NLP and the Web – WS 2024/2025

Lecture 6
Information Retrieval IV



Dr. Thomas Arnold
Hovhannes Tamoyan
Kexin Wang

Ubiquitous Knowledge Processing Lab
Technische Universität Darmstadt

Syllabus (tentative)

Nr.	Lecture
01	Introduction / NLP basics
02	Foundations of Text Classification
03	IR – Introduction, Evaluation
04	IR – Word Representation
05	IR – Transformer/BERT
06	**IR – Dense Retrieval**
07	IR – Neural Re-Ranking
08	LLM – Language Modeling Foundations
09	LLM – Neural LLM, Tokenization
10	LLM – Adaption, LoRa, Prompting
11	LLM – Alignment, Instruction Tuning
12	LLM – Long Contexts, RAG
13	LLM – Scaling, Computation Cost
14	Review & Preparation for the Exam

Recap from last week:

1. Word Embeddings
   - Byte-Pair-Encoding
2. Simple Neural Techniques
   - Convolutional NN
   - Recurrent NN
   - Encoder-Decoder Architecture
3. Transformer Architecture
   - BERT Pre-Training

Today

IR – Dense Retrieval & Destillation

1 **Dense Retrieval**

- The Bert-Dot model
- (Approximate) nearest neighbor search

2 Knowledge Destillation

- Cross-architecture: KL Divergence

TU WIEN

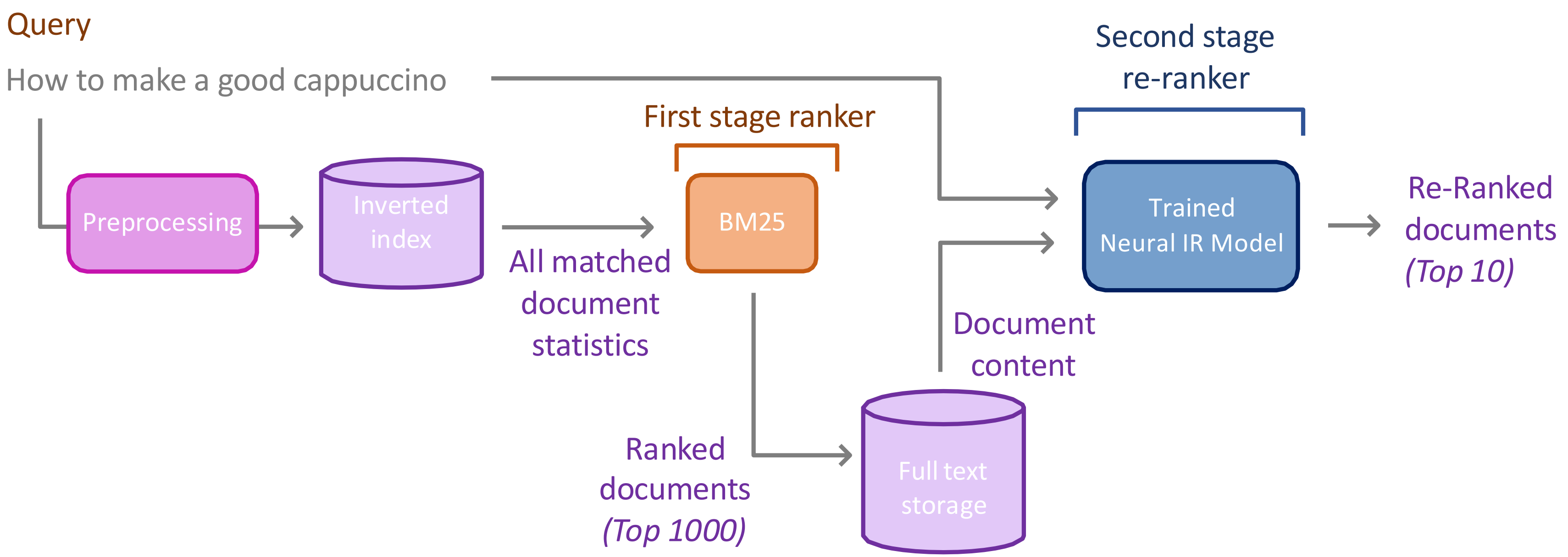
Based on:
https://github.com/sebastian-hofstaetter/teaching/tree/master/introduction-to-information-retrieval/lectures

Neural Methods for IR Beyond Re-ranking

- Re-Ranking depends on a candidate selection (bottleneck)
  - How to bring neural advances in this first-stage phase
- Today we look at dense retrieval as (inverted index) BM25 alternative
- Many other neural approaches to improve first-stage retrieval:
  - **Doc2query**: Document expansion with query text that would semantically match the document. Exists in both BERT and T5 variants. Then index the expanded documents with BM25
  - **DeepCT**: Assign term weights based on BERT output during indexing -> retrieval with inverted index & BM25
  - **COIL**: Fuses contextual vectors into an inverted index structure, for faster lookup of semantic matches

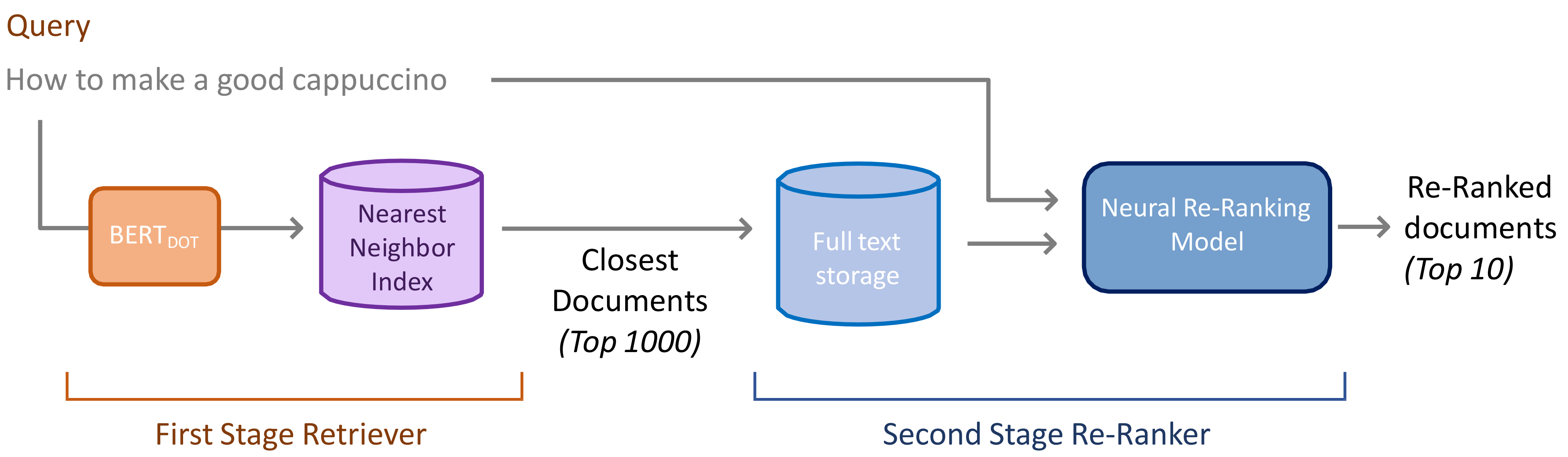
Neural Re-Ranking

- Re-rankers: They change the ranking of a pre-selected list of results
  - Same interface as classical ranking methods: $\mathrm{score}(q, d)$
- Query workflow:



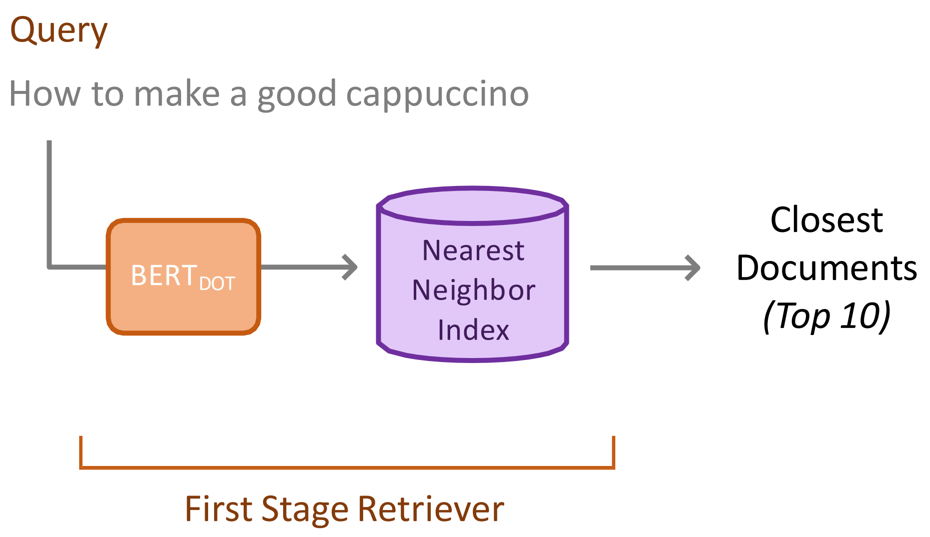
Dense Retrieval (with Re-Ranking)

- Dense retrieval replaces the traditional first stage
  - Using a neural encoder & nearest neighbor vector index
  - Can be used as part of a larger pipeline



Standalone Dense Retrieval

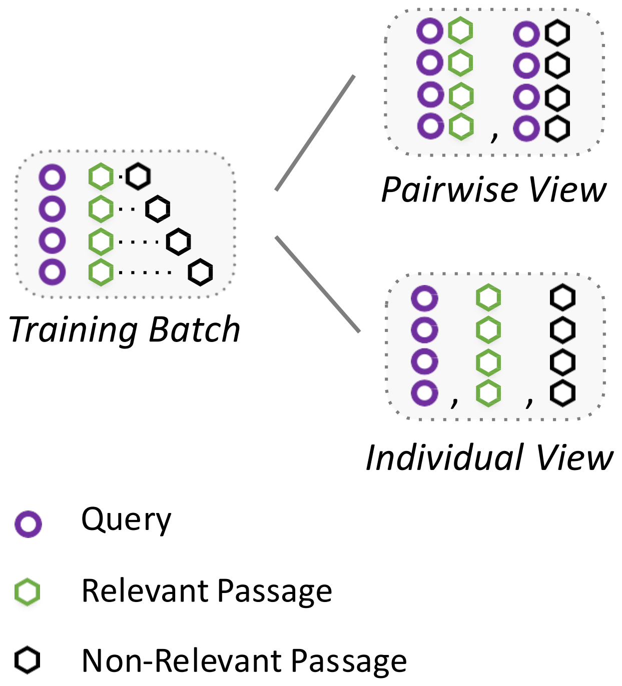
- If dense retrieval is effective enough for our goals:
  - We can also use it as a standalone solution
  - Much faster + less complexity if we remove re-ranking stage



Training

- Neural IR models are typically trained with triples (pairwise +,-)
  - Triple: 1 query, 1 relevant, 1 non-relevant document
  - Generate embeddings for query, relevant doc, non-relevant doc
  - Loss function: Maximize margin between rel/non-rel document
- All model components are trained end-to-end
  - Of course we could decide to freeze some parts for more efficient training

Creating Training Batches



- We form a batch by sampling as many triples as is allowed by the GPU memory
  - Typical batch size: 16-128
  - We mix different queries together
  - Depending on the model we need to create query-passage pairs or run each of the three sequences individually through the model
- We run a backward pass & gradient update per batch
- Sequency inputs come as a single matrix, so we need to pad different length inputs

Sampling Non-Relevant Passages

- Most collections only come with judgements of relevant (or false-positive selections from other models) and not truly non-relevant judgements
  - It doesn’t make sense to spend resources annotating random pairs
- We need to tell the model what is non-relevant
  - Simple procedure to sample non-relevant passages:
    - Run BM25 and get the top-1000 results per training query and randomly select a few of those results as non-relevant
    - The non-relevant selections provide some signal
      (as there must be at least some lexical overlap)
    - But mostly non-relevant passages -> works pretty good in practice
    - A bit of noise is good (we don’t know the degree of non-relevance, but that’s ok)

Loss Function

- Choice of different methods that aim to maximize the margin between rel & non-rel document
  - Plain Margin Loss:

    $$\text{loss} = \max(0, s_{nonrel} - s_{rel} + 1)$$

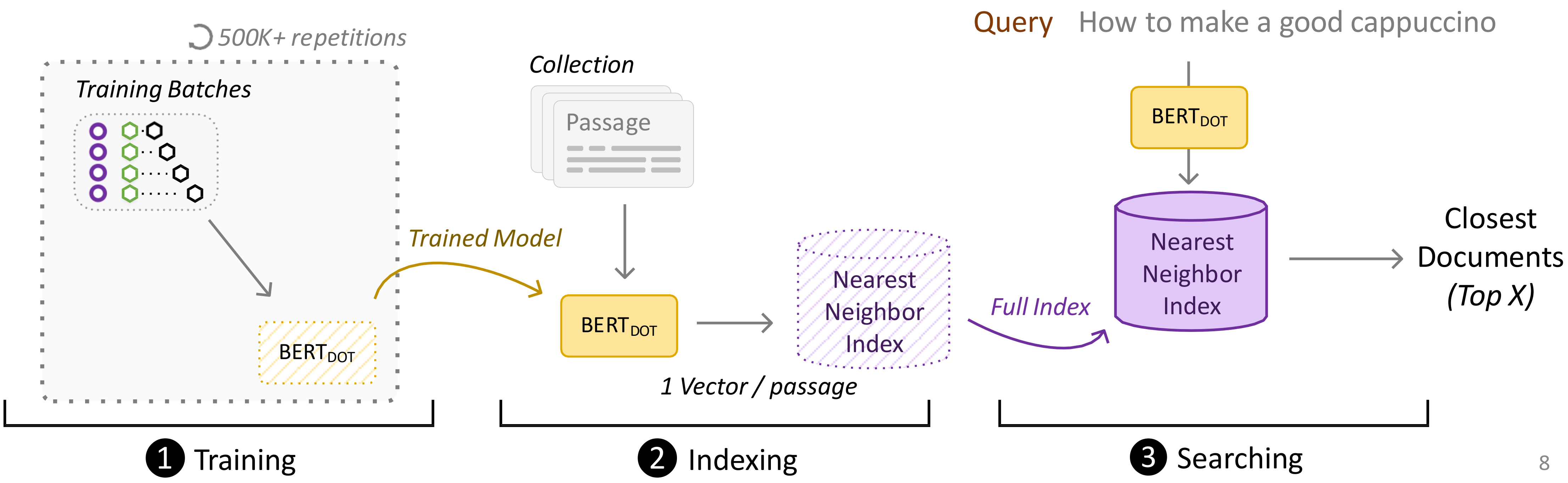
    - Native support in PyTorch: `torch.nn.MarginRankingLoss()`
  - RankNet:

    $$\text{loss} = \text{BinaryCrossEntropy}(s_{rel} - s_{nonrel})$$

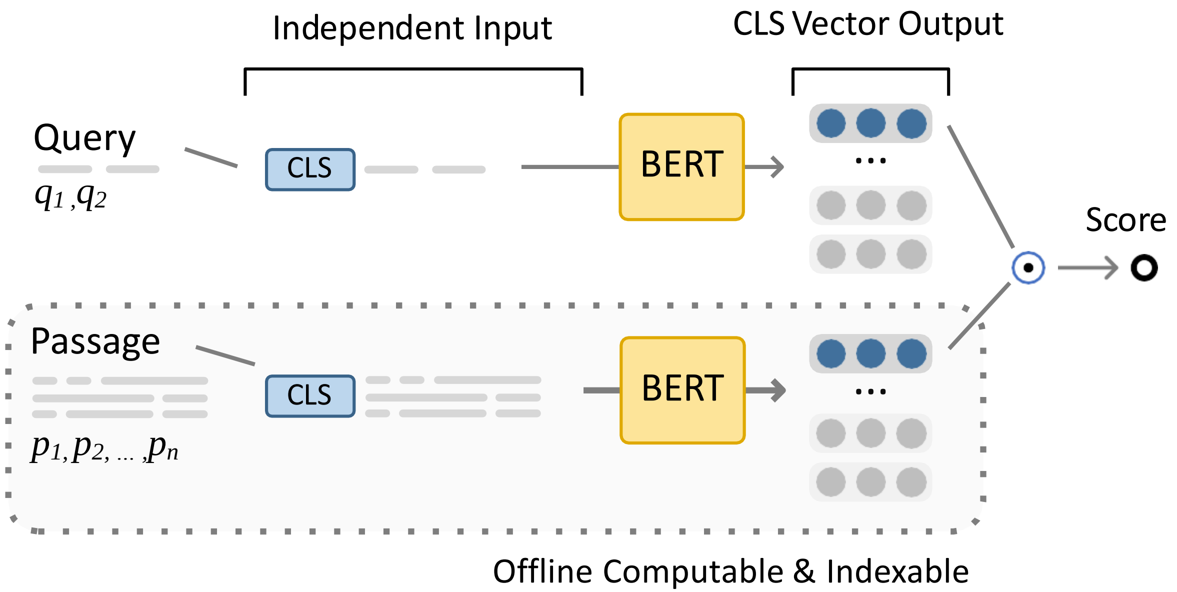
- Both losses assume binary relevance

Dense Retrieval Lifecycle

- 3 major phases in the dense retrieval lifecycle
  - Each comes with several complex choices and required techniques
  - Could skip ❶ if we use a pre-trained model



$BERT_{DOT}$ Model



- Passages and queries are compressed into a single vector
  - Passages are completely independent -> moves most computation into the indexing phase
  - Only need query encoding at runtime
- Relevance is scored with a dot-product
  - Cosine-sim variants also exist
  - This allows easy use of an (approximate) nearest neighbor index

BERT_{DOT}

- Simple formula (as long as we abstract BERT):

Encoding

$$\hat{q} = \text{BERT}([CLS]; q_{1..n})_{CLS}$$

$$\hat{p} = \text{BERT}([CLS]; p_{1..m})_{CLS}$$

Independent computation

Matching

$$s = \hat{q} \cdot \hat{p}$$

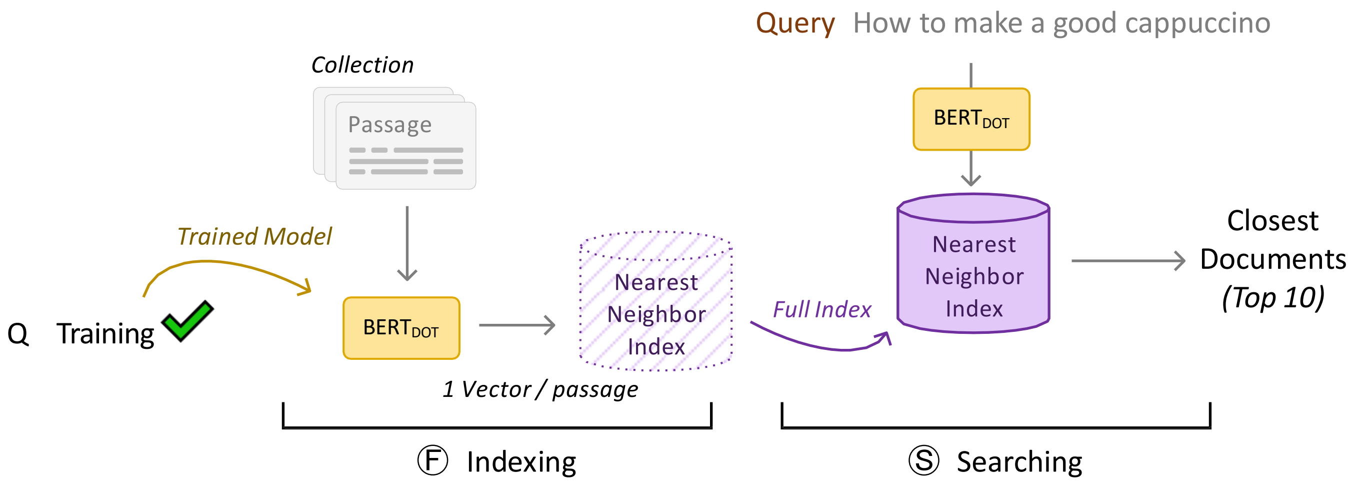
Can be done "outside" the model
(with a nearest neighbor library)

- Optional compression of $\hat{q}, \hat{p}$ with a single linear layer (to reduce dimensionality)

$q_{1..n}$	Query tokens
$p_{1..m}$	Passage tokens
BERT	Pre-trained BERT model
[CLS]	Special tokens
x_{CLS}	Pool the CLS vector
s	Output score

Nearest Neighbor Search

- Once we have a trained DR model,
  we encode every passage in our collection
  - We save passages in an (approximate) nearest neighbor index
- During search we encode the query on the fly
  and search for nearest neighbor vectors in the passage index



NN Search: GPU Brute-Force

- Retrieving the top-1K from 9 million vectors is fast
  - We need to do 9M dot-products (a very big matrix multiplication) with 768 dim. vectors
- GPUs are made for this
  - Vectors must fit in GPU memory
  - 70ms latency / query
  - Incredible scale when increasing the batch size
  - Using a CPU this takes ~1 sec. / q

Table 1: Latency analysis of Top-1000 retrieval using our BERT_{DOT} retrieval setup for all MSMARCO passages using DistilBERT and Faiss (FlatIP) on a single Titan RTX GPU

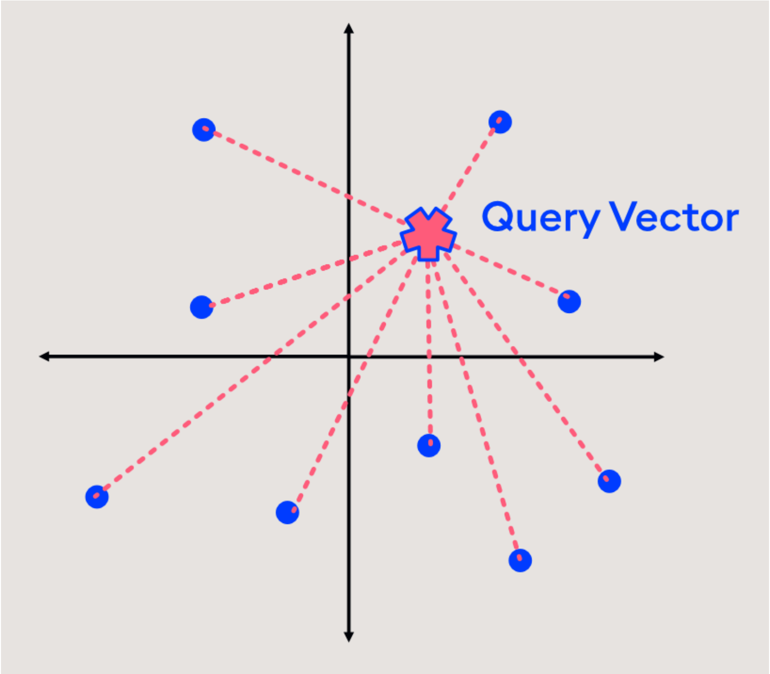
Batch Size	Q. Encoding Avg.	Q. Encoding 99^{th} Per.	Faiss Retrieval Avg.	Faiss Retrieval 99^{th} Per.	Total Avg.	Total 99^{th} Per.
1	8 ms	11 ms	54 ms	55 ms	64 ms	68 ms
10	8 ms	9 ms	141 ms	144 ms	162 ms	176 ms
2,000	273 ms	329 ms	2,515 ms	2,524 ms	4,780 ms	4,877 ms

Efficiently Teaching an Effective Dense Retriever with Balanced Topic Aware Sampling; Hofstätter et al. SIGIR 2021
https://arxiv.org/abs/2104.06967

Indexing Techniques: Flat Index

Flat Index = Brute Force

- No additional processing, using raw vector embeddings
- Calculates distance for each pair, slow
- Exhaustive search, best accuracy



Images from https://kdb.ai/learning-hub/articles/indexing-basics/

Indexing Techniques: Inverted File Index (IVF)

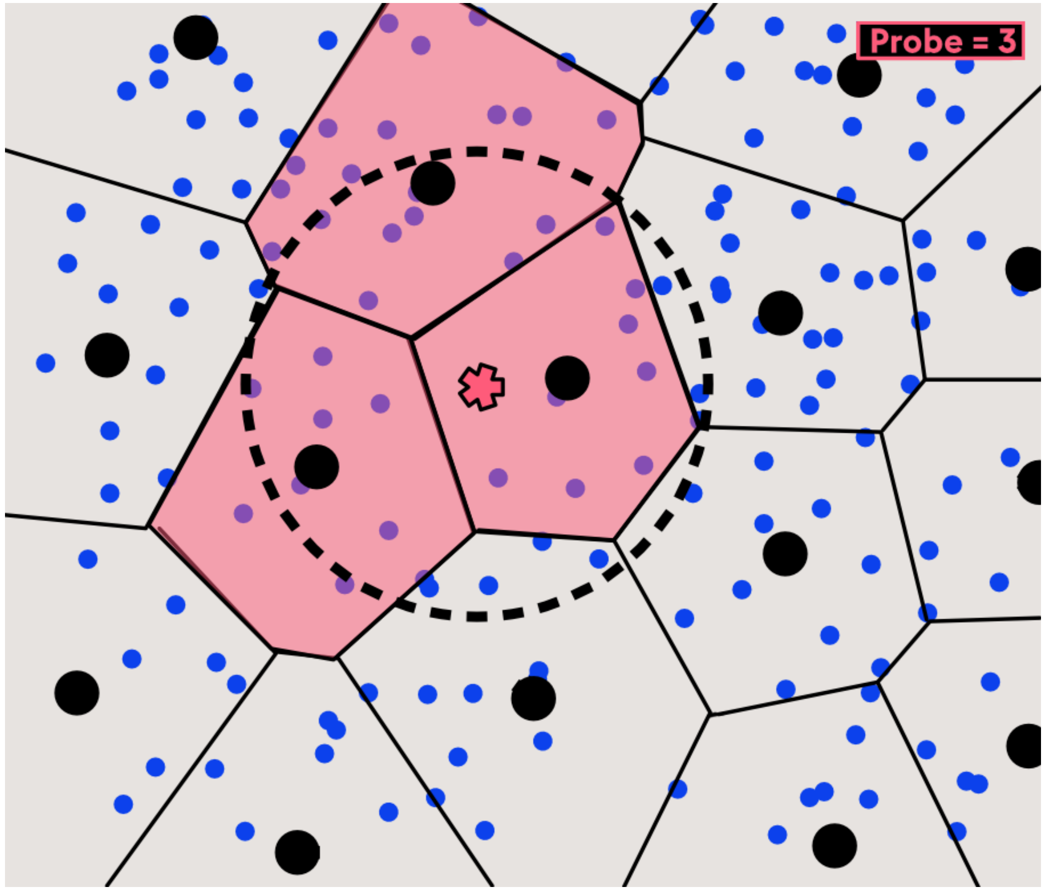
Partition the dataset into clusters

- Use clustering algorithm (e.g. k-means) to divide into k clusters
- Compute the centroids of each cluster
- For each cluster, store:
    - The centroid vector
    - An inverted index list of the vectors assigned to that cluster

Indexing Techniques: Inverted File Index (IVF)

Query time:

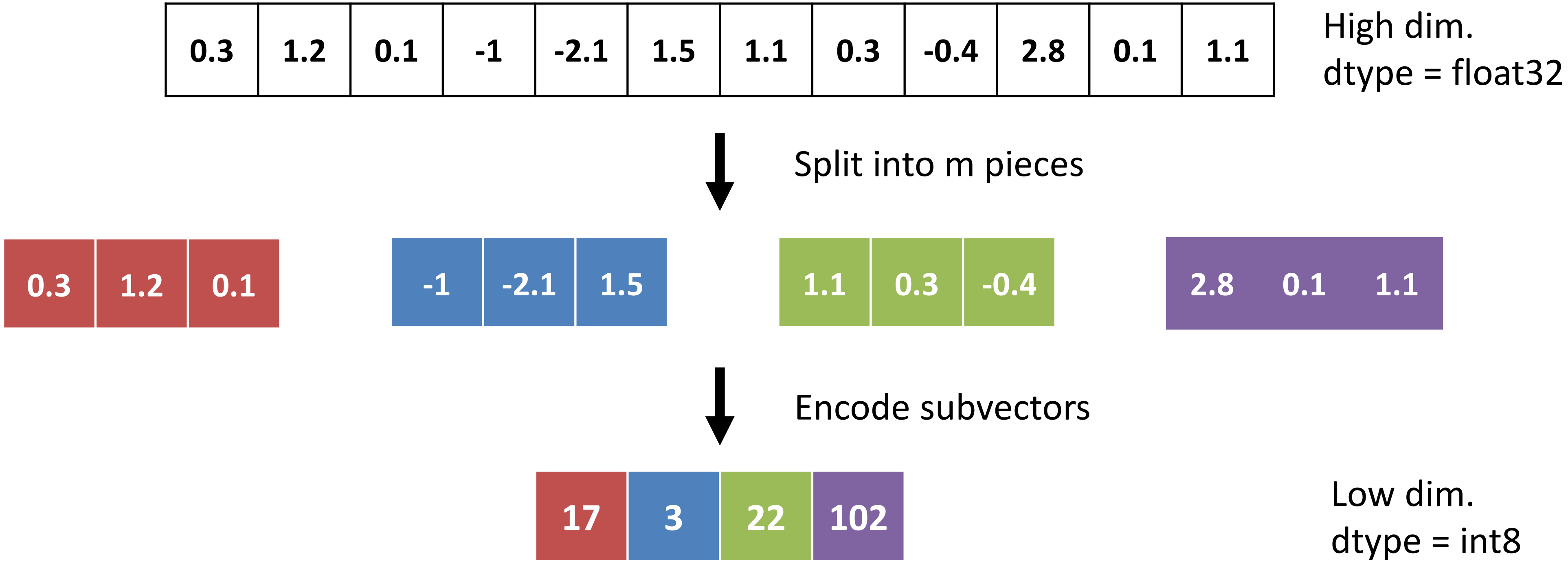
- Compute similarity between query and centroids
- Select top n clusters
- Compute similarity to all vectors of these clusters



- Large reduction of search space
- More overhead

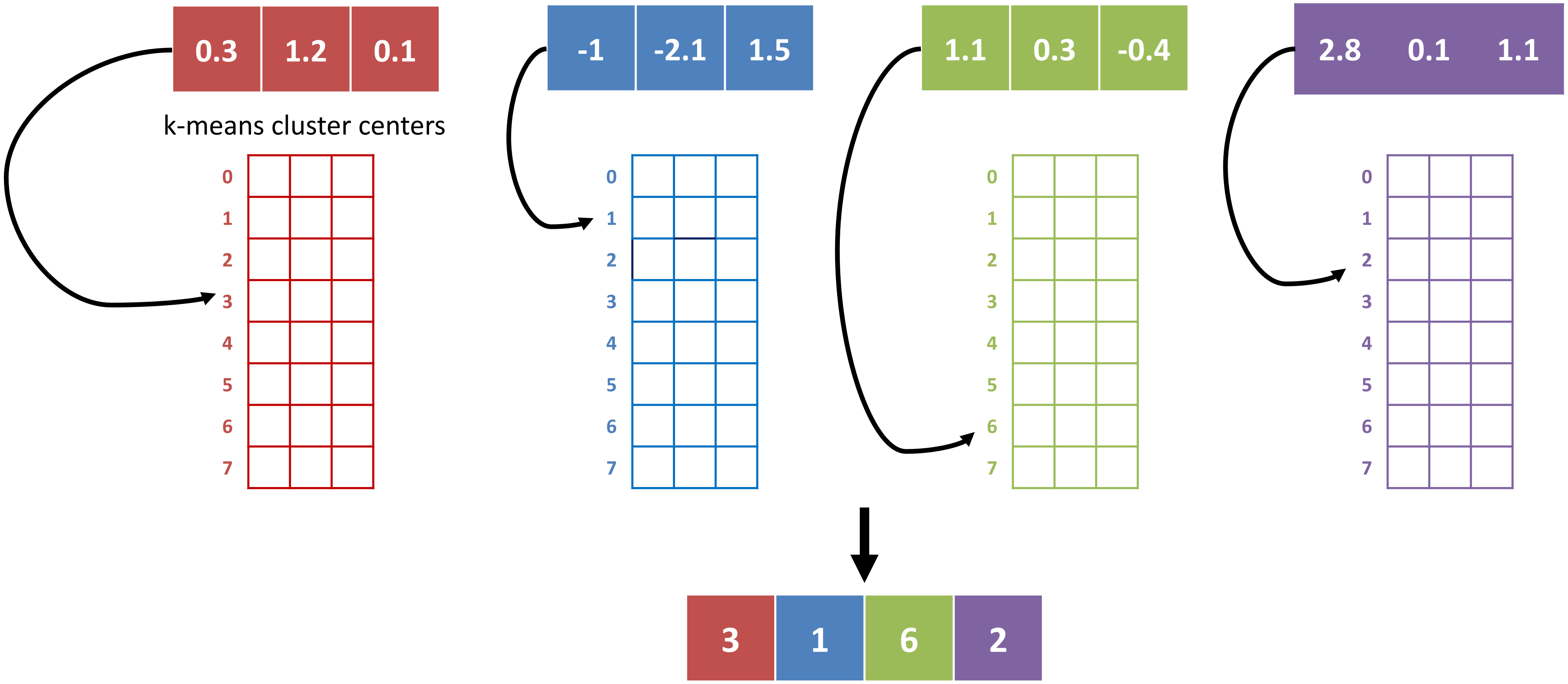
Indexing Techniques: Product Quantization

Main idea: Replace original vector of floats with lower dimensional vector of integers



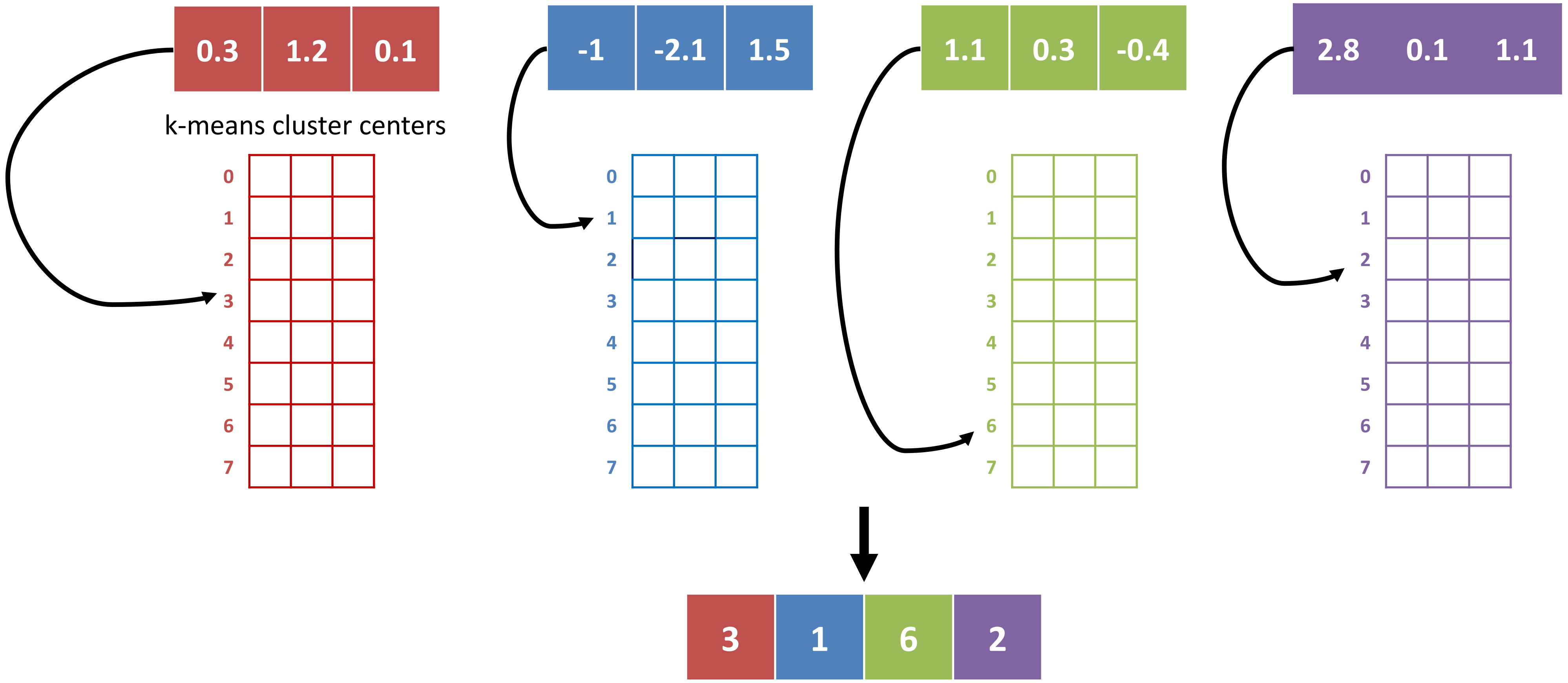
Indexing Techniques: Product Quantization

Use k-means clustering on each sub-space



Indexing Techniques: Product Quantization

Use k-means clustering on each sub-space



Indexing Techniques: Product Quantization

Reduces n * d (embedding space) matrix of floats to n * m integers

Additional preprocessing: Store distance from sub-vectors to centroids

Query time:

- Encode query vector in the same way 
- Approximate distance from query to doc by sum of stored distance from doc sub-vector to query cluster centroid

Indexing Techniques: Product Quantization

Pros:

- Much faster
- Memory efficient

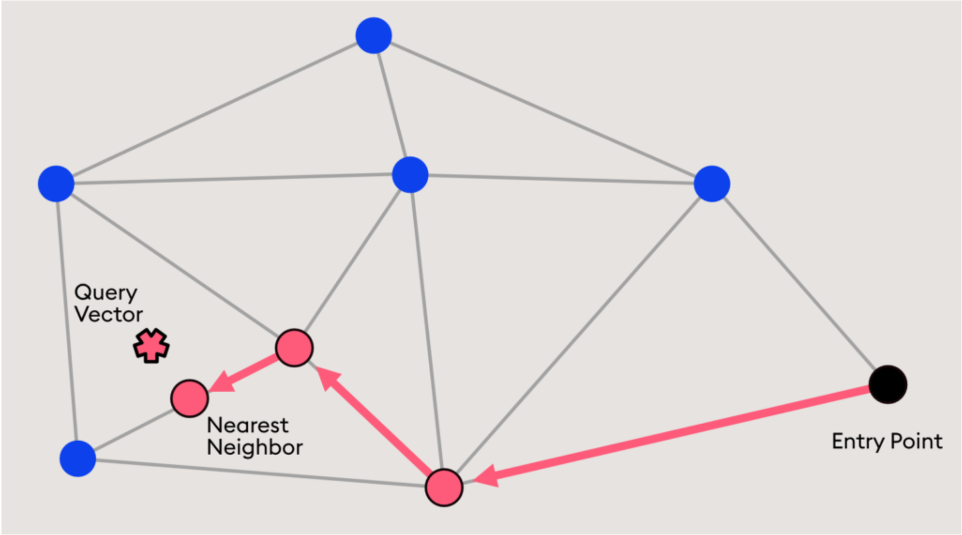
Cons:

- Results are approximate
- Quality depends on split and clustering parameters

Indexing Techniques: Graph Indices

e.g.: HNSW (Hierarchical Navigable Small Words)

- Proximity graph, vectors are linked with similar “friends”
- Search starts at predefined “entry point”, visit “friends” until no nearer vertex is found

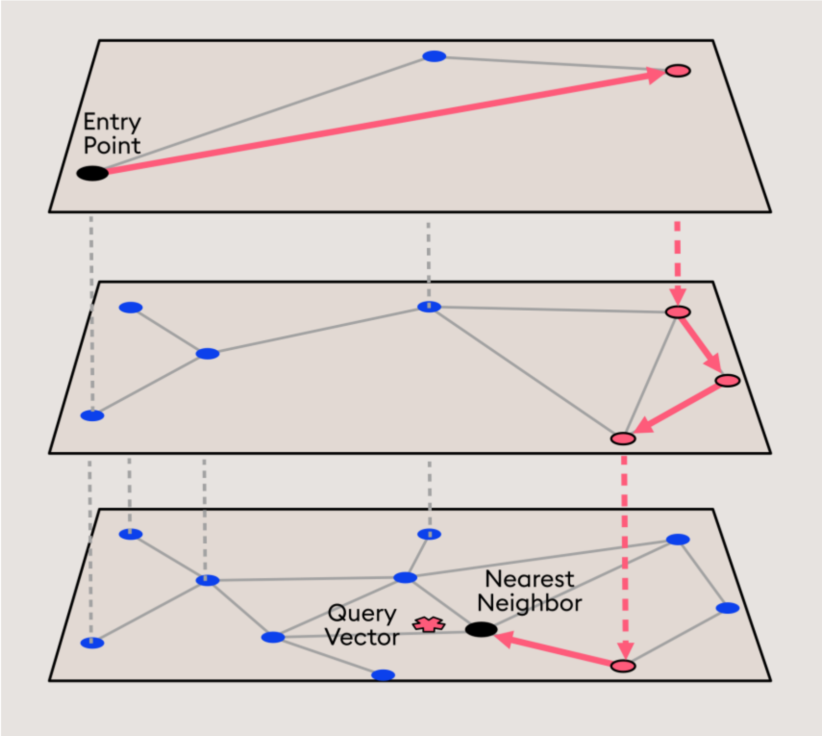


Indexing Techniques: Graph Indices

Search space is split into hierarchical layers

- Top layer has longest distances
- When at a local minimum: drop one layer and keep searching
- Repeat until NN at lowest layer

- Needs additional pre-processing and memory, but scales much better to huge data sets



Approximate NN Search

- Brute-force search does not scale well beyond a couple of million vectors
- Fortunately, nearest neighbor search of vectors is a very common and broadly used technique in ML
  - many techniques and libraries to speed up search
- Popular library: FAISS
  - Offering many algorithms (brute-force, inverted lists on clusters, HNSW, …)
  - CPU and GPU supported
- Approximate search is another tradeoff between latency-effectiveness
  - We add a lot of complexity to the search system,
    but necessary for low-latency CPU serving

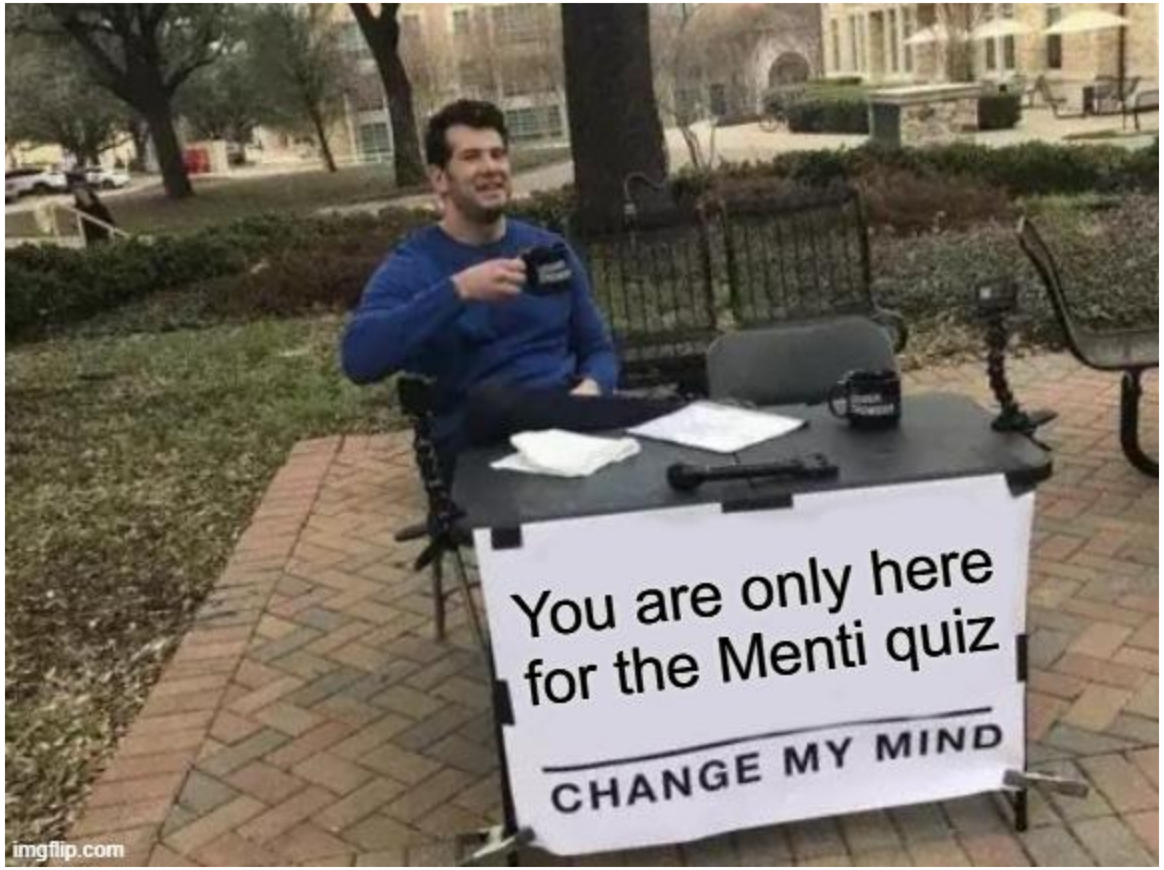
Production Support

- Dense retrieval is gaining more and more support in production systems
  - HuggingFace model hub gives us a common format to share models
  - Search engine must incorporate indexing & query encoding + provide nearest neighbor search
- Projects include Vespa.ai & Pyserini (integrates with Lucene)
  - Vespa provides deep integration of dense retrieval in common search features, such as filtering on properties
    - Important to filter during search, not after as to avoid empty result lists
  - Pyserini is a project focused on reproducing as many dense retrieval models as possible
    - Including easy hybrid search options between BM25 and DR

Other Uses for the BERT$_{DOT}$ Model

- Semantic comparisons of all sorts:
  - Sentence, passage, document similarity -> all compressed into 1 vector
  - Recommendation models
- S-BERT (Sentence transformers) library provides many models & scenarios
  - Based on the HuggingFace transformer library
  - Offers many scenarios and built models out of the box
- Adaptions based on dot-product similarity also allow for multi-modal comparisons
  - For example: Encoding images and text in the same vector-space

S-BERT library: https://github.com/UKPLab/sentence-transformers


You are only here for the Menti quiz
CHANGE MY MIND
imgflip.com

Today

IR – Dense Retrieval & Destillation

1 Dense Retrieval

- The Bert-Dot model
- (Approximate) nearest neighbor search

2 **Knowledge Destillation**

- Cross-architecture: KL Divergence

Based on:
https://github.com/sebastian-hofstaetter/teaching/tree/master/introduction-to-information-retrieval/lectures

The Idea of Knowledge Distillation

- Most training data is noisy & not very fine-granular
  - In the case of MSMARCO we only have one labeled relevant per query
  - Might have a lot of false-negatives (positives that are not labeled)

- **Teacher:**
  - Large, powerful neural network
  - High accuracy, but (too) slow

- **Student:**
  - Smaller and more efficient neural network
  - Trained to approximate teacher's predictions (not ground truth)

Different Levels of Supervision

- We may use the final output scores (or class distribution, etc..) as supervision signal
  - This makes it possible to operate architecture independent
  - Easy to use an ensemble of teachers

- We also may use intermediate results in some or all layers as signal (i.e. activations, attention distribution)
  - This locks us into a certain architecture
  - Potentially also similar parameter settings
  - Much more supervision signals, than just using final score

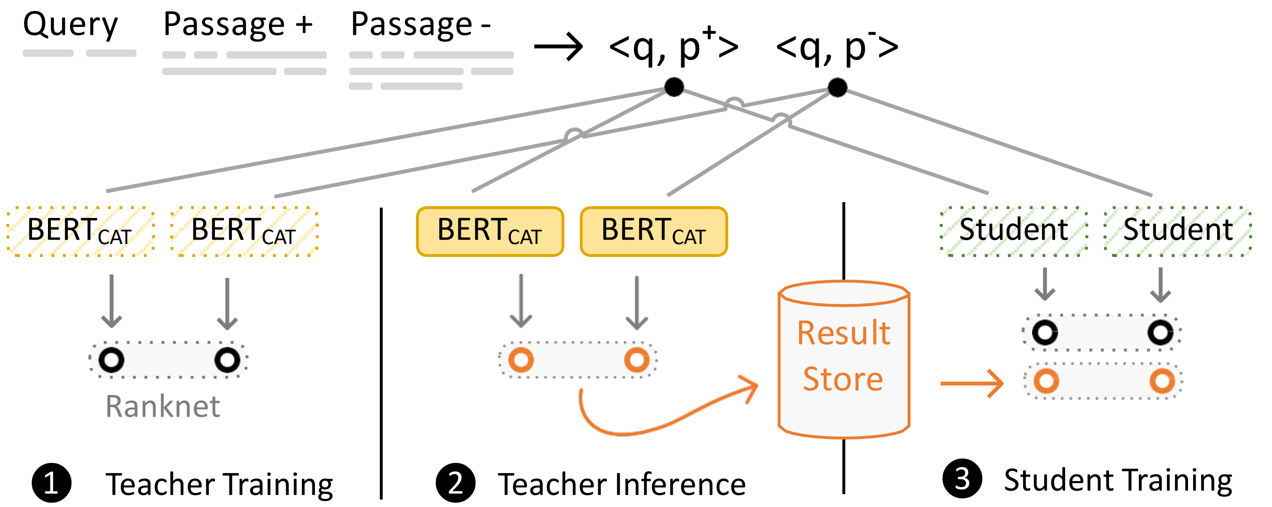
DistilBERT

- A distilled, smaller version of the general-purpose BERT model
  - Shares the same vocabulary
  - Shares the general-purpose nature (ready to fine-tune)
- Size: 6 (instead of 12) Layers with 768 output dimensions
- Trained with knowledge distillation, retains 97% of effectiveness
- Using DistilBERT as a base model in IR works very well
  - We consistently get good results for different architectures (including BERT_{DOT})
  - If we also apply knowledge distillation during the IR training, hardly a difference to larger BERT instances

DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter; Sanh et al. https://arxiv.org/abs/1910.01108
Model on the HuggingFace hub: https://huggingface.co/distilbert-base-uncased

Distillation in IR

- Train setup remains the same with triples
  - We first need to train a teacher on a binary loss
  - Then, get teacher scores from trained teacher (we can do that once)
  - Finally use those scores to train student models



Loss calculated on difference between Student output and Teacher result score

Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation; Hofstätter et al. https://arxiv.org/abs/2010.02666

Margin-MSE Loss

- Potential improvement: Optimize the margin between relevant and non-relevant sampled passages
  - Exact scores do not matter, only the relative differences
  - Margin-MSE loss definition:

$$L(Q,P^{+},P^{-}) = MSE(Ms(Q,P^{+}) - Ms(Q,P^{-}),\\ Mt(Q,P^{+}) - Mt(Q,P^{-}))$$

- Loss makes no assumption about the model architecture
  - We can mix and match different neural ranking models
- We can pre-compute the teacher scores once and re-use them

Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation; Hofstätter et al. https://arxiv.org/abs/2010.02666

Margin-MSE Dense Retrieval

- Margin-MSE also possible for BERT_{DOT} retrieval
  - Even though we do nothing specific for dense retrieval
  - Results are quite respectable and close to much more complex and costly training approaches

Model	Index Size	Teach.	TREC DL Passages 2019 nDCG@10	MRR@10	Recall@1K	MSMARCO DEV nDCG@10	MRR@10	Recall@1K
Baselines								
BM25	2 GB	–	.501	.689	.739	.241	.194	.868
$\text{BERT-Base}_{\text{DOT}}$ ANCE [44]		–	.648	–	–	–	.330	.959
TCT-ColBERT [26]		–	.670	–	.720	–	.335	.964
RocketQA [12]		–	–	–	–	–	.370	.979
Our Dense Retrieval Student Models								
$\text{BERT-Base}_{\text{DOT}}$	12.7 GB	–	.593	.757	.664	.347	.294	.913
		T1	.631	.771	.702	.358	.304	.931
		T2	**.668**	**.826**	**.737**	**.371**	**.315**	**.947**
$\text{DistilBERT}_{\text{DOT}}$	12.7 GB	–	.626	.836	.713	.354	.299	.930
		T1	.687	.818	.749	.379	.321	.954
		T2	**.697**	**.868**	**.769**	**.381**	**.323**	**.957**

- Here, DistilBERT is even better than BERT-Base
- T2 (Teacher ensemble) improves over T1 (single teacher) which improves over no teacher

Classic Way: KL-divergence

- Kullback-Leibler (KL) divergence is used to compare distributions
- For two probability distributions P and Q, the KL-divergence from P to Q is:

$$D_{KL}(P||Q) = \sum_{x \in X} P(x) \log(\frac{P(x)}{Q(x)})$$

- In case of Knowledge Destillation, we use the class probabilities from the teacher and student networks as P and Q

(use softmax to get correct probability distributions)

Cross-Domain: BEIR Zero-Shot Benchmark

- Ultimately, we want Dense Retrieval models to be plug-n-play usable
  - This is referred to as zero-shot transfer *(because we use no training data from the target collection)*
  - Zero-shot scenario is harder than in-domain evaluation *(because it solely tests generalization instead of a mix of memorization & generalization)*
- BEIR 🍺 benchmark brings many IR collections in 1 format
  - Including framework to run HuggingFace models on all collections
  - Paper showed that many DR models struggle in zero-shot transfer
  - BM25 is more consistent

BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models Thakur et al.
https://arxiv.org/abs/2104.08663

Why do DR Models Struggle on Zero-Shot?

- Possible explanations:

❶ **Generalization:** DR models just don't generalize to other query distributions

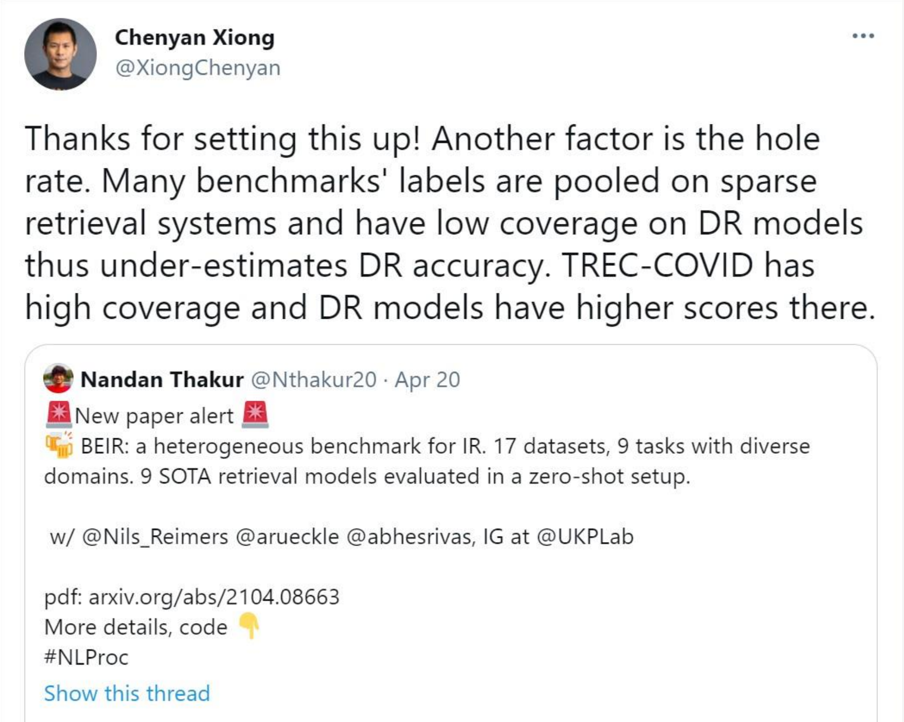
- That would be bad, back to the drawing board

❷ **Quirks:** MSMARCO training data contains too many quirks, specific to a collection

- Need adaption to training data

❸ **Pool Bias:** Many (older or smaller) collections are heavily biased towards BM25 results

- Ultimately needs re-annotation campaigns



See thread:
https://twitter.com/XiongChenyan/status/1384594186683387905

TAS-Balanced Zero-Shot Results

- More wins than losses against BM25 (and any other DR model) 🎉

Model	Size (MB)	Rank	Average	MSMARCO	TREC-COVID	NFCorpus	NQ	HotpotQA	FiQA	Signal-1M	TREC-NEWS	ArguAna	DBPedia	SCIDOCS	FEVER	Climate-FEVER	SciFact	Robust04
BM25	-	-	0.384	0.218	0.616	0.297	0.31	**0.601**	0.239	**0.388**	0.371	**0.441**	0.288	**0.156**	0.648	0.179	0.62	0.387
DistilBERT (TAS-B)	250	**1**	**0.412**	**0.408**	0.481	**0.319**	0.463	0.584	**0.3**	0.289	0.377	0.427	**0.384**	0.149	**0.7**	**0.228**	**0.643**	**0.427**
ANCE	500	**2**	0.379	0.388	**0.654**	0.237	0.446	0.456	0.295	0.249	**0.382**	0.415	0.281	0.122	0.669	0.198	0.507	0.392
DistilBERT	270	**3**	0.375	0.389	0.482	0.257	0.45	0.513	0.258	0.261	0.367	0.429	0.339	0.133	0.67	0.205	0.531	0.337
MiniLM-L-12	130	**4**	0.349	0.385	0.473	0.251	0.422	0.456	0.24	0.271	0.36	0.407	0.307	0.113	0.571	0.179	0.503	0.295
MiniLM-L-6	90	**5**	0.342	0.379	0.479	0.255	0.394	0.448	0.231	0.259	0.342	0.394	0.292	0.116	0.595	0.165	0.495	0.293
DPR (Multi)	500	**6**	0.252	0.177	0.332	0.189	**0.474**	0.391	0.112	0.155	0.161	0.175	0.263	0.077	0.562	0.148	0.318	0.252

Leaderboard snapshot from 10.01.2023 (they removed Touche-2020)

Summary: Dense Retrieval & Knowledge Distillation

1. Dense retrieval is a promising direction for the future of search

2. Knowledge distillation from strong teachers helps DR a lot