Joint Optimization of Multi-vector Representation with Product Quantization

Yan Fang[1], Jingtao Zhan[1], Yiqun Liu[1⋆], Jiaxin Mao[2], Min Zhang[1], and Shaoping Ma[1]

[1] Department of Computer Science and Technology, Institute for Artificial Intelligence, Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing 100084, China
[2] Beijing Key Laboratory of Big Data Management and Analysis Methods, Gaoling School of Artificial Intelligence, Renmin University of China, Beijing 100872, China
`fangy21@mails.tsinghua.edu.cn,jingtaozhan@gmail.com,yiqunliu@tsinghua.edu.cn`
`maojiaxin@gmail.com,z-m@tsinghua.edu.cn,msp@tsinghua.edu.cn`

Abstract. Dense retrieval models represent queries and documents with one or multiple fixed-width vectors and retrieve relevant documents via nearest neighbor search. Recently these models have shown improvement in retrieval performance and have drawn increasing attention from the IR community. Among a variety of dense retrieval models, the models that employ multiple vectors to represent texts achieve the state-of-the-art ranking performance. However, the multi-vector representation schema imposes tremendous storage overhead compared with single-vector representation, which may hinder its application in practical scenarios. We therefore intend to apply vector compression methods such as Product Quantization (PQ) to reduce the storage cost and improve retrieval efficiency. However, the gap between the original embeddings and the quantized vectors may degenerate retrieval performance. Recently, improved dense retrieval models such as JPQ have been proposed to reduce storage space while maintaining ranking effectiveness by jointly training the encoder and PQ index. They have achieved promising improvement in the single-vector dense retrieval scenario. We therefore try to introduce this joint optimization framework to tackle the storage overhead of the multi-vector models. The key idea is to Jointly optimize Multi-vector representations with Product Quantization (JMPQ). JMPQ prevents effectiveness degeneration by leveraging a joint optimization framework for the query encoding and index compressing processes. We evaluate the performance of JMPQ on publicly available ad-hoc retrieval benchmarks. Extensive experimental results show that JMPQ substantially reduces the memory footprint while achieving ranking effectiveness on par with or even better than its uncompressed counterpart.

Keywords: Dense retrieval, Index compression, Neural ranking

⋆ corresponding author

1 Introduction

Dense Retrieval[7, 13] has become increasingly popular in recent years and has achieved the state-of-the-art ranking effectiveness. It effectively leverages the pre-trained language models[4, 15, 20] to abstract the text and uses nearest neighbor search for retrieval. Experimental results show that dense retrieval substantially outperforms the traditional lexical retrieval methods like BM25[18].

Dense retrieval models can be classified into two categories, namely single-vector representation models[22, 25, 14] and multi-vector representation models[12, 5, 16]. The single-vector models encode text to one dense vector, while the multi-vector models utilize multiple vectors to represent the text. The multi-vector models facilitate more fine-grained interactions compared with the single-vector models and thus lead to better ranking effectiveness. One of the most popular multi-vector models is ColBERT[12], which utilizes token-level multi-vector representations for queries and documents and establishes the state-of-the-art ranking effectiveness. However, multi-vector models require multiple vectors for each document, leading to a huge embedding index, which is usually tens of times larger than that of the single-vector models. The large embedding index questions its application ability in practical use.

To compress the embedding index of multi-vector models, we propose JMPQ, which stands for **J**ointly optimizing **M**ulti-vector representations with **P**roduct **Q**uanti-zation. JMPQ is inspired by JPQ[23], which is used to compress the embedding index of single-vector dense retrieval models. However, the large number of vectors in the multi-vector scenario brings more pressure on search efficiency, and multi-vector models require a more detailed aggregation method compared to single-vector models. Following JPQ, JMPQ utilizes PQ to compress the embedding index of multi-vector models to reduce storage cost. It further employs Inverted File System (IVF) to accelerate the search and introduces vector reconstruction for aggregation. JMPQ also leverages a joint optimization method to prevent effectiveness degeneration caused by the PQ compression. During training, JMPQ end-to-end retrieves top-ranked documents and computes ranking loss based on the retrieval results. It then back-propagates the gradients to the query encoder and PQ index. With PQ and the joint optimization strategy, JMPQ can improve ranking effectiveness in an storage-efficient way.

To verify the effectiveness and efficiency of JMPQ, we base on ColBERT to conduct extensive experiments on publicly available ad-hoc retrieval benchmarks and compare JMPQ against a wide range of existing dense retrieval models and compression methods. Experimental results show that: 1) JMPQ significantly compresses the embedding index of multi-vector models (e.g. ColBERT) by over 10 times and still achieves comparable ranking effectiveness. 2) JMPQ substantially outperforms other compression methods, including unsupervised methods and supervised methods. 3) JMPQ substantially outperforms the competitive single-vector dense retrieval baselines.

2 Related Works

In this section, we recap related work in dense retrieval and index compression.

2.1 Dense Retrieval

Dense retrieval models encode the query and the document into dense vectors and use nearest neighbor search to retrieve documents. Based on the number of representations per text, dense retrieval models can be classified as single-vector models, such as ANCE[22] and ADORE[25], and multi-vector models, such as COIL[5], MEBERT[16], and ColBERT[12, 19]. The single-vector models encode text to one dense vector and thus may result in a limited capacity to abstract sufficient semantic information[16]. On the contrary, the multi-vector models encode text to multiple dense vectors and are capable of modeling token-level interactions. One of the most famous multi-vector retrieval models is ColBERT[12], which represents text with token-level embeddings. Given a query $q = q_0q_1...q_l$ and a document $d = d_0d_1...d_n$, ColBERT computes the token-level term embeddings $\boldsymbol{q}$ and $\boldsymbol{d}$:

$$\boldsymbol{q} = \{\boldsymbol{q_0}, \boldsymbol{q_1}, ..., \boldsymbol{q_l}\} = \text{Encoder}(\mathrm{q_0q_1...q_l}) \tag{1}$$

$$\boldsymbol{d} = \{\boldsymbol{d_0}, \boldsymbol{d_1}, ..., \boldsymbol{d_n}\} = \text{Encoder}(\mathrm{d_0d_1...d_n}) \tag{2}$$

During retrieval, ColBERT employs the MaxSim function to aggregate token-level relevance scores as the document relevance scores:

$$s(q, d) := \sum_{i \in [|\boldsymbol{q}|]} \max_{j \in [|\boldsymbol{d}|]} \boldsymbol{q_i} \cdot \boldsymbol{d_j}^T \tag{3}$$

Although multi-vector models perform better on ranking effectiveness, they also increase the storage overhead by a large margin.

2.2 Index Compression

Vector compression methods have been widely applied. According to the training process, they can be categorized as unsupervised and supervised methods.

Unsupervised Methods Popular unsupervised compression methods include Product Quantization (PQ)[6, 10] and Locality Sensitive Hashing (LSH)[9]. There are several variants of PQ, such as OPQ[6] and RQ[1]. OPQ adds a linear transformation before quantization. RQ utilizes residual for compression. Most unsupervised methods optimize the task-independent reconstruction error and thus cannot benefit from the supervised signals.

Supervised Methods Several studies have explored supervised methods for compression. MoPQ[21] proposes a novel objective MCL and a sample augmentation strategy DCS, which together can effectively contribute to the optimal retrieval accuracy. JPQ[23] jointly trains the encoder and PQ index with hard negative sampling, which is specifically designed for dense retrieval and directly optimizes the ranking effectiveness. RepCONC[24] models quantization as a constrained clustering process and further supports optimization of the dual-encoders and the quantization index in an end-to-end manner.

JMPQ is inspired by JPQ, but has the following two main distinctions from JPQ. Firstly, JMPQ introduces IVF to improve retrieval efficiency in the case of huge number of vectors in the multi-vector scenario. Note that RepCONC also utilizes IVF in the inference stage for efficiency, while JMPQ involves it in both training and inference stages. Secondly, both JPQ and RepCONC directly rely on the PQ index to calculate the vector inner-product as the relevance score. However, multi-vector models require a more detailed aggregation of the inner product. Thus, JMPQ has an additional vector reconstruction process for aggregation and re-ranking.

3 JMPQ Model

We propose JMPQ, Jointly optimize Multi-vector representations with Product Quantization. In this section, we will outline the overall architecture of JMPQ, describe the training strategy and analyze its efficiency.

3.1 Overall Architecture

Figure 1 illustrates the overall architecture of JMPQ. Following the procedure of ColBERT, JMPQ pre-computes token-level document embeddings and builds the IVFPQ index. It performs a two-stage retrieval. At the first stage, top-K document term embeddings are retrieved by the compressed index. At the second stage, JMPQ reconstructs the candidate document embeddings and then uses the scoring function Eq. (3) for re-ranking.

3.2 The IVFPQ Index

The IVFPQ index supports compressed index storage and efficient retrieval. It consists of the Inverted File System (IVF) and the Product Quantization (PQ).

Inverted File System JMPQ employs IVF to accelerate the search. IVF first uses K-means[8] to generate P clusters and assigns each document embedding to its nearest cluster. For a given query embedding, only the nearest n clusters are searched. IVF stores the center embeddings of each cluster:

$$\boldsymbol{\mathcal{C}}_i \in R^D (1 \leq i \leq P) \tag{4}$$

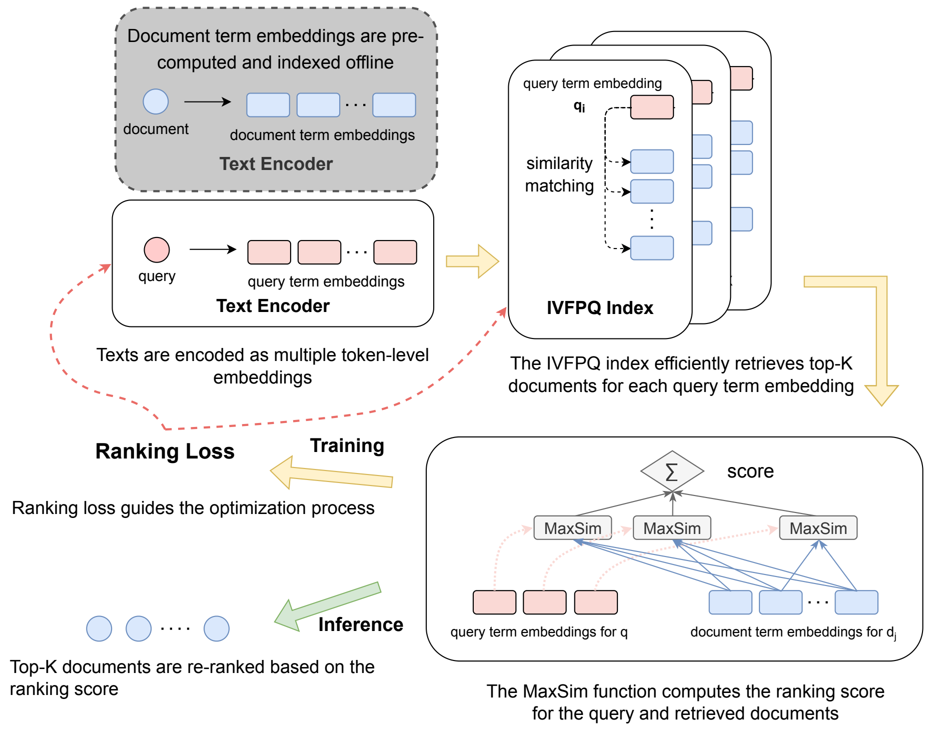
Document term embeddings are pre-computed and indexed offline
document
document term embeddings
Text Encoder
query
query term embeddings
Text Encoder
Texts are encoded as multiple token-level embeddings
query term embedding
q_i
similarity matching
IVFPQ Index
The IVFPQ index efficiently retrieves top-K documents for each query term embedding
Ranking Loss
Training
Ranking loss guides the optimization process
score
MaxSim
MaxSim
MaxSim
query term embeddings for q
document term embeddings for d_j
Inference
Top-K documents are re-ranked based on the ranking score
The MaxSim function computes the ranking score for the query and retrieved documents

Fig. 1: Overall Architecture of JMPQ. Token-level document term embeddings are pre-computed offline for indexing. Given a query, JMPQ encodes it as multiple query term embeddings. The IVFPQ index performs efficient retrieval to get top-K documents for each query term embedding. The irrelevant documents are treated as hard negatives during training. At inference time, JMPQ uses the MaxSim function to re-rank the retrieved documents.

JMPQ then uses PQ to quantize the residual of the document term embedding to its cluster center. Let $\rho(\boldsymbol{d_j})$ denote the cluster $\boldsymbol{d_j}$ is assigned to, the residual embedding equals to:

$$\boldsymbol{r}_{\boldsymbol{d_j}} = \boldsymbol{d_j} - \mathcal{C}_{\rho(\boldsymbol{d_j})} \tag{5}$$

Product Quantization PQ defines M sets of embeddings, each including K embeddings of dimension D/M, where D denotes the embedding dimension:

$$\boldsymbol{c}_{i,j} \in R^{\frac{D}{M}} \quad (1 \leq i \leq M, 1 \leq j \leq K) \tag{6}$$

For a given residual embedding $\boldsymbol{r}_{\boldsymbol{d_j}}$, PQ picks one centroid embedding from each set and concatenates them as $\boldsymbol{r}^{\dagger}_{d_j}$:

$$\boldsymbol{r}_{\boldsymbol{d_j}} \rightarrow \boldsymbol{r}^{\dagger}_{\boldsymbol{d_j}} = \boldsymbol{c}_{1,\phi_1(\boldsymbol{d_j})}, \boldsymbol{c}_{2,\phi_2(\boldsymbol{d_j})} ..., \boldsymbol{c}_{M,\phi_M(\boldsymbol{d_j})} \in R^D \tag{7}$$

where $\phi_i(\boldsymbol{d_j})$ denotes the picked centroid embedding for the ith set.

IVFPQ Index Size We now analyze the storage compression ratio of the IVFPQ index. IVF stores the Cluster Center Embeddings $\{\boldsymbol{\mathcal{C}}_i\}$ and Cluster Assignments $\{\rho_i(\boldsymbol{d_j})\}$, which cost 8 bytes for a single vector. PQ stores the PQ Centroid Embeddings $\{\boldsymbol{c}_{i,j}\}$ and Index Assignments $\{\phi_i(\boldsymbol{d_j})\}$. As K is usually less than 256, $\phi_i(\boldsymbol{d_j})$ can be stored in one byte. A D dimension vector takes $M+8$ bytes in total. As ColBERT uses 2-byte float to store its embeddings, the compression ratio is $2D/(M+8)$.

3.3 Joint Optimization

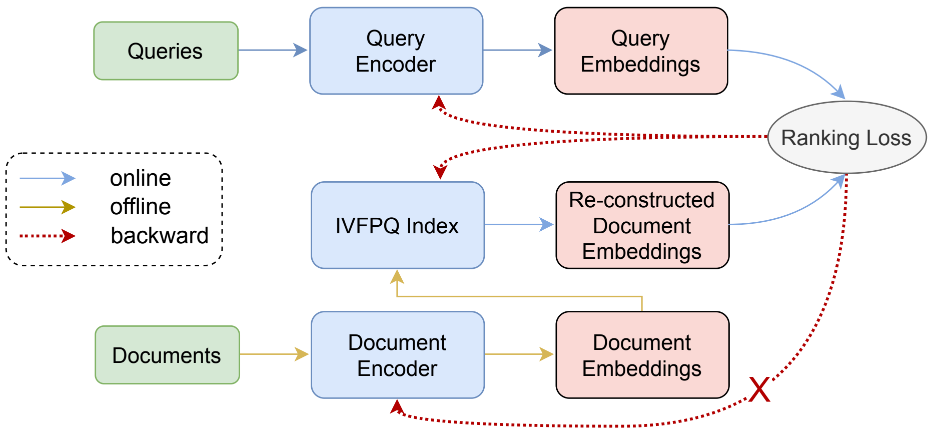

Fig. 2: The training workflow of JMPQ

Figure 2 illustrates the training workflow of JMPQ. Firstly, JMPQ generates the document term embeddings using the document encoder initialized with the well-trained ColBERT and unsupervisely builds the IVFPQ index in the offline stage. Secondly, JMPQ reconstructs the quantized term embeddings from the IVFPQ index by:

$$\boldsymbol{d}_j^{\dagger} = \boldsymbol{c}_{1,\phi_1(\boldsymbol{d}_j)}, \boldsymbol{c}_{2,\phi_2(\boldsymbol{d}_j)} \dots, \boldsymbol{c}_{M,\phi_M(\boldsymbol{d}_j)} + \boldsymbol{C}_{\rho(\boldsymbol{d}_j)} \tag{8}$$

Thirdly, the query term embeddings generated by the query encoder are used to compute the relevance scores with the reconstructed document term embeddings:

$$s(\boldsymbol{q}, \boldsymbol{d}^{\dagger}) := \sum_{i \in [|\boldsymbol{q}|]} \max_{j \in [|\boldsymbol{d}^{\dagger}|]} \boldsymbol{q_i} \cdot \boldsymbol{d_j}^{\dagger T} \tag{9}$$

Finally, the relevance scores are used to compute the ranking loss and then update the query encoder and PQ centroid embeddings with gradient descent:

$$\mathrm{loss} = \mathcal{L}(s(\boldsymbol{q}, \boldsymbol{d}^{\dagger +}), s(\boldsymbol{q}, \boldsymbol{d}^{\dagger -})) \tag{10}$$

Since the loss is computed based on the end-to-end retrieval results, the parameters are updated to directly improve the ranking effectiveness. Note that document encoder is fixed during the training procedure for the following two reasons. Firstly, JMPQ uses the reconstructed embeddings to compute the relevance scores and the computation cost of the document encoder can be saved. Secondly, such approach eliminates the need to rebuild the IVFPQ index after each parameter update, thus enables JMPQ to utilize dynamic hard negatives[26] for better ranking effectiveness.

4 Experiment Setup

In this section, we present our experimental settings, including datasets, baselines, and implementation details.

4.1 Dataset and Metrics

We conduct experiments with popular ad-hoc retrieval benchmarks from the TREC 2019 Deep Learning Track and the TREC 2020 Deep Learning Track[17, 3, 2]. MS MARCO Passage Retrieval has a corpus of 8.8M passages, 0.5M training queries, 7k development queries (**MARCO Passage**), 43 test queries from the TREC 2019 (**DL2019**), and 43 test queries from the TREC 2020 (**DL2020**). We report MRR@10, Recall@100 for MARCO Passage, and nDCG@10, Recall@100 for both the TREC test sets. All the metrics are based on the full-corpus retrieval results.

4.2 Baselines

We consider both the uncompressed and compressed retrieval models as our baselines. For uncompressed retrieval models, we compare our JMPQ with traditional Bag-of-Words models such as BM25[18] and single-vector neural models such as ANCE[22], ADORE[25], and TCT-ColBERT[14]. We also include ColBERT[12] as the multi-vector retrieval model baseline. For the compressed retrieval models, we select JPQ[23] and RepCONC[24] as single-vector supervised baselines. They share the same model architecture and initialization. We include unsupervised PQ[10] and PQ+RQ[19] as compression baselines for ColBERT. Notice that we also report ColBERTv2[19] (which is further trained with hard negatives based on v1) in our results.

4.3 Implementation Details

We build our models based on ColBERTv1[12] and Faiss ANNS Library[11]. The embedding dimension is 128. When implementing JMPQ, K is set to 256, and M is set to 16. We train the unsupervised IVFPQ index as initialization with 30% of the corpus embeddings. As for the training settings of JMPQ, we use AdamW optimizer, batch, size of 32, and cross-entropy loss. For the query encoder, the learning rate is set to 5e-6, and for PQ parameters, the learning rate is set to 1e-5. The number of clusters P is set to 32,768, and the multi-probing parameter n is set to 32. At the first stage, the top 1,024 matches are retrieved for each query term embedding. Considering the computation cost during training, we do not re-rank the retrieved documents to get the top-irrelevant documents, but we randomly sample 255 irrelevant documents from the candidates as hard negatives. At inference time, we re-rank the candidates to get the top 1,000 documents.

5 Experiments

Now we empirically evaluate the proposed JMPQ and compare it with different types of baselines. We summarize the ranking effectiveness, index size and query latency in Table 1. Note that the multi-vector models are initialized with ColBERTv1[12]. We also report the performance of the multi-vector models initialized with ColBERTv2[19] in Table 2. We next compare the overall performance of JMPQ and baseline models, and then analyze the details from the following three aspects.

5.1 Overall Comparison with Retrieval Models

Table 1: Overall Comparison with Retrieval Models on MARCO Passage, DL2019 and DL2020. */** denotes the difference between JMPQ and the baselines at $p < 0.05/0.01$ level using the two-tailed pairwise t-test.

Model	**Index**	**MARCO**		**DL2019**		**DL2020**		**Latency**
	GB	M@10	R@100	N@10	R@100	N@10	R@100	ms
BM25[18]	0.59	0.187**	0.670**	0.497**	0.497**	0.488**	0.567**	60
ANCE[25]	25	0.330**	0.852**	0.645**	0.548**	0.646*	0.640**	7600
ADORE[25]	25	0.347*	0.876	0.683	0.582**	0.665	0.673*	7600
JPQ[23]	0.83	0.341**	0.868**	0.677	0.575**	0.671	0.670*	720
RepCONC[24]	0.47	0.340**	0.864**	0.668*	0.569**	0.666	0.640**	346
ColBERT[12]	147	**0.361**	0.873**	0.706	0.587**	0.676	0.683*	423
PQ+RQ[19]	23	0.360	0.866**	0.704	0.588**	0.681	0.669*	230
PQ[10]	14	0.344**	0.860**	0.684	0.564**	0.650**	0.656**	522
JMPQ	14	0.356	**0.881**	**0.717**	**0.636**	**0.693**	**0.715**	522

According to the results in Table 1, JMPQ achieves competitive ranking effectiveness compared with the uncompressed ColBERT[12] with a 10x smaller index. Meanwhile, JMPQ significantly outperforms ColBERT on DL2019 and

DL2020. As for the query latency, the encoding and re-ranking stages of ColBERT, PQ+RQ[19], PQ[10] and JMPQ are done with one GeForce 2080Ti GPU, and the index searching stage is measured with one Intel Xeon E5-2630 V4 CPU (single thread). JMPQ is slightly slower than the original ColBERT. The reason is that JMPQ introduces extra time cost in the reconstruction process, while ColBERT simply loads the embeddings from disk. We also notice that PQ+RQ has a larger reduction in latency. We contribute this to its Index Inversion technique. Table 2 shows a similar result that the proposed JMPQ is able to achieve comparable results with a compressed index. Note that the R@100 we report on DL2019 are different from the results in JPQ paper because different thresholds are used to calculate the metrics.

Table 2: Comparison with Multi-vector Retrieval Models with ColBERTv2 as initialization. PQ+RQ denotes the compression method used in ColBERTv2

Model	**Index**	**MARCO**		**DL2019**		**DL2020**	
	GB	M@10	R@100	N@10	R@100	N@10	R@100
ColBERT v2[19]	147	**0.399**	**0.911**	0.744	0.638	**0.754**	0.755
PQ+RQ (ColBERT v2)	23	0.396	0.907	0.747	0.643	0.750	**0.761**
PQ (ColBERT v2)	14	0.386	0.904	0.744	0.636	0.735	0.741
JMPQ (ColBERT v2)	14	0.390	**0.911**	**0.752**	**0.646**	0.742	0.748

5.2 Comparison with Multi-vector Retrieval Models

This section compares JMPQ with the uncompressed multi-vector retrieval models. Since the index compression process of JMPQ introduces information loss, it is reasonable that its ranking effectiveness is inferior to the uncompressed ColBERT[12]. However, according to the results, the ranking effectiveness of JMPQ is competitive with or even outperforms the uncompressed ColBERT, which indicates that joint optimization can lead to significant effectiveness improvement without extra supervised signals.

From Table 2, we can see that JMPQ has only marginal improvement compared to the uncompressed ColBERTv2, which is less than the improvement of ColBERTv1 in Table 1. We believe the primary reason is that ColBERTv2 introduces hard negative sampling during training, which is consistent with the training strategy of JMPQ, and thus the improvement is not as large as expected.

5.3 Comparison with Other Compression Methods

This section compares JMPQ with other compression methods.

According to Table 1 and Table 2, the unsupervised PQ[10] severely hurts the ranking effectiveness compared with the uncompressed ColBERT[12], while JMPQ can significantly outperform the PQ baseline by leveraging a joint optimization framework. Compared with PQ+RQ[19], which is similar to JMPQ and quantizes the residual embeddings with PQ and RQ, JMPQ still substantially outperforms it with a relatively smaller index at most of the metrics for both the ColBERTv1 and v2 initialization. It further demonstrates the benefits of the supervised signals and the joint optimization framework.

As for the supervised compression methods, which also utilize joint optimization, RepCONC[24] and JPQ[23] achieve notable ranking effectiveness among the single-vector models while significantly reducing the index size. JMPQ outperforms them by a large margin. We attribute this to the powerful representation capabilities of the multi-vector models over the single-vector ones. It is worth noticing that the compressed index of single-vector models, such as 0.83G for JPQ, is much smaller and is about 1/17 of the index size of JMPQ. This is because, in the multi-vector scenario, the number of embedding vectors of the corpus could be hundreds of times greater than the single-vector scenario. In our experiments, the corpus has 8.8M embeddings for JPQ and RepCONC, while it has over 590M embeddings for JMPQ, which is over 60x than the single-vector models. Additionally, according to Section 3.2, a single vector takes an extra 8 bytes in the IVF for just storing the embedding ids, which in total occupies 4.7G storage space.

5.4 Comparison with Single-vector Retrieval Models

We now compare JMPQ with competitive single-vector dense retrieval baselines. According to the results, JMPQ substantially outperforms all the single-vector baselines. Compared with ANCE[22], ADORE[25], and TCT-ColBERT[14], which are uncompressed dense retrieval models, JMPQ achieves impressive effectiveness improvement while the compressed index size is almost halved, as JMPQ enables modeling token-level interactions with multi-vector representations.

6 Conclusions

This paper presents JMPQ based on JPQ. It jointly optimizes the encoding and the compression processes in an end-to-end manner in the multi-vector representation scenario. We conduct experiments on popular ad-hoc retrieval benchmarks, where JMPQ achieves competitive or better ranking effectiveness than the uncompressed dense retrieval models, with over 10x compression on index size. JMPQ also substantially improves the ranking performance compared with JPQ and RepCONC due to the strong representation capabilities of the multi-vector models. The results demonstrate the effectiveness of JMPQ and highlight that a compressed embedding index can benefit from supervised signals and be effective in the first-stage retrieval. It is worth noticing that although we only conduct experiments based on ColBERT and the MaxSim aggregation method, we believe that JMPQ can be applied to any multi-vector models as well as to any inner-product-based aggregation methods.

7 Acknowledgment

This work is supported by the Natural Science Foundation of China (Grant No. 61732008) and Tsinghua University Guoqiang Research Institute.

References

1. Barnes, C.F., Rizvi, S.A., Nasrabadi, N.M.: Advances in residual vector quantization: A review. IEEE transactions on image processing **5**(2), 226–262 (1996)
2. Craswell, N., Mitra, B., Yilmaz, E., Campos, D.: Overview of the trec 2020 deep learning track (2021)
3. Craswell, N., Mitra, B., Yilmaz, E., Campos, D., Voorhees, E.M.: Overview of the trec 2019 deep learning track. arXiv preprint arXiv:2003.07820 (2020)
4. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
5. Gao, L., Dai, Z., Callan, J.: Coil: Revisit exact lexical match in information retrieval with contextualized inverted list. arXiv preprint arXiv:2104.07186 (2021)
6. Ge, T., He, K., Ke, Q., Sun, J.: Optimized product quantization. IEEE transactions on pattern analysis and machine intelligence **36**(4), 744–755 (2013)
7. Guo, J., Cai, Y., Fan, Y., Sun, F., Zhang, R., Cheng, X.: Semantic models for the first-stage retrieval: A comprehensive review. arXiv preprint arXiv:2103.04831 (2021)
8. Hartigan, J.A., Wong, M.A.: Algorithm as 136: A k-means clustering algorithm. Journal of the royal statistical society. series c (applied statistics) **28**(1), 100–108 (1979)
9. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. pp. 604–613 (1998)
10. Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE transactions on pattern analysis and machine intelligence **33**(1), 117–128 (2010)
11. Johnson, J., Douze, M., Jégou, H.: Billion-scale similarity search with gpus. IEEE Transactions on Big Data **7**(3), 535–547 (2019)
12. Khattab, O., Zaharia, M.: Colbert: Efficient and effective passage search via contextualized late interaction over bert. In: Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval. pp. 39–48 (2020)
13. Lin, J., Nogueira, R., Yates, A.: Pretrained transformers for text ranking: Bert and beyond. Synthesis Lectures on Human Language Technologies **14**(4), 1–325 (2021)
14. Lin, S.C., Yang, J.H., Lin, J.: Distilling dense representations for ranking using tightly-coupled teachers. arXiv preprint arXiv:2010.11386 (2020)
15. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692 (2019)
16. Luan, Y., Eisenstein, J., Toutanova, K., Collins, M.: Sparse, dense, and attentional representations for text retrieval. Transactions of the Association for Computational Linguistics **9**, 329–345 (2021)

17. Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., Deng, L.: Ms marco: A human generated machine reading comprehension dataset. In: CoCo@ NIPS (2016)
18. Robertson, S.E., Walker, S.: Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In: SIGIR'94. pp. 232–241. Springer (1994)
19. Santhanam, K., Khattab, O., Saad-Falcon, J., Potts, C., Zaharia, M.: Colbertv2: Effective and efficient retrieval via lightweight late interaction (2021)
20. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
21. Xiao, S., Liu, Z., Shao, Y., Lian, D., Xie, X.: Matching-oriented product quantization for ad-hoc retrieval. arXiv preprint arXiv:2104.07858 (2021)
22. Xiong, L., Xiong, C., Li, Y., Tang, K.F., Liu, J., Bennett, P., Ahmed, J., Overwijk, A.: Approximate nearest neighbor negative contrastive learning for dense text retrieval. arXiv preprint arXiv:2007.00808 (2020)
23. Zhan, J., Mao, J., Liu, Y., Guo, J., Zhang, M., Ma, S.: Jointly optimizing query encoder and product quantization to improve retrieval performance. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management. pp. 2487–2496 (2021)
24. Zhan, J., Mao, J., Liu, Y., Guo, J., Zhang, M., Ma, S.: Learning discrete representations via constrained clustering for effective and efficient dense retrieval (2021)
25. Zhan, J., Mao, J., Liu, Y., Guo, J., Zhang, M., Ma, S.: Optimizing dense retrieval model training with hard negatives. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 1503–1512 (2021)
26. Zhan, J., Mao, J., Liu, Y., Guo, J., Zhang, M., Ma, S.: Optimizing dense retrieval model training with hard negatives. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 1503–1512 (2021)