

# Efficient Constant-Space Multi-Vector Retrieval

Sean MacAvaney<sup>1</sup>, Antonio Mallia<sup>2</sup>, and Nicola Tonellotto<sup>3</sup>


<sup>1</sup> University of Glasgow, UK

<sup>2</sup> Pinecone, US

<sup>3</sup> University of Pisa, Italy

**Abstract.** Multi-vector retrieval methods, exemplified by the ColBERT architecture, have shown substantial promise for retrieval by providing strong trade-offs in terms of retrieval latency and effectiveness. However, they come at a high cost in terms of storage since a (potentially compressed) vector needs to be stored for every token in the input collection. To overcome this issue, we propose encoding documents to a fixed number of vectors, which are no longer necessarily tied to the input tokens. Beyond reducing the storage costs, our approach has the advantage that document representations become of a fixed size on disk, allowing for better OS paging management. Through experiments using the MSMARCO passage corpus and BEIR with the ColBERT-v2 architecture, a representative multi-vector ranking model architecture, we find that passages can be effectively encoded into a fixed number of vectors while retaining most of the original effectiveness.

**Keywords:** Multi-Vector Retrieval · Efficiency · Dense Retrieval

 <https://github.com/pisa-engine/ConstBERT>

## 1 Introduction

Pre-trained contextualized language models, such as BERT [7], learn semantic embeddings from word contexts, enabling them to better capture the relevance of documents with respect to the queries. Notably, they outperform classical ranking approaches [16]. More specifically, cross-encoders concatenate a query and a document texts, and feed them into BERT to compute the query document similarity scores, while bi-encoders compute compact representations of documents as real-valued vectors, both for queries and documents, and the query-document similarity is computed using the cosine similarity or the inner product between query and document embeddings.

Cross-encoder can be computationally expensive for estimating query-document similarities due to the complexity of the underlying transformer neural network [10, 11, 13, 27]. On the other side, bi-encoders are much more efficient from a computational perspective, since all document embeddings can be pre-computed offline and store in specialised vector indexes such as FAISS [12]. Instead of relying on a single vector per text as done by bi-encoders, multi-representation systems such as [13] use a vector per token in a text, being able

to capture more semantics than a single embedding. While ColBERT achieves more effective results than single representations, it comes at the cost of higher response times and memory usage [18].

ColBERTv2 [21] employs a compression method that leverages centroids to represent passage embeddings more efficiently. This method records the ID of the nearest centroid for each embedding and compresses the residuals—the differences between the original embeddings and the centroids—using. This compression strategy helps reduce the storage demands of multi-vector embeddings, but makes retrieval significantly less efficient. To speed up the search latency of ColBERTv2, PLAID [22] uses a centroid interaction mechanism and centroid pruning to eliminate low-scoring passages early in the search process, thus reducing response times significantly. This approach allows multi-vector retrieval models to maintain retrieval quality while reducing retrieval latency.

XTR (ConteXtualized Token Retriever) [15] introduces a streamlined approach to multi-vector retrieval by emphasizing efficient token selection during retrieval. Unlike ColBERT’s three-stage process (token retrieval, gathering, scoring), XTR simplifies the retrieval pipeline by training the model to prioritize key document tokens, thus only scoring based on these retrieved tokens. Another recent approach worth mentioning is Static Pruning for Multi-Representation Dense Retrieval [1], which addresses the storage challenges in multi-vector models like ColBERT by pruning embeddings for less impactful tokens. By adapting static pruning techniques traditionally used in sparse indexes to embedding-based indexes, this method reduces storage requirements while maintaining retrieval effectiveness. In addition to pruning techniques, a recent advancement named Token Pooling [4] aims to reduce the storage requirements of multi-vector retrieval models like ColBERT. This approach clusters and pools similar token embeddings at indexing time, substantially lowering vector counts without model modifications or query-time processing.

Multi-vector models like ColBERT are also highly effective as reranking techniques [17], where they refine the ranking of a candidate set generated by simpler retrieval methods, such as BM25. In this setting, ColBERT’s pre-computed, token-level document embeddings allow for efficient late interaction with the query, balancing computational efficiency with strong retrieval performance. This approach leverages the richness of multi-vector representations while maintaining low latency, as document vectors can be cached across queries.

Recently, MUVERA (Multi-Vector Retrieval Algorithm) [8] introduced a mechanism to bridge the gap between single-vector and multi-vector retrieval. MUVERA employs Fixed Dimensional Encodings (FDEs) to approximate multi-vector similarities, enabling the use of optimized Maximum Inner Product Search (MIPS) solvers. This approach significantly enhances retrieval efficiency compared to methods like PLAID. Although MUVERA achieves a good approximation of PLAID using a single-vector representation and MIPS operations, which makes its retrieval algorithm faster, it is able to do so only by employing large high-dimensional vectors. This results in substantial memory consumption, presenting a trade-off between retrieval speed and storage efficiency.

In this paper, we propose a novel approach, ConstBERT, to reduce the storage footprint of multi-vector retrieval by encoding each document with a fixed, smaller set of learned embeddings. Instead of relying on token-level embeddings across all document tokens, we introduce a pooling mechanism that projects these token embeddings into a reduced set of document-level embeddings, each capturing distinct semantic facets. This learned pooling reduces the number of embeddings stored per document, achieving considerable space savings in the index while retaining retrieval effectiveness. The fixed number of vectors per document also eases the use of ConstBERT as a reranking method, simplifying integration with initial retrieval systems and allowing efficient late interaction with pre-computed document representations. This reduction is complementary to other methods, such as dimensionality reduction, and enables efficient memory alignment with OS-level paging, ultimately improving both storage efficiency and query processing speed.

## 2 ConstBERT: Multi-Vector Compression

Given a query  $q$ , our task is to retrieve relevant documents  $d$  from a corpus  $D$  by ranking them with a relevance scoring function  $s(q, d)$ . Queries and documents are sequences of tokens from a given vocabulary. Each document  $d$  comprises  $M$  tokens, and each query  $q$  comprises  $N$  tokens, with padding/masking tokens if necessary. In a multi-representation dense IR system, any token is represented by a  $k$ -dimensional real-valued vector, called embedding. Let  $q_1, \dots, q_N$  denote the token embeddings for the query  $q$ , and  $d_1, \dots, d_M$  denote the token embeddings for the document  $d$ . The relevance score  $s(q, d)$  between the query  $q$  and the document  $d$  is computed with a late interaction mechanism:

$$s(q, d) = \sum_{i=1}^N \max_{j=1, \dots, M} q_i^T d_j.$$

This late interaction mechanism sums up the contributions of the most relevant document token for each query token. For each query token, the max operator can be interpreted as an heuristic pooling mechanism over the token embeddings of the document. Instead of relying on this heuristic pooling across all document tokens, in this paper we propose a new *learned* pooling, where instead of using the document embeddings  $d_1, \dots, d_M$ , i.e., an embedding per document token, we use  $C < M$  new embeddings  $\delta_1, \dots, \delta_C$ , that are learned with an additional projection layer with parameters  $W \in \mathbb{R}^{Mk \times Ck}$ :

$$[\delta_1 | \dots | \delta_C] = W^T [d_1 | \dots | d_M].$$

This layer is learned end-to-end during training takes as input the token embeddings of a document computed by the multi-representation dense IR system, and with a linear transformation projects them in a fixed number of embeddings, of the same dimensions. In doing so, the new embeddings can be seen as different single document-level embeddings, each one encoding some semantic facet of the

document, given its token embeddings. The relevance score  $s(q, d)$  between the query  $q$  and the document  $d$  is computed now as:

$$s(q, d) = \sum_{i=1}^N \max_{j=1, \dots, C} q_i^T \delta_j.$$

As a result, the total number of embeddings per document to store in the embedding index decreases by a factor  $M/C$ . This reduces the space required to store the index on disk and in main memory, as well as the query processing time. This space reduction is orthogonal to any further space reduction obtained, for example, by reducing the number of dimensions  $k$  per embedding. Both reductions can be further exploited to align the space occupancy per document to the memory page size, so to exploit more efficiently the underlying memory management mechanisms provided by the operating system.

### 3 Experimental Results

In this section, we evaluate the performance and efficiency of our proposed fixed-vector model, denoted as `ConstBERTC`, where  $C$  represents the number of fixed embeddings per document. We compare it against the baseline, `ColBERT`, which uses token-level embeddings for each document token.

#### 3.1 Experimental Setup

*Datasets & Queries.* Our experimental framework utilizes the MSMARCO v1 passage corpus [2], which consists of approximately 8.8 million passages. To assess both the effectiveness and efficiency of query processing, we benchmark our approach against established methodologies using the MSMARCO Dev Queries, as well as datasets from the TREC Deep Learning Tracks of 2019 and 2020 [5, 6]. Furthermore, we conduct evaluations on an additional 13 collections drawn from the BEIR benchmark [23], allowing for a comprehensive analysis across datasets.

*Metrics.* To evaluate effectiveness, we employ the official metrics designated for each query set: Mean Reciprocal Rank at cutoff 10 (MRR@10) for MSMARCO Dev queries and Normalized Discounted Cumulative Gain at cutoff 10 (NDCG@10) for both TREC queries and the BEIR benchmark. Additionally, we report recall across varying cutoff thresholds for the MSMARCO experiments. For efficiency analysis, we compute the Mean Response Time (MRT) for both the Dev and TREC queries, measured in milliseconds. Additionally, we examine the index sizes to demonstrate the substantial storage efficiency gains achieved by our method.

*Implementations.* `ConstBERT` has been trained following the approach proposed by Santhanam et al. [21]. `ColBERTSP` refers to the modification of `ColBERT` proposed by Acquavia et al. [1], where token embeddings are statically pruned

at indexing time. RetroMAE [26] is a single-representation dense retrieval model for which we have used the official checkpoint<sup>4</sup>.

*Platform.* All experiments were carried out in memory on a Linux system, using a single processing thread. The hardware configuration included dual 2.8 GHz Intel Xeon CPUs and 256 GiB of RAM. For end-to-end retrieval experiments, we used the official PLAID [17, 22] codebase<sup>5</sup>. In our two-stage retrieval experiments, we used BMP [20] and efficient SPLADE [14] for candidate generation and then we performed reranking using our ConstBERT<sub>32</sub> model. We tested other learned sparse models as first-stage retrieval methods, such as DeeperImpact [3, 19], and obtained similar results; however, we did not include them in the main results section due to limited space. Our code is written in Python and is available at <https://github.com/pisa-engine/ConstBERT>.

### 3.2 Overall

Table 1 presents the results on MSMARCO with different configurations of ConstBERT<sub>C</sub> (varying the number of fixed embeddings per document  $C$ ) and the baseline ColBERT. As expected, ConstBERT’s performance improves with larger token-level configurations, but at the cost of substantial increases in index size. Our proposed ConstBERT<sub>32</sub> model achieves comparable MRR on the development set and NDCG@10 on both TREC 2019 and TREC 2020 benchmarks, while using a fixed number of vectors per document, which allows for more efficient storage. ConstBERT<sub>C</sub> has a variety of tradeoffs compared to the existing static pruning approach ColBERT<sub>SP</sub>. The performance of ColBERT<sub>SP</sub> lines on the storage-effectiveness Pareto frontier set by various settings of  $C$ . On the one hand, ConstBERT<sub>C</sub> offers advantages in flexibility since it can be tuned directly to a target (and constant-space) representation. On the other hand, it requires re-training learn the weights  $W$ , while ColBERT<sub>SP</sub> does not require retraining.

To evaluate the robustness of ConstBERT<sub>C</sub> across different retrieval tasks, we further assess it on the BEIR benchmark (Table 2). The results highlight that our model performs competitively with ColBERT on most tasks, achieving comparable or even superior NDCG@10 scores while requiring much less storage.

A major advantage of our fixed-vector approach lies in its reduced storage footprint. Unlike ColBERT, which scales linearly with the number of token embeddings per document, ConstBERT<sub>C</sub> maintains a consistent index size by using a fixed number of embeddings. This efficiency extends across the BEIR datasets, with our approach consistently reducing index sizes by over 50% compared to ColBERT at equivalent effectiveness.

The reduced storage requirement of ConstBERT<sub>C</sub> also translates into lower memory usage and faster retrieval times, as fewer embeddings need to be processed per query-document pair. This efficiency gain is particularly advantageous when using ConstBERT<sub>C</sub> as a reranking method, where computational speed is crucial.

<sup>4</sup> [https://huggingface.co/Shitao/RetroMAE\\_MSMARCO\\_distill](https://huggingface.co/Shitao/RetroMAE_MSMARCO_distill)

<sup>5</sup> <https://github.com/stanford-futuredata/ColBERT>

**Table 1.** Effectiveness metrics and index space consumption on different query sets for the MSMARCO benchmark.

	Dev		TREC 2019				TREC 2020						
	Index Size	MRR	Recall			NDCG@10	Recall			NDCG@10	Recall		
			50	200	1000		50	200	1000		50	200	1000
ColBERT	22G	39.99	86.52	94.47	97.34	74.64	45.64	68.88	83.11	73.99	53.80	72.64	82.70
ColBERT <sub>SP</sub>	14G	39.12	85.81	93.80	97.00	74.42	45.41	66.58	79.74	72.36	52.94	71.74	82.04
ConstBERT <sub>16</sub>	5G	37.84	84.04	91.74	94.11	71.15	41.38	61.31	72.62	73.75	48.98	65.80	73.89
ConstBERT <sub>32</sub>	11G	39.04	85.86	93.72	96.34	73.14	44.93	65.46	78.37	73.29	51.57	69.74	79.14
ConstBERT <sub>64</sub>	20G	39.15	86.27	94.06	96.90	74.29	46.07	66.97	79.64	73.47	52.85	71.98	81.62
ConstBERT <sub>128</sub>	40G	39.53	86.46	94.39	97.29	74.37	46.79	68.05	81.28	73.31	52.04	71.80	82.36

**Table 2.** NDCG@10 and index space consumption on 13 datasets of the BEIR benchmark.

	arguana	cfever	dbpedia	fever	fiqa	hotpot	nf	nq	quora	scidocs	scifact	covid	touche
<b>nDCG@10</b>													
ColBERT	0.452	<b>0.163</b>	<b>0.434</b>	<b>0.751</b>	<b>0.338</b>	<b>0.679</b>	<b>0.329</b>	<b>0.554</b>	<b>0.846</b>	0.154	<b>0.638</b>	0.705	<b>0.261</b>
(RetroMAE)	0.366	<b>0.197</b>	<b>0.469</b>	0.719	<b>0.339</b>	0.602	0.311	0.526	<b>0.864</b>	0.139	<b>0.647</b>	0.649	<b>0.326</b>
ConstBERT <sub>32</sub>	0.451	0.142	0.418	0.696	0.312	0.621	0.327	0.534	0.821	0.156	0.607	<b>0.745</b>	0.260
ConstBERT <sub>64</sub>	<b>0.465</b>	0.139	0.420	0.695	0.335	0.639	0.323	0.537	0.818	<b>0.159</b>	0.631	0.719	0.250
<b>Index Size</b>													
ColBERT	49M	17G	11G	17G	232M	12G	23M	8.3G	<b>0.3G</b>	143M	32M	770M	1.5G
(RetroMAE)	26M	17G	15G	17G	170M	15G	11M	7.7G	1.5G	77M	16M	505M	2.3G
ConstBERT <sub>32</sub>	<b>13M</b>	<b>6G</b>	<b>5G</b>	<b>6G</b>	<b>72M</b>	<b>6G</b>	<b>5M</b>	<b>3.2G</b>	0.6G	<b>33M</b>	<b>7M</b>	<b>212M</b>	<b>0.5G</b>
ConstBERT <sub>64</sub>	22M	13G	11G	13G	138M	12G	10M	6.1G	1.2G	64M	15M	408M	0.9G

### 3.3 Reranking

In Table 3, we evaluate the performance of using ConstBERT as a reranking model instead of employing it in an end-to-end retrieval system. Specifically, we compare PLAID with a two-stage retrieval process incorporating ESPLADE as the model used for candidate generation and our ConstBERT<sub>32</sub>. The results show the retrieval effectiveness in terms of MRR and nDCG@10, and the average computational efficiency represented by MRT. Combining ESPLADE with our lightweight ColBERT<sub>32</sub> version balances both performance and efficiency. This combination improves nDCG@10 close to that of the standalone ColBERT while maintaining MRT below 6 ms, showcasing a practical trade-off. ConstBERT<sub>32</sub> is particularly advantageous not only due to its smaller index size but also because all documents are embedded with the same number of vectors. This uniformity simplifies implementation and allows for optimized memory usage through aligned memory reads, thereby leveraging the underlying memory management mechanisms provided by the operating system more efficiently.

**Table 3.** Effectiveness metrics and mean response time (MRT, in ms) for top-10 retrieval using PLAID vs. two-stage on Dev Queries, TREC 2019, and TREC 2020.

	Dev		TREC 2019		TREC 2020	
	MRR	MRT	nDCG@10	MRT	nDCG@10	MRT
ColBERT	39.99	51.25	74.26	51.46	73.99	50.21
ESPLADE	38.75	3.07	71.33	3.13	71.14	3.20
+ ConstBERT <sub>32</sub>	39.52	4.95	74.38	5.50	74.33	5.23

## 4 Conclusion

Our experimental results demonstrate that the fixed-vector approach, ConstBERT, effectively balances retrieval effectiveness and storage efficiency. By encoding each document with a fixed, smaller set of learned embeddings, our proposal achieves competitive performance across TREC and BEIR benchmarks, while substantially reducing index size and computational demands. This makes it a scalable and practical solution for real-world information retrieval applications, where both storage efficiency and retrieval speed are essential.

There are various opportunities for future work in this space. For instance, several studies related to the interpretability of late interaction models have been conducted, given their alignment between tokens and their corresponding representations (e.g., [9, 18, 25]). With our approach, this direct vector-token alignment is no longer present. However, there still may be ways to interpret the interactions, so it may be worth revisiting these studies. Another interesting direction is the application of Pseudo-Relevance Feedback (PRF) with late interaction models (e.g., [24]). Future studies could explore whether our approach is complementary to PRF.

**Acknowledgments.** This work was partially supported by the Spoke “FutureHPC & BigData” of the ICSC – Centro Nazionale di Ricerca in High-Performance Computing, Big Data and Quantum Computing funded by the Italian Government, the FoRe-Lab and CrossLab projects (Departments of Excellence), the NEREO PRIN project (Research Grant no. 2022AEFHAZ) funded by the Italian Ministry of Education and Research (MUR), and the FUN project (SGA 2024FSTPC2PN30) funded by the Open-WebSearch.eu project (GA 101070014).

## Bibliography

- [1] Antonio Acquavia, Craig Macdonald, and Nicola Tonellotto. Static pruning for multi-representation dense retrieval. In *Proceedings of the ACM Symposium on Document Engineering 2023*, pages 1–10, 2023.
- [2] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. Ms marco: A human generated machine reading comprehension dataset. In *Proc. InCoCo@NIPS*, 2016.
- [3] Soyuj Basnet, Jerry Gou, Antonio Mallia, and Torsten Suel. Deeper-impact: Optimizing sparse learned index structures. *arXiv preprint arXiv:2405.17093*, 2024.
- [4] Benjamin Clavié, Antoine Chaffin, and Griffin Adams. Reducing the footprint of multi-vector retrieval with minimal performance impact via token pooling. *arXiv preprint arXiv:2409.14683*, 2024.
- [5] Nick Craswell, Bhaskar Mitra, Daniel Campos, and Emine Yilmaz. Overview of the TREC 2019 Deep Learning Track. In *Proc. TREC 2019*, 2020.
- [6] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. Overview of the trec 2020 deep learning track. In *Proc. TREC 2020*, 2021.
- [7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proc. NAACL*, 2019.
- [8] Laxman Dhulipala, Majid Hadian, Rajesh Jayaram, Jason Lee, and Vahab Mirrokni. Muvera: Multi-vector retrieval via fixed dimensional encodings. *arXiv preprint arXiv:2405.19504*, 2024.
- [9] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. A white box analysis of colbert. In *Proc. ECIR*, pages 257–263, 2021.
- [10] Sebastian Hofstätter and Allan Hanbury. Let’s measure run time! Extending the IR replicability infrastructure to include performance aspects. In *OSIRRC@SIGIR*, 2019.
- [11] Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. Improving efficient neural ranking models with cross-architecture knowledge distillation, 2020.
- [12] J. Johnson, M. Douze, and H. Jegou. Billion-scale similarity search with gpus. *IEEE Trans. Big Data*, 7(03):535–547, 2021. ISSN 2332-7790.
- [13] Omar Khattab and Matei Zaharia. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proc. SIGIR*, page 39–48, 2020.
- [14] Carlos Lassance and Stéphane Clinchant. An efficiency study for splade models. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2220–2226, 2022.



- [15] Jinhyuk Lee, Zhuyun Dai, Sai Meher Karthik Duddu, Tao Lei, Iftexhar Naim, Ming-Wei Chang, and Vincent Zhao. Rethinking the role of token retrieval in multi-vector retrieval. *Advances in Neural Information Processing Systems*, 36, 2024.
- [16] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. *Pretrained Transformers for Text Ranking: BERT and Beyond*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2021.
- [17] Sean MacAvaney and Nicola Tonellotto. A reproducibility study of plaid. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1411–1419, 2024.
- [18] Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. On single and multiple representations in dense passage retrieval. In *Proc. IIR*, 2021.
- [19] 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.
- [20] Antonio Mallia, Torsten Suel, and Nicola Tonellotto. Faster learned sparse retrieval with block-max pruning. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2411–2415, 2024.
- [21] Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. Colbertv2: Effective and efficient retrieval via lightweight late interaction. *CoRR*, abs/2112.01488, 2021.
- [22] Keshav Santhanam, Omar Khattab, Christopher Potts, and Matei Zaharia. Plaid: an efficient engine for late interaction retrieval. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 1747–1756, 2022.
- [23] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Proc. NeurIPS*, 2021.
- [24] Xiao Wang, Craig Macdonald, and Iadh Ounis. Improving zero-shot retrieval using dense external expansion. *Information Processing & Management*, 59(5):103026, 2022. ISSN 0306-4573. <https://doi.org/https://doi.org/10.1016/j.ipm.2022.103026>. URL <https://www.sciencedirect.com/science/article/pii/S0306457322001364>.
- [25] Xiao Wang, Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. Reproducibility, replicability, and insights into dense multi-representation retrieval models: from colbert to col. In Hsin-Hsi Chen, Wei-Jou (Edward) Duh, Hen-Hsen Huang, Makoto P. Kato, Josiane Mothe, and Barbara Poblete, editors, *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2023, Taipei, Taiwan, July 23-27, 2023*, pages 2552–2561. ACM, 2023. <https://doi.org/10.1145/3539618.3591916>. URL <https://doi.org/10.1145/3539618.3591916>.

- [26] Shitao Xiao, Zheng Liu, Yingxia Shao, and Zhao Cao. Retromae: Pre-training retrieval-oriented language models via masked auto-encoder. *arXiv preprint arXiv:2205.12035*, 2022.
- [27] 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 *Proc. CIKM*, pages 497–506, 2018.