Sparse embeddings

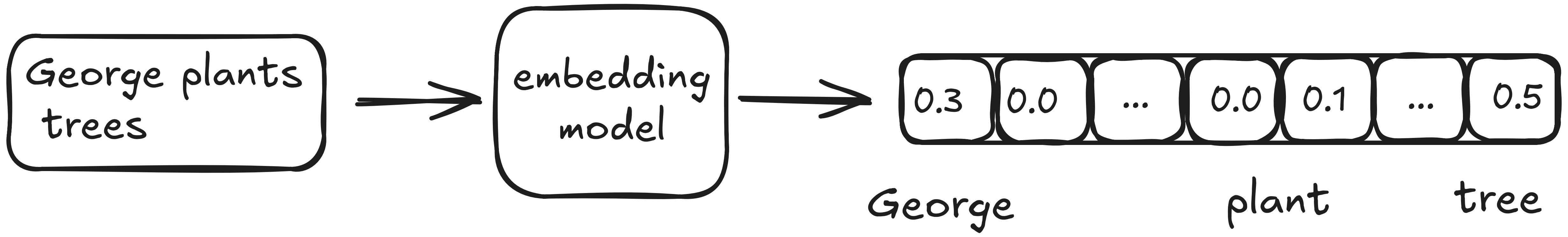
George Panchuk
HSE AI Fall 2025

github.com/joein/vector-search-course

Sparse embeddings

Sparse embeddings are high-dimensional vectors where only a small number of dims have non-zero values, and each non-zero dim typically (not always) corresponds to an explicit token or feature.

- Easy to interpret
- Good at exact match
- Complements dense vectors in hybrid search
- Requires tricks to understand semantics



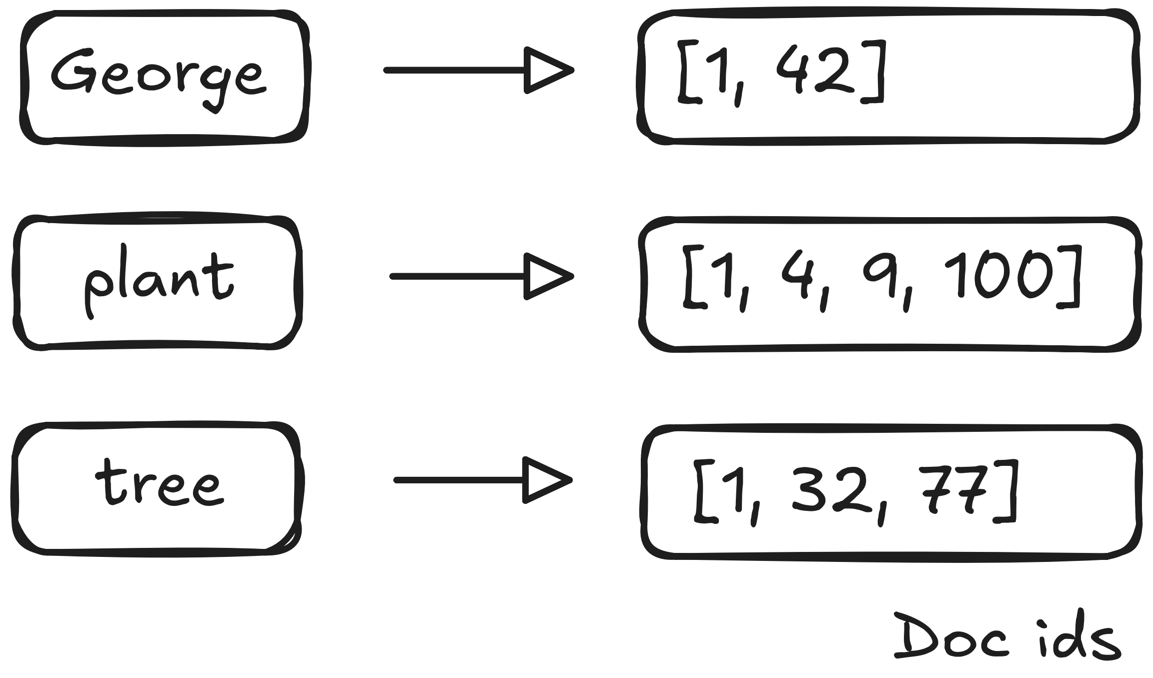
George plants trees → embedding model → 0.3 | 0.0 | ... | 0.0 | 0.1 | ... | 0.5

George

plant

tree

Typical structure for sparse search: Inverted index



George	[1, 42]
plant	[1, 4, 9, 100]
tree	[1, 32, 77]

Doc ids

Bm25

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

- 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

Dense vs Sparse (Bm25)

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

Dense vs Sparse (Bm25)

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

How to decouple bm25 computation

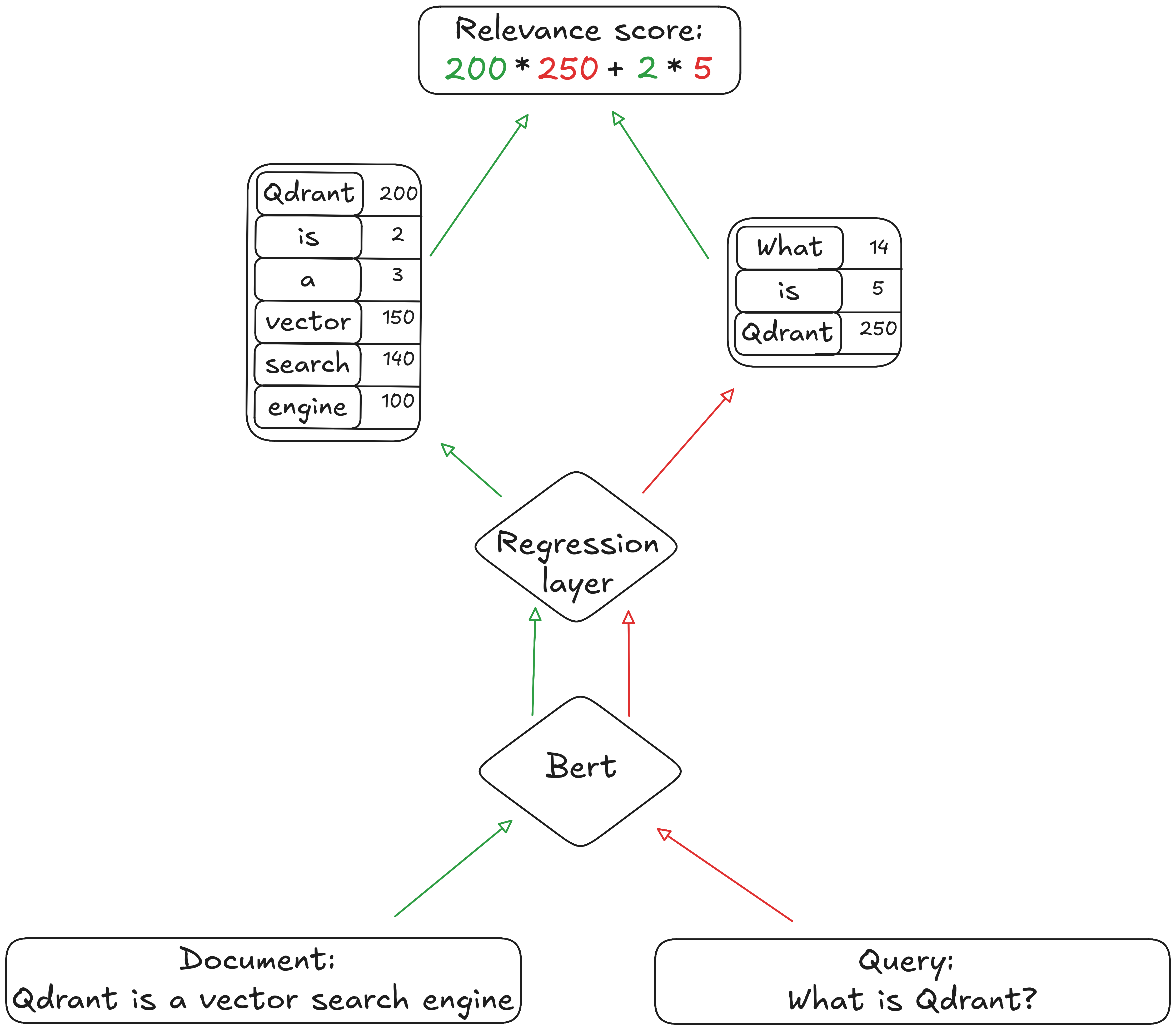
- Vector search engines are designed to work with many forms of embeddings, not just bm25
- The goal is not to replicate a full-text engine with a highly optimised BM25-centric pipeline, thus, it has to be integrated into the existing infrastructure
- The general API assumes that users send vectors to the engine as a part of the points

- *BM25* can be represented as $TF \cdot IDF$, where *TF* is term frequency, and *IDF* is inverse document frequency
- *IDF* component depends on global corpus statistics and has to be handled by the engine
- *TF* component can be computed independently for each document; only the average document length needs to be provided in advance

Bring the meaning

DeepCT

- BM25 like models lack semantic meaning capturing
- Idea: let's replace TF in BM25 with a predicted term importance value
- Train a simple regression layer to estimate each token's importance from its contextual BERT embedding
- Compute importance values for docs offline and store them in the inverted index, for queries - online
- Drawback: difficult to define reliable ground-truth scores for training



Relevance score:
200 * 250 + 2 * 5

Qdrant	200
is	2
a	3
vector	150
search	140
engine	100

What	14
is	5
Qdrant	250

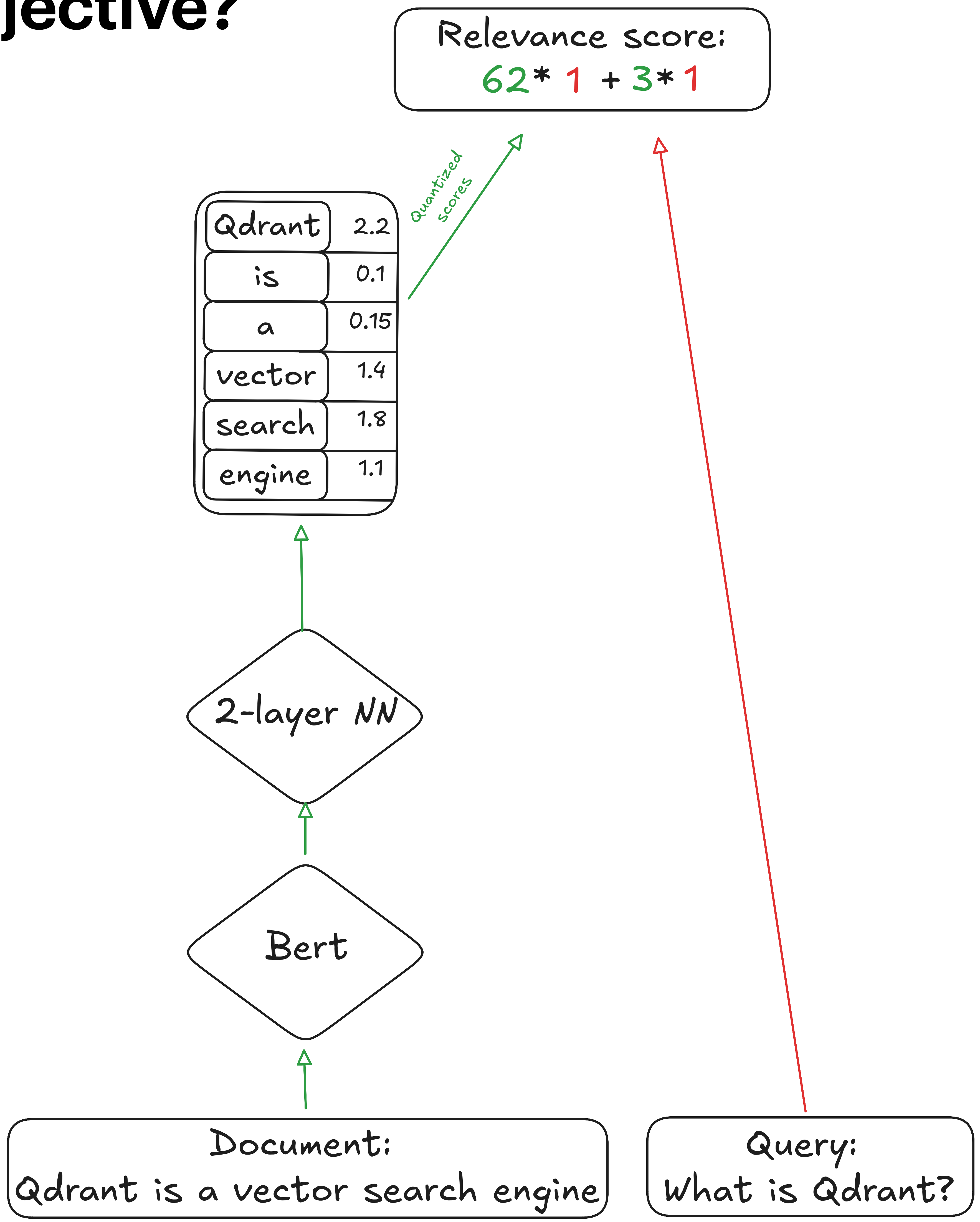
Document:
Qdrant is a vector search engine

Query:
What is Qdrant?

DeepImpact

What if we train on a relevance objective?

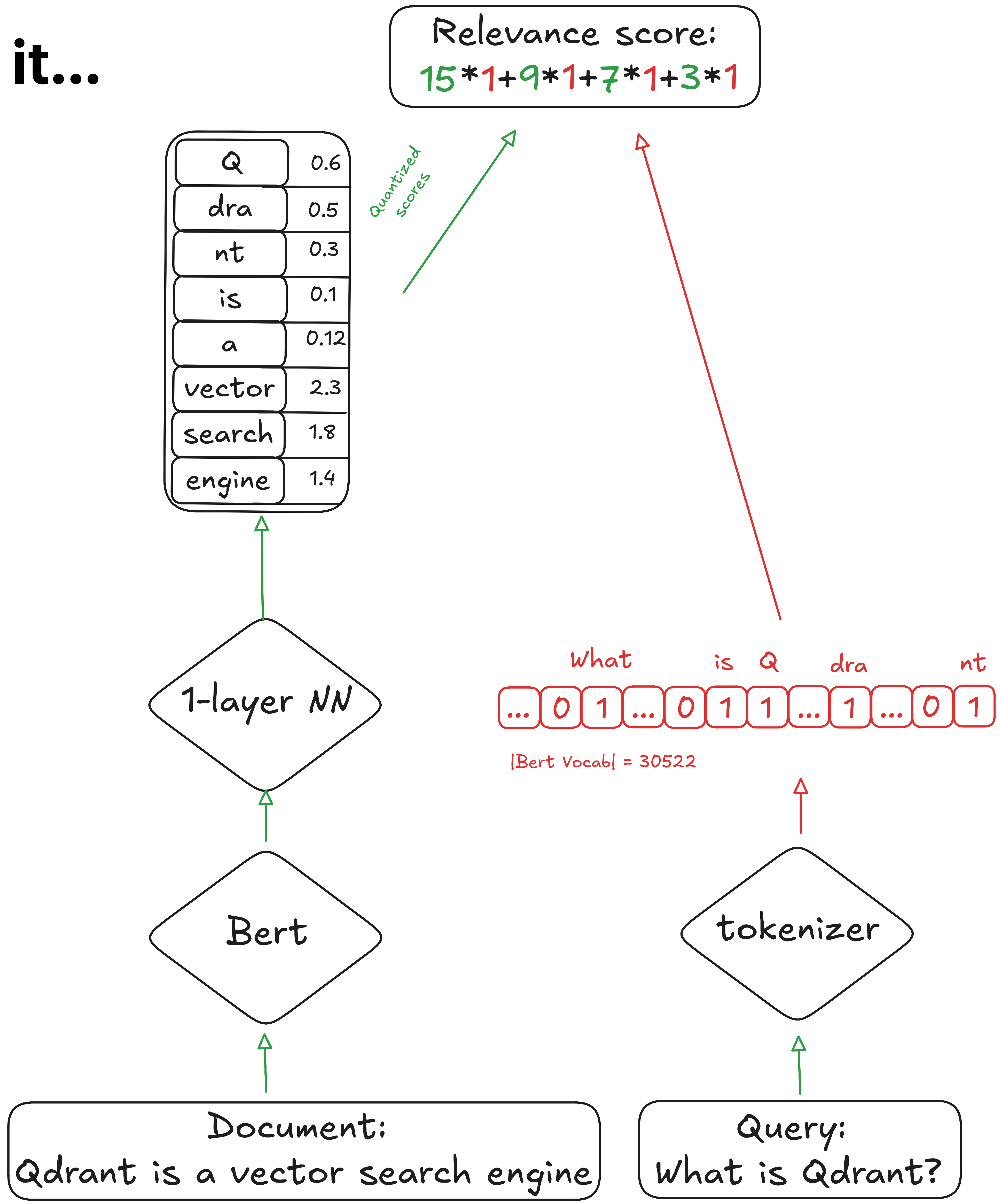
- DeepCT relies on mined per-term labels, which are difficult and noisy
- Idea: let's use whole doc-query relevance as the training signal
- Train a simple 2-layer network to produce a single scalar score for each unique document token; sum these scores up for each word overlapping with a query during retrieval
- Drawback: BERT computes embeddings for tokens, not words (e.g. Qdrant can be split into "Q", "#dra", "#nt"). DeepImpact (like DeepCT) takes the first token of the word and discards the rest.



TILDEv2

They said it should fix it...

- DeepImpact looses important information for rare words
- Idea: let's use DeepImpact architecture, but generate sparse encodings on a level of BERT's representations
- Drawback: a single scalar importance score value might not be enough to capture all distinct meanings of a word. *Homonyms* (spring: the season, a coil, to jump, a mountain spring) are one of the troublemakers in IR

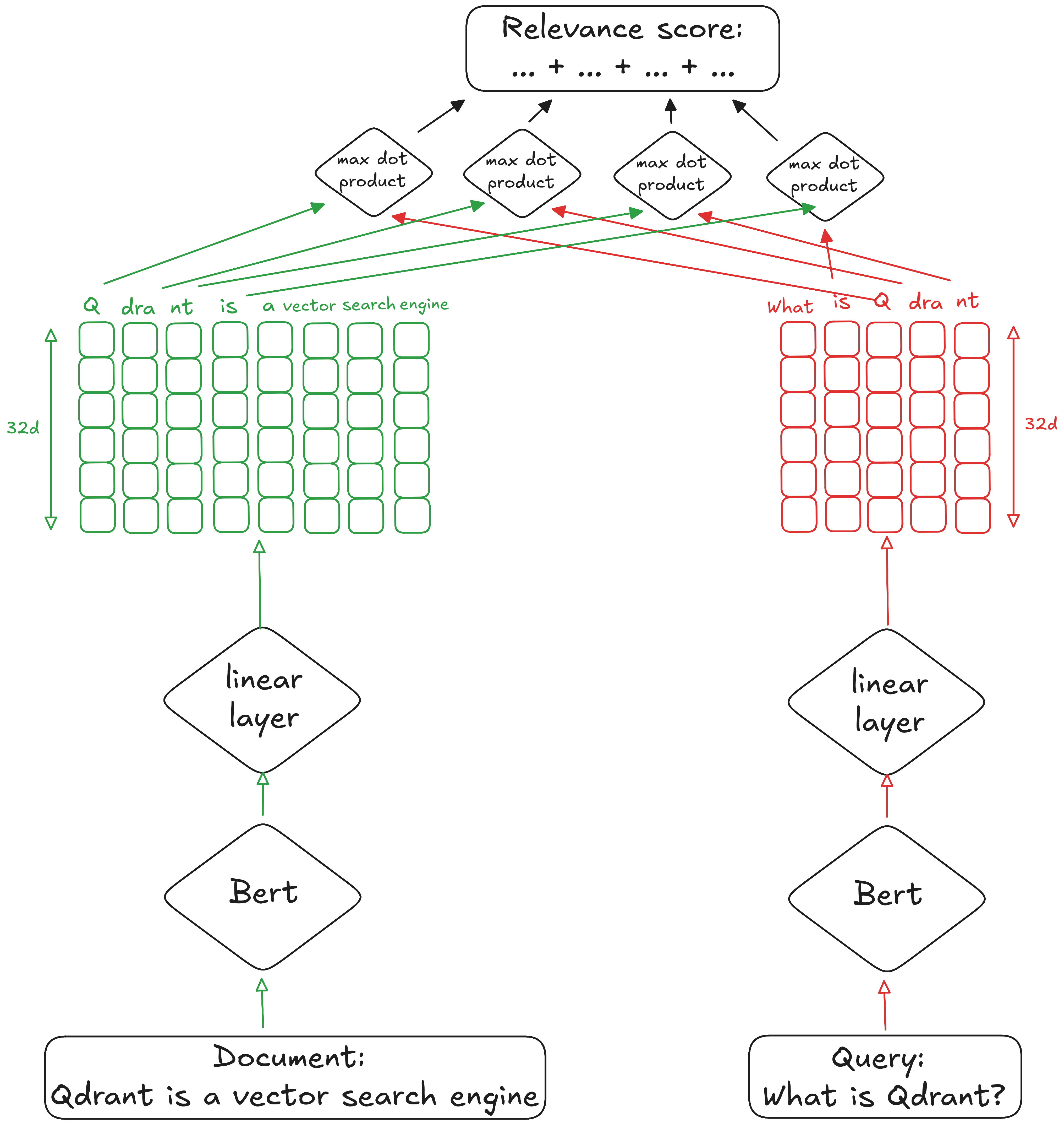


Q	0.6
dra	0.5
nt	0.3
is	0.1
a	0.12
vector	2.3
search	1.8
engine	1.4

COIL (COntextualised inverted list)

Not exactly sparse vectors with homonyms understanding

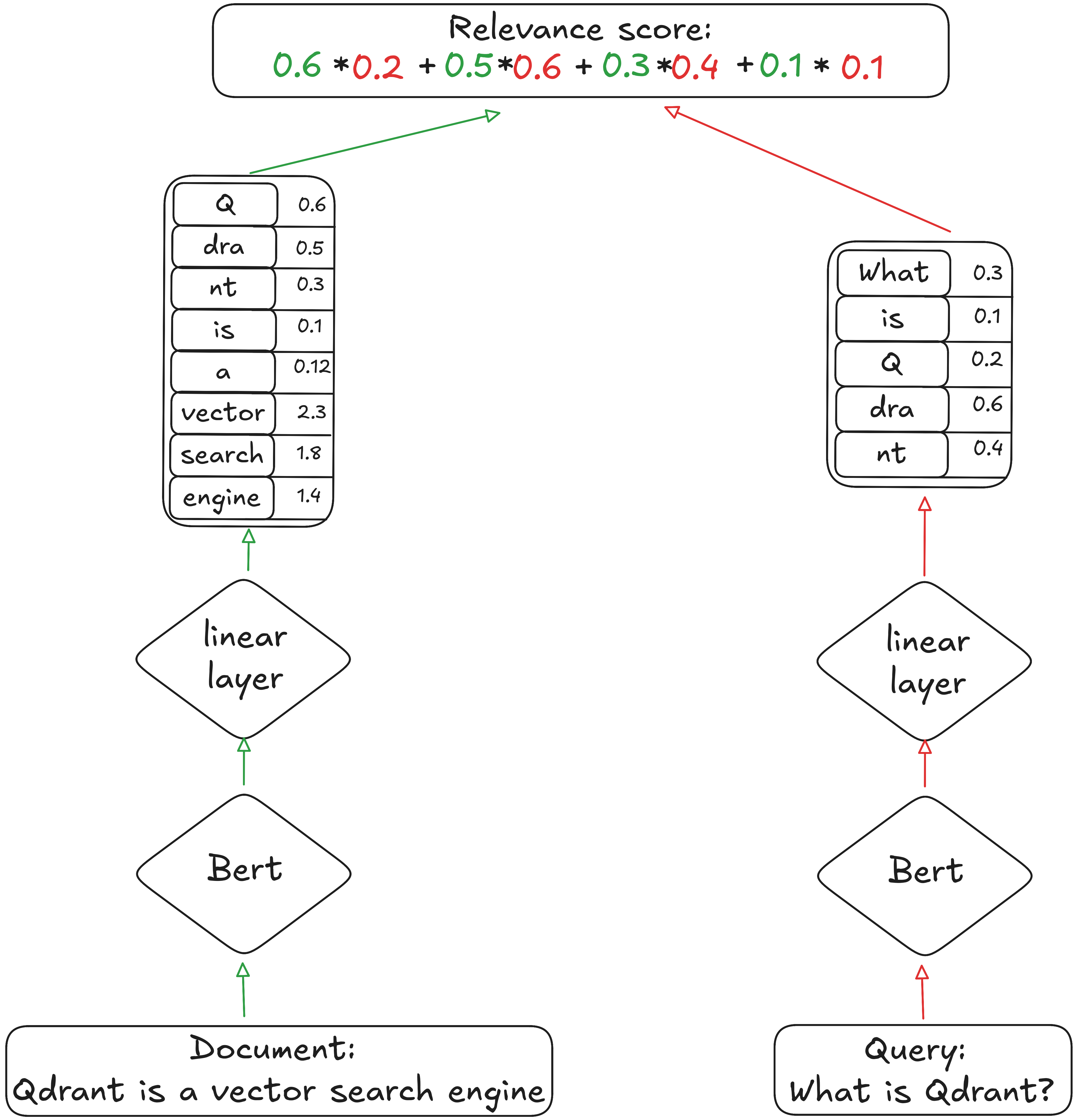
- Problem: a single scalar value is often insufficient to capture a term's contextual meaning
- Idea: store a small contextual vector (~32-d) for every token occurrence using BERT + projection; apply the same encoding to queries
- For each query token, compare its vector with all document vectors of the same token and take the maximum dot product as that token's contribution
- Drawback: 32dim vector for every term is far more expensive, and an inverted index does not work as-is with this architecture



UNICOIL

It has never happened and here we go again..

- Universal COntextualized Inverted List (UniCOIL), made by the authors of COIL as a follow-up, goes back to producing a scalar value as the importance score rather than a vector, leaving unchanged all other COIL design decisions.
- It optimizes resources consumption but the deep semantics understanding tied to COIL architecture is again lost.



Relevance score:
0.6 *0.2 + 0.5*0.6 + 0.3*0.4 + 0.1 * 0.1

Q	0.6
dra	0.5
nt	0.3
is	0.1
a	0.12
vector	2.3
search	1.8
engine	1.4

What	0.3
is	0.1
Q	0.2
dra	0.6
nt	0.4

Document:
Qdrant is a vector search engine

Query:
What is Qdrant?

Vocabulary mismatch

Query expansion

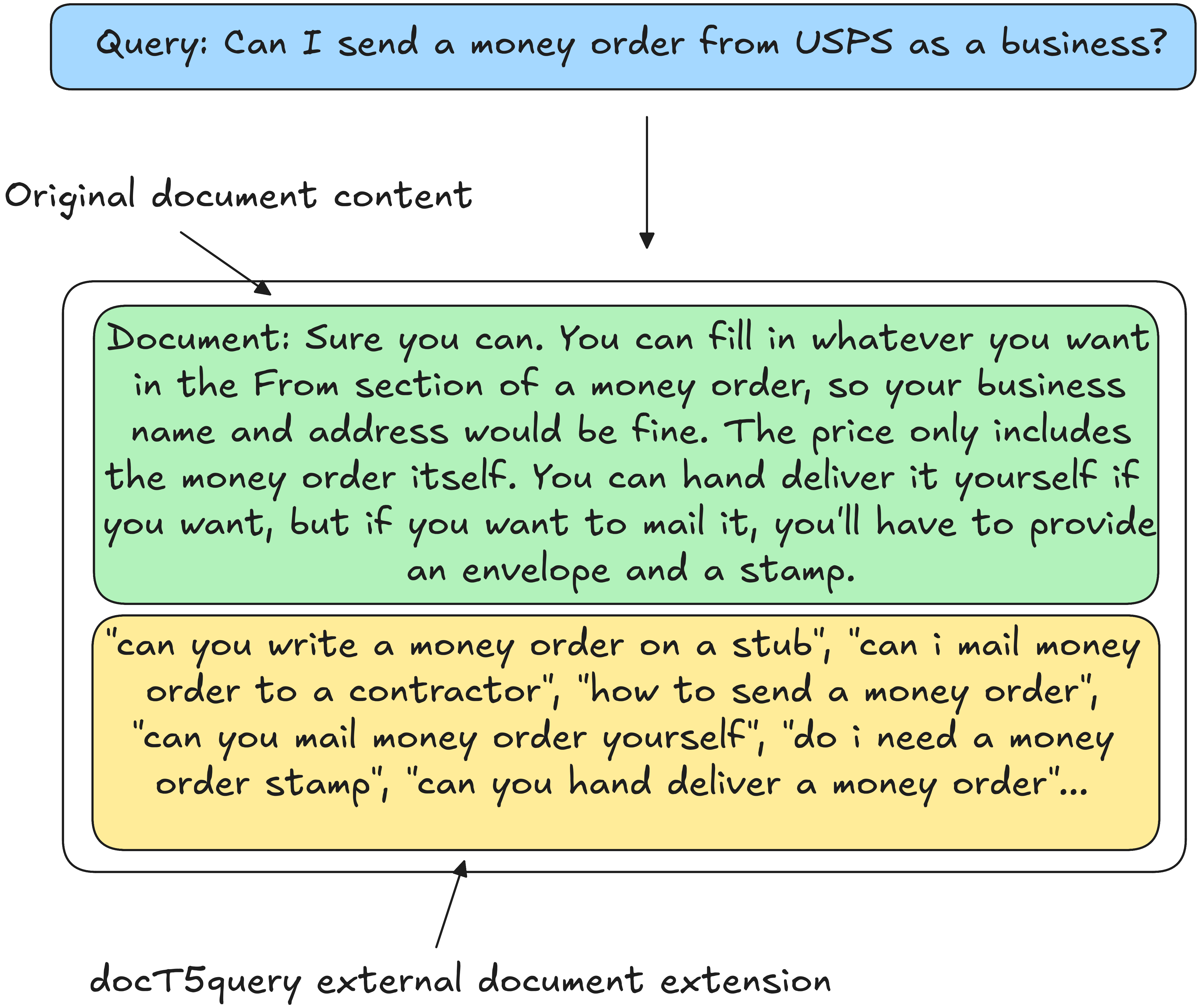
- Exact-match retrieval fails whenever relevant documents lack the query terms
- Term importance modelling (DeepCT, DeepImpact, COIL) does not solve vocabulary mismatch, it cannot create new lexical matches
- Document expansion: append likely query words to the document
- Document expansion can be external (a separate model generates extra terms) or internal (the retriever itself produces expanded term weights).

External document expansion

Doc-t5

docT5query: the standard external expansion model

- T5 model generates top-k possible queries for which the document could be an answer
- Generated queries are appended to the document
- Repetitions naturally increase *TF*
- Drawback: generative decoding is slow, expensive at scale and might not be suitable for the environments requiring fast responses



Query: Can I send a money order from USPS as a business?

Original document content

Document: Sure you can. You can fill in whatever you want in the From section of a money order, so your business name and address would be fine. The price only includes the money order itself. You can hand deliver it yourself if you want, but if you want to mail it, you'll have to provide an envelope and a stamp.

"can you write a money order on a stub", "can i mail money order to a contractor", "how to send a money order", "can you mail money order yourself", "do i need a money order stamp", "can you hand deliver a money order"...

docT5query external document extension

External document expansion

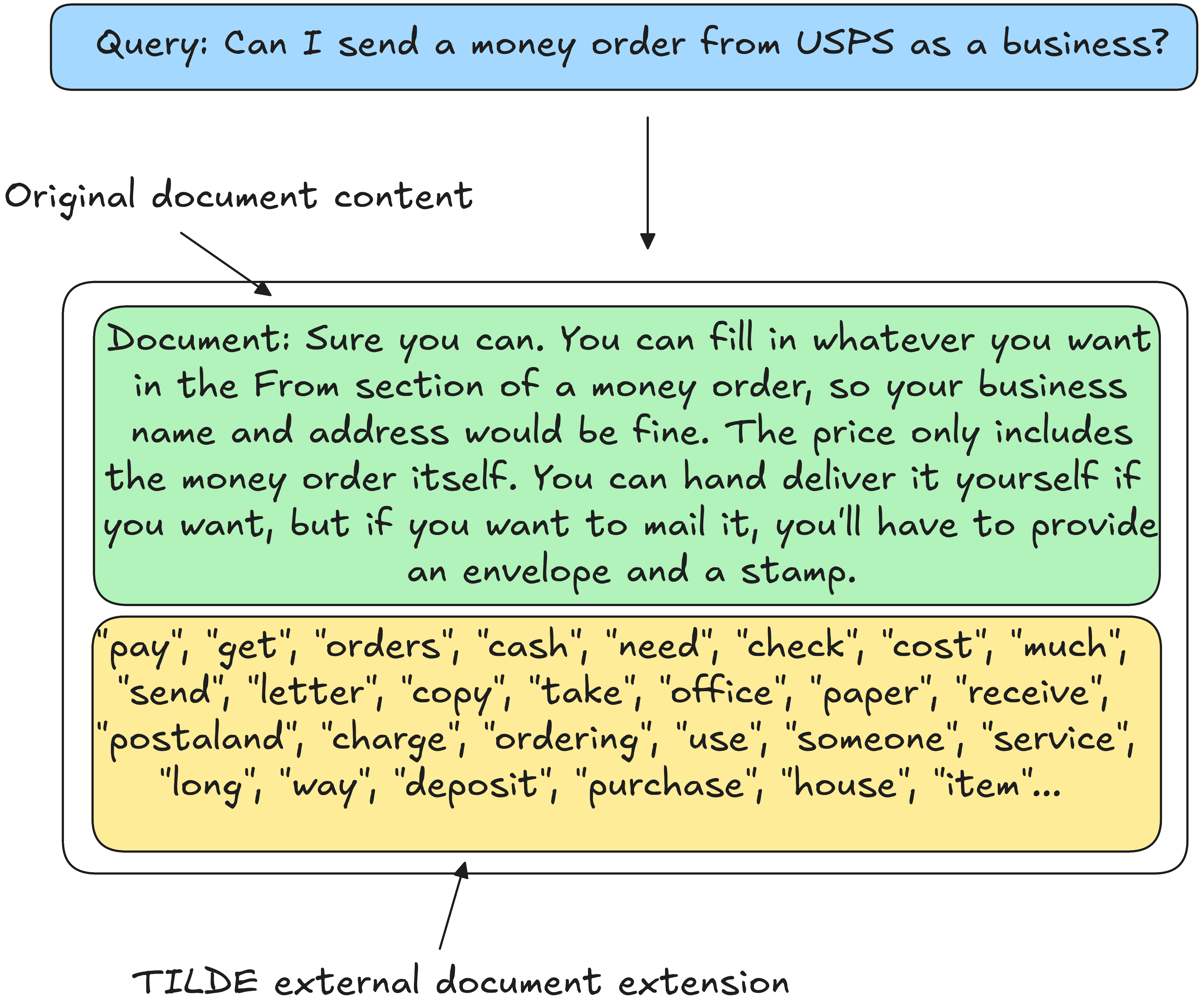
TILDE

TILDE: fast external expansion

- Predicts the most likely terms given the document
- Assumes terms are independent (allows full parallelisation)
- Cuts expansion time by ~98% compared to docT5query
- Appends top-k most likely terms without repetition

Limitations of external expansion:

- Requires extra compute before indexing
- Not always feasible for large collections or real-time ingestion
- Expansion and retrieval live in separate models



Query: Can I send a money order from USPS as a business?

Original document content

Document: Sure you can. You can fill in whatever you want in the From section of a money order, so your business name and address would be fine. The price only includes the money order itself. You can hand deliver it yourself if you want, but if you want to mail it, you'll have to provide an envelope and a stamp.

"pay", "get", "orders", "cash", "need", "check", "cost", "much", "send", "letter", "copy", "take", "office", "paper", "receive", "postaland", "charge", "ordering", "use", "someone", "service", "long", "way", "deposit", "purchase", "house", "item"...

TILDE external document extension

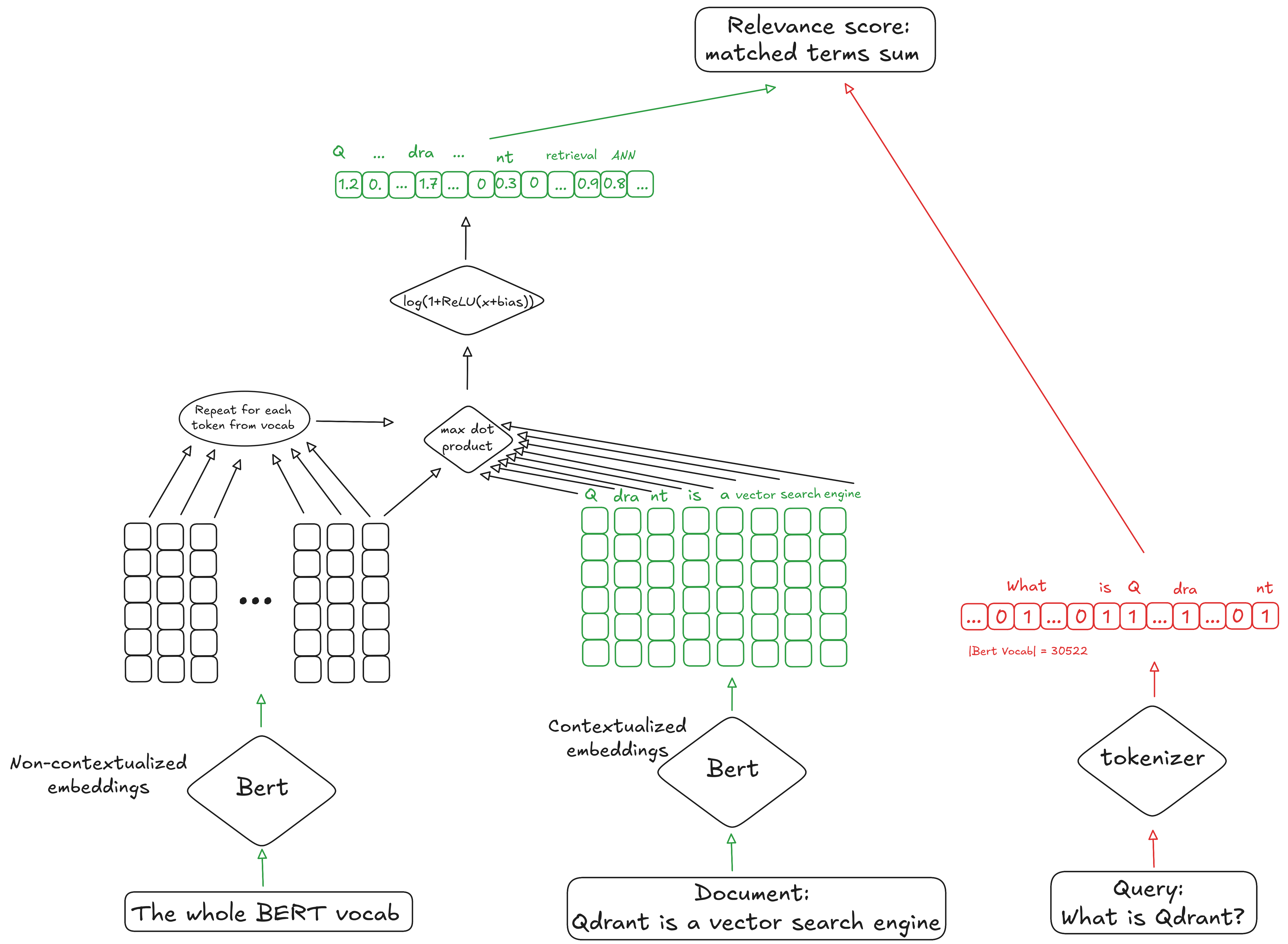
Internal document expansion

- Goal: combine expansion and retrieval in a single model
- Idea: treat every vocabulary term as a possible query term the document might match (Each contextualized token predicts how strongly it supports each vocabulary word.)
- Build a single document-wide vocabulary vector (For each vocabulary word, take the highest score predicted by any token in the document. Non-zero coordinates mark query terms for which the document should be retrievable
- For example, by pre-computing vectors for the document "*vector search*" on a vocabulary of 50,000 most used English words, for this small document of two words, we will get a 50,000-dimensional vector of zeros, where non-zero values will be for a "*vector*", "*embedding*", "*retrieval*", "*ANN*", "*HNSW*", "*search*", "*semantic*" and "*index*".

Sparse Transformer Matching (SPARTA)

Using internal term expansion

- Problem: Exact-match retrieval fails whenever relevant documents lack the query terms
- Idea: let's use internal term expansion with a learned threshold of non-zero effect
- For each token in BERT vocabulary find the maximum dot product between it and contextualised tokens in a document and learn a threshold of a considerable (non-zero) effect.
- Retrieval: just sum up all scores of query tokens in the document
- Drawback: as many other sparse retrievers, it was trained on the MS MARCO, it show good results on it, but struggling when it comes to generalisation and could perform worse than BM25



Relevance score: matched terms sum
Q ... dra ... nt retrieval ANN
1.2 0. ... 1.7 ... 0 0.3 0 ... 0.9 0.8 ...
log(1+ReLU(x+bias))
Repeat for each token from vocab
max dot product
Q dra nt is a vector search engine
What is Q dra nt
... 0 1 ... 0 1 1 ... 1 ... 0 1
|Bert Vocab| = 30522
Non-contextualized embeddings
Bert
Contextualized embeddings
Bert
tokenizer
The whole BERT vocab
Document: Qdrant is a vector search engine
Query: What is Qdrant?

Sparse Lexical and Expansion model (SPLADE)

- SPARTA model is not sparse enough by construction and producing too many non-zero values
- SPARTA uses BERT as is, and does not care about IR problems
- Idea: produce less non-zero values :) tune for IR, learn from a bigger model, expand not only documents, but also queries
- SPLADE introduces sparsity regularisation, to enable highlight less terms, apply IR-related tricks during training, leverages knowledge distillation and learns from a bigger model
- Among the drawbacks of SPLADE is that it might be expensive and slow, as well as it might generalise not that good for new domains

Here could be a picture of how SPLADE works but it's too large to fit nicely into the slide

Our attempts to make sparse retrieval better

- Bm42 - BM25 might struggle when working with applications working on small chunks of data, like RAG, since *TF* might not saturate well. SPLADE might be too slow and struggle with generalisation and on unseen data. Use attention weights from any underlying transformer model instead of TF.
- miniCOIL - COIL made a good attempt representing words as small vectors to handle homonyms, however, not small enough. miniCOIL pushes it further and compresses these small vectors to a smaller dimension (e.g. 4).
  Other than that, it does not rely on some relevance objective and dependence on labelled datasets, instead it makes use of the existing retrievers.

Questions?