作者：Alex
链接：https://www.zhihu.com/question/510987022/answer/3587373048
来源：知乎

文本召回是信息检索领域非常重要的一个模块。信息检索的通常由 retriever 和 re-ranker 构成，本文专注的内容在 retriever 环节。更具体的，文本所讲的召回是指向量化召回。文本召回，如 BM25，虽然被广泛使用，但其在复杂语义理解、上下文等场景的固有缺陷，已经逐步被向量化召回模型所替代。
本篇是文本召回的上篇，主要讨论以编码器结构做 dense retrieval，以解码器为基础结构的分支少量涉及，有兴趣的同学可以阅读下篇，敬请期待 :)

句子向量编码

1 先验知识

1.1 文本召回的演化历程

文本召回分为 2 个方向：字面量召回（sparse retrieval 或 lexical retrieva）和向量化召回（dense retrieval）。

1.1.1 字面量召回

字面量文本召回是一种基于字面量、使用词袋（Bag of Words, BoW）模型的文本匹配算法。在这种方法中，文档被表示为由词项和它们的权重组成的稀疏向量，其中每个维度对应于一个唯一的词项，而且很多维度的值为零（即稀疏）。稀疏检索在处理长尾词项和具有明确语义的查询时效果较好，但可能在理解上下文和词义多样性方面受限。稀疏检索依赖于明确的词项匹配和统计权重。划重点，这一句很重要。
典型的算法如：

1. 基于 term weight 的算法，如 TF-IDF（Term Frequency-Inverse Document Frequency）
2. 基于文本相似性（lexical similarity）的算法，如 BM25. 这个分支沿用的思路是统计语言模型那套思想。有一定工作经验的读者可以仔细回忆回忆自己很多年前啃的统计学习书籍。

文本召回的方法，你得到的结果是一个超级大的 Phrase 集合以及其权重信息。在使用的时候，通常都需要依赖查询工具。最经典的工具就是 elastic search（ES）了，其底层最基础的索引算法是倒排索引（Inverted Index）。
这个方向并没有完全消失，直到今天依然有一些学者在研究，只是在尝试和最热门的趋势结合起来，主要是基于预训练模型（PTM）增强字面量检索这个方向。更具体的方向有 2 个：

1. Term weighting：根据每个 token 的上下文表示，针对性赋予词汇不同的权重。
    1. 这里提供几篇论文，有兴趣的同学自己阅读。不过都是两三年前的工作了，看来做研究的人还是挺喜欢追热点的 :)
    2. Dai, Z., & Callan, J. (2020, July). Context-aware term weighting for first stage passage retrieval. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval* (pp. 1533-1536).
    3. Dai, Z., & Callan, J. (2020, April). Context-aware document term weighting for ad-hoc search. In *Proceedings of The Web Conference 2020* (pp. 1897-1907).
    4. Gao, L., Dai, Z., & Callan, J. (2021). COIL: Revisit exact lexical match in information retrieval with contextualized inverted list. *arXiv preprint arXiv:2104.07186*.
    5. Lin, J., & Ma, X. (2021). A few brief notes on deepimpact, coil, and a conceptual framework for information retrieval techniques. *arXiv preprint arXiv:2106.14807*.

6. 一眼扫过去这几篇论文，你看出来有什么彩蛋没？搞学术的人不是一般的精明啊 :）正好这几天有清华的大佬发了学术研究的 GitHub，是不是开窍了？

2. Term expansion：解决 vocabulary mismatching problem，即：用户的查询词和文档的词汇之间存在不一致或不匹配的问题，导致检索系统无法准确地找到相关的文档。不理解的同学可以参考 NLP 基础中的新词发现、Query 改写等任务。解决的思路，主要是利用 PTM 扩展查询词或文档，即根据词汇的语义关联，增加一些与查询词或文档相关的词汇，以增加检索的召回率。

1.2 向量化召回

而后随着机器学习、深度学习的发展，初步产生了 word2vec、LDA、BERT 等等 milestone 级别的算法。向量化召回以及排序模块如雨后春笋般涌现。想必不少读者能脱口而出 sentence-bert、sim-cse 这两个模型。后文我也会涉及到网红学者的论文。

向量化召回（dense retrieval），是指：基于用户的 Query，通过向量化的方法从文本库中检索出与之相似的文本。

1.3 检索系统的 pipeline 模块

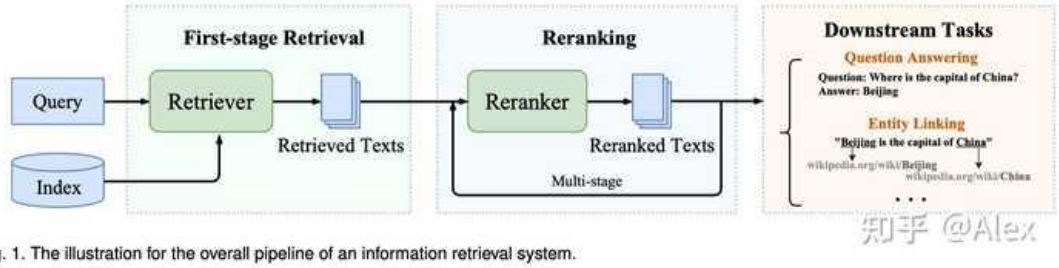


Fig. 1. The illustration for the overall pipeline of an information retrieval system.

2 数据集与评估指标

2.1 数据集

检索领域的常见学术数据集如下：

TABLE 1
Detailed statistics of available retrieval datasets. Here, "q" is the abbreviation of queries, and "instance" denotes a candidate text in the collection.

Categorization	Domain	Dataset	#q in train	#label in train	#q in dev	#q in test	#instance
Information retrieval	Web	MS MARCO [32]	502,939	532,761	6,980	6,837	8,841,823
	Web	mMARCO [44]	808,731	–	101,093	–	8,841,823
	News	TREC-NEWS [43]	–	–	–	57	594,977
	Biomedical	TREC-COVID [45]	–	–	–	50	171,332
	Biomedical	NFCorpus [46]	5,922	110,575	324	323	3,633
	Twitter	Signal-1M [47]	–	–	–	97	2,866,316
	Argument	Touché-2020 [48]	–	–	–	249	528,155
	Argument	ArguAna [49]	–	–	–	1,406	8,674
	Wikipedia	DBPedia [50]	–	–	67	400	4,635,922
	Web	ORCAS [51]	10.4M	18.8M	–	–	3,213,835
	Wikipedia	EntityQuestions [52]	176,560	186,367	22,068	22,075	–
	Web	MS MARCO v2 [53]	277,144	284,212	8,184	–	138,364,198
	Web	$\text{DuReader}_{\text{retrieval}}$ [54]	97,343	86,395	2,000	8,948	8,096,668
Question answering	Wikipedia	Natural Questions [33]	152,148	152,148	6,515	3,610	2,681,468
	Wikipedia	SQuAD [55]	78,713	78,713	8,886	10,570	23,215
	Wikipedia	TriviaQA [56]	78,785	78,785	8,837	11,313	740K
	Wikipedia	HotpotQA [57]	85,000	170,000	5,447	7,405	5,233,329
	Web	WebQuestions [58]	3,417	3,417	361	2,032	–
	Web	CuratedTREC [59]	1,353	1,353	133	694	–
	Finance	FiQA-2018 [60]	5,500	14,166	500	648	57,638
	Biomedical	BioASQ [61]	3,743	35,285	–	497	15,559,157
	StackEx.	CQADupStack [62]	–	–	–	13,145	457,199
	Quora	Quora [63]	–	–	5,000	10,000	522,931
	News	ArchivalQA [64]	853,644	853,644	106,706	106,706	483,604
	Web	CCQA [65]	55M	130M	–	–	–
Other tasks	Wikipedia	FEVER [66]	–	140,085	6,666	6,666	5,416,568
	Wikipedia	Climate-FEVER [67]	–	–	–	1 535	5,416,593
	Scitific	SciFact [68]	809	920	–	300	5,183
	Scitific	SciDocs [69]	–	–	–	1,000	25,657

扫一眼有个印象即可。对比方法效果的时候再针对性的看就好了。

2.2 评估指标

只给出几个常用的指标。

2.2.1 召回指标

召回模块的评估指标。

1. precision@K
    1. 最常用的指标之一。其含义是：召回的 Top K 个中有多少和真值相关。
    2. 分母是召回 Top K 中的 K。分子是：Top K 中有多少和真值相关。
2. recall@K
    1. 最常用的指标之一。其含义是：所有和真值相关的文档，有多少个被召回了。
    2. 分母是所有和真值相关的文档数。分子是召回的 K 个中有多少个和真值相关。
3. accuracy@k
    1. 这个指标不常用，不介绍了，很少很少用

其他指标就不做介绍了，实用价值不大。

2.2.2 排序指标

尽管本文的主要内容是讲召回，这里也给出排序模块常用的评估指标，这里仅介绍 NDCG，最常用的指标之一。

NDCG，即 Normalized Discounted Cumulative Gain（归一化折损累积增益），特点是综合考虑了结果的相关性和排序位置的影响。

在介绍 NDCG 之前先介绍 Discounted Cumulative Gain (DCG) 这个概念，这是 NDCG 的基础，旨在量化检索或推荐列表中每个位置项目的价值。它考虑了项目相关性（通常由专家或用户打分）以及该项目在列表中的位置。位置越靠前的高相关性项目贡献的增益越高。

DCG 的计算公式为：

DCGp=∑i=1prelilog⁡2(i+1)\begin{align} DCG_p = \sum_{i=1}^p \frac {rel_i} {log⁡2(i+1)} \end{align}

其中，rel_i 表示第 i 个检索结果的相关性评分，p 是考虑的项目数。

由于 DCG 值会随着查询的不同而变化，NDCG 通过对 DCG 进行归一化处理，使得不同查询的结果具有可比性。归一化的依据是假设有一个理想的排序（即所有结果按照相关性从高到低排序），计算出的理想 DCG（IDCG）。

NDCG 的计算公式为： NDCGp=DCGpIDCGp NDCG_p = \frac{DCG_p}{IDCG_p} 如果 IDCG 为 0，则 NDCG 也为 0。NDCG 的值域在 0 到 1 之间，值越接近 1 表示实际排序越接近理想排序，性能越好。

3 开源工具

1. Tevatron
    1. https://github.com/TextTron/Tevatron
    2. text processing, model training, text encoding, and text retrieval
2. Pyserini
    1. http://pyserini.io/
    2. 提供了一系列工具集，常用来快速复现一些论文的方法。
    3. 提供了一系列评估方法
3. Asyncval
    1. https://github.com/ielab/asyncval
    2. 用来：在训练的过程中，基于保存的 checkpoint 验证模型效果
4. OpenMatch
    1. https://github.com/OpenMatch/OpenMatch
5. MatchZoo

1. https://github.com/NTMC-Community/MatchZoo
2. 文本匹配的包。很好用的库，赞

6. PyTerrier
    1. https://github.com/terrier-org/pyterrier
    2. 一整套的召回框架，包括索引构建、文档召回等功能
7. SentenceTransformers
    1. https://www.sbert.net/
    2. 这个就非常常用了，获取句子/段落的 embedding，里面集成了很多模型

下面就是本文的核心内容了：dense retrieval。

dense retrieval 使用预训练的神经网络模型（如 BERT）来生成文档和查询的密集向量表示。在这种表示中，每个文档或查询都被映射到一个连续的向量空间，其中的维度不再对应于特定的词项。这种表示试图捕捉文本的潜在语义和上下文信息，从而可以更好地处理语言的复杂性和歧义。与稀疏检索相比，密集检索通常能提供更精确的语义匹配，特别是在短文本或语义相关但词汇不重叠的情况下，比如一词多义、需要上下文才能理解的语境。

4 几种典型的模型架构

4.1 模型架构简介

一幅图说明所有，不写晦涩难懂的大段文章。

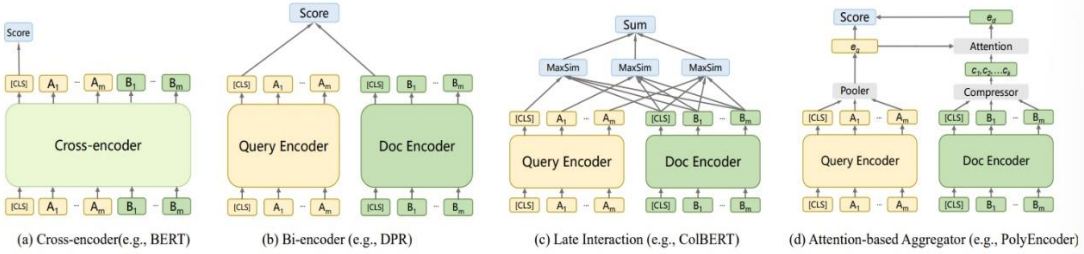


Figure 2: The comparison of different model architectures designed for retrieval/re-ranking.

图来自：Multiview document representation learning for open-domain dense retrieval

上面虽然列了四种结构，但后面两种其实都是对 bi-encoder 的改进，严格来说，还是遵循了 bi-encoder 的思想。更宽泛的意义上，检索领域一般只会归类为 bi-encoder 和 cross-encoder 两种架构。

另外，有一些论文中喜欢将 bi-encoder 叫做 dual-encoder，估计是叫双塔模型叫习惯了。

4.2 cross-encoder vs bi-encoder

Cross-encoder 主要用于 re-rank 模块，文本更专注于召回模块，这里只简明扼要介绍一下 cross-encoder 的特点。仔细看上面的图，主要有两个点：

1. 模型架构角度：
    1. bi-encoder 是两个编码器，输入端是 2 个分支，一个是 query，一个是 document；【recall】
    2. Cross-encoder 是一个编码器。输入端只有一个，是将 Query 通过[SEP]标签和 document 拼接在一起，并在 Query 的前面添加[CLS]标签。【re-rank】
2. 模型 loss 角度：

1. Bi-encoder 会取两个编码器的[CLS]的位置计算 score。当然也有一些工作是其他计算方式，比如去 MEAN。上图的后面两个工作都是在这块做改进的。
2. Cross-encoder 是取最开始位置的[CLS]，然后计算 score；

3. 应用的时候（推理阶段）：
    1. bi-encoder 训练完成之后，我们会将文档库中所有句子通过 bi-encoder 获取对应的 embedding。召回阶段通过 ES、faiss、scann 之类的工具进行向量化召回；
    2. Cross-encoder 推理时，和训练时没有太多区别，给一个 query 和 document pair 对，返回 similar score。
4. 项目选型角度：

通常，Cross-Encoder 的效果比 Bi-Encoders 要好，但是在大规模的数据集上性能不佳。一般将 bi-enocder 用做召回模型、cross-encoder 用作排序模型。具体项目选型的时候，基于这个思路再逐个挑适合自己的模型方案。召回模型，一般会召回 Top100.

4.2.1 为何 Cross-encoder 用在排序上，bi-encoder 用在召回？

一些文章上会说 Cross-encoder 用在排序上，bi-encoder 用在召回，为什么会这样？

这句话的原话是 Cross-Encoder 的效果比 Bi-Encoders 要好，但是在大规模的数据集上性能不佳。一般将 bi-enocder 用做召回模型、cross-encoder 用作排序模型。所以，我们对这句话进行拆分，一步步分析其背后的原理：

1. Bi-encoder 使用两个独立的编码器（通常是 BERT 模型），一个编码查询，一个编码文档。查询和文档被分别编码成固定维度的向量，然后通过计算它们的相似性（例如，使用点积或余弦相似度）来评估相关性。带来的好处是：
    1. 由于查询和文档是独立编码的，文档的向量表示可以预先计算并存储。查询处理时只需一次向量计算，可以利用高效的向量检索方法（例如，最近邻搜索）。因为 document 可以提前计算，所以可以提前对文档库计算一遍 embedding。因为可以提前算、推理时只用计算 Query 的 embedding，自然高效。适用于大规模数据集的点就在这儿。
    2. 坏处是：Query 和 document 是独立编码，导致无法捕捉查询和文档之间的细粒度交互特征。相关性评估受限于预先定义的向量相似度度量方式，可能损失一些复杂的语义信息。
2. Cross-Encoder
    1. Cross-encoder 将查询和文档一起输入单一的模型（通常也是 BERT），通过关注机制捕捉两者间的所有交互特征。输出一个直接的相关性分数。
    2. 因为可以捕捉查询和文档之间的细粒度互动，例如精确的词汇对齐和语义关系。通常在小规模数据集或高要求准确性的任务中表现更好。所以，说 cross-encoder 适用在排序模块，点在这儿。
    3. cross-encoder 每对查询和文档都需要一起通过模型，这导致计算开销极高。每一条新的查询请求都需要重新计算，不适合实时大规模检索。说 cross-encoder 不适用在大规模的文档库上的点在这儿。

5 背景知识

文本召回，通常你见到的绝大多数论文都是在说 text dense representation，即：如何设计更好的模型，获得更精确的 dense representation。

但是，文本召回想用起来并不是只要一个模型就可以，这和分类等领域完全不同。

5.1 文本召回的通用处理流程

通常的流程是：

1. 训练态
    1. 训练一个模型，输入是一个句子，输出是 representation。
    2. 对文档库，使用模型获取其 representation。
    3. 放到检索库中，构建索引，比如 faiss、scann、elasticsearch 中。
2. 推理态
    1. 用户输入一个 Query，调用模型，这个模型可以和上面的文档库的模型用一个，也可以不用，具体取决于你用的算法是什么。
    2. 拿到 Query representation，取索引库中查找出 Top K 个 documents.

5.2 搜索算法

拿到 Query representation，取索引库中查找出 Top K 个 documents. 这个过程，涉及到查找算法，比如朴素的暴力搜索，全部文档库数据。这里不介绍朴素算法了，没人会用。

对于做 NLP 的人来说，这个领域没有太多可以研究的，统计学习的那些算法，三四十年前就有了，但到今天也没几个人能再提出新的方法。关于这个方向，只给出导读内容，有兴趣的读者可以自己精深。

最常用的算法是 Approximate Nearest Neighbor Search (ANNS)。进一步拆分，可以成 2 个类别：

1. Non-exhaustive ANNS methods：

    这个类别不会压缩索引，而是通过减少召回的候选 document 数量，以达到加速的目的。常用的算法有 3 类：

    1. Tree-based methods
        1. Annoy：annoy · PyPI
        2. Scalable Nearest Neighbor algorithms for high dimensional data
    2. Inverted file methods

        这个就不说了，ElasticSearch 最核心的算法就是这个。有兴趣的同学可以看这个论文：

        1. Artem Babenko and Victor Lempitsky. 2014. The inverted multi-index. IEEE transactions on pattern analysis and machine intelligence 37, 6 (2014), 1247 – 1260.
        2. Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. IEEE transactions on pattern analysis and machine intelligence 33, 1 (2010), 117 – 128.
    3. Graph-based methods：（HNSW）
        1. Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World Graphs. IEEE transactions on pattern analysis and machine intelligence 42, 4 (2018), 824 – 836.
2. Vector compression

    主要思想是：压缩索引以及学习到一个打分函数，用于减少索引 document 的数量，和快速计算。

    1. 基于 Hash 的方法：Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In Proceedings of the thirtieth annual ACM symposium on Theory of computing. 604 – 613.
    2. quantization 的方法：

1. Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization. IEEE transactions on pattern analysis and machine intelligence 36, 4 (2013), 744 – 755
2. Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. IEEE transactions on pattern analysis and machine intelligence 33, 1 (2010), 117 – 128.

下面正式进入本文的核心章节。

6 Bi-encoder 的几种典型架构

Bi-encoder 架构的工作可以细分为以下类别：

1. **Single-representation bi-encoder**
2. **Multi-representation bi-encoder**
3. **Phrase-level representation**
4. Composite architectures
5. Lightweight architectures
6. Incorporating relevance feedback
7. Parameter-efficient tuning.

虽然从学术上，可以分为 7 个类别，但是本文主要对前面 3 个类别的一些典型工作进行详读，其他类别有的放矢的进行详读。剩下的类别，有兴趣的读者可以根据项目需要进一步梳理。

6.1 Single-representation bi-encoder

大名鼎鼎的 sentence-bert 就是这个方向的工作。这个方向对 sentence 或者 passage 的表示，都是用一个 representation。

这里给出两篇这个方向的典型工作导读。

关于 PTM 的选择，文献 Exploring the limits of transfer learning with a unified text-to-text transformer 中说 基于 T5 的 sentence embedding 比基于 BERT 的好。

6.1.1 sentence-bert (2019, EMNLP)

- 论文题目：Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks
- 发表会议：EMNLP
- 发表年份：2019
- 作者单位：达姆施塔特工业大学- 维基百科
- 论文链接：1908.10084 (arxiv.org)
- 代码：Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks | Papers With Code
    - 框架：PyTorch

6.1.1.1 一句话总结

作者对 BERT 结构进行修改，在 BERT 上添加 pooling 层，提出 sentence-bert，用于获取文本的 embedding，使之适用于 STS 任务。

6.1.1.2 方法

6.1.1.2.1 模型结构

模型结构如下。主要是在 BERT 上加了一个 Pooling 层，Pooling 层为变长的句子创建固定长度的表示向量。

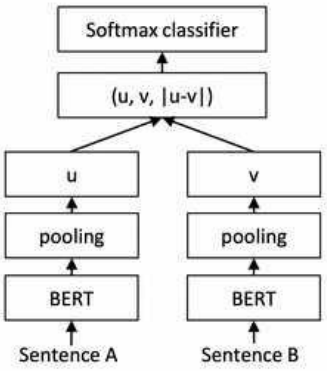


Figure 1: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).

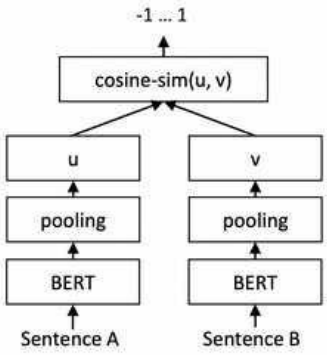


Figure 2: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function.

很多人可能看不懂上面这个图，觉得作者提出了两种 sentence-bert 结构。不是的，下面我拆开了说。 作者说的两种结构，更准确的说是针对分类和回归两种不同任务改造的模型结构，底层的网络结构都是一样的，只有最后的 loss 不同。

右边一个图中的 inference 这个词可能会让很多人误解，误以为推理时是模型实时这样一个句子 pair，过模型，然后算相似度。个人推测作者写作的时候，应该也是个新手小白。工业落地时，上面 cosine-sim 以下的部分是提前对文档库扫一遍，获取 embedding，然后存起来；Query 会用 sentence-bert 实时获取，再通过向量化召回工具召回 Top K，然后用一些相似度计算函数，比如 consine，计算获得相似度。

6.1.1.2.2 3 种 pooling strategy

1. 取 BERT [CLS]位置的 embedding
2. 对 BERT 的 logits 取 MEAN。默认取 MEAN
3. 对 BERT 的 logits 取 MAX

3 种 pooling 策略，MEAN 最好，CLS 次之，相差不大，MAX 效果差很多。

6.1.1.2.3 3 种任务的 loss

针对 3 种不同的数据集场景，设计了 3 种不同的 loss。

1. 分类任务的 loss
    1. similarity score 计算方式见上图左边
    2. loss 是 cross-entropy loss
2. 回归任务的 loss
    1. similarity score 计算方式见上图右边
    2. loss 是 mean squared error
3. triplet loss
    1. 适用于同时有正例和负例的场景，即给一个 Query，你的训练集要同时有 Query 的正例和负例
    2. loss 的公式如下：

$$max(||s_a - s_p|| - ||s_a - s_n|| + \epsilon, 0)$$

6.1.1.3 效果

现在已经有非常好的改进方法了，不再列图表。

6.1.1.4 个人看法

1. 作者论文 abstract 中对 BERT 不好的点，感觉不在点上，直接按我这样理解更准确，不会跑偏。作者工作的主要意义在于 embedding 的获取。
2. Sentence-bert 的最大问题在于不一致性：训练的时候，通过 loss function 让网络更新参数，但是在用的时候，却是用 consine 等相似度计算方法。当然，现在已经有很好的方法解决了。
3. 针对 sentence-bert 的缺陷，可以考虑使用 CoSent，主要是用了 circle loss 这个损失函数
    1. 参考 CoSENT：比 Sentence-BERT 更有效的句向量方案
    2. https://github.com/bojone/CoSENT

6.1.2 DPR（2020，EMNLP）

- 论文题目：Dense Passage Retrieval for Open-Domain Question Answering
- 发表会议：EMNLP
- 发表年份：2020
- 作者单位：脸书
- 论文链接：https://arxiv.org/abs/2004.04906
- 代码：https://github.com/facebookresearch/DPR
    - 框架：PyTorch

6.1.2.1 一句话总结

基于 BERT 的 dense retrieval 模型，用在对话任务上。

6.1.2.2 方法

6.1.2.2.1 模型结构

模型结构和 sentence-bert 没有太多区别。只是没有 pooling 层。取的 BERT [CLS]位置的 representation。

6.1.2.2.2 负采样

负采样，是整篇论文个人感觉唯一可以说的点。但这些方法我也在 2020 年之前的论文中见过 :)

说人话，就是如何构造高质量的负样本。具体作者提了 3 个点：

1. 方法一：从 corpus 中随机筛选一个
2. 方法二：BM25 算法返回的 Top K 个 passages，但 passages 不包含答案，并且和 question 的大部分 token 都相同
3. 方法三：其他问题的 positive passages

6.1.2.2.3 In-batch negatives

简单说，就是训练时在一个 batch 内，如何创造负例。这样训练数据就更多。

这一点就不说了，只能算一个小的 Tips，后文会有论文进一步讲解。

6.1.2.2.4 loss function

log likelihood 损失函数。分母是累加了负例的 similar score。

$$L(q_i, p_i^+, p_{i,1}^-, \cdots, p_{i,n}^-) \quad (2)$$
$$= -\log \frac{e^{\text{sim}(q_i, p_i^+)}}{e^{\text{sim}(q_i, p_i^+)} + \sum_{j=1}^{n} e^{\text{sim}(q_i, p_{i,j}^-)}}.$$

6.1.2.2.5 在 QA 任务上应用

1. representation 取的是 BERT CLS 位置的输出。Question 和 passage 都是这样，维度用的 768
2. 向量化索引是用的 FAISS

6.1.2.3 效果

passages 的个数是 21 M，召回 K 的个数 20 到 100.

Training	Retriever	Top-20					Top-100				
		NQ	TriviaQA	WQ	TREC	SQuAD	NQ	TriviaQA	WQ	TREC	SQuAD
None	BM25	59.1	66.9	55.0	70.9	68.8	73.7	76.7	71.1	84.1	80.0
Single	DPR	78.4	79.4	73.2	79.8	63.2	85.4	**85.0**	81.4	89.1	77.2
	BM25 + DPR	76.6	79.8	71.0	85.2	**71.5**	83.8	84.5	80.5	92.7	**81.3**
Multi	DPR	**79.4**	78.8	**75.0**	**89.1**	51.6	**86.0**	84.7	**82.9**	93.9	67.6
	BM25 + DPR	78.0	**79.9**	74.7	88.5	66.2	83.9	84.4	82.3	**94.1**	78.6

Table 2: Top-20 & Top-100 retrieval accuracy on test sets, measured as the percentage of top 20/100 retrieved passages that contain the answer. *Single* and *Multi* denote that our Dense Passage Retriever (DPR) was trained using individial or combined training datasets (all the datasets excluding SQuAD). See text for more details.

上图中，更应该关注的是 Single 那一行，这才是项目落地时可能的效果。不过，BM25+DPR 的效果反而差，值的特别关注. :warning:

6.1.2.4 改进方案

后续有学者提出了改进方案。

1. RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering（2021，NAACL）
    1. 论文链接：https://aclanthology.org/2021.naacl-main.466/
    2. 一句话描述：改进的点集中在负采样层面。不再细读这篇论文了，感兴趣的读者可以细细品读 :)
2. Approximate nearest neighbor negative contrastive learning for dense text retrieval（2021，ICLR）
    1. 论文链接：https://arxiv.org/pdf/2007.00808
    2. 代码：https://github.com/microsoft/ANCE
    3. 作者单位：微软
    4. 一句话描述：
        1. 发现 dense retrieval 的瓶颈在于 batch 内没有有效的负例，导致梯度范数减小、随机梯度方差增大以及学习收敛速度变慢。
        2. 提出一种难负例的采样方法。

6.1.2.5 个人看法

1. 从方法上看，和 sentence-bert 没太多区别，添加了两种负采样的方法。当然如果非要对比的话，能对比出来几个点，比如一个是 sentence embedding，一个是用在对话任务上，但方法上，没啥区别。
2. 从应用场景上看，突出了对话场景。相比于 sentence-bert，论文中的流程和工业应用时流程相似。简单说的话，sentence-bert 是一种 embedding 方法，这个像是 sentence-bert 在对话任务上的落地。
3. 作者解决的问题是：用 Bert 基座，使用 question 和 passage pair 训练出来一个模型，效果比 BM25 好。基于这个看的话，没看出来有啥特别的，不知道当初审稿人咋通过的？DPR 和 sentence-bert 都是 EMNLP，一个是 2019 年，一个是 2020 年，差了一年整，DPR 全文也没有引用和对比 sentence-bert，看不懂这是啥操作。。

6.2 Multi-representation bi-encoder

Multi-representation bi-encoder，解决 Single-representation bi-encoder 无法更细粒度捕捉 query 和 text 之间联系的问题。

具体是：对 text 学习到 M 个不同的 embeddings，拼接在一起，在与 query embedding 计算相关性。

这个分支侧重于上下文理解，侧重于解决【吃的苹果和用的苹果手机在不同上下文时语义不同】的问题。

效果较好，但是需要的硬件资源很多，比如存 contextual embedding 时的内存消耗大。

下面给出 6 篇论文导读。

6.2.1 poly-encoder（2020，ICLR）

- 论文题目：Polyencoders: Architectures and pre-training strategies for fast and accurate multi-sentence scoring
- 发表会议：ICLR
- 发表年份：2020
- 作者单位：脸书
- 论文链接：https://arxiv.org/abs/1905.01969
- 代码：https://github.com/sfzhou5678/PolyEncoder
    - 框架：

6.2.1.1 一句话总结

对 input 使用 transformer 生成 m 个向量。具体看下面的图即可。

6.2.1.2 模型结构

直接看图。图画的很好。那种没有图的，通常我都会觉得是故弄玄虚，故意耍一堆公式，迷惑人觉得文章特别牛逼哄哄的，反而喜欢那种简单明了有效果的工作。

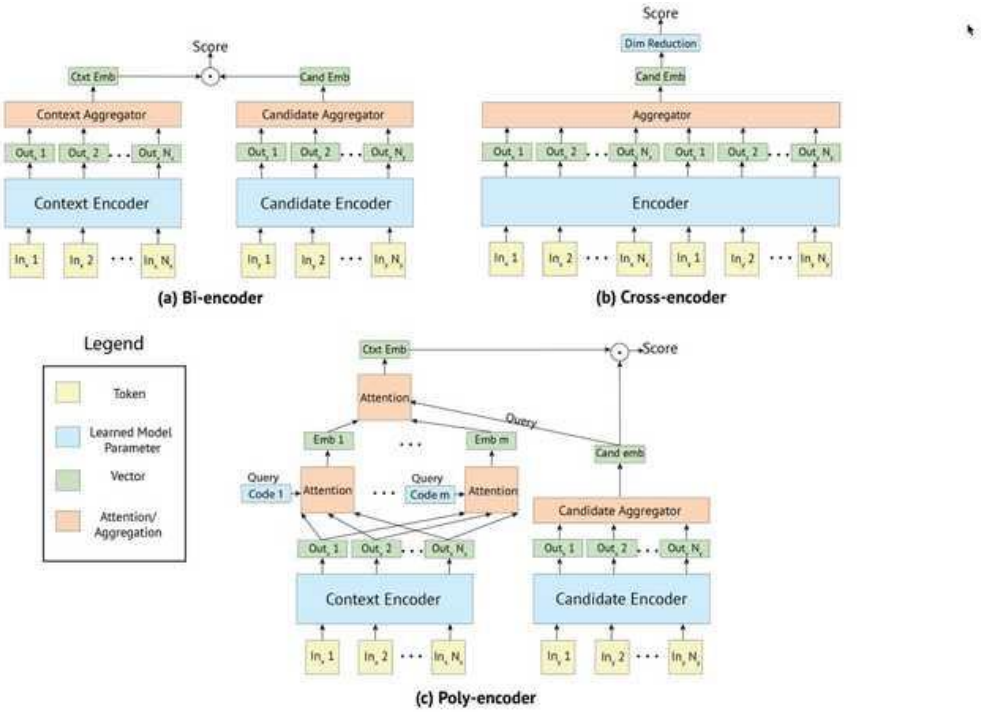


Figure 1: Diagrams of the three model architectures we consider. (a) The Bi-encoder encodes the context and candidate separately, allowing for the caching of candidate representations during inference. (b) The Cross-encoder jointly encodes the context and candidate in a single transformer, yielding richer interactions between context and candidate at the cost of slower computation. (c) The Poly-encoder combines the strengths of the Bi-encoder and Cross-encoder by both allowing for caching of candidate representations and adding a final attention mechanism between global features of the input and a given candidate to give richer interactions before computing a final score.

6.2.1.3 效果

Poly-encoders are faster than Cross-encoders and more accurate than Bi-encoder。

1. 模型效果

Dataset	ConvAI2	DSTC 7		Ubuntu v2		Wikipedia IR
split	test	test		test		test
metric	R@1/20	R@1/100	MRR	R@1/10	MRR	R@1/10001
(Wolf et al., 2019)	80.7					
(Gu et al., 2018)	-	60.8	69.1	-	-	-
(Chen & Wang, 2019)	-	64.5	73.5	-	-	-
(Yoon et al., 2018)	-	-	-	65.2	-	-
(Dong & Huang, 2018)	-	-	-	75.9	84.8	-
(Wu et al., 2018)	-	-	-	-	-	56.8
pre-trained BERT weights from (Devlin et al., 2019) - Toronto Books + Wikipedia						
Bi-encoder	81.7 ± 0.2	66.8 ± 0.7	74.6 ± 0.5	80.6 ± 0.4	88.0 ± 0.3	-
Poly-encoder 16	83.2 ± 0.1	67.8 ± 0.3	75.1 ± 0.2	81.2 ± 0.2	88.3 ± 0.1	-
Poly-encoder 64	83.7 ± 0.2	67.0 ± 0.9	74.7 ± 0.6	81.3 ± 0.2	88.4 ± 0.1	-
Poly-encoder 360	83.7 ± 0.2	68.9 ± 0.4	76.2 ± 0.2	80.9 ± 0.0	88.1 ± 0.1	-
Cross-encoder	84.8 ± 0.3	67.4 ± 0.7	75.6 ± 0.4	82.8 ± 0.3	89.4 ± 0.2	-
Our pre-training on Toronto Books + Wikipedia						
Bi-encoder	82.0 ± 0.1	64.5 ± 0.5	72.6 ± 0.4	80.8 ± 0.5	88.2 ± 0.4	-
Poly-encoder 16	82.7 ± 0.1	65.3 ± 0.9	73.2 ± 0.7	83.4 ± 0.2	89.9 ± 0.1	-
Poly-encoder 64	83.3 ± 0.1	65.8 ± 0.7	73.5 ± 0.5	83.4 ± 0.1	89.9 ± 0.0	-
Poly-encoder 360	83.8 ± 0.1	65.8 ± 0.7	73.6 ± 0.6	83.7 ± 0.0	90.1 ± 0.0	-
Cross-encoder	84.9 ± 0.3	65.3 ± 1.0	73.8 ± 0.6	83.1 ± 0.7	89.7 ± 0.5	-
Our pre-training on Reddit						
Bi-encoder	84.8 ± 0.1	70.9 ± 0.5	78.1 ± 0.3	83.6 ± 0.7	90.1 ± 0.4	71.0
Poly-encoder 16	86.3 ± 0.3	71.6 ± 0.6	78.4 ± 0.4	86.0 ± 0.1	91.5 ± 0.1	71.5
Poly-encoder 64	86.5 ± 0.2	71.2 ± 0.8	78.2 ± 0.7	85.9 ± 0.1	91.5 ± 0.1	71.3
Poly-encoder 360	86.8 ± 0.1	71.4 ± 1.0	78.3 ± 0.7	85.9 ± 0.1	91.5 ± 0.0	**71.8**
Cross-encoder	**87.9 ± 0.2**	**71.7 ± 0.3**	**79.0 ± 0.2**	**86.5 ± 0.1**	**91.9 ± 0.0**	-

Table 4: Test performance of Bi-, Poly- and Cross-encoders on our selected tasks.

2. 推理耗时

	Scoring time (ms)			
	CPU		GPU	
Candidates	1k	100k	1k	100k
Bi-encoder	115	160	19	22
Poly-encoder 16	122	678	18	38
Poly-encoder 64	126	692	23	46
Poly-encoder 360	160	837	57	88
Cross-encoder	21.7k	2.2M*	2.6k	266k*

Table 5: Average time in milliseconds to predict the next dialogue utterance from C possible candidates on ConvAI2. * are inferred.

6.2.1.4 个人看法

1. transformer 出来的时候，我也见过各种造 attention 的工作，不好说有多少实用价值。

6.2.2 ME-BERT（2021，TACL）

- 论文题目：Sparse, dense, and attentional representations for text retrieval
- 发表会议：TACL
- 发表年份：2021
- 作者单位：Google
- 论文链接：https://arxiv.org/pdf/2005.00181
- 代码：https://github.com/google-research/language/tree/master/language/multivec

- 框架：tensorflow v1

6.2.2.1 一句话总结

从理论和公式角度，论证：对于长文本，使用固定维度的向量，会导致效果欠佳。此外，作者提出 Multi-Vector Encoding from BERT 和 Sparse-Dense Hybrids 两种方法，其中 multi-vector 再加 Sparse-Dense Hybrids 效果最好。

6.2.2.2 模型结构

论文中前面堆了大量数学公式，只是为了论证：对于长文本，使用 fixed-length 维度的向量，会导致效果欠佳。抛开这个点之外，方法上的创新主要是下面两个：

1. Multi-Vector Encoding from BERT (ME-BERT)
    1. Multi-vector 上，其实就是取 BERT 的前 M 个输出 embedding。作者实验，觉得 M 取 3 和 4 比较好。
    2. 另外作者还有一个 ME-BERT 版本，具体是在 BERT 后面加一个 feed-forward layer，维度是 768 * k。
2. Sparse-Dense Hybrids (HYBRID)
    1. Hybrid 的过程不复杂，是 将 sparse 和 dense combine 在一起，然后过一个 full connected NN。

6.2.2.3 效果

文章中有一个是用的 dev set，不懂看 dev set 有啥意义？这里我贴了 test set 的效果。

Model	MRR(MS)	RR	NDCG@10	Holes@10
Passage Retrieval				
BM25-Anserini	0.186	0.825	0.506	**0.000**
DE-BERT	0.295	0.936	0.639	0.165
ME-BERT	0.323	0.968	0.687	0.109
DE-HYBRID-E	0.306	0.951	0.659	0.105
ME-HYBRID-E	**0.336**	**0.977**	**0.706**	0.051
Document Retrieval				
Base-Indri	0.192	0.785	0.517	**0.002**
DE-BERT	-	0.841	0.510	0.188
ME-BERT	-	0.877	0.588	0.109
DE-HYBRID-E	0.287	0.890	0.595	0.084
ME-HYBRID-E	0.310	**0.914**	**0.610**	0.063

Table 2: Test set first-pass retrieval results on the passage and document TREC 2019 DL evaluation as well as MS MARCO eval MRR@10 (passage) and MRR@100 (document) under MRR(MS).

6.2.2.4 个人看法

1. 从写作手法看，为投稿人定向优化，大量公式，但只要你耐下心来，也能看懂。除去最开始的论证，方法的创新主要集中在 Multi-vector 上，其实就是取 BERT 的前 M 个输出 embedding。很多论文都有为投稿人定向优化的写作手法，想说些啥，但又戛然而止，打不过就只能加入?
2. 从方法创新上，排除堆公式，显得很有含金量之外，方法上的创新略显不足，取 BERT 的前 M 个 embedding，还有 hybrid 的方法，都只能算微创新，这或许也是为何发 TACL 的原因吧?
3. 取 BERT 的前 M 个 embedding，论文第四章，说 M 用 3 和 4 比较好，按作者大量数学公式的论证思路，却在这个关键地方戛然而止，看不懂
4. 提出的 2 个方法，这本是最核心的内容，却都只是文字描述，用公式是最直观明了知道具体的方法是什么，却在这个位置戛然而止，表示看不懂。比如 hybrid 的方法中，作者用了 combine，会本能的想是 concat 还是 add，但如果用公式，就会非常严谨。并且，这些方法都非常简单。一开始觉得论文很牛逼，但是好多个我特别想关注的点，都戛然而止，一连串点号。。。
5. 本文作者主打的点是 long document，即便项目上要选用这个方法，也要看自己是不是 long document 这个场景
6. 作者的实验论证 Sparse-Dense Hybrids (HYBRID) 方法，但是有的论文中却说 hybrid 方法效果不好，这一点再看看 :warning:

6.2.3 ColBERT（2020，SIGIR）

- 论文题目：Colbert: Efficient and effective passage search via contextualized late interaction over BERT
- 发表会议：SIGIR
- 发表年份：2020
- 作者单位：Stanford
- 论文链接：https://arxiv.org/abs/2004.12832
- 代码：https://github.com/stanford-futuredata/ColBERT
    - 框架：PyTorch

6.2.3.1 一句话总结

1. 提出 late interaction 结构。
2. 论文里面的 delaying query-document interaction 观点（第三章节开头）也值得仔细精深。原话是：delaying the query − document interaction can facilitate cheap neural re-ranking (i.e., through pre-computation) and even support practical end-to-end neural retrieval (i.e., through pruning via vector-similarity search).

6.2.3.2 方法

6.2.3.2.1 query & document encoders

query 和 document 的编码方法如下：

$$E_q := \text{Normalize}(\ \text{CNN}(\ \text{BERT}(\text{“}[Q]q_0q_1...q_l\#\#...\#\text{”})\ )\ ) \quad (1)$$

$$E_d := \text{Filter}(\ \text{Normalize}(\ \text{CNN}(\ \text{BERT}(\text{“}[D]d_0d_1...d_n\text{”})\ )\ )\ ) \quad (2)$$

论文中写了很多，只看上面两个公式就知道怎么做的了。这里选用了 CNN，看不懂为何用 CNN。

CNN 十年之前就有人说用来提取特征，但是在文本领域，CNN 的作用并没有想象中的那么大，所以，后来就很少有人再用 CNN。但这里却用了 CNN，我也没有找到作者为何这样设计的动机。这点，再反刍反刍。

6.2.3.2.2 late interaction 结构

一张图看懂作者的方法，言简意赅。

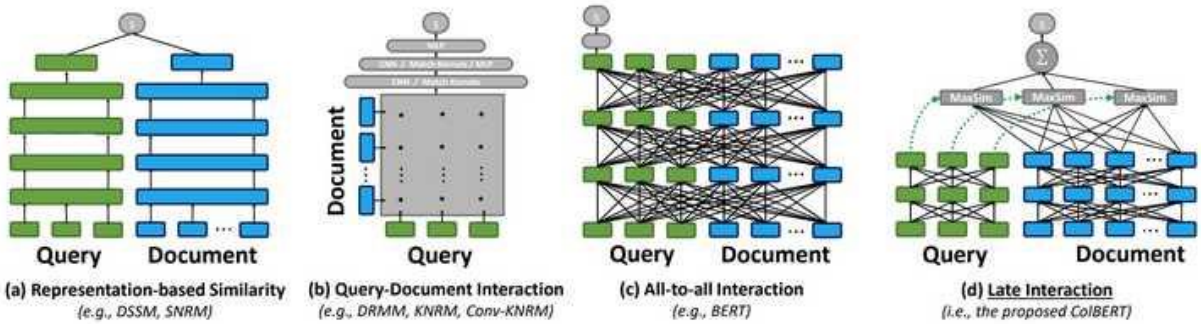


Figure 2: Schematic diagrams illustrating query–document matching paradigms in neural IR. The figure contrasts existing approaches (sub-figures (a), (b), and (c)) with the proposed late interaction paradigm (sub-figure (d)).

注意对比 C 和 D 两个图，就明白了作者说的 late interaction 是什么意思。

出个问题，看看你看懂了没。Multi-vector representation 体现在哪儿?
注意看图片最上面的多根连线，绿色的和灰色的。这和以往通用做法，如 CLS 位置、MEAN、MAX pooling 的做法，有所区别。

仔细看这个图，你发现作者的小心思了没？下面给出我的拆解。
这张图的前 2 个不用看，都是过时的模型，凑数的。所以，只用看后两个模型。后面两幅图，作者的小心思真的是不要不要的。第三副图用的是 cross-encoder 的架构，第四幅图用的是 bi-encoder 的架构，乍一看你会觉得第四幅图比第三幅图简洁很多，但如果你明白【第三副图用的是 cross-encoder 的架构，第四幅图用的是 bi-encoder 的架构】，第四幅图真正应该对比的是 bi-encoder 架构，此时就会发现作者的创新只有上面的 MaxSim。并且，作者在后面说应用的时候，主打的点是排序。简言之，模型结构上，用 Bi-encoder 和 cross-encoder 进行对比，但是在应用的时候，却又重新提起排序。我说完这个点之后，你对作者的这篇论文会怎么看？然后作者起了一个很新颖的名字。不知道当初的审稿人有没有仔细看透这一点 :)

6.2.3.2.3 MaxSim

这个名字很唬人。拆解出来就是：ColBERT 不再是只取一个位置的 representation，而是最后一排位置的 representation 我都要用，那这里就需要解决一个问题：一排 representation，我如何算相似度？之前一个位置是用的 consine，于是这里用的是取 Max，就是每个位置算 consine，然后取 Max.

公式如下：

$$S_{q,d} = \sum_{i=1}^{N} \max_{j=1}^{M} Q_i \cdot D_j^T$$

6.2.3.2.4 ColBERT 用作排序模型

作者在 abstract 章节中说 ColBERT 模型是 rank model，但仔细拆解 late interaction 后，却发现其只是 bi-encoder 架构，如果不懂 dense retrieval，甚至如果普通级别的懂，可能会被 rank model 这个词汇带偏。

具体方法是：query 用 ColBERT 获取 embedding，document 也用 ColBERT 获取 embedding，只不过因为 document 是 K 个，作者将其放置成一个矩阵，最后对每个 document embedding 通过 MaxSim 计算 score。

这块是纯工程了，把工程步骤说了一遍。作者后面还写了一个用 ColBERT 做 end-to-end retrieval 的，有兴趣的读者自己读好了。

6.2.3.3 效果

效果看原始论文好了。

这里给出一个我自己的视角，读者也可以看看我说的对不对?

论文中 Table 1 和 Table 2，作者用的都是召回 1000，为何不敢召回 Top 100 或者 20、50 呢？因为很多论文都会对比这几个数字

6.2.3.4 个人看法

1. 文章提到了一个非常有意思的观点： delaying the query − document interaction can facilitate cheap neural re-ranking (i.e., through pre-computation) and even support practical end-to-end neural retrieval (i.e., through pruning via vector-similarity search).
2. ColBERT 论文中用的是 Max，也给出了消融实验的结果。但是一些论文中却说 max 是效果最差的，这一点再看看
3. 在 late interaction 这个章节中，我详细拆解了作者的方法。论文前前后后写了好多页，去伪存真之后，其实并没有太多新的东西，但是却中了 SIGIR。作为读者的你我，可能需要好好想想打不过就加入的写作手法

6.2.4 ColBERTer（2022, CIKM）

- 论文题目：Introducing neural bag of whole-words with colberter: Contextualized late interactions using enhanced reduction
- 发表会议：CIKM
- 发表年份：2022
- 作者单位：stanford
- 论文链接：https://arxiv.org/abs/2203.13088
- 代码：sebastian-hofstaetter/colberter (github.com)
    - 框架：PyTorch

6.2.4.1 一句话总结

提出 ColBERTer 方法，论文属于剑走偏锋的从 ColBERT 的内存使用量角度进行优化。这种选题风险很高，如果有审稿人觉得 ColBERT 不好，这种改进版的文章可能会被直接毙掉。

6.2.4.2 方法

一如往常，一图说所有。

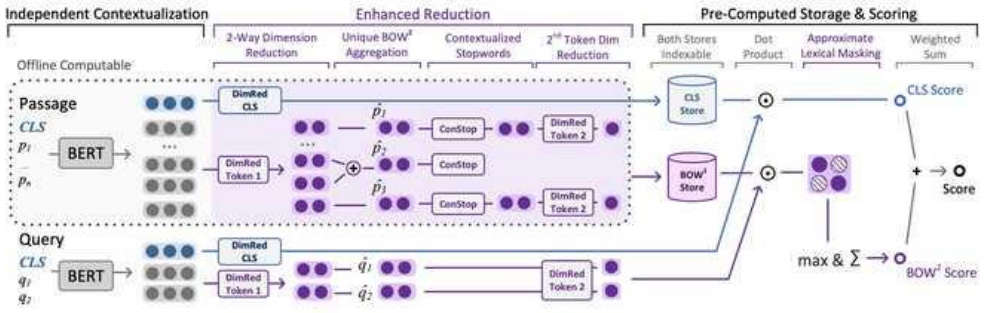


Figure 2: The ColBERTer encoding architecture, followed by the query-time workflow. The passage representations (both the single CLS and token vectors) are pre-computed during indexing time. The enhanced reductions with the 2-way dimension reduction, the unique BOW² aggregation, contextualized stopwords and token dimensionality reduction are applied symmetrically to passages and queries (except for the stopword removal).

下面我对上图一步步拆解，看作者是如何减少内存使用的。具体措施有：

1. 减少模型输出的 embedding 维度
    1. 主要是对 BERT 输出的 representation 进行量化。具体是两个：CLS 位置以及剩下的位置。感觉作者应该是看了量化的文章，想出来的这个方法。
    2. 第一个是：BERT 输出的第一个位置是 CLS，过一个 linear layer，维度会减少到 128 维度。原始的 BERT 一般都是用 384 或者 768；
    3. 第二个是：除 CLS 位置之外的输出。同样的手法，过一个矩阵，这个矩阵维度会更低，作者用的 32 维度；
2. BERT tokenizer 用的是 WordPiece 算法，作者对 WordPiece 出的词汇片段进行合并，从而减少存储量；
3. 对 stopwords 的 embedding 进行量化
    1. 通常来说，stopwords 都被认为是没有语义的，作者对这些词汇进行极限压缩。
4. Max-pooling & aggregation
    1. 对除 CLS 的位置通过 Max-pooling，获得 representation；
    2. 至于其他细节，看论文，也没有什么特别复杂的，作者使用 aggregation 比较常见。

6.2.4.3 效果

还是我在 ColBERT 中说的那个点，为啥总是对比召回 Top 1000 的时候？有几个项目落地时召回用 Top 1000 的？

6.2.4.4 个人看法

1. 全文的方法都是在做量化，这是看了量化的文章，拿过来移花接木的；但是量化都是有损的。
2. 这种方法只可以用在端上，但是对端上来说，这 6G 内存占用量，端上能用的不多；
3. 从落地角度看，内存占用量有点多，效果也没拉开差距，有点鸡肋；

6.2.5 MVR（2022，ACL）

- 论文题目：Multiview document representation learning for open-domain dense retrieval
- 发表会议：ACL
- 发表年份：2022
- 作者单位：微软
- 论文链接：https://arxiv.org/abs/2203.08372
- 代码：https://github.com/wuyaoxuehun/colbert?tab=readme-ov-file

- 框架：

6.2.5.1 一句话总结

模型结构沿用了 ColBERT 的思路，改进点主要在 encoder 的前面[VIE]标签，Query encoder 中添加一个，document encoder 添加多个。

6.2.5.2 方法

6.2.5.2.1 多个[VIE]标签

方法刚才已经一句话总结了。这里给出作者这样做的动机：

a document can usually answer multiple potential queries from different views. So the single vector representation of a document is hard to match with multi-view queries, and faces a semantic mismatch prob.

很有意思的视角。multi-representation 有的论文是截取前 M 个 embedding，从而得到多个 vector，作者这里直接是用多个标签替代。interesting

6.2.5.2.2 Global-Local Loss

Global-Local Loss，由 2 部分组成：

global loss 来自原本 bi-encoder 的，local loss 是作者新加的。

L=Lglobal+λLlocal \begin{align} \mathcal{L} &= \mathcal{L}_{global} + \lambda \mathcal{L}_{local} \ \end{align}

1. global loss 这个没什么好说的，直接看下面的公式即可。

$$\mathcal{L}_{global} = -\log \frac{e^{f(q,d^+)/\tau}}{e^{f(q,d^+)/\tau} + \sum_l e^{f(q,d_l^-)/\tau}}$$

2. local loss 这个特别讲解一下：

前文说过，作者在 document 前面加入了 K 个[VIE]标签，

$$\mathcal{L}_{local} = -\log \frac{e^{f(q,d^+)/\tau}}{\sum_k e^{fi(q,d^+)/\tau}}$$

上面公式中的 τ\tau 如下。关于设计 τ\tau 的目的，原文是这样说的：鼓励更多不同位置的[VIE]标签能被激活。在初始阶段，会取 0.3，这样 softmax 更容易获得一个均匀分布。之后随着训练进行，不再进行这么激发，倾向于让优化更稳定。但是，作者后续的实验结果，似乎证明没啥用，用 fixed 效果基本是最优的。此外，感觉这儿的设计是借鉴了 GPT 中 temperature。

$$\tau = max\{0.3, exp(-\alpha t)\} \quad (8)$$

Where α is a hyper-parameter to control the annealing speed, t denotes the training epochs, and the temperature updates every epoch. To simplify

6.2.5.3 效果

1. 算法效果

Method	SQuAD			Natural Question			Trivia QA		
	R@5	R@20	R@100	R@5	R@20	R@100	R@5	R@20	R@100
BM25 (Yang et al., 2017)	-	-	-	-	59.1	73.7	-	66.9	76.7
DPR (Karpukhin et al., 2020)	-	76.4	84.8	-	74.4	85.3	-	79.3	84.9
ANCE (Xiong et al., 2020)	-	-	-	-	81.9	87.5	-	80.3	85.3
RocketQA (Qu et al., 2021)	-	-	-	74.0	82.7	88.5	-	-	-
Condenser (Gao and Callan, 2021a)	-	-	-	-	83.2	88.4	-	81.9	86.2
DPR-PAQ (Oğuz et al., 2021)	-	-	-	74.5	83.7	88.6	-	-	-
DRPQ (Tang et al., 2021)	-	80.5	88.6	-	82.3	88.2	-	80.5	85.8
coCondenser (Gao and Callan, 2021b)	-	-	-	75.8	84.3	89.0	76.8	83.2	87.3
coCondenser(reproduced)	73.2	81.8	88.7	75.4	84.1	88.8	76.4	82.7	86.8
MVR	**76.4**	**84.2**	**89.8**	**76.2**	**84.8**	**89.3**	**77.1**	**83.4**	**87.4**

Table 1: Retrieval performance on SQuAD dev, Natural Question test and Trivia QA test. The best performing models are marked bold and the results unavailable are left blank.

耗时对比的时候，对比了 ColBERT，ColBERT 也提供了召回的思路，为啥这里没有对比？:question:

2. 耗时

Method	Doc Encoding	Retrieval
DPR	2.5ms	10ms
ColBERT	2.5ms	320ms
MEBERT	2.5ms	25ms
DRPQ	5.3ms	45ms
MVR	2.5ms	25ms

Table 4: Time cost of online and offline computing in SQuAD retrieval task.

6.2.5.4 个人看法

1. 少见的好论文，方法直观明了，同时，也有很好的 loss 设计；

2. 写作手法，也直截了当，不像有的论文，花拳绣腿的一堆公式，关键是这些公式还都是冗余的。不是说公式不好，关键是公式得有用才行。个人感觉这种风格也为中ACL 加了一点分数；
3. 对比的方法中缺少同期有影响力的方法，不太好
4. 适用于 document 场景，如果是 sentence 场景，不太合适。也就是说，适用于 document QA 领域。这一点读者可以好好思考，我为何这样说

6.2.6 MADRM（2022，KDD）

- 论文题目：Multi-Aspect Dense Retrieval
- 发表会议：KDD
- 发表年份：2022
- 作者单位：Google
- 论文链接：Multi-Aspect Dense Retrieval (acm.org)
- 代码：
    - 框架：

6.2.6.1 一句话总结

1. 提出 aspect extract network 和 aspect fusion network 提取以及融合 aspect embedding。（aspect 可以理解成领域标签，这里为了和原文保持一致，沿用 aspect。）
2. 引入 aspect prediction task 用于强化模型学习领域知识，适用于垂类领域。

6.2.6.2 方法

备注：文章中的 aspect 是提前有多个 aspect vocabulary。

具体步骤有：

1. 第一步：获取 bert 的输出
2. 第二步：使用 Aspect Extraction Network 提取 aspect embedding
3. 第三步：Aspect Learning
4. 第四步：Aspect Fusion Network

6.2.6.2.1 模型结构

模型整体结构是：

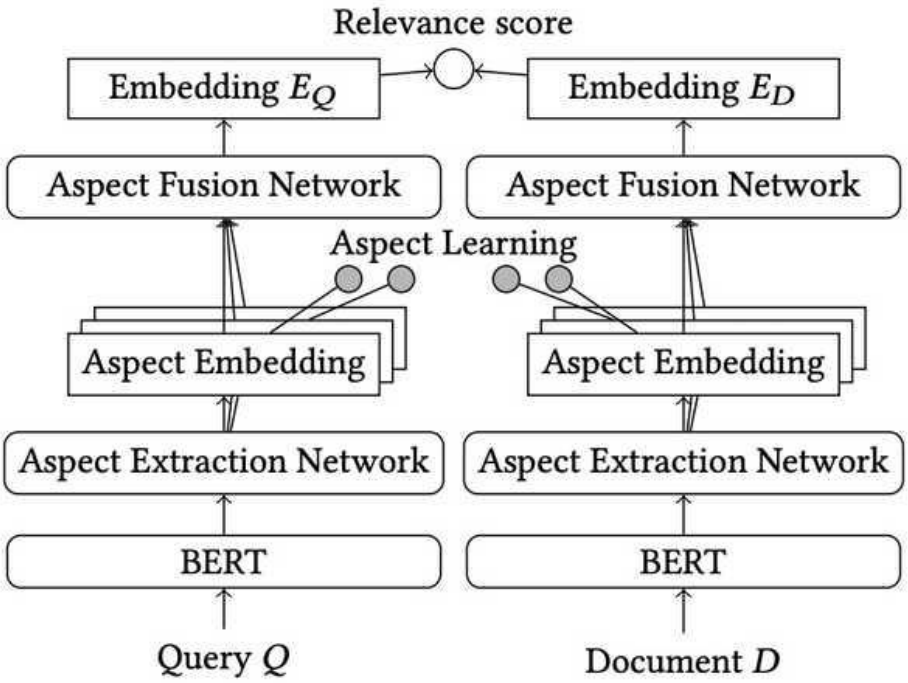


Figure 1: Multi-Aspect Dense Retrieval model (MADRM).

6.2.6.2.2 Aspect Extraction Network

aspect extraction network 的结构如下：

作用： extract embeddings for each aspect, called “aspect embeddings”

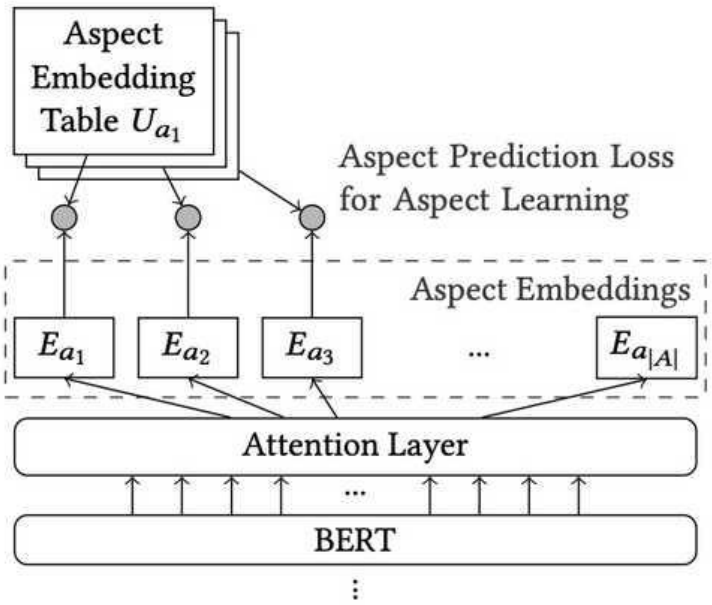


Figure 2: Aspect Extraction Network (AEN). Section 4 and Section 5 respectively describe the network structure and its Aspect Learning process.

除了预先定义的 aspect，这里还定义了一个 other 类别。

6.2.6.2.3 Aspect Learning

Aspect Learning 的目的是：上面得到的 aspect embedding 对于当前的句子来说，不精确，通过 training 进一步精细化 aspect embedding。

具体是：设计了 Aspect prediction loss 以及 aspect prediction task 进行 training。

1. aspect prediction task
    1. 一个例子说明 aspect prediction task 是什么：ugg sandals 这个 Query，训练数据有其 category 和 brand 标签（分别是 Shoes 和 UGG），基于上一步的 aspect embedding，预测 category 和 brand 标签。
2. aspect prediction loss，具体是基于 softmax cross entropy loss

6.2.6.2.4 Aspect Fusion Network

这一步的目的是：通过 aspect learning 学习的多个 aspect embedding，需要融合成一个 single-vector embedding。这一步就是在说怎么融合的。

具体分为 2 步：

1. Presence Weighting
    1. Weighted Sum 存在的问题是：aspect weight 不依赖于 input，导致的结果是任何 input，都会计算所有 aspect。作者这里设计了一个二分类的思路，判断 aspect 是否在 query 中出现，又乘上一个超参权重。
2. CLS-Gating
    1. 公式： Linear(CLS)Linear(_{CLS})
    2. linear 输出的维度是 aspects 个数。但这儿很让人疑惑，CLS 是 Bert 输出的，并没有用到 aspect，为何用 CLS 就能达到作者的目标?
    3. 这里作者写的时候又扯到了 MOE，不知道是不是因为 MOE 热，又”贴近时事“?

6.2.6.3 效果

不贴效果了，场景和文本场景有点远，适合那种点击事件的场景，比如电商。

6.2.6.4 个人看法

1. 从垂类应用角度看：文章强调了垂类领域，可以尝试一下。方法需要有领域词典，用的时候需要准备一下领域词典。
2. 从方法上看，因为用到了垂类词典，感觉有点像强化 keyword 的思想，只是这个 keyword 不再是用 attention 获得，而是用词典进一步进行强化。
3. 未开源代码。一方面，论文花了三个章节讲自己的方法，方法也不是很简单。另一方面，因为牵扯到领域词典，垂类词典，是个很大的变化量，也没法做对比实验，不好说具体情况。
4. 虽然文章中很多作者自己构造的名词，通篇的感觉就是瞄准了垂类领域，引入垂类领域词典，为此设计了一套 NN 网络。又没开源代码，打的旗号又是垂类，这究竟是想让你用呢，还是不想呢? 看不懂

6.3 Phrase-level representation

如字面所示，这个方向的粒度是在短语级别，前面多是 document 级别。从 doc 中找到一些短语句子回答 Query，普遍应用在 open QA 场景，如搜索引擎。

6.3.1 DenSPI（2019，ACL）

- 论文题目：Real-time open-domain question answering with dense-sparse phrase index
- 发表会议：ACL
- 发表年份：2019
- 作者单位：华盛顿大学

- 论文链接：Real-Time Open-Domain Question Answering with Dense-Sparse Phrase Index - ACL Anthology
- 代码：https://github.com/seominjoon/denspi
    - 框架：PyTorch

6.3.1.1 一句话总结

使用基于 Bert large 的 dense model，和基于 tf-idf 的 sparse model，获得 document 中的 phrase representation，并用 fails 索引。并用同样的方法对 Query 进行 encoder，拼接了 dense 和 sparse vector，用于 QA 任务。

6.3.1.2 方法

这是好几年的论文，我简略扫读一下论文，方法的介绍会很简洁。看不懂的话，可以再参考论文。这篇论文非常简单，我花了很短的时间就弄懂了，你跟着我的思路应该也很快就弄懂了。

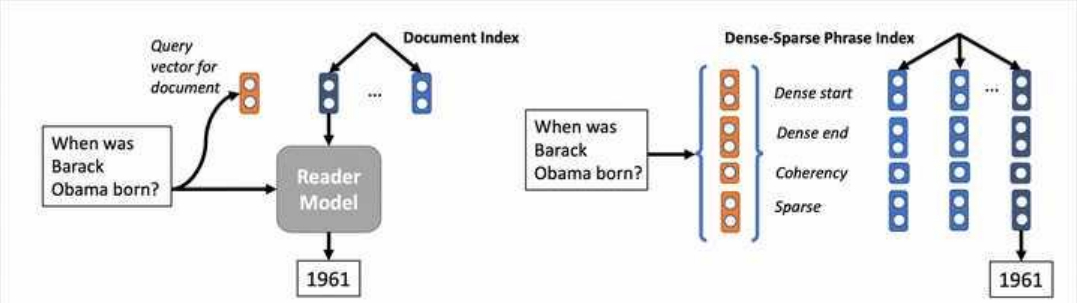


Figure 1: An illustrative comparison between a pipelined QA system, e.g. DrQA (Chen et al., 2017) (left) and our proposed Dense-Sparse Phrase Index (right) for open-domain QA, best viewed in color. Dark bi[illegible] indicate the retrieved items from the index by the query.

1. 方法是做 phrase-level。不是 sentence-level。
2. 模型有 2 个
    1. dense model
        1. 基于 Bert large
        2. dense vector 由 3 部分组成：
            1. 第一个：vector，标识一个 phrase 在 document 中的起始位置
            2. 第二个：vector，标识一个 phrase 在 document 中的结束位置
            3. 第三个：scalar value，标识起始位置到结束位置之间的内容是否连贯。举个例子：
                1. 文档库中 answer 是：“Barack Obama was the 44th President of the US. He was also a lawyer.”
                2. Query 是：What was Barack Obama’s job?
                3. 此时，回答：44th President of the US 或者 lawyer，都是对的。但是，通常我们倾向于回答 44th President of the US。所以，scalar value 的作用是避免这种情况，让 结束位置在 44th President of the US 的末尾即可。
            4. 这块的训练过程，具体参考第 5.1 章节
        3. dense vector 由 3 部分组成，但是模型并不是输入 3 个 Part，模型输出 4 个 vector。每个 token 都会有这 4 个字段。前两个用于标识 start

和 end vector，后面两个做 inner product，用于标识 scalar value。这个设计非常像 NER 的思路。此外，作者为何用点乘？不懂。不知道你怎么看？

2. sparse model
    1. 算法：2-gram-based TensorFlow-idf
3. dense 和 sparse model 的融合
    1. 没有融合，直接拼接的。具体看 3.2 章节

3. 推理时
    1. Query 的 embedding 取的是 BERT CLS 位置的输出
4. 索引工具：faiss
5. 量化
    1. 将 float32 量化成 int8

6.3.1.3 效果

好几年的方法了，没啥好说的。

6.3.1.4 个人看法

1. 看一下，感受一下思路就可以了。不要想套用，这篇文章中的方法，内存要占用 1.2T，项目上，什么家底能让你这么造？
2. 算是中规中矩的方法，很多方法都用上了，比如量化。题外说一句，你觉得作者为何量化？我的答案是：作者不得不量化，不量化，内存占用得好几个 T，这会导致论文直接被毙掉。所以，量化并不是作者主观故弄玄虚，而是水到渠成必须要做。不知道我推测出来的这个点，对你做论文有帮助没？
3. 名字起的很唬人。

6.3.2 DensePhrases（2021，ACL）

- 论文题目：Learning dense representations of phrases at scale
- 发表会议：ACL
- 发表年份：2021
- 作者单位：普林斯顿。网红作者
- 论文链接：https://aclanthology.org/2021.acl-long.518.pdf
- 代码：https://github.com/princeton-nlp/DensePhrases
    - 框架：

6.3.2.1 一句话总结

如论文题目所示，学习 phrase 的 dense representation，通过数据增强、知识蒸馏、负采样、query-side fine-tuning 提升效果。

6.3.2.2 方法

6.3.2.2.1 模型结构

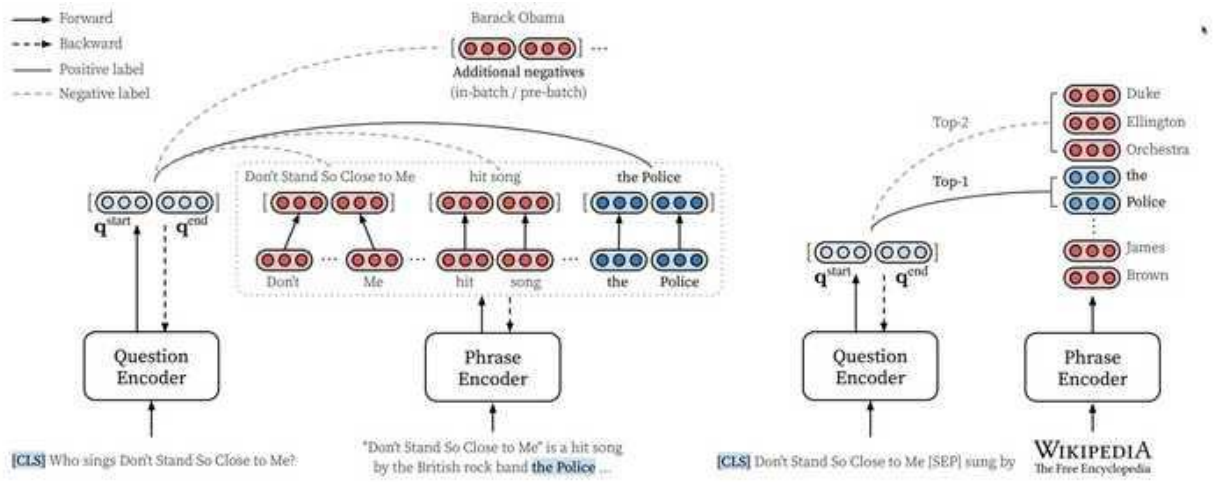


Figure 1: An overview of DensePhrases. (a) We learn dense phrase representations in a single passage (§4.1) along with in-batch and pre-batch negatives (§4.2, §4.3). (b) With the top-k retrieved phrase representations from the entire text corpus (§5), we further perform query-side fine-tuning to optimize the question encoder (§6). During inference, our model simply returns the top-1 prediction.

基座是 SpanBERT-base-cased。维度是 768。

21 年流行玩 span，作者也踩风口用了 spanBERT.

6.3.2.2.2 phrase representations

建模方法这块没啥特别的，典型的 bi-encoder 思路，两个 encoder，一个 question，一个 phrase。最后也是取的 CLS 位置的 embedding，因为是用的 span-bert，与用 BERT 的区别是最后的输出是有 start 和 end 两个 representation.

6.3.2.2.3 数据增强

所谓的数据增强，就是构造更多正例数据。下面简明扼要的通过说人话的方式说明作者的方法具体是什么。 用 T5-large model，用 passage 和 answer 微调 T5，然后对文档库抽取所有和 answer 相近的实体，将抽取到的实体和句子用在 dense-phrases 模型上，作为新的训练数据.

6.3.2.2.4 蒸馏

没啥特别的，作者也一笔带过。唯一的区别是 loss fucntion 的设计，作者这里使用的是 Kullback – Leibler divergence，KL 散度分支的方法。蒸馏另一种常用的 loss function 分支是 cross entropy。算是中规中矩的设计.

6.3.2.2.5 negative sampling

具体有两种负采样的方法。

一个是 in-batch，一个是 pre-batch。具体看下面的图。

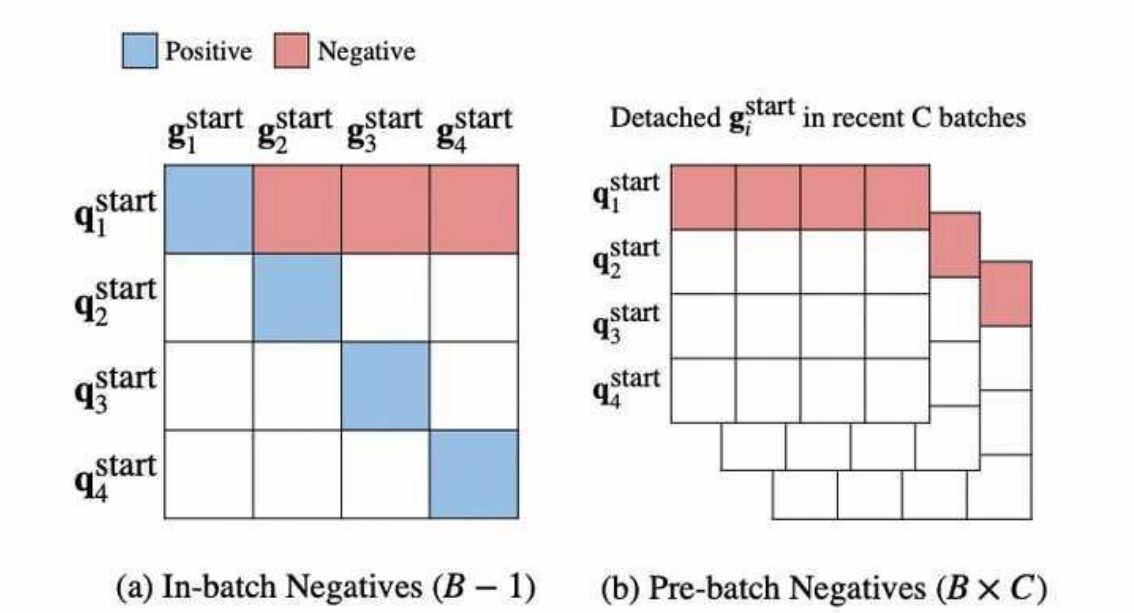


Figure 2: Two types of negative samples for the first batch item ($\mathbf{q}_1^{\text{start}}$) in a mini-batch of size $B = 4$ and $C = 3$. Note that the negative samples for the end representations ($\mathbf{q}_i^{\text{end}}$) are obtained in a similar manner. See §4.2 and §4.3 for more details.

1. in-batch sampling.
    1. 在一个 Batch 内，对于 i-th 行的 sample，当前行的 score 算正例，其他行对当前行来说算负例，由此计算 loss。上面的图，看左边一个，一看就懂了
    2. loss 函数的设计

ples. We compute $\mathbf{S}^{\text{start}} = \mathbf{Q}^{\text{start}}\mathbf{G}^{\text{start}\intercal}$ and $\mathbf{S}^{\text{end}} = \mathbf{Q}^{\text{end}}\mathbf{G}^{\text{end}\intercal}$ and the i-th row of $\mathbf{S}^{\text{start}}$ and $\mathbf{S}^{\text{end}}$ return B scores each, including a positive score and $B-1$ negative scores: $s_1^{\text{start}}, \ldots, s_B^{\text{start}}$ and $s_1^{\text{end}}, \ldots, s_B^{\text{end}}$. Similar to Eq. (5), we can compute the loss function for the i-th example as:

$$
\begin{aligned}
P_i^{\text{start\_ib}} &= \text{softmax}(s_1^{\text{start}}, \ldots, s_B^{\text{start}}), \\
P_i^{\text{end\_ib}} &= \text{softmax}(s_1^{\text{end}}, \ldots, s_B^{\text{end}}), \\
\mathcal{L}_{\text{neg}} &= -\frac{\log P_i^{\text{start\_ib}} + \log P_i^{\text{end\_ib}}}{2},
\end{aligned} \tag{8}
$$

We also attempted using non-gold phrases from other passages as negatives but did not find a meaningful improvement.

这个图的底面有句话很有意思，non-gold phrases from other passages as negatives 没有效果，interesting。

这个方法要求 batch size 不会很小，落地的时候，机器的配置可能需要留意

2. pre-batch sampling

使用 FIFO 队列，缓存前 C 个 batch，用以提供更多负例。这里还有一个点是：pre-batch negatives 是不需要反向传播梯度的

这两种方法，都不好以今天的视角反过来回头看的方式评价，因为我在很多篇论文中都看到这两种方法，不好说是作者自己想出来的，还是约定俗成的方法。

6.3.2.2.6 损失函数

$$
\mathcal{L} = \lambda_1 \mathcal{L}_{\text{single}} + \lambda_2 \mathcal{L}_{\text{distill}} + \lambda_3 \mathcal{L}_{\text{neg}}, \tag{9}
$$

where $\lambda_1, \lambda_2, \lambda_3$ determine the importance of each loss term. We found that $\lambda_1 = 1, \lambda_2 = 2$, and $\lambda_3 = 4$ works well in practice. See Table 5 and Table 6

注意看这个表：作者给了各个参数的最佳实践。

一般来说，损失函数的设计是最核心的工作之一。作者这里的设计中规中矩，并没有特别的内容。整篇论文都没有给出 loss 曲线，估计这个模型不好训练，各种方法都有，loss 下降的应该不快。。

6.3.2.2.7 Query-side Fine-tuning

这个章节得好好说说，不然论文的方法你就完全理解错了。

论文的方法是有两个 encoder 模型，先训练 phrase encoder，等 phrase encoder 训练完成了之后，再通过 query-side fine-tuning 固定 phrase encoder，微调 query encoder。不知道你理解的和我说的是否一致？如果不一致，你就读错了。

论文写的，初看觉得名字很唬人，其实就是召回 Top 100 后，只更新 query encoder。作者说这个方法对效果有很大提升，估计是通过 phrase 校准 query encoder，是 query 的语义和 phrase 的语义更匹配，从而带来最终效果更优。说人话就是：phrase 把苹果量化成数字 3，让 query 也学成接近 3

6.3.2.3 效果

实验做的很全，好多个测试集。不过，在 open QA 场景，并不是所有测试集都很好。另外作者的方法体现出 10K 训练集比 5K 效果好的情况，并且，在阅读理解任务上效果好，推测：作者的方法更多适用于 document 场景。段文本效果应该不佳。

6.3.2.4 个人看法

1. 方法比 DenSPI 好很多，有很多眼前一亮的方法
2. 从落地角度看，论文中有个数据增强的方法，但这个数据增强的方法很重，落地的时候，有难度。第二，如果你增强的不好，dense-phrases 的效果是不是也会很不好？
3. 论文中的技法太多，又是数据增强，又是蒸馏，还有 negative sample，不能说这些方法没用，但总有一种差生文具多的想法？狗头 :)
4. 方法很重，两套 encoder，Indexing 需要的资源应该也不少，很重量级的方法
5. 作者给出的数据显示训练耗时已经很多了。这么重的方法，推理耗时不会少吧？落地又一难题

6.3.3 Phrase Retrieval Learns Passage Retrieval, Too（EMNLP，2021）

- 论文题目：Phrase Retrieval Learns Passage Retrieval, Too
- 发表会议：EMNLP
- 发表年份：2021
- 作者单位：普林斯顿
- 论文链接：https://arxiv.org/pdf/2109.08133.pdf
- 代码：https://github.com/princeton-nlp/DensePhrases
    - 框架：

6.3.3.1 一句话总结

用 phrase retrieval 的方式做 passage retrieval，passage retrieval 是取的 phrase retrieval 的最大值。

一个问题，检验你是否真的读懂了论文：论文的方法召回最终返回的是 passage 还是 phrase？ 答案是：passage。论文中说的 phrase-level passage retrieval，是说以 phrase 粒度计算 score，因为 phrase 隶属于一个 passage，取最大的 phrase score 作为这个 passage 的 score。具体可参考第 3 章节。

6.3.3.2 方法

6.3.3.2.1 模型 1：document-side 模型

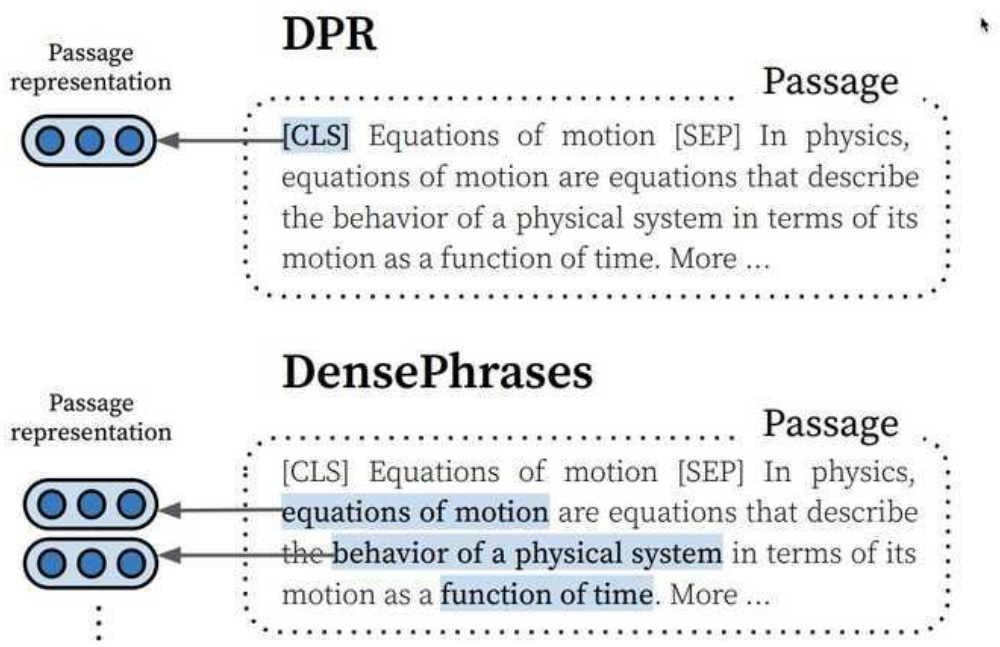


Figure 1: Comparison of passage representations from DPR (Karpukhin et al., 2020) and DensePhrases (Lee et al., 2021). Unlike using a single vector for each passage, DensePhrases represents each passage with multiple phrase vectors and the score of a passage can be computed by the maximum score of phrases within it.

1. 具体的方法在一句话总结中已经描述了，没有什么特别复杂的。
2. 损失函数：
    1. 不知道你看了下面的公式，会怎么想？是不是似曾相识的 loss 设计？

$$\mathcal{L} = -\log \frac{e^{f(x^+,q)}}{e^{f(x^+,q)} + \sum_{x^- \in \mathcal{X}^-} e^{f(x^-,q}}$$

3. 负采样

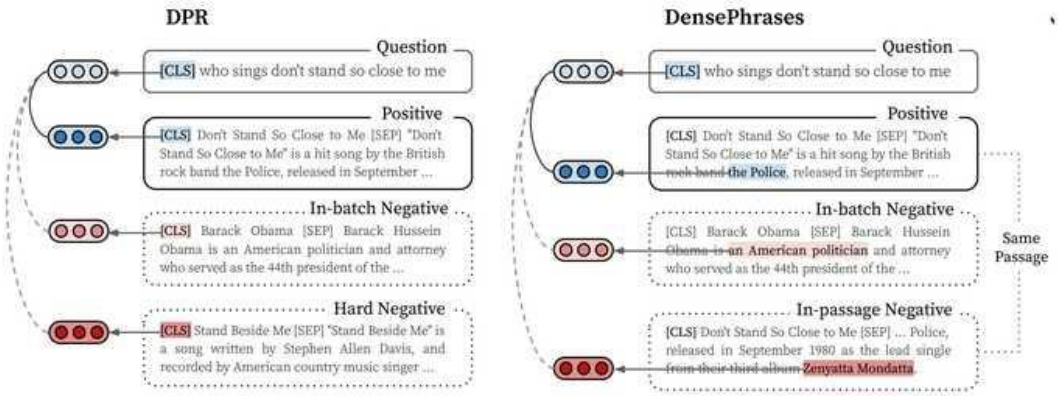


Figure 2: Comparison of training objectives of DPR and DensePhrases. While both models use in-batch negatives, DensePhrases use in-passage negatives (phrases) compared to BM25 hard-negative passages in DPR. Note that each phrase in DensePhrases can directly serve as an answer to open-domain questions.

1. 对于召回模型来说，比拼的重点都是在正负例构造上。作者为了丰富文档工作量，在这块也设计了正负例的方法，个人感觉这才是本文的核心点。一共 3 种：

1. 方法一：In-batch negatives

在一个 batch 中，将剩下的 example 都当作负例。这块为了爆内存，做了采样。作者也说了，这是 typical 方法，不算创新。

2. 方法二：Hard negatives

基于 BM25 的方法。具体是：找和 question BM25 分数很高，但又不包含 answer 的 passage 文本，将其作为负例。

3. 方法三：In-passage negatives

动机是：一个 passage 内的 phrase，大多是在说一个相同的 topic，并且有相同 context，对其做负采样，会得到很好负样本。三个负采样方法，这个是唯一我觉得创新的点。三个负采样方法，这个是唯一我觉得创新的点。

仔细看 4.2 章节的实现，作者也通过实验论证这个方法特别好，好到其他两个负采样方法都没有额外的收益了。

这个方法要再论证看看，感觉作者的数据不 solid，应该再对比一下：在其他两个方法的基础上再加上 in-passage negatives 的效果。

6.3.3.2.2 模型 2: query-side 模型

不要看晕了，这个 query-side 其实是说用在推理阶段的 query embedding.

动机是：第 5 章中说了 DensePhrases 方法的缺陷：如果检索任务更侧重于找到与查询主题紧密相关的文档，而不是寻找包含具体答案的特定短语或句子，DensePhrases 可能就不是最佳选择。比如实体链接（根据一个词汇链接到某篇文章）

损失函数：

$$\mathcal{L}_{\text{doc}} = -\log \frac{\sum_{s \in \tilde{\mathcal{S}}(q), d(s) \in \mathcal{D}^*} e^{f(s,q)}}{\sum_{s \in \tilde{\mathcal{S}}(q)} e^{f(s,q)}}, \quad (5)$$

6.3.3.2.3 量化

量化方法：Optimized Product Quantization (Ge et al., 2013). 落地的时候可以参考这个量化方法。

6.3.3.3 效果

phrase retrieval 不需要任何 re-training 就可以比 passage retrievers 效果好 3-5%

6.3.3.4 实践 Tips

一个 passage 用 28 个 vector 表示，效果最好，至少应该在 9 个以上。

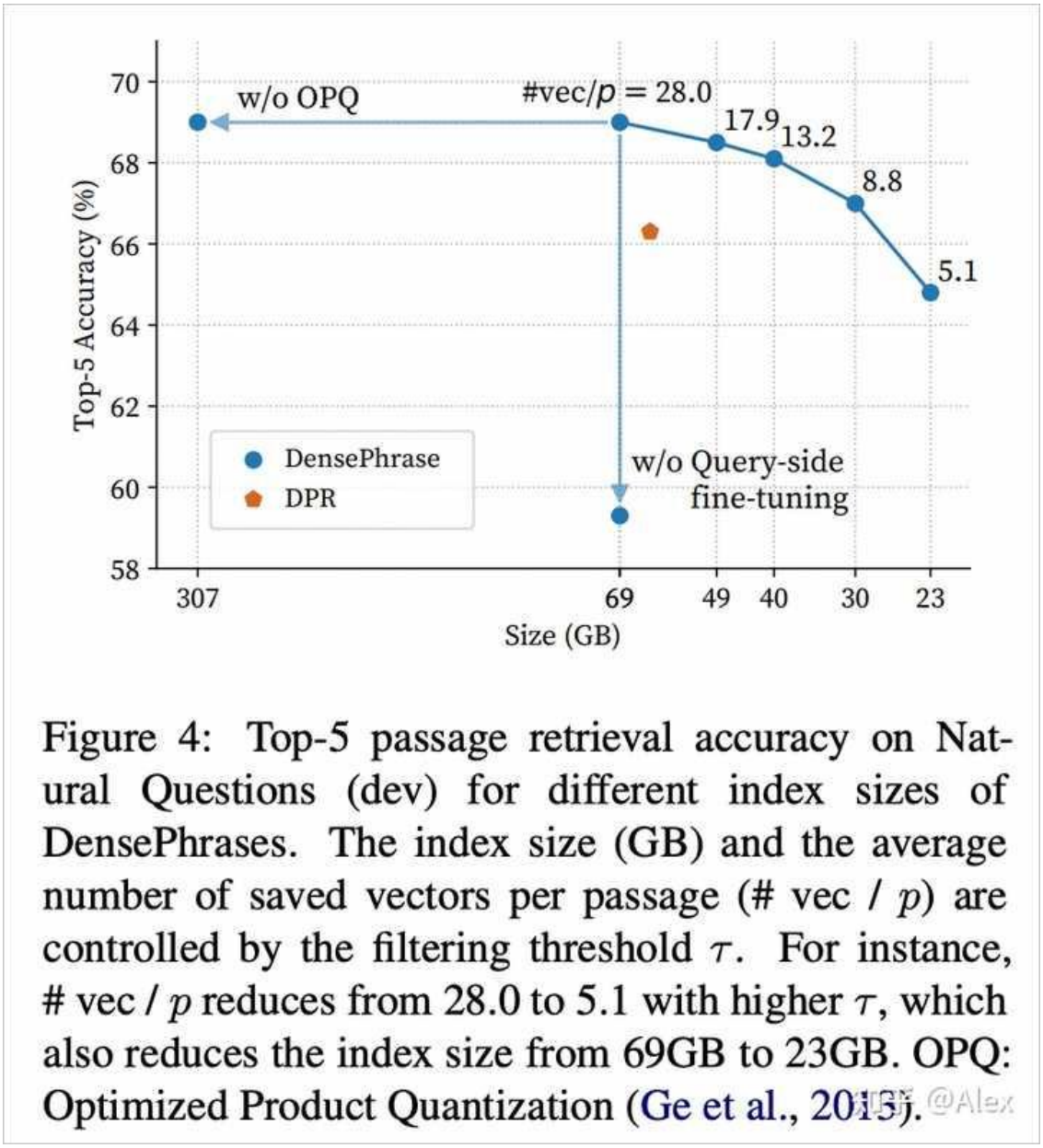


Figure 4: Top-5 passage retrieval accuracy on Natural Questions (dev) for different index sizes of DensePhrases. The index size (GB) and the average number of saved vectors per passage (# vec / p) are controlled by the filtering threshold τ. For instance, # vec / p reduces from 28.0 to 5.1 with higher τ, which also reduces the index size from 69GB to 23GB. OPQ: Optimized Product Quantization (Ge et al., 2013).

6.3.3.5 个人看法

1. 没看出来论文有啥特别的创新点。论文只是用 phrase-level 来做 passage retrieval，这只是句子和段落级别的区别，这个创新点足够扎实?
2. 没啥特别的亮点。文中的方法，在我没有看这篇论文的时候，那些 negative sample 方法我都已经用了
3. 论文从创新角度看，并没有太多创新，但很多工作量，估计也是因为这个才中的 EMNLP

6.4 组合方法

将 document-level、passage-level、phrase-level 进行组合，或者是多个模型组合，以获得不同粒度的语义信息。

6.4.1 Dense hierarchical retrieval for open-domain question answering（EMNLP findings，2021）

- 论文题目：Dense hierarchical retrieval for open-domain question answering
- 发表会议：EMNLP findings
- 发表年份：2021

- 作者单位：salesforce
- 论文链接：https://aclanthology.org/2021.findings-emnlp.19.pdf
- 代码：yeliu918/DHR: This is the repository of the Dense Hierarchical Retrieval for Open-Domain Question Answering (github.com)
    - 框架：PyTorch

6.4.1.1 一句话总结

Dense retrieval 通常会将 document 切分成句子，这种会失去上下文语义。作者提出先通过 document- level retrieval 定位相关的 document、再通过 passage-level retrieval 定位相关 passages，最后通过 ranker 返回 Top-K passages 的方法。

6.4.1.2 模型结构

wikipedia 的文章结构：

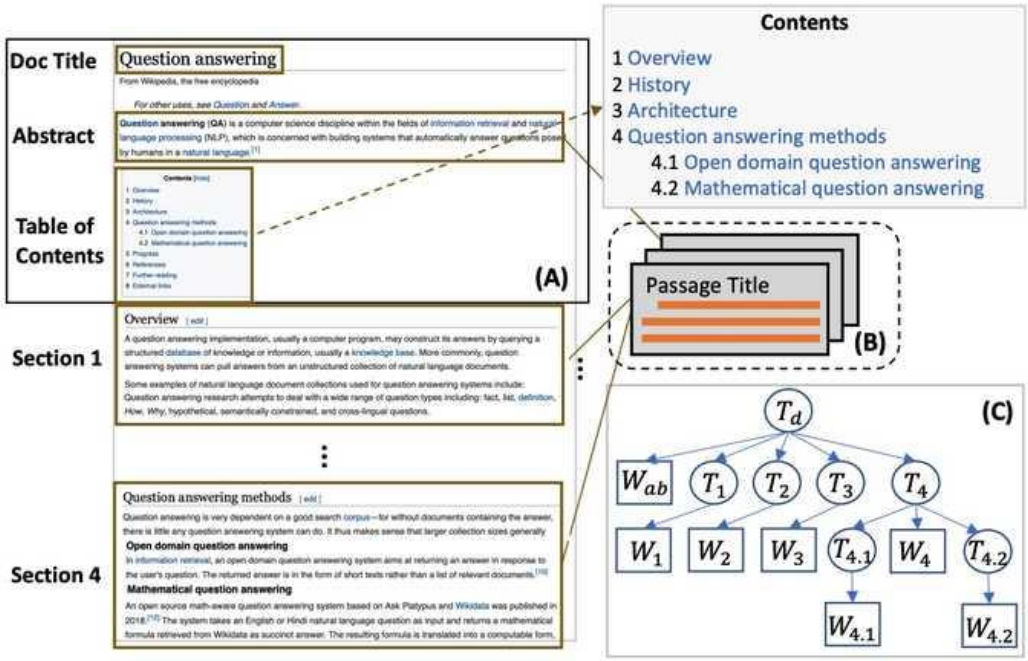


Figure 2: An illustration of a typical Wikipedia page. (A) Document representation. (B) Passage representations. (C) Hierarchical title structure, i.e., title tree.

作者提出的方法：

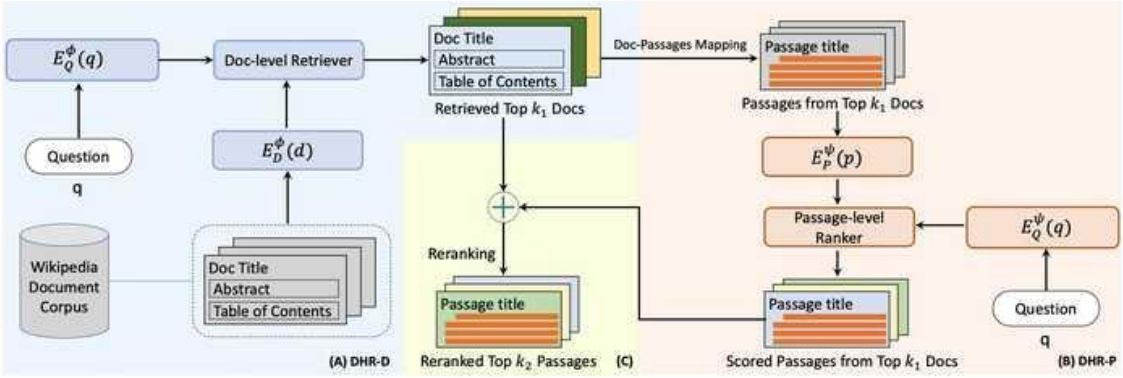


Figure 3: An overview of DHR. During inference, document-level retriever first retrieves top-k_1 documents (as shown in (A)). Then, passage-level retriever scores the passages in top-k_1 documents (as shown in (B)). At last, DHR reranks passages based on two levels of relevance scores and return top-k_2 passages (as shown in (C)).

仔细读下面的标题或者我刚才的总结，就知道作者的方法了。

论文除此之外，还增加了一些负采样的方法。这里就不介绍了，基本都看过，大同小异。

6.4.1.3 效果

这块自己看论文就好，对比实验很少，不做评价。

6.4.1.4 个人看法

1. 方案很重，两个 retrieval、一个 rerank，但是最终只是干的召回这个环节。这么多的模块，别人的方法都直接做完了排序。

6.4.2 DrBoost（NAACL，2022）

- 论文题目：Boosted Dense Retriever
- 发表会议：NAACL
- 发表年份：2022
- 作者单位：脸书
- 论文链接：https://aclanthology.org/2022.naacl-main.226.pdf
- 代码：https://github.com/facebookresearch/drboos
    - 框架：

6.4.2.1 一句话总结

集成学习的思想：集成了多个 dense retriever。因为作者用的 embedding 小四倍，反而效率更高，还可以通过轻量化进一步提升性能。

6.4.2.2 方法

这篇不详解了。思路挺好的，集成的思想。

6.4.2.3 效果

对比的方法、数据集很少。估计也知道只能有选择性的对比。

6.4.3 Multi-Task Dense Retrieval via Model Uncertainty Fusion for Open-Domain Question Answering

- 论文题目：Multi-Task Dense Retrieval via Model Uncertainty Fusion for Open-Domain Question Answering
- 发表会议：EMNLP findings
- 发表年份：2021
- 作者单位：滑铁卢大学
- 论文链接：https://aclanthology.org/2021.findings-emnlp.26.pdf
- 代码：alexlimh/DPR_MUF (github.com)
    - 框架：

6.4.3.1 一句话总结

将多个 DPR 模型融合在一起，以达到更强的效果。

6.4.3.2 方法

仔细看标题就懂了。有兴趣的读者可以阅读一下 Fusion 这个章节，因为效果没拉开差距，我就不做介绍了。

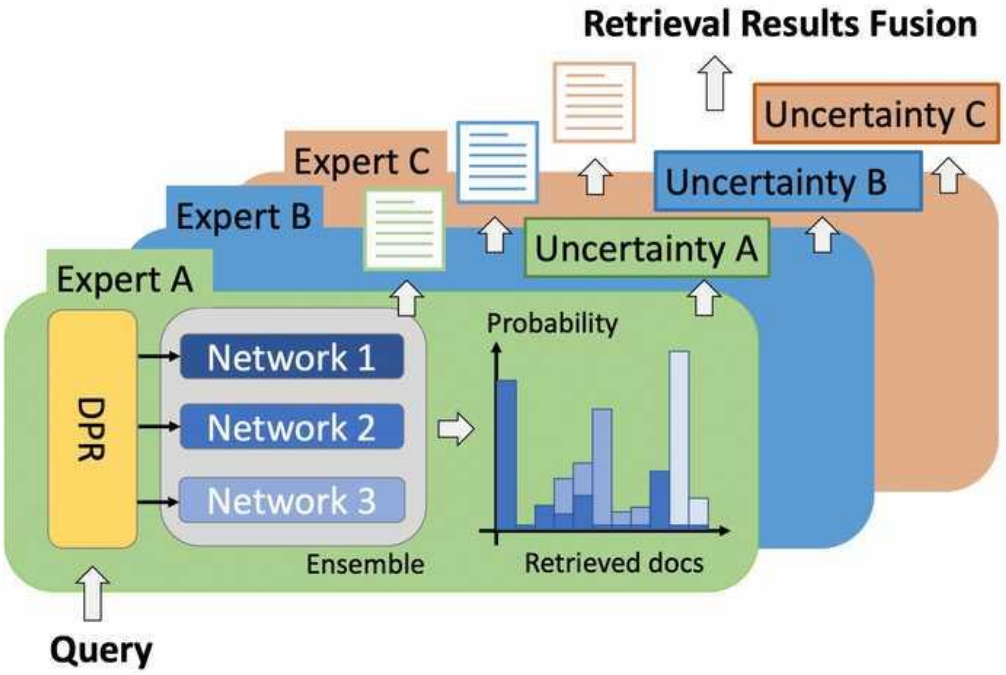


Figure 1: An illustration of model uncertainty fusion of 3 DPR experts, each with an ensemble of 3 fully-connected neural networks. Given a query, each DPR expert first retrieves top-k documents, followed by the uncertainty estimation using the corresponding ensemble. The weighted sum of predictions is then used to rerank the union of the retrieved documents.

6.4.3.3 效果

对比实验有欠缺，总是在和 DPR 进行对比。但是最新的方法已经不是 DPR 了。abstract 中说的 10%提升更具有迷惑性，其只是和 BM25 对比提升的。部分数据集，效果提升非常有限，只有 0.2%。

6.4.3.4 个人看法

1. 刚才已经从效果层面给出了我自己的分析；
2. 与 MOE 不同的是，多个 DPR 模型都需要激活，计算量过大。

6.5 Lightweight architectures

思想：learn lightweight representations。

注意：这个方向是为了轻量化设备而生，带来的缺陷是：模型效果会差一些，和大模型领域一样，目前还没有能超过对标模型的。

这个方向值得注意一下，很好的方向，好多年的汽车、芯片发展，最后都是低功耗的技术存活了下来，最近大模型也开始逐步往“节能”、“轻量化”方向走，也许这个是一个有学术潜力的方向。
这个方向最常用的量化方法是：product quantization。这里当作背景知识讲解 product quantization，不然这个方向的论文看起来会比较懵。

6.5.1 背景知识：Product Quantization

下面我们来详细讲解 Product Quantization 是如何量化的。这里仅给出白话文版本，有兴趣的读者可以参考论文：Product Quantization for Nearest Neighbor Search (hal.science)
具体的步骤有：

1. Approximate Score Function
    1. 功能是：构造一个新的打分函数。这个打分函数会将 document representation 进行量化，基于量化后的 representation 计算相似度。
2. Quantizing Document Embeddings
    1. 定义 M 个质心（可以理解成聚类中的 M 个簇），质心叫做 centroid embedding。
    2. 对一个 document embedding，基于这 M 个 centroid embedding，将其拆分成 M 个 sub embedding，最后合起来的新的 embedding 就是其对应的 PQ centroid embedding;
3. Optimization Objective
    1. 上面一个有个将 document embedding 拆分的过程，这个理解成映射函数，或者理解成索引。这个类似于映射的函数叫做 Index Assignments $\phi_i (d) $. 这一步是通过 MSE loss 优化两个 embedding。
4. Search Procedure
    1. 同样，将 query embedding 拆分成 M 个 sub-vectors，每个的维度是 D/M;
    2. 然后计算每个 query sub-vector 和 PQ centroid embeddings 的相似度;
    3. 这个过程中会用 lookup table 加速;
5. Time Complexity
    1. 耗时最大的主要是 search procedure 中计算相似度。时间复杂度是 O(NM + Nlogn)。N 是整个文档库的个数，M 是 central 的个数。
6. Index Size
    1. PQ 不会存储全部的 document embedding，而是存储 PQ centroid embedding 和 Index assignment。
    2. central 的个数是 M 个，每个 central 中有 embedding 的个数是 K=D/M 个，通常 K 不会超过 256 个，原因是 Index assignment $\phi_i (d) $ 这样才能用 1 个 bite 表示.

6.5.2 Query Embedding Pruning for Dense Retrieval（CIKM，2021）

- 论文题目：Query Embedding Pruning for Dense Retrieval
- 发表会议：CIKM
- 发表年份：2021
- 作者单位：University of Pisa。QS 排名 382
- 论文链接：Query Embedding Pruning for Dense Retrieval (arxiv.org)
- 代码：无
    - 框架：

6.5.2.1 一句话总结

前文说过 multi-representation 这个方向。本文是对这个方向的优化。具体是：Query encoder 后最后一层的输出有很多 embedding，本文是不需要取全部的 embedding，而是取部分几个。这句话看不懂的同学记得多读几遍。

6.5.2.2 方法

最开始我以为这个工作是类似于量化的工作。一直看不懂文章说的 Query embedding 从 32 个减少到 3 个，于是一怒之下看了整篇论文，原来是这个意思：query encoder 后，最后一层的输出，通常有 32 个位置，通常我们都是取 CLS 位置的输出，也有用 MEAN 或者 MAX pooling 的，作者的意思是，我不用 32 个，我只取其中部分几个位置，所以 abstract 中的从 32 减少到 3 个，就是这个含义。最开始我还以为是 embedding 单纯减少到 3 个浮点数了，我心想这么牛么？读了论文才知道是怎么回事，不知道你怎么看作者的这个方法。。

先介绍一下背景知识：

Sparse retrieval，比如 BM25，是用的 extract match。就是计算相似度 score 的时候，会全量 documents 扫一遍。Dense retrieval 用的是 ANN（approximate nearest neighbor），它是基于某种距离函数，比如 consine，找到最相似的 K 个返回即可，这个方法不会全“表”扫描。

作者优化的工作点是：Query 会有多个 representation，某些 representation 基于上面 dense retrieval 的过程计算出来的结果，并不会合并到最终的 Top K。这些 representation 能不能去掉？

重点来了，作者是怎么取“某些 representation”的？答案就一句话：就是根据 TF-IDF 算法中的 IDF。原文中用的 ICF，我这里为了方便理解说的 IDF。有兴趣的读者可以再查查两者的区别，作者也小字说了排序结果一样。再说人话就是：根据 query token 在文档中的频数。query token 就是 tokenizer 之后的结果。作者写了好多页，结果就一句话。读到这里，正常来说，这里还牵扯到一个数量的问题，但作者的方法竟然是试出来的，正好对应到作者 abstract 中的 32 减少到 3.

6.5.2.3 效果

有兴趣的读者自己看好了。

6.5.2.4 个人看法

只一句话，非常好奇这论文当初审稿人的意见是啥？

6.5.3 ColBERTv2（NAACL，2022）

- 论文题目：ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction
- 发表会议：NAACL
- 发表年份：2022
- 作者单位：斯坦福
- 论文链接：https://aclanthology.org/2022.naacl-main.272.pdf
- 代码：stanford-futuredata/ColBERT: ColBERT: state-of-the-art neural search (SIGIR'20, TACL'21, NeurIPS'21, NAACL'22, CIKM'22, ACL'23, EMNLP'23) (github.com)
    - 框架：

6.5.3.1 一句话总结

ColBERT 中的 late interaction 会为每个 token 生成 multi-vector representation，比较耗内存。本篇通过优化正负例训练数据构造进一步提升 ColBERT 的效果，同时，通过 residual representation 方法减少内存消耗。

6.5.3.2 方法

6.5.3.2.1 去噪监督训练

论文中原始的说法是：Denoised training with hard negatives。不知道这个是作者自己创造的名字么？也没有对应的引用。从作者的方法看，其实就是作者想了一个构造难负例的方法。

言归正传，讲解具体是怎么做的。

1. 正例：人工标注；
2. 负例：
    1. 使用基于 BM25 算法的段落 retriever，先召回 top-k 个段落；
    2. re-rank 模型进行排序。
        1. 模型是：22M-parameter MiniLM (Wang et al., 2020) cross-encoder trained with distillation by Thakur et al. (2021)。
        2. 对 K 个段落排序，选出 64 个段落
    3. 因为这里面涉及到 retrieval 和 re-rank 两个模型，作者采用 join 的方式训练，具体的细节可以参考：RocketQAv2: A Joint Training Method for Dense Passage Retrieval and Passage Re-ranking (arxiv.org)

6.5.3.2.2 Residual representation

这篇论文的动机是：减少 ColBERT 的内存消耗，residual representation 就是解决这个问题的。具体的：

作者先假设 ColBERT 得到的 representation 已经能够清晰的进行 cluster 了，就是很好的语义归类到不同的簇上，那减少内存的消耗就是：簇，就好比是圆，我不需要像以往那样记录簇上的每个点，而是：我记录圆心（就是C_t），然后记录每个点到这个C_t 的半径（就是 r）。听我这样讲解，是不是对圆的理解又近了一丢丢 :) 这个思想的 LLM 领域的量化也有出现。有兴趣的同学可以看原文，量化的路数真的很像。。。

听完我上面白话的讲解，下面对照着作者的方法，再理解作者说的 one or two bits 就理解了说的是什么了。

1. 第一步：找到近似表示：每个向量 v 被表示成：用 v-最靠近 v 的中心 c，记作 r. 即：r = v-C
2. 第二步：对 r 进行量化成 1 或 2 bit，记作r_{hat}，r_{hat} 就是作者说的 residual representation。b-bit encoding of n-dimensional vectors needs ceil(log|C|) + bn bits per vector.
3. 第三步：推理时，用中心 $c+r_{hat}$ 进行近似。

上面的 C 等于 embedding 的维度开平方根，感觉是从 Bert 中学来的。后续会用 k-means 对 representation 进行聚类，簇的个数设置为 C。

作者这里起了 residual representation，residual 这个词一直会让我联想到图像领域的 residual，仔细一看，完全不是一回事，这不是造词艺术么。。。

关于 LoTTE，有兴趣的读者自己阅读好了，我一般对另起炉灶这种做法不感兴趣。只有那种专门发布 benchmark 的做法才是一个好的方法，如果作者真的有料，应该专题发榜，让后续的学者过来打榜，而不是自己既发榜又打榜。

6.5.3.3 效果

具体的效果自己看好了。

不太懂很多论文都是放 dev 上的效果，这有用么？

6.5.3.4 个人看法

1. 从方法看，作者的工作更多体现在“量化”，就是减少内存消耗上。
2. 作者虽然说可以减少内存的消耗，但是为了得到最后 ColbertV2 的少内存消耗，中间过程的内存消耗并不少，比如 k-means。

6.5.4 Jointly Optimizing Query Encoder and Product Quantization to Improve Retrieval Performance（CIKM，2021）

- 论文题目：Jointly Optimizing Query Encoder and Product Quantization to Improve Retrieval Performance
- 发表会议：CIKM
- 发表年份：2021
- 作者单位：中科院
- 论文链接：https://arxiv.org/pdf/2108.00644.pdf
- 代码：jingtaozhan/JPQ: CIKM'21: JPQ substantially improves the efficiency of Dense Retrieval with 30x compression ratio, 10x CPU speedup and 2x GPU speedup. (github.com)
    - 框架：PyTorch

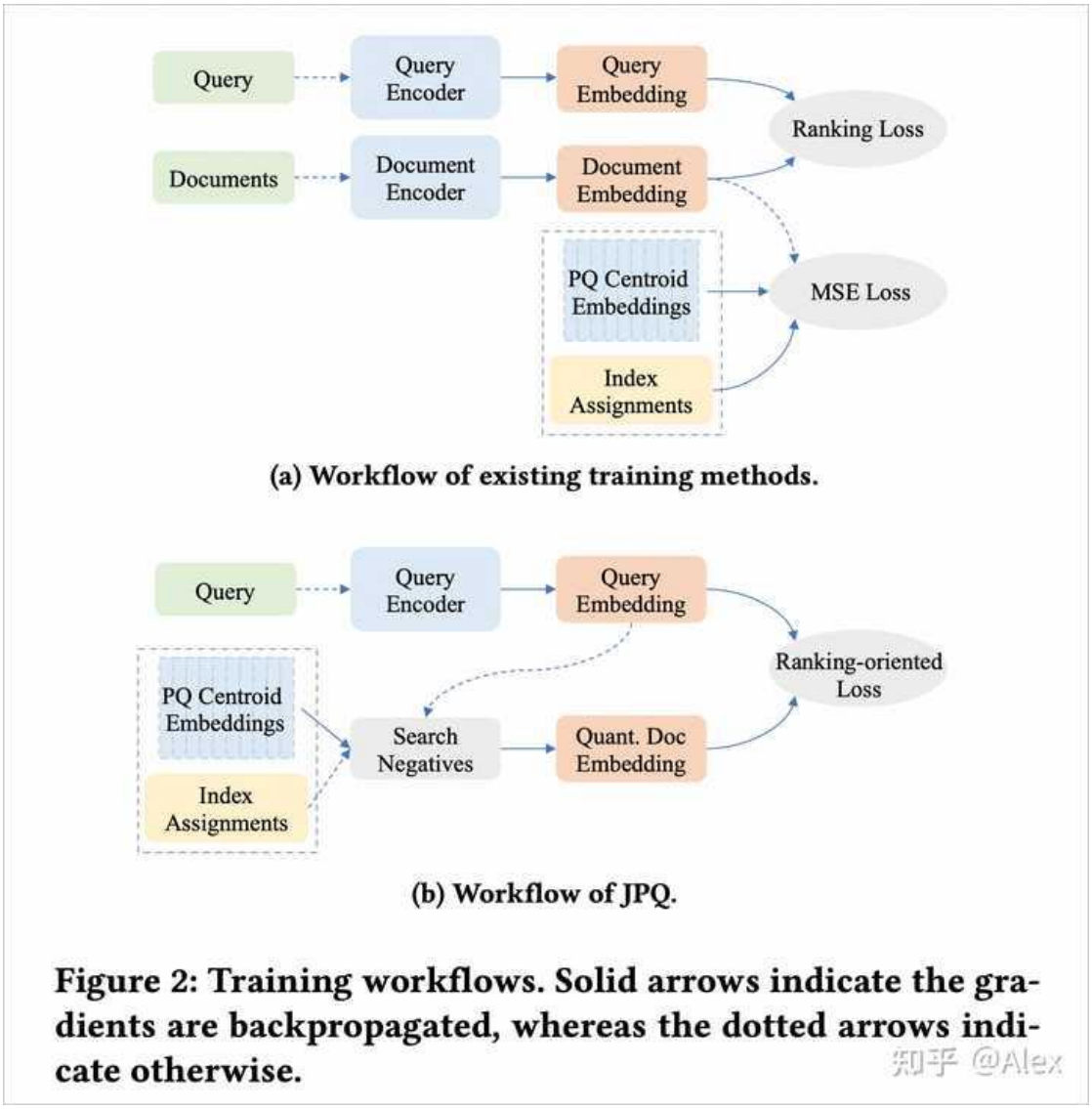
6.5.4.1 一句话总结

文本召回轻量化方面的工作。将 document embedding 进行量化，并基于 query 构造难负例，结合 ranking-oriented loss 训练，使之在不降低模型效果的情况下提升 representation 压缩率。

6.5.4.2 方法

6.5.4.2.1 模型结构

方法的具体流程，如下：



仔细对比一下，你发现了哪些点？

1. 从简化流程看，并没有简化；
2. 作者设计的是 ranking-oriented loss，不再是之前 ranking loss 和 MSE loss 混合在一起。原文作者没有说的很明白，我猜作者的真实意图是：一致性。之前的流程是需要两个 loss，训练时的 MSE loss 和最终使用的 ranking loss 不一致，作者的方法是只有一个 loss，从而提升效果。
3. 作者这里用了 search negatives，这个点很常见了。

6.5.4.2.2 Ranking-oriented loss

首先，作者这里用 ranking loss 会有非常大的歧义，很容易让别人联想到 cross-encoder 中的 loss。说完这个可能的误区之后，再说作者的 ranking-oriented loss 是怎么回事：
上面的图 a 中，作者画的 ranking loss 其实就是 bi-encoder 中常用的判断 pair 是否相似的 loss。作者原文也给出了公式：

$$\ell(s(q,d^+),s(q,d^-)) \quad (13)$$

因为这 score 函数，其实在 ANNS 算法中并不会使用，所以这里存在一个 Gap，即：训练时的 score 和 最后推理时用的 score 函数不一致。又因为作者的方法是对 document representation 进行了量化，损失函数也不同，所以也会存在这个问题。为此作者才提出了 ranking-oriented loss。
说了这么多，最后上作者的 loss 公式：

$$\ell(s^\dagger(q,d^+),s^\dagger(q,d^-)) \quad (16)$$

其中的 S*(q,d)S^* (q, d) 是 query representation 和 quantized document representation 的相似度。
仔细看，你感觉有啥区别？我觉得创新很小。。

6.5.4.2.3 PQ Centroid Optimization

这个章节，有兴趣的读者自己阅读好了，我看的不是很懂，有一些细节写的过于粗糙，这里只给出我的阅读理解和疑惑，有兴趣的读者欢迎交流。
这个章节，作者的行文思路依然是遇到问题解决问题的演进思路。这个点很重要，是在说论文的演化路径。说完学术的求解路径，在说术的层面：即作者是如何解决问题的。
问题：作者说 PQ 无法使用梯度下降求解。具体是和 Index Assignments 有关，Index assignments 用于选择 PQ centroid embeddings，这个环节无法使用梯度下降优化。
解决方案是：一言以蔽之，设计了类似于 relu 的打分函数，就是 query 和 document 的相似度函数，也可以说是 loss。具体参看公式 18 和 19.
我看不懂的点在于：作者说 JPQ 只会训练一小部分 crucial number of PQ parameters，PQ centroid embeddings。从哪儿得出来只训练只训练一小部分的，又为何说能够解决 overfitting?

6.5.4.2.4 hard negatives samples

一言以蔽之，从召回的 Top K 个中，将不和 Query 相似的作为负例。
原文并没有给出判断不相似的依据，这儿写的不好。

6.5.4.3 效果

效果很好。

6.5.4.4 个人看法

1. 作者给出的数据，都挺好的。需要结合源码进一步确认细节。

7 多语言 Dense Retrieval

本章节是讨论关于多语言这个方向。多语言这个方向具体有 2 种演化路径：

1. 第一种是：多语言的 PTM。比如：mBERT、mT5、XLM-R
2. 第二种是：多语言数据的生成与整合，以此训练 multi-lingual text embeddings. 比如：MIRACL, mMARCO, Mr. TyDi, MKQA。稍后细读的论文 BGE-M3 就是属于这个方向。

智源这两年出了好几个工作，社区对智源关注越来越多。智源出了 text embedding 模型之后，很多人开始在项目中应用 BGE。BGE 有很多个模型，这里简要梳理一下各个概念。BGE 是 BAAI General Embeddings 的简称，智源发布了好几个 embedding 模型，BGE embedding 现在变成了一整个 text embedding 工具集合。

原生的 BGE 是使用 retroMAE 架构，retrolMAE 是一个基于 auto-encoder 架构的模型. retrolMAE 的论文是 RetroMAE: Pre-Training Retrieval-oriented Language Models Via Masked Auto-Encoder (arxiv.org)

代码是 staoxiao/RetroMAE: Codebase for RetroMAE and beyond. (github.com)。

这里简要说一下 retrolMAE 的结构：

1. encoder 是 BERT;
2. decoder 是只有一层的 transformer;
3. mask 的比例，在 encoder 中是 15 到 30%，decoder 中是 50%到 70%;

BGE 模型一共有 3 个版本：small (24M), base (102M), and large (326M)。

BGE 对应的论文是：C-Pack: Packed Resources For General Chinese Embeddings (arxiv.org)

代码：FlagEmbedding/FlagEmbedding/baai_general_embedding/README.md at master · FlagOpen/FlagEmbedding (github.com)

这里选择 M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation 这篇论文进行详读。

7.1 M3-Embedding （2024，ACL 在投）

- 论文题目：M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation
- 发表会议：ACL 在投
- 发表年份：2024
- 作者单位：智源
- 论文链接：BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation (arxiv.org)
- 代码：FlagEmbedding/FlagEmbedding/BGE_M3 at master · FlagOpen/FlagEmbedding (github.com)
    - 框架：PyTorch

7.1.1 一句话总结

基于 XLM-RoBERTa、改进的 InfoNCE loss 训练而成的一个多语言、支持三种不同粒度（dense retrieval, multi-vector retrieval, and sparse retrieval）、支持最长不超过 8K 输入的 text embedding。

7.1.2 背景知识

对比学习、以及 InfoNCE loss

3.1.1 Contrastive learning

Contrastive learning is an approach that relies on the fact that every document is, in some way, unique. This signal is the only information available in the absence of manual supervision. A contrastive loss is used to learn by discriminating between documents. This loss compares either positive (from the same document) or negative (from different documents) pairs of document representations. Formally, given a query q with an associated positive document k_+, and a pool of negative documents $(k_i)_{i=1..K}$, the contrastive InfoNCE loss is defined as:

$$\mathcal{L}(q, k_+) = -\frac{\exp(s(q, k_+)/\tau)}{\exp(s(q, k_+)/\tau) + \sum_{i=1}^{K} \exp(s(q, k_i)/\tau)}, \quad (1)$$

where τ is a temperature parameter. This loss encourages positive pairs to have high scores and negative pairs to have low scores. Another interpretation of this loss function is the following: given the query representation q, the goal is to recover, or retrieve, the representation k_+ corresponding to the positive document, among all the negatives k_i. In the following, we refer to the left-hand side representations in the score as queries and the right-hand side representations as keys.

7.1.3 方法

7.1.3.1 模型结构

基座模型是：XLM-RoBERTa。这是 roBERTa 模型的多语言版本。复习：roBERTa 去掉了 NSP 任务，用了更大的数据集、更大的 batch size、动态 mask（在 train step 中随机 mask）

7.1.3.2 数据

1. 各个数据源、以及数量

Data Source	Language	Size
Unsupervised Data		
MTP	EN, ZH	291.1M
S2ORC, Wikipeida	EN	48.3M
xP3, mC4, CC-News	Multi-Lingual	488.4M
NLLB, CCMatrix	Cross-Lingual	391.3M
CodeSearchNet	Text-Code	344.1K
Total	–	1.2B
Fine-tuning Data		
MS MARCO, HotpotQA, NQ, NLI, etc.	EN	1.1M
DuReader, T^2-Ranking, NLI-zh, etc.	ZH	386.6K
MIRACL, Mr.TyDi	Multi-Lingual	88.9K
MultiLongDoc	Multi-Lingual	41.4K

Table 1: Specification of training data

2. 除上面的数据外，作者还合成了 Multi-lingual long document。具体是：从 wiki 和 MC4 数据集中，用 GPT3.5 对 document 生成问题，以此构建 question-document 数据

7.1.3.3 hybrid retrieval

所谓 hybrid retrieval 是同时支持 dense retrieval、lexical retrieval、Multi-vector retrieval.

7.1.3.3.1 dense retrieval

1. 取的 CLS 位置，Query 和 retrieval 都是 CLS 位置。
2. 公式： eq=norm(Hq[0])e_q = norm(H_q[0]) 。 ep=norm(Hp[0])e_p= norm(H_p[0])
3. 相似度是用 inner product，公式： sdense←⟨ep,eq⟩s_{dense}← ⟨ e_p, e_q⟩ .

7.1.3.3.2 lexical retrieval

1. 这里取的是 除 CLS 位置之外的其他位置；
2. term weight 的计算方法：
    1. wqt←Relu(WlexTHq[i]))w_{q_t} ← Relu(W_{lex}^T H_q[i])) ， Wlex∈Rd* 1W_{lex} \in R^{d * 1} 。
    2. 这里的 term 就是 token。query 和 passage 都是这样算
3. 相似度的计算方法：取在 Query 和 passage 中同时存在的 term，用 term weight 相乘

公式： slex←∑t∈q∩p(wqt*wpt)s_{lex} ← ∑{t ∈ q∩p} (w{qt} * w_{pt}) .

7.1.3.3.3 Multi-vector retrieval

1. 是取的整行的 logits
2. 公式：
    1. Eq=norm(WmulTHq)E_q= norm(W_{mul}^T H_q)
    2. Ep=norm(WmulTHp)E_p = norm(W_{mul}^T H_p)
3. 相似度
    1. 是用的 ColBERT 中的 late-interaction，以此捕获细粒度的相似度语义。
    2. smul←1N∑i=1Nmaxj=1MEq[i]·EpT[j]s_{mul} ← \frac{1}{N} \sum_{i=1}^N max_{j=1}^M E_q[i] \cdot E_p^T [j]
    3. 其中，N and M are the lengths of query and passage.

7.1.3.3.4 3 种 retrieval 的最终相似度

srank←sdense+slex+smuls_{rank} ← s_{dense} + s_{lex} + s_{mul}

7.1.3.4 self-knowledge distillation

1. 动机：text embedding 这个领域训练模型的时候，训练数据是正负例的 pair 对形式，loss 函数通常采用对比学习的 InfoNCE loss。因为作者这里采用的是 3 种不同粒度的 retrieval，会相互冲突。作者在原来的 InfoNCE loss 的基础上加了一个新的 Part。
2. InfoNCE loss

$$\mathcal{L} = -\log \frac{\exp(s(q,p^*)/\tau)}{\sum_{p\in\{p^*,P'\}} \exp(s(q,p)/\tau)}. \quad (1)$$

3. 新加的 loss Part

1. 原文说的是集成学习的思想，将多个 retrieval functions 集成在一起，因为我看过很多多任务的 loss 设计，都是这么玩的，和集成学习没太多关系

2. 每个 retrieval，其 loss 是

2021). In this place, we simply employ the integration score s_{inter} as the teacher, where the loss function of each retrieval method is modified as:

$$\mathcal{L}'_* \leftarrow -p(s_{inter}) * \log p(s_*). \quad (3)$$

Here, $p(\cdot)$ is the softmax activation; s_* is any of the members within s_{dense}, s_{lex}, and s_{mul}. We further

其中，s_* 表示每个 retrieval。 sinters_{inter} 的公式是：

$$s_{inter} \leftarrow s_{dense} + s_{lex} + s_{mul}. \quad (2)$$

最后再相加：

$$\mathcal{L}' \leftarrow \left(\mathcal{L}'_{dense} + \mathcal{L}'_{lex} + \mathcal{L}'_{mul}\right)/3. \quad (4)$$

3. 最终的 loss

Lfinal←L+L′ L_{final} \leftarrow L + L^{'}

4. 训练的过程

就是在自有的[多语言数据集](#)上进行 Pre-training，然后再 fine-tuning

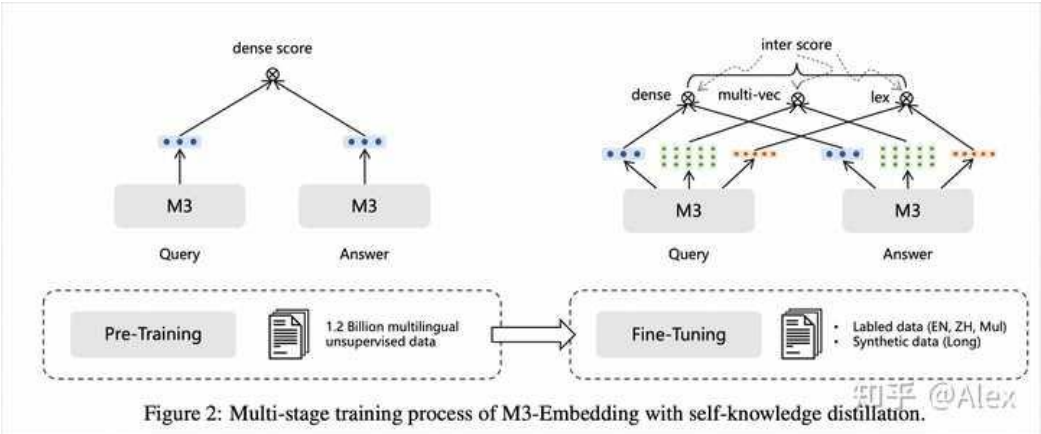
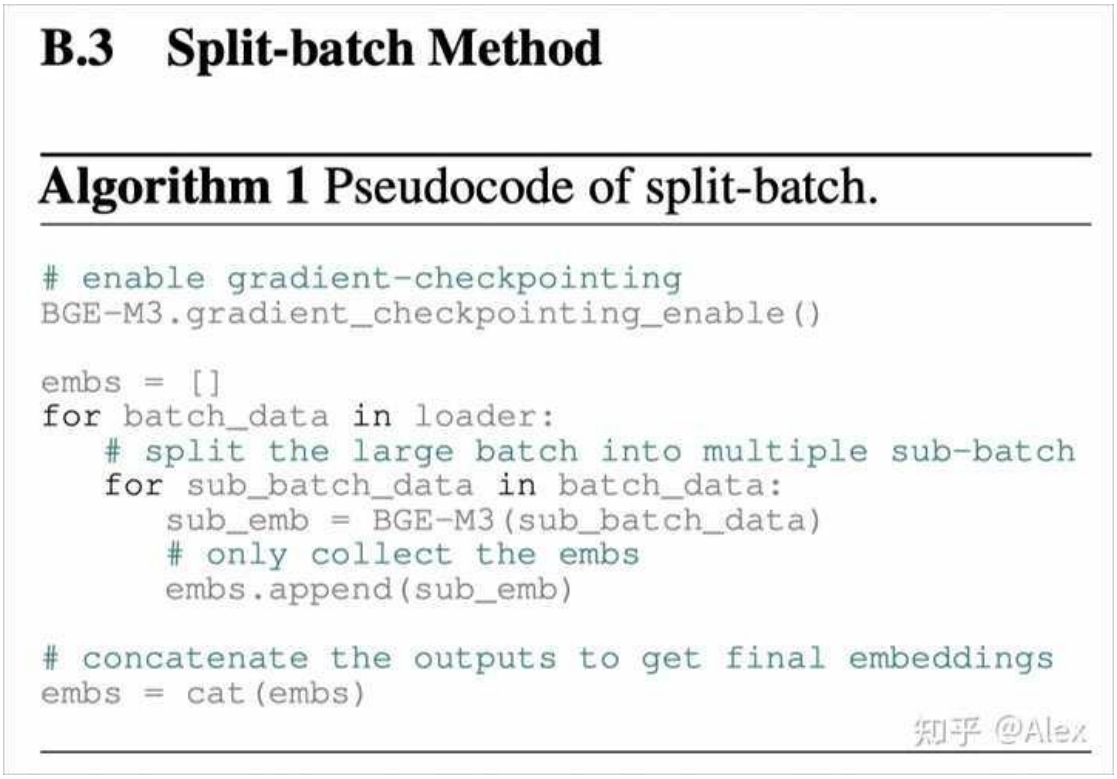


Figure 2: Multi-stage training process of M3-Embedding with self-knowledge distillation.

7.1.3.5 optimized batching strategy

1. 优化 1: 按照句子长度进行排序，因为一个 batch 里面长度都差不多，Padding 的长度都差不多，这样无效的 Padding 就会少，更能利用 GPU
2. 优化 2: batch 之后进行 split，这样长文本的时候，GPU 能塞得下。直接看伪代码:



B.3 Split-batch Method

Algorithm 1 Pseudocode of split-batch.

```
# enable gradient-checkpointing
BGE-M3.gradient_checkpointing_enable()

embs = []
for batch_data in loader:
   # split the large batch into multiple sub-batch
   for sub_batch_data in batch_data:
      sub_emb = BGE-M3(sub_batch_data)
      # only collect the embs
      embs.append(sub_emb)

# concatenate the outputs to get final embeddings
embs = cat(embs)
```


7.1.4 效果

1. 多语言任务
    1. 表示看不懂为何放 dev set 的效果，不贴图了
2. 跨语言任务

效果好很多

	Baselines (*Prior Work*)						M3-Embedding (*Our Work*)				
	BM25	mDPR	mContriever	$mE5_{large}$	$E5_{mistral\text{-}7b}$	OpenAI-3	Dense	Sparse	Multi-vec	Dense+Sparse	All
ar	18.9	48.2	58.2	68.7	59.6	65.6	71.1	23.5	71.4	71.1	**71.5**
da	49.3	67.4	73.9	77.4	**77.8**	73.6	77.2	55.4	77.5	77.4	77.6
de	35.4	65.8	71.7	76.9	**77.0**	73.6	76.2	43.3	76.3	76.4	76.3
es	43.4	66.8	72.6	76.4	**77.4**	73.9	76.4	50.6	76.6	76.7	76.9
fi	46.3	56.2	70.2	74.0	72.0	72.7	75.1	51.1	75.3	75.3	**75.5**
fr	45.3	68.2	72.8	75.5	**78.0**	74.1	76.2	53.9	76.4	76.6	76.6
he	26.9	49.7	63.8	69.6	47.2	58.1	72.4	31.1	72.9	72.5	**73.0**
hu	38.2	60.4	69.7	74.7	**75.0**	71.2	74.7	44.6	74.6	74.9	**75.0**
it	45.2	66.0	72.3	76.8	**77.1**	73.6	76.0	52.5	76.4	76.3	76.5
ja	24.5	60.3	64.8	71.5	65.1	71.9	75.0	31.3	75.1	75.0	**75.2**
km	27.8	29.5	26.8	28.1	34.3	33.9	68.6	30.1	69.1	68.8	**69.2**
ko	27.9	50.9	59.7	68.1	59.4	63.9	71.6	31.4	71.7	71.6	**71.8**
ms	55.9	65.5	74.1	76.3	77.2	73.3	77.2	62.4	**77.4**	**77.4**	**77.4**
nl	56.2	68.2	73.7	77.8	**79.1**	74.2	77.4	62.4	77.6	77.7	77.6
no	52.1	66.7	73.5	77.3	76.6	73.3	77.1	57.9	77.2	**77.4**	77.3
pl	40.8	63.3	71.6	76.7	**77.1**	72.7	76.3	46.1	76.5	76.3	76.6
pt	44.9	65.5	72.0	73.5	**77.5**	73.7	76.3	50.9	76.4	76.5	76.4
ru	33.2	62.7	69.8	**76.8**	75.5	72.0	76.2	36.9	76.4	76.2	76.5
sv	54.6	66.9	73.2	77.6	**78.3**	74.0	76.9	59.6	77.2	77.4	77.4
th	37.8	53.8	66.9	76.0	67.4	65.2	76.4	42.0	76.5	76.5	**76.6**
tr	45.8	59.1	71.1	74.3	73.0	71.8	75.6	51.8	75.9	**76.0**	**76.0**
vi	46.6	63.4	70.9	75.4	70.9	71.1	76.6	51.8	76.7	76.8	**76.9**
zh_cn	31.0	63.7	68.1	56.6	69.3	70.7	74.6	35.4	74.9	74.7	**75.0**
zh_hk	35.0	62.8	68.0	58.1	65.1	69.6	73.8	39.8	74.1	74.0	**74.3**
zh_tw	33.5	64.0	67.9	58.1	65.8	69.7	73.5	37.7	73.5	**73.6**	**73.6**
Avg	39.9	60.6	67.9	70.9	70.1	69.5	75.1	45.3	75.3	75.3	**75.5**

Table 3: Cross-lingual retrieval performance on MKQA (measured by Recall@100).

3. 长文本任务

这好的不是一点点了。。

	Max Length	Avg	ar	de	en	es	fr	hi	it	ja	ko	pt	ru	th	zh
Baselines (*Prior Work*)															
BM25	8192	53.6	45.1	52.6	57.0	78.0	75.7	43.7	70.9	36.2	25.7	82.6	61.3	33.6	34.6
mDPR	512	23.5	15.6	17.1	23.9	34.1	39.6	14.6	35.4	23.7	16.5	43.3	28.8	3.4	9.5
mContriever	512	31.0	25.4	24.2	28.7	44.6	50.3	17.2	43.2	27.3	23.6	56.6	37.7	9.0	15.3
$mE5_{large}$	512	34.2	33.0	26.9	33.0	51.1	49.5	21.0	43.1	29.9	27.1	58.7	42.4	15.9	13.2
$E5_{mistral\text{-}7b}$	8192	42.6	29.6	40.6	43.3	70.2	60.5	23.2	55.3	41.6	32.7	69.5	52.4	18.2	16.8
text-embedding-ada-002	8191	32.5	16.3	34.4	38.7	59.8	53.9	8.0	46.5	28.6	20.7	60.6	34.8	9.0	11.2
jina-embeddings-v2-base-en	8192	-	-	-	37.0	-	-	-	-	-	-	-	-	-	-
M3-Embedding (*Our Work*)															
Dense	8192	52.5	47.6	46.1	48.9	74.8	73.8	40.7	62.7	50.9	42.9	74.4	59.5	33.6	26.0
Sparse	8192	62.2	58.7	53.0	62.1	87.4	82.7	49.6	74.7	53.9	47.9	85.2	72.9	40.3	40.5
Multi-vec	8192	57.6	56.6	50.4	55.8	79.5	77.2	46.6	66.8	52.8	48.8	77.5	64.2	39.4	32.7
Dense+Sparse	8192	64.8	63.0	56.4	**64.2**	**88.7**	**84.2**	**52.3**	**75.8**	58.5	53.1	**86.0**	**75.6**	42.9	**42.0**
All	8192	**65.0**	**64.7**	**57.9**	63.8	86.8	83.9	52.2	75.5	**60.1**	**55.7**	85.4	73.8	**44.7**	40.0
M3-w.o.long															
Dense-w.o.long	8192	41.2	35.4	35.2	37.5	64.0	59.3	28.8	53.1	41.7	29.8	63.5	51.1	19.5	16.5
Dense-w.o.long (MCLS)	8192	45.0	37.9	43.3	41.2	67.7	64.6	32.0	55.8	43.4	33.1	67.8	52.8	27.2	18.2

Table 4: Evaluation of multilingual long-doc retrieval on the MLDR test set (measured by nDCG@10).

7.1.5 个人看法

1. representation 的 size 不会小，落地难度大。不知道量化后使用效果如何?
2. 从方法创新上看
    1. optimized batching strategy 这个算是纯粹工程技巧。
    2. loss 函数的设计，中规中矩，多任务时都是这样的加权方式。

3. 最后的效果，只在长文本上拉开差距，这个可能和长文本的构建策略有关，所以 loss 函数和 hybrid retrieval 的设计在中短文本上可能没有想象的起作用。

8 LLM 做 retrieval

传统基于 PTM 做 retrieval 的方式，似乎在大模型出来之后，都一窝蜂的涌到 LLM 上了。除了做 LLM 本身的研究，用 LLM 做底座、与各个下游任务结合的并不多，似乎和这个路径不好做创新有关，这里给出 2 篇文章的解读。

8.1 D2LLM （2024，ACL）

- 论文题目：D2LLM: Decomposed and Distilled Large Language Models for Semantic Search
- 发表会议：ACL
- 发表年份：2024
- 作者单位：华东师范大学
- 论文链接：D2LLM: Decomposed and Distilled Large Language Models for Semantic Search (arxiv.org)
- 代码：https://github.com/codefuse-ai/D2LLM
    - 框架：

8.1.1 一句话总结

通过蒸馏的方式，将 LLM 的 teacher 模型蒸馏到改造的 student 模型，teacher 和 student 模块是基于 Qwen-7B-chat 模型，同时融入作者提出的 Pooling by Multihead Attention (PMA) and the Interaction Emulation Module (IEM)两个模块，并设计了 4 种 loss。

8.1.2 先验知识

1. symmetric search：Query 和 passage 都是相同类型，比如 Query 是问题、passage 也是问题
2. asymmetric search：Query 和 passage 是不同类型，比如 Query 是问题、passage 也是 document

8.1.3 方法

8.1.3.1 模型结构

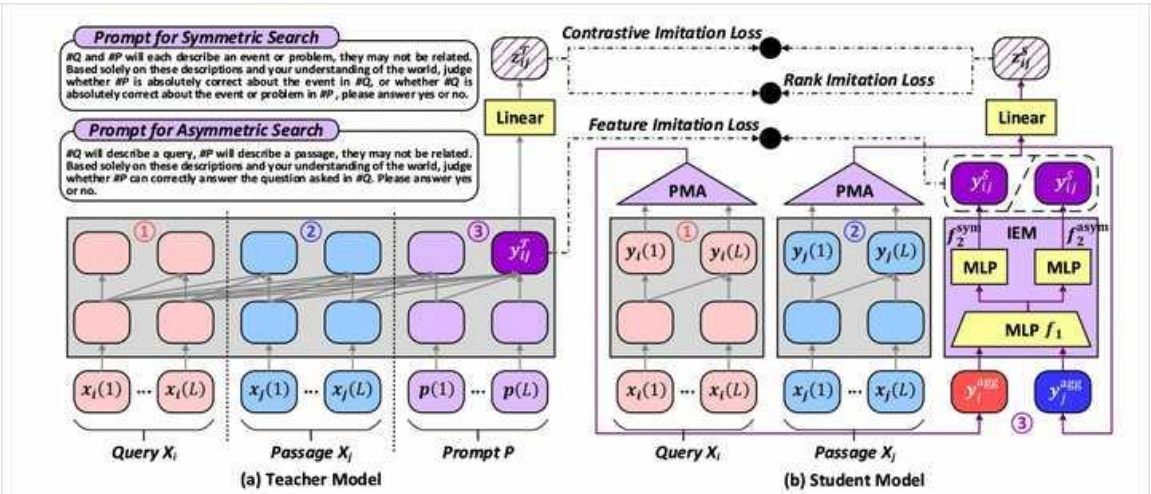


Figure 1: Architecture of D2LLM: The teacher model is decomposed into three segments corresponding to the input of the ① query, ② passage, and ③ prompt, represented in light red, light blue, and light purple. Its output, represented by a dark purple square, is the classification token embedding, which, after a linear layer, yields logits. The student model maintains the query and passage components (① and ②) from the teacher but adds ③ the PMA and IEM to capture the interplay between the query and passage, as well as their combined interaction with the prompt. It also outputs classification token embeddings respectively for symmetric and asymmetric search and then derives logits via a linear layer.

Teacher 和 student 都是选用的： Qwen-7B-Chat 作为基座模型

8.1.3.1.1 teacher 模型

1. 输入：query、passage、prompt。 prompt 有 2 种
2. 输出：yes 或 no。表示 query 和 passage 是否相似
3. Teacher 模型微调的使用使用 Lora，秩用的 8

8.1.3.1.2 student 模型

1. Pooling by Multihead Attention (PMA)模块
    1. 公式如下，简洁明了：

$$\boldsymbol{y}_i^{\text{agg}} = \text{PMA}_{\boldsymbol{q}}(\mathbf{Y}_i) = \text{LN}(\boldsymbol{h} + \text{FFN}(\boldsymbol{h})), \quad (4)$$

$$\boldsymbol{h} = \text{LN}(\text{MHA}(\boldsymbol{q}, \mathbf{Y}_i, \mathbf{Y}_i) + \boldsymbol{q}), \quad (5)$$

where LN, FFN, and MHA($\mathbf{Q}, \mathbf{K}, \mathbf{V}$) respectively denote layer normalization, feedforward network, and multihead attention with query $\mathbf{Q}$, key $\mathbf{K}$, and

1. 上面的公式中的 q 含义是：The PMA’s query q is a learnable vector that functions as an anchor, extracting information from the L tokens based on their similarity to the query q for semantic search.
2. 作者说这种方法比 fix-ed weight 方法下的 mean/min/max 方法都好
3. 选用的 32 个头

2. Interaction Emulation Module (IEM)

1. 目的是：captures the query-passage interaction
2. 公式如下：

$$\boldsymbol{y}_{ij}^{\mathcal{S}} = f_2(f_1([\boldsymbol{y}_i^{\text{agg}}, \boldsymbol{y}_j^{\text{agg}}])), \quad (6)$$

where both f_1 and f_2 are MLPs. f_1 extracts elementary features from the combined embeddings, while $f_2 \in \{f_2^{\text{sym}}, f_2^{\text{asym}}\}$ is tailored to the two branches, processing symmetric and asymmetric searches.

上面的两个 f 函数都是 MLP， Multi-Layer Perceptron (MLP)。用的 2 层，加 ReLU.。输入的维度是 8192，输出的维度是 512.

3. IEM 之后通过 linear 计算获得 logits，再之后计算获得 score

8.1.3.2 数据集构造

其实就是正负例构造。更细一点是：in-batch 和 hard negatives。老生常谈了，就那几种策略，这里主要说 hard negatives：通过 BM25 和 bi-encoder 方法构造

8.1.3.3 loss 设计 :+1:

8.1.3.3.1 Contrastive Imitation (CI) loss

1. 目的
    1. Contrastive Imitation (CI) loss effectively handles true positives and easy negatives
    2. 侧重于学习正例、简单负例
    3. CI and RI aim to align the student’s output with that of the teacher, emphasizing output distillation.
2. 公式

$$\mathcal{L}^{\mathrm{CI}} = -\frac{1}{|\mathbb{D}^+|}\sum_{j\in\mathbb{D}^+}\log\frac{\exp(s_{ij}^{\mathcal{T}} z_{ij}^{\mathcal{S}}/\tau)}{\sum_{k\in\mathbb{D}^-}\exp((1-s_{ik}^{\mathcal{T}}) z_{ik}^{\mathcal{S}}/\tau)},$$

where τ is the temperature parameter, $s_{ij}^{\mathcal{T}}$ is the teacher's probability score for a "yes" between pairs (i, j), and $z_{ij}^{\mathcal{S}}$ is the student's corresponding logit (unnormalized probability). The CI loss di-

3. 复习一下对比学习中的 InfoNCE loss

1. 公式：

3.1.1 Contrastive learning

Contrastive learning is an approach that relies on the fact that every document is, in some way, unique. This signal is the only information available in the absence of manual supervision. A contrastive loss is used to learn by discriminating between documents. This loss compares either positive (from the same document) or negative (from different documents) pairs of document representations. Formally, given a query q with an associated positive document k_+, and a pool of negative documents $(k_i)_{i=1..K}$, the contrastive InfoNCE loss is defined as:

$$\mathcal{L}(q, k_+) = -\frac{\exp(s(q, k_+)/\tau)}{\exp(s(q, k_+)/\tau) + \sum_{i=1}^{K}\exp(s(q, k_i)/\tau)}, \quad (1)$$

where τ is a temperature parameter. This loss encourages positive pairs to have high scores and negative pairs to have low scores. Another interpretation of this loss function is the following: given the query representation q, the goal is to recover, or retrieve, the representation k_+ corresponding to the positive document, among all the negatives k_i. In the following, we refer to the left-hand side representations in the score as queries and the right-hand side representations as keys.

2. 再回看上面的公式，是不是觉得很像很像？这公式就是衍生自原始对比学习的 loss，是不是和 softmax 公式很像？

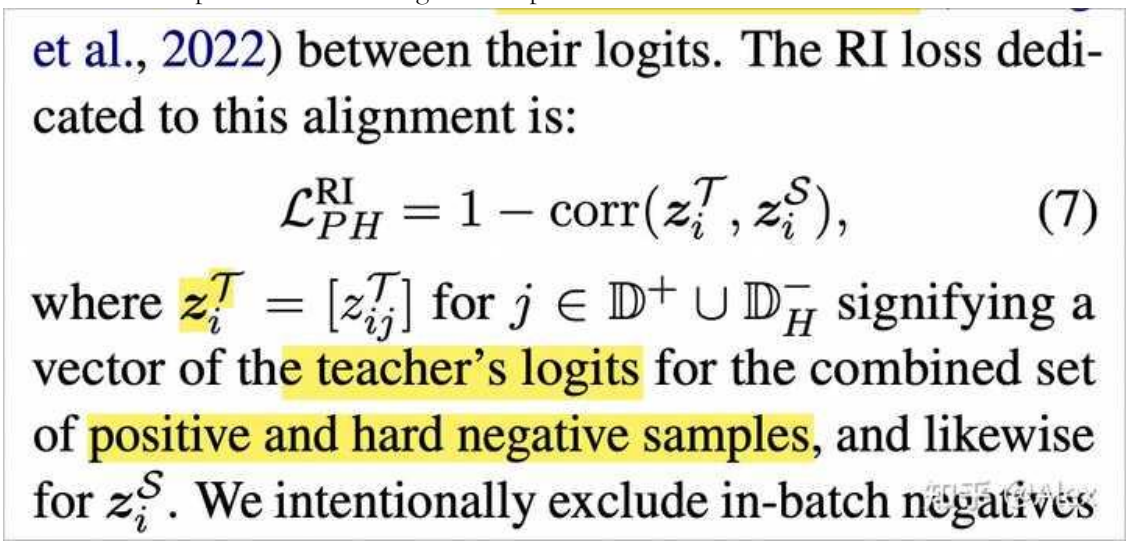
4. 再补充一个点：基于 BM25 或 bi-encoder 构造的难负例，会有一些错误，作者说这样的设计能弥补这些错误，原话是 assigning higher weights to easy negatives than hard ones. 这句话再反刍反刍。

8.1.3.3.2 Rank Imitation (RI) loss

1. 目的

1. Rank Imitation (RI) , focusing on distinguishing between positive and hard negative samples, as well as discerning easy from hard negatives, thus enabling the student to replicate the teacher’s subtle ranking nuances.
2. 侧重于学习正例、难负例，这样也能保证 teacher model 的知识迁移到 student model 上。具体实现是：maximize the Pearson correlation between their logits
3. 再把之前的话拷贝过来，这句话值得仔细品：CI and RI aim to align the student’s output with that of the teacher, emphasizing output distillation.

2. 公式
    1. positive and hard negative samples

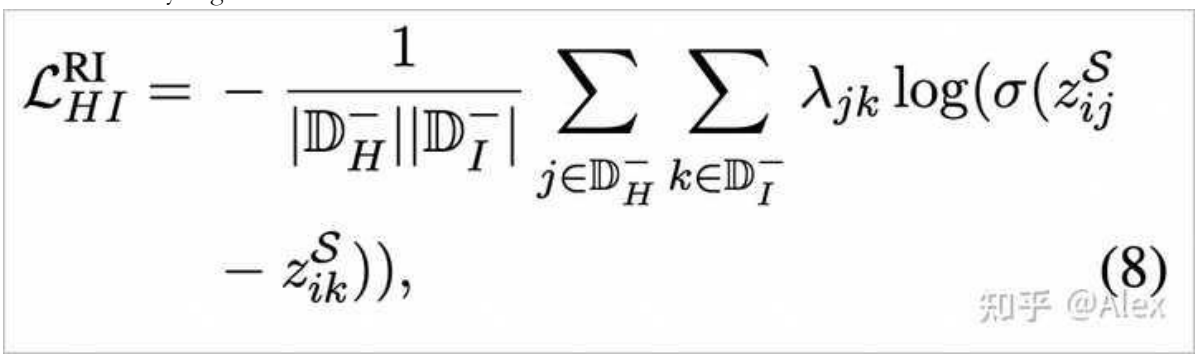


et al., 2022) between their logits. The RI loss dedicated to this alignment is:

$$\mathcal{L}_{PH}^{\mathrm{RI}} = 1 - \mathrm{corr}(\boldsymbol{z}_i^{\mathcal{T}}, \boldsymbol{z}_i^{\mathcal{S}}), \quad (7)$$

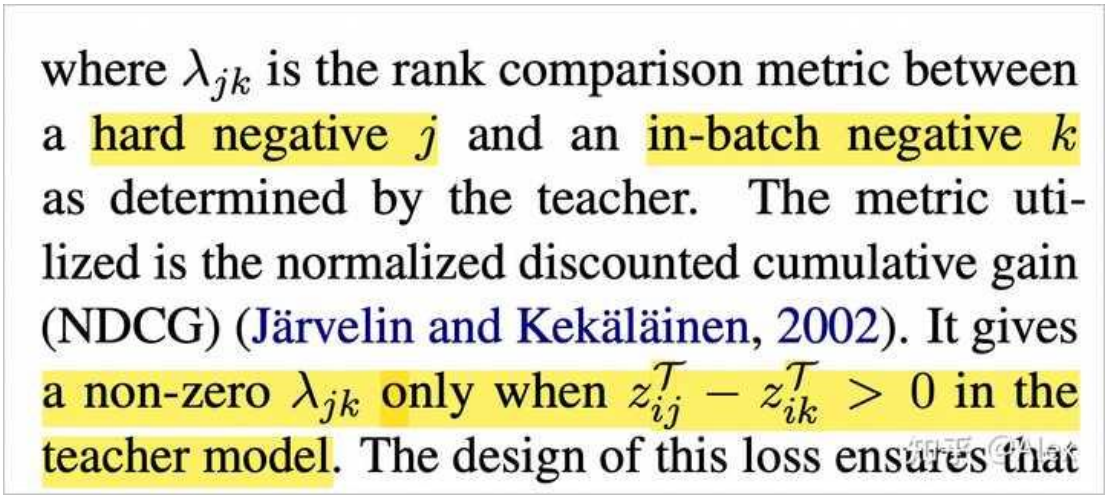
where $\boldsymbol{z}_i^{\mathcal{T}} = [z_{ij}^{\mathcal{T}}]$ for $j \in \mathbb{D}^+ \cup \mathbb{D}_H^-$ signifying a vector of the teacher's logits for the combined set of positive and hard negative samples, and likewise for $\boldsymbol{z}_i^{\mathcal{S}}$. We intentionally exclude in-batch negatives

2. hard and easy negatives



$$\mathcal{L}_{HI}^{\mathrm{RI}} = -\frac{1}{|\mathbb{D}_H^-||\mathbb{D}_I^-|} \sum_{j \in \mathbb{D}_H^-} \sum_{k \in \mathbb{D}_I^-} \lambda_{jk} \log(\sigma(z_{ij}^{\mathcal{S}} - z_{ik}^{\mathcal{S}})), \quad (8)$$

上面公式中各个字段的含义：


where λ_{jk} is the rank comparison metric between a hard negative j and an in-batch negative k as determined by the teacher. The metric utilized is the normalized discounted cumulative gain (NDCG) (Järvelin and Kekäläinen, 2002). It gives a non-zero λ_{jk} only when $z_{ij}^{\mathcal{T}} - z_{ik}^{\mathcal{T}} > 0$ in the teacher model. The design of this loss ensures that

3. 思考：这里根据正负例的类别，设计了 2 种不同的 loss，怎么做到的？数据集传入的时候，还能根据标签类别启用不启用？后续看了源代码再过来补充 :)

8.1.3.3.3 feature loss

1. 目的
    1. 难负例和 Query 之间有某种程度的联系，这种联系狠重要，设计新的 loss，强化这种关系。
2. 公式

$\boldsymbol{y}_k^{\text{agg}}$ alone. The goal of FI is to minimize the ℓ_2 norm of the difference between the teacher's and student's similarity matrices for all positive and hard negative sample combinations (i.e., $\boldsymbol{r}_i^{\mathcal{T}} = [r_{ijk}^{\mathcal{T}}]$ and $\boldsymbol{r}_i^{\mathcal{S}} = [r_{ijk}^{\mathcal{S}}]$ for all j and k):

$$\mathcal{L}^{\text{FI}} = \|\boldsymbol{r}_i^{\mathcal{T}} - \boldsymbol{r}_i^{\mathcal{S}}\|_2^2. \quad (10)$$

3. 最终的 loss 是用的 L2 norm

8.1.3.3.4 最终的 loss

1. 公式

$$\mathcal{L} = \mathcal{L}^{\text{CI}} + \alpha\mathcal{L}_{PH}^{\text{RI}} + \beta\mathcal{L}_{HI}^{\text{RI}} + \gamma\mathcal{L}^{\text{FI}}, \quad (11)$$

朴素方法是各个 loss 相加，这里的 loss 每个项前面都有一个 weights，但依然都是试出来的超参，alpha 是 1，beta 是 0.3，gamma 是 0.1

8.1.4 效果

8.1.4.1 消融实验

Table 3: Ablation Study on the NLI Task.

	OCNLI				CMNLI				Average
Methods	ACC	AP	Prec.	Recall	ACC	AP	Prec.	Recall	difference
−CI+CL	0.7572	0.7791	0.7411	0.7887	0.7755	0.8471	0.7692	0.8018	-3.59%
−RI_PH	0.7293	0.7658	0.7106	0.7589	0.7567	0.8129	0.7433	0.7885	-6.57%
−RI_HI	0.7375	0.7726	0.7390	0.7711	0.7721	0.8194	0.7609	0.7990	-4.92%
−FI	0.7666	0.8012	0.7518	0.8006	0.7939	0.8542	0.7771	0.8056	-2.17%
−PMA+mean	0.7734	0.7997	0.7586	0.8022	0.7954	0.8611	0.7823	0.8087	-1.71%
−PMA+[EOS]	0.7739	0.8023	0.7604	0.8025	0.7958	0.8642	0.7845	0.8107	-1.51%
−IEM+cos	0.7461	0.7886	0.7224	0.8025	0.7867	0.8377	0.7682	0.7921	-3.83%
D2LLM-1.8B	0.6907	0.7399	0.6769	0.6261	0.7102	0.7947	0.7107	0.5840	-14.76%
D2LLM	0.7889	0.8145	0.7736	0.8149	0.8014	0.8759	0.7960	0.8241	-

1. Teacher model 用小模型，效果损失很多很多。Teacher 如果只能用大模型，落地成本就会很大
2. 这个图仔细看，效果更多是来自于 RI、PH、HI 这几个 loss，PMA 和 IEM 的提升效果不多，这也说明对问题的理解程度也就是 loss 的设计才是决定效果的关键。

8.1.4.2 耗时

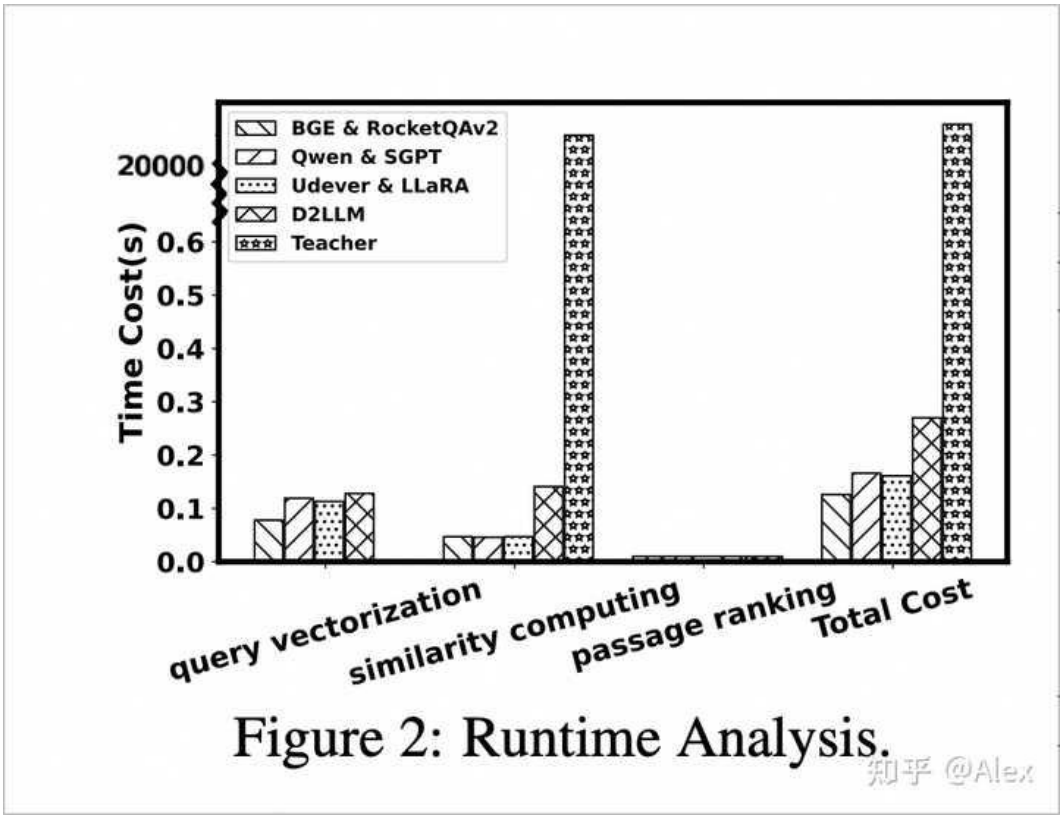
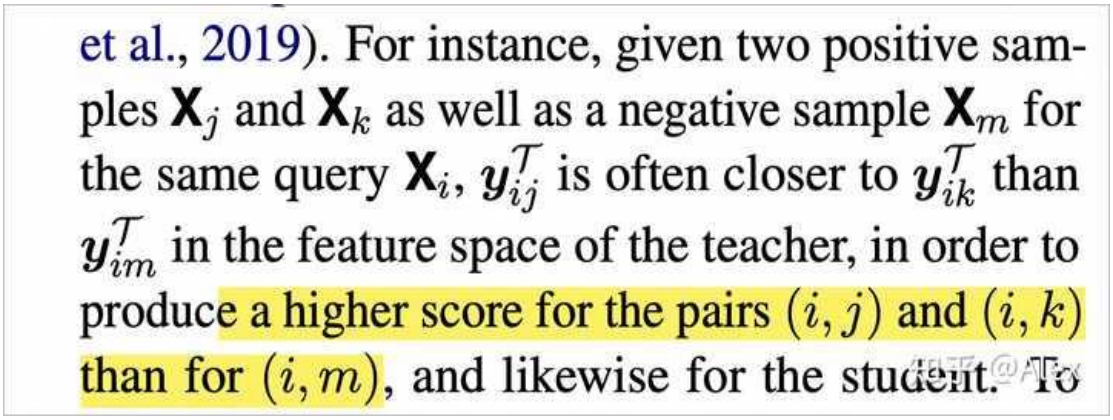


Figure 2: Runtime Analysis.

很多。。

8.1.5 个人看法

1. 从落地的硬件角度看，LLM 都是吃内存、也吃算力的巨兽，作者训练时用了 8 A100，单卡 80G。但文章并没有给出内存的消耗，虽然效果有提升，但如果应用难度太大，不计工本式的提升效果又有何意义?
2. 从问题理解角度看，作者详细讨论了正例和正例的相似度现象：一个正例与其他正例 pair 中的任何一个的相似度，可能比和自己本身负例的相似度还高。但我有更深入的理解，这将在文本召回的下篇给出。


et al., 2019). For instance, given two positive samples $\mathbf{X}_j$ and $\mathbf{X}_k$ as well as a negative sample $\mathbf{X}_m$ for the same query $\mathbf{X}_i$, $\boldsymbol{y}_{ij}^{\mathcal{T}}$ is often closer to $\boldsymbol{y}_{ik}^{\mathcal{T}}$ than $\boldsymbol{y}_{im}^{\mathcal{T}}$ in the feature space of the teacher, in order to produce a higher score for the pairs (i, j) and (i, k) than for (i, m), and likewise for the student. To

3. 思考：设计了各种 loss，对于垂类，如果构造不出来这么多难负例，或者训练数据，这些 loss 还有用么?

9 Summary

1. 从文本召回的技术演化路径角度看，大概经历了 N 个阶段：

    1. 阶段一：single-vector 阶段。以 BERT 为基座模型，最后的 representation 层只会取一个位置，关于这个取的位置，各式各样。同时，这个阶段有各式各样的负采样方法。
    2. 阶段二：multi-vector 阶段。依然以 BERT 为基座模型，最后的 representation 层会取多个位置；
    3. 阶段三：各种粒度阶段。比如 passage-level、phrase-level、document-level，各说各的好。这里面还有一个子分支是：将 text embedding 转换成 term weights，以此增强 sparse 或者 lexical retrieval。这里给出几个工作：
    4. 阶段四：各种组合。将很多种方法组合在一起，取众家之长的思路；
    5. 阶段五：轻量化阶段。最近几年这个方向开始收到关注。
2. 驱动这几个阶段发展的背后的技术是：
    1. Pre-training model：BERT 等；
    2. 对比学习（contrastive learning）：尤其是负采样、难负例构造；
    3. 蒸馏（knowledge distillation）：早几年比较火热，近些年因为边缘计算开始流行，基于蒸馏的方法也比较多，或许是个可以尝试发论文的方向。
3. 多语言这个方向，最近也有很多优秀的工作出来。从模型角度看，和单一语言区别不大，更多是多个语言的数据方面带来的提升。