Compact Token Representations with Contextual Quantization for Efficient Document Re-ranking

Yingrui Yang, Yifan Qiao, Tao Yang
Department of Computer Science, University of California at Santa Barbara, USA
{yingruiyang, yifanqiao, tyang}@cs.ucsb.edu

Abstract

Transformer based re-ranking models can achieve high search relevance through context-aware soft matching of query tokens with document tokens. To alleviate runtime complexity of such inference, previous work has adopted a late interaction architecture with pre-computed contextual token representations at the cost of a large online storage. This paper proposes contextual quantization of token embeddings by decoupling document-specific and document-independent ranking contributions during codebook-based compression. This allows effective online decompression and embedding composition for better search relevance. This paper presents an evaluation of the above compact token representation model in terms of relevance and space efficiency.

1 Introduction

Modern search engines for text documents typically employ multi-stage ranking. The first retrieval stage extracts top candidate documents matching a query from a large search index with a simple ranking method. The second stage or a later stage uses a more complex machine learning algorithm to re-rank top results thoroughly. Recently neural re-ranking techniques from transformer-based architectures have achieved impressive relevance scores for top k document re-ranking, such as MacAvaney et al. (2019). However, using a transformer-based model to rank or re-rank is extremely expensive during the online inference (Lin et al., 2020). Various efforts have been made to reduce its computational complexity (e.g. Gao et al. (2020)).

A noticeable success in time efficiency improvement is accomplished in ColBERT (Khattab and Zaharia, 2020) which conducts late interaction of query terms and document terms during runtime inference so that token embeddings for documents can be pre-computed. Using ColBERT re-ranking after a sparse retrieval model called DeepImpact (Mallia et al., 2021) can further enhance relevance. Similarly BECR (Yang et al., 2022), CEDR-KNRM (MacAvaney et al., 2019), and PreTTR (MacAvaney et al., 2020) have also adopted the late interaction architecture in their efficient transformer based re-ranking schemes.

While the above work delivers good search relevance with late interaction, their improvement in time efficiency has come at the cost of a large storage space in hosting token-based precomputed document embeddings. For example, for the MS MARCO document corpus, the footprint of embedding vectors in ColBERT takes up to 1.6TB and hosting them in a disk incurs substantial time cost when many embeddings are fetched for re-ranking. It is highly desirable to reduce embedding footprints and host them in memory as much as possible for fast and high-throughput access and for I/O latency and contention avoidance, especially when an online re-ranking server is required to efficiently process many queries simultaneously.

The **contribution** of this paper is to propose a compact representation for contextual token embeddings of documents called Contextual Quantization (CQ). Specifically, we adopt codebook-based quantization to compress embeddings while explicitly decoupling the ranking contributions of document specific and document-independent information in contextual embeddings. These ranking contributions are recovered with weighted composition after quantization decoding during online inference. Our CQ scheme includes a neural network model that jointly learns context-aware decomposition and quantization with an objective to preserve correct ranking scores and order margins. Our evaluation shows that CQ can effectively reduce the storage space of contextual representation by about 14 times for the tested datasets with insignificant online embedding recovery overhead and a small relevance degradation for re-ranking passages or documents.

Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 695 - 707

2 Problem Definition and Related Work

The problem of neural text document re-ranking is defined as follows. Given a query with multiple terms and a set of candidate documents, rank these documents mainly based on their embeddings and query-document similarity. With a BERT-based re-ranking algorithm, typically a term is represented by a token, and thus in this paper, word "term" is used interchangeably with "token". This paper is focused on minimizing the space cost of token embeddings for fast online re-ranking inference.

Deep contextual re-ranking models. Neural re-ranking has pursued representation-based or interaction-based algorithms (Guo et al., 2016; Dai et al., 2018; Xiong et al., 2017). Embedding interaction based on query and document terms shows an advantage in these studies. The transformer architecture based on BERT (Devlin et al., 2019) has been adopted to re-ranking tasks by using BERT's [CLS] token representation to summarize query and document interactions (Nogueira and Cho, 2019; Yang et al., 2019; Dai and Callan, 2019; Nogueira et al., 2019a; Li et al., 2020). Recently BERT is integrated in late term interaction (MacAvaney et al., 2019; Hofstätter et al., 2020c,b; Mitra et al., 2021) which delivers strong relevance scores for re-ranking.

Efficiency optimization for transformer-based re-ranking. Several approaches have been proposed to reduce the time complexity of transformer-based ranking. For example, architecture simplification (Hofstätter et al., 2020c; Mitra et al., 2021), late interaction with precomputed token embeddings (MacAvaney et al., 2020), early exiting (Xin et al., 2020), and model distillation (Gao et al., 2020; Hofstätter et al., 2020a; Chen et al., 2020b). We will focus on the compression of token representation following the late-interaction work of ColBERT (Khattab and Zaharia, 2020) and BECR (Yang et al., 2022) as they deliver fairly competitive relevance scores for several well-known ad-hoc TREC datasets. These late-interaction approaches follow a dual-encoder design that separately encodes the two sets of texts, studied in various NLP tasks (Zhan et al., 2020; Chen et al., 2020a; Reimers and Gurevych, 2019; Karpukhin et al., 2020; Zhang et al., 2020).

Several previous re-ranking model attempted to reduce the space need for contextual token embeddings. ColBERT has considered an option of using a smaller dimension per vector and limiting 2 bytes per number as a scalar quantization. BECR (Yang et al., 2022) uses LSH for hashing-based contextual embedding compression (Ji et al., 2019). PreTTR (MacAvaney et al., 2020) uses a single layer encoder model to reduce the dimensionality of each token embedding. Following PreTTR, a contemporaneous work called SDR in Cohen et al. (2021) considers an autoencoder to reduce the dimension of representations, followed by an off-the-shelf scalar quantizer. For the autoencoder, it combines static BERT embeddings with contextual embeddings. Inspired by this study, our work decomposes contextual embeddings to decouple ranking contributions during vector quantization. Unlike SDR, CQ jointly learns the codebooks and decomposition for the document-independent and dependent components guided by a ranking loss.

Vector quantization. Vector quantization with codebooks was developed for data compression to assist approximate nearest neighbor search, for example, product quantizer (PQ) from Jégou et al. (2011), optimized product quantizer (OPQ) from Ge et al. (2013); residual additive quantizer(RQ) from Ai et al. (2015) and local search additive quantizer (LSQ) from Martinez et al. (2018). Recently such a technique has been used for compressing static word embeddings (Shu and Nakayama, 2018) and document representation vectors in a dense retrieval scheme called JPQ (Zhan et al., 2021a). None of the previous work has worked on quantization of contextual token vectors for the re-ranking task, and that is the focus of this paper.

3 Contextual Quantization

Applying vector quantization naively to token embedding compression does not ensure the ranking effectiveness because a quantizer-based compression is not lossless, and critical ranking signals could be lost during data transformation. To achieve a high compression ratio while maintaining the competitiveness in relevance, we consider the ranking contribution of a contextual token embedding for soft matching containing two components: 1) document specific component derived from the self attention among context in a document, 2) document-independent and corpus-specific component generated by the transformer model. Since for a reasonable sized document set, the second component is invariant to documents, its storage space is negligible compared to the first component. Thus the second part does not need compres-

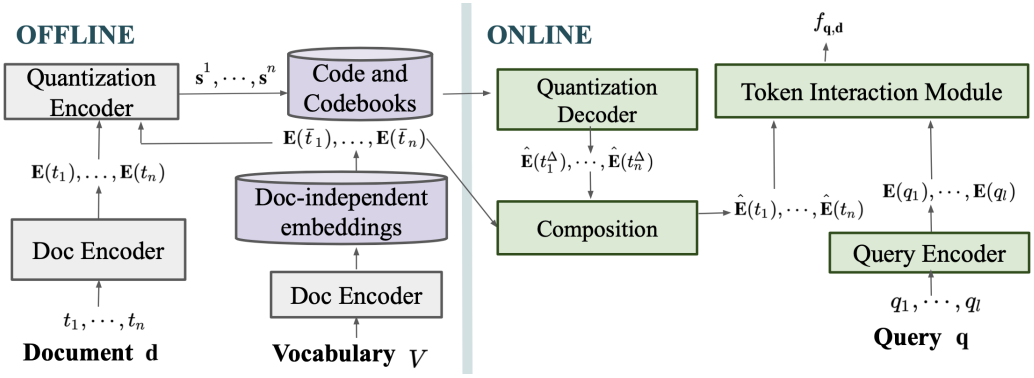


Figure 1: Offline processing and online ranking with contextual quantization

sion. We focus on compressing the first component using codebooks. This decomposition strategy can reduce the relevance loss due to compression approximation, which allows a more aggressive compression ratio. Our integrated vector quantizer with contextual decomposition contains a ranking-oriented scheme with an encoder and decoder network for jointly learning codebooks and composition weights. Thus, the online composition of decompressed document-dependent information with document-independent information can retain a good relevance.

3.1 Vector Quantization and Contextual Decomposition

A vector quantizer consists of two steps as discussed in Shu and Nakayama (2018). In the compression step, it encodes a real-valued vector (such as a token embedding vector in our case) into a short code using a neural encoder. The short code is a list of reference indices to the codewords in codebooks. During the decompression step, a neural decoder is employed to reconstruct the original vector from the code and codebooks.

The quantizer learns a set of M codebooks $\{\mathcal{C}^1, \mathcal{C}^2, \cdots, \mathcal{C}^M\}$ and each codebook contains K codewords ($\mathcal{C}^m = \{\mathbf{c}_1^m, \mathbf{c}_2^m, \cdots, \mathbf{c}_K^m\}$) of dimension h. Then for any D-dimensional real valued vector $\mathbf{x} \in \mathbb{R}^D$, the encoder compresses $\mathbf{x}$ into an M dimensional code vector $\mathbf{s}$. Each entry of code $\mathbf{s}$ is an integer j, denoting the j-th codeword in codebook $\mathcal{C}^m$. After locating all M codewords as $[\mathbf{c}^1, \cdots, \mathbf{c}^M]$, the original vector can be recovered with two options. For a product quantizer, the dimension of codeword is $h = D/M$, and the decompressed vector is $\hat{\mathbf{x}} = \mathbf{c}^1 \circ \mathbf{c}^2 \cdots \circ \mathbf{c}^M$ where symbol $\circ$ denotes vector concatenation. For an additive quantizerthe decompressed vector is $\hat{\mathbf{x}} = \sum_{j=1}^{M} \mathbf{c}^j$.

Codebook-based contextual quantization. Now we describe how codebook-based compression is used in our contextual quantization. Given a token t, we consider its contextual embedding vector $\mathbf{E}(t)$ as a weighted combination of two components: $\mathbf{E}(t^{\Delta})$ and $\mathbf{E}(\bar{t})$. $\mathbf{E}(t^{\Delta})$ captures the document-dependent component, and $\mathbf{E}(\bar{t})$ captures the document-independent component discussed earlier. For a transformer model such as BERT, $\mathbf{E}(t)$ is the token output from the last encoder layer, and we obtain $\mathbf{E}(\bar{t})$ by feeding [CLS] $\circ$ t $\circ$ [SEP] into BERT model and taking last layer's output for t.

During offline data compression, we do not explicitly derive $\mathbf{E}(t^{\Delta})$ as we only need to store the compressed format of such a value, represented as a code. Let $\hat{\mathbf{E}}(t^{\Delta})$ be the recovered vector with codebook-based decompression, as a close approximation of $\mathbf{E}(t^{\Delta})$. Let $\hat{\mathbf{E}}(t)$ be the final composed embedding used for online ranking with late-interaction. Then $\hat{\mathbf{E}}(t) = g(\hat{\mathbf{E}}(t^{\Delta}), \mathbf{E}(\bar{t}))$ where $g(.)$ is a simple feed-forward network to combine two ranking contribution components.

The encoder/decoder neural architecture for contextual quantization. We denote a token in a document $\mathbf{d}$ as t. The input to the quantization encoder is $\mathbf{E}(t) \circ \mathbf{E}(\bar{t})$. The output of the quantization encoder is the code vector $\mathbf{s}$ of dimension M. Let code $\mathbf{s}$ be $(s_1, \cdots, s_m, \cdots, s_M)$ and each entry s_m will be computed below in Eq. 4. This computation uses the hidden layer $\mathbf{h}$ defined as:

$$\mathbf{h} = \tanh(\mathbf{w}_0(\mathbf{E}(t) \circ \mathbf{E}(\bar{t})) + \mathbf{b}_0). \quad (1)$$

The dimension of $\mathbf{h}$ is fixed as $1 \times MK/2$. The hidden layer $\mathbf{a}$ is computed by a feed forward layer with a softplus activation (Eq. 2) with an output dimension of $M \times K$ after reshaping, Let $\mathbf{a}^m$ be

the m-th row of this output.

$$\mathbf{a}^m = \text{softplus}(\mathbf{w}_1^m \mathbf{h} + \mathbf{b}_1^m). \quad (2)$$

To derive a discrete code entry for s_m, following the previous work (Shu and Nakayama, 2018), we apply the Gumbel-softmax trick (Maddison et al., 2017; Jang et al., 2017) as shown in Eq. 3, where the temperature τ is fixed at 1 and ϵ_k is a noise term sampled from the Gumbel distribution $-\log(-\log(\text{Uniform}[0,1]))$. Here $\mathbf{p}^m$ is a vector with dimension K. $(\mathbf{p}^m)_j$ is the j-th entry of the vector. Similarly, $(\mathbf{a}^m)_j$ is the j-th entry of $\mathbf{a}^m$.

$$(\mathbf{p}^m)_j = \frac{\exp(\log((\mathbf{a}^m)_j + \epsilon_j)/\tau)}{\sum_{j'=1}^{K} \exp(\log((\mathbf{a}^m)_{j'} + \epsilon_{j'})/\tau)}. \quad (3)$$

$$s_m = \arg \max_{1 \le j \le K} (\mathbf{p}^m)_j. \quad (4)$$

In the decompression stage, the input to the quantization decoder is the code $\mathbf{s}$, and this decoder accesses M codebooks $\{\mathcal{C}^1, \mathcal{C}^2, \cdots, \mathcal{C}^M\}$ as M parameter matrices of size $K \times h$ which will be learned. For each m-entry of code $\mathbf{s}$, s_m value is the index of row vector in $\mathcal{C}^m$ to be used as its corresponding codeword. Once all codewords $\mathbf{c}^1$ to $\mathbf{c}^M$ are fetched, we recover the approximate vector $\hat{\mathbf{E}}(t^{\Delta})$ as $\sum_{j=1}^{M} \mathbf{c}^j$ for additive quantization or $\mathbf{c}^1 \circ \mathbf{c}^2 \cdots \circ \mathbf{c}^M$ for product quantization.

Next, we perform a composition with a one-layer or two-layer feed-forward network to derive the contextual embedding as $\hat{\mathbf{E}}(t) = g(\hat{\mathbf{E}}(t^{\Delta}, \mathbf{E}(\bar{t}))$. With one feed-forward layer,

$$\hat{\mathbf{E}}(t) = \tanh(\mathbf{w}_2(\hat{\mathbf{E}}(t^{\Delta}) \circ \mathbf{E}(\bar{t})) + \mathbf{b}_2). \quad (5)$$

The above encoder and decoder for quantization have parameter $\mathbf{w}_0, \mathbf{b}_0, \mathbf{w}_1, \mathbf{b}_1, \mathbf{w}_2, \mathbf{b}_2$, and $\{\mathcal{C}^1, \mathcal{C}^2, \cdots, \mathcal{C}^M\}$. These parameters are learned through training. Once these parameters are learned, the quantization model is fixed and the code for any new token embedding can be computed using Eq. 4 in offline processing.

Figure 1 depicts the flow of offline learning and the online inference with context quantization. Given a query with l tokens $\{q_1, q_2, ..q_l\}$, and a documents with n tokens $\{t_1, t_2, ..t_n\}$, The query token embeddings encoded with a transformer based model (e.g. BERT) are denoted as $\mathbf{E}(q_1), \cdots, \mathbf{E}(q_l)$. The embeddings for document tokens through codebook base decompression are $\hat{\mathbf{E}}(t_1), \cdots \hat{\mathbf{E}}(t_n)$. The online inference then uses the interaction of query tokens and document tokens defined in a re-ranking algorithm such as ColBERT to derive a ranking score (denoted as $f_{\mathbf{q},\mathbf{d}}$).

The purpose of injecting $\mathbf{E}(\bar{t})$ in Eq. 1 is to decouple the document-independent ranking contribution from contextual embedding $\hat{\mathbf{E}}(t^{\Delta})$ so that this quantization encoder model will be learned to implicitly extract and compress the document-dependent ranking contribution.

Table 1 gives an example with several token codes produced by CQ for different sentences representing different contexts, and illustrates context awareness of CQ's encoding with a small codebook dimension (M=K=4). For example, 1 in code [4, 4, 3, 1] means the 4-th dimension uses the first codeword of the corresponding codebook. Training of CQ uses the MS MARCO passage dataset discussed in Section 4 and these sentences are not from this dataset. Our observation from this example is described as follows. First, in general token codes in the same sentences are closer to each other, and token codes in different sentences, even with the same word "bank", are far away with a visible Hamming distance. Thus CQ coding allows a context-based separation among tokens residing in different contexts. Second, by looking at boldfaced tokens at each sentence, their distance in terms of contextual semantics and proximity is reflected to some degree in their CQ codes. For instance, a small Hamming code distance of three words "actor", "poet" and "writer" resembles their semantic and positional closeness. A larger code distance of two "bank"s in the 3^{rd} and 4^{th} sentences relates with their word sense and positional difference.

Training loss for parameter learning. We have explored three training loss functions. The first option is to follow a general quantizer (Shu and Nakayama, 2018) using the mean squared error (MSE) between the reconstructed and original embedding vectors of all token t_i. Namely $\mathcal{L}_{MSE} = \sum \|\mathbf{E}(t_i) - \hat{\mathbf{E}}(t_i)\|_2^2$.

The second option is the pairwise cross-entropy loss based on rank orders. After warming up with the MSE loss, we further train the quantizer using $\mathcal{L}_{PairwiseCE} = \sum(-\sum_{j=\mathbf{d}^+,\mathbf{d}^-} P_j \log P_j)$ where $\mathbf{d}^+$ and $\mathbf{d}^-$ are positive and negative documents for query q.

We adopt the third option which borrows the idea of MarginMSE loss from Hofstätter et al. (2020a) proposed for BERT-based ranking model distillation. In MarginMSE, a student model is trained to

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

Table 1: Example context-aware token codes produced by CQ using M=K=4 for the illustration purpose.

mimic the teacher model in terms of both ranking scores as well as the document relative order margins. In our case, the teacher model is the ranking model without quantization and the student model is the ranking model with quantization. It is defined as $\mathcal{L}_{MarginMSE} = \sum((f_{\mathbf{q},\mathbf{d}^+} - f_{\mathbf{q},\mathbf{d}^-}) - (\hat{f}_{\mathbf{q},\mathbf{d}^+} - \hat{f}_{\mathbf{q},\mathbf{d}^-}))^2$, where $f_{\mathbf{q},\mathbf{d}}$ and $\hat{f}_{\mathbf{q},\mathbf{d}}$ denote the ranking score with and without quantization, respectively. The above loss function distills the ColBERT ranking characteristics into the CQ model for better preservation of ranking effectiveness.

3.2 Related Online Space and Time Cost

Online space for document embeddings. The storage cost of the precomputed document embeddings in a late-interaction re-ranking algorithm is dominating its online space need. To recover token-based document embeddings, an online server with contextual quantization stores three parts: codebooks, the short codes of tokens in each document, and the document-independent embeddings.

Given a document collection of Z documents of length n tokens on average, let V be the number of the distinct tokens. For M codebooks with $M * K$ codewords of dimension h, we store each entry of a codeword with a 4-byte floating point number. Thus the space cost of codebooks is $M * K * h * 4$ bytes, and the space for document-independent embeddings of dimension D is $V * D * 4$ bytes. When $M = 16, K = 256, D = 128$ as in our experiments, if we use the product quantization with the hidden dimension $h = 8$, the codebook size is 131 MB. In the WordPiece English token set for BERT, $V \approx 32K$ and the space for document-independent embeddings cost about 16.4 MB. Thus the space cost of the above two parts is insignificant.

The online space cost of token-based document embeddings is $Z * n * (\frac{M \log_2 K}{8} + 2)$ bytes. Here each contextual token embedding of length D is encoded into a code of length M and the space of each code costs $\log_2 K$ bits. For each document, we also need to store the IDs of its tokens in order to access document-independent embeddings. We use 2 bytes per token ID in our evaluation because the BERT dictionary based on WordPiece (Wu et al., 2016) tokenizer has about 32,000 tokens.

In comparison, the space for document embeddings in ColBERT with 2 bytes per number costs $Z*D*n*2$ bytes. Then the space ratio of ColBERT without CQ and with CQ is about $\frac{2D \times 8}{M \log_2 K + 2 \times 8}$, which is about 14:1 when $D = 128$, $M = 16$ and $K = 256$. BECR uses 5 layers of the refinement outcome with the BERT encoder for each token and stores each layer of the embedding with a 256 bit LSH signature. Thus the space cost ratio of BECR over ColBERT-CQ is approximately $\frac{5 \times 256}{M \log_2 K + 2 \times 8}$, which is about 9:1 when $M = 16$ and $K = 256$. We can adjust the parameters of each of ColBERT, BECR, and ColBERT-CQ for a smaller space with a degraded relevance, and their space ratio to CQ remains large, which will be discussed in Section 4.

Time cost for online decompression and composition. Let k be the number of documents to re-rank. The cost of decompression with the short code of a token using the cookbooks is $O(M * h)$ for a product quantizer and $O(M * D)$ for an additive quantizer. Notice $M * h = D$. For a one-layer feed-forward network as a composition to recover the final embedding, the total time cost for decompression and composition is $O(\mathrm{k} * n * D^2)$ with a product quantizer, and $O(\mathrm{k} * n(M * D + D^2))$ with an additive quantizer. When using two hidden layers with D dimensions in the first layer output, there is some extra time cost but the order of time complexity remains unchanged.

Noted that because of using feed-forward layers in final recovery, our contextual quantizer cannot take advantage of an efficiency optimization called

asymmetric distance computation in Jégou et al. (2011). Since embedding recovery is only applied to top k documents after the first-stage retrieval, the time efficiency for re-ranking is still reasonable without such an optimization.

4 Experiments and Evaluation Results

4.1 Settings

Dataset	# Query	# Doc	Mean Doc Length	# Judgments per query
MS MARCO passage Dev	6980	8.8M	67.5	1
TREC DL 19 passage	200	–	–	21
TREC DL 20 passage	200	–	–	18
MS MARCO doc Dev	5193	3.2M	1460	1
TREC DL 19 doc	200	–	–	33

Table 2: Dataset statistics. Mean doc length is the average number of WordPiece (Wu et al., 2016) tokens.

Datasets and metrics. The well-known MS MARCO passage and document ranking datasets are used. As summarized the in Table 2, our evaluation uses the MS MARCO document and passage collections for document and passage ranking (Craswell et al., 2020; Campos et al., 2016). The original document and passage ranking tasks provide 367,013 and 502,940 training queries respectively, with about one judgment label per query. The development query sets are used for relevance evaluation. The TREC Deep Learning (DL) 2019 and 2020 tracks provide 200 test queries with many judgment labels per query for each task.

Following the official leader-board standard, for the development sets, we report mean reciprocal rank (MRR@10, MRR@100) for relevance instead of using normalized discounted cumulative gain (NDCG) (Järvelin and Kekäläinen, 2002) because such a set has about one judgment label per query, which is too sparse to use NDCG. For TREC DL test sets which have many judgement lables per query, we report the commonly used NDCG@10 score. We also measure the dominating space need of the embeddings in bytes and re-ranking time latency in milliseconds. To evaluate latency, we uses an Amazon AWS g4dn instance with Intel Cascade Lake CPUs and an NVIDIA T4 GPU.

In all tables below that compare relevance, we perform paired t-test on 95% confidence levels. In Tables 3, 4, and 5, we mark the results with '†' if the compression method result in statistically significant degradation from the ColBERT baseline. In Table 6, '†' is marked for numbers with statistically significant degradation from default setting in the first row.

Choices of first-stage retrieval models. To retrieve top 1,000 results before re-ranking, we consider the standard fast BM25 method (Robertson and Zaragoza, 2009). We have also considered sparse and dense retrievers that outperform BM25. We have used uniCOIL (Lin and Ma, 2021; Gao et al., 2021) as an alternative sparse retriever in Table 3 because it achieves a similar level of relevance as end-to-end ColBERT with a dense retriever, and that of other learned sparse representations (Mallia et al., 2021; Formal et al., 2021b,a). ColBERT+uniCOIL has 0.369 MRR while end-to-end ColBERT has 0.360 MRR on MSMARCO Dev set. Moreover, retrieval with a sparse representation such as uniCOIL and BM25 normally uses much less computing resources than a dense retriever. Relevance numbers reported in some of the previous work on dense retrieval are derived from the exact search as an upper bound of accuracy. When non-exact retrieval techniques such as approximate nearest neighbor or maximum inner product search are used on a more affordable platform for large datasets, there is a visible loss of relevance (Lewis et al., 2021). It should be emphasized that the first stage model can be done by either a sparse or a dense retrieval, and this does not affect the applicability of CQ for the second stage as the focus of this paper.

Re-ranking models and quantizers compared. We demonstrate the use of CQ for token compression in ColBERT in this paper. We compare its relevance with ColBERT, BECR and PreTTR. We chose to apply CQ to ColBERT because assuming embeddings are in memory, ColBERT is one of the fastest recent online re-ranking algorithms with strong relevance scores and CQ addresses its embedding storage weakness. Other re-ranking models compared include: BERT-base (Devlin et al., 2019), a cross encoder re-ranker, which takes a query and a document at run time and uses the last layers output from the BERT [CLS] token to generate a ranking score; TILDEv2 (Zhuang and Zuccon, 2021), which expands each document and additively aggregates precomputed neural scores.

We also evaluate the use of unsupervised quantization methods discussed in Section 2 for ColBERT, including two product quantizers (PQ and OPQ), and two additive quantizers (RQ and LSQ).

Appendix A has additional details on the retrievers considered, re-ranker implementation, training, and relevance numbers cited.

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

Table 3: Relevance scores for MS MARCO passage ranking. The % degradation from ColBERT is listed and '†' is marked for statistically significant drop.

Model Specs.	Dev MRR@100	TREC DL19 NDCG@10
	Retrieval choices	
BM25	0.203	0.446
docT5query	0.289	0.569
DeepCT*	0.320	0.544
TCT-ColBERT(v2)	0.351	–
JPQ*	0.401	0.623
uniCOIL	0.343	0.641
	Re-ranking baselines (+BM25 retrieval)	
BERT-base*	0.393	0.670
ColBERT	0.410	0.714
	Quantization (+BM25 retrieval)	
ColBERT-PQ	0.400† (-2.4%)	0.702 (-1.7%)
ColBERT-OPQ	0.404† (-1.5%)	0.704 (-1.4%)
ColBERT-RQ	–	0.704 (-1.4%)
ColBERT-LSQ	–	0.707 (-1.0%)
ColBERT-CQ	0.405† (-1.2%)	0.712 (-0.3%)

Table 4: Relevance scores for MS MARCO document ranking. The % degradation from ColBERT is listed and '†' is marked for statistically significant drop.

4.2 A Comparison of Relevance

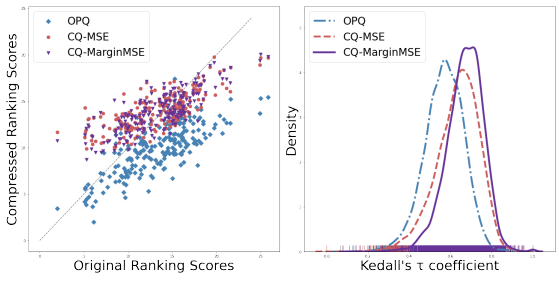
Table 3 and Table 4 show the ranking relevance in NDCG and MRR of the different methods and compare against the use of CQ with ColBERT (marked as ColBERT-CQ). We either report our experiment results or cite the relevance numbers from other papers with a * mark for such a model. For quantization approaches, we adopt M=16, K=256, i.e. compression ratio 14:1 compared to ColBERT.

For the passage task, ColBERT outperforms other re-rankers in relevance for the tested cases. ColBERT-CQ after BM25 or uniCOIL retrieval only has a small relevance degradation with around 1% or less, while only requiring 3% of the storage of ColBERT. The relevance of the ColBERT-CQ+uniCOIL combination is also competitive to the one reported in Mallia et al. (2021) for the ColBERT+DeepImpact combination which has MRR 0.362 for the Dev query set, NDCG@10 0.722 for TREC DL 2019 and 0.691 for TREC DL 2020.

For the document re-ranking task, Table 4 similarly confirms the effectiveness of ColBERT-CQ. ColBERT-CQ and ColBERT after BM25 retrieval also perform well in general compared to the relevance results of the other baselines.

From both Table 3 and Table 4, we observe that in general, CQ significantly outperforms the other quantization approaches (PQ, OPQ, RQ, and LSQ). As an example, we further explain this by plotting the ranking score of ColBERT with and without a quantizer in Figure 2(a). Compared to OPQ, CQ trained with two loss functions generates ranking scores much closer to the original ColBERT ranking score, and this is also reflected in Kendall's τ correlation coefficients of top 1,000 re-ranked results between a quantized ColBERT and the original ColBERT (Figure 2(b)). There are two reasons that CQ outperforms the other quantizers: 1) The previous quantizers do not perform contextual decomposition to isolate intrinsic context-independent information in embeddings, and thus their approximation yields more relevance loss; 2) Their training loss function is not tailored to the re-ranking task.

4.3 Effectiveness on Space Reduction



(a) (b)

Figure 2: (a) Ranking score by quantized ColBERT with OPQ and CQ using two loss functions, vs. original ColBERT score. (b) Distribution of Kendall's τ correlation coefficient between the 1,000 ranked results of quantized and original ColBERT.

Table 5 shows the estimated space size in bytes for embeddings in the MS MARCO document and

Model	Doc task Space	Passage task Space	Disk I/O	Latency	MRR@10
BECR	791G	89.9G	–	8ms	0.323
PreTTR*	–	2.6T	>182ms	>1000ms	0.358
TILDEv2*	–	5.2G	–	–	0.326
ColBERT	1.6T	143G	>182ms	16ms	0.355
ColBERT-small*	300G	26.8G	–	–	0.339
ColBERT-OPQ	112G	10.2G	–	56ms	0.324†
ColBERT-CQ					
undecomposed	112G	10.2G	–	17ms	0.339†
K=256	112G	10.2G	–	17ms	0.352
K=16	62G	5.6G	–	17ms	0.339†
K=4	37G	3.4G	–	17ms	0.326†

Table 5: Embedding space size in bytes for the document ranking task and for the passage ranking task. Re-ranking time per query and relevance for top 1,000 passages in milliseconds on a GPU using the Dev query set. M=16. For ColBERT-OPQ and ColBERT-CQ-undecomposed, K=256. Assume passage embeddings in PreTTR and ColBERT do not fit in memory. '†' is marked for MRR numbers with statistically significant degradation from the ColBERT baseline.

passage corpora, and compares CQ with other approaches. Each MS MARCO document is divided into overlapped passage segments of size up to 400 tokens, and there are 60 tokens overlapped between two consecutive passage segments, following the ColBERT setup. As a result, the number of WordPiece tokens per document changes from 1460 to about 2031 due to the addition of overlapping contextual tokens.

To demonstrate the tradeoff, we also list their estimated time latency and relevance in passage re-ranking as a reference and notice that more relevance comparison results are in Tables 3 and 4. The latency is the total time for embedding decompression/recovery and re-ranking.

For PreTTR and ColBERT, we assume that their passage embedding data cannot fit in memory given their large data sizes. The disk I/O latency number is based on their passage embedding size and our test on a Samsung 870 QVO solid-state disk drive to fetch 1,000 passage embeddings randomly. Their I/O latency takes 110ms or 182ms with single-thread I/O and with no I/O contention, and their disk access can incur much more time when multiple queries are processed in parallel in a server dealing with many clients. For example, fetching 1,000 passage embeddings for each of ColBERT and PreTTR takes about 1,001ms and 3,870ms respectively when the server is handling 16 and 64 queries simultaneously with multiple threads.

For other methods, their passage embedding data is relatively small and we assume that it can be preloaded in memory. The query latency reported in the 4-th column of Table 5 excludes the first-stage retrieval time. The default ColBERT uses embedding dimension 128 and 2 byte floating numbers. ColBERT-small denotes an optional configuration suggested from the ColBERT paper using 24 embedding dimensions and 2-byte floating numbers with a degraded relevance performance.

As shown in Table 5, the embedding footprint of ColBERT CQ uses about 112GB and 10.2GB, respectively for document and passage re-ranking tasks. By looking at the latency difference of ColBERT with and without CQ, the time overhead of CQ for decompression and embedding recovery takes 1ms per query, which is insignificant.

Compared with another quantizer ColBERT-OPQ, ColBERT-CQ can achieve the same level of space saving with $K = 256$ while having a substantial relevance improvement. ColBERT-CQ with $K = 4$ achieves the same level of relevance as ColBERT-OPQ while yielding a storage reduction of 67% and a latency reduction of about 70%. Comparing ColBERT-CQ with no contextual decomposition, under the same space cost, ColBERT-CQ's relevance is 4% higher. CQ with $K = 16$ achieves the same level relevance as ColBERT-CQ-undecomposed with $K = 256$, while the storage of CQ reduces by 44%. Comparing with ColBERT-small which adopts more aggressive space reduction, ColBERT-CQ with $K = 16$ would be competitive in relevance while its space is about 4x smaller.

Comparing with other non-ColBERT baselines (BECR, PreTTR, and TILDEv2), ColBERT-CQ strikes a good balance across relevance, space and latency. For the fast CPU based model (BECR, TILDEv2), our model achieves better relevance with either lower or comparable space usage. For BECR, its embedding footprint with 89.9GB may fit in memory for MS MARCO passages, it becomes very expensive to configure a machine with much more memory for BECR's MS MARCO document embeddings with about 791GB.

4.4 Design Options for CQ

Table 6 shows the relevance scores for the TREC deep learning passage ranking task with different design options for CQ. As an alternative setting, the codebooks in this table use M=16 and K=32 with compression ratio 21:1 compared to ColBERT. Row 1 is the default design configuration for CQ with product operators and 1 composition layer,

	TREC19	TREC20
CQ, Product, 1 layer, MarginMSE	0.687	0.713
Different model configurations		
No decomposition. Product	0.663†	0.686
No decomposition. Additive	0.656†	0.693
CQ, Product, 1 layer, raw static embedding	0.655†	0.683†
CQ, Additive, 1 layer	0.693	0.703
CQ, Product, 2 layers	0.683	0.707
CQ, Additive, 2 layers	0.688	0.703
Different training loss functions		
CQ, Product, 1 layer, MSE	0.679	0.704
CQ, Product, 1 layer, PairwiseCE	0.683	0.705

Table 6: NDCG@10 of different design options for CQ in TREC DL passage ranking. If the compression method result in statistically significant degradation from the default setting, '†' is marked.

and the MarginMSE loss function.

Different architecture or quantization options. Rows 2 and 3 of Table 6 denote CQ using product or additive operators without decomposing each embedding into two components, and there is about 4% degradation without such decomposition.

Row 4 changes CQ using the raw static embeddings of tokens from BERT instead of the upper layer outcome of BERT encoder and there is an up to 4.7% degradation. Notice such a strategy is used in SDR. From Row 5 to Row 7, we change CQ to use additive operators or use a two-layer composition. The performance of product or additive operators is in a similar level while the benefit of using two layers is relatively small.

Different training loss functions for CQ. Last two rows of Table 6 use the MSE and PairwiseCE loss functions, respectively. There is an about 1.2% improvement using MarginMSE. Figure 2 gives an explanation why MarginMSE is more effective. While CQ trained with MSE and MarginMSE generates ranking scores close to the original ranking scores in Figure 2(a), the distribution of Kendall's τ correlation coefficients of 1,000 passages in Figure 2(b) shows that the passage rank order derived by CQ with the MarginMSE loss has a better correlation with that by ColBERT.

5 Concluding Remarks

Our evaluation shows the effectiveness of CQ used for ColBERT in compressing the space of token embeddings with about 14:1 ratio while incurring a small relevance degradation in MS MARCO passage and document re-ranking tasks. The quantized token-based document embeddings for the tested cases can be hosted in memory for fast and high-throughput access. This is accomplished by a neural network that decomposes ranking contributions of contextual embeddings, and jointly trains context-aware decomposition and quantization with a loss function preserving ranking accuracy. The online time cost to decompress and recover embeddings is insignificant with 1ms for the tested cases. The CQ implementation is available at https://github.com/yingrui-yang/ContextualQuantizer.

Our CQ framework is also applicable to the contemporaneous work ColBERTv2 (Santhanam et al., 2021). Using uniCOIL scores for the first-stage sparse retrieval and ColBERTv2+CQ (M=16, K=256) for top 1,000 passage reranking, we achieve 0.387 MRR@10 on the MSMARCO passage Dev set, 0.746 NDCG@10 on TREC DL19, and 0.726 NDCG@10 on DL20 with about 10.2GB embedding space footprint. Notice that ColBERTv2 achieves a higher MRR@10 number 0.397 for the passage Dev set when used as a standalone retriever (Santhanam et al., 2021) and dense retrieval with such a multi-vector representation is likely to be much more expensive than retrieval with a sparse representation on a large dataset. The previous work in dense retrieval has often employed faster but approximate search, but that comes with a visible loss of relevance (Lewis et al., 2021). Thus the above relevance number using ColBERTv2+CQ for re-ranking with uniCOIL sparse retrieval is fairly strong, achievable with a reasonable latency and limited computing resource. Its embedding space size is 2.8x smaller than the 29GB space cost in the standalone ColBERTv2 (Santhanam et al., 2021) for MS MARCO passages. Our future work is to investigate the above issue further and study the use of CQ in the other late-interaction re-ranking methods.

Acknowledgments. We thank Cindy Zhao, Jiahua Wang, and anonymous referees for their valuable comments and/or help. This work is supported in part by NSF IIS-2040146 and by a Google faculty research award. It has used the Extreme Science and Engineering Discovery Environment supported by NSF ACI-1548562. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

Liefu Ai, Junqing Yu, Zenbin Wu, Yunfeng He, and Tao Guan. 2015. Optimized residual vector quantization for efficient approximate nearest neighbor search. *Multimedia Systems*, 23:169–181.

Daniel Fernando Campos, Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, Li Deng, and Bhaskar Mitra. 2016. Ms marco: A human generated machine reading comprehension dataset. *ArXiv*, abs/1611.09268.

Jiecao Chen, Liu Yang, Karthik Raman, Michael Bendersky, Jung-Jung Yeh, Yun Zhou, Marc Najork, D. Cai, and Ehsan Emadzadeh. 2020a. Dipair: Fast and accurate distillation for trillion-scale text matching and pair modeling. In *EMNLP*.

Xuanang Chen, B. He, Kai Hui, L. Sun, and Yingfei Sun. 2020b. Simplified tinybert: Knowledge distillation for document retrieval. *ArXiv*, abs/2009.07531.

Nachshon Cohen, Amit Portnoy, Besnik Fetahu, and Amir Ingber. 2021. Sdr: Efficient neural re-ranking using succinct document representation. *ArXiv*, 2110.02065.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Fernando Campos, and Ellen M. Voorhees. 2020. Overview of the trec 2020 deep learning track. *ArXiv*, abs/2102.07662.

Zhuyun Dai and J. Callan. 2019. Deeper text understanding for ir with contextual neural language modeling. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Zhuyun Dai and Jamie Callan. 2020. Context-aware term weighting for first stage passage retrieval. *SIGIR*.

Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. 2018. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *WSDM*, pages 126–134.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

Thibault Formal, C. Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2021a. Splade v2: Sparse lexical and expansion model for information retrieval. *ArXiv*, abs/2109.10086.

Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021b. *SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking*, pages 2288–2292. ACM.

Luyu Gao and Jamie Callan. 2021. Condenser: a pre-training architecture for dense retrieval. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 981–993, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Luyu Gao, Zhuyun Dai, and J. Callan. 2020. Understanding bert rankers under distillation. *Proceedings of SIGIR*.

Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021. COIL: revisit exact lexical match in information retrieval with contextualized inverted list. *NAACL*.

Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization for approximate nearest neighbor search. *CVPR*, pages 2946–2953.

J. Guo, Y. Fan, Qingyao Ai, and W. Croft. 2016. A deep relevance matching model for ad-hoc retrieval. *CIKM*.

Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. 2020a. Improving efficient neural ranking models with cross-architecture knowledge distillation. *ArXiv*, abs/2010.02666.

Sebastian Hofstätter, Hamed Zamani, Bhaskar Mitra, Nick Craswell, and A. Hanbury. 2020b. Local self-attention over long text for efficient document retrieval. *SIGIR*.

Sebastian Hofstätter, Markus Zlabinger, and A. Hanbury. 2020c. Interpretable & time-budget-constrained contextualization for re-ranking. In *ECAI*.

Eric Jang, Shixiang Shane Gu, and Ben Poole. 2017. Categorical reparameterization with gumbel-softmax. *ICLR*.

Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446.

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:117–128.

Shiyu Ji, Jinjin Shao, and Tao Yang. 2019. Efficient interaction-based neural ranking with locality sensitive hashing. In *WWW*.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*.

V. Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Yu Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. 2020. Dense passage retrieval for open-domain question answering. *ArXiv*, abs/2010.08191.

O. Khattab and M. Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. *SIGIR*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Patrick Lewis, Barlas Oğuz, Wenhan Xiong, Fabio Petroni, Wen tau Yih, and Sebastian Riedel. 2021. Boosted dense retriever.

Canjia Li, A. Yates, Sean MacAvaney, B. He, and Yingfei Sun. 2020. Parade: Passage representation aggregation for document reranking. *ArXiv*, abs/2008.09093.

Jimmy Lin, Rodrigo Nogueira, and A. Yates. 2020. Pretrained transformers for text ranking: Bert and beyond. *ArXiv*, abs/2010.06467.

Jimmy J. Lin and Xueguang Ma. 2021. A few brief notes on deepimpact, coil, and a conceptual framework for information retrieval techniques. *ArXiv*, abs/2106.14807.

Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy J. Lin. 2021. In-batch negatives for knowledge distillation with tightly-coupled teachers for dense retrieval. In *REPL4NLP*.

Sean MacAvaney, F. Nardini, R. Perego, N. Tonellotto, Nazli Goharian, and O. Frieder. 2020. Efficient document re-ranking for transformers by precomputing term representations. *SIGIR*.

Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. 2019. Cedr: Contextualized embeddings for document ranking. *SIGIR*.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. 2017. The concrete distribution: A continuous relaxation of discrete random variables. *ICLR*.

Antonio Mallia, O. Khattab, Nicola Tonellotto, and Torsten Suel. 2021. Learning passage impacts for inverted indexes. *SIGIR*.

Julieta Martinez, Shobhit Zakhmi, Holger H. Hoos, and J. Little. 2018. Lsq++: Lower running time and higher recall in multi-codebook quantization. In *ECCV*.

Bhaskar Mitra, Sebastian Hofstätter, Hamed Zamani, and Nick Craswell. 2021. Conformer-kernel with query term independence for document retrieval. *SIGIR*.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with bert. *ArXiv*, abs/1901.04085.

Rodrigo Nogueira, W. Yang, Kyunghyun Cho, and Jimmy Lin. 2019a. Multi-stage document ranking with bert. *ArXiv*, abs/1910.14424.

Rodrigo Nogueira, Wei Yang, Jimmy J. Lin, and Kyunghyun Cho. 2019b. Document expansion by query prediction. *ArXiv*, abs/1904.08375.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP/IJCNLP*.

Ruiyang Ren, Yingqi Qu, Jing Liu, Wayne Xin Zhao, QiaoQiao She, Hua Wu, Haifeng Wang, and Ji-Rong Wen. 2021. RocketQAv2: A joint training method for dense passage retrieval and passage re-ranking. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2825–2835, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Stephen E. Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3:333–389.

Keshav Santhanam, O. Khattab, Jon Saad-Falcon, Christopher Potts, and Matei A. Zaharia. 2021. Colbertv2: Effective and efficient retrieval via lightweight late interaction. *ArXiv*, abs/2112.01488.

Raphael Shu and Hideki Nakayama. 2018. Compressing word embeddings via deep compositional code learning. *ICLR*.

Y. Wu, M. Schuster, Z. Chen, Q. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144.

J. Xin, Rodrigo Nogueira, Y. Yu, and Jimmy Lin. 2020. Early exiting bert for efficient document ranking. In *SUSTAINLP*.

Chenyan Xiong, Zhuyun Dai, J. Callan, Zhiyuan Liu, and R. Power. 2017. End-to-end neural ad-hoc ranking with kernel pooling. *SIGIR*.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *International Conference on Learning Representations*.

Wei Yang, Haotian Zhang, and Jimmy Lin. 2019. Simple applications of bert for ad hoc document retrieval. *ArXiv*, abs/1903.10972.

Yingrui Yang, Yifan Qiao, Jinjin Shao, Xifeng Yan, and Tao Yang. 2022. Lightweight composite re-ranking for efficient keyword search with BERT. *WSDM*.

Jingtao Zhan, J. Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. 2020. Learning to retrieve: How to train a dense retrieval model effectively and efficiently. *ArXiv*, abs/2010.10469.

Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2021a. Jointly optimizing query encoder and product quantization to improve retrieval performance. *CIKM*.

Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2021b. Optimizing dense retrieval model training with hard negatives. *CoRR*, abs/2104.08051.

Y. Zhang, Ping Nie, Xiubo Geng, A. Ramamurthy, L. Song, and Daxin Jiang. 2020. Dc-bert: Decoupling question and document for efficient contextual encoding. In *SIGIR*.

Shengyao Zhuang and G. Zuccon. 2021. Fast passage reranking with contextualized exact term matching and efficient passage expansion. *ArXiv*, abs/2108.08513.

A Details on Retrieval Choices, Numbers Cited, and Model Implementations

First-stage retrieval models considered. To retrieve top results before re-ranking, we have considered the recent work in sparse and dense retrieval that outperforms BM25. For sparse retrieval with inverted indices, DeepCT (Dai and Callan, 2020) uses deep learning to assign more sophisticated term weights for soft matching. The docT5query work (Nogueira et al., 2019b) uses a neural model to pre-process and expand documents. The recent work on sparse representations includes DeepImpact (Mallia et al., 2021), uniCOIL (Lin and Ma, 2021; Gao et al., 2021), and SPLADE (Formal et al., 2021b,a), for learning neural contextualized term weights with document expansion. Instead of using a sparse inverted index, an alternative retrieval method is to use a dense representation of each document, e.g. (Lin et al., 2021; Zhan et al., 2021a; Xiong et al., 2021; Gao and Callan, 2021; Zhan et al., 2021b; Ren et al., 2021). We use BM25 because it is a standard reference point. We have also used uniCOIL for passage re-ranking because a uniCOIL-based sparse retriever is fairly efficient and its tested relevance result is comparable to that of the end-to-end ColBERT as a dense retriever and other learned sparse representations mentioned above. Certainly CQ is applicable for re-ranking with any of dense or sparse retrievers or their hybrid combination.

Model numbers cited from other papers. As marked in Tables 3 and 4, for DeepCT, JPQ and TILDEv2, we copy the relevance numbers reported in their papers. For TCT-ColBERT(v2), DeepImpact and uniCOIL, we obtain their performance using the released checkpoints of Pyserini [1]. For PreTTR (MacAvaney et al., 2020) on the passage task and BERT-base on the document task, we cite the relevance performance reported in Hofstätter et al. (2020a). There are two reasons to list the relevance numbers from other papers. One reason is that for some chosen algorithms, the running of our implementation version or their code delivers a performance lower than what has been reported in the authors' original papers, perhaps due to the difference in training setup. Thus, we think it is fairer to report the results from the authors' papers. Another reason is that for some algorithms, the authors did not release code and we do not have implementations.

In storage space estimation of Table 5, for BECR, we use the default 128 bit LSH footprint with 5 layers. For PreTTR we uses 3 layers with dimension 768 and two bytes per number following Hofstätter et al. (2020a). For TILDEv2, we directly cite the space cost from its paper.

Model implementation and training. For baseline model parameters, we use the recommended set of parameters from the authors' original papers. For ColBERT, we use the default version that the authors selected for fair comparison. The ColBERT code follows the original version released [2] and BERT implementation is from Huggingface [3]. For BERT-base and ColBERT, training uses pairwise softmax cross-entropy loss over the released or derived triples in a form of $(\mathbf{q}, \mathbf{d}^+, \mathbf{d}^-)$ for the MS MARCO passage task. For the MS MARCO document re-ranking task, we split each positive long document into segments with 400 tokens each and transfer the positive label of such a document to each divided segment. The negative samples are obtained using the BM25 top 100 negative documents. The above way we select training triples for document re-ranking may be less ideal and can deserve an improvement in the future.

When training ColBERT, we use gradient accumulation and perform batch propagation every 32 training triplets. All models are trained using Adam optimizer (Kingma and Ba, 2015). The learning rate is 3e-6 for ColBERT and 2e-5 for BERT-base following the setup in its original paper. For ColBERT on the document dataset, we obtained the model checkpoint from the authors.

Our CQ implementation leverages the open source code [4] for Shu and Nakayama (2018). For PQ, OPQ, RQ, and LSQ, we uses off-the-shelf implementation from Facebook's faiss[5] library (Johnson et al., 2017). To get training instances for each quantizer, we generate the contextual embeddings of randomly-selected 500,000 tokens from passages or documents using ColBERT.

When using the MSE loss, learning rate is 0.0001, batch size is 128, and the number of training epochs is 200,000. When fine-tuning with PairwiseCE or MarginMSE, we freeze the encoder based on the MSE loss, set the learning rate to be 3e-6, and then train for additional 800 batch iterations with 32 training pairs per batch.

[1] https://github.com/castorini/pyserini/

[2] https://github.com/stanford-futuredata/ColBERT

[3] https://huggingface.co/transformers/model_doc/bert.html

[4] github.com/mingu600/compositional_code_learning.git

[5] https://github.com/facebookresearch/faiss