SparseEmbed: Learning Sparse Lexical Representations with Contextual Embeddings for Retrieval

Weize Kong
weize@google.com
Google Research

Jeffrey M. Dudek
jdudek@google.com
Google Research

Cheng Li
chgli@google.com
Google Research

Mingyang Zhang
mingyang@google.com
Google Research

Michael Bendersky
bemike@google.com
Google Research

ABSTRACT

In dense retrieval, prior work has largely improved retrieval effectiveness using multi-vector dense representations, exemplified by ColBERT. In sparse retrieval, more recent work, such as SPLADE, demonstrated that one can also learn sparse lexical representations to achieve comparable effectiveness while enjoying better interpretability. In this work, we combine the strengths of both the sparse and dense representations for first-stage retrieval. Specifically, we propose SparseEmbed – a novel retrieval model that learns sparse lexical representations with contextual embeddings. Compared with SPLADE, our model leverages the contextual embeddings to improve model expressiveness. Compared with ColBERT, our sparse representations are trained end-to-end to optimize both efficiency and effectiveness.

CCS CONCEPTS

• **Information systems** → **Document representation**; **Query representation**; **Retrieval models and ranking**.

KEYWORDS

Sparse Retrieval; Dense Retrieval; Contextual Embeddings

ACM Reference Format:
Weize Kong, Jeffrey M. Dudek, Cheng Li, Mingyang Zhang, and Michael Bendersky. 2023. SparseEmbed: Learning Sparse Lexical Representations with Contextual Embeddings for Retrieval. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '23), July 23–27, 2023, Taipei, Taiwan.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3539618.3592065

1 INTRODUCTION

Retrieval, in contrast to ranking, concerns retrieving a relatively small set of documents from a large corpus. Dense retrieval [25] represents queries and documents using dense vectors (also called embeddings) and retrieval is usually accomplished via approximate nearest neighbor search (ANNS) [1]. Sparse retrieval [2, 6, 26] instead uses sparse representations, mostly based on lexical features (e.g., BM25), and can be served via inverted index.

SIGIR '23, July 23–27, 2023, Taipei, Taiwan

ACM ISBN 978-1-4503-9408-6/23/07.
https://doi.org/10.1145/3539618.3592065

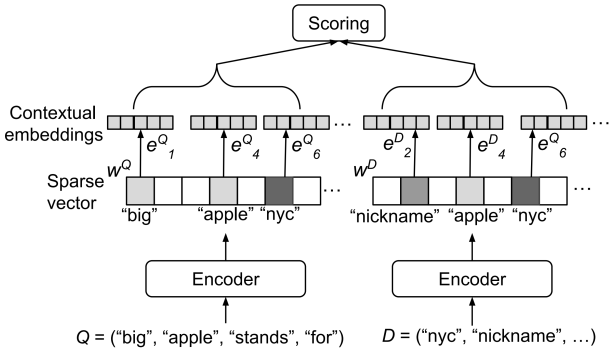

Figure 1: SparseEmbed architecture overview.

There are exciting advancements from neural information retrieval for both dense and sparse retrieval. For dense retrieval, prior work [14, 17] found that single-vector representations could be inadequate to capture all the key information and proposed to use multi-vector representations to improve model expressiveness, as exemplified by ColBERT [13]. However, ColBERT is also expensive to be deployed for large scale retrieval systems due to its large ANNS index size and quadratic-time scoring method (Section 4). For sparse retrieval, SPLADE [5–7] and other recent work [2] demonstrated that one can also learn sparse lexical representations to achieve comparable performance while offer enhanced interpretability and easier deployment.

In this work, we aim to combine the strengths of sparse and dense representations for retrieval, and propose SparseEmbed – a novel retrieval model that learns sparse lexical representations with contextual embeddings for retrieval. As illustrated in Figure 1, SparseEmbed first encodes a given query (or document) into a sparse vector over the lexical vocabulary feature space, following SPLADE. For example, query "big apple stands for" could be encoded as {"big", "apple", "nyc", ...} with weights for each word. Note that the encoder could generate expansion terms (e.g., "nyc") which do not appear in the original input.

To improve model expressiveness, we borrow the idea from ColBERT to use contextual embeddings. Specifically, SparseEmbed further generates a contextual embedding for each term activated in the sparse vector. The contextual embeddings are pooled from underlying transformer sequence encodings to capture contextual information for each term. For example, embeddings for "apple" can then capture semantic difference when the word appears in the context of "big apple" versus "apple stock".

SparseEmbed offers several strengths, inherited from SPLADE, ColBERT. First, SparseEmbed improves model expressiveness with contextual embeddings, compared against SPLADE which is limited to lexical matching. Second, SparseEmbed provides efficiency and cost advantages compared with ColBERT. We apply sparsity loss (Section 2.4) on the sparse vector to control the number of contextual embeddings. This enables optimizing index and querying cost during training. In addition, SparseEmbed is more efficient than ColBERT in scoring contextual embeddings (Section 2.3), since it only needs to compare contextual embeddings for matching query and document terms (linear complexity) instead of all query-document term pairs in ColBERT's late interaction (quadratic complexity). Third, SparseEmbed can be served using inverted index via attaching the contextual embeddings in the posting lists (Section 2.5), as with COIL [9]. Different from COIL, SparseEmbed can encode input text with expansion terms addressing the classic lexical mismatch problem COIL faces.

Our contributions can be summarized as follows:

- We propose SparseEmbed, a sparse-dense hybrid model that learns sparse lexical representations with dense contextual embedding for first stage retrieval.
- We design a lightweight contextual embedding layer (Section 2.2) and use a top-k layer (Section 2.1) for SparseEmbed. These enable combining sparse retrieval as in SPLADE and dense retrieval as in ColBERT effectively and practically.
- We conduct experiments on public datasets which test the effectiveness of SparseEmbed and demonstrate its capability of effectiveness-efficiency trade-off.

2 MODEL

As illustrated in Figure 1, SparseEmbed first encodes a query $Q = (q_1, q_2, ..., q_{|Q|})$ into a sparse vector $w \in \mathbb{R}^{|V|}$ over a vocabulary V. w can be viewed as a term weight vector with only a few terms activated, i.e., having non-zero weights. Second, the model computes a contextual embedding for each activated term. Lastly, we apply scoring on top of the sparse-dense hybrid representations from the query and document side. Note that we encode the document $D = (d_1, d_2, ..., d_{|D|})$ in the same way as the query. We describe each parts in detail below, using the query side as an example.

2.1 Sparse Vector

We follow SPLADE [5, 6] to compute the sparse vector $w \in \mathbb{R}^{|V|}$, as illustrated in Figure 2. We first feed the query Q into a BERT encoder, producing the sequence encodings $S \in \mathbb{R}^{|Q| \times H}$, where H is the hidden size. Then we use BERT's MLM head to compute the MLM logits, $M \in \mathbb{R}^{|Q| \times |V|}$. Lastly, we transform the logits and apply max-pooling to compute the sparse vector w as follows,

$$w_i = \max_{j=1..|Q|} \log\left(1 + \texttt{ReLU}(m_{j,i})\right), \tag{1}$$

where w_i is the i-th value in w and $m_{j,i}$ is the logits in M. See SPLADE [5] for more details.

Different from SPLADE, to facilitate the computation of contextual embeddings in the next step, we apply a top-k layer which select k dimensions with the highest weights in w, and zero-mask the other dimensions. This process helps bound the number of contextual embeddings we need to process.

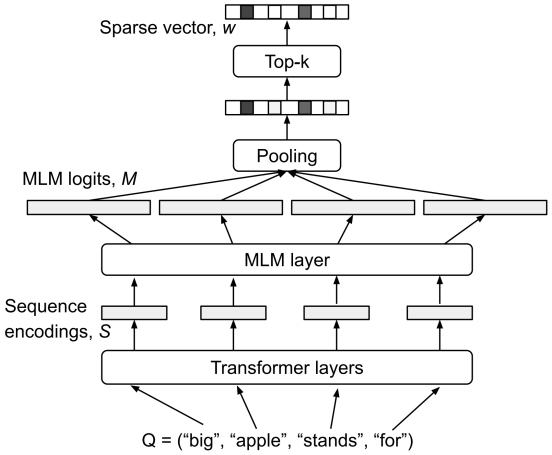

Figure 2: Sparse vector computation.

2.2 Contextual Embedding

We compute a context embedding for each term activated in the sparse vector, i.e., terms with $w_i > 0$. Different from ColBERT [3], we can't directly use the sequence encodings from the BERT encoder as contextual embeddings. This is because some activated terms may not appear in the input, as a result there are no corresponding sequence encodings for them.

To address this, we use an attention layer to pool from the sequence encodings for each activated term, e.g., term #2 and #6 as illustrated in Figure 3. Our attention layer is lightweight. Since the

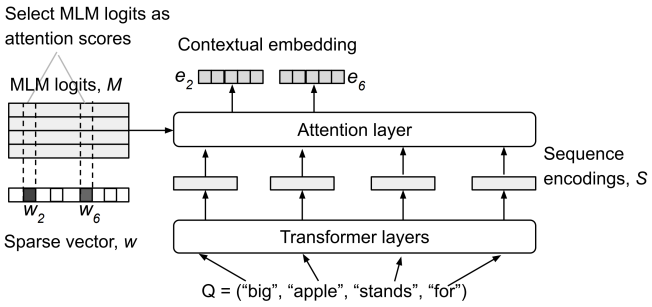

Figure 3: Contextual embedding computation.

MLM logits already measure the association between tokens in the input sequence and terms in the vocabulary, we can directly use the logits to compute the attention scores over the input sequence. Specifically, we compute the contextual embedding for the i-th term in the vocabulary, $e_i \in \mathbb{R}^H$, as,

$$e_i = \texttt{softmax}(m_i^T)S, \tag{2}$$

where $m_i \in \mathbb{R}^{|Q|}$ is the i-th column in MLM logits M and $S \in \mathbb{R}^{|Q| \times H}$ is the sequence encodings. With this attention layer we can represent all the terms including expansion terms.

Lastly, we project the resulting embedding from hidden size H to a smaller size using a linear layer and apply ReLU activation to ensure values are all non-negatives, i.e., $\texttt{ReLU}(\texttt{linear\_layer}(e_i))$. The dimension reduction helps reduce querying and index space

cost (see Section 2.5). The non-negative values in embeddings ensure dot-products computed upon are also non-negative. This enables querying time optimization when aggregating scores, e.g., early stop on candidate documents with low accumulated scores. In the end, we have a set of contextual embeddings $\{e_i|i \in I\}$ for the activated terms in the vocabulary, where I are the indices (to the vocabulary) for the activated terms, $I = \{i|w_i > 0\}$.

2.3 Scoring

After encoding a query and a document, we compute their relevance score using dot-products between query and document contextual embeddings of matching terms, or more formally,

$$s(Q, D) = \sum_{(i,j) \in I^Q \times I^D, i=j} (e_i^Q)^T e_j^D, \tag{3}$$

where superscript Q and D indicate query and document variables, e.g., e_i^Q is a contextual embedding from the query side. I^Q and I^D are the indices to the vocabulary for the activated terms. $i = j$ means e_i^Q and e_j^D are for the same term. This scoring uses linear time complexity, $O\left(\min(\|w^Q\|_0, \|w^D\|_0)\right)$; while late interaction scoring in ColBERT [3] uses quadratic time, $O(|Q||D|)$.

2.4 Losses

We train SparseEmbed with two types of losses – sparsity loss and ranking loss. Sparsity loss is important as it not only encourages sparsity in the sparse vector w, but also in turn controls the number of contextual embeddings. Using sparsity loss, we can reduce both scoring cost (or the pairs of contextual embeddings we need to compare) and index space cost (or the number of document contextual embeddings to store).

We follow SPLADE [6] to use FLOPS loss [19]. It is a smooth relaxation of the average number of floating-point operations necessary to score a document based on its sparse vector, or equivalently the average number of contextual embedding pairs we need to compute dot-product for SparseEmbed scoring (See Section 2.3). L1 regularization loss is also applicable, but was found to produce less balanced sparse vectors than FLOPS loss [19].

Ranking loss aims to improve ranking. In our experiment, we use a distillation dataset for training. Each example is a triplet, containing a query Q, a positive document D^+ and a negative document D^-, with distillation scores from a teacher model. We use `MarginMSE` loss [10] on two heads – the score based on the contextual embeddings e_i in Equation 3 and the score based on sparse vectors defined as $s(Q, D) = (w^Q)^T w^D$. We denote them as $\mathcal{L}^e_{\texttt{MarginMSE}}$ and $\mathcal{L}^w_{\texttt{MarginMSE}}$ respectively. $\mathcal{L}^w_{\texttt{MarginMSE}}$ helps the model to learn to select terms for generating contextual embeddings.

Finally, we combine sparsity and ranking losses together,

$$\mathcal{L} = \mathcal{L}^e_{\texttt{MarginMSE}} + \lambda^w \mathcal{L}^w_{\texttt{MarginMSE}} + \lambda^Q \mathcal{L}^Q_{\texttt{FLOPS}} + \lambda^D \mathcal{L}^D_{\texttt{FLOPS}}, \tag{4}$$

where weight λ^Q, λ^D can be used to control model sparsity for efficiency-effectiveness trade-off, and weight λ^w is fixed to 0.1 in our experiment.

2.5 Inverted Index

Similar to COIL [9], SparseEmbed can be served with inverted index. We index documents based on their activated terms in the same way as other lexical IR systems, except that their associated contextual embeddings are also attached to the posting lists. At querying time, for each activated term of the query, we retrieve documents from its posting list together with the document contextual embeddings. Instead of scoring based on term frequency and inverse document frequency, the relevance score is computed based the contextual embeddings as defined in Equation 3.

3 EXPERIMENTS

3.1 Experimental Setup

Data. We follow SPLADE [7] to use the MS MARCO passage dataset[1] for in-domain retrieval experiments and the BEIR [22] benchmark datasets for zero-shot experiments. The MS MARCO dataset contains 8.8M passages, 500k training queries and 6980 dev queries. We train models on the public `msmarco-hard-negatives` distillation dataset[2]. It contains 50 hard negatives mined from BM25 and 12 dense retrievers for each training queries with distillation scores from a cross-attention teacher model. From this dataset, we sample 25.6M triplets (Q, D^+, D^-) for training.

Implementation. Following SPLADE++ [7], we implement SparseEmbed using `bert-base-uncased` BERT encoder initialized from the CoCondenser [8] pretrained checkpoint. Queries and documents share the same BERT encoder, but use distinct MLM heads and contextual embedding projection layers (Section 2.2). k in the top-k layer (Section 2.1) is set to 64, 256 for queries and documents respectively. We train models with the MSEMargin loss and the FLOPS loss (Section 2.4). We quadratically increase the FLOPS loss weights (Equation 4) at each training step until 50k steps, from which it remains constant, as in SPLADE[6]. We train for 150k steps with batch size 128. We re-implement SPLADE++ [7], an improved hard-negative distillation version of the original model [5, 6], using the same settings as SparseEmbed. For other baselines, including ColBERTv2 [21], we cite metrics from prior work for comparison.

Metrics & Evaluation. We evaluate models on MS MARCO dev queries for in-domain evaluation and on BEIR datasets for zero-shot evaluation. In addition to ranking metrics, we report an efficiency measure called TERMS, which estimates the average number of matched terms in a random query and a random document based on their sparse vectors:

$$\text{TERMS} = \sum_{Q \in \mathcal{Q}} \frac{1}{|\mathcal{Q}|} \|w^Q\|_0 \cdot \sum_{D \in \mathcal{D}} \frac{1}{|\mathcal{D}|} \|w^D\|_0, \tag{5}$$

where $\mathcal{Q}$ is the test query set and $\mathcal{D}$ is the document corpus, w is the sparse vectors. TERMS is closely related to the FLOPS measure [6] that estimates the average number of floating-point operations for scoring one document. For SPLADE model, the prior work [6] estimated its FLOPS exactly the same as TERMS. For SparseEmbed, we can estimate FLOPS = TERMS $\times H'$, where H' is the contextual embedding size after projection and accounts for the floating-point operations involved in an embedding dot-product (Section 2.3).

[1] https://github.com/microsoft/MSMARCO-Passage-Ranking
[2] https://huggingface.co/datasets/sentence-transformers/msmarco-hard-negatives

3.2 Results

In-domain Evaluation. We report the in-domain evaluation results on MS MARCO in Table 1. TERMS and FLOPS are described in Section 3.1. λ^q, λ^d are FLOPS loss weights (Equation 4). SPLADE^o++ is our SPLADE++ re-implementation. Superscript *S* and *L* for SparseEmbed denote different FLOPS loss weights used. Subscript *16*, *32*, *64* denote the projection dimension for contextual embeddings (Section 2.2). Upper section numbers are copied from the cited papers.

Model	MRR@10	R@1k	TERMS	FLOPS	λ^Q, λ^D
BM25 [7]	18.4	85.4	-	-	-
COIL-full [9]	35.5	96.4	-	-	-
ColBERT [13]	36.8	96.9	-	-	-
ColBERTv2 [21]	39.7	98.3	-	-	-
SPLADE [6]	32.2	95.5	-	-	
SPLADE++ [7]	38.0	98.2	-	-	-
SPLADE^o++	37.8	98.2	1.22	1.22	$4e^{-1}, 5e^{-1}$
SparseEmbed$^S_{32}$	38.4	98.2	0.57	0.57×32	$4e^{-2}, 5e^{-2}$
SparseEmbed$^L_{16}$	38.8	98.1	0.74	0.74×16	$4e^{-3}, 5e^{-3}$
SparseEmbed$^L_{32}$	39.0	98.2	4.46	4.46×32	$4e^{-3}, 5e^{-3}$
SparseEmbed$^L_{64}$	39.2	98.1	1.63	1.63×64	$4e^{-3}, 5e^{-3}$

Table 1: Evaluation results for MS MARCO passage retrieval.

First, all runs of SparseEmbed outperform SPLADE models on MRR@10, demonstrating the effectiveness of SparseEmbed. For example, SparseEmbed$^S_{16}$ improves SPLADE^o++ by +2.6% on MRR@10, while using less TERMS (0.74 v.s. 1.22). This indicates the performance gain is not due to SparseEmbed using more activated terms but due to its contextual embeddings. In fact, we find SparseEmbed is always more effective at a similar TERMS measure, when comparing metrics reported in Figure 1 of the SPLADE++ paper [7]. That said, SPLADE++ is still more efficient according to FLOPS.

Second, comparing SparseEmbed$^S_{32}$ and SparseEmbed$^L_{32}$, we show that one can adjust the FLOPS loss weights (Section 2.4) to make trade-off between efficiency and effectiveness. Comparing different SparseEmbed$^L_*$ runs, we find larger contextual embedding projection size is more effective. Third, ColBERTv2 is still more effective than SparseEmbed. However, ColBERT is also more expensive to serve due to its quadratic time-complexity scoring (Section 2.3).

Zero-shot Evaluation. We report zero-shot evaluation results on the same 13 BEIR datasets as the prior work [5, 7] in Table 2.

First, SparseEmbed demonstrates strong out-of-domain generalizability. It achieves the best average NDCG@10, slightly better than SPLADE++. Second, while ColBERTv2 perform strongly in the in-domain setting (Table 1), both SparseEmbed and SPLADE++ perform better in the zero-shot setting than ColBERT on average NDCG@10. This indicates sparse retrieval models may have some inductive bias for out-of-domain generalizability.

4 RELATED WORK

Our work is related to both dense retrieval and sparse retrieval. For dense retrieval, prior work [14, 17] found single-vector dense representations could be inadequate. ColBERT [13] proposed to use multi-vector representations by directly using contextual embeddings from a BERT encoder. This significantly improves model expressiveness, but is also expensive to scale. Follow-up work [12, 15, 21, 23] has explored pruning or compression methods to improve efficiency. We also use contextual embeddings, but we rely on the underlying sparse lexical representation learning to optimize contextual embedding selection for both efficiency and effectiveness. This is achieved by sparsity loss and ranking loss (Section 2.4). Our model is also more efficient in scoring, as it avoids the quadratic contextual embedding interaction in ColBERT (Section 2.3). Another orthogonal research direction, started from dense retrieval, is hard negative mining [20, 24] and distillation [10, 11, 16, 20].

Dataset	BM25	ColBERTv2[21]	SPLADE++[7]	SparseEmbed$^L_{64}$
ArguAna	31.5	46.3	**52.5**	51.2
Climate-FEVER	21.3	17.6	**23.0**	21.8
DBPedia	31.3	44.6	43.6	**45.7**
FEVER	75.3	78.5	79.3	**79.6**
FiQA-2018	23.6	**35.6**	34.8	33.5
HotpotQA	60.3	66.7	68.7	**69.7**
NFCorpus	32.5	33.8	**34.8**	34.1
NQ	32.9	**56.2**	53.7	54.4
Quora	78.9	**85.2**	83.4	84.9
SCIDOCS	15.8	15.4	15.9	**16.0**
SciFact	66.5	69.3	70.2	**70.6**
TREC-COVID	65.6	73.8	**72.7**	72.4
Touché-2020	36.7	26.3	24.5	**27.3**
Average	44.0	49.9	50.5	**50.9**

Table 2: NDCG@10 on BEIR datasets in zero-shot setting. Baseline metric values are copied from the cited papers.

Sparse retrieval such as BM25 has been studied for decades. More recent work [2, 4, 6, 18, 26] started learning sparse lexical representation using neural networks. In this category, SPLADE [5–7] have shown that one could learn a sparse lexical encoder to achieve comparable performance as other dense retrieval models, and offers enhanced interpretability. We build SparseEmbed on top of SPLADE and add contextual embeddings to improve model expressiveness.

To the best of our knowledge, there is limited work in sparse-dense hybrid representations for retrieval. COIL [9] uses the term-level contextual embeddings to perform lexical match. However, COIL does not learn the sparse representation as SparseEmbed (Section 2.1), it simply encodes the text input based on term occurrence, which can lead to lexical mismatch issues.

5 CONCLUSIONS

We present SparseEmbed, a retrieval model that learns sparse lexical representations with contextual embeddings. The model combines the strengths of sparse retrieval like SPLADE and multi-vector dense retrieval like ColBERT. With contextual embeddings, SparseEmbed improves model expressiveness over SPLADE. With the sparse representations and sparsity loss, SparseEmbed provides efficiency advantages over ColBERT in both querying and index space cost. This sparse-dense hybrid representations can also be served via inverted index. Our experiments demonstrate both in-domain and out-of-domain effectiveness, as well as the effectiveness-efficiency trade-off SparseEmbed offers.

REFERENCES

[1] Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. Approximate nearest neighbor search in high dimensions. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3287–3318. World Scientific, 2018.

[2] Yang Bai, Xiaoguang Li, Gang Wang, Chaoliang Zhang, Lifeng Shang, Jun Xu, Zhaowei Wang, Fangshan Wang, and Qun Liu. Sparterm: Learning term-based sparse representation for fast text retrieval. *arXiv preprint arXiv:2010.00768*, 2020.

[3] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proc. of ICML*, pages 160–167, 2008.

[4] Zhuyun Dai and Jamie Callan. Context-aware term weighting for first stage passage retrieval. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 1533–1536, 2020.

[5] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. Splade v2: Sparse lexical and expansion model for information retrieval. *arXiv preprint arXiv:2109.10086*, 2021.

[6] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. Splade: Sparse lexical and expansion model for first stage ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2288–2292, 2021.

[7] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. From distillation to hard negative sampling: Making sparse neural ir models more effective. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2353–2359, 2022.

[8] Luyu Gao and Jamie Callan. Unsupervised corpus aware language model pre-training for dense passage retrieval. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2843–2853, 2022.

[9] Luyu Gao, Zhuyun Dai, and Jamie Callan. Coil: Revisit exact lexical match in information retrieval with contextualized inverted list. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3030–3042, 2021.

[10] Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. Improving efficient neural ranking models with cross-architecture knowledge distillation. *arXiv preprint arXiv:2010.02666*, 2020.

[11] Sebastian Hofstätter, Sheng-Chieh Lin, Jheng-Hong Yang, Jimmy Lin, and Allan Hanbury. Efficiently teaching an effective dense retriever with balanced topic aware sampling. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 113–122, 2021.

[12] Sebastian Hofstätter, Omar Khattab, Sophia Althammer, Mete Sertkan, and Allan Hanbury. Introducing neural bag of whole-words with colberter: Contextualized late interactions using enhanced reduction. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management*, page 737–747, 2022.

[13] Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proc. of SIGIR*, pages 39–48, 2020.

[14] Weize Kong, Swaraj Khadanga, Cheng Li, Shaleen Kumar Gupta, Mingyang Zhang, Wensong Xu, and Michael Bendersky. Multi-aspect dense retrieval. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3178–3186, 2022.

[15] Carlos Lassance, Maroua Maachou, Joohee Park, and Stéphane Clinchant. A study on token pruning for colbert. *arXiv preprint arXiv:2112.06540*, 2021.

[16] Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. In-batch negatives for knowledge distillation with tightly-coupled teachers for dense retrieval. In *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*, pages 163–173, 2021.

[17] Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. Sparse, dense, and attentional representations for text retrieval. *arXiv preprint arXiv:2005.00181*, 2020.

[18] Antonio Mallia, Omar Khattab, Torsten Suel, and Nicola Tonellotto. Learning passage impacts for inverted indexes. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1723–1727, 2021.

[19] Biswajit Paria, Chih-Kuan Yeh, Ian EH Yen, Ning Xu, Pradeep Ravikumar, and Barnabás Póczos. Minimizing flops to learn efficient sparse representations. In *International Conference on Learning Representations*, 2019.

[20] Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2010.08191*, 2020.

[21] Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. Colbertv2: Effective and efficient retrieval via lightweight late interaction. *arXiv preprint arXiv:2112.01488*, 2021.

[22] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[23] Nicola Tonellotto and Craig Macdonald. Query embedding pruning for dense retrieval. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3453–3457, 2021.

[24] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *arXiv preprint arXiv:2007.00808*, 2020.

[25] Andrew Yates, Rodrigo Nogueira, and Jimmy Lin. Pretrained transformers for text ranking: Bert and beyond. In *Proceedings of the 14th ACM International Conference on web search and data mining*, pages 1154–1156, 2021.

[26] Hamed Zamani, Mostafa Dehghani, W Bruce Croft, Erik Learned-Miller, and Jaap Kamps. From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing. In *Proceedings of the 27th ACM international conference on information and knowledge management*, pages 497–506, 2018.