

# **CoBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT**

Stanford University

SIGIR'20

# 모르는 용어 정리

- Information Retrieval (IR) : 정보탐색. 구조화되지 않은 텍스트 데이터를 찾는 작업. **쿼리를 받아 수 많은 문서들 중 적절한 문서들을 찾고 순위를 매겨 돌려주는 작업.**
- Mean Reciprocal Rank (MRR) : 사용자가 선호하는 아이템이 리스트 중 어디에 위치해 있는지를 중점을 둔 평가 기법. (MRR은 랭킹에서 첫 번째 정확한 결과를 찾는 데 중점)

추천된 리스트가 있고 사용자가 선호하는 아이템이  $k_u$ 라면, reciprocal rank는  $\frac{1}{k_u}$ 이다.

10개의 추천 리스트 중 사용자가 선호하는 아이템이 7번째에 있다면 reciprocal rank는  $\frac{1}{7}$ 이 되는 것이다.

그리고 MRR은 각 사용자들에게 추천된 아이템 리스트의 RR을 구해 평균값을 낸 것이다. MRR의 수식은 다음과 같다:

$$MRR(O, U) = \frac{1}{|U|} \sum_{u \in U} \frac{1}{k_u}$$

예제)

사용자 1, 2, 3, 4에게 아이템 A, B, C, D가 추천되었다.

(이해를 돋기 위해 모두에게 같은 순서의 리스트가 제공되었다고 가정하였다. 따라서 실제 추천 시나리오에서는 모델에서 각각의 사용자에게 제공되는 리스트가 다를 수도 있다.)

선호도 리스트:

1: [A, B, C, D]

[X, X, O, O]

2: [A, B, C, D]

[O, X, X, O]

3: [A, B, C, D]

[X, O, X, X]

4: [A, B, C, D]

[X, X, X, O]

아이템 선호 RR

1 1/3

2 1

3 1/2

4 1/4

위의 경우 MRR은  $\frac{\frac{1}{3} + 1 + \frac{1}{2} + \frac{1}{4}}{4}$  가 된다.

# 모르는 용어 정리

- BM25
  - 주어진 쿼리에 대해 문서와의 연관성을 평가하는 랭킹 함수
  - TF-IDF 계열의 검색 알고리즘 중 SoTA
  - <https://littlefoxdiary.tistory.com/12>

$$score(D, Q) = \sum_{i=1}^n IDF(q_i) * \frac{f(q_i, D) * (k_1 + 1)}{f(q_i, D) + k_1 * (1 - b + b * \frac{|D|}{avgdl})}$$

문서  $D$ 에서  $q_i$ 의 term frequency  
문서  $D$ 의 길이  
 $|D|$   
문서 집합의 평균 문서 길이  
 $avgdl$

파라메터

# 모르는 용어 정리

- Contextualized embedding ( feat. ChatGPT )

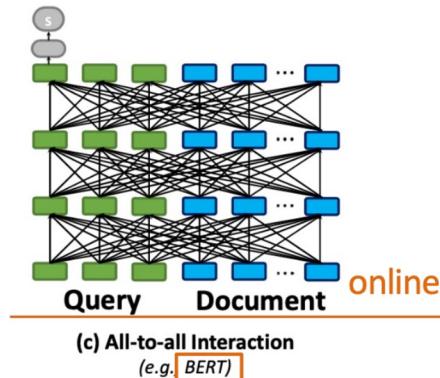
- 기본적인 단어 임베딩(Word Embedding)은 각 단어를 고정된 벡터로 매핑하는 방법을 사용합니다. 이러한 기본적인 임베딩은 모든 문장에서 동일한 임베딩을 사용하며, 단어의 의미나 역할에 대한 컨텍스트를 고려하지 않습니다. 예를 들어, "바다에서 낚시를 한다"와 "낚시를 한 뒤에 바다에서 휴식을 취한다"라는 두 문장에서 "바다"라는 단어는 다른 의미를 가집니다. 그러나 기본적인 단어 임베딩은 이러한 차이를 인식하지 못합니다. -> 기본적인 단어 임베딩 방식으로는 동일한 단어는 동일한 임베딩 값을 가짐
- 컨텍스트 임베딩은 이러한 한계를 극복하기 위해 도입되었습니다. 이는 주변 문맥을 고려하여 단어의 임베딩을 생성합니다. 예를 들어, "바다에서 낚시를 한다" 문장에서 "바다"라는 단어의 임베딩은 주변 문맥에 따라 달라질 수 있으며, "낚시를 한 뒤에 바다에서 휴식을 취한다" 문장에서는 다른 임베딩을 가질 수 있습니다. 이렇게 하면 단어의 의미가 주변 문맥에 따라 조금씩 변화하는 것을 캡처할 수 있습니다. -> contextualized 임베딩은 주변 문맥을 고려하여 단어의 임베딩을 생성하는 것
- 컨텍스트 임베딩을 생성하는 방법 중 하나는 Transformer 모델을 사용하는 것인데, 이 모델은 주로 "BERT"나 "GPT"와 같은 다양한 NLP 작업에 사용되며, 단어나 문장의 컨텍스트를 효과적으로 파악할 수 있습니다. 따라서 컨텍스트 임베딩은 텍스트 처리 분야에서 넓은 활용 범위를 가진 기초적인 개념입니다.

# 모르는 용어 정리

- Late Interaction

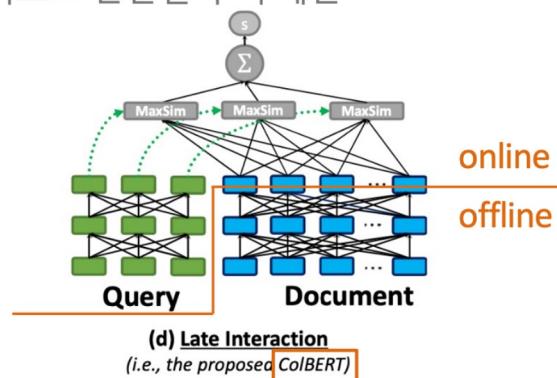
## All-to-all Interaction (BERT)

- 질의와 문서를 함께 계산함
- 검색에서 쓰기에 비현실적인 연산량



## Late Interaction (ColBERT)

- 질의와 문서를 분리하여 계산
- 문서의 임베딩만 미리 구축 가능
- 질의와 문서 연관도를 계산하기 위해 빠르고 간단한 수식 제안



# Content

1. Introduction
2. Related Work
3. ColBERT
4. Experimental Evaluation
5. Conclusions

# Introduction

- Information Retrieval (IR) community에서 neural ranking models이 많이 소개되었음
  - E.g. DRMM, KNRM, and Duet
- 상기 neural ranking 모델은 queries, documents의 embedding-based representations를 적용하여 q, d 간 local interactions을 계산함
- 최근 연구는 relevance를 계산하기 위해 deep pre-trained language models (LMs)을 fine-tunes하여 사용함
  - E.g. ELMo, BERT
  - 이렇게 deeply-contextualized semantic representations of query-document pairs를 계산함으로써, LMs은 q, d 간 pervasive vocab mismatch(만연한 어휘 불일치)를 완화할 수 있음
- 실제로 BERT를 사용한 ranking models은 SoTA 달성

# Introduction

- 그러나 이러한 LMs 의 remarkable gains 는 computational cost 을 증가시켰음
- Quality-cost tradeoff
  - Re-rank the MS MARCO’s official top-1000 documents per query
    - ColBERT (re-rank), the Neural Matching Models, the Deep LMs
  - Directly retrieve the top-1000 results from the entire collection
    - Other methods, including ColBERT (full retrieval)

# Introduction

- 그러나 이러한 LMs 의 remarkable gains ㄴ computational cost 으 증가시켜요
- Quality-cost tradeoff

- Re-rank the MS MARCO's official results
  - ColBERT (re-rank), the Neural Matching Model
- Directly retrieve the top-1000 results
  - Other methods, including

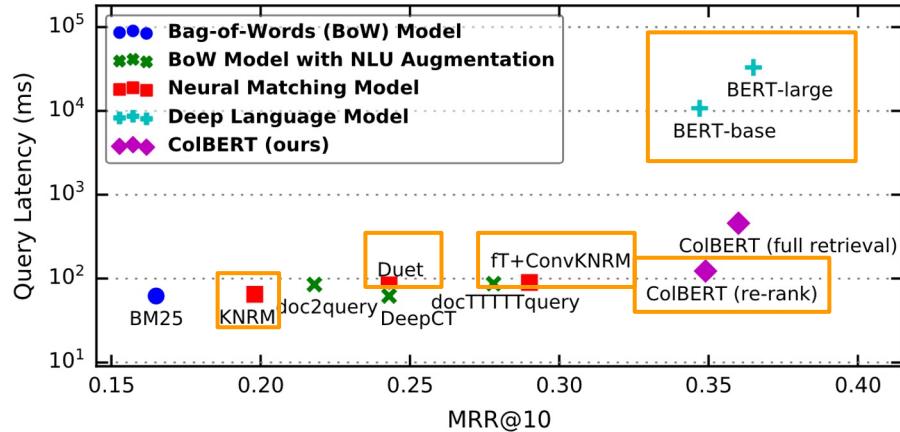


Figure 1: Effectiveness (MRR@10) versus Mean Query Latency (log-scale) for a number of representative ranking models on MS MARCO Ranking [24]. The figure also shows ColBERT. Neural re-rankers run on top of the official BM25 top-1000 results and use a Tesla V100 GPU. Methodology and detailed results are in §4.

# Introduction

- Figure 1에서 볼 수 있듯이, BERT는 MRR@10을 거의 7% 정도 향상시켰으나, high-end GPU임에도 불구하고 latency도 함께 증가했음
- This poses a challenging tradeoff since raising query response times by as little as 100ms is known to impact user experience and even measurably diminish revenue
- 위 문제를 해결하기 위해, 최근 연구에선 Natural Language Understanding (NLU)을 traditional retrieval models (BM25)에 augment시키기도 하였음. 그 결과 latency는 감소하였으나, 일반적으로 BERT에 비해 상당히 정확도를 감소시킴

# Introduction

- IR에서 efficiency와 contextualization을 모두 고려하기 위해서, **CoBERT** 제안
  - Ranking model based on **contextualized late interaction over BERT**
- **CoBERT**
  - q, d 간 relevance를 측정하기 위한 새로운 late interaction paradigm 제안
  - q, d는 각각 contextual embedding으로 embedding되며, relevance는 q, d 간 cheap and pruning-friendly computations으로 계산됨

# Introduction

- Contrasts with existing neural matching paradigms

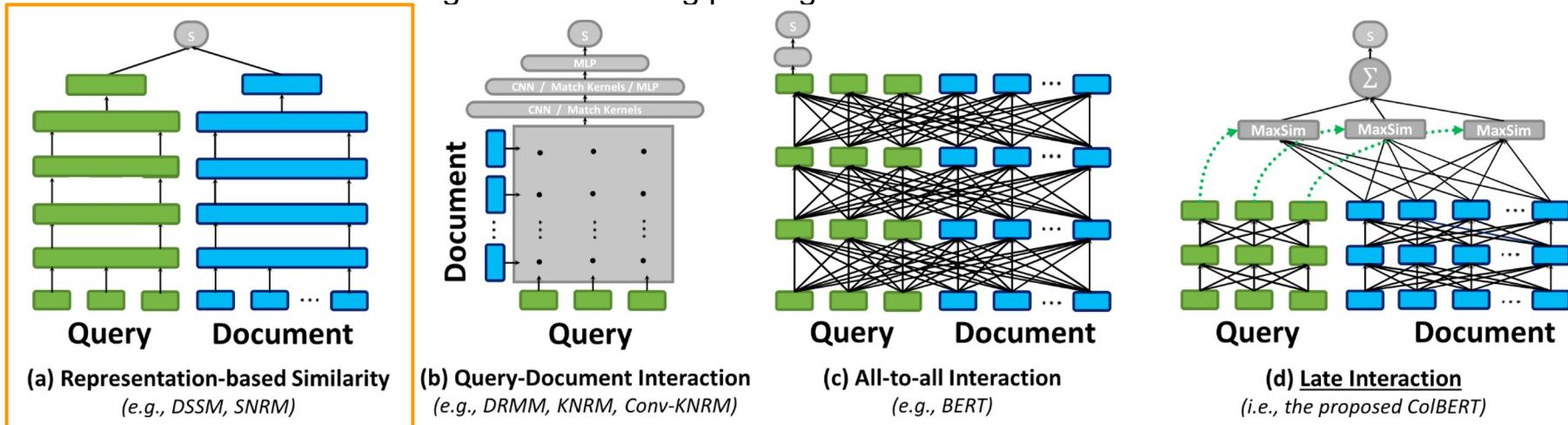


Figure 2: Schematic diagrams illustrating query–document matching paradigms in neural IR. The figure contrasts existing approaches (sub-figures (a), (b), and (c)) with the proposed late interaction paradigm (sub-figure (d)).

- Representation-focused rankers** : Independently compute an embedding for  $q$  and another for  $d$  and estimate relevance as a single similarity score between two vectors

# Introduction

- Contrasts with existing neural matching paradigms

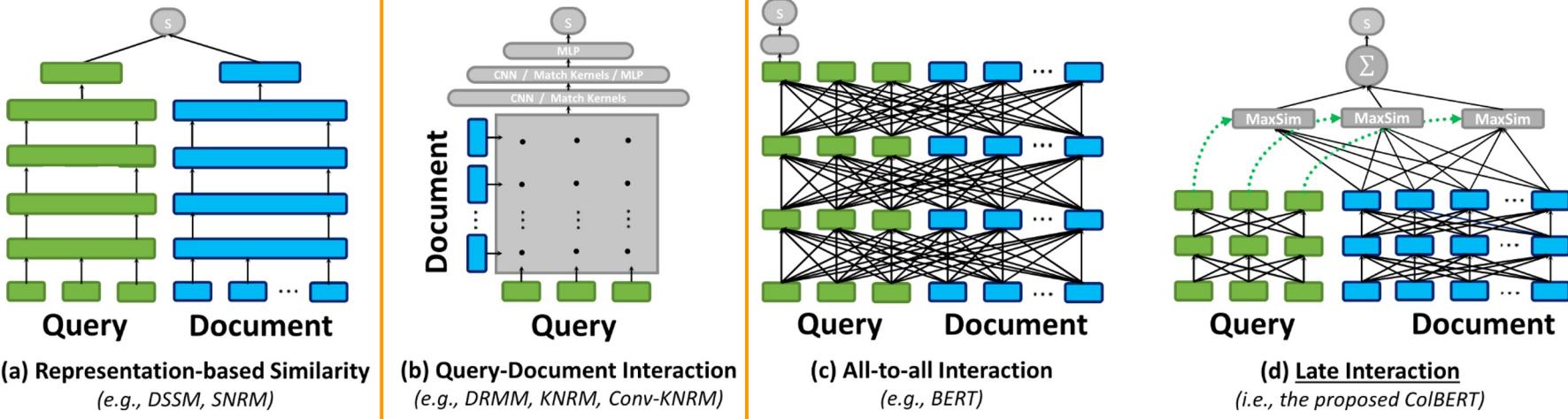
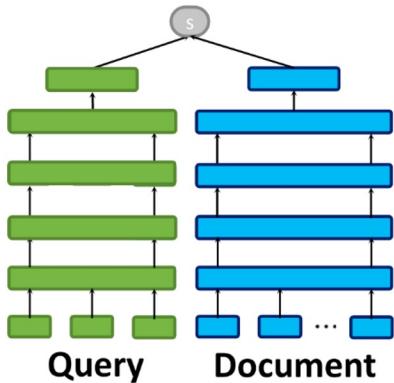


Figure 2: Schematic diagrams illustrating query–document matching paradigms in neural IR. The figure contrasts existing approaches (sub-figures (a), (b), and (c)) with the proposed late interaction paradigm (sub-figure (d)).

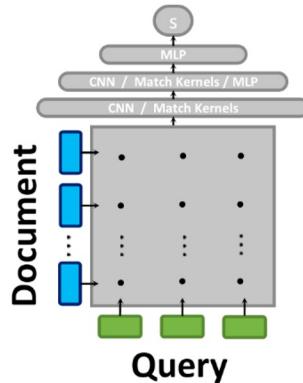
- Interaction-focused rankers**: These rankers model word- and phrase-level relationships across  $q$  and  $d$  and match them using a deep neural network (e.g., with CNNs/MLPs or kernels)

# Introduction

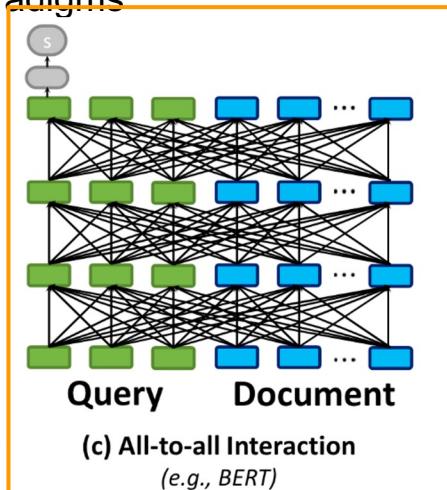
- Contrasts with existing neural matching paradigms



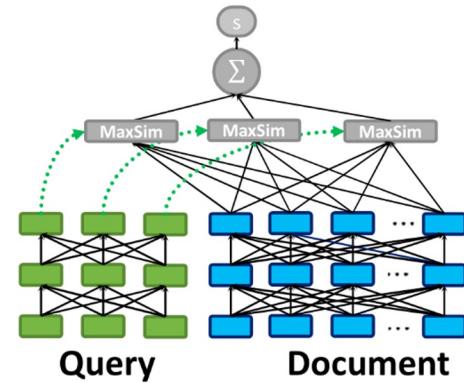
(a) Representation-based Similarity  
(e.g., DSSM, SNRM)



(b) Query-Document Interaction  
(e.g., DRMM, KNRM, Conv-KNRM)



(c) All-to-all Interaction  
(e.g., BERT)



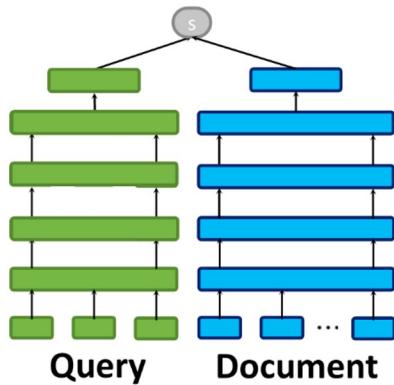
(d) Late Interaction  
(i.e., the proposed ColBERT)

Figure 2: Schematic diagrams illustrating query–document matching paradigms in neural IR. The figure contrasts existing approaches (sub-figures (a), (b), and (c)) with the proposed late interaction paradigm (sub-figure (d)).

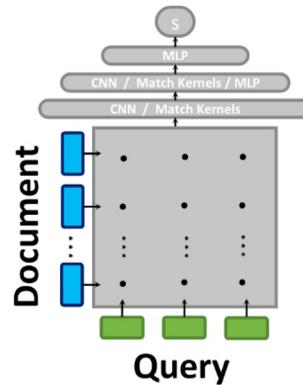
- Interactions between words within as well as across q and d at the same time, as in BERT's transformer architecture

# Introduction

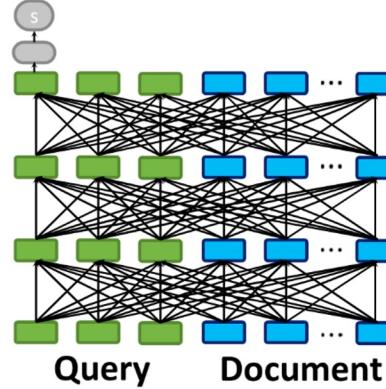
- Contrasts with existing neural matching paradigms



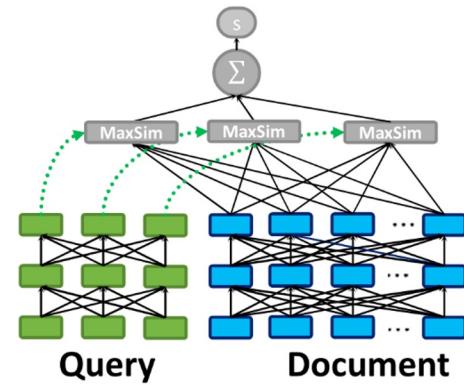
(a) Representation-based Similarity  
(e.g., DSSM, SNRM)



(b) Query-Document Interaction  
(e.g., DRMM, KNRM, Conv-KNRM)



(c) All-to-all Interaction  
(e.g., BERT)



(d) Late Interaction  
(i.e., the proposed CoBERT)

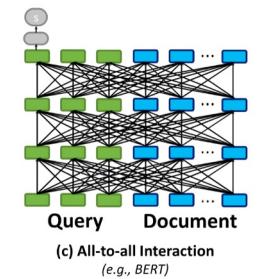
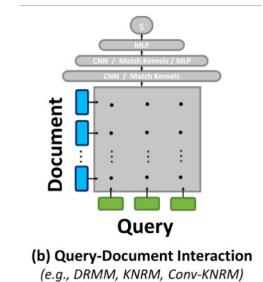
- document representation 을 미리 만 들어 둘 수 있음 (offline)
  - IR 성능 좋음

# Introduction

- Main contributions
  - Propose late interaction as a paradigm for efficient and effective neural ranking
  - Present ColBERT, a highly-effective model that employs novel BERT-based query and document encoders within the late interaction paradigm
  - Show how to leverage ColBERT both for re-ranking on top of a term-based retrieval model and for searching a full collection using vector similarity indexes
  - Evaluate ColBERT on MS MARCO and TREC CAR, two recent passage search collections

# Related Work

- Neural Matching Models
  - IR researchers have introduced numerous neural architectures for ranking
  - KNRM, Duet, ConvKNRM, and fastText+ConvKNRM
- Language Model Pretraining for IR
  - 최근 NLU에서 downstream task로 fine-tuning하기 전에 pre-training language representation models의 중요성이 강조되고 있음
  - BERT, ELMo 등



# Related Work

- BERT Optimizations
  - Distilling, compressing, pruning BERT 를 NLU 을 최적화했을 때, 약간 (smaller) speedups 을 달성할 수 있었으나, quality 가 떨어졌음
- Efficient NLU-based Models
  - NLU 모델을 traditional retrieval models like BM25 와 augment
  - BoW Model with NLU Augmentation
  - doc2query, DeepCT, docTTTTquery
  - 그러나 성능이 떨어짐

# COLBERT

- Prescribes a simple framework for balancing the quality and cost of neural IR
  - Particularly deep language models like BERT
- ColBERT's late-interaction framework 는 다양한 아키텍처 (e.g., CNNs, RNNs, transformers, etc)에서 사용될 수 있지만 본 논문에선 BERT (bi-directional transformer-based encoders) 를 사용했음

# COLBERT

- COLBERT 아키텍처 (3개 구성요소)
  - A query encoder  $f_{\{Q\}}$
  - A document encoder  $f_{\{D\}}$
  - Late interaction mechanism
- query  $q$ , document  $d$  가 주어졌을 때,  $f_Q$  는  $q$  를 fixed-size embeddings  $E_q$  로 embedding,  $f_D$  는  $d$  를  $E_d$  로 embedding
- 이 때의  $E_q$ ,  $E_d$  의 contextualized 함
- $E_q$ ,  $E_d$  를 통해 relevance score 계산 via late interaction ( We define MaxSim operators )

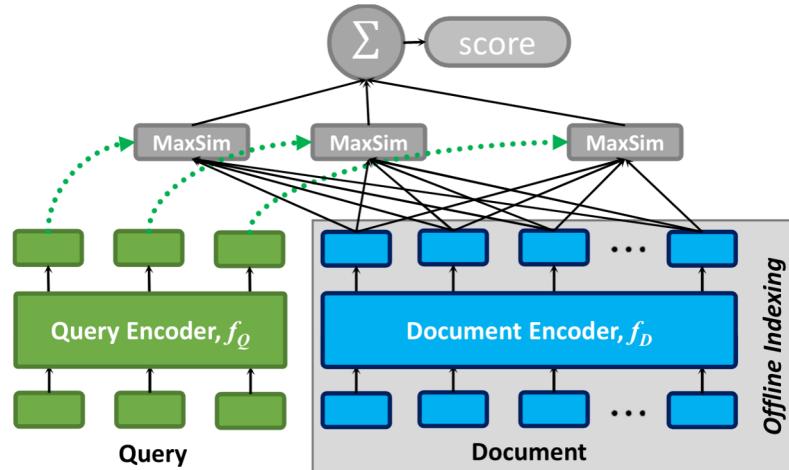


Figure 3: The general architecture of ColBERT given a query  $q$  and a document  $d$ .

# COLBERT

- Query & Document Encoders
  - ColBERT encodes each query or document into a bag of embeddings, employing BERT-based encoders
  - Query & document encoder 는 single BERT model 을 공유함
  - Special token [Q] to queries, [D] to documents 를 사용하여 input sequences 를 구분

# COLBERT

- Query Encoder
  - Given a textual query  $q$ , we tokenize it into its BERT-based WordPiece tokens  $q_1 q_2 \dots q_l$
  - [CLS] token + [Q] token + query 순으로 배치
  - 만약 query 가 pre-defined 한 토큰 길이보다 짧다면, BERT's special [mask] tokens 을 padding, 만약 길다면 truncate it to the first  $N_q$  tokens
  - 이렇게 padding 한 input tokens 은 BERT deep transformer architecture 로 입력되어 각 token 의 contextualized representation 이 계산됨
  - **mask token** 을 **query augmentation** 라고 함. 이는 BERT 가 mask 에 해당하는 위치에서 쿼리 기반 임베딩 (query-based embedding) 을 생성할 수 있도록 함. 이 작업은 ColBERT 의 효과에 필수적임
  - Token 별로 contextualized output representation 이 생성되면 linear layer with no activations 태워  $m$ -dimensional embedding 함 (일반적으로  $m$ 은 BERT dimen 보다 작게하려고 함)

# COLBERT

- Document Encoder
  - First segment a document  $d$  into its constituent tokens  $d_1 d_2 \dots d_m$
  - [CLS] token + [D] token + document 순으로 나열
  - Query 와는 다르게 [mask] token 사용하지 않음
  - 위 input sequence 를 BERT, linear layer 를 거침
  - 이후 미리 정한 리스트를 통해 구두점 기호 등 필터링하여 문서당 임베딩 수 줄임
- In summary, given  $q = q_0 q_1 \dots q_l$  and  $d = d_0 d_1 \dots d_n$ , we compute the bags of embeddings  $E_q$  and  $E_d$  in the following manner, where  $\#$  refers to the [mask] token.

---

$$E_q := \text{Normalize}(\text{CNN}(\text{BERT}("[Q]q_0 q_1 \dots q_l \#\#\#\#")))) \quad (1)$$

$$E_d := \text{Filter}(\text{Normalize}(\text{CNN}(\text{BERT}("[D]d_0 d_1 \dots d_n"))))) \quad (2)$$

# COLBERT

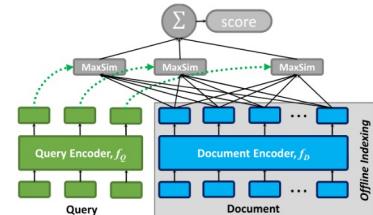


Figure 3: The general architecture of ColBERT given a query  $q$  and a document  $d$ .

- Late Interaction

- query  $q$  와 document  $d$  의 relevance score  $S_{\{q,d\}}$
- $S_{\{q,d\}}$  은 각 contextualized embeddings 간 late interaction 을 통해 계산됨
- Sum of maximum similarity computations, namely cosine similarity ( or squared L2 distance )

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

- Interaction mechanism 은 trainable parameters 가 필요 없음
- Given a triple  $\langle q, d+, d-\rangle$  with query  $q$ , positive document  $d+$  and negative document  $d-$ , ColBERT is used to produce a score for each document individually
- ColBERT is optimized via pairwise softmax cross entropy loss over the computed scores of  $d+$ ,  $d-$

# COLBERT

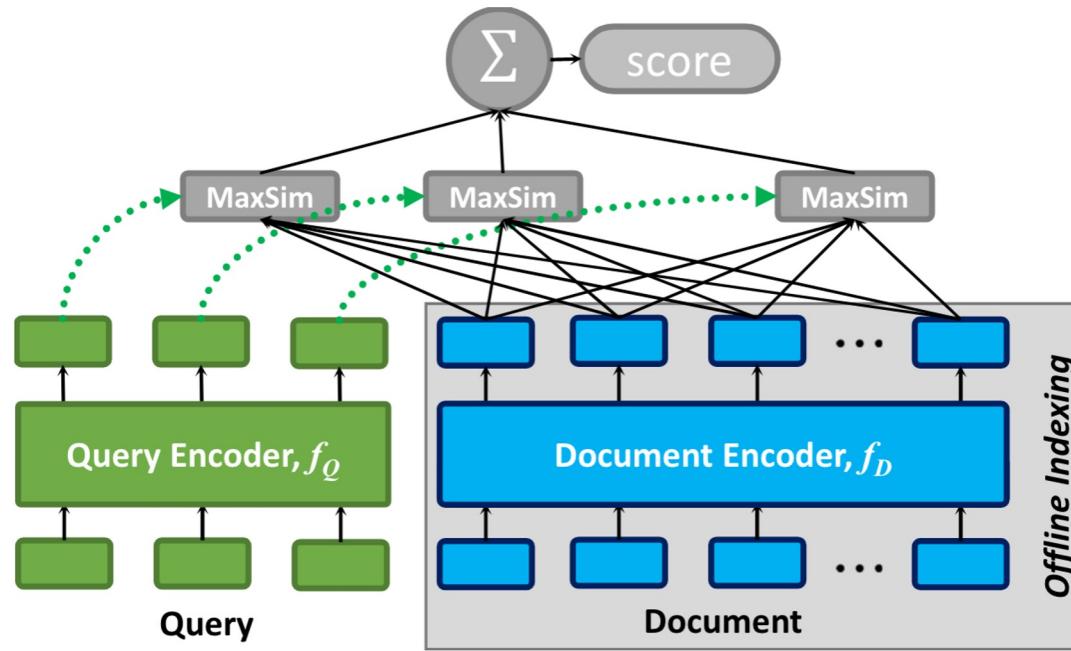


Figure 3: The general architecture of ColBERT given a query  $q$  and a document  $d$ .

# COLBERT

- Offline indexing: Computing & Storing Document Embeddings
  - Query 와 document indexing 과정을 분리시켜 document representations 을 offline 으로 미리 계산할 수 있음
  - Document indexing 은 offline process 이지만, throughput of indexing 을 향상시키기 위해 simple optimizations 사용
  - Multi GPU 병렬 처리
  - 효율적인 처리를 위해서 문서의 길이 별로 정렬한 후, 비슷한 길이의 문서를 하나의 batch 에 할당. 이렇게 length-based bucketing 은 BucketIterator 라고 불리며 allenNLP 등 라이브러리 사용 가능
  - Batch 내에서 가장 긴 문서 길이로 padding

# COLBERT

- Offline indexing: Computing & Storing Document Embeddings
  - 대부분의 연산이 GPU에서 일어나고 있을 때, CPU 병렬화로 text pre-processing 연산
  - Text pre-processing 은 BERT's WordPiece tokenization
  - Document representations 이 계산되면 32-bit or 16-bit values 로 저장

# COLBERT

- Indexing Throughput & Footprint

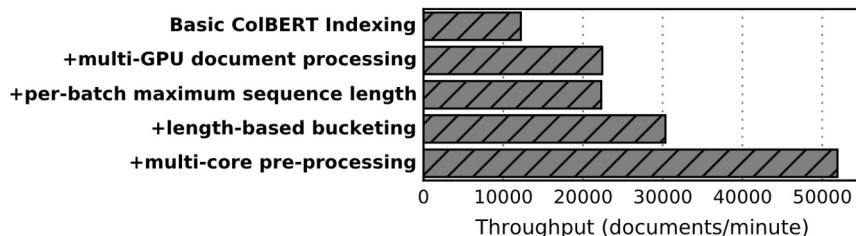


Figure 6: Effect of ColBERT’s indexing optimizations on the offline indexing throughput.

Setting	Dimension( $m$ )	Bytes/Dim	Space(GiBs)	MRR@10
Re-rank Cosine	128	4	286	34.9
End-to-end L2	128	2	154	36.0
Re-rank L2	128	2	143	34.8
Re-rank Cosine	48	4	54	34.4
Re-rank Cosine	24	2	27	33.9

Table 4: Space Footprint vs MRR@10 (Dev) on MS MARCO.

Table 4 reports the space footprint of ColBERT under various settings as we reduce the embeddings dimension and/or the bytes per dimension.

space-efficient setting 이 제일 좋을 때와 나머지 성능에 거의 차이가 없음

# COLBERT

- Top-k Re-ranking with ColBERT
  - ColBERT는 reranking과 end-to-end retrieval에 모두 사용할 수 있음. 이 장에선 ColBERT 를 ranking a small set of k (e.g., k=1000) documents 에 사용하는 방법
  - 메모리에 indexing 된 document representation 을 matrix of embedding 으로 표현
  - 3-dimensional tensor D consisting of k document matrices
  - 그리고 k documents 중 가장 긴 길이에 맞춰 padding
  - Eq 와 D 의 batch dot-product 계산을 통해 k documents 를 total scores 정렬
  - Computation is very cheap

# COLBERT

- End-to-end Top-k Retrieval with ColBERT
  - Focus on retrieving the top-k results directly from a large document collection with  $N$  (e.g.,  $N = 10,000,000$ ) documents, where  $k \ll N$
  - 하나의 query와 모든 document 를 계산하지 않음
  - Two-stage procedure
    - Faiss 와 같은 vector-similarity 라이브러리를 이용해  $\text{top-}k'$  (e.g.,  $k' = k/2$ ) 의 query 와 가까운 document 선정 (중복 제거)
    - 이후 앞서 re-ranking 했던 것과 동일하게  $K$  documents re-ranking

# EXPERIMENTAL EVALUATION

- Addressing the following research questions
  - In a typical re-ranking setup, how well can ColBERT bridge the existing **gap between highly-efficient (효율적인) and highly-effective (효과적인)** neural models? (§4.2)
  - Beyond re-ranking, can ColBERT effectively support **end-to-end retrieval directly** from a large collection? (§4.3)
  - What does **each component of ColBERT (e.g., late interaction, query augmentation) contribute to its quality?** (§4.4)
  - What are **ColBERT's indexing-related costs** in terms of offline computation and memory overhead? (§4.5) => 기 확인함 (ppt 23 page)

# EXPERIMENTAL EVALUATION

- Methodology
  - Datasets
    - MS MARCO Ranking
      - It is a collection of 8.8M passages from Web pages, which were gathered from Bing's results to 1M real-world queries
      - MRR@10 사용하여 평가
    - TREC Complex Answer Retrieval datasets
      - Synthetic dataset based on Wikipedia that consists of about 29M passages

# EXPERIMENTAL EVALUATION

- Methodology
  - Implementation
    - Using Python3 and PyTorch , transformers library for the pre-trained BERT model
    - lr 3\*10-6, batch size 32, fix the number of embeddings per query at Nq=32, ColBERT embedding dimension m to be 128
    - MS MARCO
      - Google's official pre-trained BERTbase model
      - 200k iters
    - TREC Complex Answer Retrieval datasets
      - Nogueira and Cho's 저 BERTlarge pre-trained model
      - 125k iters

# EXPERIMENTAL EVALUATION

In a typical **re-ranking** setup, how well can ColBERT bridge the existing gap between highly-efficient and highly-effective neural models? (§4.2)

- Quality–Cost Tradeoff: Top-k Re-ranking
  - [25] : We compare against the natural adaptation of BERT for ranking by Nogueira and Cho
  - BERTbase (our training) : Same model but is trained with the same loss function as ColBERT

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×)
BERT <sub>base</sub> [25]	34.7	-	10,700	97T (13,900×)
BERT <sub>base</sub> (our training)	36.0	-	10,700	97T (13,900×)
BERT <sub>large</sub> [25]	36.5	35.9	32,900	340T (48,600×)
ColBERT (over BERT <sub>base</sub> )	34.9	34.9	61	7B (1×)

Table 1: “Re-ranking” results on MS MARCO. Each neural model re-ranks the official top-1000 results produced by BM25. Latency is reported for re-ranking only. To obtain the end-to-end latency in Figure 1, we add the BM25 latency from Table 2.

# EXPERIMENTAL EVALUATION

In a typical **re-ranking** setup, how well can ColBERT bridge the existing gap between highly-efficient and highly-effective neural models? (§4.2)

- Quality–Cost Tradeoff: Top-k Re-ranking
  - Top-K에 따른 FLOPs effectiveness (MRR@10) relationships

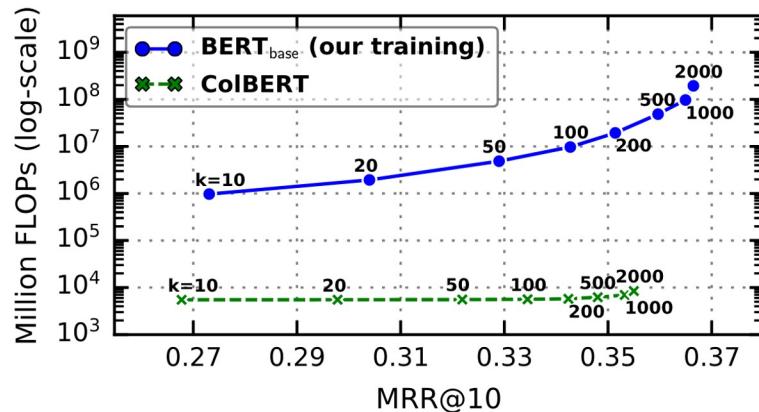


Figure 4: FLOPs (in millions) and MRR@10 as functions of the re-ranking depth  $k$ . Since the official BM25 ranking is not ordered, the initial top- $k$  retrieval is conducted with Anserini’s BM25.

# EXPERIMENTAL EVALUATION

In a typical **re-ranking** setup, how well can ColBERT bridge the existing gap between highly-efficient and highly-effective neural models? (§4.2)

- Quality–Cost Tradeoff: Top-k Re-ranking
  - TREC CAR
  - MAP (Mean Average Precision)

Method	MAP	MRR@10
BM25 (Anserini)	15.3	-
doc2query	18.1	-
DeepCT	24.6	33.2
BM25 + BERT <sub>base</sub>	31.0	-
BM25 + BERT <sub>large</sub>	33.5	-
BM25 + ColBERT	31.3	44.3

**Table 3: Results on TREC CAR.**

# EXPERIMENTAL EVALUATION

Beyond re-ranking, can ColBERT effectively support **end-to-end retrieval directly** from a large collection? (§4.3)

- End-to-end Top-k Retrieval
  - Model retrieves the top-1000 documents directly from MS MARCO’s 8.8M documents per query

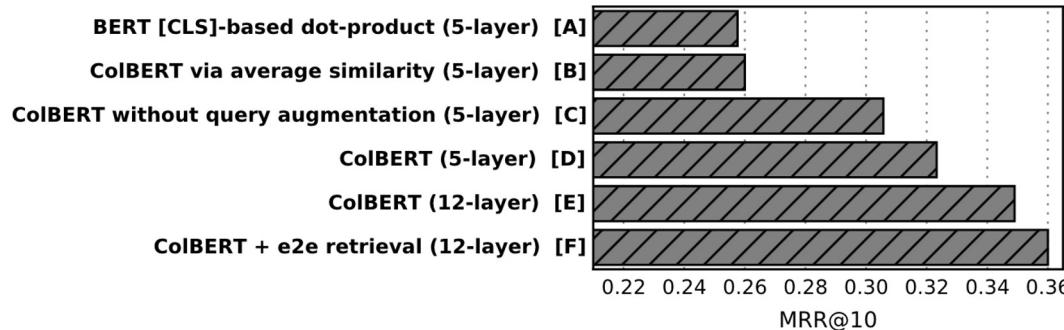
Method	MRR@10 (Dev)	MRR@10 (Local Eval)	Latency (ms)	Recall@50	Recall@200	Recall@1000
BM25 (official)	16.7	-	-	-	-	81.4
BM25 (Anserini)	18.7	19.5	62	59.2	73.8	85.7
doc2query	21.5	22.8	85	64.4	77.9	89.1
DeepCT	24.3	-	62 (est.)	69 [2]	82 [2]	91 [2]
docTTTTTquery	27.7	28.4	87	75.6	86.9	94.7
ColBERT <sub>L2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

Table 2: End-to-end retrieval results on MS MARCO. Each model retrieves the top-1000 documents per query *directly* from the entire 8.8M document collection.

# EXPERIMENTAL EVALUATION

What does each component of ColBERT (e.g., late interaction, query augmentation) contribute to its quality? (§4.4)

- Ablation Studies
  - Examine a number of important details in ColBERT’s interaction and encoder architecture
  - Our main re-ranking ColBERT model, with MRR@10 of 34.9%



**Figure 5: Ablation results on MS MARCO (Dev). Between brackets is the number of BERT layers used in each model.**

# EXPERIMENTAL EVALUATION

What are ColBERT’s indexing-related costs in terms of offline computation and memory overhead? (§4.5)

- Indexing Throughput & Footprint

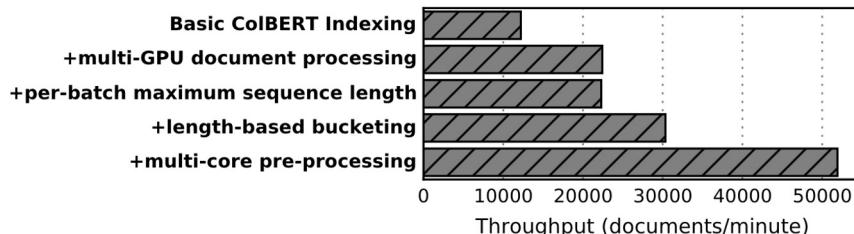


Figure 6: Effect of ColBERT’s indexing optimizations on the offline indexing throughput.

Setting	Dimension( $m$ )	Bytes/Dim	Space(GiBs)	MRR@10
Re-rank Cosine	128	4	286	34.9
End-to-end L2	128	2	154	36.0
Re-rank L2	128	2	143	34.8
Re-rank Cosine	48	4	54	34.4
Re-rank Cosine	24	2	27	33.9

Table 4: Space Footprint vs MRR@10 (Dev) on MS MARCO.

Table 4 reports the space footprint of ColBERT under various settings as we reduce the embeddings dimension and/or the bytes per dimension.

space-efficient setting 이 제일 좋을 때와 나머지 성능에 거의 차이가 없음

# CONCLUSIONS

- Introduced ColBERT, a novel ranking model
- ColBERT employs contextualized late interaction over deep LMs (BERT)
- query 와 document 를 독립적으로 인코딩하면서 cheap and pruning-friendly computations 를 사용하여 ColBERT 는 속도를 향상시키면서 deep LM 의 expressiveness 을 활용하였음
- 또한 ColBERT 는 속도를 BERT-based models 보다 170x 향상시키면서 end-to-end retrieval 이 가능함
- Quality 에는 최소한의 영향을 미쳤고, non-BERT baseline 보다는 항상 성능이 좋음

Thank You

# 추가조사

**Q1. Eq := Normalize( CNN( BERT("[Q]q0q1...ql ##...#") ) )** 수식에 나와있는 CNN 이란 무엇인가?

- embedding 차원 축소를 위한 linear layer 가 맞는 것 같음 -> 왜 CNN 으로 표기했는지 모르겠음
- linear layer 라고 생각한 이유
  - ColBERT 논문에서 CNN 에 대한 추가 언급 없음. our encoder passes the contextualized output representations through a linear layer with no activations. 이렇게 나와있음
  - [스탠포드 코드](#) 에서도 linear 로 구현되어 있음 ( 87line Q = self.linear(Q) )
  - [ColBERT v2](#) 논문에서 ColBERT 간단 리뷰할 때 ‘Queries and passages are independently encoded with BERT (Devlin et al., 2019), and the output embeddings encoding each token are projected to a lower dimension.’ 이렇게 나와있음
  - 여타 사이트를 찾아봐도 CNN 을 사용한다는 말이 없음..

# 추가조사

Q2. 추론에 사용할 때 document의 어떤 embedding 값을 사용하는가?

- Inference 과정
  - ⊖ k 개의 document 를 3-dimension 으로 저장함 ( 즉, #token \* embedding dimension \* k 인 듯? )
  - ⊖ 3-dimension 과 query embedding 을 비교하여 re-ranking 결과를 냅
  - ⊖ 즉, document embedding 은 token embedding 을 그대로 계산해서 사용
  - ⊖ 마찬가지로 e2e 에서 document embedding 이 저장될 때 faiss 에 바로 indexing 해서 사용했다고 함

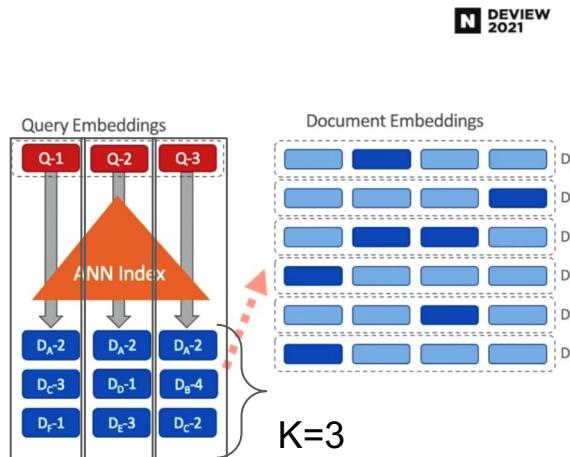
# 추가조사

## Q2. e2e에서 retrieval 과정은 어떻게 되는가?

- Query embedding (Q-1, Q-2, Q-3) 별로 가장 유사한 문서가 아닌 문서 임베딩(DA-2, .., DC-2)을 찾음 (ANN)
- [1.5 ColBERT Retrieval](#)

### ColBERT Retrieval

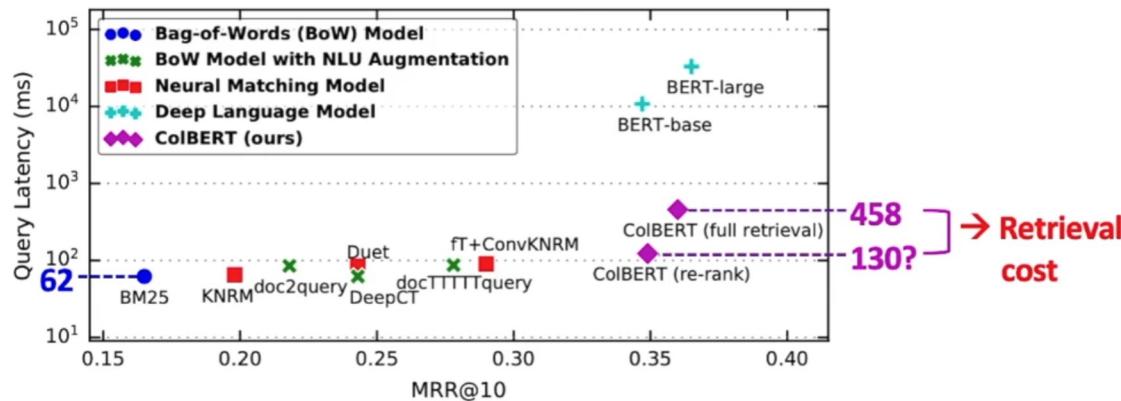
1. 질의 임베딩과 유사한 문서 임베딩을 ANN에서 찾음
2. 문서 임베딩이 나온 모든 문서의 목록을 구함 (union)



# 추가조사

## Q2. e2e에서 retrieval 과정은 어떻게 되는가?

- Retrieval이 ColBERT에서 가장 큰 비용을 차지함
- 따라서 ANN이 ColBERT Retrieval의 핵심 기술 -> 네이버에서는 Annoy, Faiss, Hnswlib 중 Hnswlib을 최적화해 사용함



### 2.4 ANN 선택은?



# 추가조사

Q2. e2e에서 retrieval 과정은 어떻게 되는가?

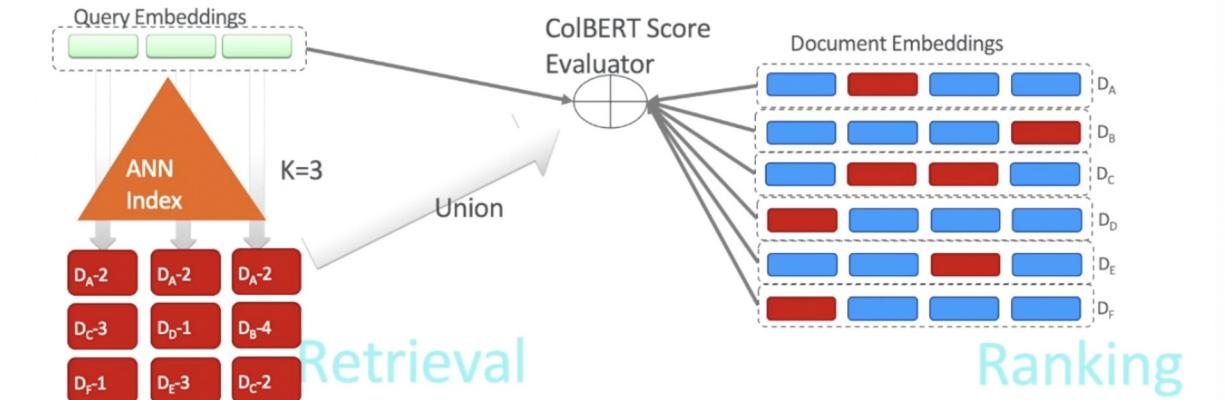
## 5.1 Vector Access in Retrieval / Ranking



ANN으로 접근 가능한 벡터로는 ColBERT score 계산 불가

문서 키로부터 document embeddings를 모두 접근 가능한 구조 필요

메모리 복잡도 큼 : ColBERT score memory access  $\simeq$  candidate count , document vector count , dimension



# 추가조사

Q2. e2e에서 retrieval 과정은 어떻게 되는가?

## 5.2 요구조건과 완화 조건

### 1. ANN Index

(요) 빠르고 정확해야 함

(요) DocID가 반환되어야 함

(완) 벡터 유사도나 벡터값 반환 필요 없음

(완) 결과가 정렬 될 필요 없음

- Key-value vector storage 필요

- 문서 key : 문서 vector



### 2. Vector Storage

(요) 문서 벡터를 빠르게 접근 가능해야 함

(요) Data는 메모리에 있어야 함

(완) 문서의 특정 벡터를 접근할 필요 없음

### 3. ColBERT Score Evaluator

(요) 빠르게 연산이 되어야 함

(요) 요구 조건, (완) 완화 조건