Hybrid search

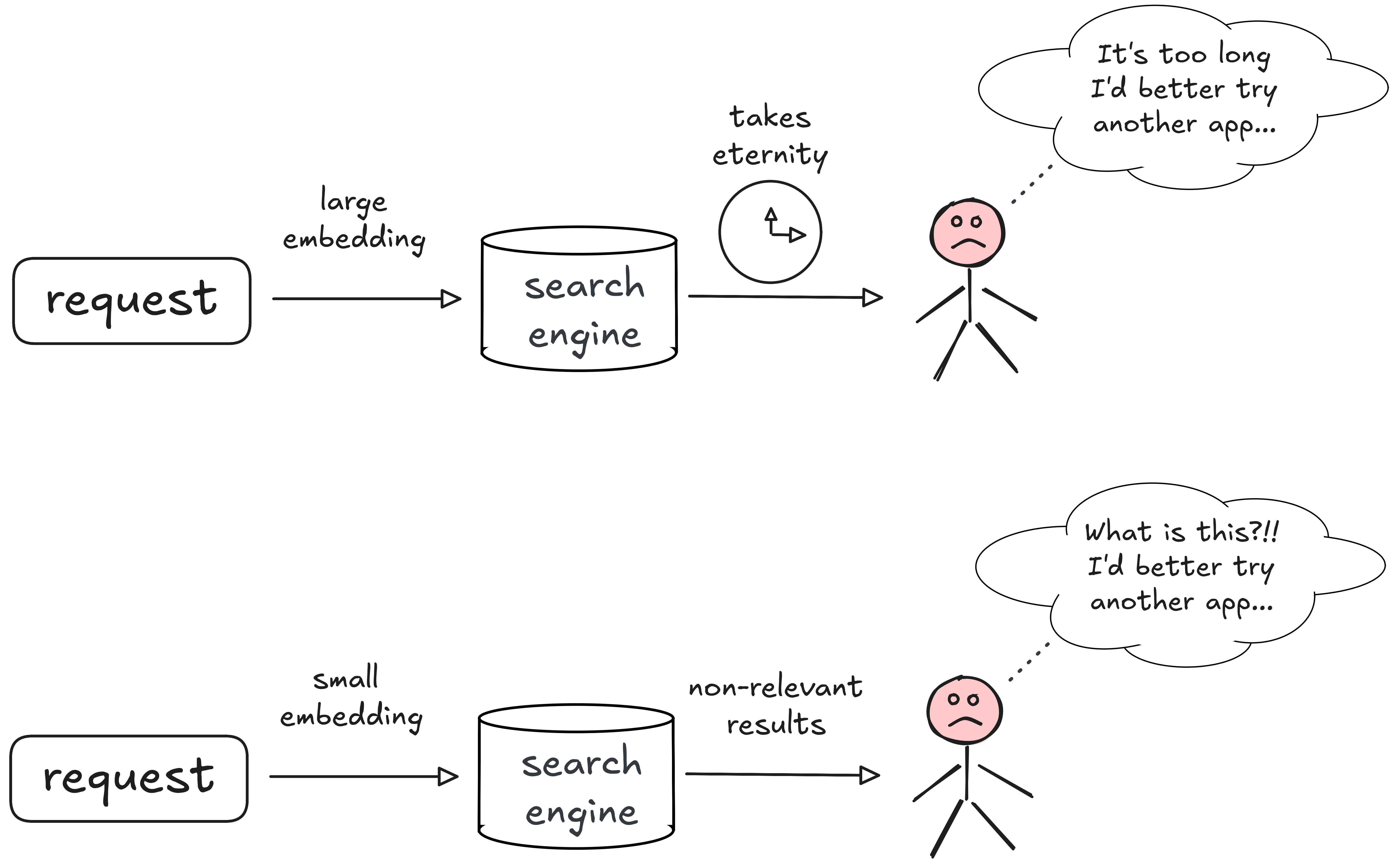
George Panchuk
HSE AI Fall 2025

github.com/joein/vector-search-course

Multistage search

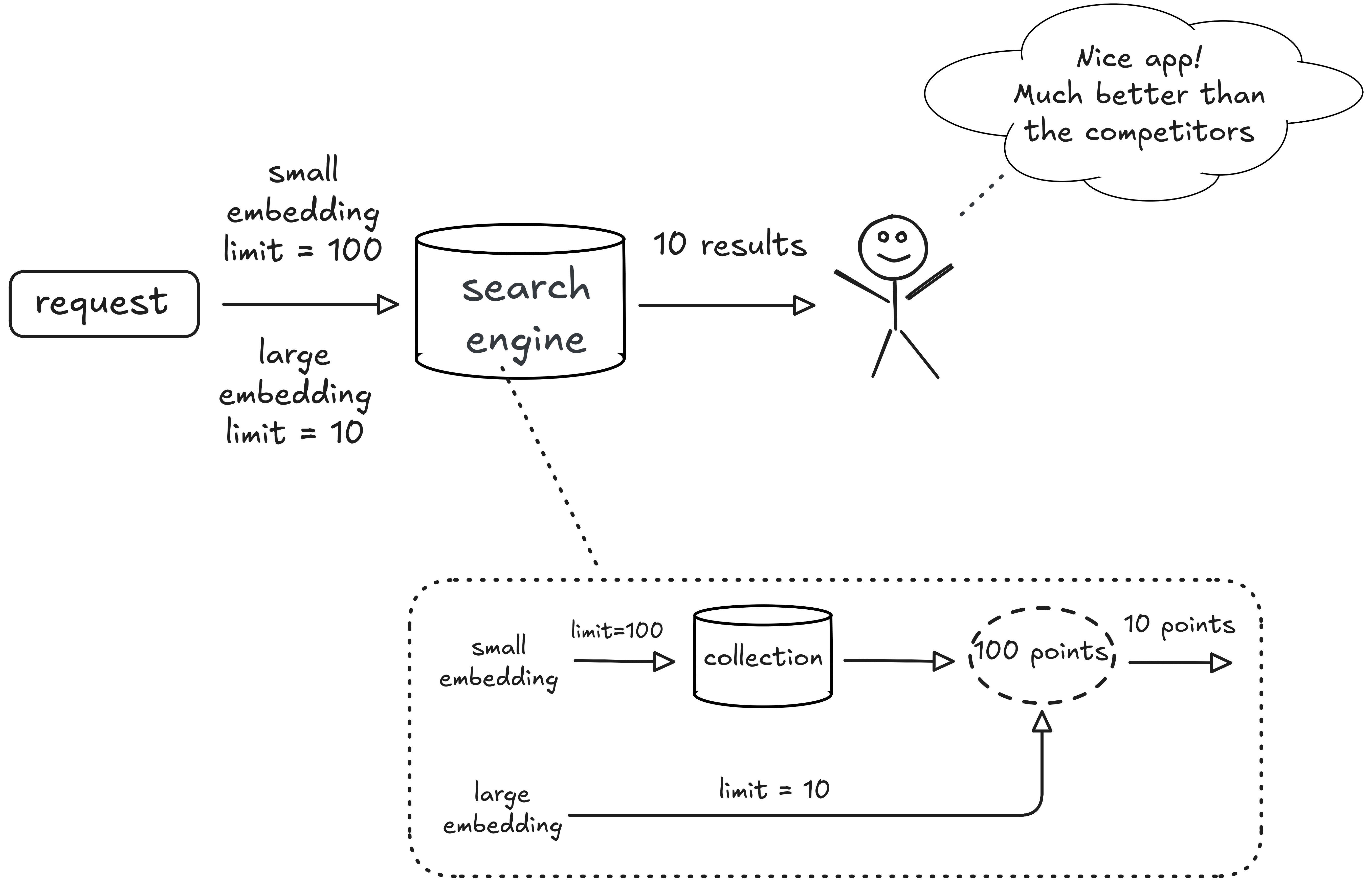
Intro

- There is no silver bullet in the world of embeddings, different applications might have different requirements
- There might not be a model which satisfies app requirements in all aspects, e.g. memory, speed or recall
- In such cases, building a multistage pipeline might help



Multistage search

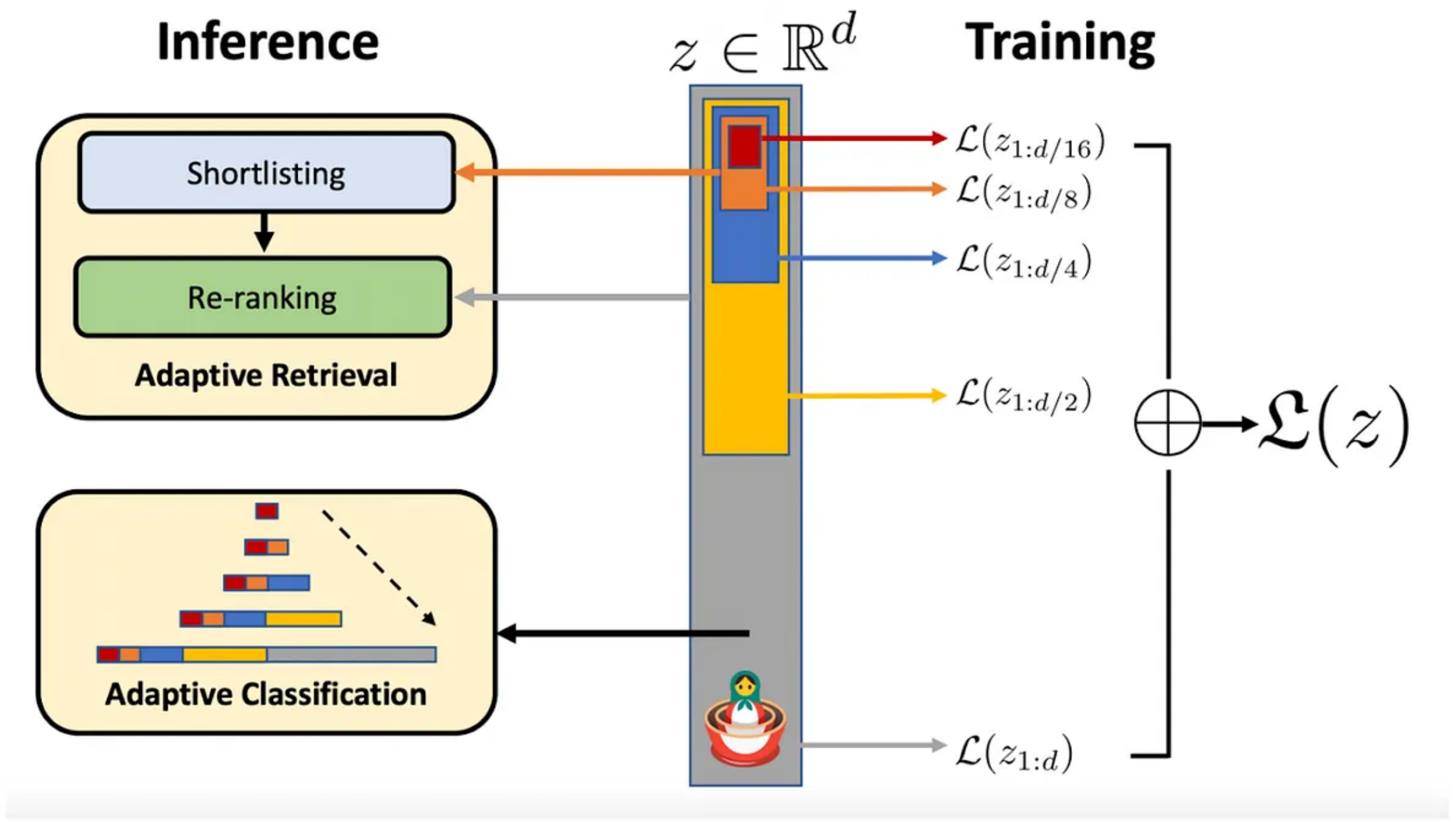
- Situation: you compared 2 models, one is fast, but recall is a bit off, another one is precise, but search takes too long
- Solution: search with the fast model with oversampling (e.g. limit=100) first, re-score results with the slow model
- Problems: you have to serve 2 models, the fast model might not fetch relevant results even with oversampling
- Examples: intfloat/multilingual-e5-small + intfloat/multilingual-e5-large, MRL, quantised vectors



Matryoshka representation learning (MRL)

- MRL is a model-agnostic method for neural networks to produce embeddings of various sizes using a single forward pass
- Optimizes several loss functions to optimise feature representations for specific dimensions - a loss per dimension.
- The initial dimensions of the features carry more significant information than the later ones: first few contain high-level details, while the later ones focus on more granular information
- Examples: openai/text-embedding-3-large, mixedbread-ai/mxbai-embed-large-v1

$$\min_{\{\mathbf{W}^{(m)}\}_{m\in\mathcal{M}},\ \theta_F} \frac{1}{N} \underbrace{\sum_{i\in[N]}}_{\text{sum over N data points}} \overbrace{\sum_{m\in\mathcal{M}}}^{\text{sum over M vector dimensions}} \underbrace{c_m}_{\text{scaling factor/ weight of each dimension}} \cdot \underbrace{\mathcal{L}\left(\mathbf{W}^{(m)} \cdot F(x_i;\theta_F)_{1:m}\ ;\ y_i\right)}_{\text{Loss function}}$$

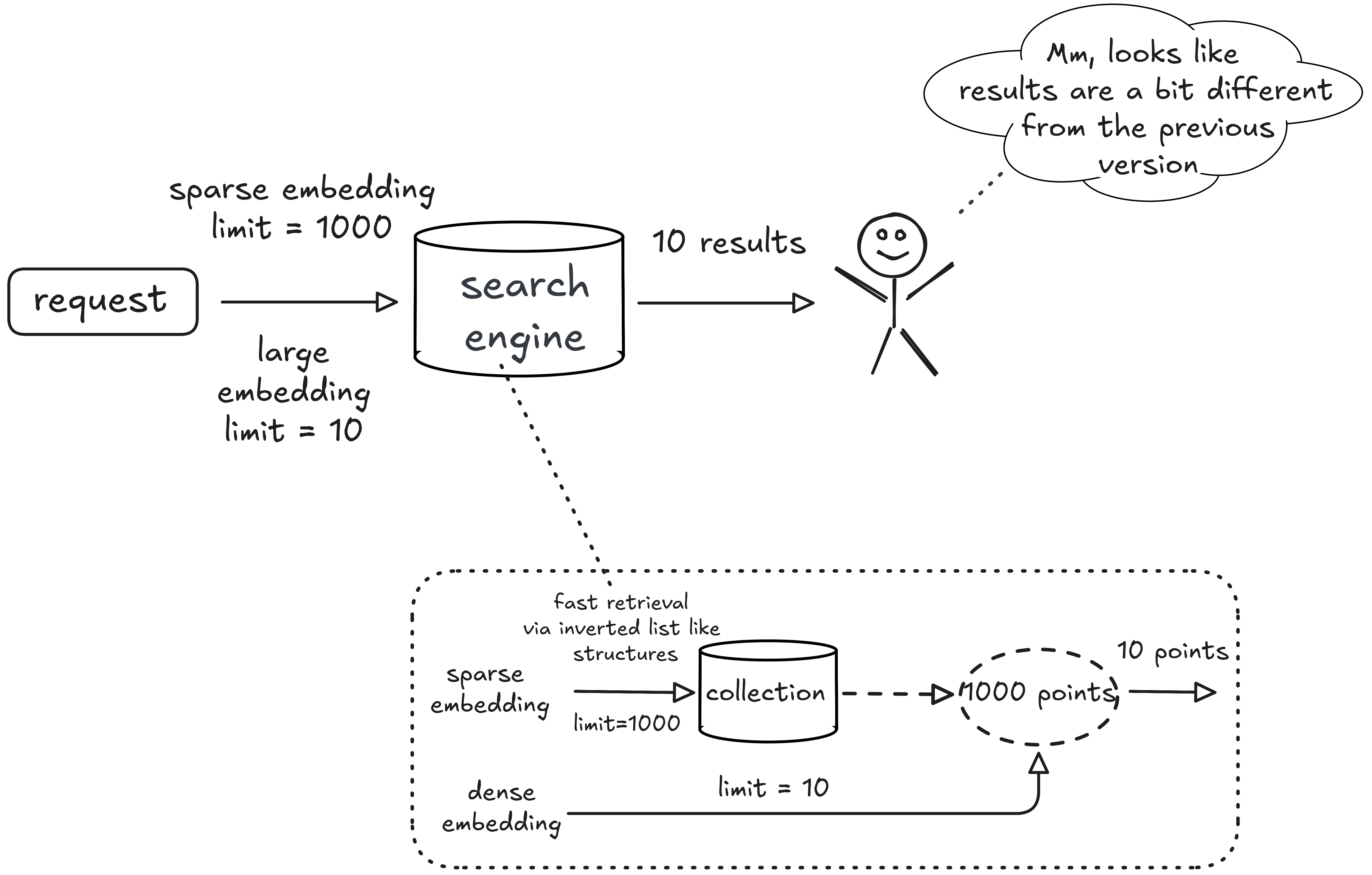


Matryoshka Representation Learning

Hybrid search?

Sparse retrieval and dense reranking

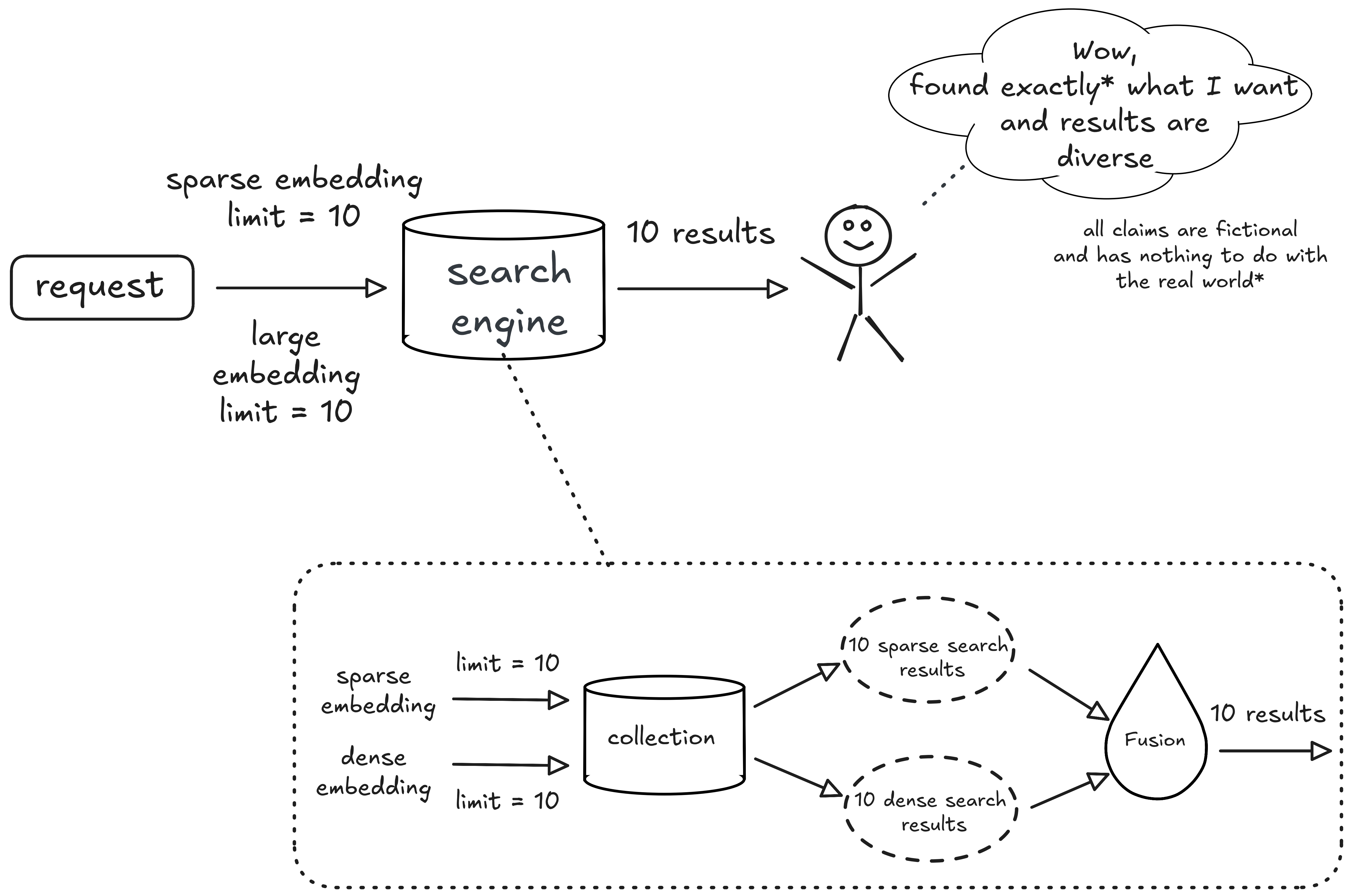
- Sparse retrieval might work even faster than search with small dense embeddings
- It can help narrow down the results for the semantic model
- As well as guarantee that the results will contain exact query terms matches
- Documents without exact matches won't make it to the results


Mm, looks like results are a bit different from the previous version
sparse embedding limit = 1000
request
large embedding limit = 10
search engine
10 results
fast retrieval via inverted list like structures
sparse embedding
limit=1000
collection
1000 points
10 points
dense embedding
limit = 10

Hybrid search

Dense and sparse with fusion

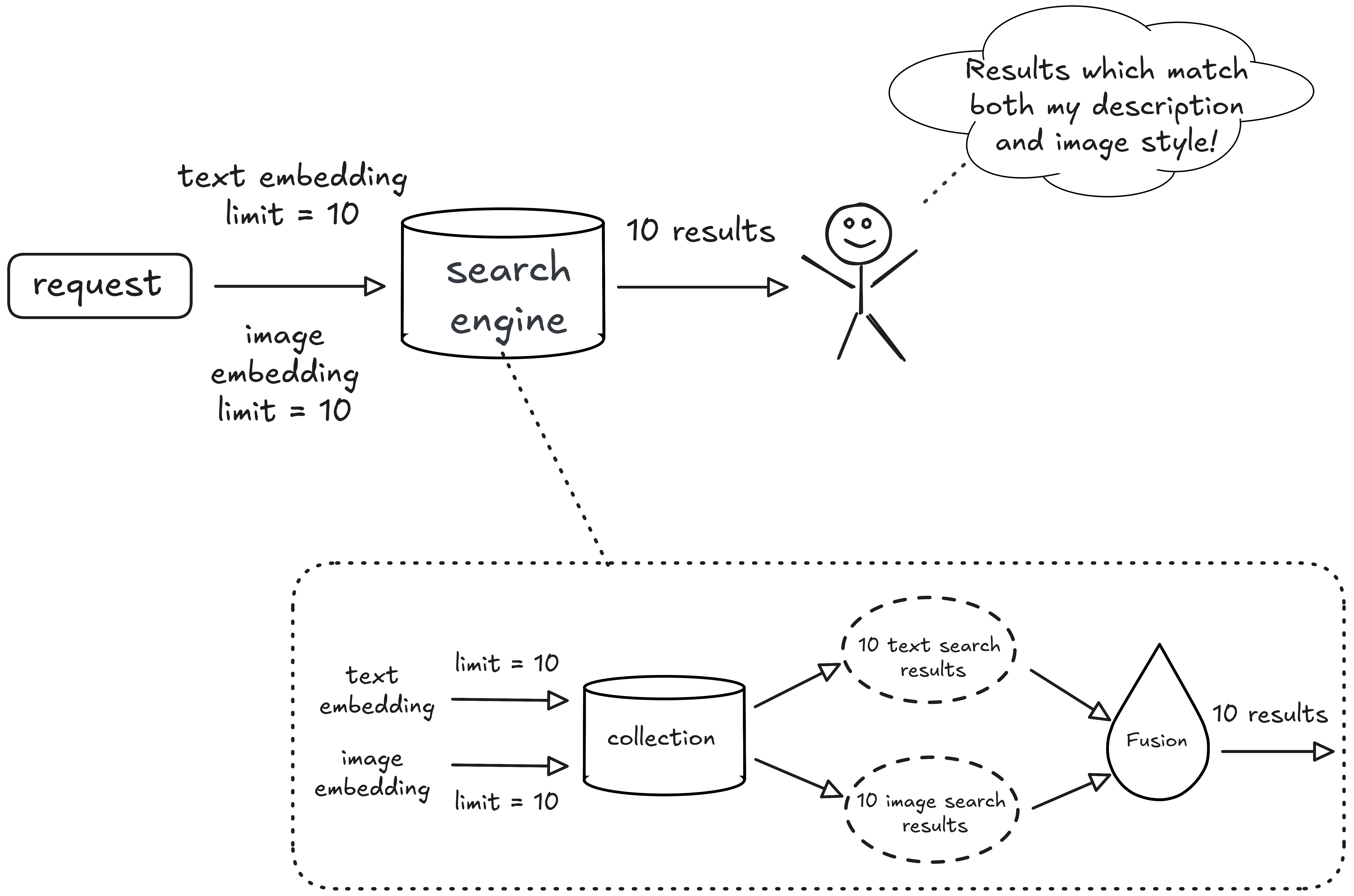
- Dense retrieval struggles with exact term matches
- Sparse retrieval struggles with semantics
- Sparse retrieval with dense reranking might miss out semantically close queries, if they don't contain exact matches
- Let's search with both models on the entire dataset and then merge the results
- But how do we merge them?...



Hybrid search?

Multimodal

- Who said we are limited to text?
- As long as we know, how to fuse the results, we can search on whatever embeddings we want
- *Be careful with multimodality gaps in multimodal models



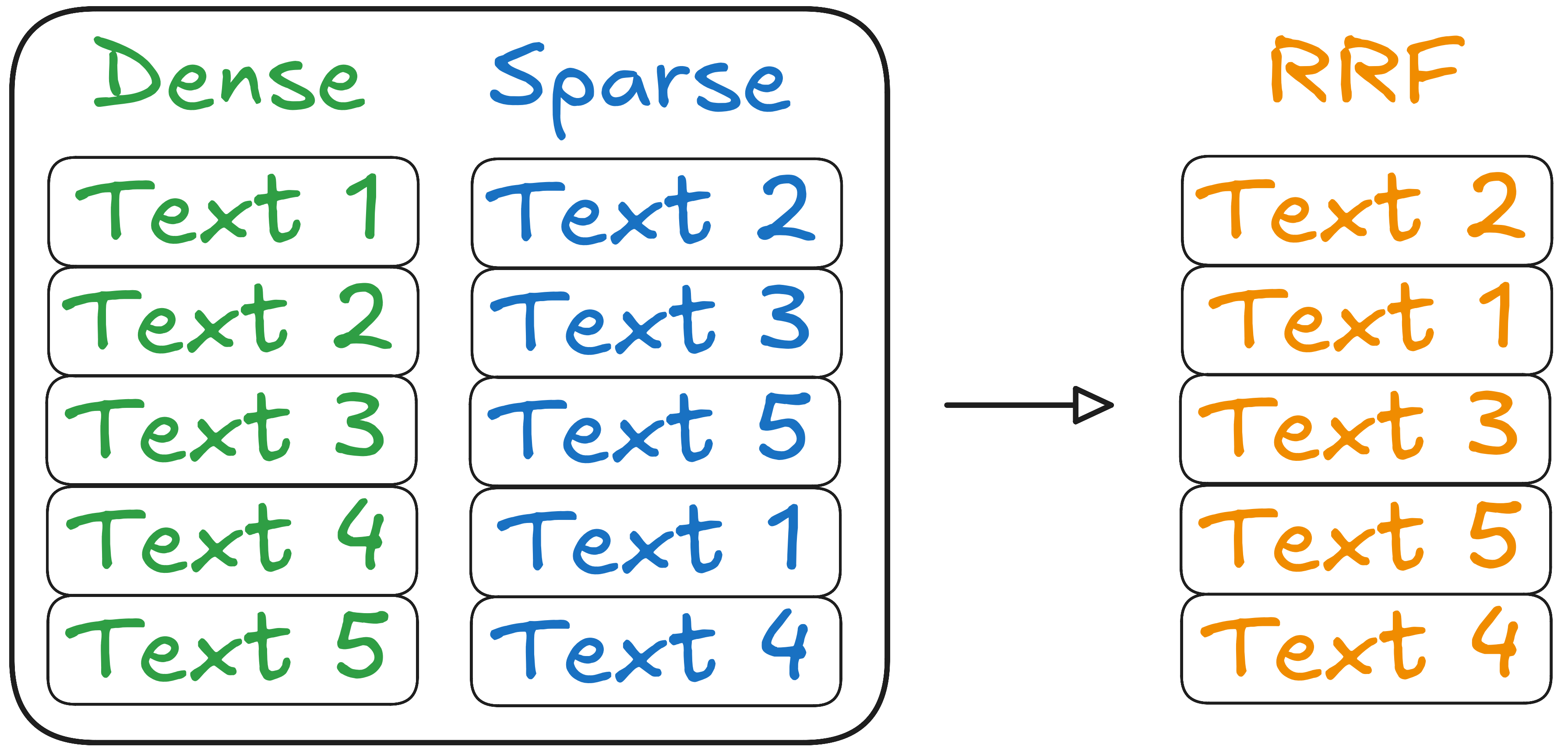
Hybrid search

Finally merging: Reciprocal rank fusion (RRF)

RRF is a rank aggregation method that combines ranking from multiple sources into a single, unified ranking

$RRF(d) = \sum_{r \in R} \frac{1}{k + r(d)}$, where *d* is a document, *R* is the set of retrievers, *k* is a constant (typically 60), *r(d)* is the rank of document *d* in retriever *r*

- Documents ranked highly by multiple retrievers are favoured in the final ranking (in general)
- Contribution to the score decreases non-linearly as rank increases
- *k* - is a smoothing factor, it prevents any single retriever from dominating the results, especially among lower-ranked items, but the value is chosen empirically
- Drawbacks: ignores absolute scores, small *k* emphasises top ranks → unstable, large *k* - everything looks similar; rewards results appearing high in any list, but does not reward consistency (doc #1 in one retriever might be chosen over doc ranked #5 in multiple retrievers)



$k = 2$

$1/(2+1) + 1/(2+4) = 0.5 \longrightarrow 2$

$1/(2+2) + 1/(2+1) = 0.583 \longrightarrow 1$

$1/(2+3) + 1/(2+2) = 0.45 \longrightarrow 3$

$1/(2+4) + 1/(2+5) = 0.309 \longrightarrow 5$

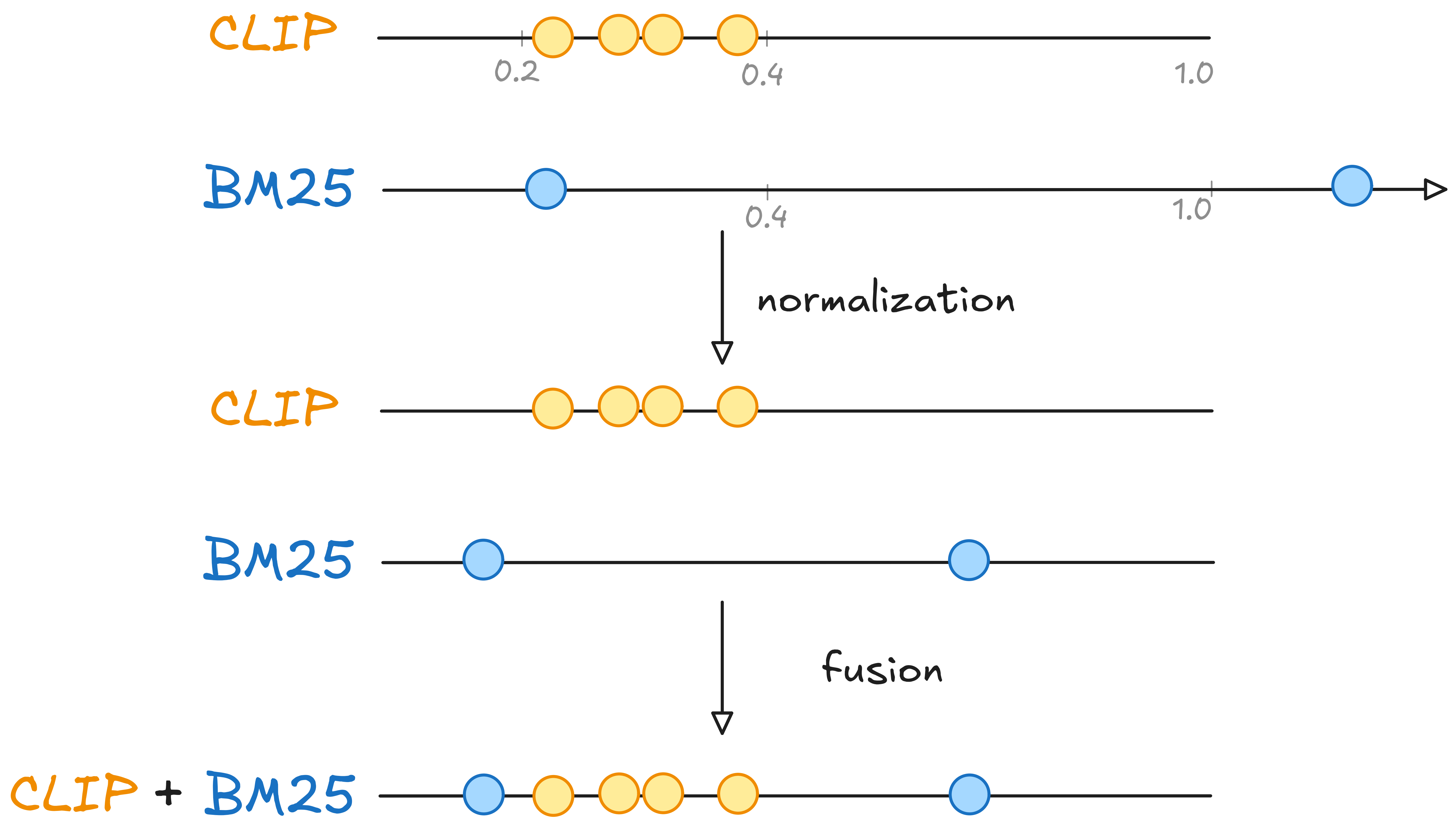
$1/(2+5) + 1/(2+3) = 0.342 \longrightarrow 4$

Hybrid search

Distribution based score fusion (DBSF)

- RRF does not take into account score values
- Different models can have different score ranges, e.g. dense models range is usually in [-1;1], while bm25 might not have an upper cap
- Dense models scores tends to concentrate around their mean values, which can vary across different models
- DBSF maps each retriever's scores to a shared range based on its distribution, preserving relative confidence
- *Qdrant does not accumulate distribution statistics and compute DBSF for each search result separately

$$S\left(\bigcup_{i \in E} \frac{x_i - (\mu_i - 3\sigma_i)}{(\mu_i + 3\sigma_i) - (\mu_i - 3\sigma_i)}\right)$$



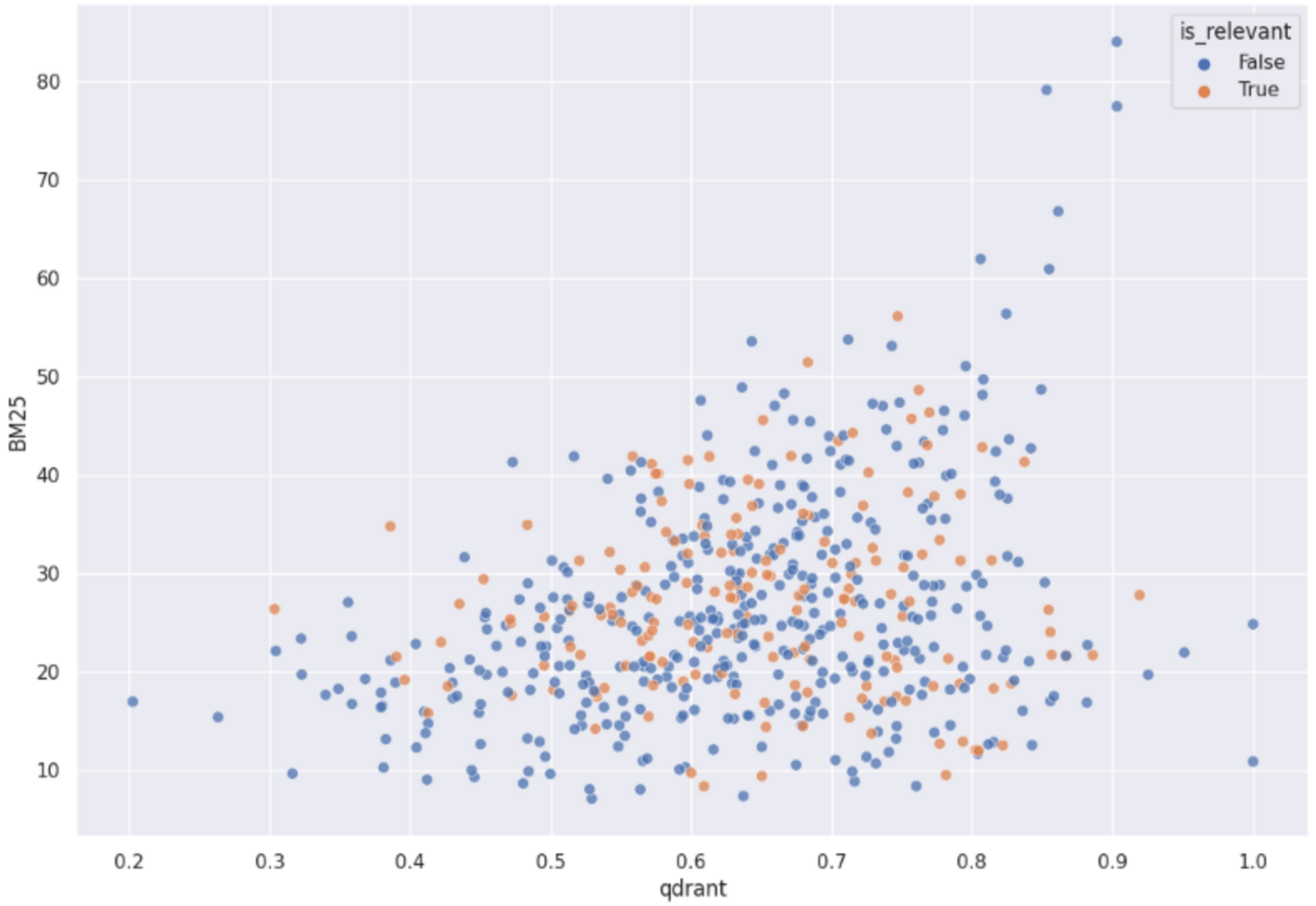
Hybrid search

Why not linear combination

- It is usually proposed to use full-text and vector search scores to form a linear combination formula to rerank the results, e.g.

$$score = 0.7 \cdot dense\_score + 0.3 \cdot sparse\_score$$

- However, relevant and non-relevant objects are often not linearly separable in a space

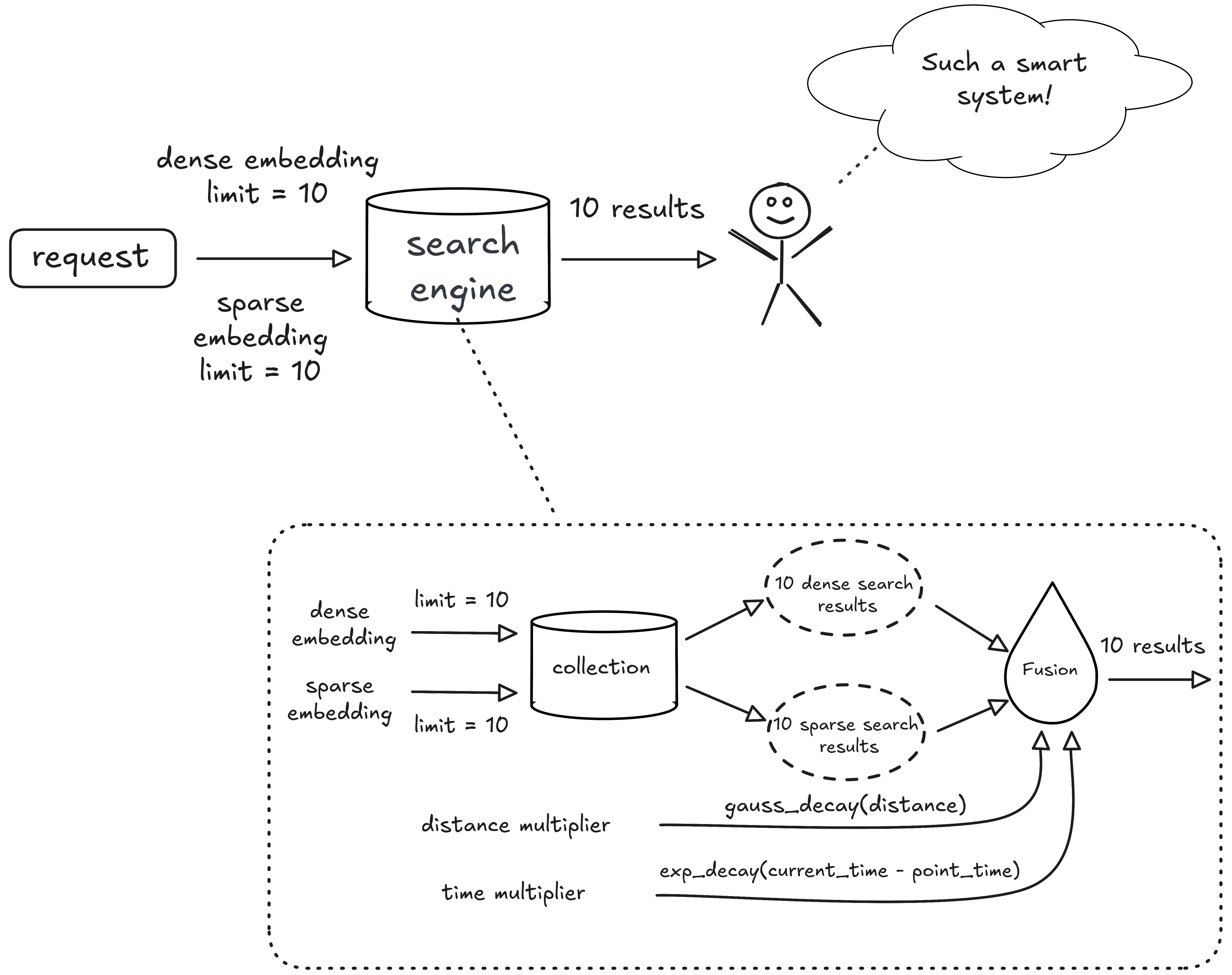


Hybrid search revamped - Building with Qdrant's Query API

Hybrid search

Arbitrary formula

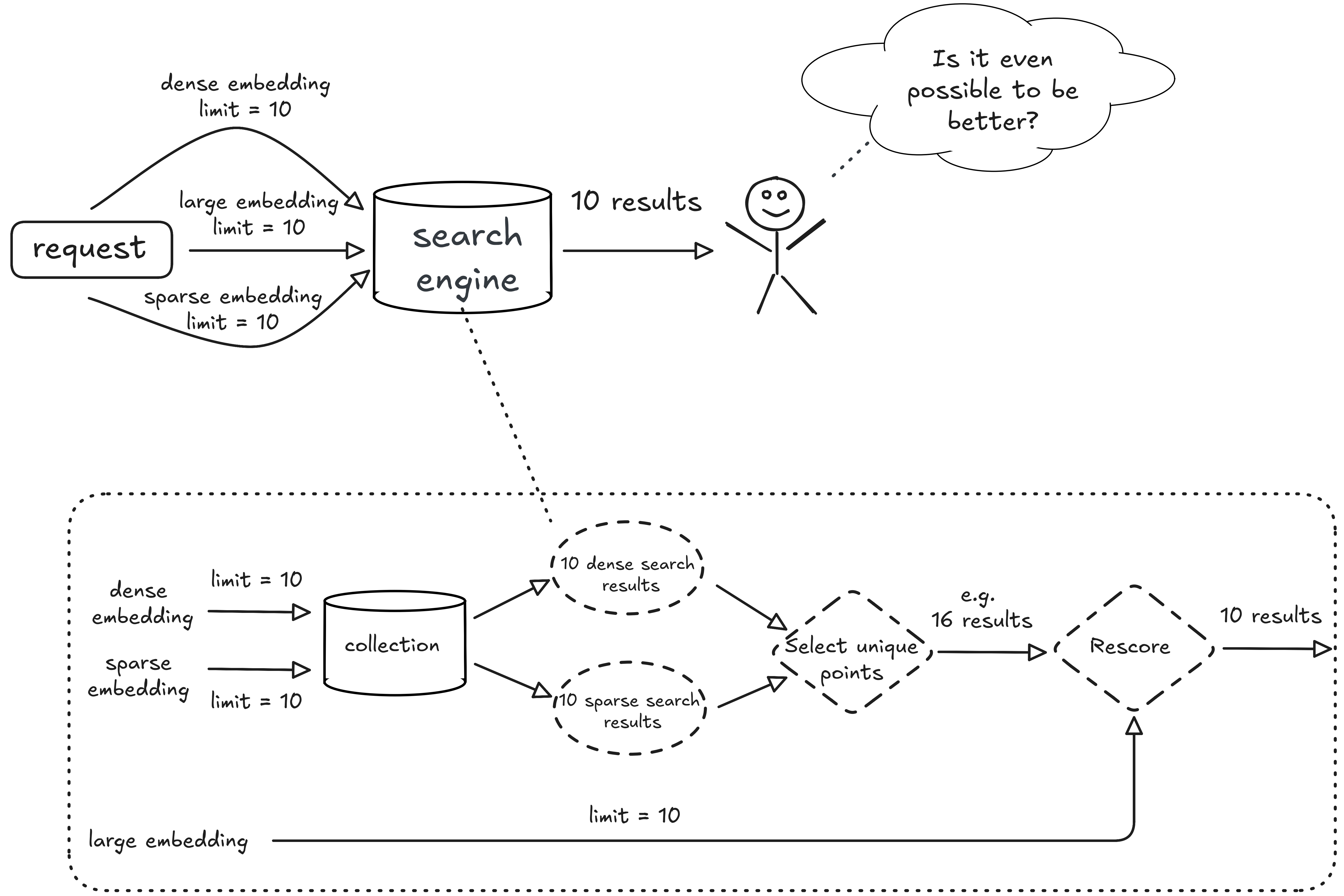
- It might not be the standard approach to build hybrid search, but no one can stop developers from shooting themselves into the foot
- If you want to make some experiments or somehow figured out the answer to life the universe and everything (including the exact formula you need to do the fusion), you can build on your own
- E.g. in Qdrant there is score boosting API, which allows to alter the scoring function and even include payload based multipliers into it (which are the original cause for introducing this API).



Hybrid search

Rescore with another model

- We've already seen how to rescore small model's results with a large model embeddings
- The same approach can be applied to several retrievers
- For that, just collect a pile of unique results from different retrievers, and search among them with the largest model embeddings

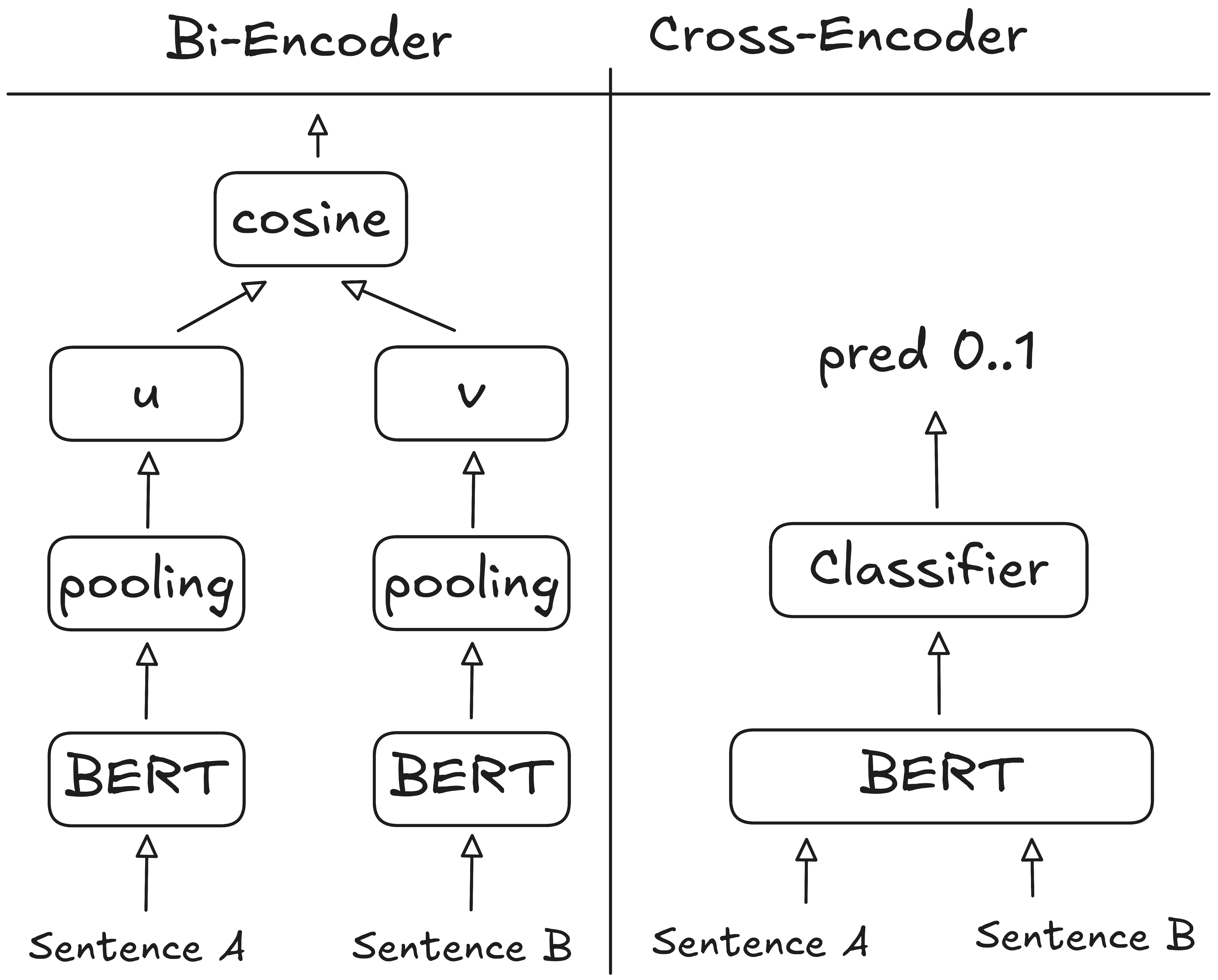


Rerank without embeddings

What are cross encoders?

Cross-encoder jointly encodes a query and a document in one pass, allowing full token-to-token interactions and producing a single relevance score.

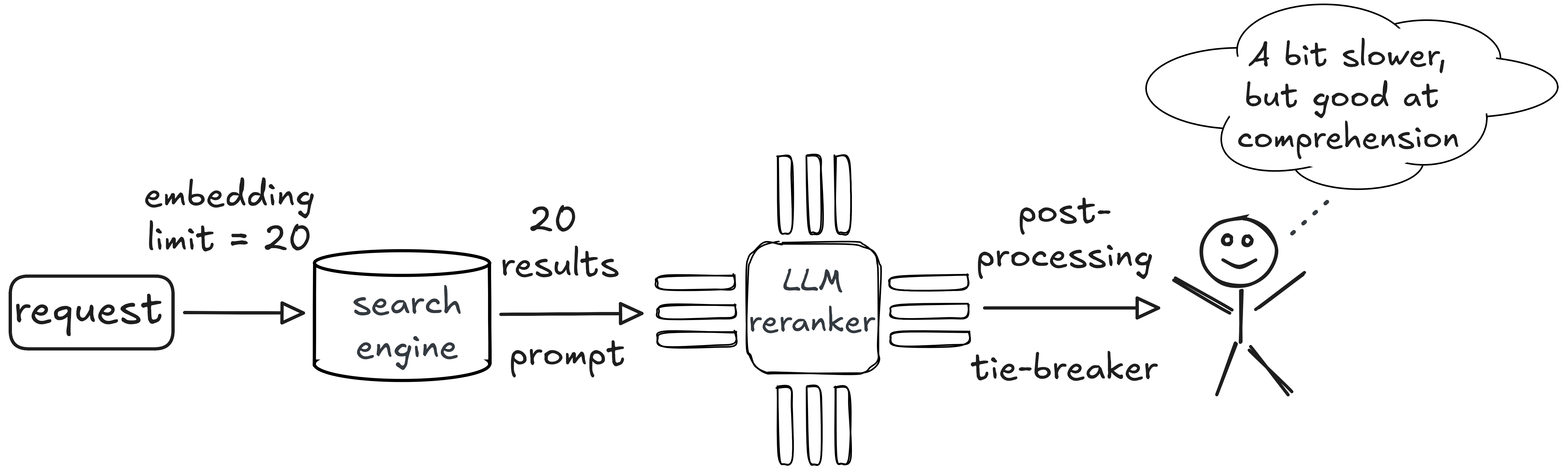
- Input: [CLS] query [SEP] doc [SEP]
- Full self-attention across all tokens
- [CLS] embedding captures combined Q-D interaction
- A small head produces final relevance score
- Can produce a noticeable increase in recall
- Used only for reranking due to high cost (one pass per document)
- Examples: BAAI/bge-reranker-v2-m3, mixedbread-ai/mxbai-rerank-large-v1



Rerank without embeddings

LLMs

- Cross-encoders might struggle with deep semantic reasoning and long-range dependencies - a gap LLM rerankers might address
- Though, LLMs require more thorough configuration
- It includes prompt tuning, as well as choosing the ranking strategy: whether it includes absolute scores or orders results by relevance; whether it compares all the pairs simultaneously or one-by-one
- Requires careful token handling and well-though-out infrastructure, otherwise might become too slow and costly
- No guarantees that it'll work better than a cross-encoder



How relevant each passage is?

pointwise [("id0", 6), ("id1", 10), ("id2", 5), ("id3", 10), ("id4", 4)]

Order passages by relevance

listwise "id1" > "id3" > "id0" > "id5" > "id2" > "id4"

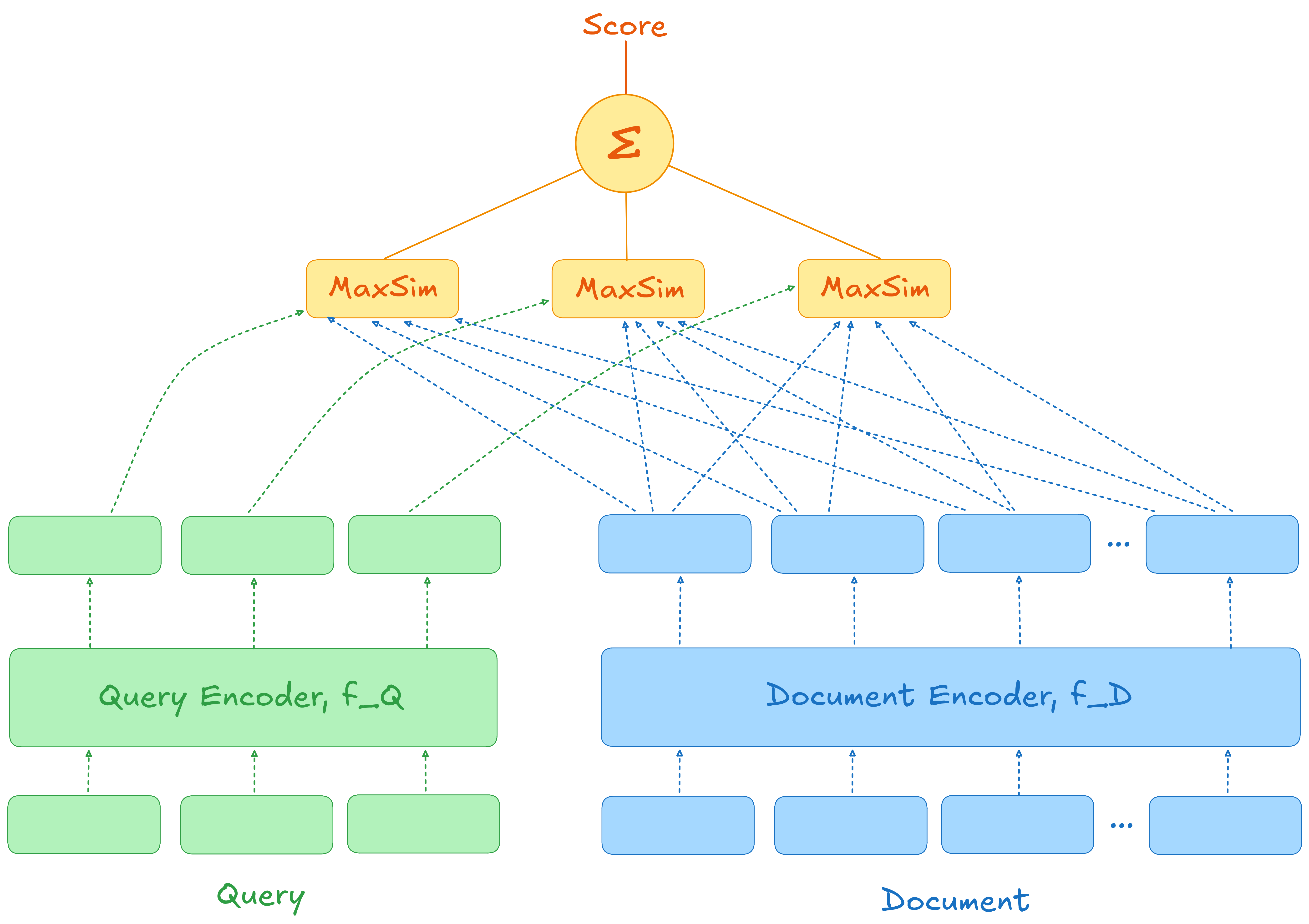
Order by answering: Which doc is more relevant to the query: "id1" or "id2"?

pairwise rel(doc_1, doc_2)

Late interaction embeddings

ColBERT

- Late interaction is a retrieval paradigm where queries and documents are encoded independently, but their token-level interactions are computed only at search time to recover fine-grained relevance signals.
- LI models fill in the gap between representation and full interaction models
- Instead of producing a single dense vector for the whole document, LI models generate a list of dense vectors, sometimes also called a multivector
- ColBERT is one of the late-interaction models: it computes relevance using *MaxSim*, matching each query token to its most similar document token
- The final relevance score is the sum of these *MaxSim* matches

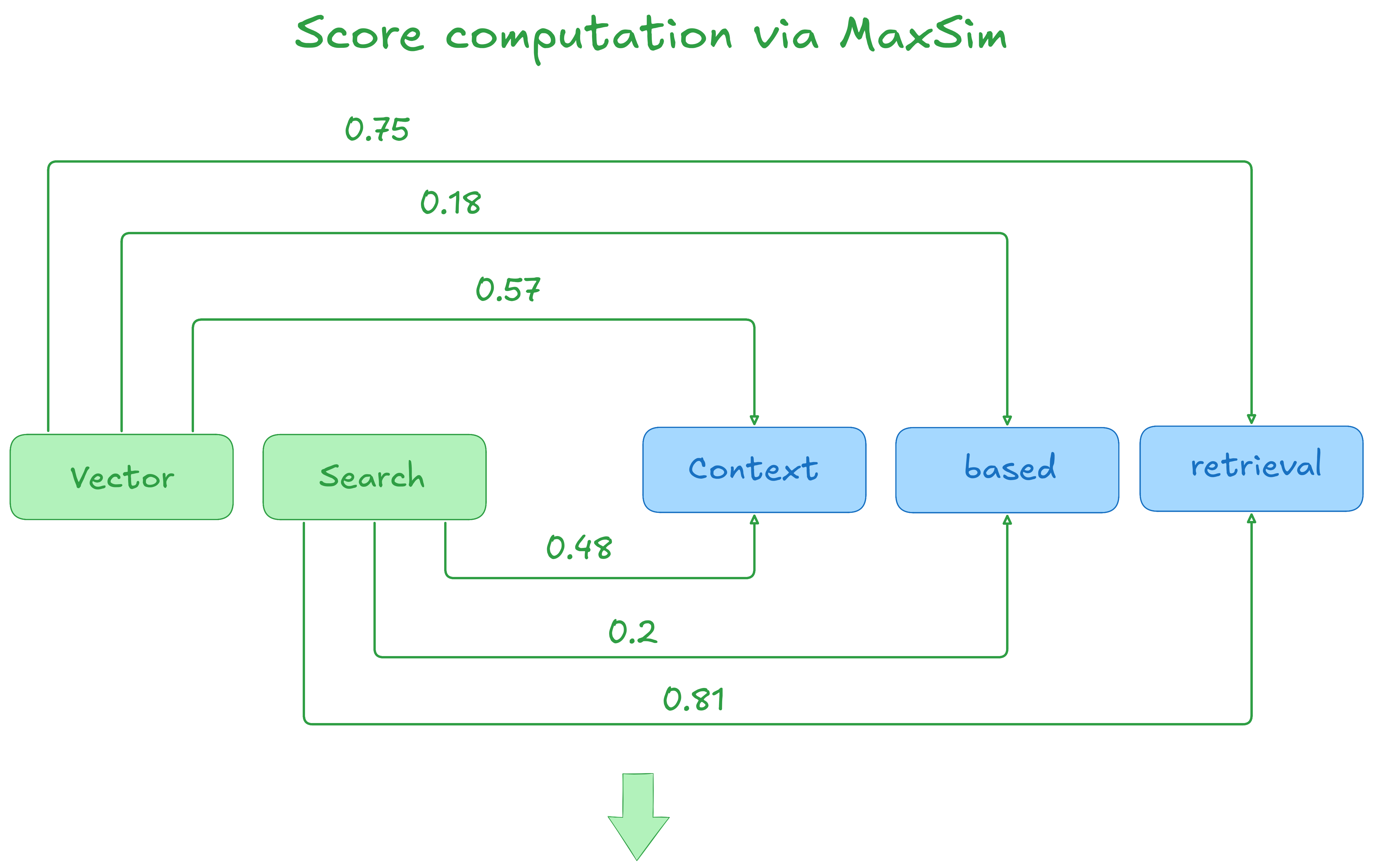


Late interaction embeddings

Colbert

- ColBERT produces a 128 dim vector for each of the tokens
- Might provide a better recall than representation based models, as well as be faster than full interaction models like cross-encoders
- Has custom preprocessing: appends special token [D] right after [CLS] and [Q] to queries. Queries are of the fixed length $N_Q = 32$, long queries are truncated, while short ones are augmented with [mask] tokens to re-weigh the existing terms based on their importance to matching the query.
- Though, authors recommend to use ColBERT for retrieval, index building might take too long, since of the number of required vector comparisons
- Usually used as second stage reranker

$S_{q,d} = \sum\limits_{i\in|E_q|} \max\limits_{j\in|E_D|} E_{q_i} \cdot E^T_{d_j}$, where $S_{q,d}$ is a score between query q and document d, E_q and E_d are lists of the corresponding token embeddings



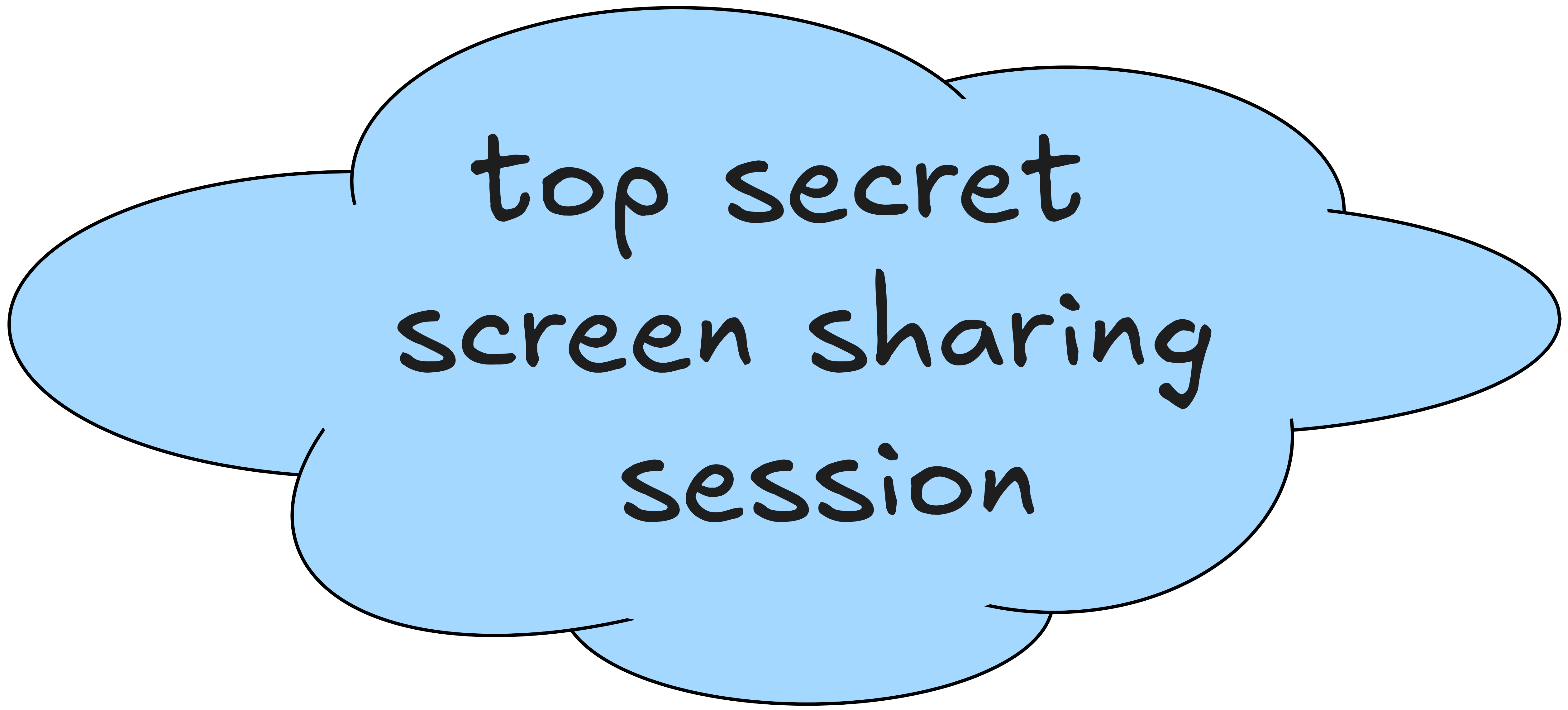
MaxSim("Vector") = max(0.75, 0.18, 0.57) = 0.75

MaxSim("Search") = max(0.48, 0.2, 0.81) = 0.81

$\sum$ = MaxSim("Vector") + MaxSim("Search") = 1.56

Demo in Qdrant

- It's time to demonstrate how the mentioned strategies can be implemented with



Questions?