University of Glasgow | Università di Pisa

CIKM 2021 1-5 NOVEMBER

IR From Bag-of-words to BERT and Beyond through Practical Experiments

A CIKM 2021 tutorial with PyTerrier and OpenNIR

Sean MacAvaney*
Craig Macdonald*
Nicola Tonellotto*

*(*Alphabetical ordering)*

PART 4
LEARNED SPARSE RETRIEVAL & DENSE RETRIEVAL

Introduction

- Re-rankers like BERT are great for **improving the precision** in IR tasks.
- But re-rankers are limited by the recall of the first-stage retrieval model:
  - Re-rankers cannot improve the **recall** below re-ranking threshold
  - Re-rankers tend to be expensive to run: if initial retrieval models had higher precision, there would be less to re-rank
- A strong enough first stage wouldn't need re-ranking!
- **Dense retrieval** can improve first-stage retrieval recall
- Alternatively, **leaned sparse retrieval** can improve both recall and precision

Intended Learning Outcomes of Part 4

ILO 4A. Understand the most recent effective retrieval architectures that learn representations of documents, both in leveraging traditional index structures (sparse retrieval), as well as new similarity search systems (dense retrieval)

ILO 4B. Experience DeepImpact sparse retrieval approaches, as well as ANCE and ColBERT dense retrieval approaches

Outline of Part 4

Università di Pisa | University of Glasgow

Part 4A: Learned Sparse Retrieval

Part 4B: Dense Retrieval

Part 4C: Nearest Neighbour Search

University of Glasgow | Università di Pisa

CIKM 2021 1-5 NOVEMBER

Learned Sparse Retrieval

PART 4A

Learned Sparse Retrieval

Università di Pisa | University of Glasgow

Main question: Can we leverage the neural IR effectiveness improvements together with the sparse retrieval efficiency query processing?

- In sparse retrieval, documents are stored in an inverted index data structure
- The inverted index is highly specialized toward efficient query processing
- Major data structure in search engines
- In neural IR, semantic models outperform lexical models in ad-hoc search tasks
- Storing semantic information is expensive
- Processing semantic information is expensive

DeepImpact

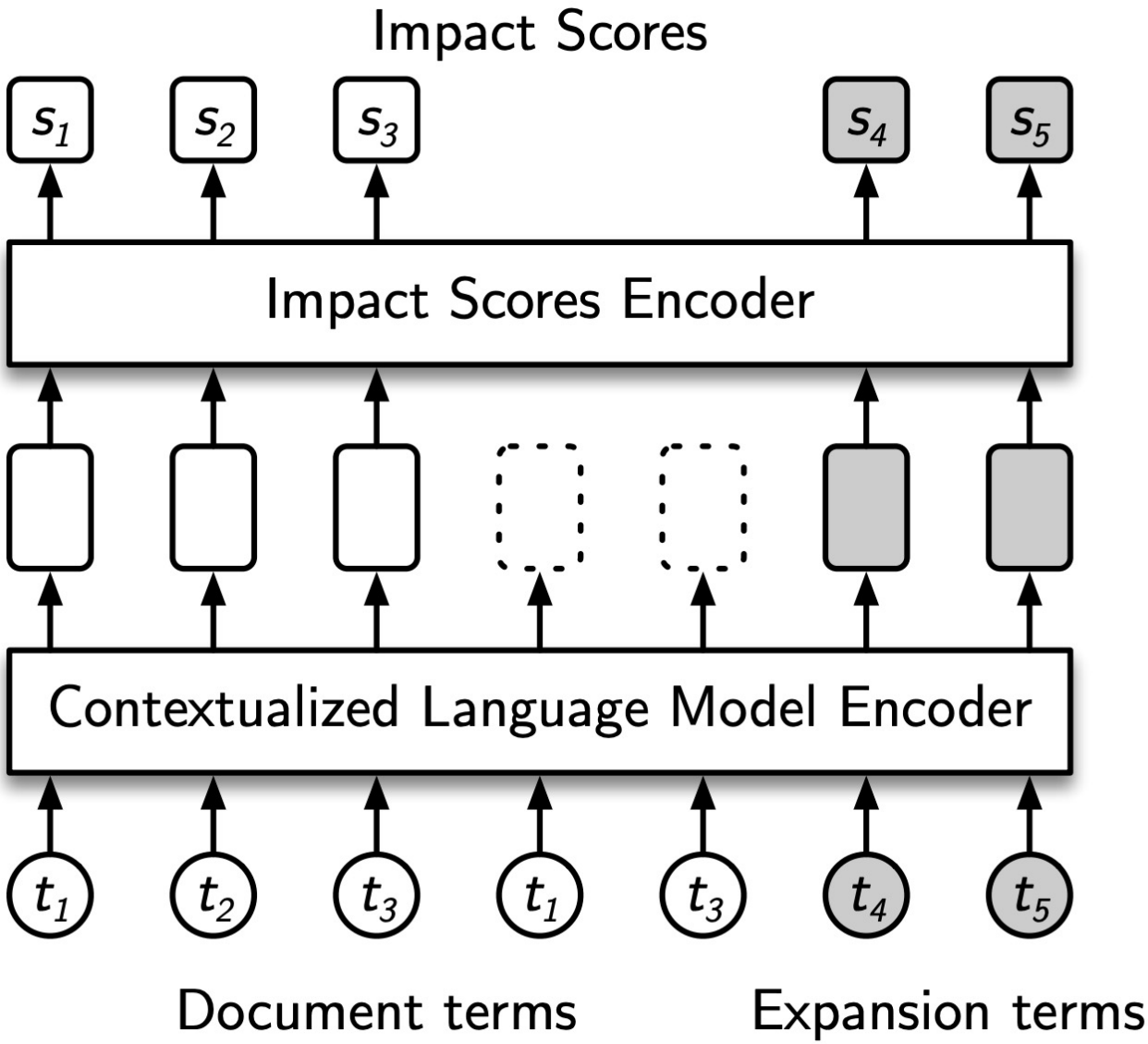
Università di Pisa | University of Glasgow

Main idea: Compute a quantized score (impact) to store in posting list

- Add text to documents using a contextualized language model
- Address the vocabulary mismatch problem
- Exploit BERT to compute a single score for each query-document pair
- Quantize linearly the scores into positive integers
- Store quantized scores as frequencies in postings, using simple sum as weighting model

Mallia, Khattab, Tonellotto, Suel. Learning Passage Impacts for Inverted Indexes. SIGIR 2021

DeepImpact



Mallia, Khattab, Tonellotto, Suel. Learning Passage Impacts for Inverted Indexes. SIGIR 2021

DeepImpact practical

- Sparse Indexing – Learned

```
vaswani = pt.get_dataset("vaswani")

pt_index_path = './terrier_di_vaswani'

parent = pt.index.IterDictIndexer(pt_index_path)
parent.setProperty("termpipelines", "")

indexer = DeepImpactIndexer(parent, batch_size=32)
indexer.index(vaswani.get_corpus_iter())

index_ref = pt.IndexRef.of(pt_index_path + "/data.properties")
index_di = pt.IndexFactory.of(index_ref)
```

```
pt_index_path = './terrier_vaswani'

indexer = pt.index.IterDictIndexer(pt_index_path)
indexer.setProperty("termpipelines", "")
index_ref = indexer.index(vaswani.get_corpus_iter())

index_ref = pt.IndexRef.of(pt_index_path + "/data.properties")
index = pt.IndexFactory.of(index_ref)
```

- End-to-end Retrieval

```
pt.Experiment([
        pt.BatchRetrieve(index, wmodel="BM25"),
        pt.BatchRetrieve(index_di, wmodel="Tf")
    ],
    vaswani.get_topics(), vaswani.get_qrels(),
    names=['bm25', "deep_impact"],
    eval_metrics=["map", "recip_rank", "ndcg_cut_10"]
)
```

	name	map	recip_rank	ndcg_cut_10
0	bm25	0.143471	0.516597	0.249405
1	deep_impact	0.186312	0.661398	0.332272

DeepImpact Documents

	Original Term	Term Frequency
0	of	1
1	the	2
2	in	1
3	origin	1
4	atmosphere	1
5	upper	1
6	nitrogen	1
7	ionization	1

	DeepImpact Term	DeepImpact Value
0	of	36
1	the	46
2	a	36
3	in	43
4	and	40
5	to	24
6	is	36
7	described	40
8	are	32
9	design	42
10	suitable	69
11	discussed	60
12	characteristics	22
13	probe	143
14	positive	103
15	small	43
16	such	42
17	experiments	77
18	density	102
19	measurement	81
20	ionosphere	182

Other Approaches

COIL (Gao, Dai, Callan, NAACL 2021)

- each document token is project into d = 32 dimensions
- all embeddings are directly stored in the inverted index (instead of frequencies)
- at inference time, query weights are computed

UniCOIL (Lin & Ma, Arxiv Jun 2021)

- corresponds to COIL with d = 1
- additional ReLU layer to deal with negative scores
- scores quantised on 8 bits

Sparta (Zhao, Lu, Lee, ACL 2021):

- focuses on Q&A tasks
- generates scores as DeepImpact

Splade (Formal, Piwowarski, Clinchant, SIGIR 2021)

- sparsification of embeddings with ReLU at training time on a token basis
- FLOPS regularisation on queries and documents included in training loss
- score aggregation over query terms with sum (v1) or max (v2)
- knowledge distillation: train a student model based on margin MSE

Dense Retrieval

PART 4B

Dense Retrieval

- Dense Retrieval, particularly for passages is of great interest
  - At indexing time, represent passages by one of more embeddings
  - At retrieval time, encode the query using the same model, and identify passages with embedding(s) **similar** to the query
- Great improvements have been proposed in large-scale **similarity search systems**
  - **E.g. FAISS** is the Facebook large scale nearest neighbour (NN) search system
  - FAISS supports for **exact** and **approximate search**
  - The type of search depends on the **problem dimension**
- In the following, we review the main dense retrieval models, then dense retrieval NN search

Sparse vs Dense Retrieval (I)

- **Sparse Retrieval**
  - Every document is represented by a single score for each term in the lexicon V (equal to 0 for terms not appearing in the document)
  - A document is a $|V|$-dimensional vector with many 0s, i.e., sparse
  - A query-document relevance score is computed as the sum of the score of the query terms in the documents
  - Only query terms appearing in the document contributes to the final score
  - A query is then a $|V|$-dimensional vector with almost all entries set to 0, only a few set to 1
- **Pros**
  - Efficient storage in inverted indexes
  - Easily scalable
  - 40+ years of query processing optimizations based on the inverted index
  - Widely adopted as first stage ranker in cascading architectures
- **Cons**
  - Very few relevance signals
  - Not considering semantics information (expect learned sparse models)
  - Limited support for proximity signals

Sparse vs Dense Retrieval (II)

- **Dense Retrieval**
  - Every document and query are represented by a *D*-dimension vector(s) of scores with few 0s, i.e., dense (embeddings)
  - The query-document relevance score is computed as the L2 or cosine similarity between the query-document vectors
- **Pros**
  - Efficient storage in similarity indexes
  - 20+ years of query processing optimizations based on the nearest neighbour search
  - Rich set of relevance signals
  - Consider semantics
- **Cons**
  - Missing a clear interpretation of the embedding space
  - Missing a clear interpretation of the captured semantics

Sparse vs Dense Representations

- **Sparse Representations**
  - Bag of words
    - [0, 0, 1, 1, 0, 0, …, 1, 0, …, 0, 0]
    - [0, 0, a, b, 0, 0, …, c, 0, …, 0, 0]
  - Issues
    - Lexical GAP: USA vs United States
    - Ambiguity: jaguar (animal) vs jaguar (car brand)
    - Ordering: “from whom?” vs “whom from?”
- **Dense Representations**
  - Learned function:
    - f(doc) → embedding (in R^n)
  - Semantic similarity ≃
    - $doc_1 \simeq doc_2 \rightarrow f(doc_1) \simeq f(doc_2)$

Learning Representations

UNIVERSITÀ DI PISA | University of Glasgow

- **Pairs** or **triples** of data per input sample
- Get an **embedding** for each pair or triple input samples
- Define a **metric** function to measure the **similarity between embeddings**
  - Cosine similarity
  - Dot product
- Train the model to produce **close embeddings** for the two inputs, in case the inputs **are similar**, or **distant embeddings** for the two inputs, in case they **are dissimilar**
- **Ranking losses** predict relative distances between input samples
  - Not predicting directly a label or a value

Examples of Ranking Losses

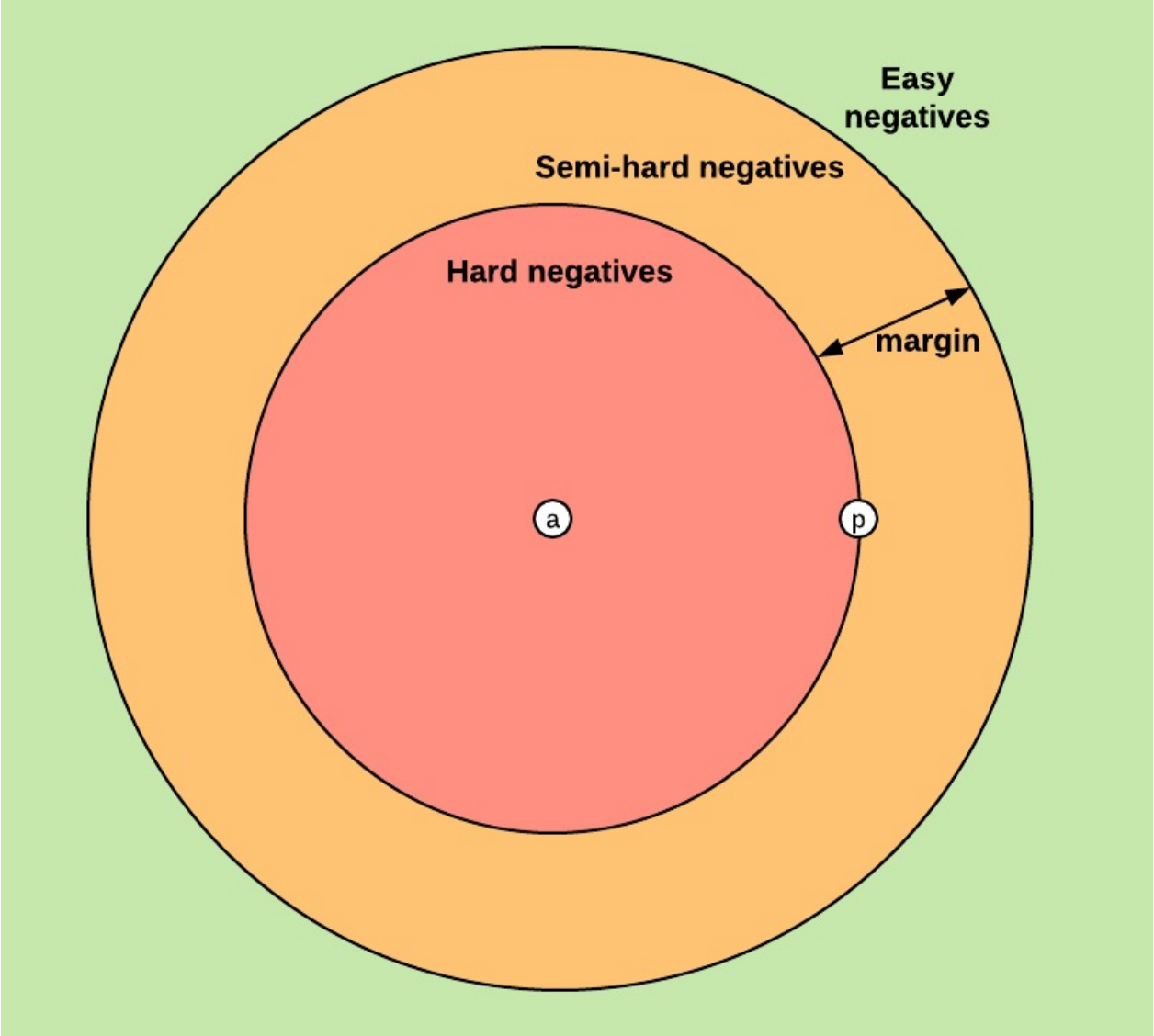
- Each sample is composed by a query q, a relevant document $d^+ \in D^+$ and a irrelevant document $d^- \in D^-$
- **Pairwise Loss**: distance between the q and d^+ should be 0, while between q and d^- should be greater than a margin m

$$L(q,d) = \begin{cases} d(q,d\,) \text{ if } d \in D^+ \\ \max(0, m - d(q,d)) \text{ if } d \in D^- \end{cases}$$

  - when the representation produced for a negative pair is distant enough from the query, no efforts are wasted on enlarging that distance, so further training can focus on more difficult pairs
- **Triplet Loss**: distance between the *q* and d^- is greater (by more than a margin *m)* than the distance between *q* and d^+

$$L(q,d^+,d^-) = \begin{cases} 0 \text{ if } d(q,d^-) - d(q,d^+) \geq m \\ d(q,d^+) - d(q,d^-) + m \text{ otherwise} \end{cases}$$

Types of negatives



https://omoindrot.github.io/triplet-loss

Dense Retrieval Training (I)

- Given a document collection D and a training dataset, a dense retrieval model M is trained
- The loss function is the (binary) pairwise loss *L*:
  - Each sample is composed by a query *q*, a relevant document d^+ and a irrelevant document d^-.

$$L(q, d^+, d^-) = \begin{cases} 1 \text{ if } s(q, d^+) \leq s(q, d^-) \\ 0 \text{ otherwise} \end{cases}$$

- Relevant documents are provided, but we cannot optimize the loss over the whole corpus
- How do we select the irrelevant documents d^-? How many?

Zhan et al. Optimizing Dense Retrieval Model Training with Hard Negatives. SIGIR 2021

Dense Retrieval Training (II)

- Let $\pi(d)$ be the ranking position of document d
- Let $\gamma(d) = \sum_{d^- \in D^-} L(d, d^-)$ be the number of irrelevant documents ranked above d
- Let $\delta(d) = \sum_{d^+ \in D^+} L(d, d^+)$ be the number of relevant documents ranked above d
- Then $\pi(d) = 1 + \gamma(d) + \delta(d)$
- The training goal is

$$M^* = \operatorname{argmin}_M \sum_q \sum_{d^+ \in D^+} \sum_{d^- \in D^-} L(d^+, d^-)$$

$$= \operatorname{argmin}_M \sum_q \sum_{d^+ \in D^+} \gamma(d^+)$$

$$= \operatorname{argmin}_M \sum_q \sum_{d^+ \in D^+} (\pi(d^+) - 1 - \delta(d^+))$$

Large (many irrelevant docs)

Small (few relevant docs)

Zhan et al. Optimizing Dense Retrieval Model Training with Hard Negatives. SIGIR 2021

Negatives Selection (I)

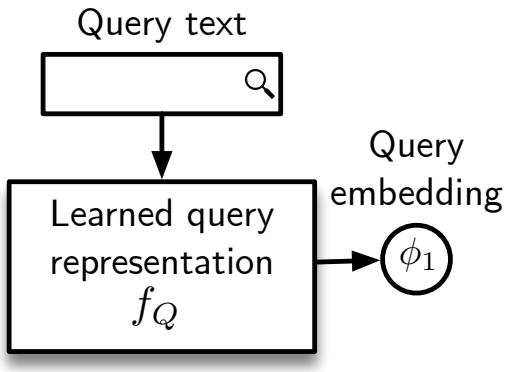
- The per-query loss is dominated by $\pi(d^+)$
- Note that $\delta(d^+)$ can't be higher than the number of relevant documents
- **In-batch selection:**
  - Use the negatives provided in the training set for the given query
- **Cross-batch selection:**
  - Use the negatives provided in the training set for the given query + other queries
- **Random sampling**:
  - Select negatives by uniform sampling over the whole corpus
  - The per-query loss is dominated by $\gamma(d^+)$, that is potentially unbounded

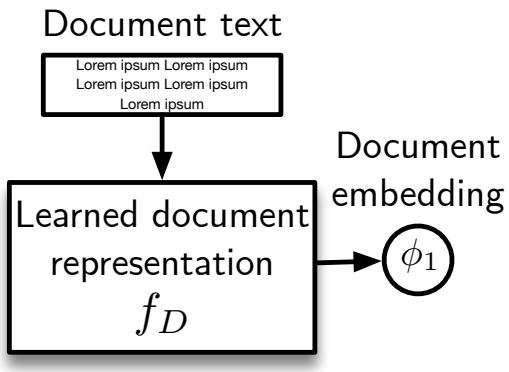
Negatives Selection (II)

- **Hard negatives sampling**
  - Select negative by uniform sampling over the top K documents
  - The per-query loss loss can't be greater than K
  - **Static hard negatives**
    - Hard negatives are generated by a sparse model or a trained dense model
    - Hard negatives are pre-computed before training and never change
  - **Dynamic hard negatives**
    - Hard negative are the current top-ranked irrelevant documents produced by the dense model under training
    - Hard negatives are computed at each training step

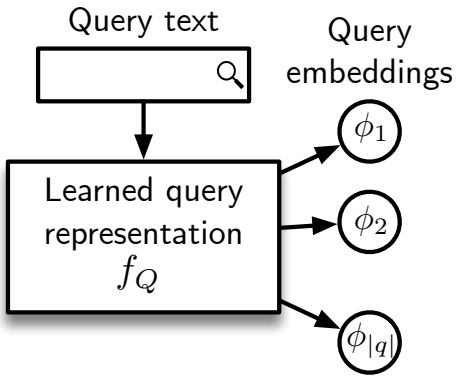
Learning Representations

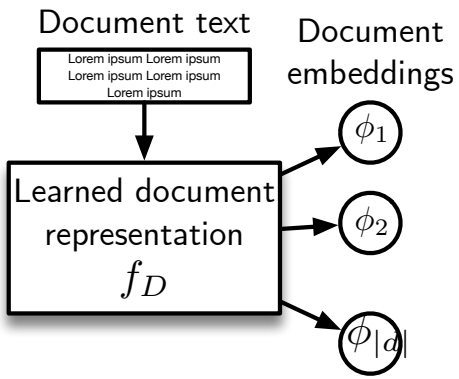
- **Single Representation (DPR, ANCE)**





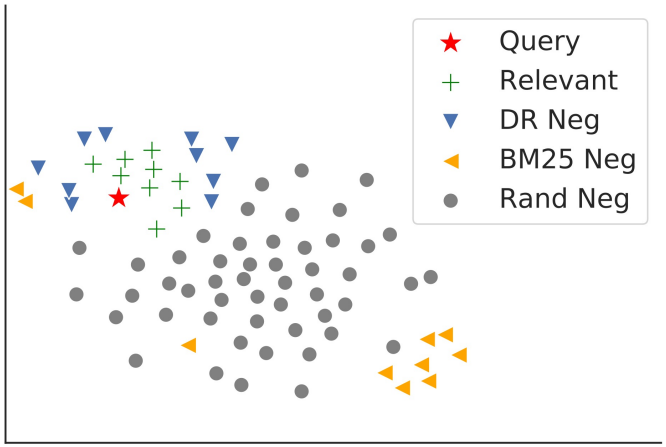
- **Multiple Representation (ColBERT)**

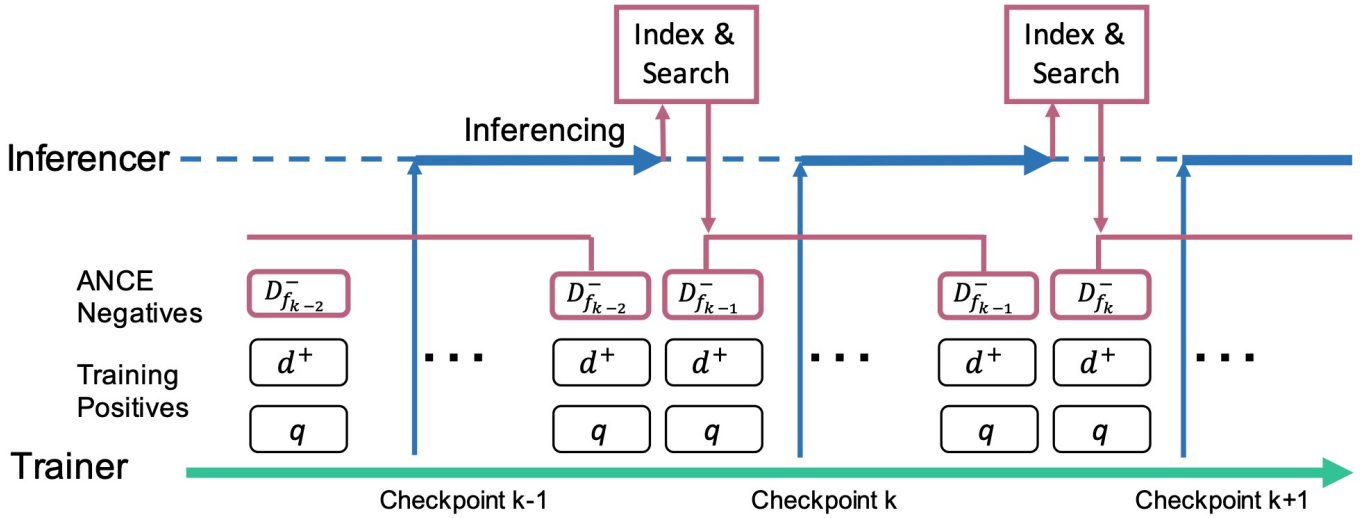




Approximate (nearest neighbor) Negative Contrastive Learning

- Fully learnable representation
- Support for (approximate) NN search
- Problem: negative examples are usually drawn from a (classical) first stage
- How do we generate global negatives?
- Use the model learned so far → dynamic hard negatives





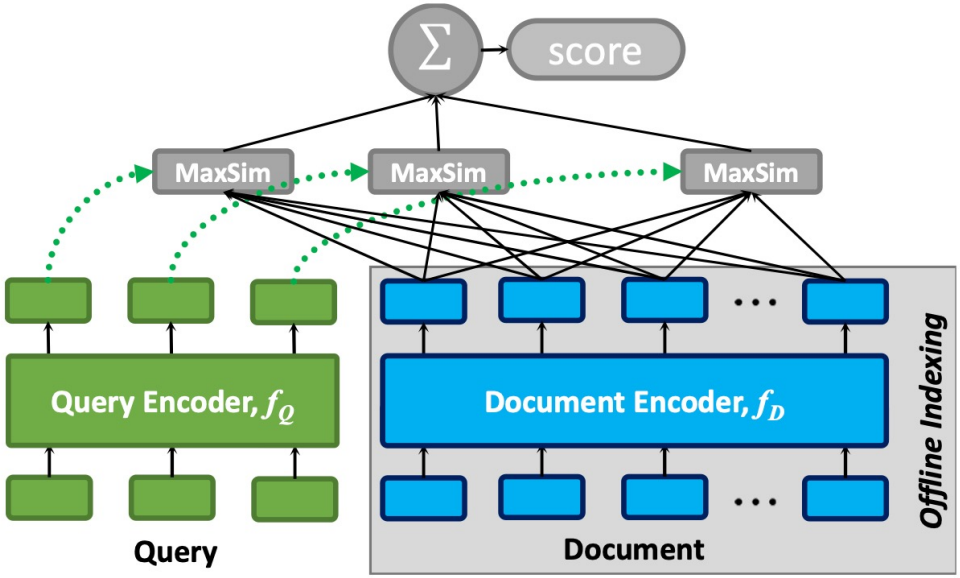
Images from: https://arxiv.org/abs/2007.00808

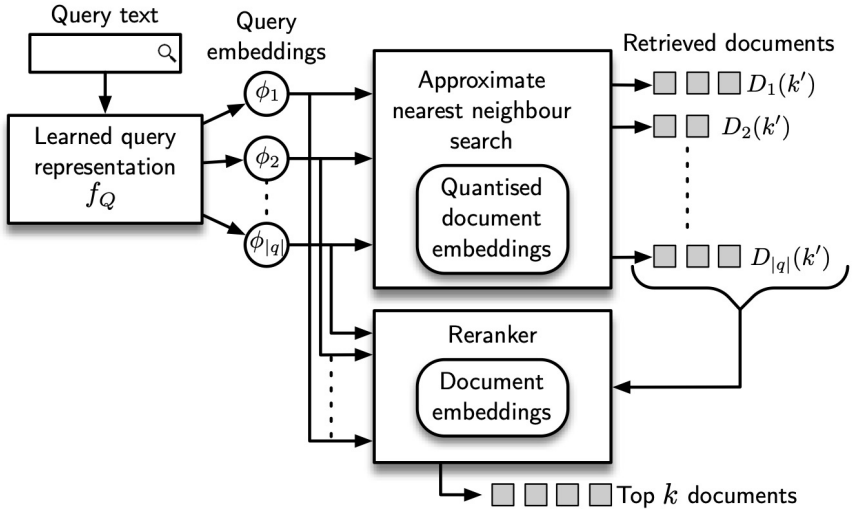
Exact NN Dense Retrieval (DPR, ANCE)

	MARCO Dev Passage Retrieval		TREC DL Passage NDCG@10	
	MRR@10	**Recall@1k**	**Rerank**	**Retrieval**
Sparse & Cascade IR				
BM25	0.240	0.814	–	0.506
Best DeepCT	0.243	n.a.	–	n.a.
Best TREC Trad Retrieval	0.240	n.a.	–	0.554
BERT Reranker	–	–	**0.742**	–
Dense Retrieval				
Rand Neg	0.261	0.949	0.605	0.552
NCE Neg	0.256	0.943	0.602	0.539
BM25 Neg	0.299	0.928	0.664	0.591
DPR (BM25 + Rand Neg)	0.311	0.952	0.653	0.600
BM25 → Rand	0.280	0.948	0.609	0.576
BM25 → NCE Neg	0.279	0.942	0.608	0.571
BM25 → BM25 + Rand	0.306	0.939	0.648	0.591
ANCE (FirstP)	**0.330**	**0.959**	0.677	**0.648**

Images from: https://arxiv.org/abs/2007.00808

ColBERT



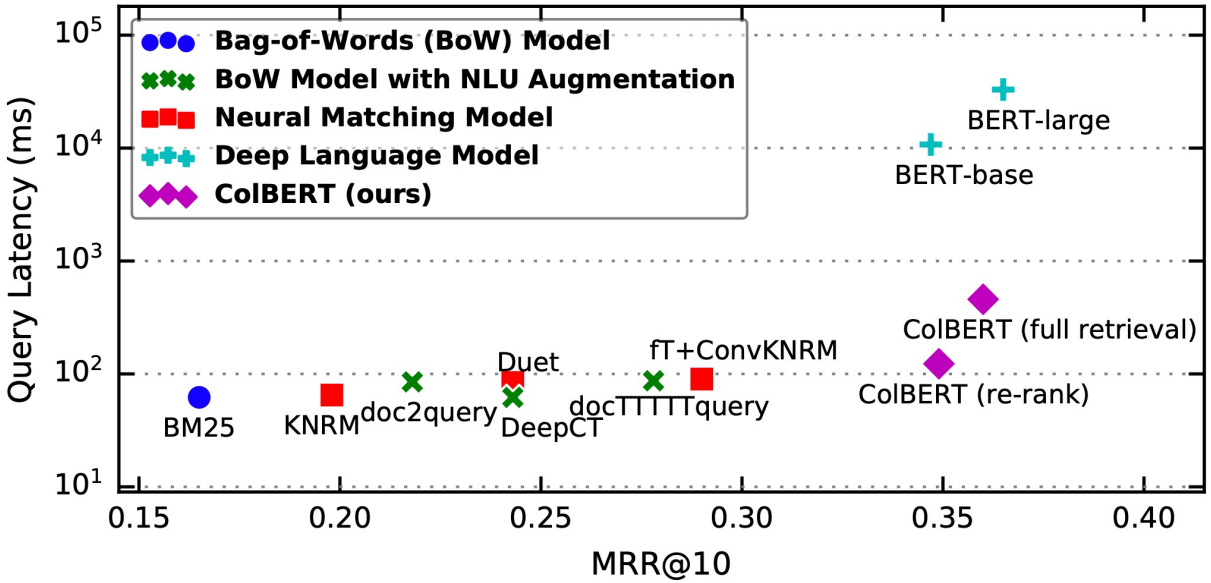



```python
def end_to_end(self) -> TransformerBase:
    """
    Returns a transformer composition that uses a ColBERT FAISS index to retrieve documents, followed by a ColBERT index
    to perform accurate scoring of the retrieved documents. Equivalent to `colbertfactory.set_retrieve() >> colbertfactory.index_scorer()`.
    """
    #input: qid, query,
    #output: qid, query, docno, score
    return self.set_retrieve() >> self.index_scorer(query_encoded=True)
```

Images from: https://arxiv.org/abs/2004.12832

Approximate NN Dense Retrieval ColBERT

Method	MRR@10 (Dev)	MRR@10 (Local Eval)	Latency (ms)	Recall@50	Recall@200	Recall@1000
BM25 (official)	16.7	-	-	-	-	81.4
BM25 (Anserini)	18.7	19.5	62	59.2	73.8	85.7
doc2query	21.5	22.8	85	64.4	77.9	89.1
DeepCT	24.3	-	62 *(est.)*	69 [2]	82 [2]	91 [2]
docTTTTTquery	27.7	28.4	87	75.6	86.9	94.7
ColBERT$_{L2}$ (re-rank)	34.8	36.4	-	75.3	80.5	81.4
ColBERT$_{L2}$ (end-to-end)	36.0	36.7	458	82.9	92.3	96.8



Images from: https://arxiv.org/abs/2004.12832

University of Glasgow | Università di Pisa

CIKM 2021 1-5 NOVEMBER

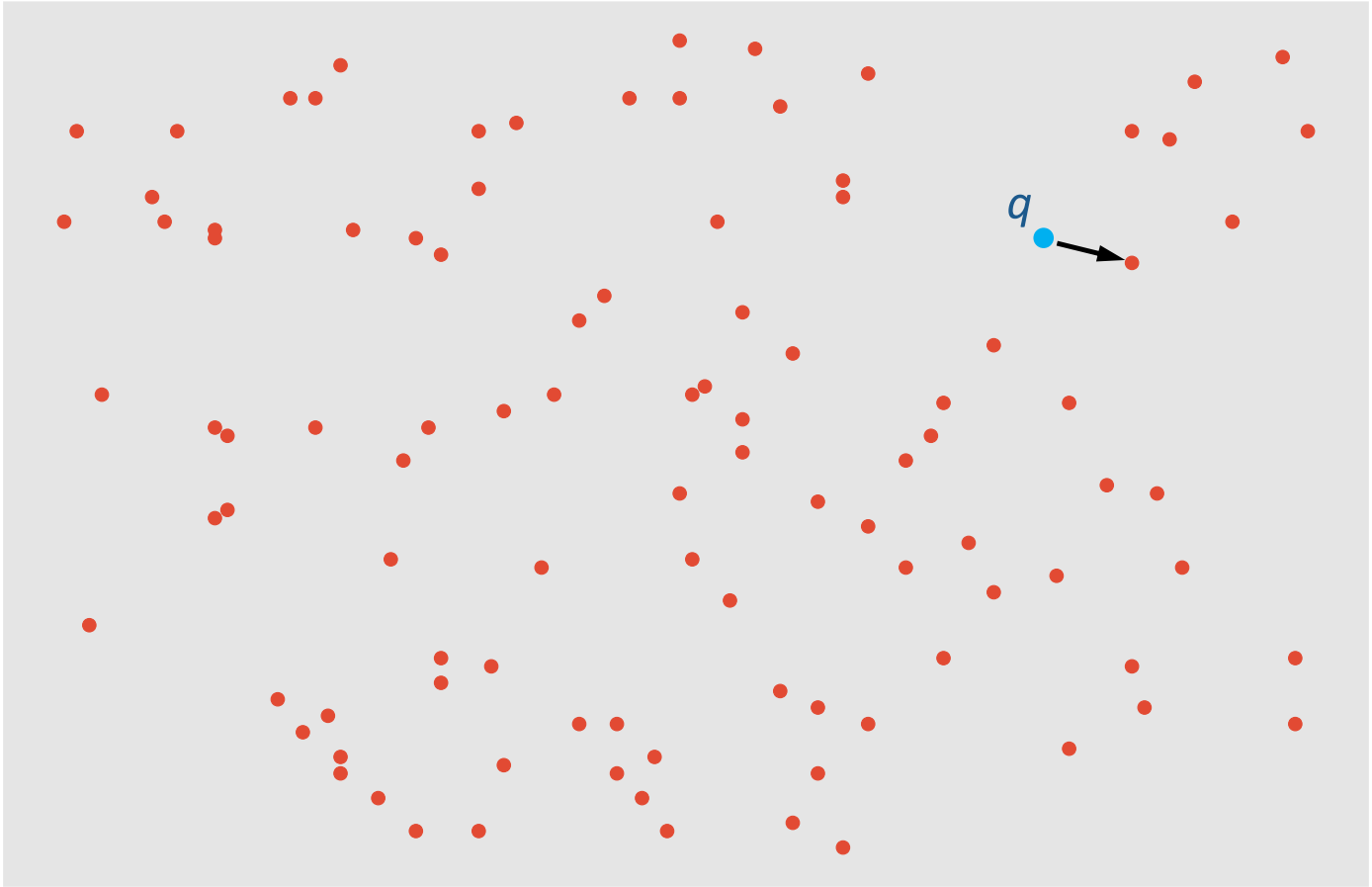
Nearest Neighbour Search

PART 4C

Nearest Neighbour

Università di Pisa | University of Glasgow

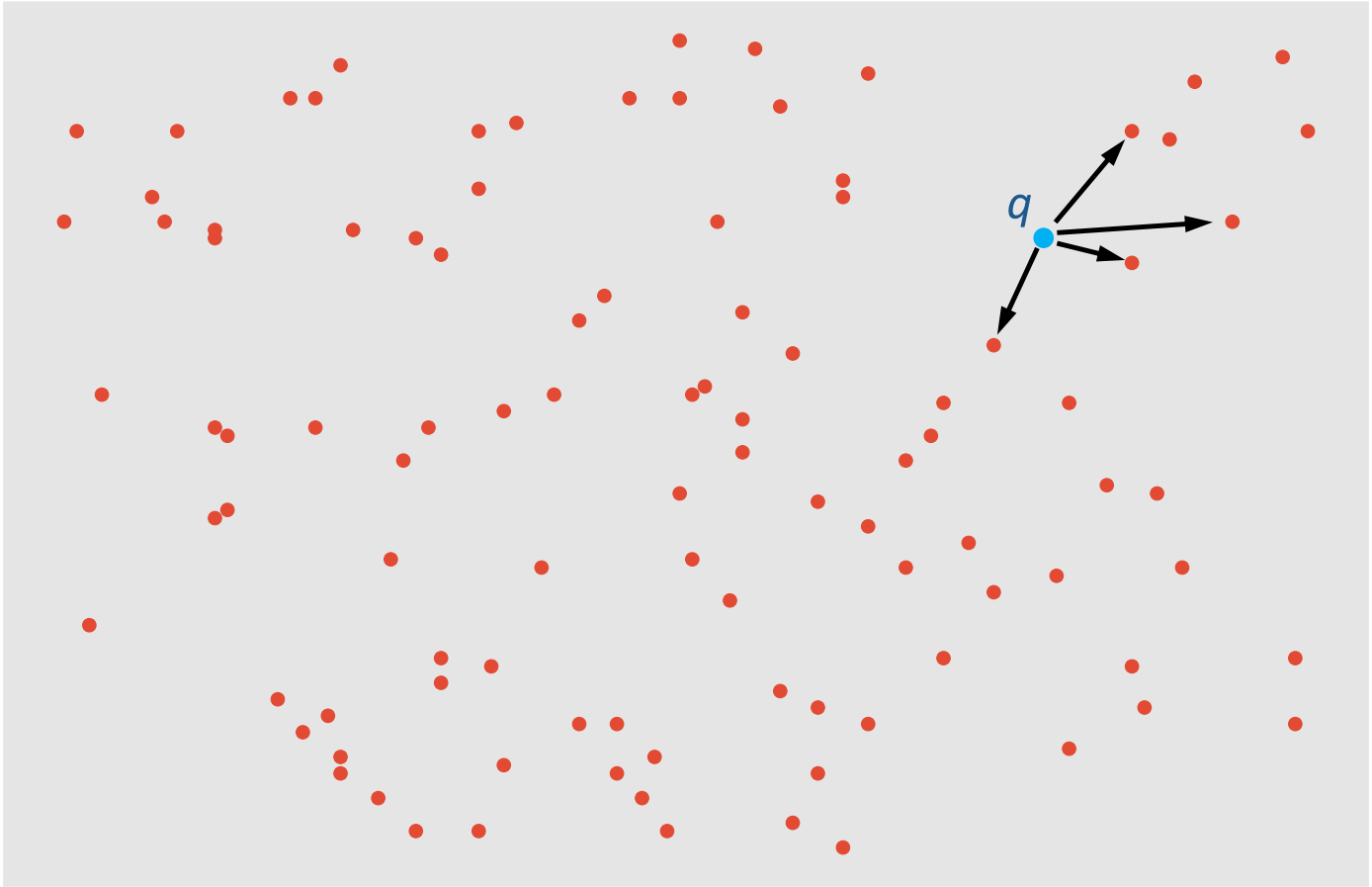
Complexity O(*n*)



k Nearest Neighbours

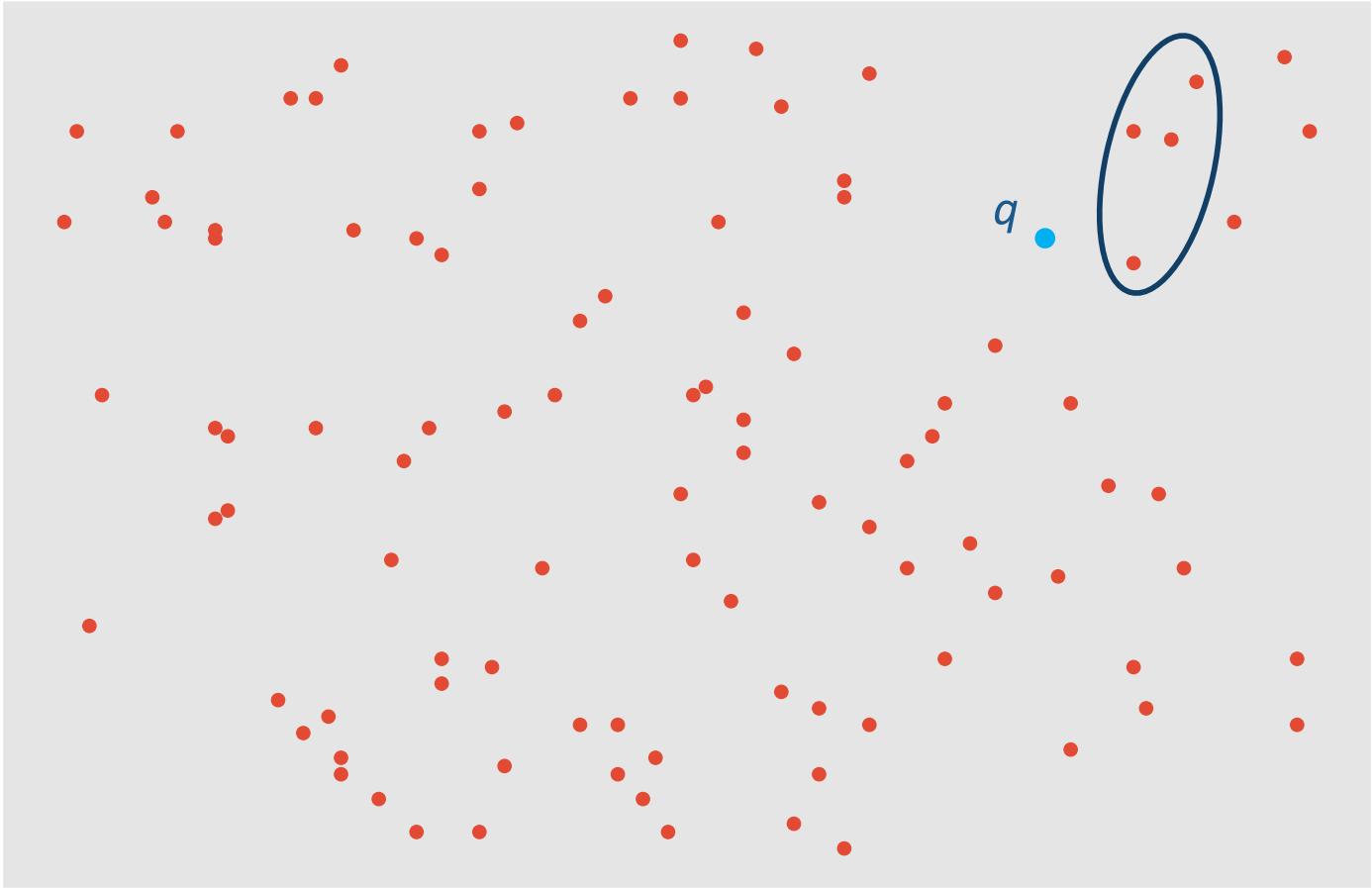
Università di Pisa | University of Glasgow

Complexity O(*n* log *k*)



Approximate Nearest Neighbours

- To make it **scalable**, we instead turn to **approximation** techniques
- **Avoid computing** the distance to every reference vector
- Return points **close** to the nearest neighbors
- Non-Exhaustive
- Common accuracy metric: recall@k



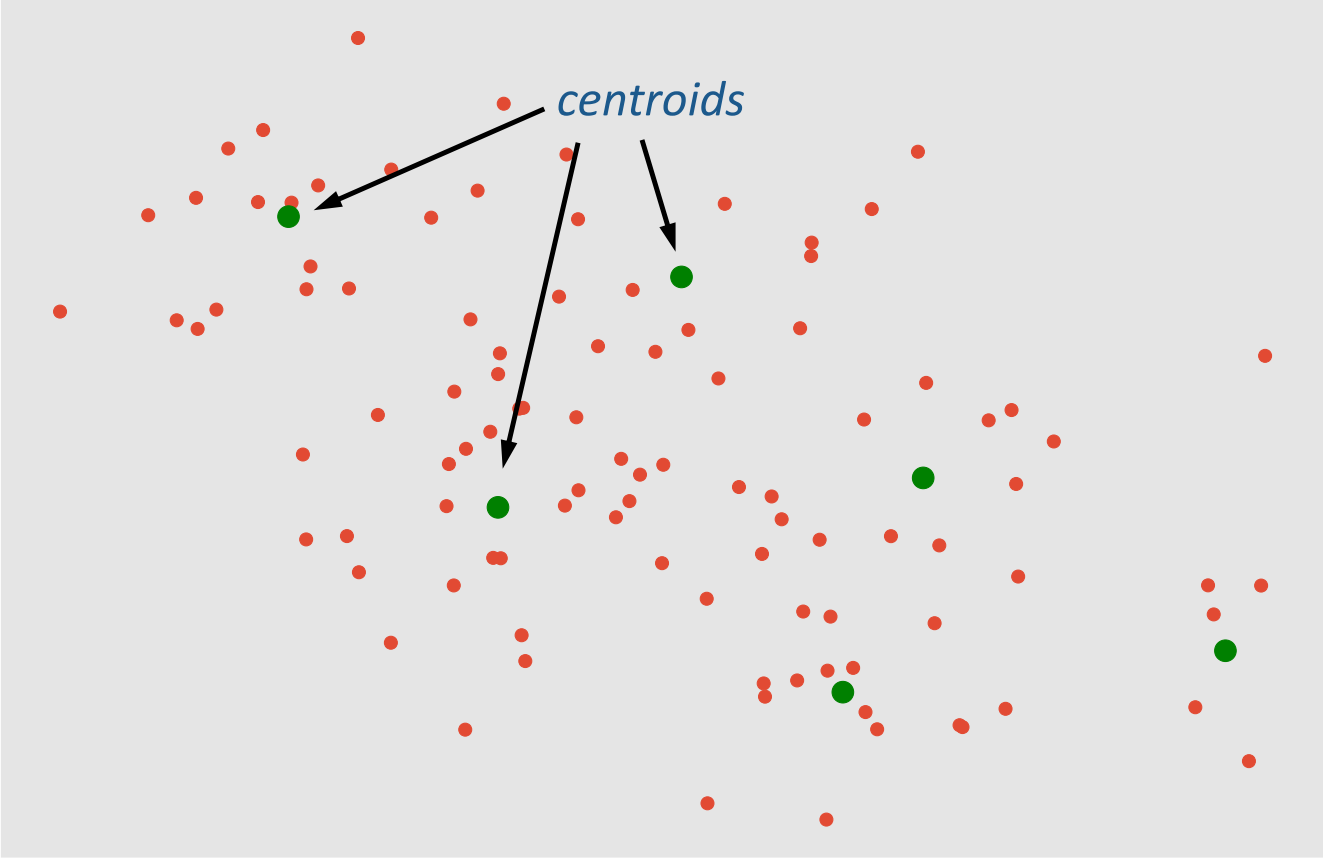
ANN Strategies

- Trees and forests
- Neighborhood graphs
- Locally Sensitive Hashing
- **Quantization**

- Most strategies address the curse of dimensionality problem
- Most strategies are O(*n*) in space and O(log *n*) in time

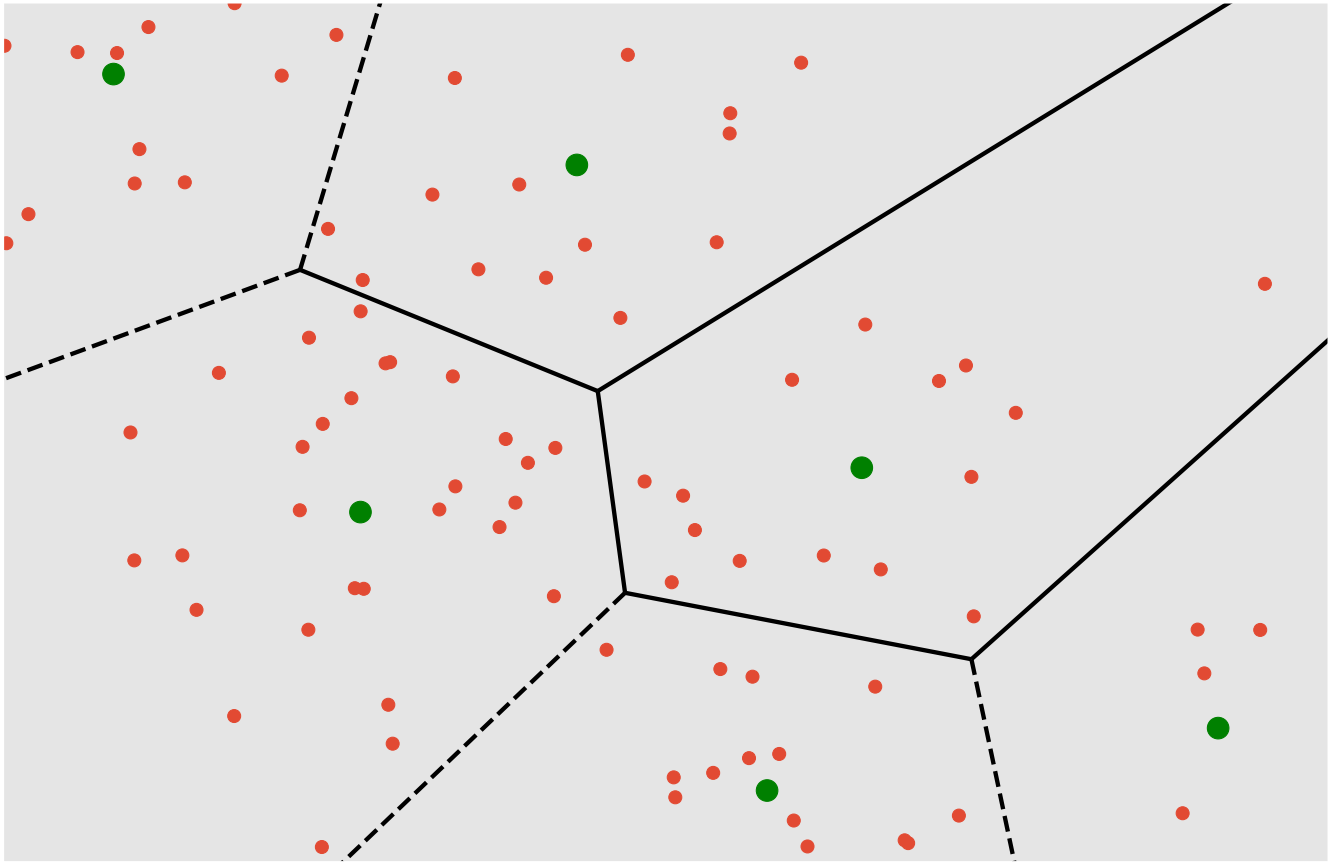
Quantization (I)

From *n* points to *k* centroids



Quantization (II)

Use k-Means to select centroids



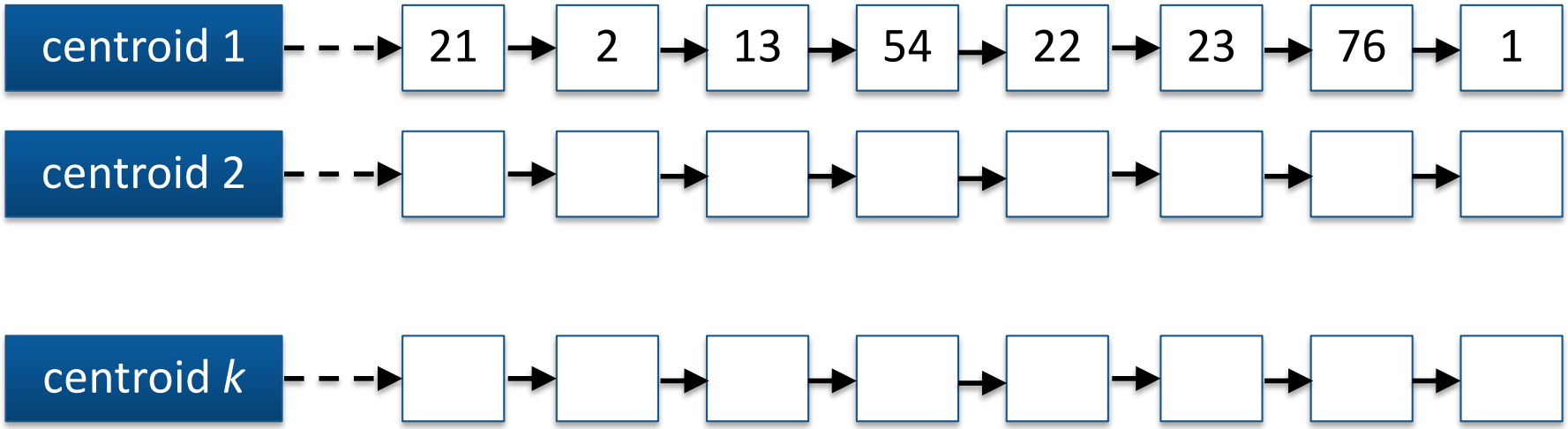
Quantization (III)

- The set of vectors mapped to a given centroid is referred to as a **Voronoi cell**
- The *k* cells form a **partition** of the sample space
- All elements lying in the **same cell** are represented by the **same centroid**
- Each centroid requires O(*D*) space
- Each element requires O(log *n* + log *k*) space for its id and centroid
- The **reduced space** comes at the cost of **distortion**, i.e., the distance between a point and its centroid

Quantization and ANN (I)

- Given a query vector *q:*
  - **Compute the distance** from *q* to all *k* centroids
  - **Scan all *n* elements** and record those falling in the centroid(s) with the smallest distance(s).
  - **Maintain closest neighbours** in a min-heap
  - Problem: it is a linear scan, fast but linear

Quantization and ANN (II)



- **Precompute** a list for each centroid, containing the ids of the elements falling into it
- **Sort elements** in a list by exact distance from the corresponding centroid
- At query time, **sort lists** by distance, and select the closest *k* elements from the first (and subsequent) list(s)
- We still have a **problem**: closest element may not be in the closest cell
- To solve that, we can look at *p* cells and **re-rank elements** in those cells, but we requires the **original element vectors**!

How many centroids?

- If $k << n$, we end up with **many non unique results**
- Consider a 768-dimensional vector (3 KB per element)
- A quantizer producing 384-bit codes (48 B per element), i.e., only 0.5 bit per dimension, contains $\boldsymbol{k = 2^{348}}$ **centroids**
- It is **impossible to use** any K-Means algorithm
- It is **impossible to store** the kD floats representing the k centroids

Product Quantization (I)

- Instead of dealing with all ***D* dimensions** at the same time, we consider ***D/m* dimensions** at a time
- We **split** each element **into *m* chunks**
- We run **K-Means on each chunk** independently

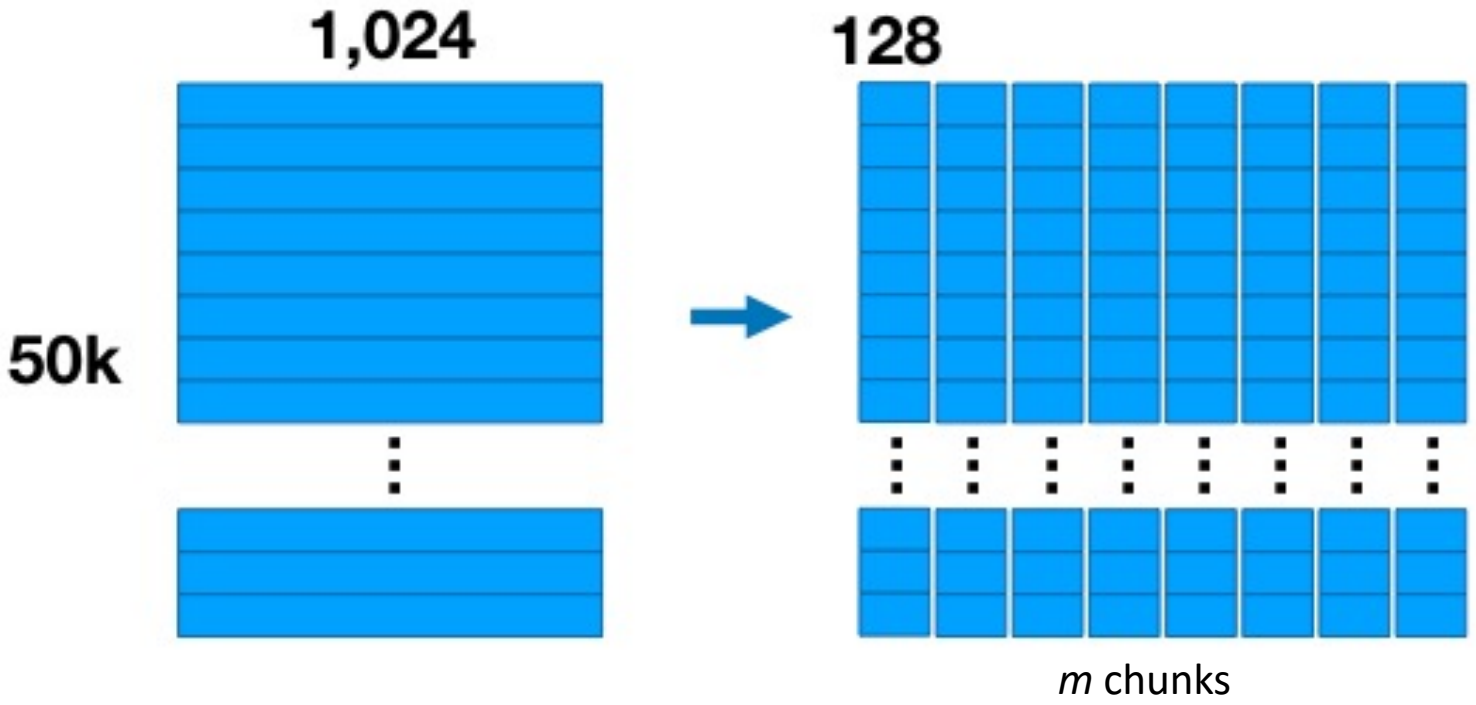


Image from: http://mccormickml.com/2017/10/13/product-quantizer-tutorial-part-1/

Product Quantization (II)

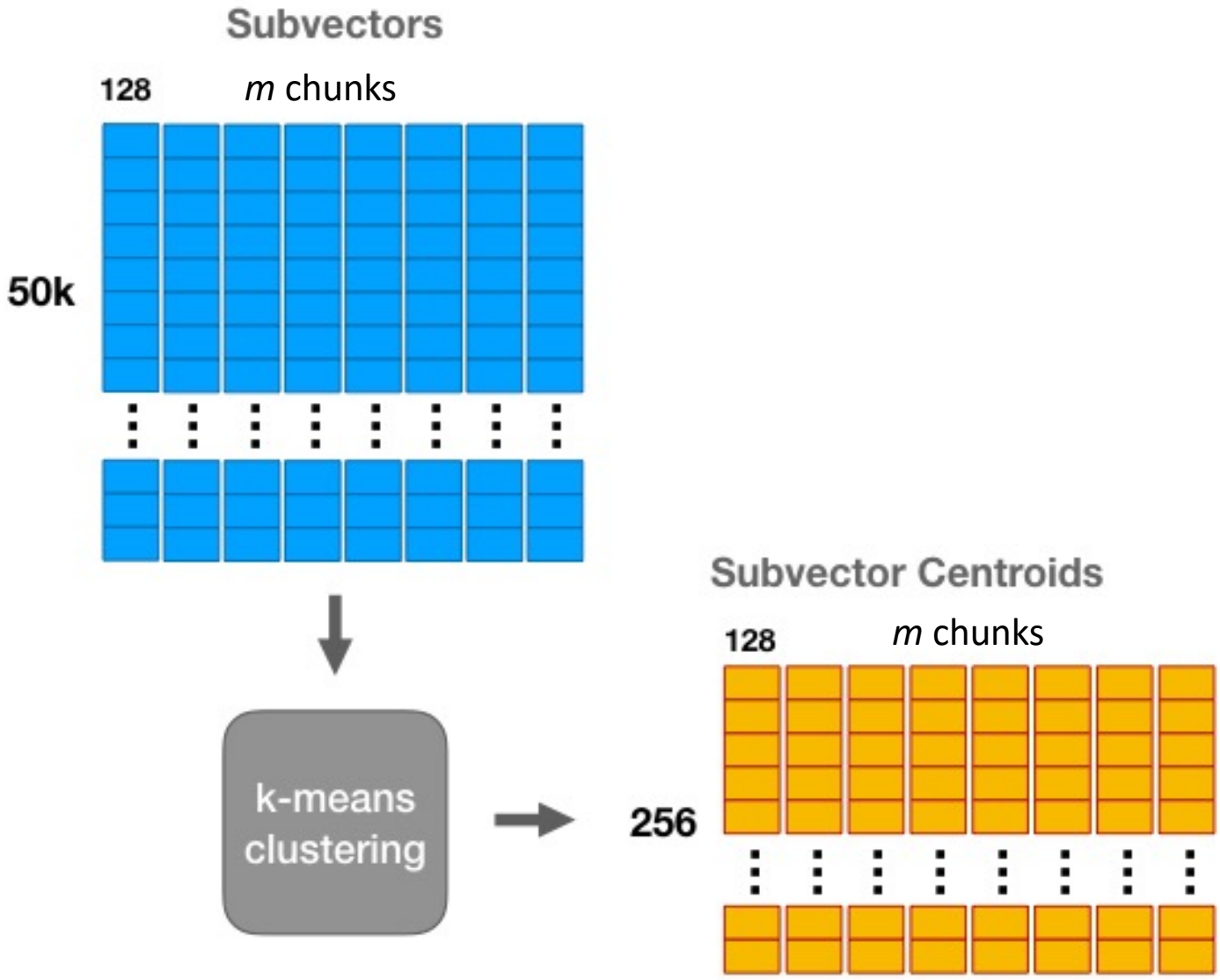


Image from: http://mccormickml.com/2017/10/13/product-quantizer-tutorial-part-1/

Product Quantization (III)

Each original element is now just a sequence of m = 8 centroid ids

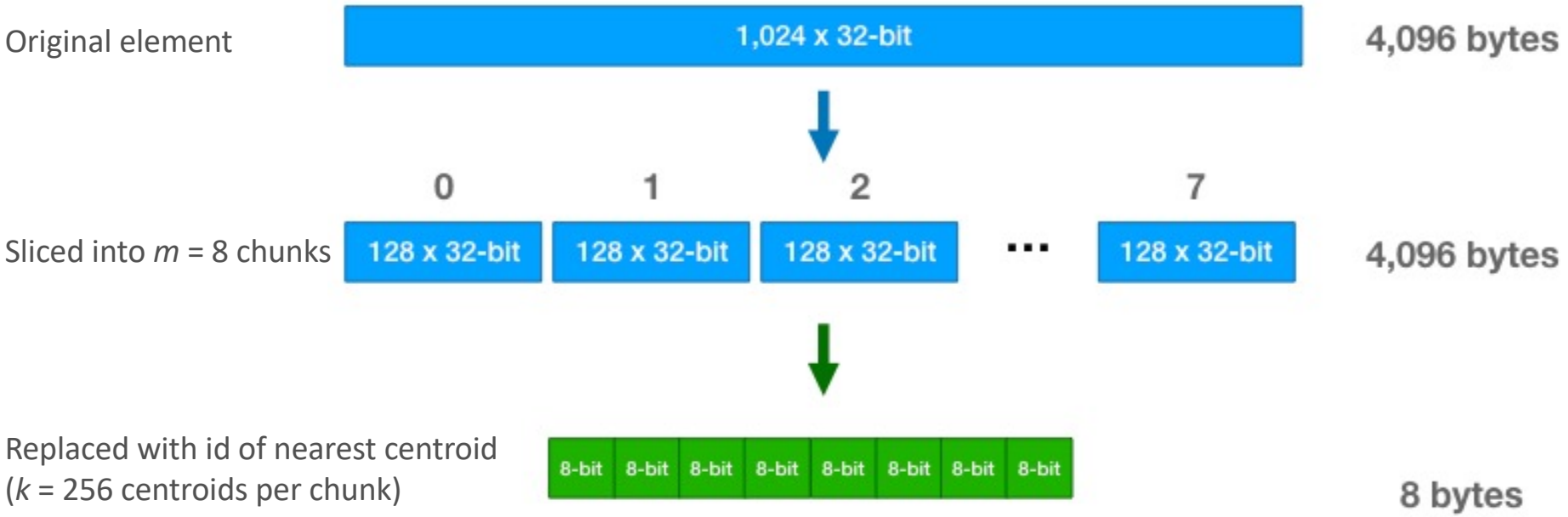


Image from: http://mccormickml.com/2017/10/13/product-quantizer-tutorial-part-1/

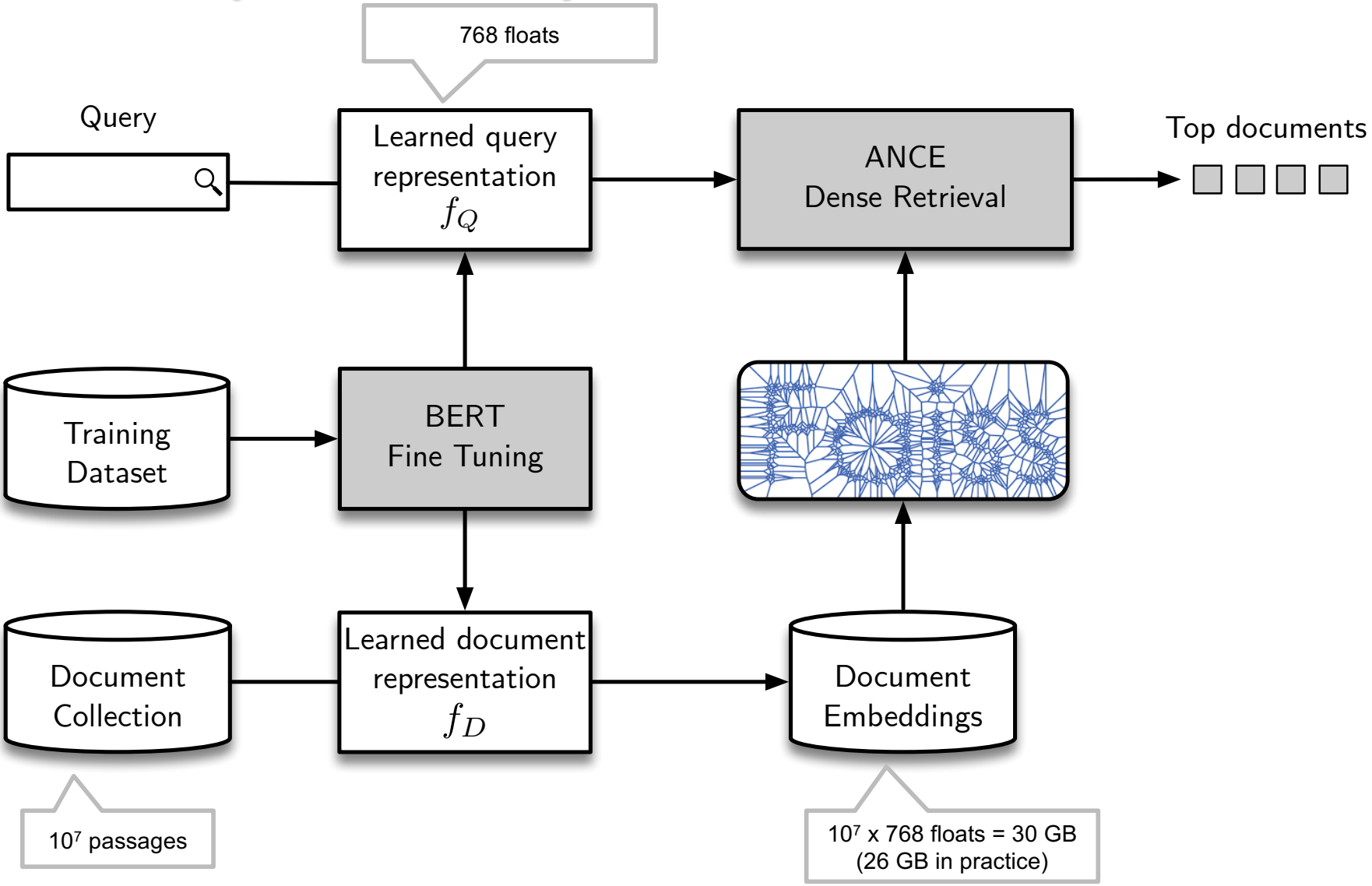
Product Quantization and ANN

- Given a query *q:*
  - **Compute** the squared distance between **each chunk** of our vector and each of the centroids for that chunk
    - This means building a table of subvector distances with *k* rows (one for each centroid) and *m* columns (one foreach chunk)
  - To compute the **approximate distance** between a given element and the query *q*, use the centroid ids to lookup the partial distances in the table, and sum them up
  - Sort the distances to find the **smallest distances**

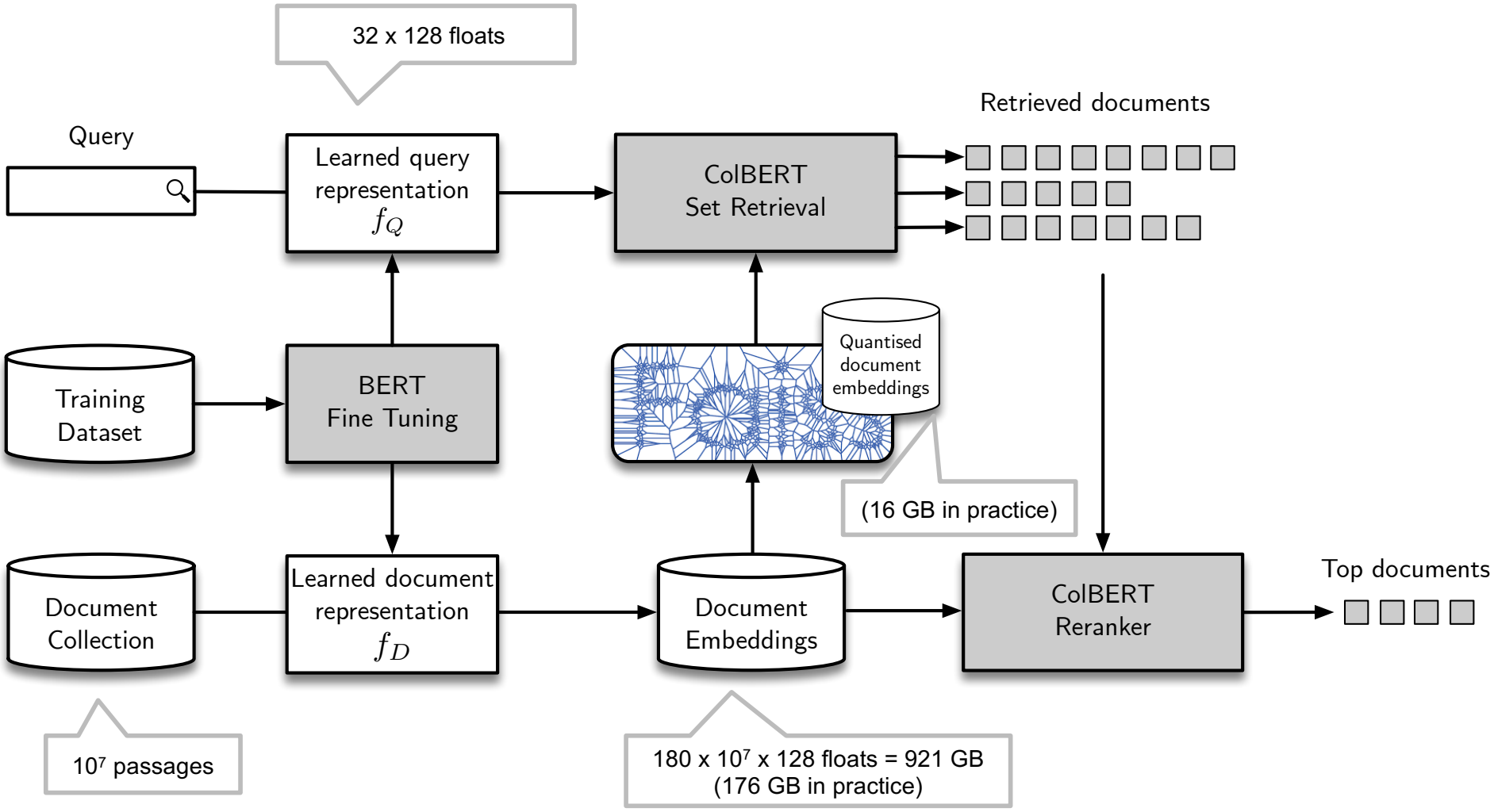
Further Optimizations

- **Inverted File Index**:
  - Pre-filtering the dataset to **avoid exhaustive search** over all elements
  - **Cluster** the dataset ahead of time with Kmeans clustering to produce a large number (e.g., 100) of dataset **partitions**
  - At query time, you compare your query vector to the partition centroids to find, e.g., the 10 closest clusters, and then you search against only the vectors in those partitions
- **Residual Encoding**:
  - For each element, instead of using the PQ to encode the original database vector instead encode the element's offset from its partition centroid

Exact NN Dense Retrieval (DPR, ANCE)



Approximate NN Dense Retrieval (ColBERT)



ColBERT PRF

- ColBERT's token embeddings can be used to
  - select representative embeddings in a set of feedback documents using clustering
  - select discriminative embeddings among the representative embeddings using corresponding token's IDF
  - select the top tokens corresponding to discriminative embeddings for query expansions
- This approach works for both ranking and re-ranking scenarios

```
dense_e2e = pytcolbert.set_retrieve() >> pytcolbert.index_scorer(query_encoded=True, add_ranks=True)
if rerank:
    prf_pipe = (
        dense_e2e
        >> ColbertPRF(pytcolbert, k=k, fb_docs=fb_docs, fb_embs=fb_embs, beta=beta, return_docs=True)
        >> (pytcolbert.index_scorer(query_encoded=True, add_ranks=True) %1000)
    )
else:
    prf_pipe = (
        dense_e2e
        >> ColbertPRF(pytcolbert, k=k, fb_docs=fb_docs, fb_embs=fb_embs, beta=beta, return_docs=False)
        >> pytcolbert.set_retrieve(query_encoded=True)
        >> (pytcolbert.index_scorer(query_encoded=True, add_ranks=True) % 1000)
    )
return prf_pipe
```

Wang et al. Pseudo-Relevance Feedback for Multiple Representation Dense Retrieval. ICTIR 2021

ColBERT Optimisations

- **Single** (ANCE) **vs Multiple** (ColBERT) **Representations**
  - ANCE is more efficient than ColBERT
  - ColBERT is more effective than ANCE for MAP and MRR@10
  - ColBERT obtains better improvements than ANCE for queries that are the hardest for BM25, as well as for definitional queries, and those with complex information needs

Macdonald, Tonellotto, Ounis. On Single and Multiple Representations in Dense Passage Retrieval. IIR 2021

- **Embedding Pruning**
  - ColBERT does not need all 32 query embeddings, just 3
  - Same effectiveness, 2.65x speedup in efficiency

Tonellotto & Macdonald. Query Embedding Pruning for Dense Retrieval. CIKM 2021

- **Approximate Scoring**
  - ColBERT can use approximate scores from ANN to decrease the number of candidate documents being fully scored, e.g. 200 documents only
  - Same effectiveness, 2x speedup in efficiency

Macdonald & Tonellotto. On Approximate Nearest Neighbour Selection for Multi-Stage Dense Retrieval. CIKM 2021

Summary

- **At index time**
  - Select the number of partitions for the coarse quantizer
  - Run clustering on the elements to place them in the partitions
  - Compute residuals in each partition
  - Select the number m of chunks for each partition
  - Run clustering on the residuals in each chunk
  - For each cluster in each chunk, organize the residuals id in a list sorted by increasing distance from cluster center
- **At query time**
  - Select the number of the closest partitions to search
  - For each partition, compute distance table and identify closest clusters
  - Select the top elements in the closest clusters

Let's see how approximate nearest neighbour search can be used for retrieval

From passages to documents

Università di Pisa | University of Glasgow

ANCE and ColBERT are designed for passages

- e.g. BERT has limited number of tokens for inference

For long documents, we need to break into passages

PyTerrier's indexing pipelines (c.f. part 2) offer a solution

```
import pyterrier_ance

ance_indexer = pt.text.sliding() >> pyterrier_ance.ANCEIndexer(
    checkpoint_path="./ance_model_checkpoint",
    index_path="/content/anceindex",
    num_docs=192509,
    text_attr='abstract' # COVID
)

ance_indexer.index(dataset.get_corpus_iter())
```

```
from pyterrier_ance import ANCERetrieval

ance_retriever = ANCERetrieval.from_dataset('trec-covid', 'ance_msmarco_psg') >> pt.text.max_passage()
```

ANCE Example

- Indexing

```
index = pt.datasets.get_dataset('trec-covid').get_index('terrier_stemmed')
ance_index = pt.datasets.get_dataset('trec-covid').get_index('ance_msmarco_psg')
```

- Retrieval

```
bm25_retriever = pt.BatchRetrieve(index, wmodel="BM25")
ance_retriever = ANCERetrieval.from_dataset('trec-covid', 'ance_msmarco_psg') >> pt.text.max_passage()
```

- Experiment

```
pt.Experiment(
    [bm25_retriever % 10, ance_retriever % 10],
    topics,
    qrels,
    eval_metrics=["map", "recip_rank", "P_10", "ndcg_cut_10", "mrt"],
    names=['BM25', 'ANCE'],
)
```

ColBERT Example

- End-to-end Retrieval

```
from pyterrier_colbert.ranking import ColBERTFactory

bm25_terrier_stemmed = pt.BatchRetrieve.from_dataset('vaswani', 'terrier_stemmed', wmodel='BM25')

factory = ColBERTFactory.from_dataset('vaswani', 'colbert_uog44k')
colbert_e2e = factory.end_to_end()
```

- Experiment

```
pt.Experiment(
    [bm25_terrier_stemmed % 10, colbert_e2e % 10],
    topics,
    qrels,
    eval_metrics=["map", "recip_rank", "P_10", "ndcg_cut_10", "mrt"],
    names=['BM25', 'ColBERT'],
)
```

Round Up

- Re-rankers like BERT are great for **improving the precision** in IR tasks
  - Re-rankers cannot improve the **recall** below re-ranking threshold
  - Re-rankers tend to be expensive to run
- We can **use neural models** to directly learn scalar values to score terms in documents and store them in an inverted index
- PyTerrier **provides a plugin for** DeepImpact

- We can use **dense retrieval systems** to **improve effectiveness** while **reducing the processing time** of queries
  - Represent **queries and documents as** one or more **embeddings** (single vs. multiple representation)
  - Embeddings are store in a system providing **efficient support for NN search**
  - Depending on the size and number of embeddings, NN search can be **exact** or **approximate**
- PyTerrier **provides plugins for** ANCE and ColBERT

Github References

- DeepImpact PyTerrier plugin:

https://github.com/terrierteam/pyterrier_deepimpact

- ANCE PyTerrier plugin

https://github.com/terrierteam/pyterrier_ance

- ColBERT PyTerrier plugin

https://github.com/terrierteam/pyterrier_colbert

Questions

Practical Time

Università di Pisa | University of Glasgow

The tutorial Github repo has links to the notebook for Part 4

- https://github.com/terrier-org/cikm2021tutorial
- Press the 

link for each notebook to start a Colab session

Timings

- Practical – in breakout rooms

When you are finished…

- Please complete our feedback quiz
  - https://forms.office.com/r/RiYSAxAKhk

Acknowledgements

The authors gratefully acknowledge other users & contributors to PyTerrier

Iadh Ounis and Xiao Wang provided input to this tutorial

Sean & Craig gratefully acknowledge the support of EPSRC grant EP/ R018634/1: Closed-Loop Data Science for Complex, Computationally- & Data-Intensive Analytics