信息检索前沿研究
论文阅读报告

方言
2021210929

2022 年 6 月 12 日

1 涉及论文概要

此次论文阅读报告选择向量检索和索引压缩领域的两篇论文，论文标题分别是《Jointly Optimizing Query Encoder and Product Quantization to Improve Retrieval Performance》和《Distill-VQ: Learning Retrieval Oriented Vector Quantization By Distilling Knowledge from Dense Embeddings》(以下将用 JPQ 和 Dist-VQ 指代)。由于这两篇论文要研究的问题，背景和相关工作都十分接近，采用的方法上也有诸多共同点，因此接下来将统一介绍这两篇论文的各个部分，并在其中对二者进行分析和比较。

2 背景与摘要

近年来，大规模预训练语言模型的快速进步促进了信息检索领域的发展，其中，向量检索技术在一阶段检索问题上取得了非常好的效果。向量检索技术通过构建嵌入向量分别表示查询和文档，并且通过向量相似度来衡量相关性。然而，尽管向量检索技术有着很出色的检索性能，嵌入向量需要大量的存储空间，并且暴力遍历搜索也带来了巨大的时间开销，这都阻碍和限制了向量检索技术在实际场景中的使用。

因此，近似最近邻搜索技术在向量检索的实践中起到了很大的作用，其中向量压缩是最主要的一部分。现有的向量压缩方法能够提升检索模型的存储和运行效率，但是却对实际检索的性能有极大损害。这是因为这些方法使用与任务无关的向量重建误差作为损失函数进行训练，而且编码器模型和压缩索引是分开训练的，因此可能并不能很好地配合在一起。近期一些工作开始将编码器模型和压缩索引联合训练，但是这些方法并不是针对信息检索领域涉及的，也没有考虑到向量检索的一些实际特点，比如负例挖掘的重要性。

在 JPQ 的文章中，作者提出了一种联合优化的方法，具体而言有几个创新点:

1. 使用排序导向的损失函数（而非向量重建误差）来端到端地联合训练编码器模型和压缩索引。

2. 提出三种训练策略，分别是排序导向的损失函数，PQ 中心嵌入向量优化，端到端的负例挖掘，能够有效提升性能。

进一步地，在 Dist-VQ 文章中，作者对上述的联合优化过程做了更细致的分类。

A. 第一类是使用排序损失函数训练编码器模型，再使用重建误差损失函数训练压缩索引，两个过程是独立的，传统的方法例如 PQ 和 OPQ 都属于这一类。

B. 第二类是使用排序损失函数训练编码器模型，再使用重建误差损失函数训练压缩索引，但两个过程是联合优化的，在 JPQ 作者设计的一个变种方法 DPQ 属于这一类。

C. 第三类是使用排序损失函数联合训练编码器模型和压缩索引，JPQ 以及后续的 RepCONC 都属于这一类。

D. 第四类是使用排序损失函数训练编码器模型，再通过知识蒸馏的方法训练压缩索引，这也是 Dist-VQ 文章提出的方法。

Dist-VQ 作者指出，虽然 JPQ 一类的方法能够有效地提高检索的效率和性能，但是训练过程依赖于大量的标注数据。为了解决这个问题，他们提出了基于知识蒸馏的方法，在优化压缩索引的阶段可以不需要标注数据，因此这种方法也能更好地和一些其他数据集充分训练的编码器模型直接结合使用。

3 相关工作

这一部分介绍一下两篇文章涉及的相关工作，主要可以分成向量检索模型，向量压缩优化方法两个部分，另外在 Dist-VQ 文章中还有一部分介绍知识蒸馏工作。

3.1 向量检索模型

向量检索模型主要是双塔结构，分为查询编码器和文档编码器。两个编码器分别得到查询的表示向量和文档的表示向量，之后使用向量相似度衡量。双塔结构的模型通常都使用负例采样策略进行训练。一些方法使用全部的非相关文档作为负例，如 RocketQA；一些方法使用 BM25 召回结果作为难负例，比如 RepBERT；最近，Zhan 等人提出了动态难负例采样，表明了其能提升头部文档的排序性能。

3.2 向量压缩优化方法

按照优化目标，向量压缩优化方法可以分为以下两种:

3.2.1 最小化重建误差损失

早期的向量压缩优化方法采用这种方式进行训练，即先对原始向量进行压缩，再通过压缩后的向量进行重建，通过最小化重建向量和原始向量的 L2 距离，可以使得压缩后的向量尽可能多地保存信息。传统的方法通常与固定的编码器模型相结合，并且依赖于无监督的方法，例如 k-means 等聚类或层次聚类的方法。PQ，LSH 等方法都是通过最小化重建误差进行训练的。为了进一步提高性能，人们提出了 OPQ 方法，通过在 PQ 之前添加一个可学习的线性变化过程，可以有效地降低重建误差。随后，一些研究提出了将压缩索引和编码器模型端到端联合训练，这样能够更好地优化重建误差。

3.2.2 最小化排序损失

上述方法存在一个主要的问题是，其只依赖于文档编码器和文档的嵌入向量表示，而没有考虑到文档和查询之间的关系。此外，一些工作也表明，仅仅优化重建误差并不一定能够提升排序的性能。因此，近期的一些工作，例如 MoPQ，JPQ，RepCONC，通过对比学习的损失函数来联合训练压缩索引和查询编码器模型，由此可以直接优化排序性能。与前者对比，这种方法在保持压缩的基础上，显著提高了排序性能。

3.3 知识蒸馏

在 Dist-VQ 文章中介绍了知识蒸馏。知识蒸馏技术是一种有效的模型压缩方法，通常使用一个轻量级的小模型去模仿重量级的大模型的输出，从而实现模型上的压缩。近年来，一些工作通过使用二阶段的重排序模型输出的分数去蒸馏得到一阶段的召回模型；还有一些工作通过伪标签蒸馏进行预训练。知识蒸馏技术能有效利用大量的无标注数据，通过表示学习，在检索性能上变得越来越有效。

4 模型和方法

这一部分将会介绍两篇文章提出的方法，由于有很多数学证明，因此只简单介绍基本思想和实现的方法，以及证明的结论。

4.1 JPQ 方法

图 1 展示了 JPQ 方法的主要框架。JPQ 方法的流程主要分成以下三个部分:

- **排序导向的损失函数:** JPQ 使用双塔模型以及 PQ 压缩索引在实际检索过程中产生的损失作为损失函数，因此成为排序导向的损失函数。为了计算这个损失函数，JPQ 会先通过 PQ 的中心向量和索引分配，重建出一个量化的文档嵌入向量，再用这个重建向量与查询编码器输出的查询嵌入向量计算相关性分数，最终将这个分数组装成一个配对形式的损失函数。由于这种损失函数引入了真实计算中的相关性分数，因此可以准确地评估排序性能，由此可以使用梯度下降的方法对其进行优化。
- **PQ 中心向量优化:** PQ 由中心向量和索引分配两部分组成。直接优化索引分配有两个困难，一是排序导向的损失函数对索引分配部分是不可微分的，由此无法使用梯度下降方法对其优化；二是即使使用一些近似的方法能够对其优化，由于索引分配的数量很大，也会出现过拟合问题。为了解决这两个问题，JPQ 模型使用无监督的 PQ 对索引分配进行初始化，之后固定索引分配部分，只对中心向量进行优化。这里 JPQ 作者对中心向量的微分和梯度更新公式进行数学求解，在此不赘述，结论是可以直接使用梯度下降法对 PQ 中心向量进行更新。
- **端到端负例采样:** 在排序导向的损失函数的基础上，JPQ 引入了端到端的负例采样去进一步提高排序性能。很多工作都表明了负例采样在双塔模型的训练中是很重要的。JPQ 的目标是惩罚那些被排序在头部的不相关文档，具体而言，借助 PQ 索引，可以很轻松的在文档集合中搜索出头部的文档，并且将其作为强负例引入训练中。

4.2 Dist-VQ 方法

图 2 展示了 Dist-VQ 方法的主要框架。Dist-VQ 方法的流程主要分成以下两个部分:

- **量化和重建:** 这部分与 JPQ 类似，主要通过压缩索引对原始的文档嵌入向量进行压缩，再进行重建。不同的是，JPQ 方法只使用了 PQ 索引，而 Dist-VQ 使用了一种组合索引 IVFPQ。IVFPQ 索引先通过 IVF（倒排索引表）部分进行聚类和倒排，再将每个嵌入向量相对其聚类中心的残差向量进行 PQ 索引压缩。相应的，在重建过程中，也需要引入 IVF 的中心向量进行重建。同时，由于相同的原因，与 JPQ 方法一样，Dist-VQ 也固定住 PQ 索引的索引分配部分，只对 PQ 的中心向量和 IVF 的中心向量进行优化。

- **知识蒸馏:** Dist-VQ 将原始的查询和文档的嵌入向量作为教师模型，量化重建后的表示作为学生模型，让学生模型的输出尽可能接近教师模型的输出，不同于 JPQ 方法的是，这样 Dist-VQ 在训练压缩索引的时候就需要有标注的数据，而只需要一组已经训好的嵌入向量表示（后者训好的编码器模型）。然而这样的步骤需要考虑两个问题，一是使用什么样的相似度函数去衡量二者输出的一致性程度，二是在哪些数据上去进行知识蒸馏。Dist-VQ 因此在相似度函数和采样策略两个方面进行了大量的探索实验。

- **相似度函数:** 学生模型可以从两个角度保留模仿教师模型的输出，第一种是保持分数不变性，即让学生模型对于给定的查询-文档组合，输出的分数与教师模型输出分数保持一致。MSE 损失函数和 MarginMSE 损失函数属于这一类。第二种是保持排序不变性，即对于一个查询和两个文档，只要求学生模型对两个文档的排序与教师模型保持一致，这实际上是对分数不变性的一种放松。Dist-VQ 作者尝试了 RankNet，KL 散度和 ListNet 损失函数。

- **采样策略:** Dist-VQ 选用了几种不同的采样策略。一是与 JPQ 一样，采样 Top-K 的不相关文档作为难负例，二是使用同批次的负例，即在训练时使用同一个批次内，其他查询对应的文档，作为当前查询的负例，这实际上是一种接近于随机负例的采样方法，但是可以很好地节约实际训练的显存开销。

5 实验设定

两篇文章在实验数据集选定上保持一致，均使用了 MSMarco-Passage Retrieval 数据集进行实验。这个数据集有大约 880 万的文档集合，50 万的训练查询以及标注，7000 个验证集查询和 43 个测试集查询。对于每一种方法，均汇报在整个数据集上进行召回的结果的 MRR@10 指标和 Recall@100 指标。

在基线模型上，由于 JPQ 模型强调的是联合优化压缩，因此其主要对比的方法是未进行压缩的模型和只通过无监督方法压缩的模型。因此其选择了如下几种基线模型:

- **传统模型:**BM25 模型，以及一些改进的方法，如 docT5query，DeepCT 等

- **未压缩的模型:** 这一类基线模型包括 ANCE，STAR，ADORE+STAR，以及多向量表示模型 ColBERT

- **压缩模型:** 这一类基线模型包括无监督的压缩方法 PQ，OPQ，以及有监督训练的变种压缩方法 DPQ。

对于 Dist-VQ 模型，其强调通过知识蒸馏来进行联合训练，因此其对比的方法主要是其他的联合优化方法。其中包括无监督的方法 IVFPQ，IVFOPQ，ScaNN，和有监督的方法 Poemm，JPQ，RepCONC。

6 实验结果和讨论

图 3，图 4，图 5 是 JPQ 的实验结果，接下来将从以下两个角度比较分析 JPQ 的性能:

- **与其他压缩方法比较:** 从图 3 中可以看出，在全部压缩 30 倍的基础上，JPQ 的排序性能以很大的优势超过了其他的压缩方法。在其他的压缩方法之中，无监督的方法 PQ 和 OPQ 会严重损害排序性能，DPQ 也进行了联合优化并且利用了监督信号的信息，因此其性能也超过了其他无监督的基线压缩方法。但是 JPQ 相比 DPQ 也有很大提升，这样的提升是由于 DPQ 使用了重建误差作为损失函数，并且也没有相应的端到端负例采样策略。而 JPQ 方法借助三个训练策略能够直接优化排序性能，因此大幅度超越了其他基线模型。

- **与未压缩的模型比较:** 从图 4 和图 5 中可以看出，在整体性能上，向量检索模型超越了如 BM25 等的传统的方法。对比未压缩的向量检索模型，JPQ 能达到接近甚至超越的性能，并且在索引大小上有 30 倍的压缩。ColBERT 模型的性能超越了 JPQ，但作者认为其是由于多向量表示带来了巨大的性能提升，并且其索引大小是 JPQ 的 186 倍以上。

此外，为了分析 JPQ 的三种训练策略对于模型性能的影响，进行了消融实验。图 6 是消融实验的结果，从中可以看出每一个策略都对整体模型有所贡献，并且在压缩倍数比较高的时候，排序导向的损失函数可以带来非常明显的提升。

图 7，图 8 是 Dist-VQ 的实验结果:

- **整体比较:**Dist-VQ 分别用 AR2-G 和 CoCondenser 进行初始化，在 MRR@10 上分别达到了 0.3607 和 0.3523，分别比最强的基线模型高出 +1.36% 和 +1.53%。并且 Dist-VQ 引入了 IVF，这个性能评估是在 IVF 搜索了 1% 的倒排列表的基础上的，这意味着比之前的模型有 100 倍的效率加速。另外，作者发现，在用一些比较强的模型进行初始化的时候，之前的联合优化方法（包括 JPQ 和 RepCONC）相比于无监督的 IVFOPQ 方法并没有非常显著的优势。这可能是由于比较强的初始化模型已经充分利用了监督信号的信息，因此只需要在压缩时保留信息就能得到不错的排序性能，而不需要针对排序任务利用监督信号继续训练了。

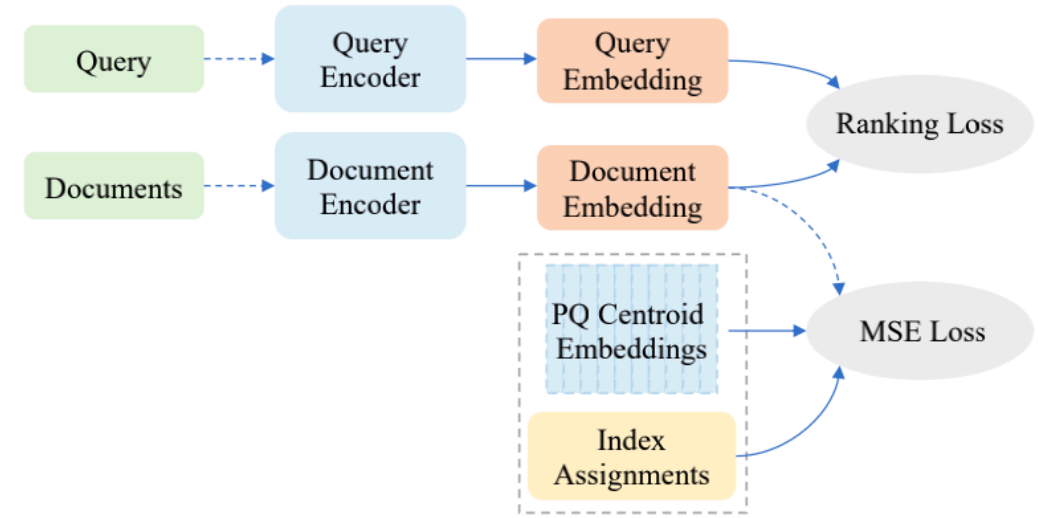
- **相似度函数分析:** 从图 8 的上半部分看出，排序不变性的相似度函数整体优于分数不变性的相似度函数，这个结论也比较符合直觉，因为不同的模型输出分数的幅度不同，分数之间可能并不可比。但是要求排序保持一致是一个相对更容易的任务，也更加符合检索场景的本质。在这些相似度函数之中，ListNet 的效果是最好的，这可能是因为其公式的形式对于模型输出分数的方差限制要求更低一些。

- **采样策略分析:** 从图 8 的下半部分看出，排序头部的难负例对于模型训练而言非常重要，所有使用了 Top-K 策略的变种都取得了比较明显的性能提升。此外，仅仅有排序头部的负例是不够的，还需要引入一些排序位置靠后的弱负例，这一部分可以通过批次内采样简单实现。因此，最好的采样策略是批次内采样和 Top-K 采样混合的形式。另外，值得注意的是，这样的采样策略可以在完全未标注的数据上进行，因此这样的蒸馏方法可以不需要标注数据。

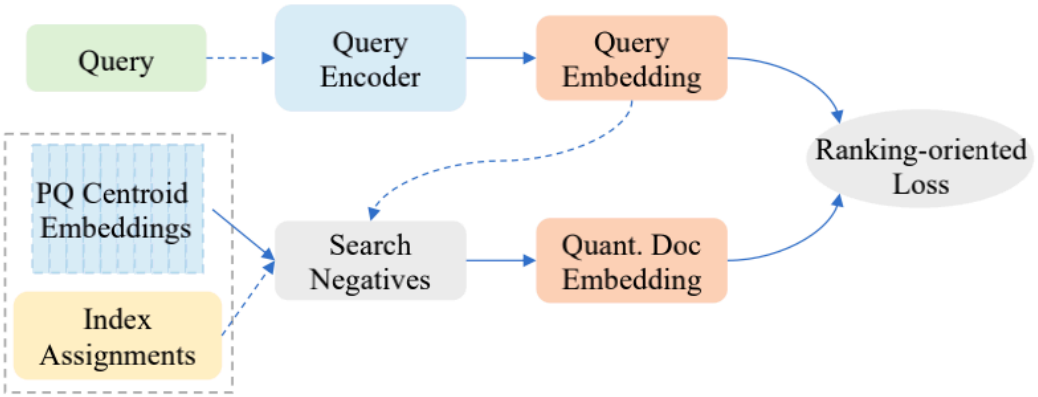
7 总结

本篇阅读报告主要分析了两篇同领域的论文，它们都是试图在保持排序性能的基础上试图对向量表示进行压缩。不同点在于，JPQ 方法将编码器模型和压缩索引融入到一个联合优化的框架内，端到端地利用监督信号直接优化排序性能。而 Dist-VQ 则考虑了更严格的条件，即可能无法使用监督信号，并由此通过知识蒸馏的方式解决这个问题。但同样也会受到一定的限制，即模型在压缩后的排序性能理论上是很难超过原始未压缩的模型的，而 JPQ 方法由于有额外的端到端训练过程，其性能是有可能超过未压缩的原始模型的。

A 附录



(a) Workflow of existing training methods.



(b) Workflow of JPQ.

Figure 2: Training workflows. Solid arrows indicate the gradients are backpropagated, whereas the dotted arrows indicate otherwise.

图 1: JPQ 整体框架

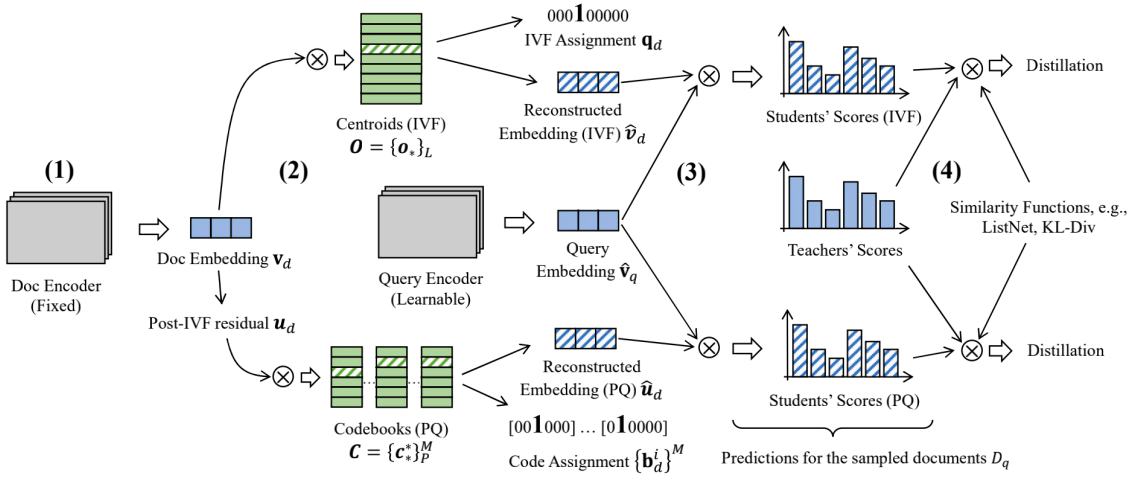


Figure 2: Distill-VQ. (1) The Doc Encoder (well-trained and fixed) infers the embeddings for all documents in the offline stage. (2) Each document embedding is assigned to IVF and PQ (the assigned entries are stripped); then, the embedding is reconstructed w.r.t. the assignments of IVF and PQ, respectively. (3) The Query Encoder infers the query embedding, which will interact (inner-product in our work) with the reconstructed embeddings for the relevance scores with the document. For each query: the relevance scores are computed for all the sampled documents. (4) The knowledge distillation is performed w.r.t. the relevance scores from teachers (computed in the offline stage).

图 2: Dist-VQ 整体框架

Table 1: Comparison with different ANNS methods on TREC 2019 Deep Learning Track. */ denotes that JPQ performs significantly better than baselines at $p < 0.05/0.01$ level using the two-tailed pairwise t-test.**

Model	Index	MARCO Passage		DL Passage		Index	MARCO Doc		DL Doc
	GB	MRR@10	R@100	NDCG@10	R@100	GB	MRR@100	R@100	NDCG@10
Non-exhaust. ANNS									
Annoy [6]	30.96	0.150**	0.274**	0.431**	0.157**	11.36	0.304**	0.638**	0.553**
FALCONN [1]	>25.34[a]	0.297**	0.825**	0.567**	0.411**	>9.20[a]	0.339**	0.869**	0.541**
FLANN [36]	32.85	0.319**	0.839**	0.607**	0.430**	10.60	0.366**	0.895**	0.593
IMI [3]	25.39	0.331**	0.828**	0.610*	0.431*	9.24	0.376**	0.869**	0.590
HNSW [34]	25.76	0.334*	0.848**	0.624	0.454	9.35	0.378**	0.879**	0.588*
Compressed ANNS									
PQ [25]	0.79	0.123**	0.527**	0.323**	0.213**	0.29	0.152**	0.544**	0.273**
LSH [26]	0.79	0.302**	0.824**	0.578**	0.417**	0.29	0.351**	0.882**	0.576**
ITQ+LSH [19]	0.80	0.296**	0.825**	0.582**	0.415**	0.29	0.347**	0.874**	0.570**
ScaNN [20]	0.79	0.288**	0.818**	0.555**	0.386**	0.29	0.335**	0.866**	0.534**
HNSW+OPQ [26]	0.95	0.309**	0.818**	0.596**	0.424**	0.35	0.357**	0.876**	0.543**
OPQ [18]	0.83	0.305**	0.839**	0.594**	0.435**	0.30	0.361**	0.892**	0.588**
OPQ+ScaNN	0.83	0.313**	0.843**	0.614**	0.442*	0.30	0.361**	0.897**	0.583**
DPQ [11, 49]	0.83	0.311**	0.848**	0.601**	0.453	0.30	0.367**	0.899**	0.585**
Ours									
JPQ	0.83	**0.341**	**0.868**	**0.677**	**0.466**	0.30	**0.401**	**0.914**	**0.623**

[a] FALCONN does not support saving index to disk and it is hard to infer the exact index size at run time.

图 3: JPQ 实验结果

Table 3: Comparison with BoW models on TREC 2019 Deep Learning Track. */ denotes that JPQ performs significantly better than baselines at $p < 0.05/0.01$ level using the two-tailed pairwise t-test.**

Model	Index	MARCO Passage		DL Passage		Index	MARCO Doc		DL Doc
	GB	MRR@10	R@100	NDCG@10	R@100	GB	MRR@100	R@100	NDCG@10
Traditional BoW									
BM25 [44]	0.59	0.187**	0.670**	0.497**	0.460	2.17	0.278**	0.807**	0.523**
Augmented BoW									
doc2query [39]	0.65	0.215**	0.713**	0.533**	0.471	n.a.	n.a.	n.a.	n.a.
DeepCT [13]	0.48	0.242**	0.754**	0.569**	0.455	n.a.	n.a.	n.a.	n.a.
HDCT [14]	n.a.	n.a.	n.a.	n.a.	n.a.	1.71	0.319**	0.843**	n.a.
docT5query [38]	0.96	0.272**	0.819**	0.642	**0.514**	2.31	0.327**	0.861**	0.597
Ours									
JPQ	0.83	**0.341**	**0.868**	**0.677**	0.466	0.30	**0.401**	**0.914**	**0.623**

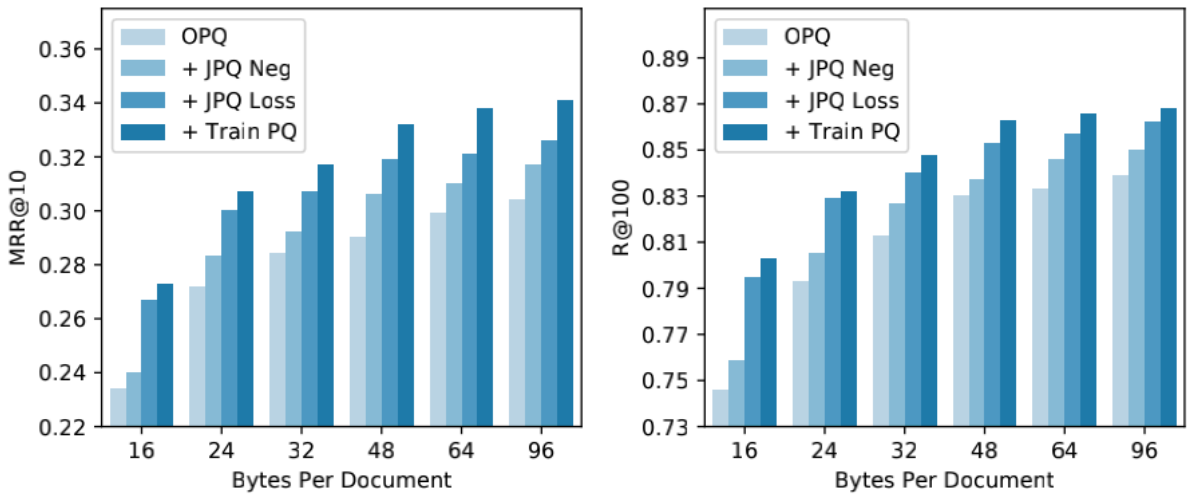
图 4: JPQ 实验结果

Table 5: Comparison with existing DR models and late-interaction models on TREC 2019 Deep Learning Track.

Model	Index	MARCO Passage		DL Passage		Index	MARCO Doc		DL Doc
	GB	MRR@10	R@100	NDCG@10	R@100	GB	MRR@100	R@100	NDCG@10
JPQ	0.83	0.341	0.868	0.677	0.466	0.30	0.401	0.914	0.623
Brute-force DR									
Rand Neg [23]	30x	−12%	−2%	−10%	−1%	30x	−18%	−6%	−8%
BM25 Neg [17]	30x	−9%	−6%	−10%	−22%	30x	−21%	−13%	−13%
ANCE FirstP [43]	30x	−1%	−1%	−4%	−5%	30x	−6%	−2%	−2%
ANCE MaxP [43]	30x	−1%	−1%	−4%	−5%	196x	−5%	−1%	+2%
TCT-ColBERT [30]	30x	−2%	−1%	−1%	−2%	196x	−17%	−5%	−2%
STAR [46]	30x	0%	0%	−5%	0%	30x	−3%	0%	−3%
ADORE+STAR [46]	30x	+2%	+1%	+1%	+2%	30x	+1%	+1%	+1%
Late-Interaction									
ColBERT [28]	186x	+6%[a]	+2%[a]	n.a.	n.a.	5833x[b]	+10%[b]	+1%[b]	n.a.

[a] We include MRR@10 reported in the paper and consult the authors about R@100 due to lack of open-sourced ColBERT checkpoint,
[b] We consult the authors about ColBERT's performance on document ranking task since it is not included in the original paper.

图 5: JPQ 实验结果



(a) MRR@10 **(b) Recall@100**

Figure 7: The ablation study of JPQ on MARCO Passage dataset.

图 6: JPQ 实验结果

		MS MARCO Passage (Dev)				Natural Questions (Test)			
Encoder	**Method**	**MRR@10**	**R@10**	**R@50**	**R@100**	**MRR@10**	**R@10**	**R@50**	**R@100**
AR2-G [49]	IVFPQ	0.2403	0.4545	0.6615	0.7325	0.2823	0.5002	0.6764	0.7240
	IVFOPQ	0.3448	0.6013	0.7881	0.8385	0.5505	0.7260	0.8047	0.8279
	ScaNN	0.2670	0.4880	0.6953	0.7963	0.4137	0.6470	0.7742	0.8124
	Poeem	0.3448	0.6077	0.7975	0.8487	0.5588	0.7443	0.8246	0.8445
	JPQ	0.3451	0.6107	0.8028	0.8558	0.5666	0.7512	0.8282	0.8504
	RepCONC	0.3449	0.6106	0.7983	0.8497	0.5500	0.7382	0.8141	0.8373
	Contrast	0.3471	0.6119	0.8037	0.8564	0.5835	0.7634	0.8412	0.8606
	Distill-VQ	**0.3607**	**0.6276**	**0.8221**	**0.8719**	**0.6235**	**0.7927**	**0.8627**	**0.8783**
CoCondenser [10]	IVFPQ	0.2252	0.4380	0.6509	0.7202	0.3871	0.6177	0.7642	0.8047
	IVFOPQ	0.3340	0.5947	0.7810	0.8325	0.5447	0.7362	0.8385	0.8626
	ScaNN	0.2470	0.4602	0.6715	0.7442	0.4520	0.6540	0.7831	0.8193
	Poeem	0.3363	0.6026	0.7908	0.8403	0.5522	0.7476	0.8354	0.8573
	JPQ	0.3360	0.5954	0.7810	0.8335	0.5436	0.7448	0.8335	0.8551
	RepCONC	0.3319	0.5895	0.7798	0.8342	0.5544	0.7495	0.8356	0.8602
	Contrast-VQ	0.3370	0.6056	0.8019	0.8542	0.5689	0.7549	0.8395	0.8627
	Distill-VQ	**0.3523**	**0.6212**	**0.8086**	**0.8612**	**0.5781**	**0.7654**	**0.8543**	**0.8762**

图 7: Dist-VQ 实验结果

	MS MARCO Passage (Dev)				Natural Question (Test)			
Method	**MRR@10**	**R@10**	**R@50**	**R@100**	**MRR@10**	**R@10**	**R@50**	**R@100**
MSE	0.3518	0.6158	0.7981	0.8457	0.5725	0.7545	0.8313	0.8520
Margin-MSE	0.3592	0.6207	0.8107	0.8595	0.5869	0.7573	0.8324	0.8504
RankNet	0.3519	0.6156	0.8044	0.8560	0.6052	0.7844	0.8623	0.8764
KL-Div	**0.3633**	0.6239	0.8153	0.8612	0.5846	0.7576	0.8315	0.8506
ListNet	0.3607	**0.6276**	**0.8221**	**0.8719**	**0.6235**	**0.7927**	**0.8627**	**0.8783**
GT+IB	0.3577	0.6240	0.8194	0.8710	0.6224	0.7910	0.8600	0.8703
GT+IB+Top-200^1	0.3607	0.6276	0.8221	0.8719	0.6235	0.7927	0.8627	0.8783
IB (*)	0.3543	0.6223	0.8115	0.8608	0.5992	0.7731	0.8457	0.8642
Top-3^1+Top-200^1 (*)	0.3595	0.6215	0.8084	0.8541	0.5941	0.7603	0.8360	0.8545
IB+Top-200^2 (*)	0.3598	0.6237	0.8126	0.8616	0.6186	0.7883	0.8479	0.8662
IB+Top-3^1+Top-200^1 (*)	0.3603	0.6265	**0.8221**	0.8727	0.6228	0.7915	0.8624	0.8786
IB+Top-10 (*)	0.3614	0.6285	0.8201	0.8698	0.6130	0.7853	0.8581	0.8753
IB+Top-100 (*)	0.3655	**0.6312**	0.8210	0.8690	0.6271	0.7933	0.8637	0.8800
IB+Top-200 (*)	**0.3656**	0.6298	0.8205	**0.8871**	**0.6295**	**0.7969**	**0.8675**	**0.8806**

Table 3: Knowledge distillation analysis. Upper: impact from similarity functions. Lower: impact from sampled documents (GT: ground-truth; IB: in-batch sampling; Top-K^1/Top-K^2: sampling 1/2 document from Top-K; Top-K: using all Top-K documents; (*): w.o. ground-truth.)

图 8: Dist-VQ 实验结果