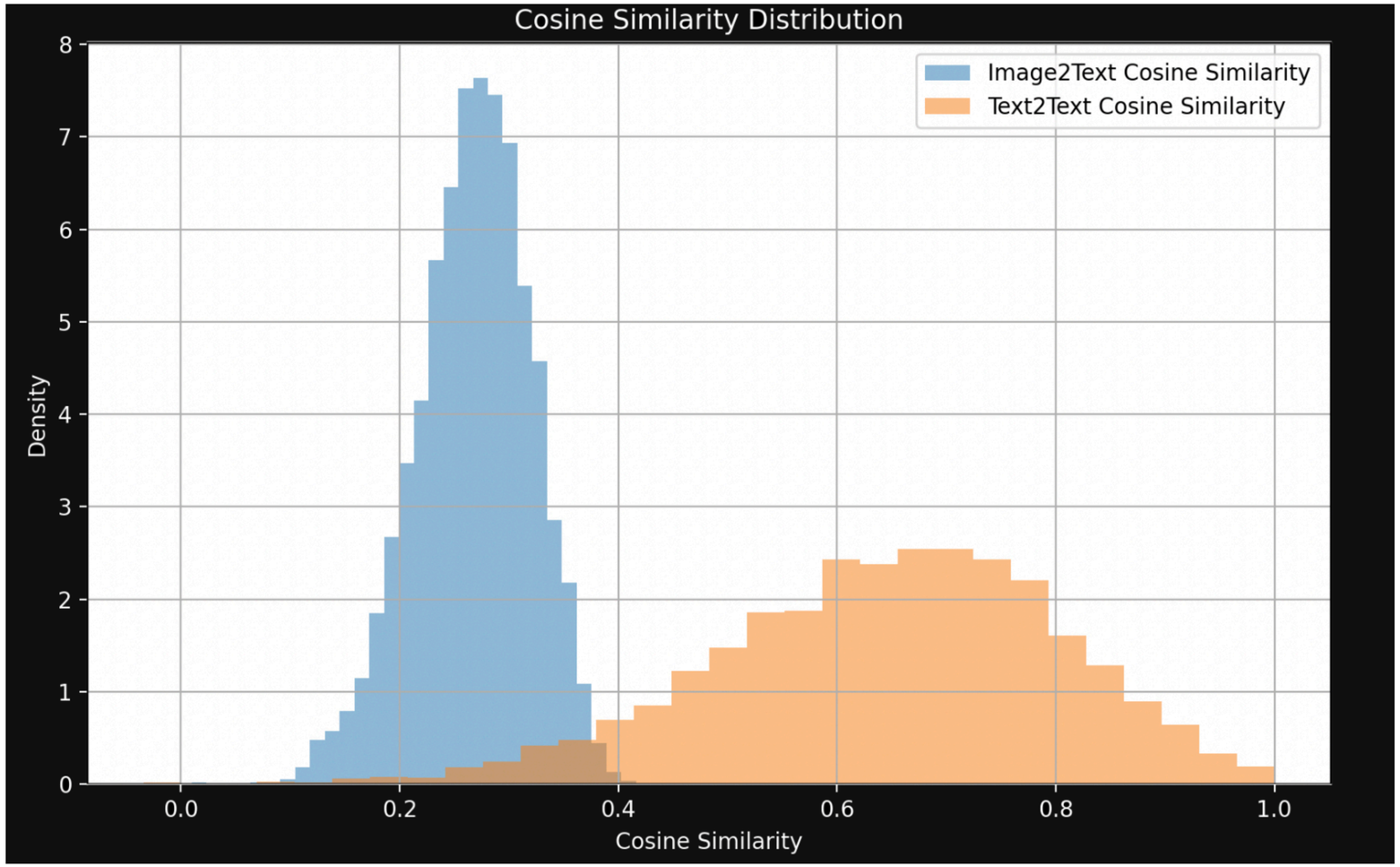


Multimodality gap

JINA AI experiments

Authors of the post embedded a dataset of images and their respective captions (5 captions per image) with Jina CLIP, then:

- Compared the cosine similarities of the image embeddings to the embeddings of their caption texts
- Compared the cosine similarities between the taken captions



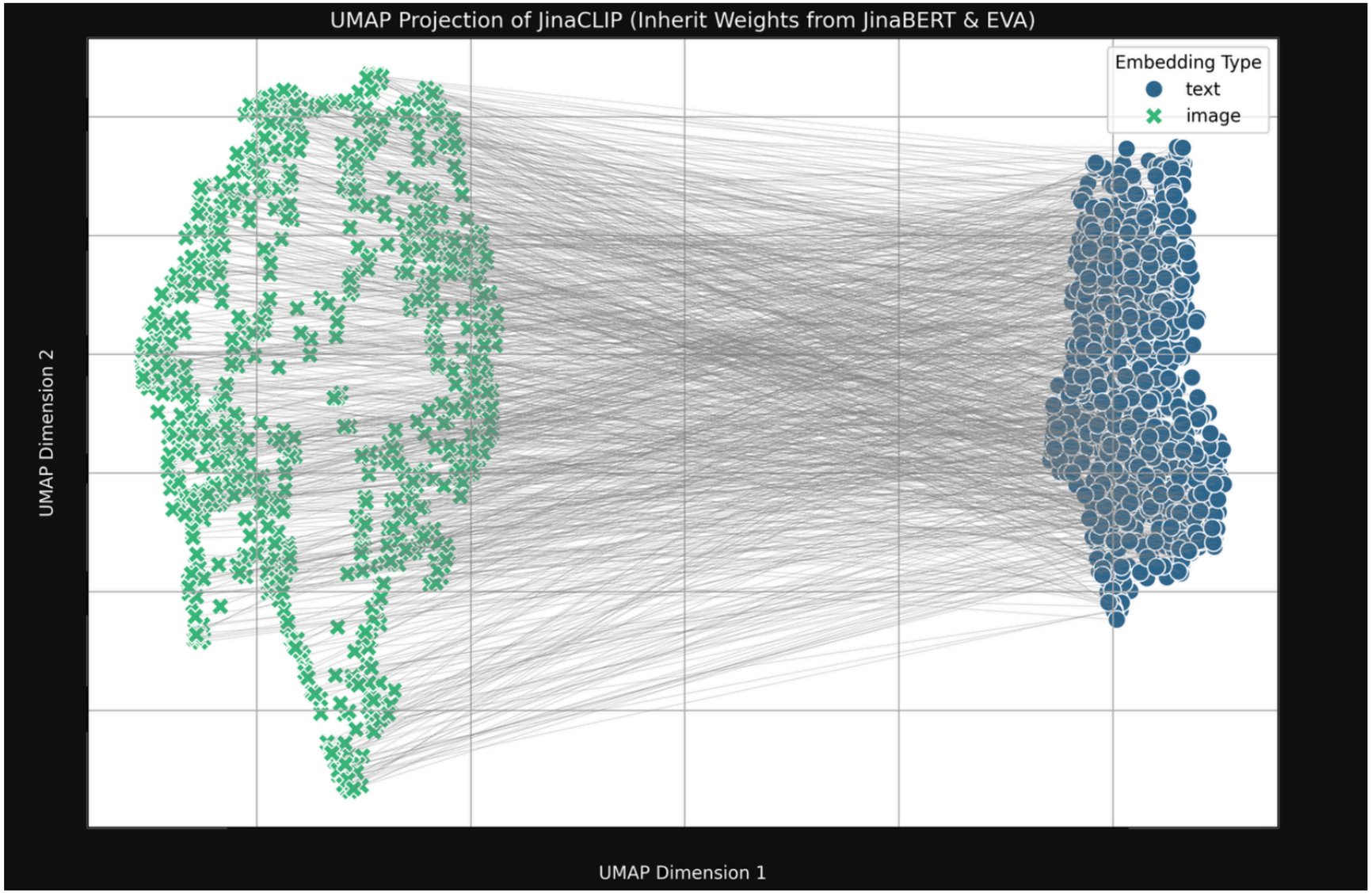
The What and Why of Text-Image Modality Gap in CLIP Models

Multimodality Gap

Jina CLIP before training

Image embeddings and text embeddings connected by the grey lines indicating matches image-text pairs.

The cone is seen even before training.

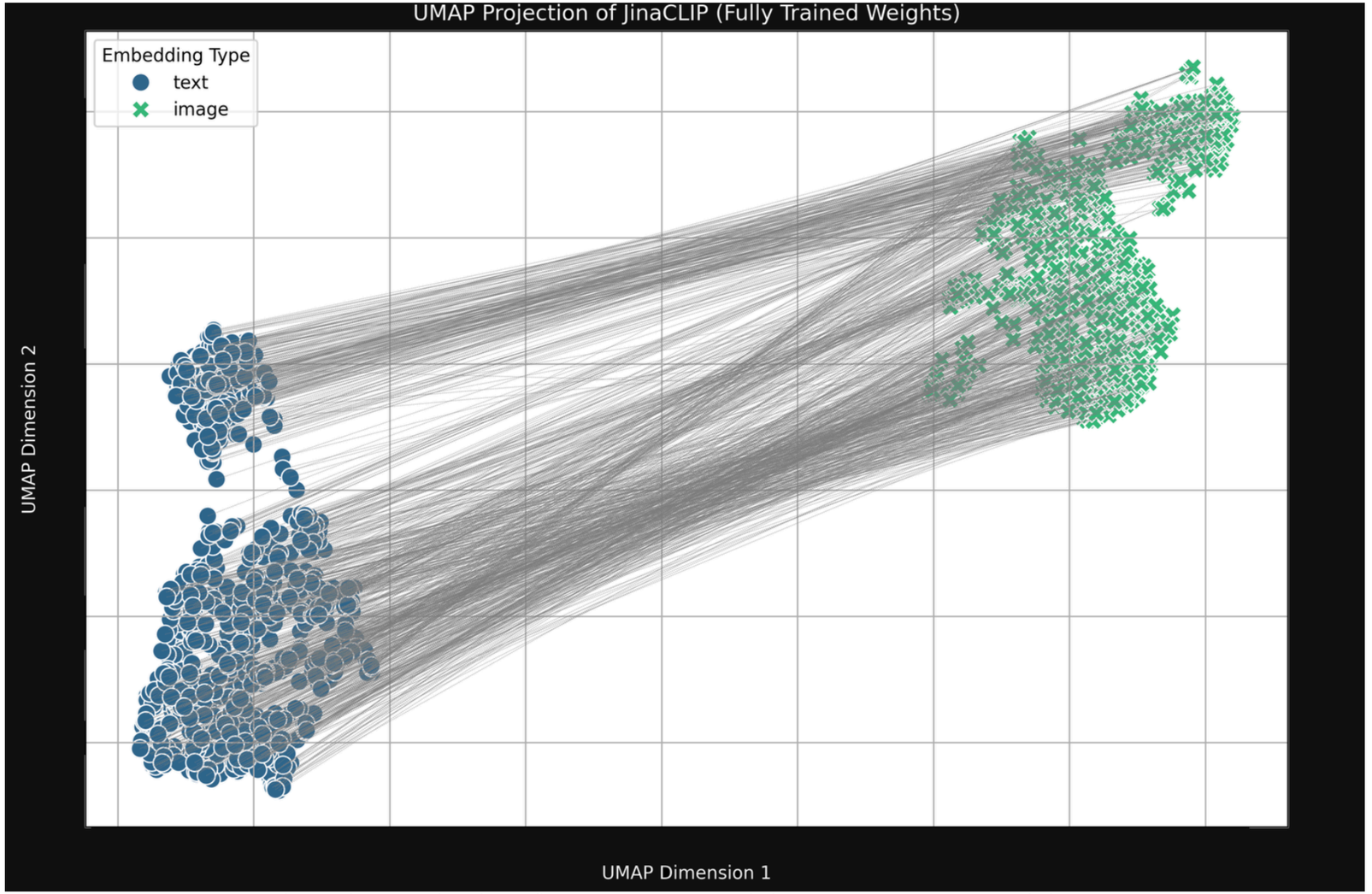


The What and Why of Text-Image Modality Gap in CLIP Models

Multimodality Gap

Jina CLIP after training

- After training, the distance between the modalities becomes even larger, confirming the loss reinforcing the gap.
- Even using the more expensive OpenAI approach with purely random initialisation, training could not get rid of the bias



The What and Why of Text-Image Modality Gap in CLIP Models

Questions?