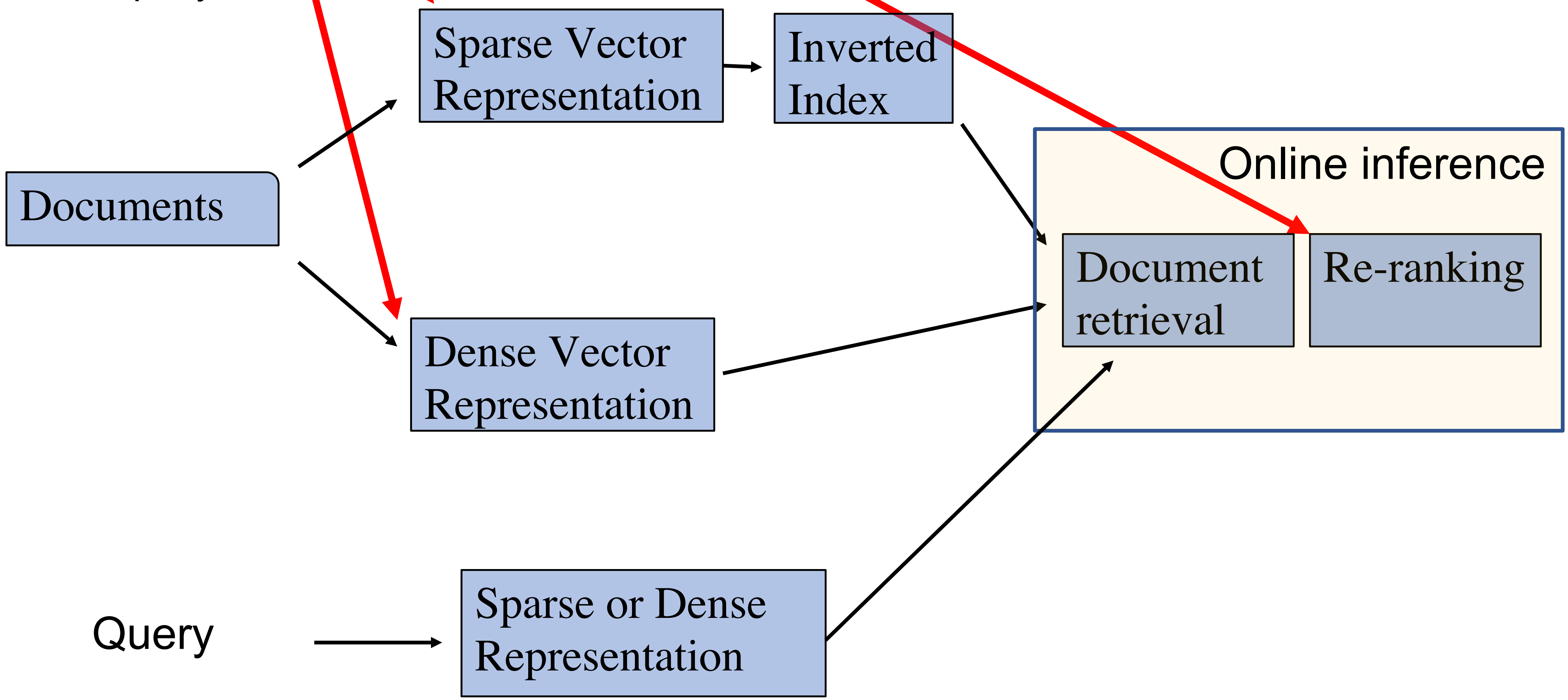
Recent Progress in Neural Information Retrieval

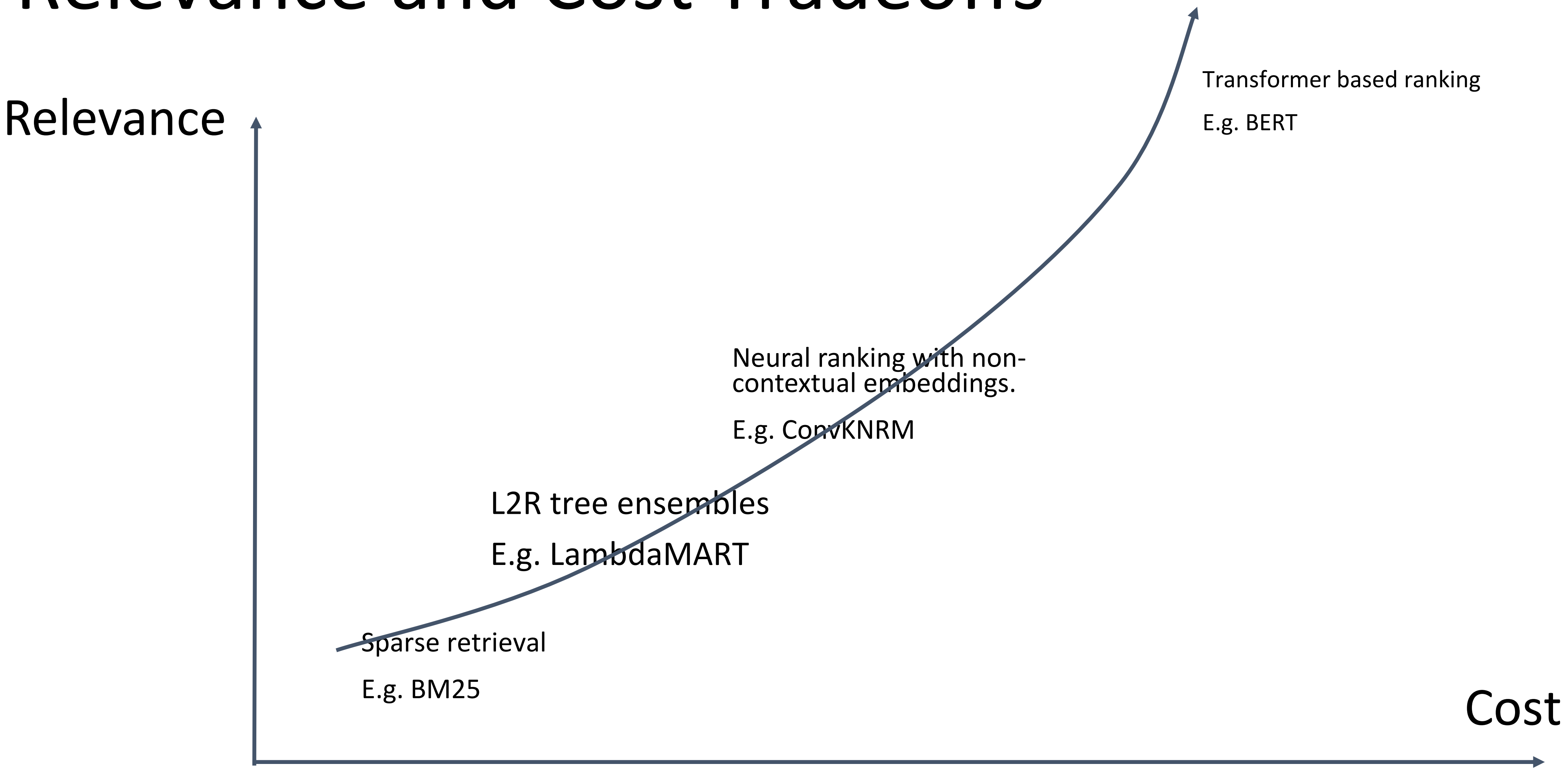
CS293S. 2022. Tao Yang

Neural Models for Information Retrieval

Every document/query is a vector



Relevance and Cost Tradeoffs



Outline

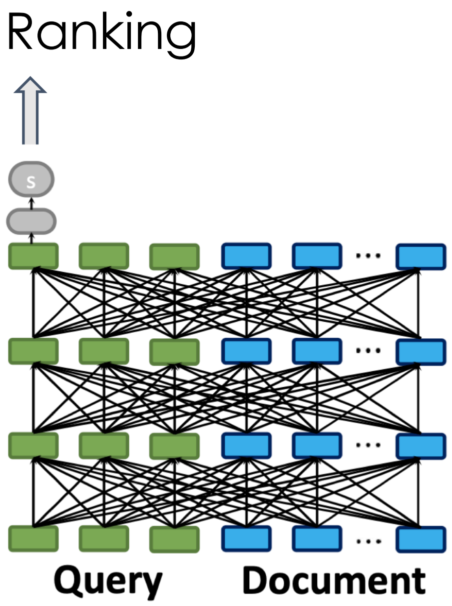
- Part 1: Time Efficiency Optimization for Faster BERT-based Neural Ranking
- Part 2: Space Efficiency Optimization for BERT-based Ranking
    - Document representation compression
- Part 3: Document Retrieval: Revisited
    - Learned sparse representations
    - Dense representations

References:

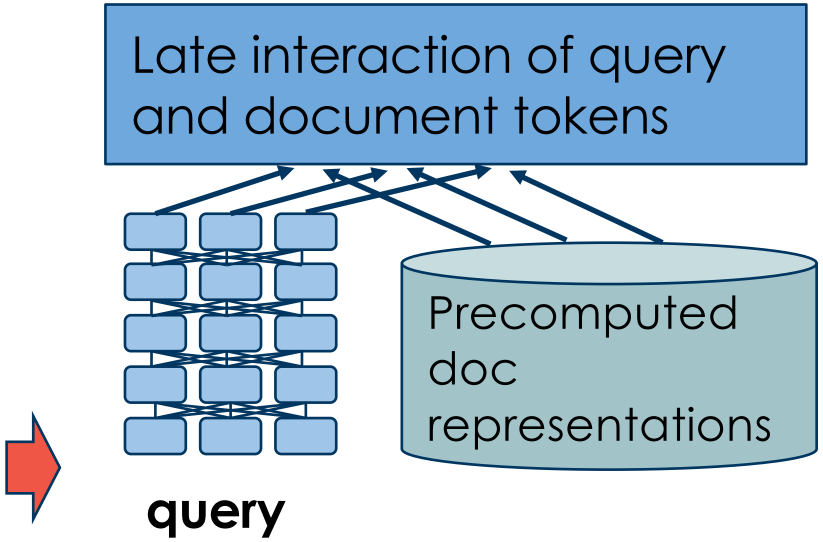
“Pretrained Transformers for Text Ranking: BERT and Beyond” by Andrew Yates, Rodrigo Nogueira, and Jimmy Lin, 2021

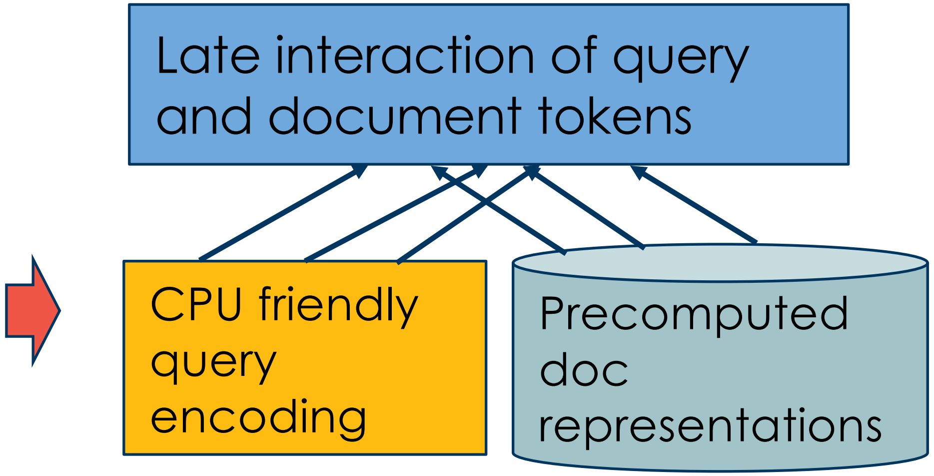
Recent papers

Part 1: Time Efficiency Optimization



Cross Encoder





1.Late Interaction

DPR (Karpukhin et al,. ACL’20),
ColBERT (Khattab et al. SIGIR’20)

2.CPU Friendly Ranking

TILDE (Zhang and Zuccon, SIGIR’21),
BECR (Yang et al., WSDM’22)

3. Simplification of neural network architectures

Efficiency Optimization: Architecture Simplification for Cross-Encoder

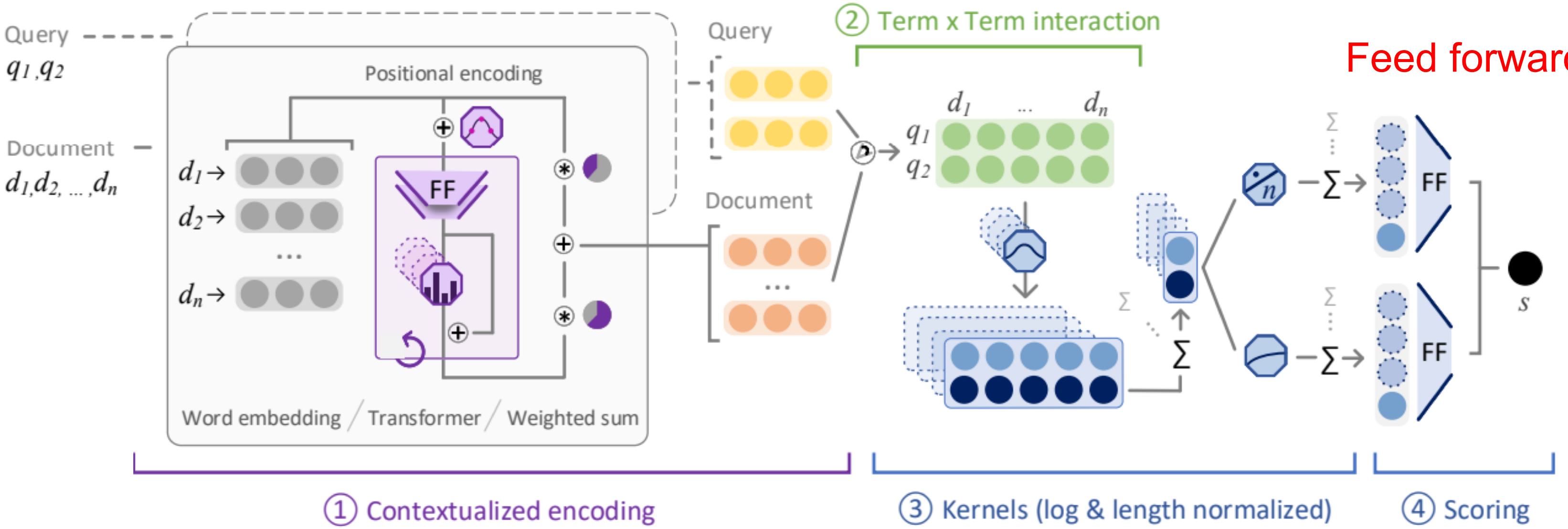
Key technique: Architecture simplification (Hofstatter et al., ECAI'20). Called TK, TKL, CK

- Reduce the number of transformer layers
- Knowledge distillation: train a simpler student model based on a complex teacher model
  - Use the outcome of a teacher ranker to construct positive/negative document pairs
  - Train the simpler student ranking model using these pairs

Simpler transformer architecture

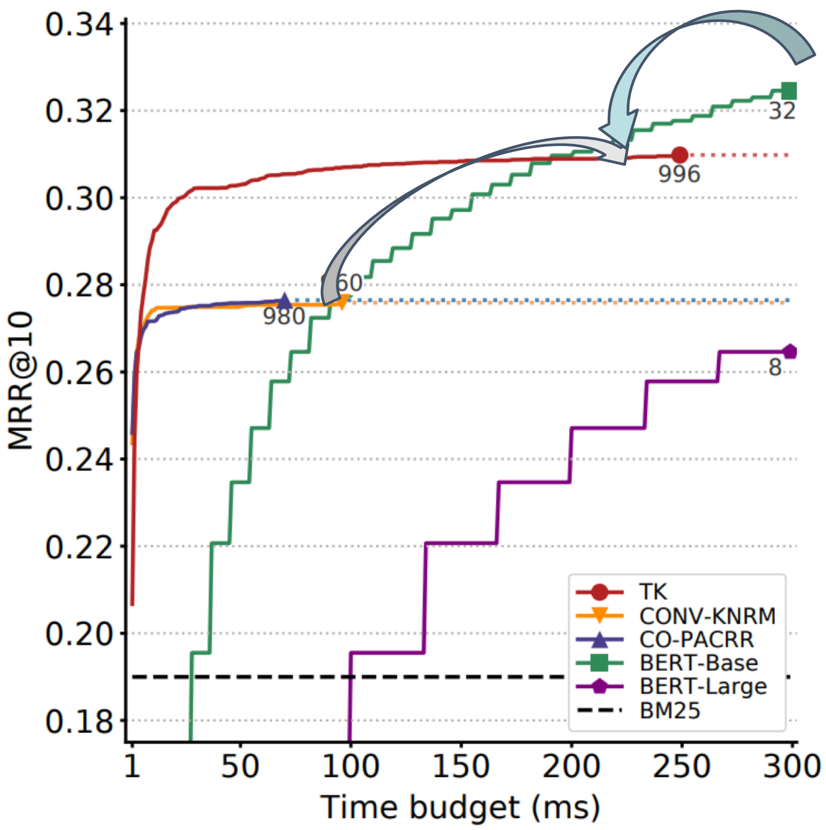
KNRM-style interaction

Feed forward net for scoring



Simplified Transformer Efficiency: TK Architecture

Model	MSMARCO-Passage MRR	Recall	nDCG	Depth	MSMARCO-Document MRR	Recall	nDCG	Depth
BM25	**0.194**	**0.402**	**0.241**	-	**0.252**	**0.500**	**0.311**	-
LM	0.171	0.358	0.213	-	0.202	0.423	0.254	-
RM3	0.169	0.388	0.219	-	0.156	0.367	0.206	-
MatchPyramid	0.249	0.476	0.301	71	0.286	0.531	0.344	15
DUET	0.248	0.468	0.299	42	0.266	0.520	0.327	15
PACRR	0.259	0.493	0.313	619	0.283	0.536	0.344	15
CO-PACRR	0.273	0.514	0.328	987	0.284	0.543	0.345	19
KNRM	0.235	0.465	0.288	127	0.261	0.519	0.323	14
CONV-KNRM	0.277	0.519	0.332	967	0.283	0.542	0.345	19
BERT-Base	**0.376**	**0.639**	**0.437**	997	**0.352**	0.623	**0.417**	58
BERT-Large	0.366	0.627	0.426	997	0.350	**0.630**	**0.417**	93
TK – 1 Layer	0.303	0.560	0.361	997	0.305	0.572	0.369	29
TK – 2 Layer	0.311	0.564	0.369	997	0.312	0.577	0.375	29
TK – 3 Layer	**0.314**	**0.570**	**0.373**	997	**0.316**	**0.586**	**0.380**	31

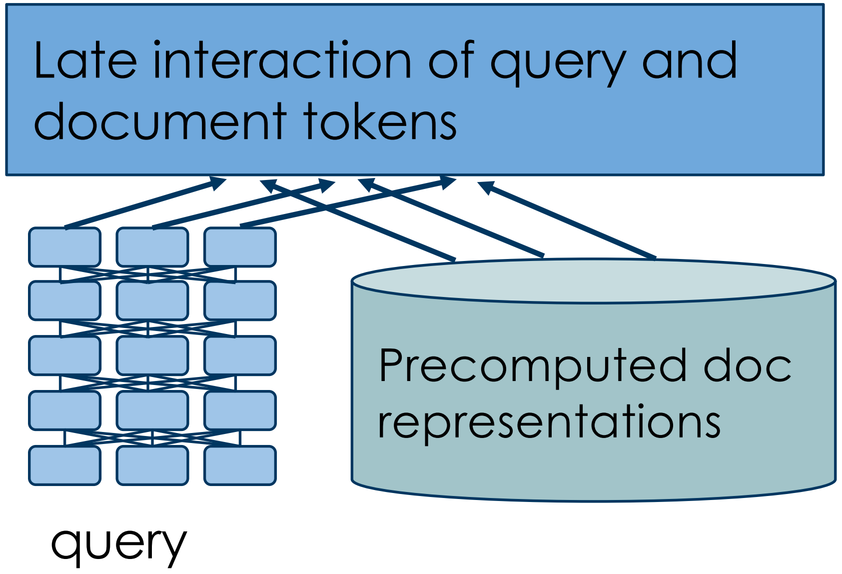


Compared to Conv-KNRM: Around 2.5x inference time, MRR 12% higher.

Compared to BERT_base, 1/37 inference time, MRR 18% lower.

Efficiency Optimization via Late Interaction between Query and Doc Embeddings: Dual-Encoder Architecture

Document representation can be pre-computed before online query processing

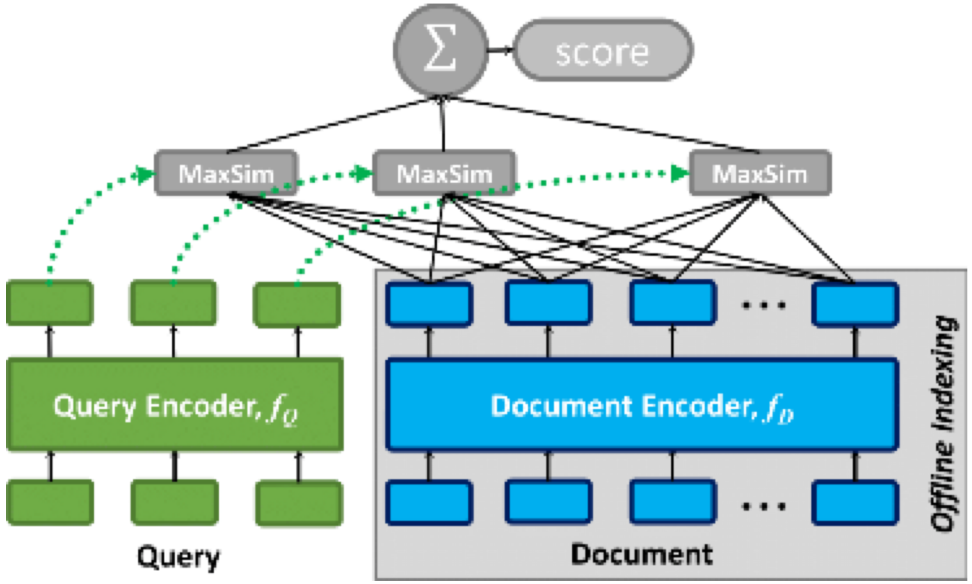


- **Single-Vector Dual Encoder (Dense Representation Models):**
  - Each document is a vector of elements
  - DPR (Karpukhin et al,. ACL’20)
  - Sentence BERT (Reimers, EMNLP’19)
  - ANCE (Xiong et al., ICLR’21)
- **Multi-vector dual encoder:**
  - Each doc is a vector of vectors
  - ColBERT (Khattab et al. SIGIR’20)
  - PreTTR (MacAvaney et al. SIGIR’20)
  - MVR (Zhang et al., ACL’22)

Multi-vector dual encoder: ColBERT (Khattab et al. Stanford, SIGIR'20)

Key technique: fine-grained contextual late interaction

- Each passage is encoded as a set of token-level embeddings during offline
- At search time, it embeds every query into another set of token embeddings
- Rank score = maximum vector similarity between query q and terms in document d based on dot products and max pooling



$$S_{q,d} := \sum_{i \in [|E_q|]} \max_{j \in [|E_d|]} E_{q_i} \cdot E_{d_j}^T$$

For each query token embeddings

For each doc token embeddings

Precomputed embedding space cost is high.

ColBERT Performance on MS MARCOS Passages

Reranking

Method	MRR@10 (Dev)	MRR@10 (Eval)	Re-ranking Latency (ms)	FLOPs/query
BM25 (official)	16.7	16.5	-	-
KNRM	19.8	19.8	3	592M (0.085×)
Duet	24.3	24.5	22	159B (23×)
fastText+ConvKNRM	29.0	27.7	28	78B (11×)
$\text{BERT}_{\text{base}}$ [25]	34.7	-	10,700	97T (13,900×)
$\text{BERT}_{\text{base}}$ (our training)	36.0	-	10,700	97T (13,900×)
$\text{BERT}_{\text{large}}$ [25]	36.5	35.9	32,900	340T (48,600×)
ColBERT (over $\text{BERT}_{\text{base}}$)	34.9	34.9	61	7B (1×)

Similar relevance as BERT-base but much lower latency.

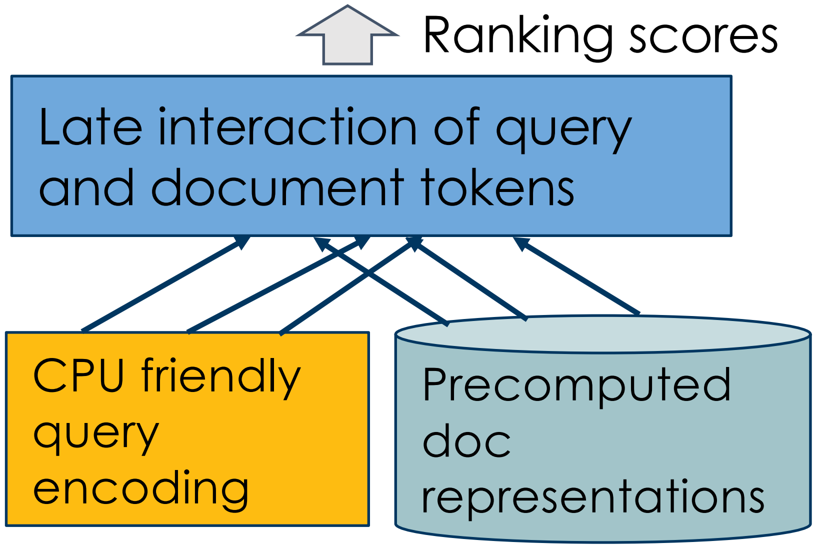
End-to-end

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
$\text{ColBERT}_{\text{L2}}$ (re-rank)	34.8	36.4	-	75.3	80.5	81.4
$\text{ColBERT}_{\text{L2}}$ (end-to-end)	36.0	36.7	458	82.9	92.3	96.8

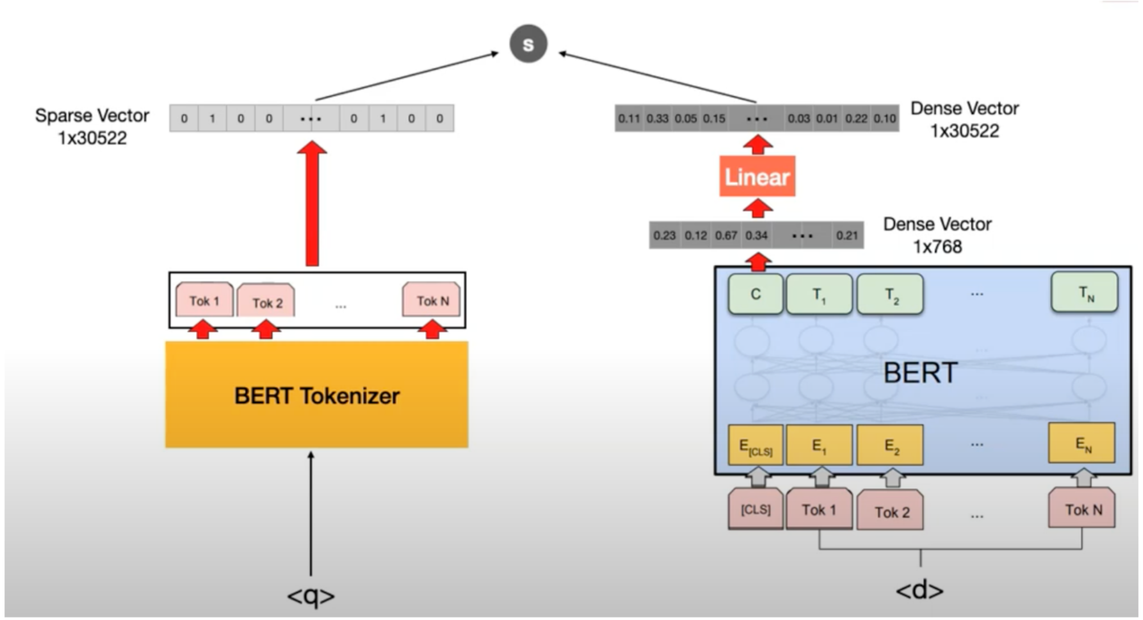
CPU-friendly Ranker

Dual encoder designs speeds up document encoding in online processing. Some work further alleviate query encoding step for online.

1. Ranker Based on Exact Match
   a. TILDE (Zhang and Zuccon, SIGIR’21)
2. Query Decomposition
   a. BECR (Yang et al., WSDM’22)



TILDE: Term Independent Likelihood Model for Passage Re-ranking (Zhang and Zuccon, SIGIR'21)



- No query encoder, so query latency is much lower
- TILDE assumes that query terms are independent.

$$\text{TILDE-QL}(q|d^k) = \sum_{i}^{|q|} \log(P_\theta(q_i|d^k))$$

Can be precomputed for all tokens.

Models that learn token weights distribution for each document can use a sparse learned inverted index for retrieval efficiency. Examples include SPLADE (Formal et al., SIGIR'21) and DeepImpact (Mallia et al., SIGIR'21).

TILDE Relevance and Latency

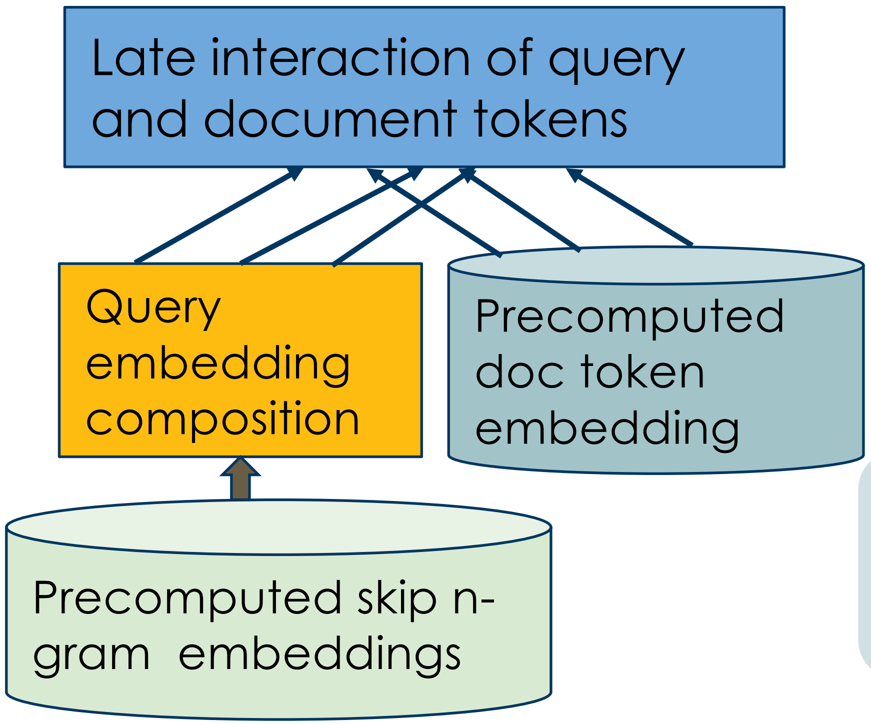
	MS MARCO		DL2019		DL2020	
Method	**MRR@10**	**Latency**	**nDCG@10**	**MAP**	**nDCG@10**	**MAP**
BM25	0.187	130	0.506	0.377	0.480	0.286
(i) Representation based						
BM25 + EPIC	0.270	356 + 108	0.609	0.411	0.576	0.349
docTquery-T5 + EPIC	0.302	279 + 20	0.686	0.473	0.624	0.405
(ii) Modified document text						
docTquery-T5	0.277	143	0.641	0.462	0.619	0.407
(iii) Direct deep language model		GPU				
BM25 + BERT-base*	0.347	2, 970	0.703	—	0.668	0.431
BM25 + BERT-large*	0.365	3, 500	0.738	0.506	—	—
(iv) Deep query likelihood		GPU				
BM25 + QLM-BERT**						
QL	0.281	4, 500	0.641	0.482	0.625	0.391
DQL	0.290	9, 000	0.662	0.484	0.635	0.401
BM25 + QLM-T5**						
QL	0.294	5, 000	0.653	0.497	0.652	0.426
DQL	0.301	10, 000	0.672	0.505	0.665	0.435
TILDE (ours)		CPU				
BM25 + TILDE						
TILDE-QL	0.269	0.5 + 29	0.579	0.406	0.620	0.406
TILDE-QDL with BiQDL	0.280	290 + 64	0.609	0.420	0.621	0.412
docTquery-T5 + TILDE						
TILDE-QL	0.285	0.5 + 0.9	0.650	0.467	0.624	0.417
TILDE-QDL with BiQDL	0.295	290 + 3.1	0.654	0.468	0.622	0.413

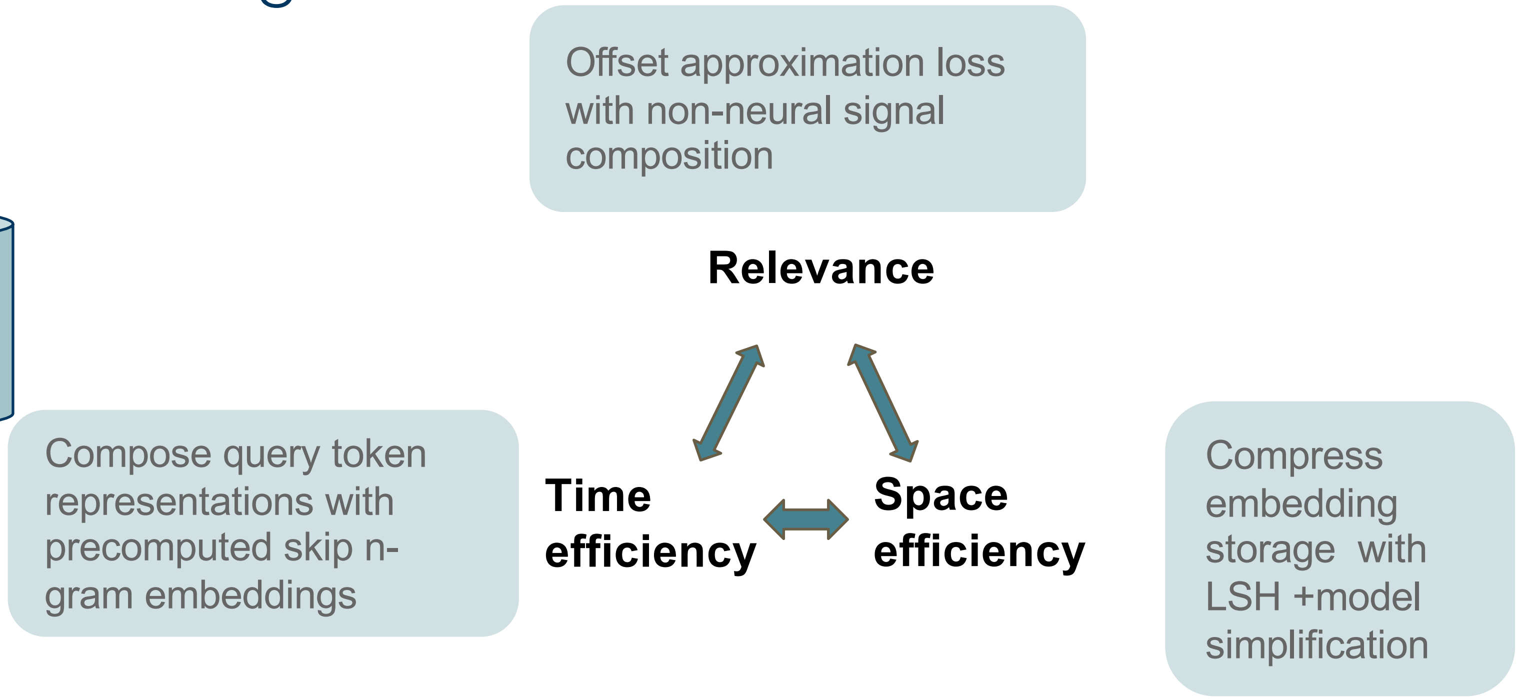
Query Latency (ms)
Query inference + rerank

The query likelihood option (QL) achieve good latency by removing query encoding.

BECR (BERT-based Composite Reranking) (Yang et al., WSDM’22)

3 key optimization techniques for a trade-off triangle





Runtime Embedding Composition for Query Tokens

Benefits: Drastically lower time cost of query token embedding computation

Pre-computed skip n-gram embeddings

...
neural
ranking
model
(neural, ranking)
(neural, model)
(ranking, model)

Example query: neural ranking model

Query token: neural

Embedding lookup for related unigrams/word pairs

Fast embedding composition for query tokens

$$E(neural) = \frac{\frac{1}{4}e(neural^{neural}) + \frac{1}{1}e(neural^{neural,ranking}) + \frac{1}{2}e(neural^{neural,model}))}{\frac{1}{4}+\frac{1}{1}+\frac{1}{2}}$$

Online Composite Re-Ranking

Strategy: Linear combination of deep and non-neural ranking signals
Benefits: Offset relevance loss due to query token embedding approximation

$$S = S_{deep} + S_{lexi} + S_{others}$$

- Deep soft matching component
  - similar to CEDR-KNRM architecture
  - The deep score is a summation of all term subscores
- Lexical matching component
  - Linear combination of BM25 features, word proximity features etc
- Other features
  - [CLS] representation of documents
  - pageRank

Flow of Training, Indexing, and Online Inference

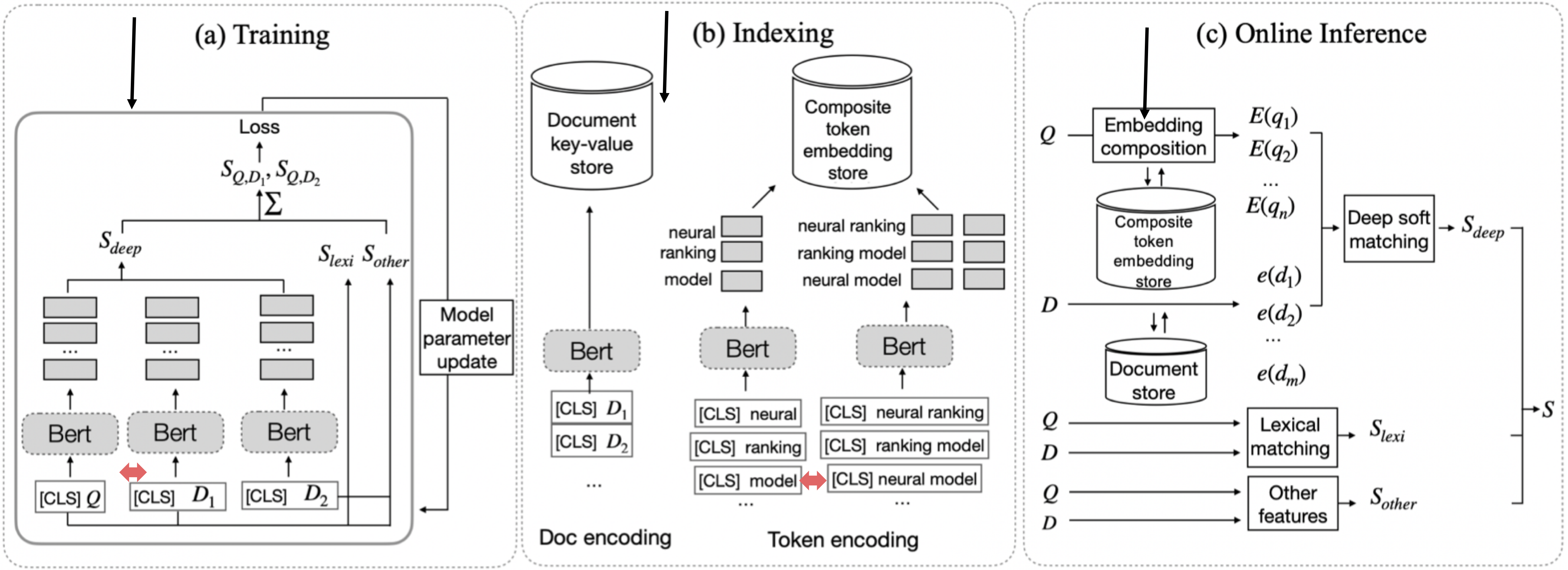


Figure 1: Training, Offline Processing and Online Inference in BECR

Relevance Evaluation on ClueWeb09CatB, Robust04, MS MARCO Dev/DL19/DL20

Model	ClueWeb09-Cat-B NDCG@5	ClueWeb09-Cat-B NDCG@20	ClueWeb09-Cat-B P@20	Robust04 NDCG@5	Robust04 NDCG@20	Robust04 P@20	MSMARCO MRR@10 Dev	DL19 NDCG@10	DL20 NDCG@10
BM25	0.2351	0.2294	0.3310	0.4594	0.4151	0.3548	0.167	0.488	0.480
ColBERT (Ours)	0.2408	0.2400	0.2067	0.3809	0.3498	0.3074	0.355	0.701	0.674
ColBERT (from [5, 25])	0.2273 [5]	0.2365 [5]	0.2507 [5]	0.4031 [5]	0.3754 [5]	0.3254 [5]	0.349 [25]	–	–
CONV-KNRM	$0.2869^{\S}$	$0.2735^{\S}$	$0.3698^{\S}$	$0.4742^{\S}$	$0.4501^{\S}$	$0.3349^{\S}$	–	–	–
BERT-base	$0.2853^{\S}$	$0.2612^{\S}$	$0.3764^{\S}$	$0.5160^{\ddagger\S}$	$0.4514^{\S}$	$0.3983^{\S}$	0.349	0.686	0.672
CEDR-KNRM (Ours)	$0.3030^{\ddagger\S}$	$0.2693^{\S}$	$0.3961^{\S}$	$0.5563^{*\ddagger\S}$	$0.4637^{\S}$	$0.4249^{\S}$	0.344	0.702	0.686
CEDR-KNRM (from [3, 34])	–	–	–	–	0.5381 [34]	0.4667 [34]	–	0.682 [3]	0.675 [3]
BECR^{-}	$0.3588^{\P*\ddagger\S}$	$0.3066^{\P*\ddagger\S}$	$0.4016^{\S}$	$0.5366^{*\ddagger\S}$	$0.4635^{\S}$	$0.4045^{\S}$	0.323	0.682	0.655
BECR	$0.3632^{\P*\ddagger\S}$	$0.3075^{\P*\ddagger\S}$	$0.3987^{\S}$	$0.5349^{\ddagger\S}$	$0.4656^{\S}$	$0.4005^{\S}$	0.319	0.658	0.647

Compared to BERT-base, better relevance for ClueWeb, Robust04, and a degradation on MS MARCO.

Operation counts (FLOP) and inference time

Re-rank 150 ClueWeb-Cat-B pages. Query length n=3 or 5

Model Specs.	n	FLOPs (ratio)	Time (ms) (ratio) GPU	CPU
KNRM	3	148M (5×)	1.3 (1×)	123.5 (5×)
	5	246M(5×)	1.6(0.5×)	312.8 (8×)
ColBERT	3	480M (15×)	13.7 (9×)	–
	5	779M (15×)	13.7 (4×)	–
BERT	3	12.2T (234k×)	4359 (2900×)	–
	5	12.2T (580k×)	4431 (1300 ×)	–
CEDR-KNRM	3	12.2T(234k×)	5577 (3700×)	–
	5	12.2T (580k×)	5601 (1700×)	–
BECR,L=13,LSH	3	81M (2.6×)	2.9 (2×)	65.3 (3×)
	5	136M (2.6×)	5.7 (2×)	117.7 (3×)
BECR,L=5,LSH	3	31M (1×)	1.5 (1×)	25.4 (1×)
	5	52M (1×)	3.3 (1×)	40.7 (1×)

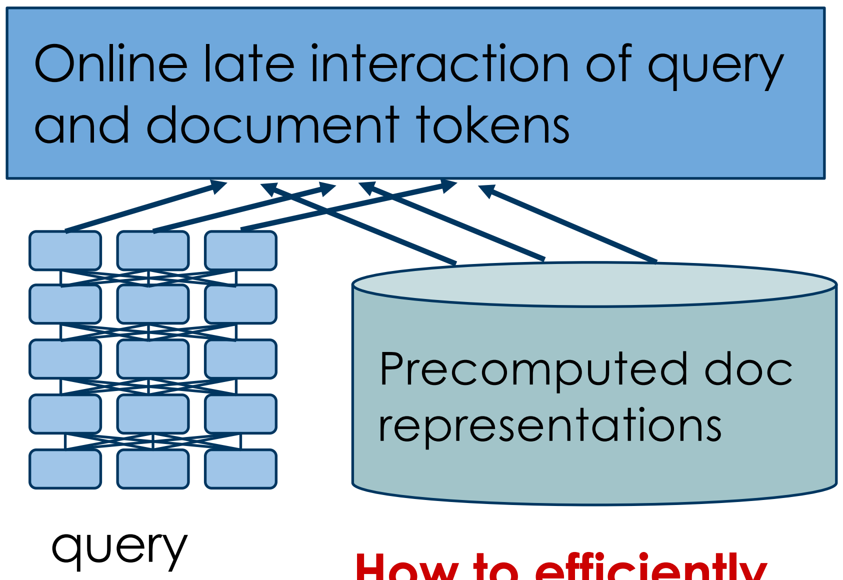
BECR: 15x less operation counts than ColBERT, 234Kx less than BERT

Tens of milliseconds without GPU

Outline

- Part 1: Time Efficiency Optimization for Faster BERT-based Neural Ranking
- Part 2: Space Efficiency Optimization for BERT-based Ranking
  - Document representation compression
- Part 3: Document Retrieval: Revisited
  - Learned sparse representations
  - Dense representations

Document Representation Compression: Why?



How to efficiently store the document representations?

- Embedding footprint of the precomputed multi-vector document representation is too large
  - ColBERT
    - 143GB (for MS MARCO 8.8M passages)
    - 1.6TB (for 3.2M documents)
- Large random I/O access latency and subject to high I/O contention
  - Compression reduces storage and speeds up inference in industrial settings.
- Challenges
  - Unsupervised compression techniques such as product quantization achieves unsatisfactory performance.

A Comparison with Related Embedding Compression Techniques

- Use an encoder to reduce the dimensionality. Slower ranking than ColBERT
  - PreTTR (MacAvaney et al., 2020)
  - SDR (Cohen et al., 2021)
- Compress embedding storage with Locality-Sensitive Hashing. Unsupervised
  - BECR (Yang et al., 2022)
- Vector quantization with codebooks
  - Product quantization (Jégou et al., 2011)
  - Codebook (Shu and Nakayama, 2018)
  - JPQ (Zhan et al., 2021)

Unsupervised, not optimized for ranking

Ranking oriented with jointly learned compression
Doesn't decompose contextual signals of tokens

Contextual Quantization (Yang et al., ACL'22)

- **Contextual decomposition of token representations with better compressibility**
- **Jointly learned compression with fast ColBERT ranking**

Example of context-aware token codes by CQ

Each token is compressed as a vector of M codewords. Each codeword has K possible values called codebook.

M =4, K=4.

Context	**Token codes**		
William Shakespeare was widely regarded as the world's greatest **actor**, **poet**, **writer** and dramatist.	writer [4,4,3,1]	actor [4,4,3,1]	poet [1,4,3,1]
I would like to have either a cup of **coffee** or a good **fiction** to kill time.	coffee [3,3,3,4]	fiction [3,1,3,4]	
She sat on the river **bank** across from a series of wide, large steps leading up a hill to the **bank** of America building.	1^{st} bank [3,1,4,2]	2^{nd} bank [4,1,3,1]	
Some language techniques can recognize word senses in phrases such as a river **bank** and a **bank** building.	1^{st} bank [4,3,2,2]	2^{nd} bank [3,1,1,4]	
If you **get** a cold, you should drink a lot of water and **get** some rest.	1^{st} get [2,2,4,2]	2^{nd} get [2,1,2,4]	

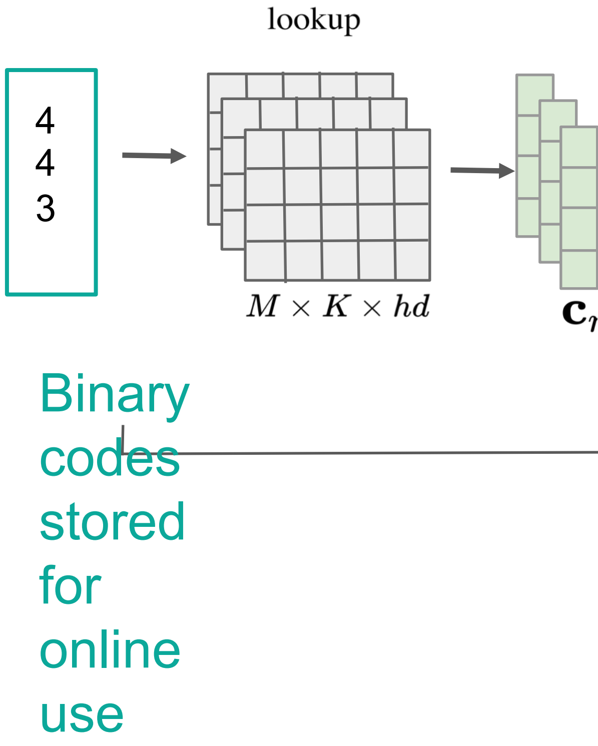
Different tokens in similar contexts have similar codes (different by 0-1 digit)

Same tokens in different contexts have different codes (3-4)

Each codeword is a vector of D/M values with product quantization. Uncompression yields a vector of D dimensions.

Example of quantization and online decoding

- Writer =[4,4,3]
- M = 3 codebooks. Use log K bits for each code (e.g. 2 bits for K=4)
- Given a compressed code vector with 3 codes, what is the uncompressed embedding for "writer"?
- Find the codeword vectors stored for code a_4 in the first book, b_4 in the second book etc.
- Product quantization: Embedding = concatenation of a_4 b_4 c_3
- Additive quantization: Embedding = sum of a_4 b_4 c_3



Compression ratio for embeddings:

Each embedding has D dimensions

M codebooks and K codewords per codebook.

Log K bits space per code: logK .

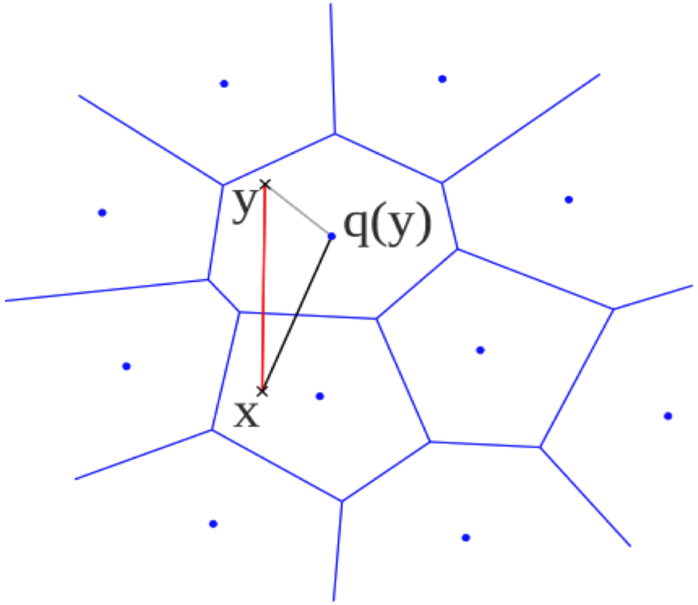
Compressed space per embedding: M logK bits

Space compression ratio: 32D/(M log K)

Example: D=128, M=16, K=256 → Ratio 32.

Traditional method to train vector quantization

Embedding y is approximated as q(y) which is decompressed from the compressed code vector for y.



Decompression in product quantization concatenates M codeword subvectors for each token through codeword lookup. Training finds M coodbooks with K codewords per book, e.g. using K-means clustering

$$min_{C^1,\ldots,C^M} \sum_y ||y - q_{(y)}||^2$$

s.t. $$q \in C^1 \times \ldots \times C^M$$

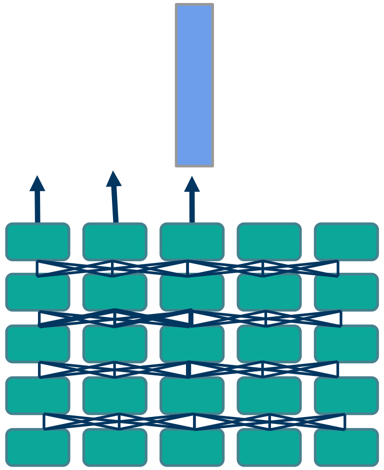
\# of codebooks: M
codewords per codebook: K

- **The above cost function does not optimize relevance**
- **Contextual Quantization: Jointly train quantization with ColBERT based ranking to maximize the relevance**

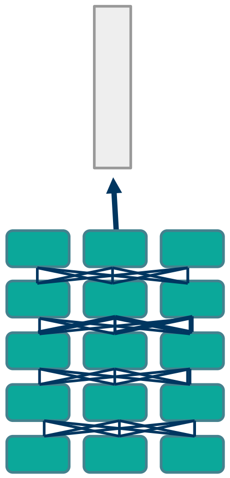
Compact Token Representations with Contextual Quantization for Efficient Document Re-ranking (Yang et al., ACL'22)

- **Key techniques:**
  - Decomposition of contextual token representations
  - Ranking oriented learning with distillation

Contextual Embedding = Doc-independent component + Doc-dependent component



[CLS] tok TOK tok tok …

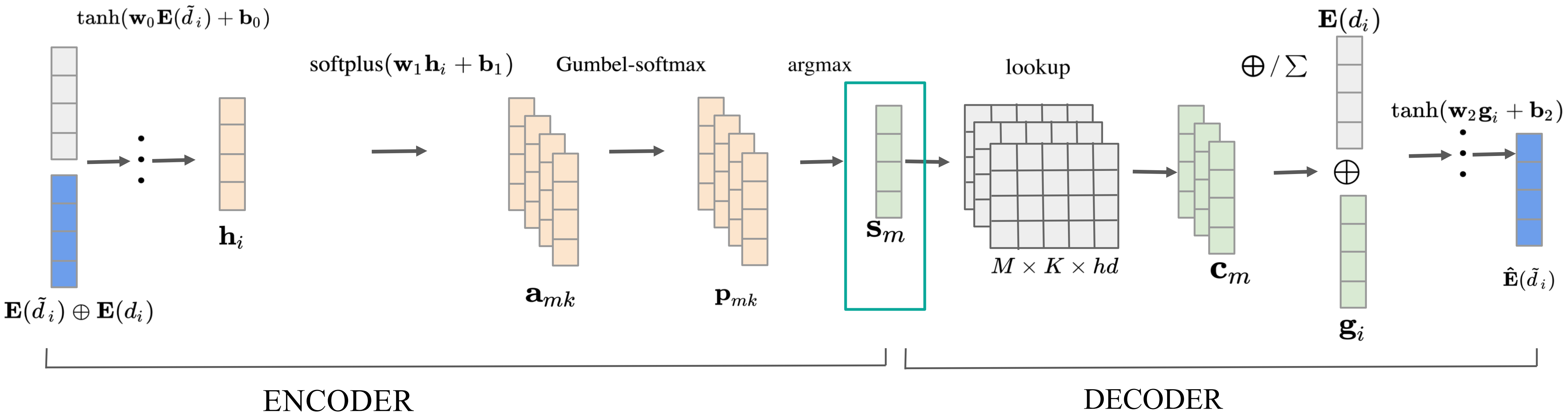


[CLS] TOK [SEP]

- Large space demand

The space of doc-independent embeddings is limited

End-to-end Encoding and Decoding for Contextual Quantization



Offline contextual quantization:

- Input E(t) is the token output from the last BERT layer as contextual document embedding.
- $E(\bar{t})$ is the last layer of BERT applied with [CLS] o t [SEP] as doc-independent embedding.

Binary codes stored for online use

Online inference recovers ranking contribution via embedding composition:

$$\hat{\mathbf{E}}(t) = \tanh(\mathbf{w}_2(\hat{\mathbf{E}}(t^{\Delta}) \circ \mathbf{E}(\bar{t})) + \mathbf{b}_2)$$

$\hat{\mathbf{E}}(t_i^{\Delta})$: estimated doc-dependent component

$\hat{\mathbf{E}}(t_i)$: estimated contextual embedding

Training Loss for Learning Codebooks and Codes

- Reconstruction $\mathcal{L}_{MSE} = \sum \|\mathbf{E}(t_i) - \hat{\mathbf{E}}(t_i)\|_2^2.$
  - General codebook learning loss
  - Doesn't optimize for ranking

Probability of being correct

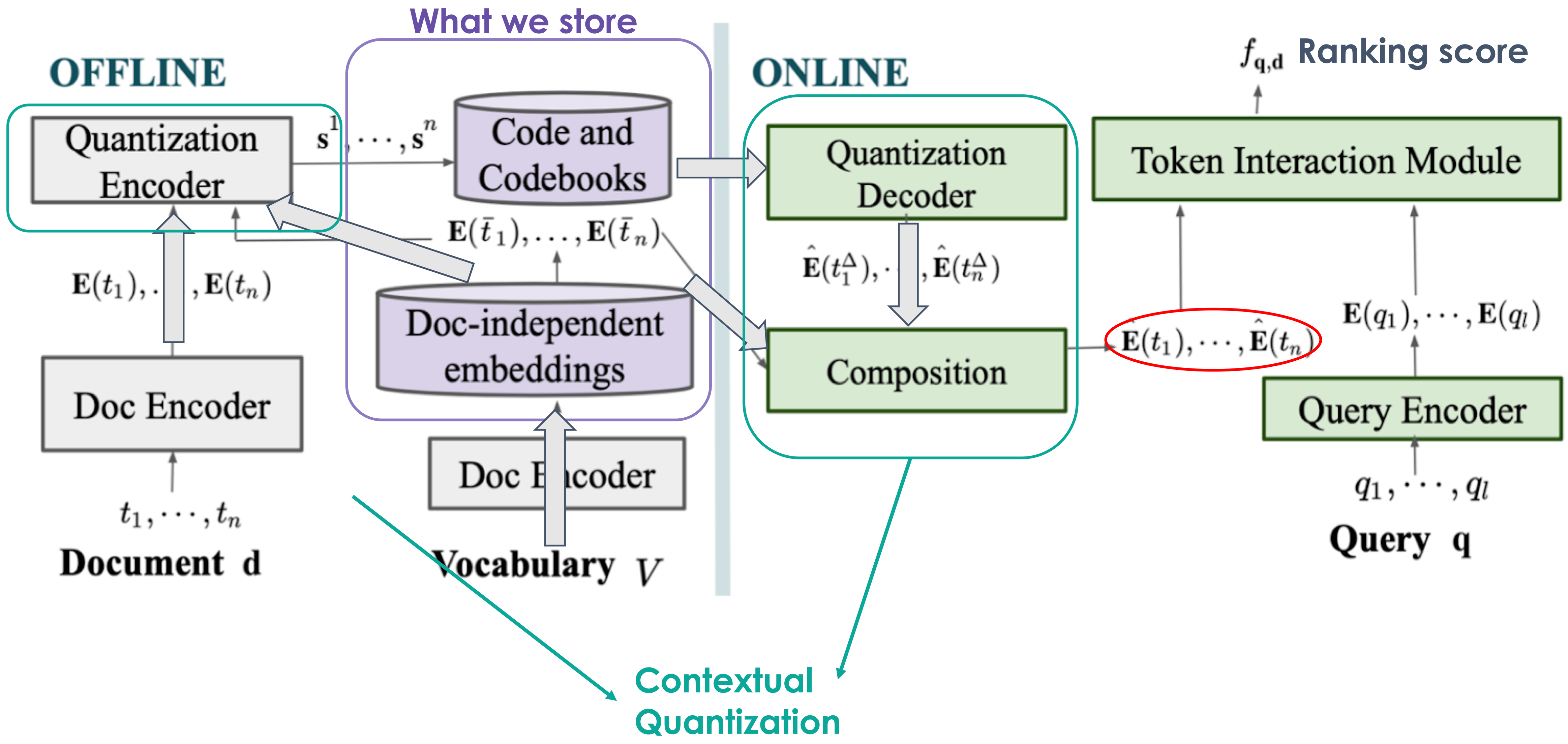
- Pairwise cross-entropy $\mathcal{L}_{PairwiseCE} = \sum(-\sum_{j=\mathbf{d}^+,\mathbf{d}^-} P_j \log P_j)$
  - Ranking oriented loss
  - Same loss for training rankers

Teacher difference   Student difference

- Distillation loss $\mathcal{L}_{MarginMSE} = \sum((f_{\mathbf{q},\mathbf{d}^+} - f_{\mathbf{q},\mathbf{d}^-}) - (\hat{f}_{\mathbf{q},\mathbf{d}^+} - \hat{f}_{\mathbf{q},\mathbf{d}^-}))^2$
  - Use the original ranking model as teacher
  - Minimize score discrepancy between reconstructed and original embeddings

- Codebook cold start or warm start
- Joint training ranker and codebook vs
  - Train ranker, freeze, then train codebook

Offline Processing and Online Ranking Pipeline



Uncompressed baseline

Compression baseline

MSMARCO Passage

Model Specs.	Dev MRR@10	TREC DL19 NDCG@10	TREC DL20 NDCG@10
	Retrieval choices		
BM25	0.172	0.425	0.453
docT5query	0.259	0.590	0.597
DeepCT*	0.243	0.572	–
TCT-ColBERT(v2)	0.358	–	–
JPQ*	0.341	0.677	–
DeepImpact	0.328	0.695	0.628
uniCOIL	0.347	0.703	0.675
	Re-ranking baselines (+BM25 retrieval)		
BERT-base	0.349	0.682	0.655
BECR	0.323	0.682	0.655
TILDEv2*	0.333	0.676	0.686
ColBERT	0.355	0.701	0.723
	Quantization (+BM25 retrieval)		
ColBERT-PQ	0.290 (-18.3%)	0.684 (-2.3%)	0.714 (-1.2%)
ColBERT-OPQ	0.324 (-8.7%)	0.691 (-1.4%)	0.688 (-4.8%)
ColBERT-RQ	–	0.675 (-3.7%)	0.696 (-3.7%)
ColBERT-LSQ	–	0.664 (-5.3%)	0.656 (-9.3%)
ColBERT-CQ	0.352 (-0.8%)	0.704 (+0.4%)	0.716 (-1.0%)
	(+uniCOIL retrieval)		
ColBERT	0.369	0.692	0.701
ColBERT-CQ	0.360 (-2.4%)	0.696 (+0.6%)	0.720 (+2.7%)

Model	Doc task Space	Passage task Space	Disk I/O	Latency	MRR@10
BECR	791G	89.9G	–	8ms	0.323
PreTTR*	–	2.6T	>182ms	>1000ms	0.358
TILDEv2*	–	5.2G	–	–	0.326
ColBERT	1.6T	143G	>182ms	16ms	0.355
ColBERT-small*	297G	26G	–	–	0.339
ColBERT-OPQ	112G	10.2G	–	56ms	0.324†
ColBERT-CQ					
undecomposed	112G	10.2G	–	17ms	0.339†
K=256	112G	10.2G	–	17ms	0.352
K=16	62G	5.6G	–	17ms	0.339†
K=4	37G	3.4G	–	17ms	0.326†

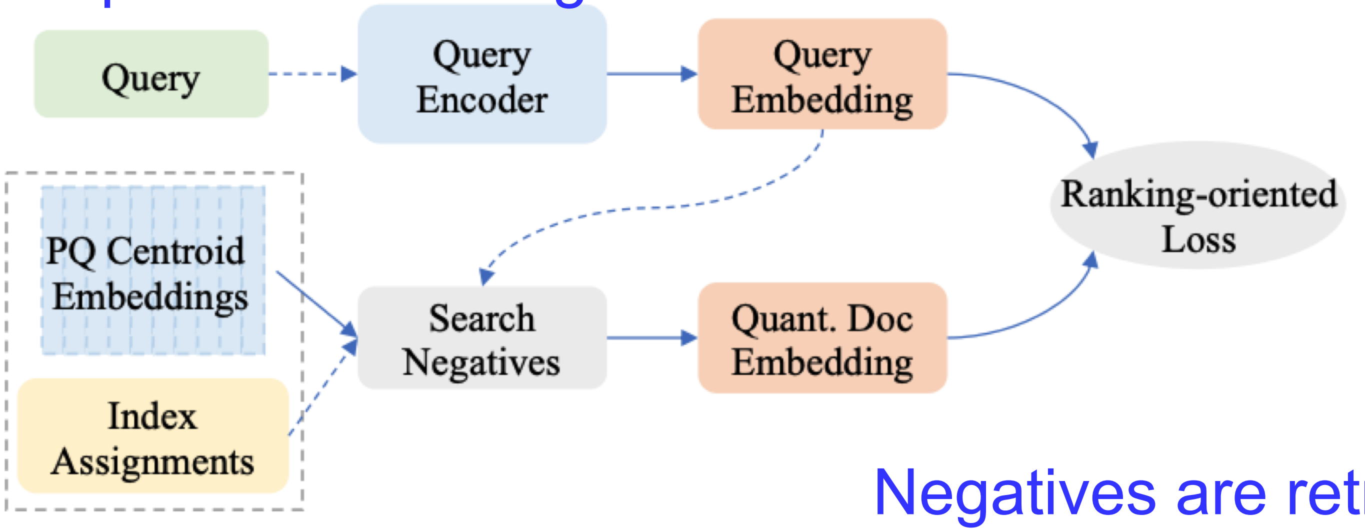
Gain from ranking oriented training

Gain from contextual decomposition

- CQ outperforms other quantization approaches in relevance effectiveness
- Small degradation of relevance compared to original ColBERT re-ranking.

Jointly Optimizing Query Encoder and Product Quantization to Improve Retrieval Performance (Zhan et al., CIKM’21)

Update PQ centroid embeddings using training triplets and ranking loss.



Warmup using traditional OPQ model to get the index assignment.

Negatives are retrieved during training using the updated query embedding and PQ centroids.

Key techniques:

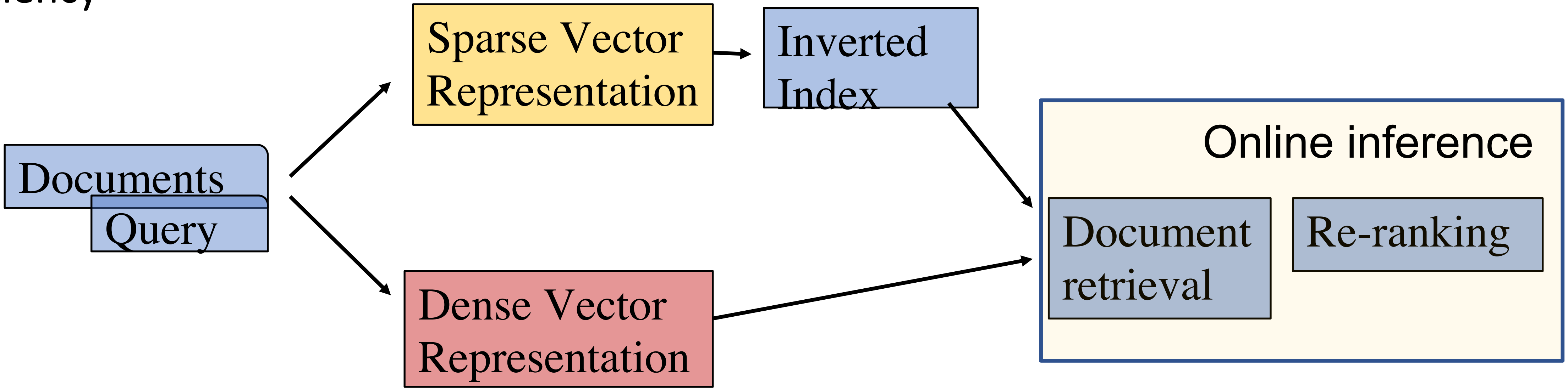
- Ranking oriented PQ centroid optimization.
- End-to-end dynamic negative sampling.

Outline

- Part 1: Time Efficiency Optimization for Faster BERT-based Neural Ranking
- Part 2: Space Efficiency Optimization for BERT-based Ranking
    - Document representation compression
- Part 3: Document Retrieval: Revisited
    - Learned sparse representations
    - Dense representations

Document Retrieval: Sparse vs. Dense Representations

- **For a web-scale large dataset**
  - Multi-stage search pipeline is more practical for better efficiency



- **For a relatively small dataset**
  - Single-stage dense retrieval with integrated ranking may be sufficient to address vocabulary mismatching queries and documents

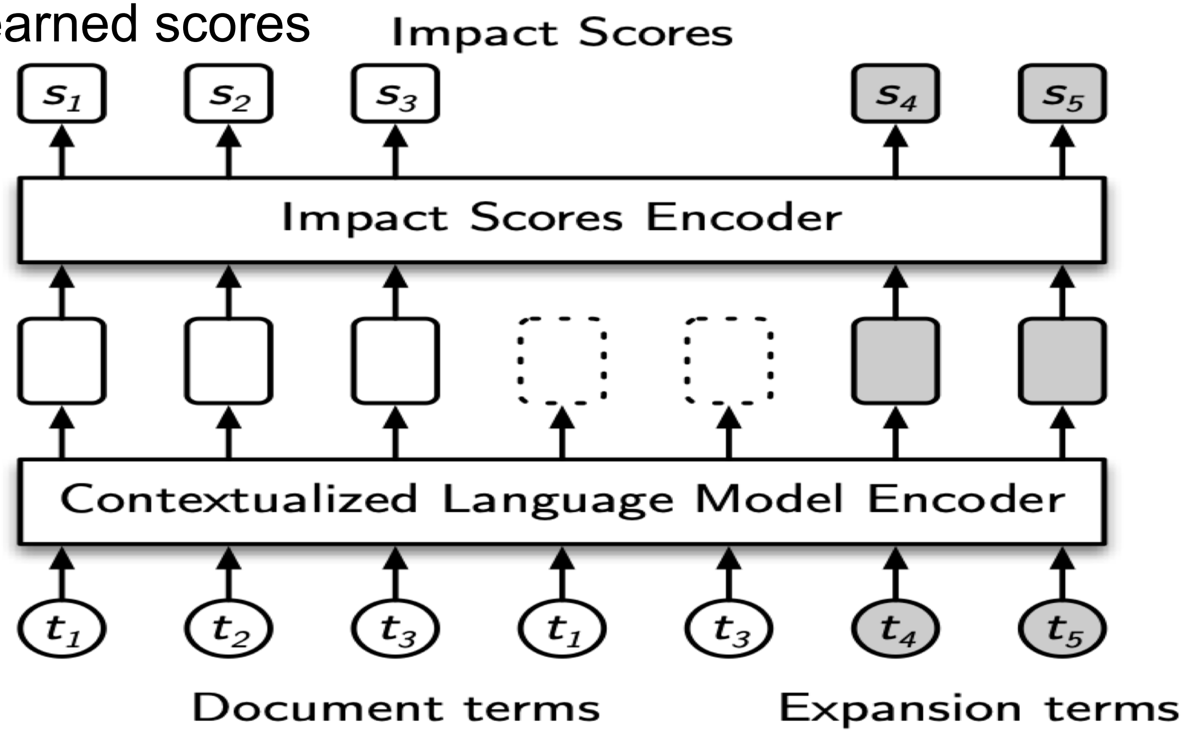
Sparse Vector Representations of Documents

- **Original idea:** Treat a document as a bag of words with BM25 weighting
- **Pros/cons**: Fast retrieval but relevant documents fail to match if query words do not appear. E.g., movie vs. film.
- **Techniques to address query-document vocabulary mismatch**
  - Document expansion
    - Doc2Query [Lin et al.]: append relevant tokens to documents
- **Use a learned contextual score from the neural model.**
  - DeepCT/HDCT [Dai&Callan, SIGIR20]:
    - Use BERT to learn term weights, replacing term frequency.
  - DeepImpact [Mallia et al., SIGIR21]: Use a transformer to learn a score.
  - COIL/UniCOIL [Gao et al. ECIR21][Lin&Ma, arXiv21] after document expansion:
    - Convert ColBERT to exact token matching, assign a vector or a scalar score to each token
- **Generate new vocabularies with SpladeV2** [Formal et al., SIGIR21]:
  - Transform token impact to a sparse vector of tokens
- **Faster retrieval with a hybrid learned representation and BM25 index**.
  - Guided traversal [Mallia et al., SIGIR22]

Sparse Retrieval DeepImpact [Mallia et al., NYU, SIGIR21]

- Documents are expanded using the DocT5Query algorithm. DocT5Query is a T5 model trained to generate queries highly relevant to a given document.
- Impact scores encoder is constructed with 2 multilayer perceptron neural layers to compute a learned score for each term in a document

Learned scores



2-layer neural net

BERT

Document

Sparse Retrieval with Splade/SpladeV2 [Formal et al., SIGIR21]

```
                      photo           film          movie
Doc d =      (0, 0, …, 0, 1, 0, …, 0, 1, 0, …, 0, 0, 0, …, 0)
Splade(d)=   (0, 0, …, 0, 2, 0, …, 0, 5, 0, …, 0, 3, 0, …, 0)
```

- Each document is represented by a sparse vector of size |V|. Compute a neural score for each term by projecting BERT embeddings to this vector.
  - For each token in the doc, calculate its impact on other possible tokens in the vocabulary set. $w_{ij} = \text{transform}(h_i)^T E_j + b_j \quad j \in \{1, \ldots, |V|\}$

    i is the token index in the doc, j is the token index in the vocabulary set.

  - Summarize the weight of each token across the whole doc by adding the impacts from other tokens in the vocabulary set, specific for this document.

$$w_j = \sum_{i \in t} \log\left(1 + \text{ReLU}(w_{ij})\right)$$

- A document vector has too many non-zeros? When training, add the regularization loss to control sparsity in the cost function

$$\ell_{\text{FLOPS}} = \sum_{j \in V} \bar{a}_j^2 = \sum_{j \in V} \left(\frac{1}{N} \sum_{i=1}^{N} w_j^{(d_i)} \right)^2$$

Average number of floating-point operations token j
Involved in all documents in a training batch of size N $\bar{a}_j = \frac{1}{N} \sum_{i=1}^{N} w_j^{(d_i)}$

Sparse Retrieval: A Comparison of Different Term Scoring Methods

Dataset: MS MARCO Passage Dev

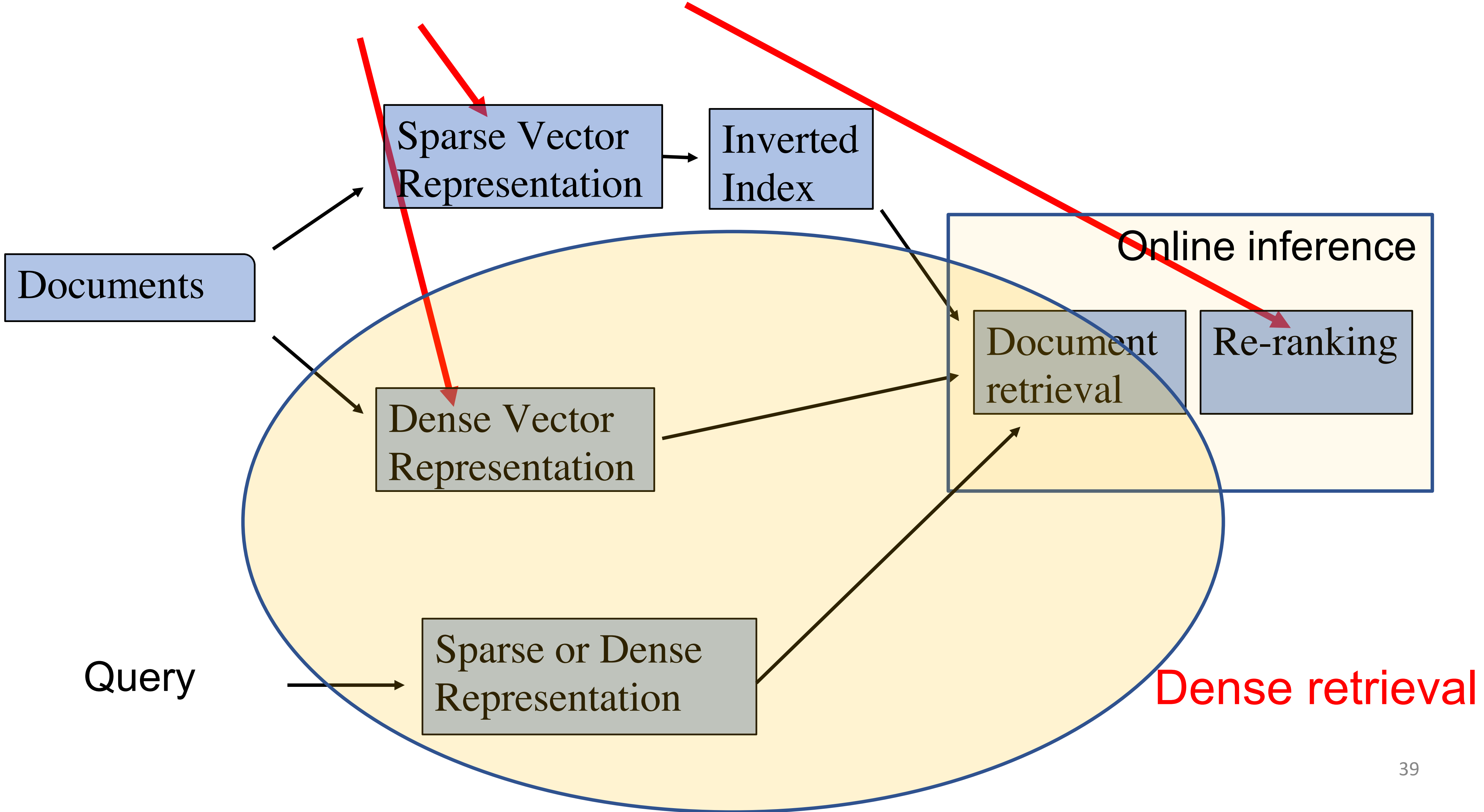
Stats of inverted index From Mallia et al., SIGIR'22.

Model	Terms	Postings	Avg. Query Length
BM25	2,660,824	266,247,718	4.5
DeepCT	989,873	128,969,826	4.5
DocT5Query	3,929,111	452,197,951	4.5
uniCOIL	27,678	587,435,995	686.3
TILDEv2	27,437	809,658,361	4.9
SPLADEv2	28,131	2,028,512,653	2037.8
DeepImpact	3,514,102	628,412,657	4.2

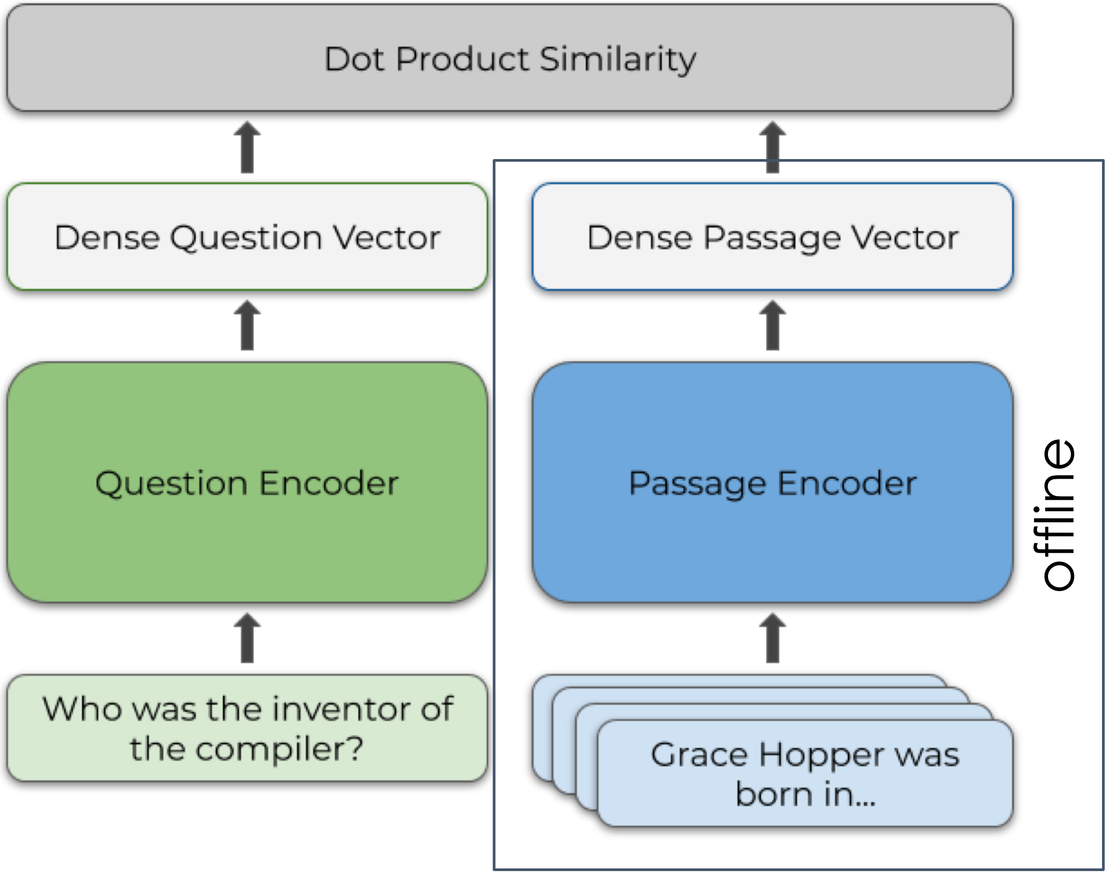
- BM25 is fast with lower relevance without semantic matching support
- DocT5Query improves query-doc matching by adding more terms per document
- DeepImpact improves relevance by addressing vocabulary mismatching, but slower than BM25
- SpladeV2 costs significantly longer times with more nonzeros in sparse vectors, but the relevance is the highest.

Model	Retrieval Time to search index (ms)	Relevance (MRR@10)
BM25	5.7	0.187
DeepCT	N/A	0.24
TILDEv2	20.7	0.333
DeepImpact	19.5	0.326
UniCOIL	37.9	0.352
SpladeV2	219.9	0.369

Neural Models for Information Retrieval: Where are we?



Dense Retrieval: Basic Computation Flow and Techniques

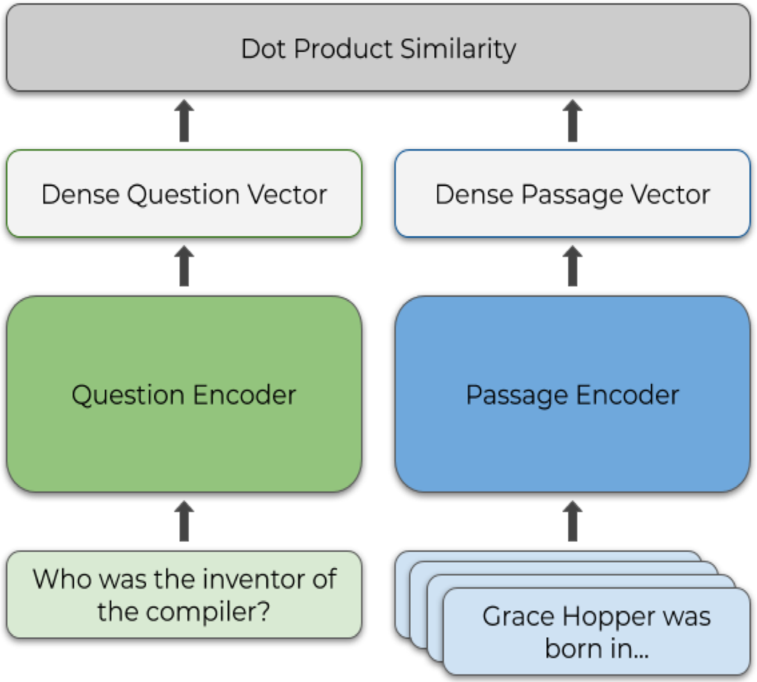


Queries and documents are encoded into single vectors respectively.

- **Time and efficiency optimization**
  a. Nearest Neighbor Search with Approximation
     - GPU implementation: FAISS [Facebook AI, 2017]
  b. Vector Compression
     - Product quantization (PQ), RepCONC [Zhan et al., WSDM22]
- **Vector representations**
  a. Multi Vectors, e.g., ColBERT [Khattab et al., SIGIR20]
  b. Single Vectors, e.g., TCT-ColBERT [Lin et al., arXiv20], DPR [Karpukhin et al., ACL20]
- **Training methods**
  a. Negative doc selection, e.g., DPR [Karpukhin et al., ACL20], ANCE [Microsoft, ICLR 21]
  b. Distillation, e.g., TCT-ColBERT [Lin et al., arXiv20], RocketQA [Qu et al., ACL21]

Dense Retrieval: Approximate Nearest Neighbor Search

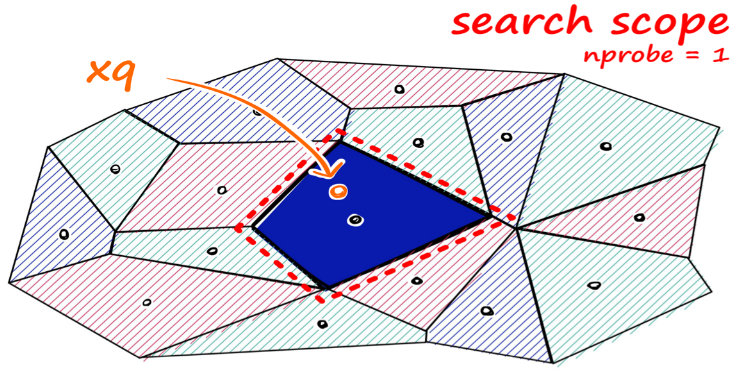
Why? Slow online dot product computing with many documents



- Given a query vector, return the list of document vectors that have the highest dot product with this query vector.
- Two-level index of document vectors with quantization.
  - First level: centroid of each cluster;
  - second level: difference to centroid with residual vectors

$$y \approx q(y) = q_1(y) + q_2(y - q_1(y))$$

- Approximate nearest neighbor search: only go into the clusters that are close to the query.



- FAISS provides fast implementation and GPU support. [Facebook, 2017. IEEE Trans. Big Data 21]

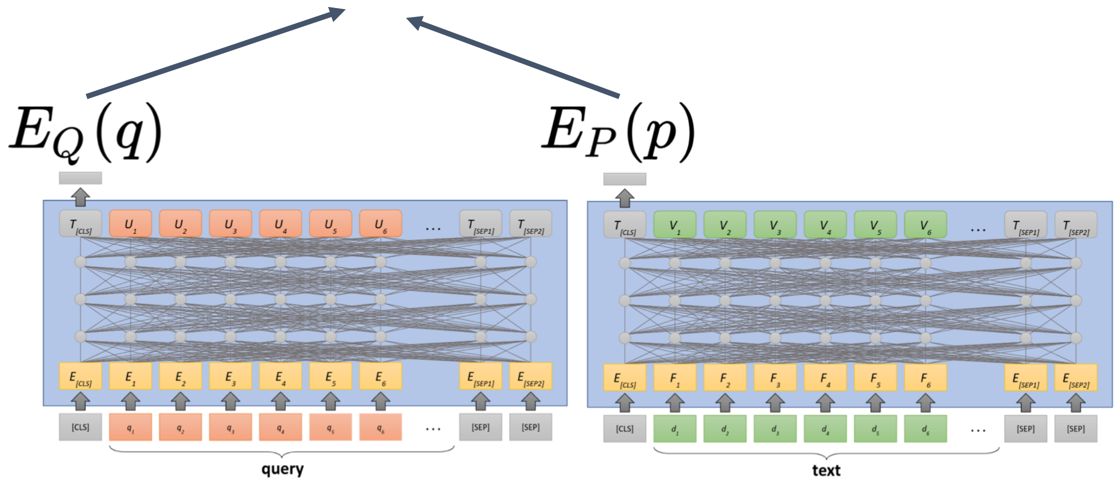
Dense Passage Retrieval (DPR)

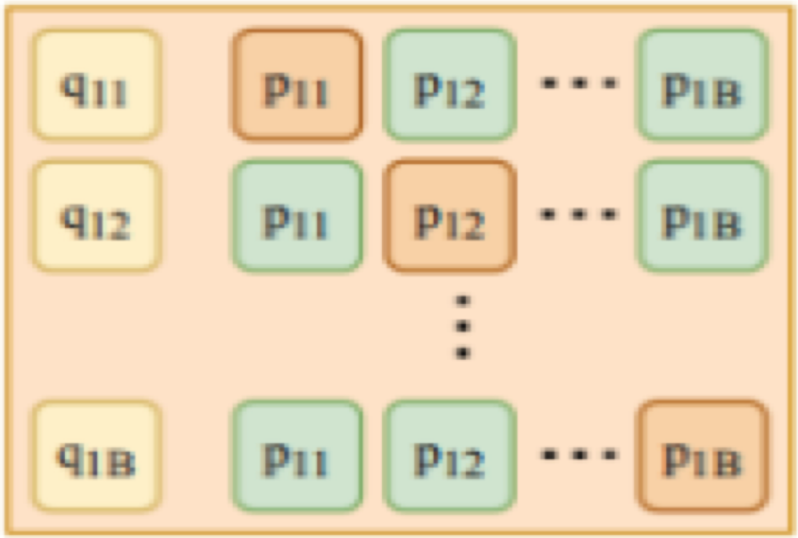
[Karpukhin et al., Facebook, EMNLP20]

- Embedding vectors of the query and the document are derived from the [CLS] token.
- Each training epoch executes a set of batches. Each batch contains training instances of (question, answer) pairs

Each instance is converted as as (question, answer, N negatives)

$$sim(p, q) = E_Q(q)^T E_P(p)$$





Strategies to chose negative passages

- Randomly
- Use BM25 retrieval to select top non-answer results
- Gold: Use answers for other questions
- In-batch Gold: Use other questions from the same batch
- In-batch Gold + 1 BM25-selected negative

DPR: Evaluation with Question Answer Datasets

Natural Question dataset with 59K training examples (Google queries, Wikipedia answers)

Batch size:8 to 128.   40 epochs (#passes to work through the entire training dataset)

Report mean recall@k: %queries that have an answer retrieve at top k.

Best performance: 127 in-batch negatives +1 BM25 hard negative

BM25, Top 20: 59.1 Top 100: 73.7

Type	#N	IB	Top-5	Top-20	Top-100
Random	7	✗	47.0	64.3	77.8
BM25	7	✗	50.0	63.3	74.8
Gold	7	✗	42.6	63.1	78.3
Gold	7	✓	51.1	69.1	80.8
Gold	31	✓	52.1	70.8	82.1
Gold	127	✓	55.8	73.0	83.1
G.+BM25$^{(1)}$	31+32	✓	65.0	77.3	84.4
G.+BM25$^{(2)}$	31+64	✓	64.5	76.4	84.0
G.+BM25$^{(1)}$	127+128	✓	**65.8**	**78.0**	**84.9**

In batch negative very effective

1 BM25 negative

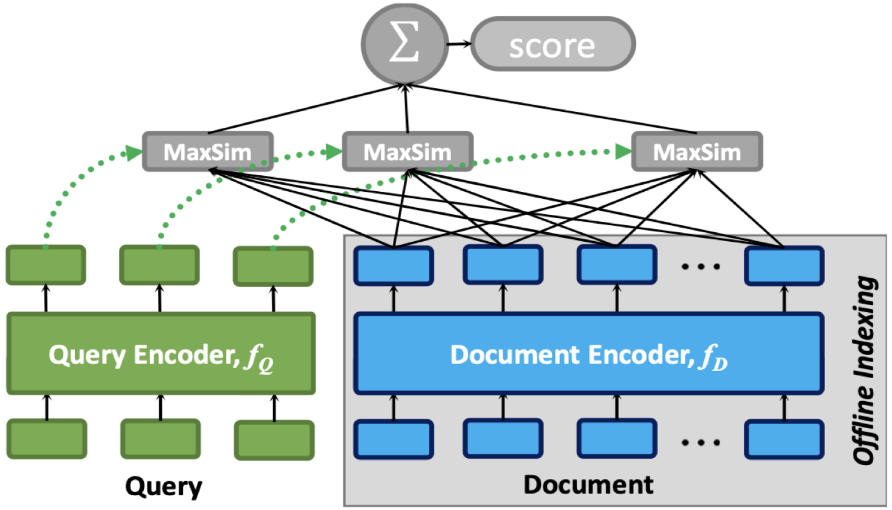
2 BM25 negatives

2 BM25 negatives

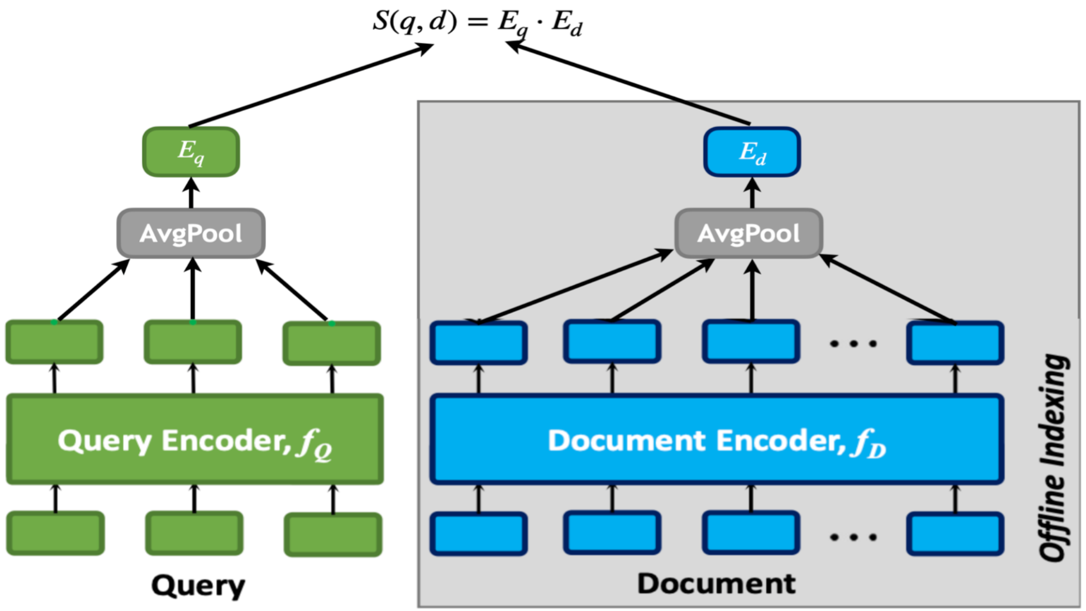
Increase batch size improve performance

A BM25 negative also boosts performance.

Dense Retrieval TCT-ColBERT [Lin et al., UWaterloo, 2020]



ColBERT



TCT-ColBERT

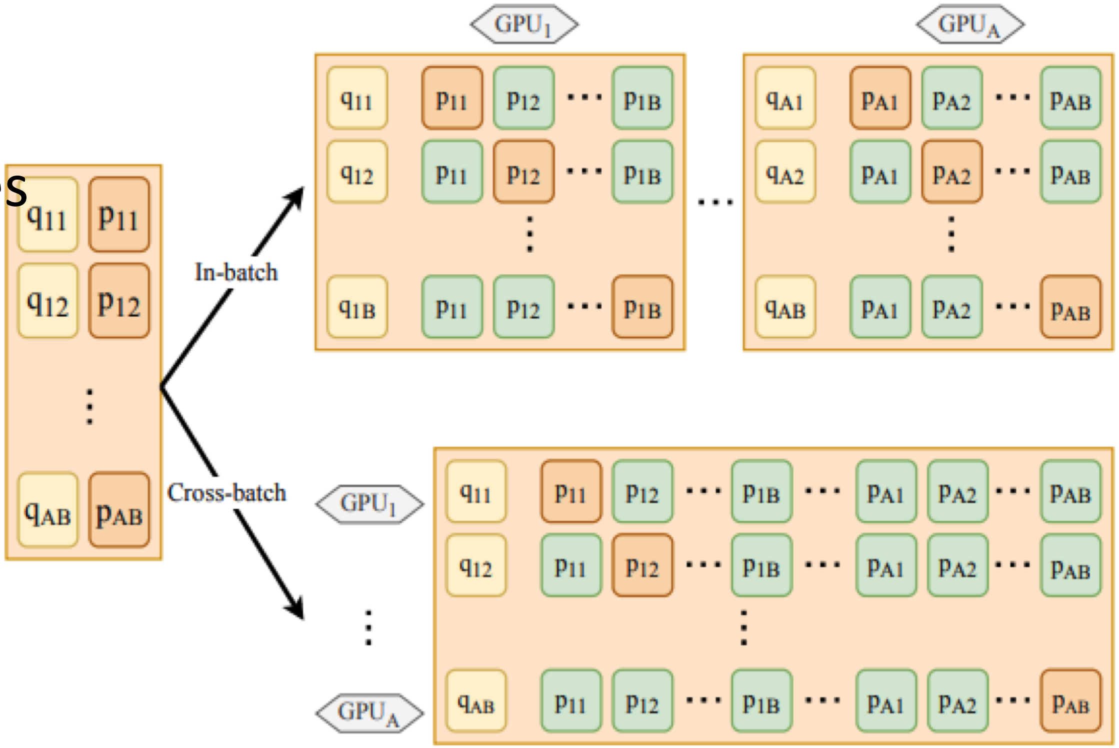
- Simplifies ColBERT structure. The embeddings of query and documents are average pooled.
- Requires knowledge distillation from the original ColBERT model. Teacher: Colbert. Student: TCT-Colbert

Dense Retrieval RocketQA [Qu et al., Baidu, ACL21]

More advanced training strategies

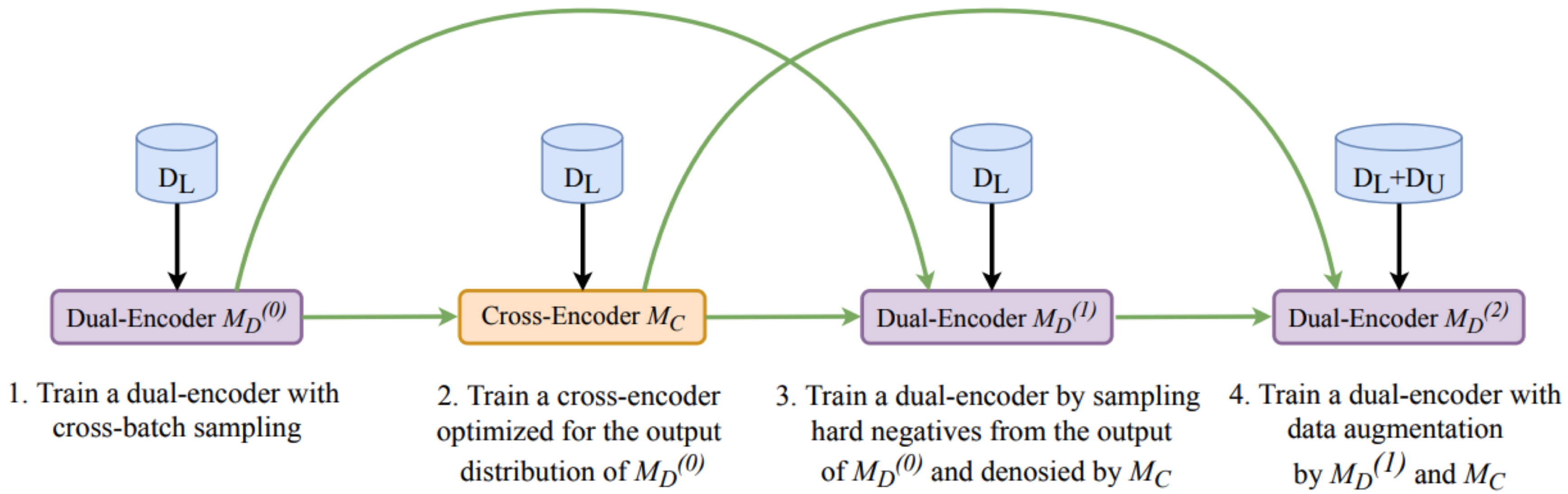
How to build positive/negative pairs

- Cross-batch negatives: Use more negatives from different batches
- Denoising hard negatives: Use a cross-encoder to remove low-confidence negatives
- Data augmentation. Use a cross-encoder to add unsupervised training examples with high-confidence positive and negative passages



RocketQA: An Optimized Training Approach to Dense Passage Retrieval for Open-Domain Question Answering (Qu et al., ACL'21)

- **Chained training pipeline**
  1. Train the dual-encoder on the original dataset.
  2. Train a cross-encoder on the original dataset. Had negatives are selected randomly from the above dual encoder.
  3. Tune the dual-encoder, using the cross-encoder de-noised hard negative samplings.
  4. Expand training data with unsupervised pseudo examples based on the cross-encoder, and use it to further train the dual-encoder.



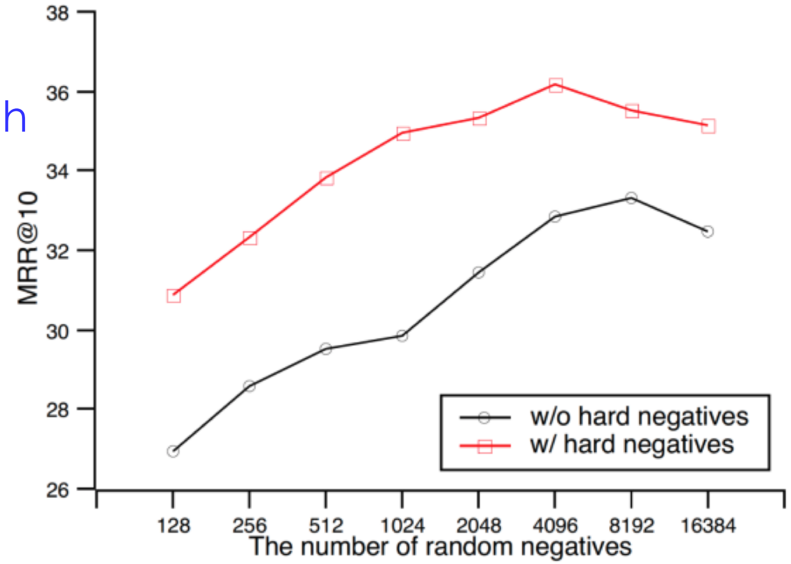
RocketQA Performance and Ablation Studies

Methods	PLMs	MSMARCO Dev			Natural Questions Test		
		MRR@10	R@50	R@1000	R@5	R@20	R@100
BM25 (anserini) (Yang et al., 2017)	-	18.7	59.2	85.7	-	59.1	73.7
doc2query (Nogueira et al., 2019c)	-	21.5	64.4	89.1	-	-	-
DeepCT (Dai and Callan, 2019)	-	24.3	69.0	91.0	-	-	-
docTTTTTquery (Nogueira et al., 2019a)	-	27.7	75.6	94.7	-	-	-
GAR (Mao et al., 2020)	-	-	-	-	-	74.4	85.3
DPR (single) (Karpukhin et al., 2020)	$BERT_{base}$	-	-	-	-	78.4	85.4
ANCE (single) (Xiong et al., 2020)	$RoBERTa_{base}$	33.0	-	95.9	-	81.9	87.5
ME-BERT (Luan et al., 2020)	$BERT_{large}$	33.8	-	-	-	-	-
RocketQA	$ERNIE_{base}$	**37.0**	**85.5**	**97.9**	**74.0**	**82.7**	**88.5**

Strategy	MRR@10
In-batch negatives	32.39
Cross-batch negatives (i.e. STEP 1)	33.32
Hard negatives w/o denoising	26.03
Hard negatives w/ denoising (i.e. STEP 3)	36.38
Data augmentation (i.e. STEP 4)	**37.02**

Cross batch

Gain from denoising



Training Optimization: Summary

- **Optimizing Training Sample Selection**

How to get negatives more easily?

a. In batch negatives: DPR (Karpukhin et al., ACL'20)
b. Cross batch negatives and denoise: RocketQA (Qu et al., ACL'21)

How to get hard negatives that can guide the model better?

a. Asynchronous negative sampling: ANCE (Xiong et al., ICLR'21)

Index update which is expensive!

- **Cross Architecture Distillation**
  - marginMSE (Hofstatter et al., 2020)
  - RocketQAv2 (Ren et al., EMNLP'21)
  - TCT-ColBERT (Lin et al., ACL-Rep4nlp'21)
  - TAS-B (Lin et al., SIGIR'21)

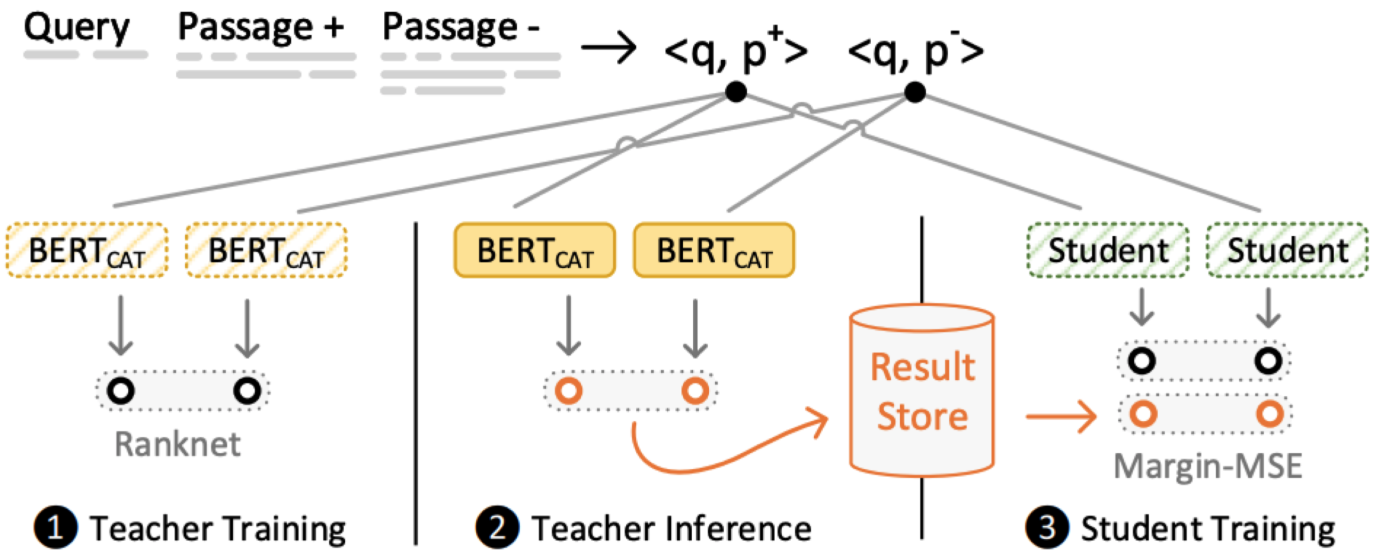
Pairwise distillation

Listwise distillation

Distill from Dual Encoder

Depending on negatives, distill from both cross-encoder and dual encoder

Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation (Hofstatter et al., 2020)



Margin-MSE loss: Relevance difference between rank scores of positive and negative passages

$$\mathcal{L}(Q, P^+, P^-) = \mathrm{MSE}(\boxed{M_s(Q, P^+) - M_s(Q, P^-)}, M_t(Q, P^+) - M_t(Q, P^-))$$

Let the student learn the difference in the teacher's model

Model	Teacher	TREC DL Passages 2019 nDCG@10	TREC DL Passages 2019 MRR@10	TREC DL Passages 2019 MAP@1000	MSMARCO DEV nDCG@10	MSMARCO DEV MRR@10	MSMARCO DEV MAP@1000
Baselines							
BM25	–	.501	.689	.295	.241	.194	.202
TREC Best Re-rank [45]	–	.738	.882	.457	–	–	–
BERT_{CAT} (6-Layer Distilled Best) [14]	–	.719	–	–	–	.356	–
$\text{BERT-Base}_{\text{DOT}}$ ANCE [44]	–	.677	–	–	–	.330	–
Teacher Models							
T1 $\text{BERT-Base}_{\text{CAT}}$	–	.730	.866	.455	.437	.376	.381
$\text{BERT-Large-WM}_{\text{CAT}}$	–	.742	.860	.484	.442	.381	.385
$\text{ALBERT-Large}_{\text{CAT}}$	–	.738	**.903**	.477	.446	.385	.388
T2 Top-3 Ensemble	–	**.743**	.889	**.495**	**.460**	**.399**	**.402**
Student Models							
$\text{DistilBERT}_{\text{CAT}}$	–	.723	.851	.454	.431	.372	.375
	T1	.739	.889	.473	.440	.380	.383
	T2	**.747**	**.891**	**.480**	**.451**	**.391**	**.394**
PreTT	–	.717	.862	.438	.418	.358	.362
	T1	**.748**	**.890**	**.475**	.439	.378	.382
	T2	.737	.859	.472	**.447**	**.386**	**.389**
ColBERT	–	.722	.874	.445	.417	.357	.361
	T1	.738	.862	.472	.431	.370	.374
	T2	**.744**	**.878**	**.478**	**.436**	**.375**	**.379**
$\text{BERT-Base}_{\text{DOT}}$	–	.675	.825	.396	.376	.320	.325
	T1	.677	.809	.427	.378	.321	.327
	T2	**.724**	**.876**	**.448**	**.390**	**.333**	**.338**
$\text{DistilBERT}_{\text{DOT}}$	–	.670	.841	.406	.373	.316	.321
	T1	.704	.821	.441	.388	.330	.335
	T2	**.712**	**.862**	**.453**	**.391**	**.332**	**.337**
TK	–	.652	.751	.403	.384	.326	.331
	T1	**.669**	**.813**	.414	.398	.339	.344
	T2	.666	.797	**.415**	**.399**	**.341**	**.345**

Model	KD Loss	nDCG@10	MRR@10	MAP@100
ColBERT	–	.417	.357	.361
	Weighted RankNet	.417	.356	.360
	Pointwise MSE	.428	.365	.369
	Margin-MSE	**.431**	**.370**	**.374**
BERT_{DOT}	–	.373	.316	.321
	Weighted RankNet	.384	.326	.332
	Pointwise MSE	.387	.328	.332
	Margin-MSE	**.388**	**.330**	**.335**
TK	–	.384	.326	.331
	Weighted RankNet	.387	.328	.333
	Pointwise MSE	.394	.335	.340
	Margin-MSE	**.398**	**.339**	**.344**

T1 vs T2: ensemble teacher leads to stronger student

margin-MSE is effective compared to other two distillation losses.

$$\mathcal{L}(Q, P^+, P^-) = \text{RankNet}(M_s(Q, P^+) - M_s(Q, P^-)) * ||M_t(Q, P^+) - M_t(Q, P^-)||$$

Summary

- Time Efficiency Optimization for Faster BERT-based Neural Ranking
    - Neural net simplification
    - Dual-encoders with precomputed document embeddings
    - CPU-friendly design with query embedding approximation
- Space Efficiency Optimization for BERT-based Ranking
    - Document representation compression with dimension reduction or encoding
    - Contextual embedding quantization
- Document Retrieval: Revisited
    - Learned sparse representations
        - Document expansion by adding more relevant terms to each document
        - Use neural models to compute weights
    - Dense representations
        - Single or multi vector representation
        - Approximation with nearest neighbor search
        - Training optimization by knowledge distillation and adding more positive/negatives