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

An ECIR 2021 tutorial with PyTerrier and OpenNIR

Sean MacAvaney*
Craig Macdonald*
Nicola Tonellotto*

*(*Alphabetical ordering)*

PART 4
NEURAL INVERTED INDEX AUGMENTATION & NEAREST NEIGHBOUR SEARCH + 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!
- Two recent approaches for improving first-stage retrieval:
  1. Augmenting the inverted index
  2. Dense retrieval

Intended Learning Outcomes of Part 4

ILO 4A. Understand the most recent effective retrieval architectures that augment traditional index structures such as doc2query and DeepCT, as well as dense retrieval approaches

ILO 4B. Experience ANCE and ColBERT dense retrieval approaches, as well as doc2query and DeepCT

Outline of Part 2

Università di Pisa | University of Glasgow

Part 4A: Augmenting the inverted index

Part 4B: Nearest neighbour search

Part 4C: Dense retrieval

University of Glasgow | Università di Pisa

ECIR 2021

Augmenting the Inverted Index

PART 4A

Augmenting the Inverted Index

Università di Pisa | University of Glasgow

We can use neural models to modify the content stored in classical inverted indices.

Two main strategies:

- Emphasizing important terms (DeepCT)
- Generating additional terms to index (doc2query)

DeepCT

Università di Pisa | University of Glasgow

Main idea: Boost the term frequency of important words by repeating them in the text.

Abstract:
Nidovirus subgenomic mRNAs contain a leader sequence derived from the 5′ end of the genome fused to different sequences (‘bodies’) derived from the 3′ end. Their generation involves a unique mechanism of discontinuous subgenomic RNA synthesis that resembles copy-choice RNA recombination. During this process, the nascent RNA strand is transferred from one site in the template to another, during either plus or minus…

Abstract:
Nidovirus Nidovirus Nidovirus subgenomic subgenomic subgenomic mRNAs mRNAs mRNAs contain a leader sequence derived from the 5′ end of the genome fused to different sequences (‘bodies’) derived from the 3′ end. Their generation involves a unique mechanism of discontinuous **subgenomic subgenomic RNA RNA** synthesis that resembles copy-choice **RNA RNA RNA** recombination. During this process, the nascent **RNA RNA RNA** strand is transferred from one site in the template to another, during either plus or minus…

Training: Predict the terms that match relevant query terms.

Dai & Callan. Context-Aware Sentence/Passage Term Importance Estimation For First Stage Retrieval. arXiv 2019.

DeepCT practical

Università di Pisa | University of Glasgow

```
deepct = pyterrier_deepct.DeepCTTransformer(
    "bert-base-uncased/bert_config.json",
    "marco/model.ckpt-65816")
```

Example DeepCT outputs:

```
df.iloc[0]['text']
```

```
'OBJECTIVE: This retrospective chart review describes the epidemiology and clinical
features of 40 patients with culture-proven Mycoplasma pneumoniae infections at King
Abdulaziz University Hospital, Jeddah, Saudi Arabia. METHODS: Patients with positive
M. pneumoniae cultures from respiratory specimens from January 1997 through December
```

```
deepct.transform(df).iloc[0]["text"]
```

```
'objective objective retrospective retrospective chart chart chart chart chart revie
w describes epidemiology epidemiology epidemiology epidemiology epidemiology epidemi
ology epidemiology epidemiology epidemiology clinical clinical clinical clinical cli
nical clinical clinical clinical features features features features patients patien
```

DeepCT practical

```
dataset = pt.get_dataset("irds:cord19/trec-covid")
index_loc = "./deepct_index_path"
indexer = deepct >> pt.IterDictIndexer(index_loc)
```

Build an *indexing* pipeline

```
indexref = indexer.index(dataset.get_corpus_iter())
```

Index the collection

TODO: performance on cord19

Doc2Query

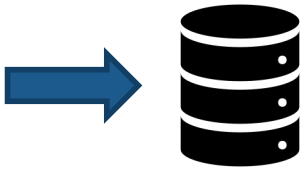
Main idea: Use a causal language model to generate additional text to add to documents when indexing.

At retrieval time, use standard retrieval model (e.g., BM25).

How are the queries generated?

Abstract:
Nidovirus subgenomic mRNAs contain a leader sequence derived from the 5′ end of the genome fused to different sequences (‘bodies’) derived from the 3′ end. Their generation involves a unique mechanism of discontinuous subgenomic RNA synthesis that resembles copy-choice RNA recombination. During this process, the nascent RNA strand is transferred from one site in the template to another, during either plus or minus…

Generated Queries:
where does mrna originate
where does subgenomic rna come from
where is the nidovirus mrna?
when a nidovirus is produced, its genome is quizlet
what is nidovirus mrna
where in a nidovirus can one mrna be derived
where is the leader sequence derived
what types of mrna are found in nidovirus
where are the nidovirus genomes derived from
…



Nogueira & Lin. From doc2query to docTTTTTquery. 2019.

Doc2Query

Università di Pisa | University of Glasgow

Training: optimize to generate queries from passages on a collection with many queries (e.g., MS-MARCO)

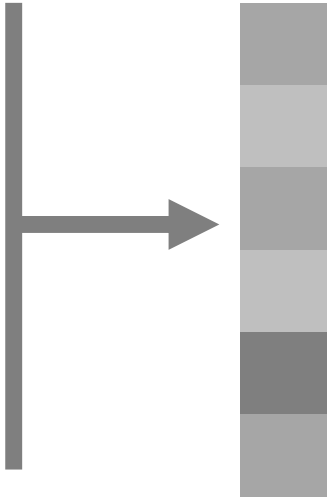
Most recent version: Use T5 model for generation (docTTTTTquery)

Nogueira & Lin. From doc2query to docTTTTTquery. 2019.

Doc2Query

Università di Pisa | University of Glasgow

Abstract:
Nidovirus subgenomic mRNAs contain a leader sequence derived from the 5′ end of the genome fused to different sequences (‘bodies’) derived from the 3′ end. Their generation involves a unique mechanism of discontinuous subgenomic RNA synthesis that resembles copy-choice RNA recombination. During this process, the nascent RNA strand is transferred from one site in the template to another, during either plus or minus…



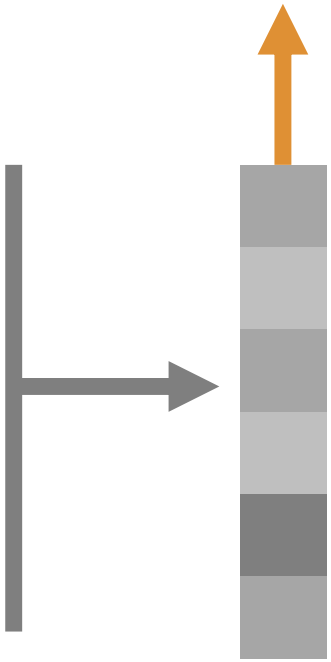
Step 1: Source text encoded

e.g., via RNN or Transformer

Nogueira & Lin. From doc2query to docTTTTTquery. 2019.

Doc2Query

Abstract:
Nidovirus subgenomic mRNAs contain a leader sequence derived from the 5′ end of the genome fused to different sequences (‘bodies’) derived from the 3′ end. Their generation involves a unique mechanism of discontinuous subgenomic RNA synthesis that resembles copy-choice RNA recombination. During this process, the nascent RNA strand is transferred from one site in the template to another, during either plus or minus…

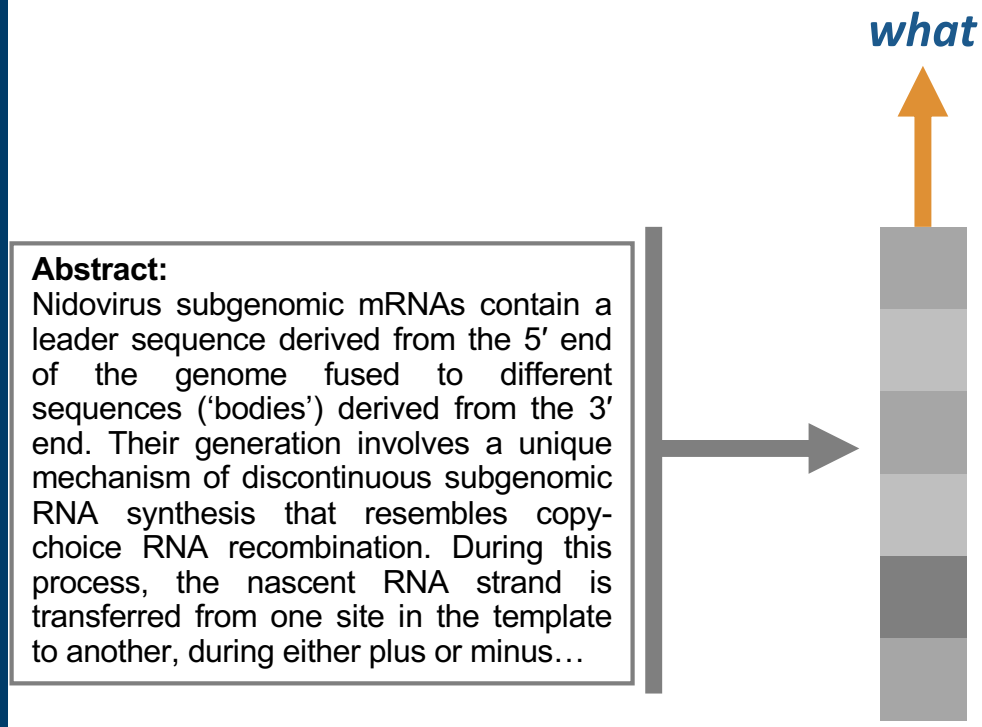


Step 2: Text iteratively generated from encoded document

e.g., via RNN or transformer, using beam search

Nogueira & Lin. From doc2query to docTTTTTquery. 2019.

Doc2Query

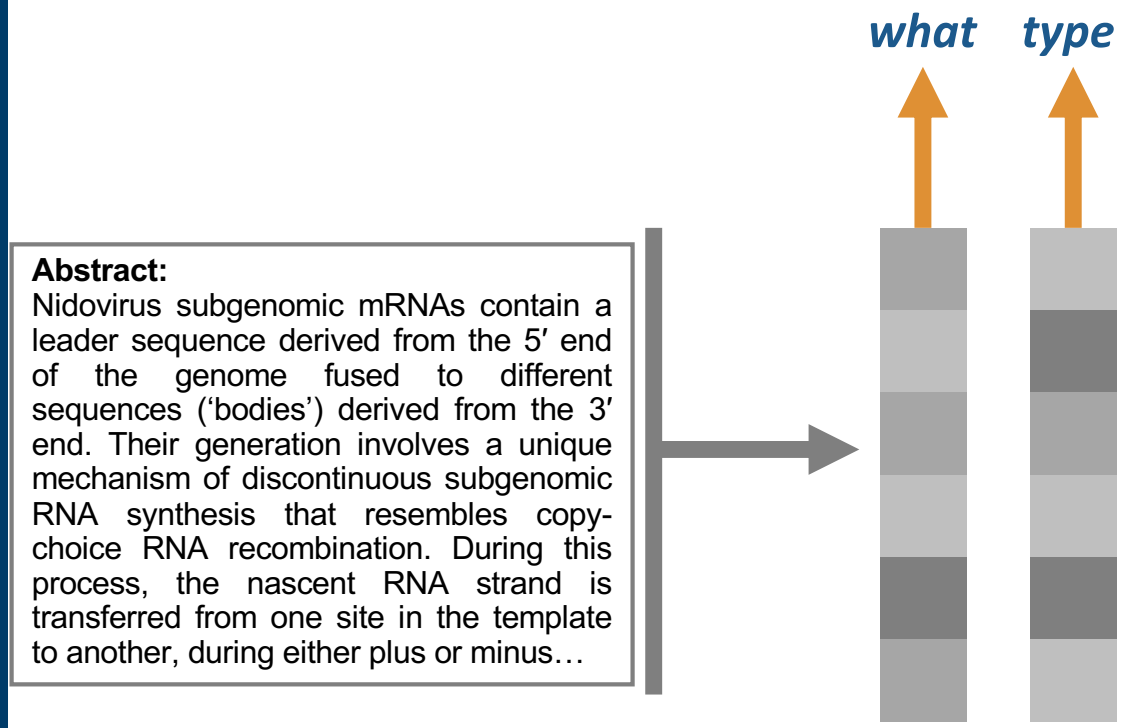


Step 2: Text iteratively generated from encoded document

e.g., via RNN or transformer, using beam search

Nogueira & Lin. From doc2query to docTTTTTquery. 2019.

Doc2Query

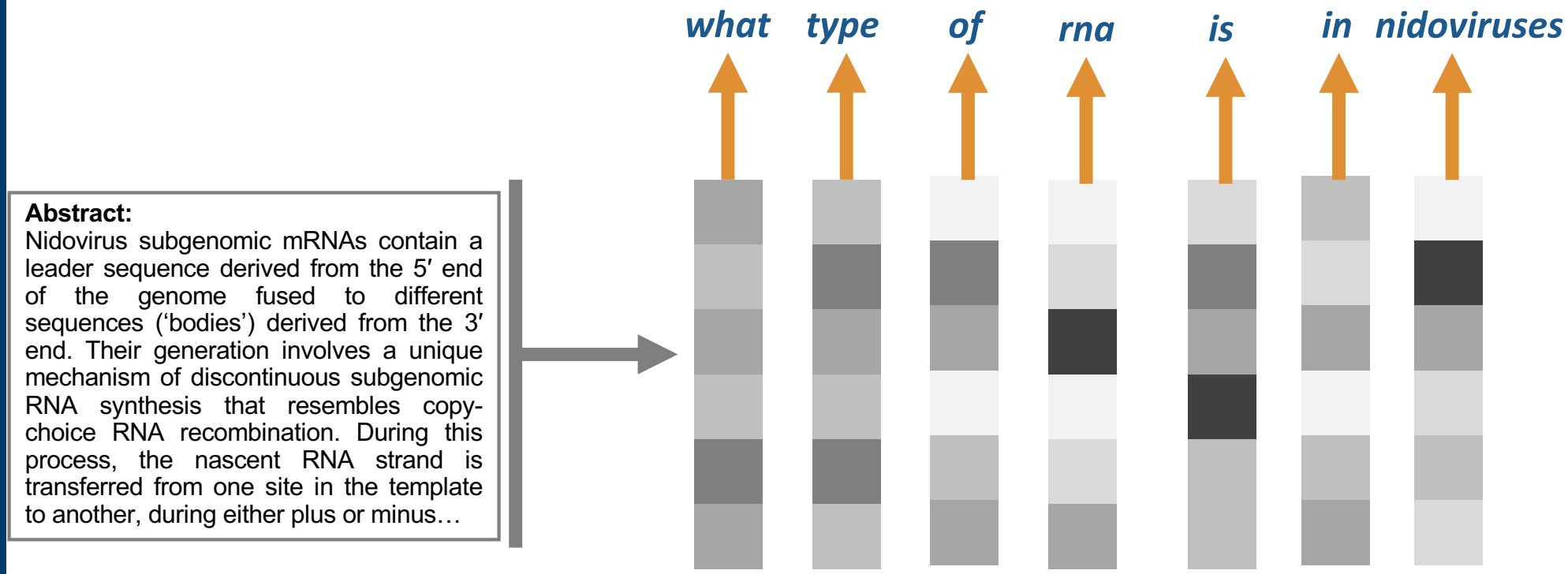


Step 2: Text iteratively generated from encoded document

e.g., via RNN or transformer, using beam search

Nogueira & Lin. From doc2query to docTTTTTquery. 2019.

Doc2Query



Step 2: Text iteratively generated from encoded document

e.g., via RNN or transformer, using beam search

Nogueira & Lin. From doc2query to docTTTTTquery. 2019.

Doc2Query Practical

```
from pyterrier_doc2query import Doc2Query
doc2query = Doc2Query(out_attr="text", batch_size=8)
```

Example doc2query outputs

```
df.iloc[0]['text']
```

```
'OBJECTIVE: This retrospective chart review describes the epidemiology and
clinical features of 40 patients with culture-proven Mycoplasma pneumoniae
infections at King Abdulaziz University Hospital, Jeddah, Saudi Arabia. MET
HODS: Patients with positive M. pneumoniae cultures from respiratory specim
```

```
doc2query.transform(df).iloc[0]['text']
```

```
'what is culture confirmed mycoplasma where is mycoplasma pneumonia located
in saudi arabia? where is mycoplasma cultured'
```

Doc2Query Practical

Università di Pisa | University of Glasgow

```
dataset = pt.get_dataset("irds:cord19/trec-covid")
index_loc = "./index_path"
indexer = doc2query >> pt.IterDictIndexer(index_loc)
indexref = indexer.index(dataset.get_corpus_iter())
```

TODO: performance on cord19

University of Glasgow | Università di Pisa

ECIR 2021

Nearest Neighbour Search

PART 4B

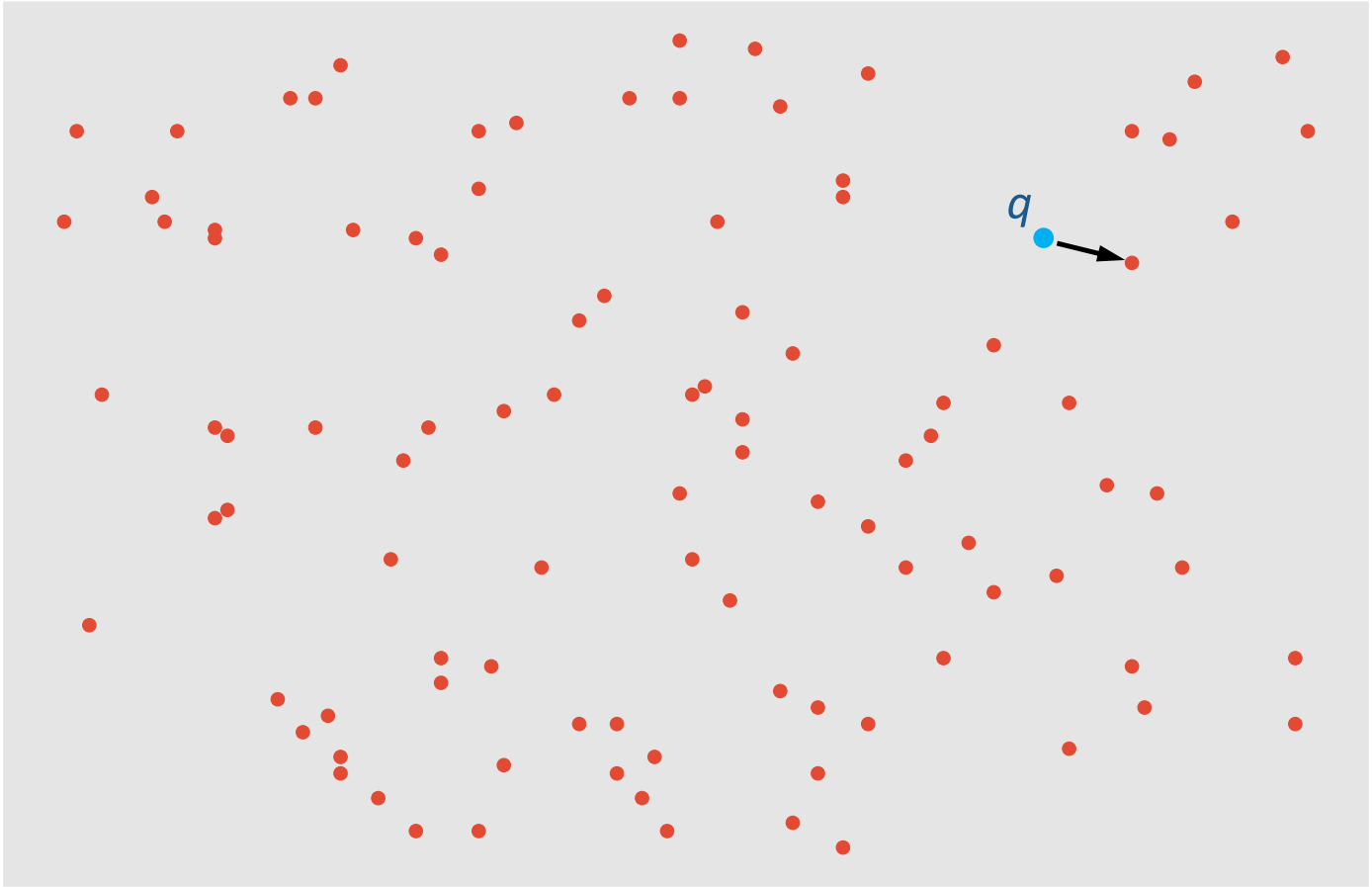
Dense Retrieval

- 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 NN search, then dense retrieval

Nearest Neighbour

Università di Pisa | University of Glasgow

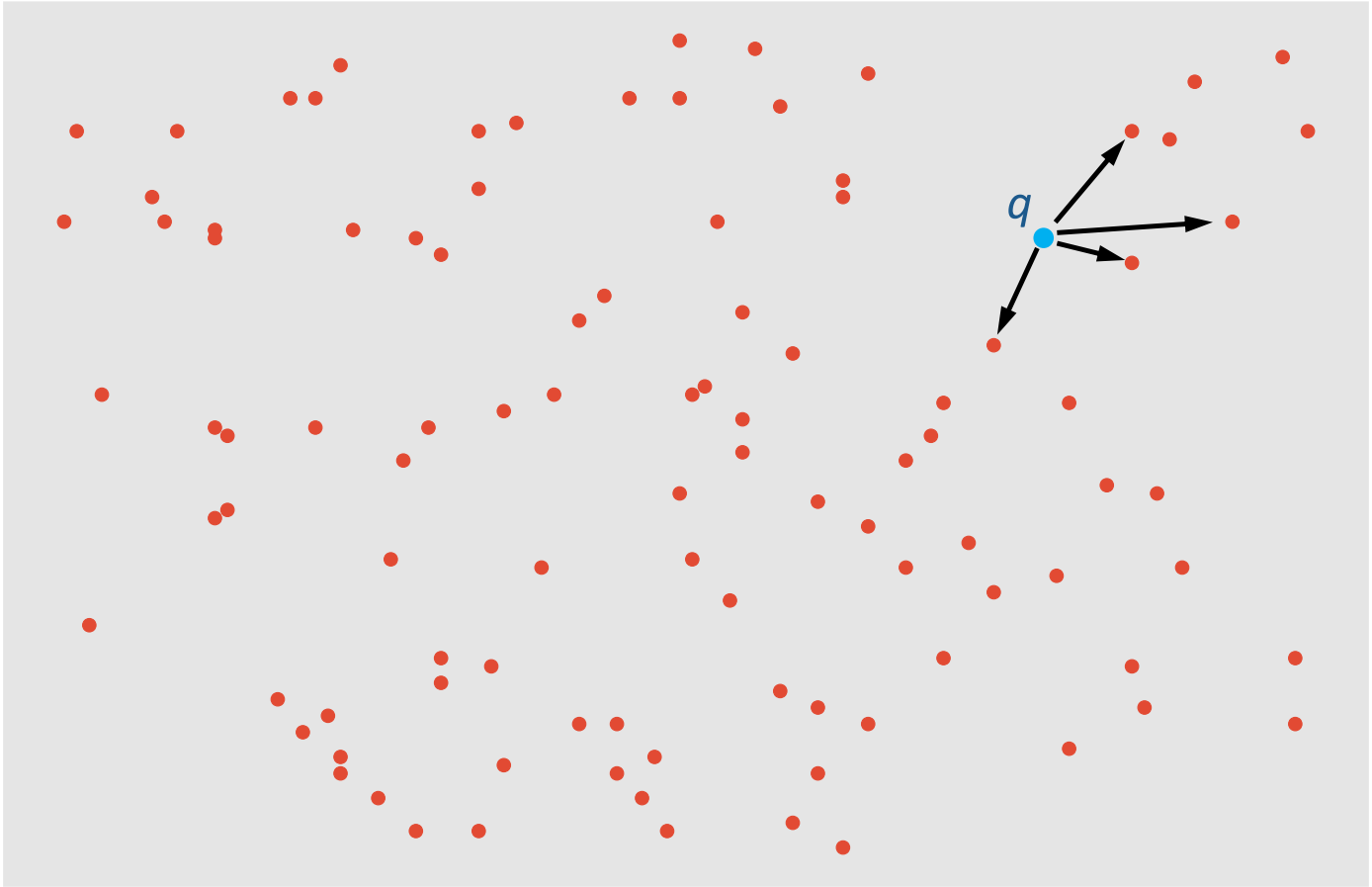
Complexity O(n)



k Nearest Neighbours

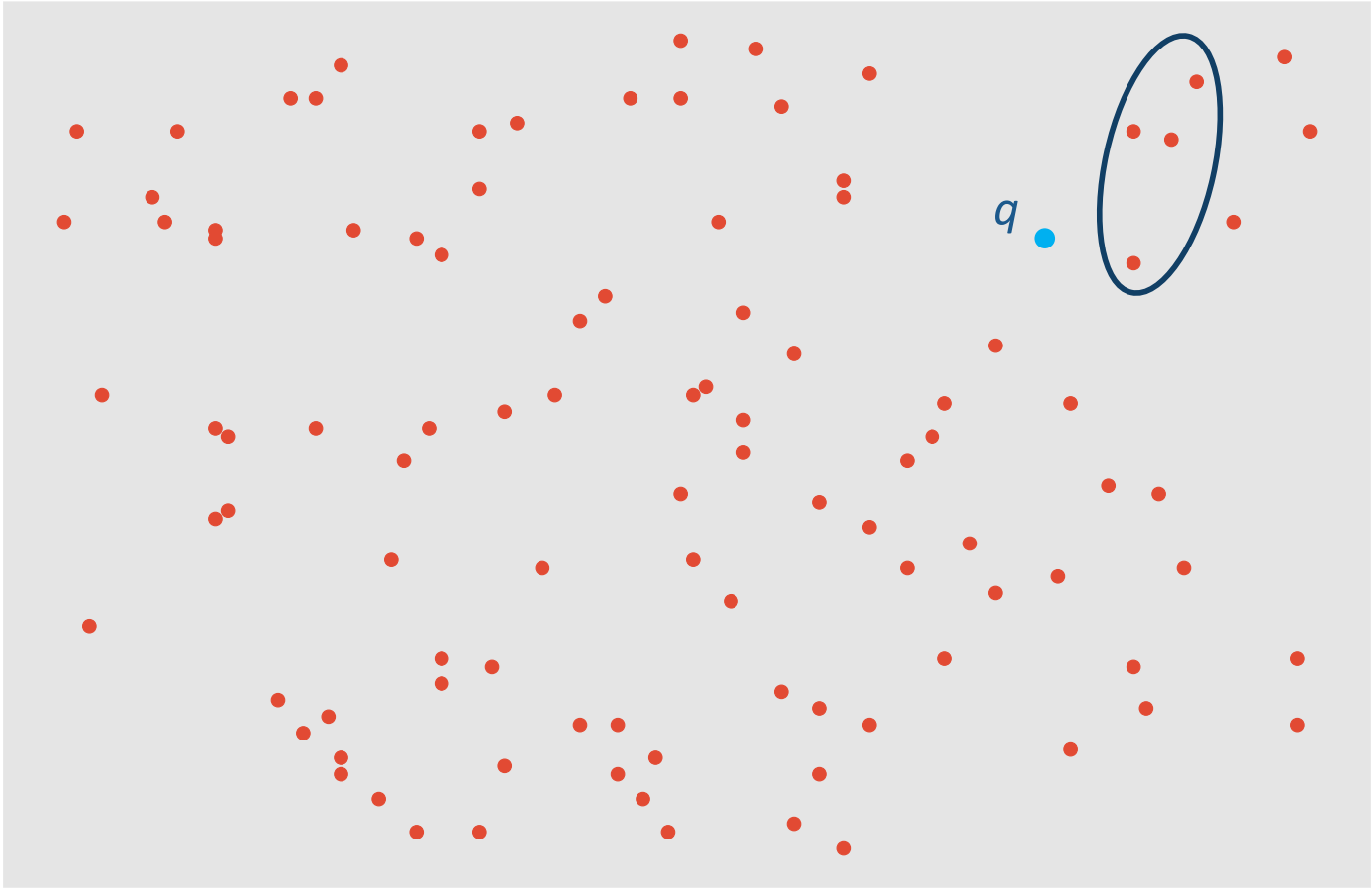
Università di Pisa | University of Glasgow

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

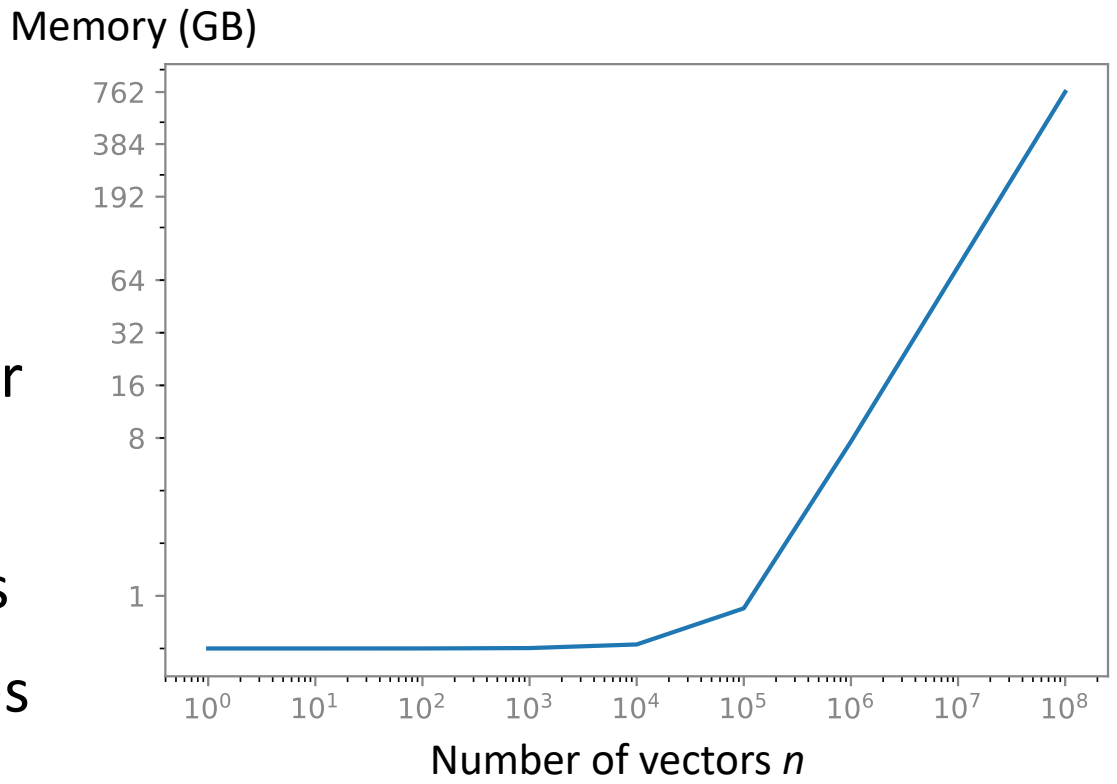
Università di Pisa | University of Glasgow

- 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

Storage Costs (I)

- Lossless storage
- d = 2048
- 8 KB storage per vector
- 1GB ~ 130K images
- 8GB ~ 1 Million images
- 7.6TB ~ 1 Billion images

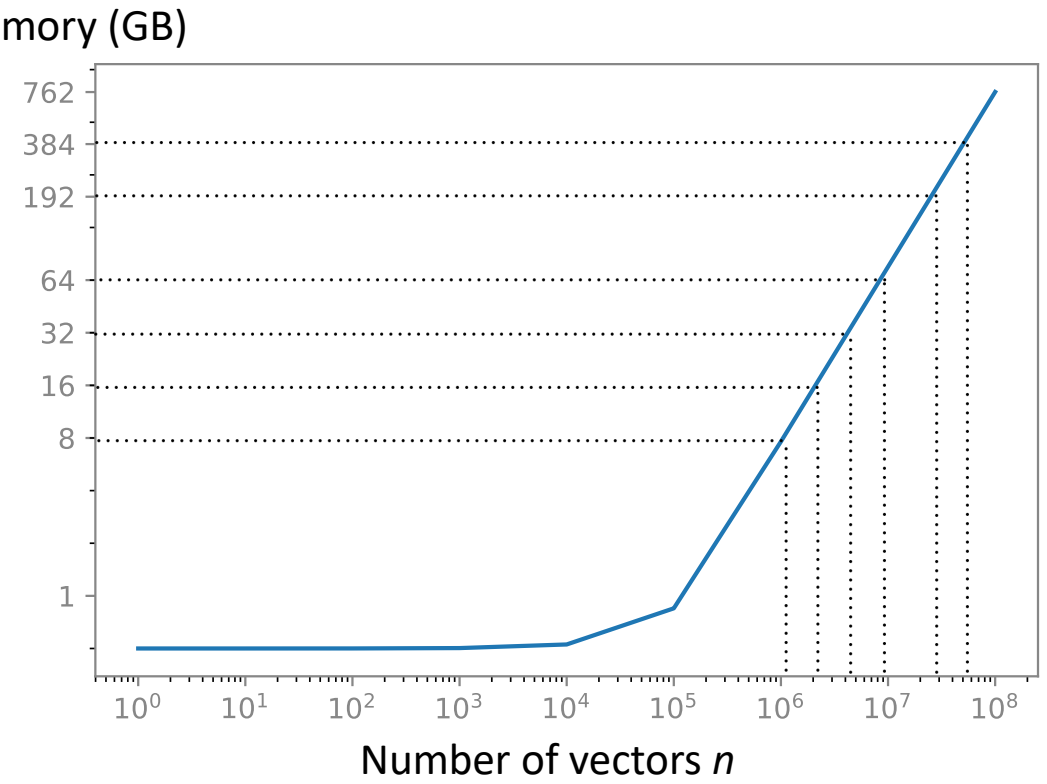


Slide from: https://lou.dev/talks/2018-08-09-lopq

Storage Costs (II)

Instance	RAM	# Images	Cost
m5.24xlarge	384	50.331.648	3.363 USD
m5.12xlarge	192	25.165.824	1.681 USD
m5.4xlarge	64	8.388.608	560 USD
m5.2xlarge	32	4.194.304	280 USD
m5.xlarge	16	2.097.152	140 USD
m5.large	8	1.048.576	70 USD

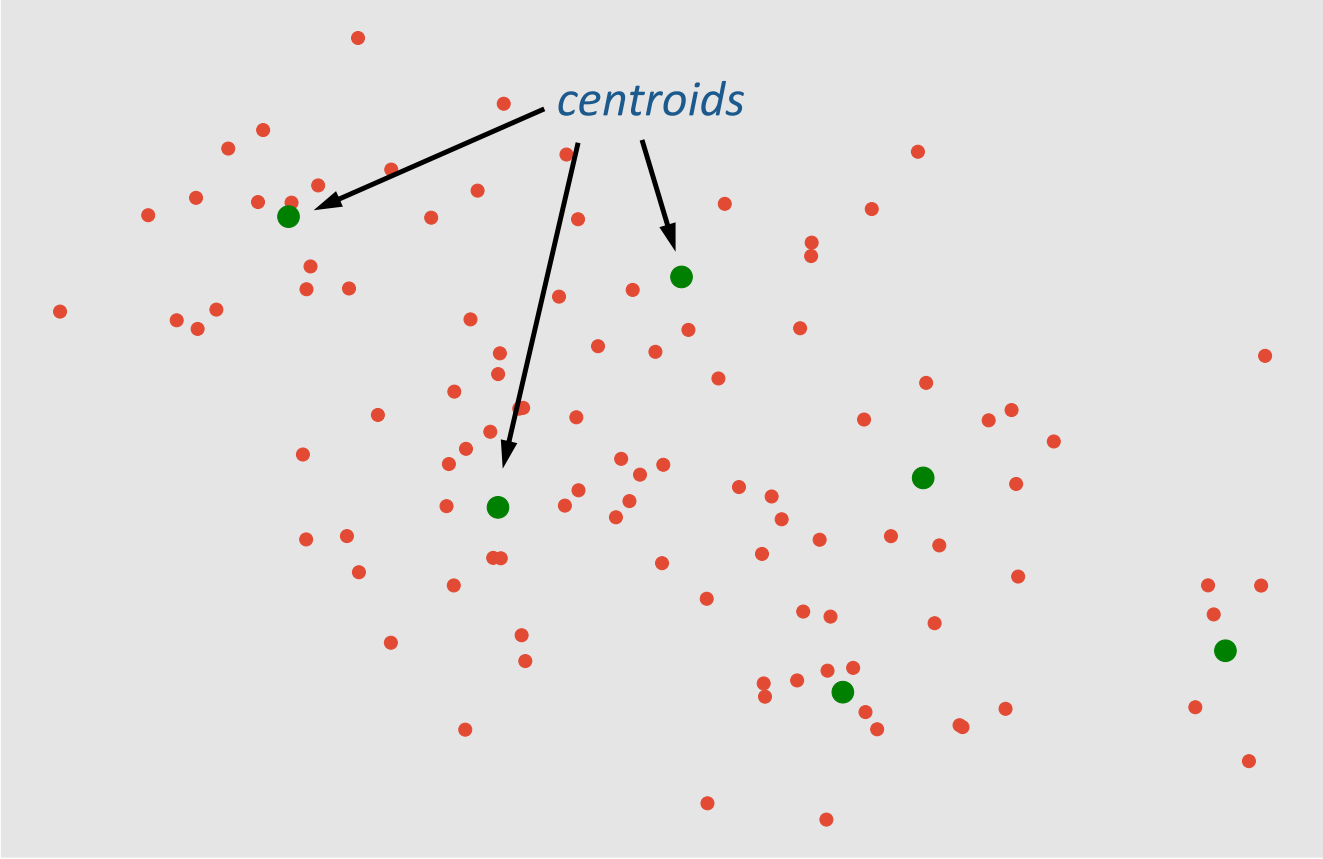
$0.07/month per 1000 images



Slide from: https://lou.dev/talks/2018-08-09-lopq

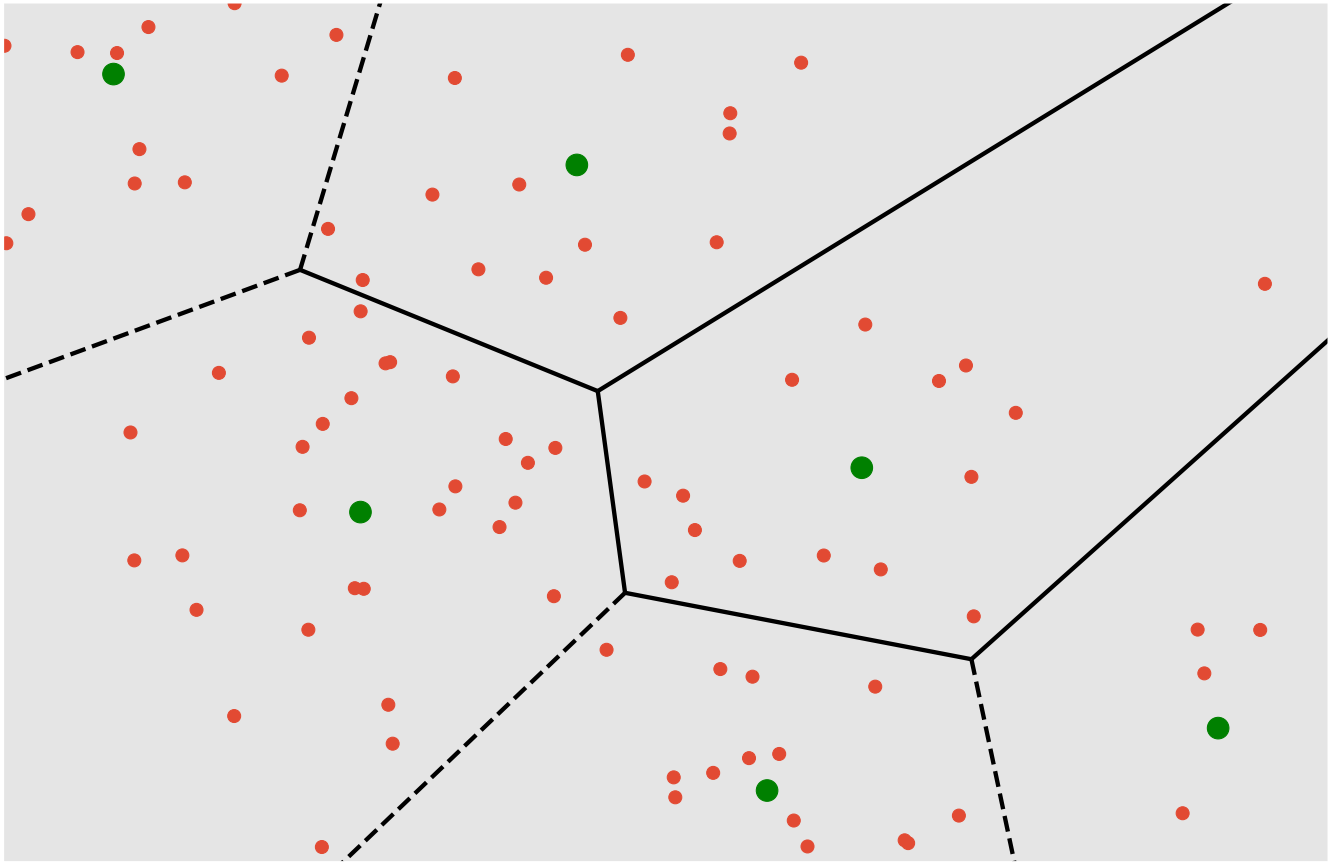
Quantization (I)

From *n* points to *k* centroids



Quantization (II)

Use k-Means to select centroids



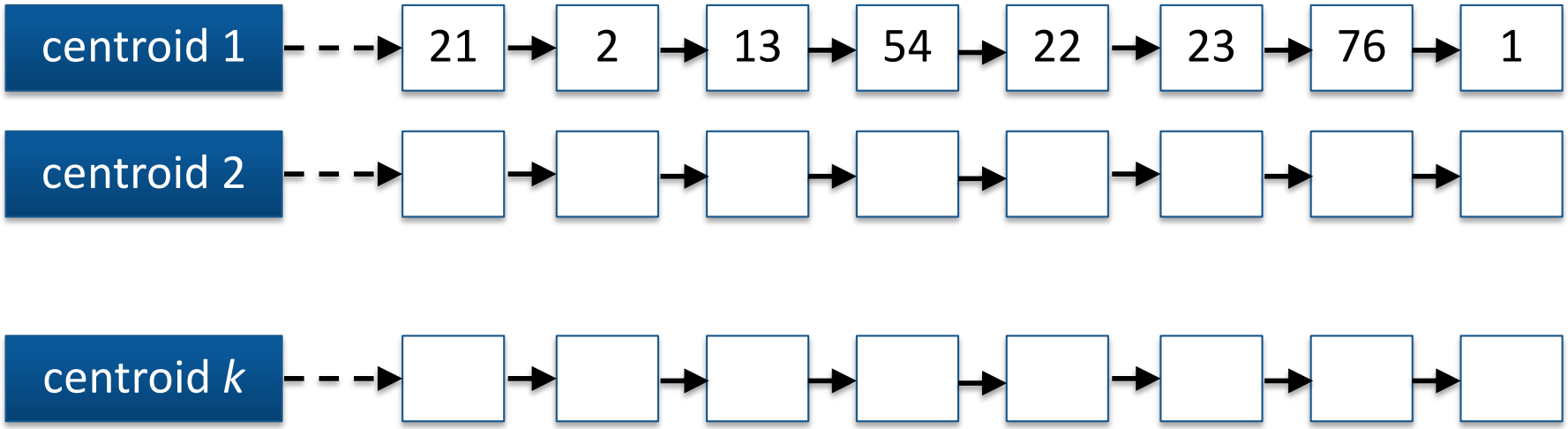
Quantization (III)

- 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 O(*n*) 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

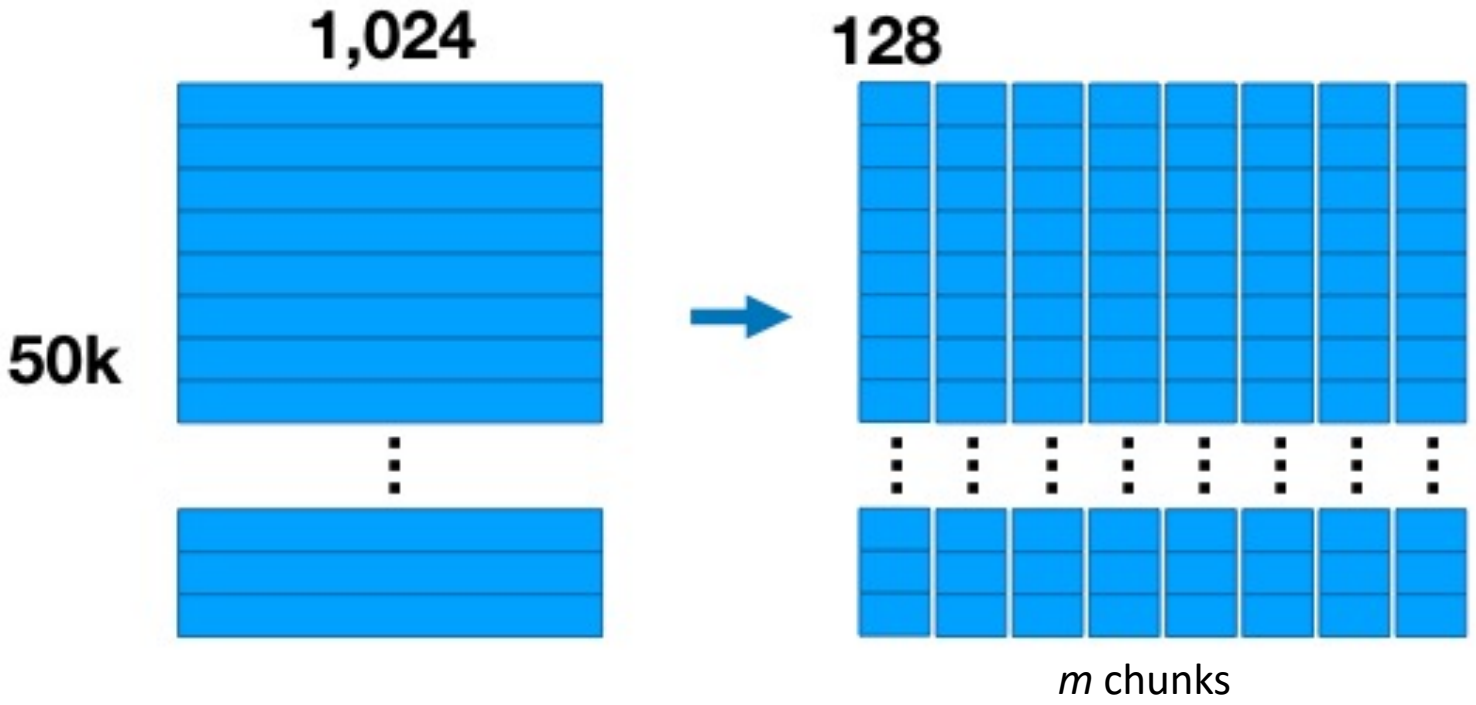


Image from: http://mccormickml.com/2017/10/13/product-quantizer-tutorial-part-1/

Product Quantization (II)

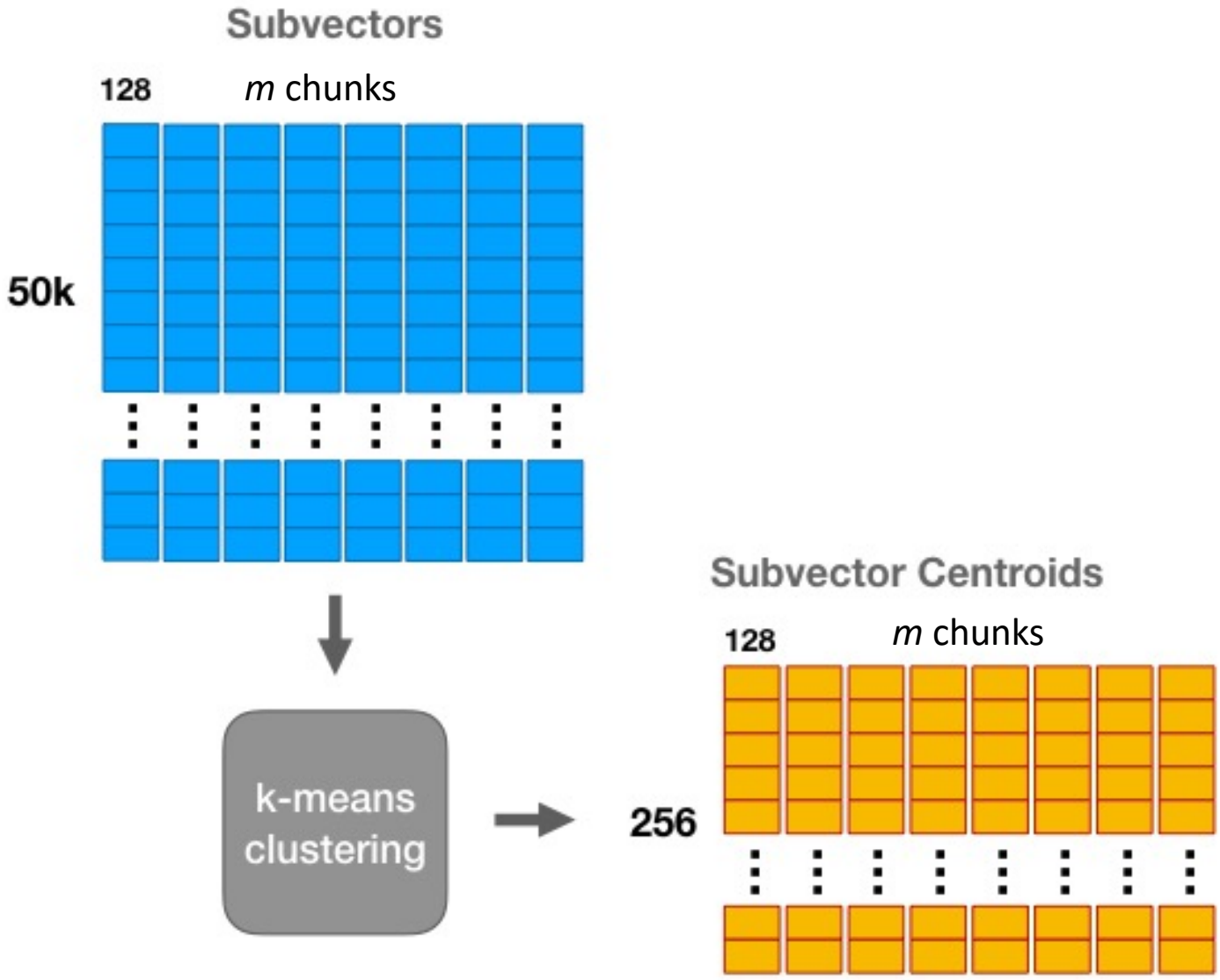


Image from: http://mccormickml.com/2017/10/13/product-quantizer-tutorial-part-1/

Product Quantization (III)

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

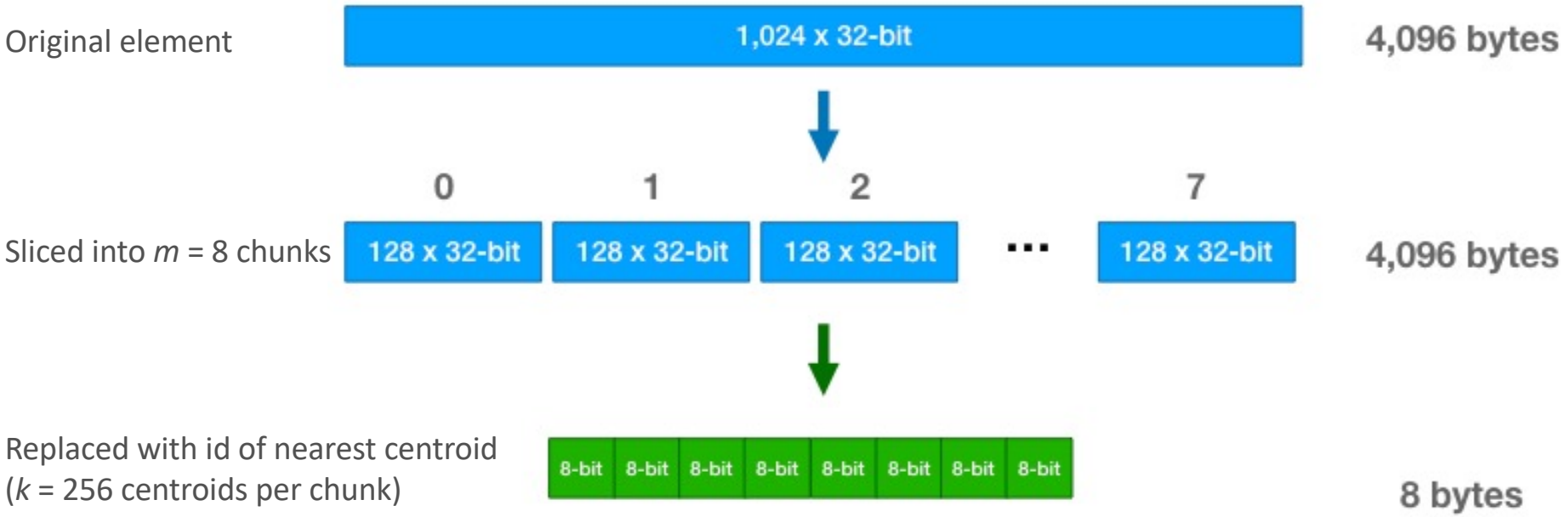


Image from: http://mccormickml.com/2017/10/13/product-quantizer-tutorial-part-1/

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

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

Dense Retrieval

PART 4C

Sparse vs Dense Retrieval (I)

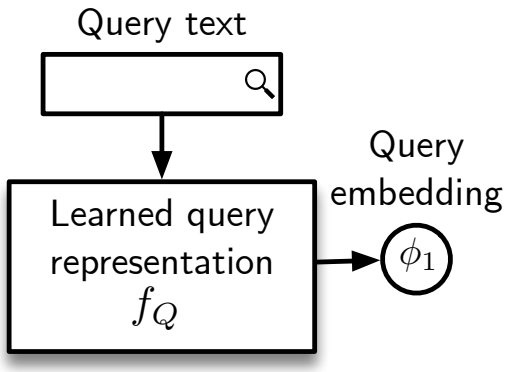
- **Sparse Retrieval**
  - Every document is represented by a single score for each term in the lexicon V (equal to 0 for terms not appearing in the document)
  - A document is a $|V|$-dimensional vector with many 0s, i.e., sparse
  - A query-document relevance score is computed as the sum of the score 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
  - Limited support for proximity signals

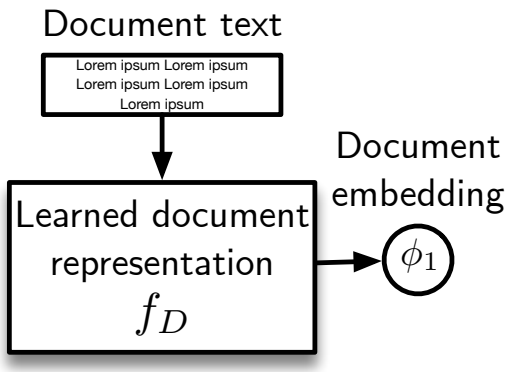
Sparse vs Dense Retrieval (II)

- **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
  - Scalability

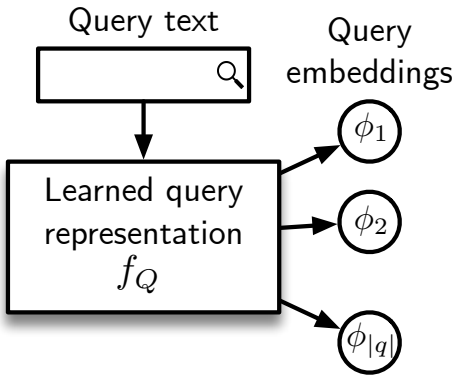
Learning Representations

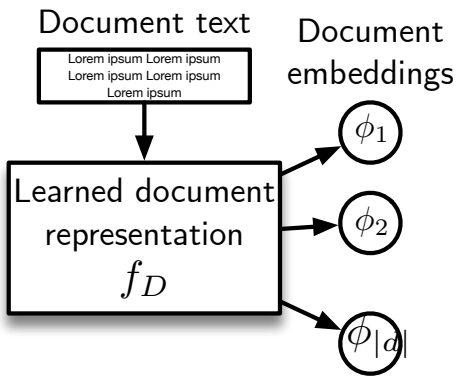
- **Single Representation (DPR, ANCE)**





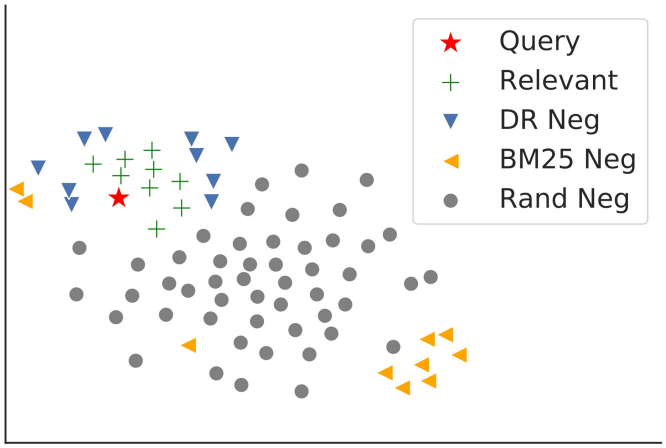
- **Multiple Representation (ColBERT)**

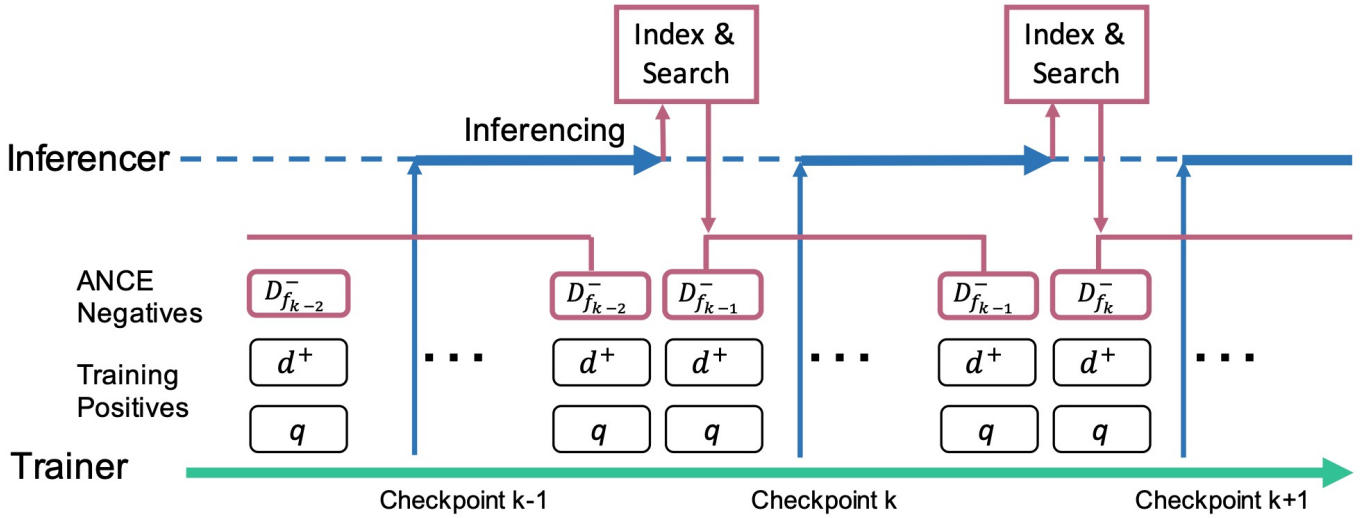




Approximate (nearest neighbor) Negative Contrastive Learning

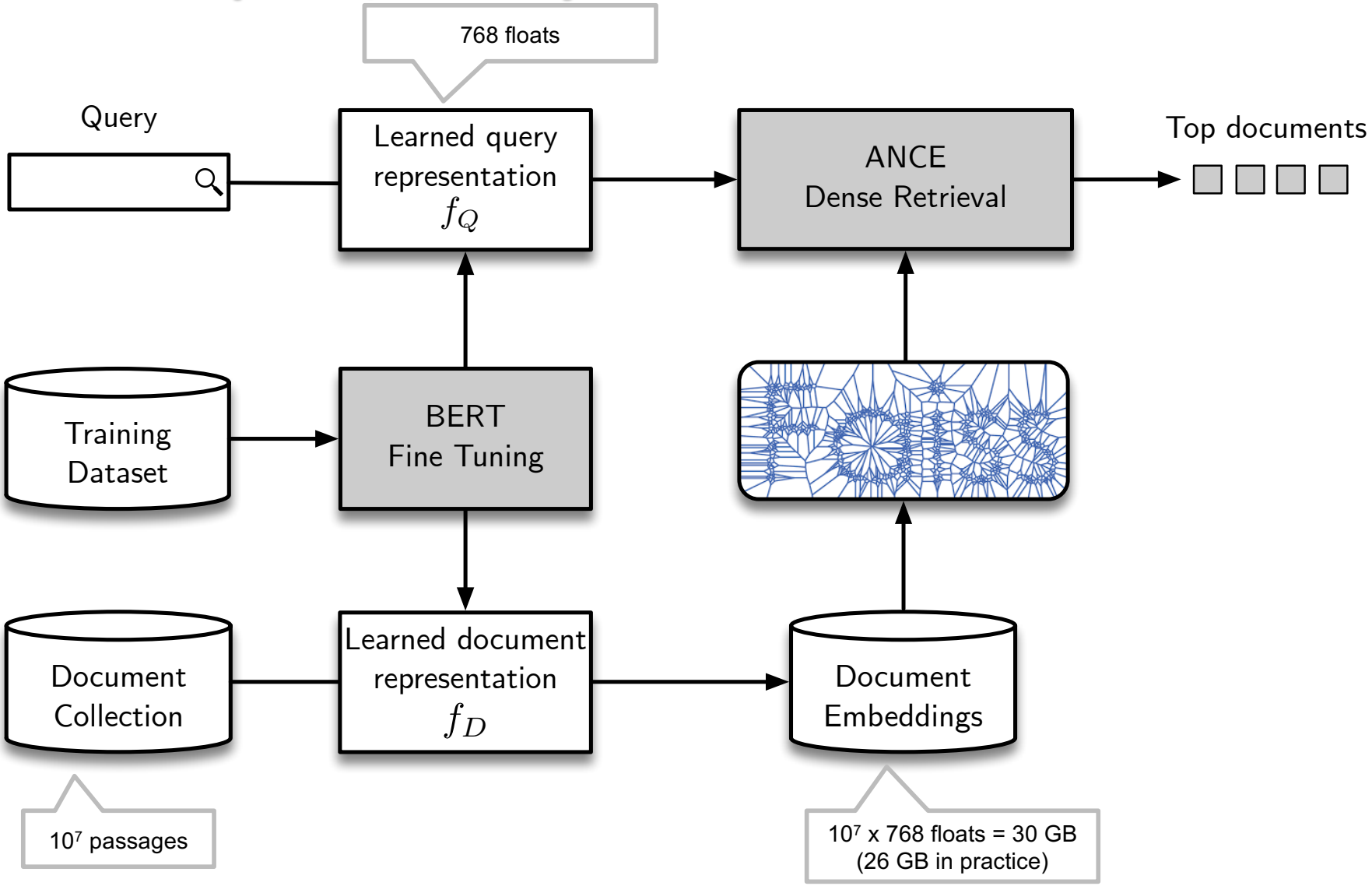
- Fully learnable representation
- Support for (approximate) NN search
- Problem: negative examples are usually drawn from a (classical) first stage
- How do we generate global negatives?
- Use the model learned so far!





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

Exact NN Dense Retrieval (DPR, ANCE)

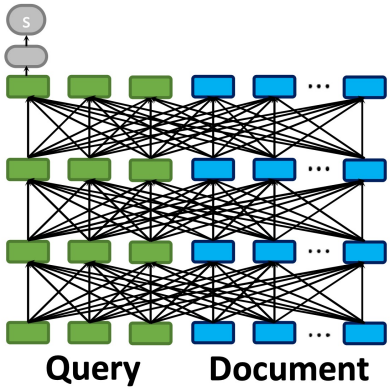


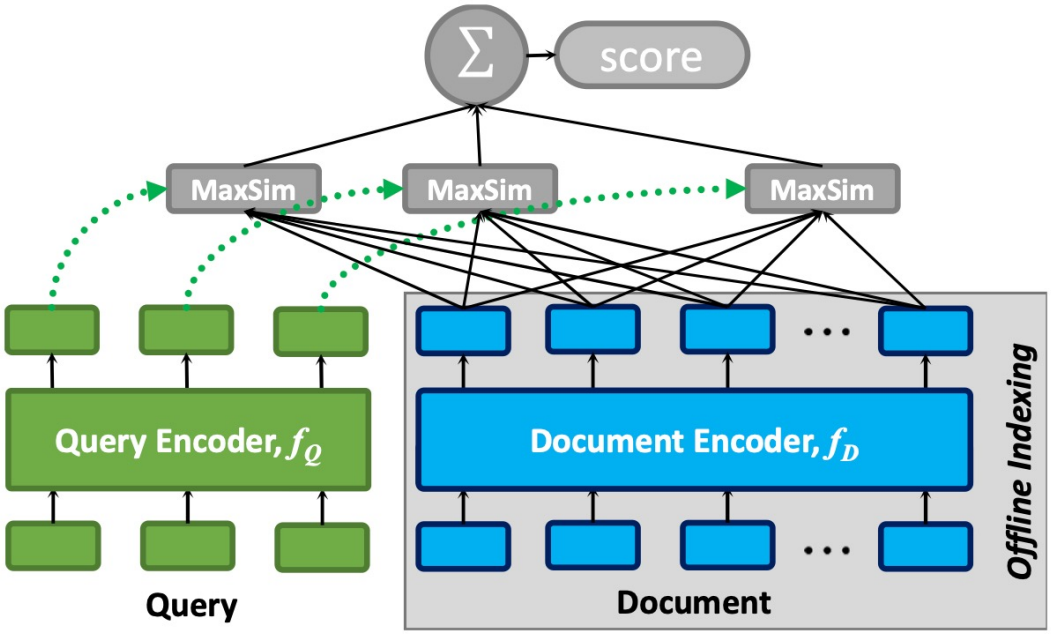
Exact NN Dense Retrieval (DPR, ANCE)

	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

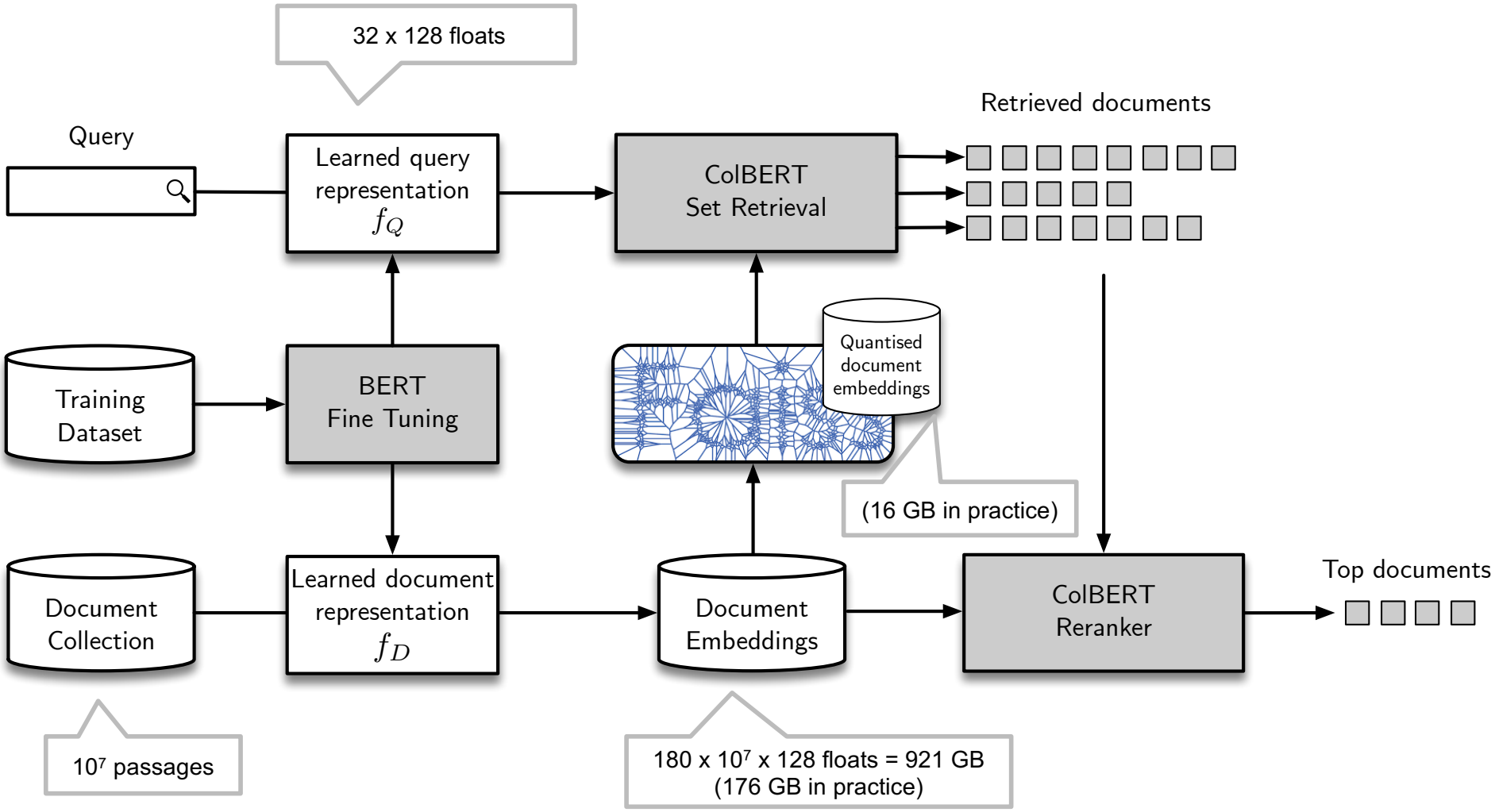
ColBERT: Recap





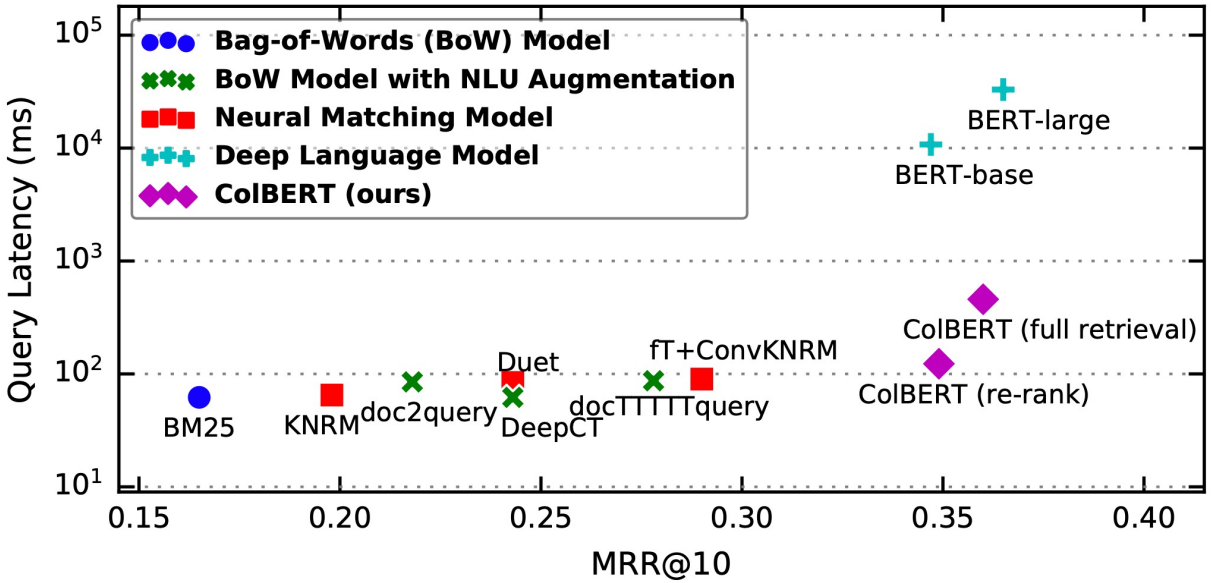
Images from: https://arxiv.org/abs/2004.12832

Approximate NN Dense Retrieval ColBERT



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

ANCE Example

- Indexing

```
dataset = pt.get_dataset("irds:vaswani")
import pyterrier_ance
indexer = pyterrier_ance.ANCEIndexer("/path/to/checkpoint", "/path/to/anceindex")
indexer.index(dataset.get_corpus_iter())
```

- Retrieval

```
anceretr = pyterrier_ance.ANCERetrieval("/path/to/checkpoint", "/path/to/anceindex")
```

- Experiment

```
pt.Experiment(
    [anceretr],
    dataset.get_topics(),
    dataset.get_qrels(),
    eval_metrics=["map"]
)
```

ColBERT Example

- Indexing

```
from pyterrier_colbert.indexing import ColBERTIndexer
indexer = ColBERTIndexer("/path/to/checkpoint.dnn", "/path/to/index", "index_name")
indexer.index(dataset.get_corpus_iter())
```

- End-to-end Retrieval

```
from pyterrier_colbert.ranking import ColBERTFactory
pytcolbert = ColBERTFactory("/path/to/checkpoint.dnn", "/path/to/index", "index_name")
dense_e2e = pytcolbert.set_retrieve() >> pytcolbert.index_scorer()
```

- Sparse retrieval + ColBERT re-ranker only

```
bm25 = pt.BatchRetrieve(terrier_index, wmodel="BM25", metadata=["docno", "text"])
sparse_colbert = bm25 >> pytcolbert.text_scorer()
```

- Experiment

```
pt.Experiment(
    [bm25, sparse_colbert, dense_e2e]
    dataset.get_topics(),
    dataset.get_qrels(),
    measures=["map", "ndcg_cut_10"],
    names=["BM25", "BM25 >> ColBERT", "Dense ColBERT"]
)
```

From passages to documents

Università di Pisa | University of Glasgow

ANCE and ColBERT (as well as DeepCT and doc2query) 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

```
# Doc2query
indexer = pt.text.sliding() >> doc2query >> pt.IterDictIndexer()
indexer.index(dataset.get_corpus_iter())

# DeepCT
indexer = pt.text.sliding() >> deepct >> pt.IterDictIndexer()
indexer.index(dataset.get_corpus_iter())

# ANCE
indexer = pt.text.sliding() >> pyt_ance.ANCEIndexer()
indexer.index(dataset.get_corpus_iter())
```

```
bm25_doc2query = pt.BatchRetrieve() >> pt.text.max_passage()
bm25_deepct = pt.BatchRetrieve() >> pt.text.max_passage()
ance = pty_ance.ANCERetrieval() >> pt.text.max_passage()
colbert = pyt_cb.ranking.ColBERTFactory( ).end_to_end() >> pt.text.max_passage()
```

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 **modify the content** stored in inverted indices
  - Improve the frequency of existing ”important” terms
  - Introduce new ”important” terms not present in a document
- PyTerrier **provides plugins for** DeepCT and Doc2Query

- 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

Bibliography

- Zhuyun Dai and Jamie Callan. 2019. Deeper text understanding for IR with contextual neural language modeling. In Proc. SIGIR
- Zhuyun Dai and Jamie Callan. 2019. Context-aware sentence/passage term importance estimation for first stage retrieval. arXiv:1910.10687
- Zhuyun Dai and Jamie Callan. 2020. Context-aware document term weighting for ad-hoc search. In Proc. WWW
- Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. 2019. Multi-stage document ranking with BERT. arXiv:1910.14424
- Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. 2019. Document expansion by query prediction. arXiv:1904.08375
- Samuel Humeau, Kurt Shuster, Marie-Anne Lachaux, and J. Weston. 2020. Poly-encoders: Transformer Architectures and Pre-training Strategies for Fast and Accurate Multi-sentence Scoring. In Proc. ICLR
- Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. arXiv:1702.08734
- Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. arXiv:2004.04906
- Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In Proc. SIGIR.
- Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent Retrieval for Weakly Supervised Open Domain Question Answering. In Proc. ACL
- Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2020. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. arXiv:2007.00808
- Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2020. Pretrained Transformers for Text Ranking: BERT and Beyond. arXiv:2010.06467

Questions

Practical Time

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

- https://github.com/terrier-org/ecir2021-tutorial
- Press the 

link for each notebook to start a Colab session

Timings:

- Practical – in breakout rooms – until 18:15

When you are finished…

- Please complete our feedback quiz
  - https://forms.office.com/r/2WbpLiQmWV

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