Aggretriever: A Simple Approach to Aggregate Textual Representations for Robust Dense Passage Retrieval

Sheng-Chieh Lin, Minghan Li, and **Jimmy Lin**
David R. Cheriton School of Computer Science
University of Waterloo, Canada
{s269lin,m692li,jimmylin}@uwaterloo.ca

Abstract

Pre-trained language models have been successful in many knowledge-intensive NLP tasks. However, recent work has shown that models such as BERT are not ''structurally ready'' to aggregate textual information into a [CLS] vector for dense passage retrieval (DPR). This ''lack of readiness'' results from the gap between language model pre-training and DPR fine-tuning. Previous solutions call for computationally expensive techniques such as hard negative mining, cross-encoder distillation, and further pre-training to learn a robust DPR model. In this work, we instead propose to fully exploit knowledge in a pre-trained language model for DPR by aggregating the contextualized token embeddings into a dense vector, which we call **agg***. By concatenating vectors from the [CLS] token and **agg***, our *Aggretriever* model substantially improves the effectiveness of dense retrieval models on both in-domain and zero-shot evaluations without introducing substantial training overhead. Code is available at https://github.com/castorini/dhr.

1 Introduction

A bi-encoder architecture (Reimers and Gurevych, 2019; Karpukhin et al., 2020) based on pre-trained language models (Devlin et al., 2018; Liu et al., 2019; Raffel et al., 2020) has been widely used for first-stage retrieval in knowledge-intensive tasks such as open-domain question answering and fact checking. Compared to bag-of-words models such as BM25, these approaches circumvent lexical mismatches between queries and passages by encoding text into dense vectors.

Despite their success, recent research calls into question the robustness of these single-vector models (Thakur et al., 2021). As shown in Figure 1, single-vector dense retrievers (e.g., BERT_{CLS} and TAS-B) trained with well-designed knowledge distillation strategies (Hofstätter et al., 2021) still underperform BM25 on out-of-domain datasets. Along the same lines, Sciavolino et al. (2021) find that simple entity-centric questions are challenging to these dense retrievers.

Recently, Gao and Callan (2021) observe that pre-trained language models such as BERT are not ''structurally ready'' for fine-tuning on downstream retrieval tasks. This is because the [CLS] token, pre-trained on the task of next sentence prediction (NSP), does not have the proper attention structure to aggregate fine-grained textual information. To address this issue, the authors propose to further pre-train the [CLS] vector before fine-tuning and show that the gap between pre-training and fine-tuning tasks can be mitigated (see $\text{coCondenser}_{\text{CLS}}$ illustrated in Figure 1). However, further pre-training introduces additional computational costs, which motivates us to ask the following question: Can we directly bridge the gap between pre-training and fine-tuning without any further pre-training?

Before diving into our proposed solution, we briefly overview the language modeling pre-training and DPR fine-tuning tasks using BERT. Figure 2(a) illustrates the BERT pre-training tasks, NSP and mask language modeling (MLM), while Figure 2(b) shows the task of fine-tuning a dense retriever. We observe that solely relying on the [CLS] vector as the dense representation does not exploit the full capacity of the pre-trained model, as the [CLS] vector participates directly only in NSP during pre-training, and therefore lacks information captured in the contextualized token embeddings. A simple solution is to aggregate the token embeddings by pooling (max or mean) into a single vector. However, information is lost in this process and empirical results do not show any consistent effectiveness gains. Hence, we see the need for better aggregation schemes.

Transactions of the Association for Computational Linguistics, vol. 11, pp. 436–452, 2023. https://doi.org/10.1162/tacl_a_00556
Action Editor: Hang Li. Submission batch: 9/2022; Revision batch: 12/2022; Published 5/2023.

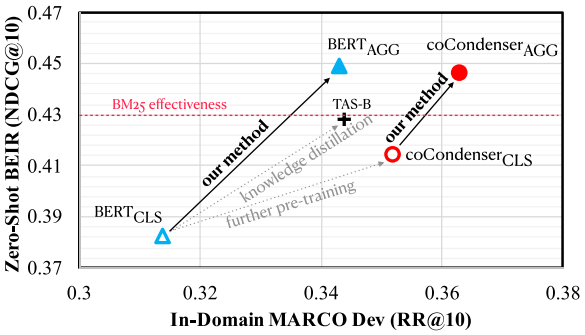


Figure 1: In-domain versus zero-shot effectiveness. All DPR models are trained with BM25 negatives.

In this paper, we propose a novel approach to generate textual representations for retrieval that fully exploit contextualized token embeddings from BERT, shown in Figure 2(c). Specifically, we reuse the pre-trained MLM head to map each contextualized token embedding into a high-dimensional wordpiece lexical space. Following a simple max-pooling and pruning strategy, we obtain a compact *lexical* vector that we call **agg***. By concatenating **agg*** and the `[CLS]` vector, our novel *Aggretriever* dense retrieval model captures representations pre-trained from both NSP and MLM, improving retrieval effectiveness by a noticeable margin compared to fine-tuned models that solely rely on the `[CLS]` vector (see BERT_{AGG} vs BERT_{CLS} in Figure 1).

Importantly, fine-tuning Aggretriever does not require any sophisticated and computationally expensive techniques, making it a simple yet competitive baseline for dense retrieval. However, our approach is orthogonal to previously proposed further pre-training strategies, and can still benefit from them to improve retrieval effectiveness even more (see $\text{coCondenser}_{\text{AGG}}$ in Figure 1). To the best of our knowledge, this is the first work in the DPR literature that leverages the BERT pre-trained MLM head to encode textual information into a single dense vector.

2 Background and Motivation

Given a query q, our task is to retrieve a list of passages to maximize some ranking metric such as nDCG or MRR. Dense retrievers (Reimers and Gurevych, 2019; Karpukhin et al., 2020) based on pre-trained language models encode queries and passages as low dimensional vectors with a bi-encoder architecture and use the dot product between the encoded vectors as the similarity score:

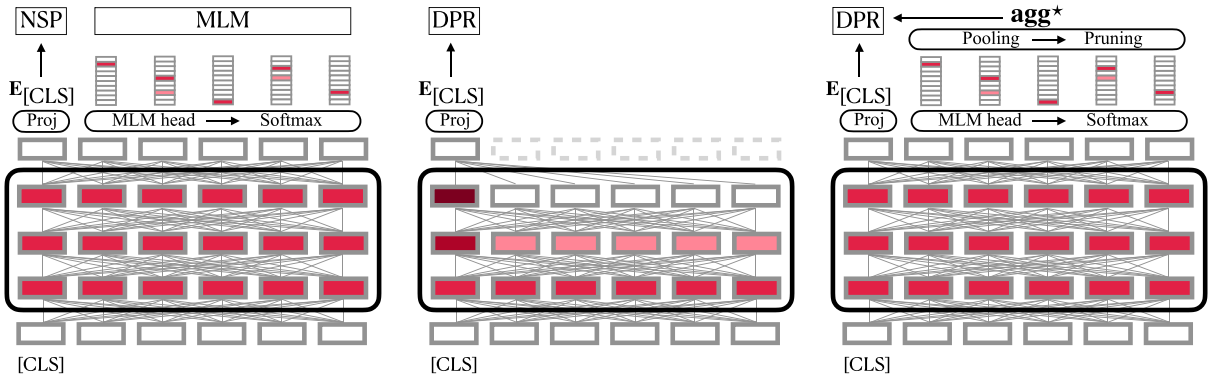
$$\text{sim}_{\texttt{[CLS]}}(q,p) \triangleq \mathbf{e}_{q_{\texttt{[CLS]}}} \cdot \mathbf{e}_{p_{\texttt{[CLS]}}}, \qquad (1)$$

where $\mathbf{e}_{q_{\texttt{[CLS]}}}$ and $\mathbf{e}_{p_{\texttt{[CLS]}}}$ are the `[CLS]` vectors at the last layer of BERT (Devlin et al., 2018). Subsequent work leverages expensive fine-tuning strategies (e.g., hard negative mining, knowledge distillation) to guide models to learn more effective and robust single-vector representations (Xiong et al., 2021; Zhan et al., 2021b; Lin et al., 2021b; Hofstätter et al., 2021; Qu et al., 2021).

Recent work (Gao and Callan, 2021; Lu et al., 2021) shows that the `[CLS]` vector remains ‘‘dormant’’ in most layers of pre-trained models and fails to adequately aggregate information from the input sequence during pre-training. Thus, researchers argue that the models are not ‘‘structurally ready’’ for fine-tuning. To tackle this issue, unsupervised contrastive learning has been proposed, which creates pseudo relevance labels from the target corpus to ‘‘prepare’’ the `[CLS]` vector for retrieval. The most representative technique is the Inverse Cloze Task (ICT; Lee et al., 2019). However, since the generated relevance data is noisy, further pre-training with ICT often requires a huge amount of computation due to the need for large batch sizes or other sophisticated training techniques (Chang et al., 2020; Izacard et al., 2021; Ni et al., 2021).

Another thread of work (Gao and Callan, 2021; Lu et al., 2021) manages to guide transformers to aggregate textual information into the `[CLS]` vector through auto-encoding. This method does not require as much computation as unsupervised contrastive learning but is still much more computationally intensive than fine-tuning. For example, Gao and Callan (2021) report that the further pre-training process still requires one week on four RTX 2080 Ti GPUs, while fine-tuning consumes less than one day in the same environment.

Recent work on neural sparse retrievers (Bai et al., 2020; Formal et al., 2021b) projects contextualized token embeddings into a high-dimensional wordpiece lexical space through the BERT pre-trained MLM projector and directly performs retrieval in wordpiece lexical space. These models demonstrate that MLM pre-trained weights can be used to learn effective lexical representations for retrieval tasks, a finding that has not been fully explored in the DPR literature. Inspired by this work, we explore reusing MLM



(a) BERT pre-training (b) [CLS] DPR fine-tuning (c) Aggretriever DPR fine-tuning

Figure 2: (a) BERT: next sentence prediction (NSP) and mask language modeling (MLM) (b) DPR: using the `[CLS]` embedding for retrieval (c) Aggretriever: aggregating knowledge from both NSP and MLM.

pre-trained weights for DPR fine-tuning and further combine the `[CLS]` vector to fully exploit textual information in a pre-trained language model.

3 Aggretriever

In this section, we first introduce our method for text aggregation to form **agg***, which consists of two steps: pooling and pruning. Then, we describe how to concatenate the aggregated text representation **agg*** and `[CLS]` into a 768-dimensional dense vector for fine-tuning and retrieval.

3.1 Text Aggregation Pooling

The goal of text aggregation is to transform contextualized token embeddings into a single-vector token representation. Let the input sequence q denote a tokenized query sequence with a length of l, $(\texttt{[CLS]}, q_1, q_2, \cdots q_l, \texttt{[SEP]})$, or alternatively, a passage p of length m, $(\texttt{[CLS]}, p_1, p_2, \cdots p_m, \texttt{[SEP]})$. One simple approach is to directly pool (mean or max) contextualized token embeddings from the final layer. Such pooling strategies have been studied in previous work (Reimers and Gurevych, 2019), but do not appear to be consistently more effective than just using the `[CLS]` token; this is also confirmed in our ablation study (Section 5.4).

We instead propose to reuse the pre-trained MLM head to project each contextualized token embedding $\mathbf{e}_{q_i}$ into a high-dimensional vector in the wordpiece lexical space:

$$\mathbf{p}_{q_i} = \text{softmax}(\mathbf{e}_{q_i} \cdot \mathbf{W}_{\text{MLM}} + \mathbf{b}_{\text{MLM}}), \quad (2)$$

where $\mathbf{e}_{q_i} \in \mathbb{R}^d$, $\mathbf{W}_{\text{MLM}} \in \mathbb{R}^{d \times |\text{V}_{\text{BERT}}|}$, and $\mathbf{b}_{\text{MLM}} \in \mathbb{R}^{|\text{V}_{\text{BERT}}|}$ are the weights of the pre-trained MLM linear projector, and $\mathbf{p}_{q_i} \in \mathbb{R}^{|\text{V}_{\text{BERT}}|}$ is the i-th contextualized token represented by a probability distribution over the 30522 tokens of BERT wordpiece vocabulary, V_{BERT}. We then perform weighted max pooling for the sequential representations $(\mathbf{p}_{q_1}, \mathbf{p}_{q_2}, \cdots, \mathbf{p}_{q_l})$ to obtain a single-vector lexical representation:

$$\mathbf{v}_q[v] = \max_{i \in (1,2,\cdots,l)} w_i \cdot \mathbf{p}_{q_i}[v], \quad (3)$$

where $w_i = |\mathbf{e}_{q_i} \cdot \mathbf{W} + \mathbf{b}| \in \mathbb{R}^1$ is a positive scalar and $v \in \{1, 2, \cdots, |\text{V}_{\text{BERT}}|\}$; $\mathbf{W} \in \mathbb{R}^{d \times 1}$ and $\mathbf{b} \in \mathbb{R}^1$ are trainable weights. Note that the scalar w_i for each token q_i is essential to capture term importance, which $\mathbf{p}_{q_i}$ alone cannot capture since it is normalized by softmax. We exclude the `[CLS]` token embedding at this stage since it is used for next-sentence prediction during pre-training and thus we argue that it does not carry much lexical information.

Our design has three advantages: (1) the MLM head with softmax is used for BERT pre-training; thus, the output probabilities can accurately model each contextualized token semantically. (2) In contrast to directly pooling contextualized embeddings, important dimensions of the token representations in the high-dimensional space are less likely to overlap, resulting in non-interfering max-pooling (Jang et al., 2021). (3) Finally, w_i and $\mathbf{p}_{q_i}$ disentangle the effects of term importance from the MLM head. We will study the effectiveness of this design in Section 5.4 through ablations. Note that compared to previous work on

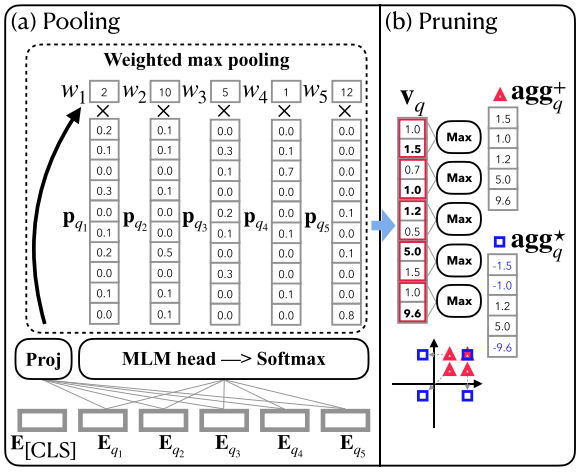


Figure 3: Illustration of text aggregation: (a) pooling of token representations to form $\mathbf{v}_q$; (b) pruning of $\mathbf{v}_q$ to form $\mathbf{agg}^{\star}_q$ (or $\mathbf{agg}^{+}_q$). While pruning, $\mathbf{agg}^{\star}_q[n]$ receives a negative value if the pooled element belongs to S^{-}_n; i.e., the second element in each slice (red box).

sparse retrieval (Bai et al., 2020; Formal et al., 2021b), which switches softmax to ReLU to create sparse representations, our design sticks to the original activation function for MLM pre-training and directly outputs 30522-dimensional dense lexical vectors ($\mathbf{v}_q$).

Figure 3(a) illustrates the generation of $\mathbf{v}_q$ with $|\text{V}_{\text{BERT}}| = 10$ for simplicity. Ideally, we can directly compute $\mathbf{v}_q \cdot \mathbf{v}_p$ as a lexical matching similarity score for the wordpiece lexical representations. However, the vectors ($\mathbf{v}_q, \mathbf{v}_p \in \mathbb{R}^{|\text{V}_{\text{BERT}}|}$) are too large for efficient retrieval using dense vector search libraries such as Faiss. To address this issue, we introduce our non-parametric pruning method to convert $\mathbf{v}_q$ ($\mathbf{v}_p$) into a low-dimensional vector for dense retrieval.

3.2 Text Aggregation Pruning

We consider $\mathbf{v}_q \in \mathbb{R}^{|\text{V}_{\text{BERT}}|}$ as a bag-of-words representation with each dimension storing the corresponding term weight. Thus, dimensions with low term weights indicate that the corresponding terms are not important and can be pruned.

Based on this intuition, we propose to prune term weights in $\mathbf{v}_q$ by evenly and randomly dividing the dimensions (vocabulary) into d slices, $(S_1, S_2, \cdots, S_d)$, where each slice consists of a set of $\frac{|\text{V}_{\text{BERT}}|}{d}$ index positions. Then, we condense $\mathbf{v}_q$ into a d-dimensional vector by pruning the term weights in each slice S_n:

$$\mathbf{agg}^{+}_q[n] = \max_{v \in S_n} \mathbf{v}_q[v]; \quad (4)$$
$$\mathbf{id}_q[n] = \arg\max_{v \in S_n} \mathbf{v}_q[v].$$

We call the operation in Eq. (4) *slice max pooling*, where each value in $\mathbf{agg}^{+}_q$ represents the weight of the most important term in the slice.[1] Slice max pooling is an important operation to prune the term weights while performing dimensionality reduction for dense passage retrieval. Other effective approaches to pruning lexical representations, e.g., top-k pruning (Yang et al., 2021) and FLOP regularization (Formal et al., 2021b), do not reduce the vector dimensionality. Thus, they generate sparse representation models that require inverted indexes for efficient retrieval.

We call $\mathbf{agg}^{+}_q \in \mathbb{R}^d$ the semi-aggregated lexical representation for query q since it only distributes vectors over the positive orthant and does not fully use the d-dimensional space. That is, $\mathbf{v}_q[v] \geq 0 \ \forall \ v \in \{1, 2, \cdots |\text{V}_{\text{BERT}}|\}$; thus, $\mathbf{agg}^{+}_q[n] \geq 0 \ \forall \ n \in \{1, 2, \cdots d\}$. Our goal is to approximate the dot product between $\mathbf{v}_q$ and $\mathbf{v}_p$ in Eq. (3) by the ones in Eq. (4):

$$\begin{aligned}\mathbf{v}_q \cdot \mathbf{v}_p &\approx \sum_{n=1}^{d} (\max_{v \in S_n} \mathbf{v}_q[v]) \cdot (\max_{v \in S_n} \mathbf{v}_p[v]) \\ &= \sum_{n=1}^{d} \mathbf{agg}^{+}_q[n] \cdot \mathbf{agg}^{+}_p[n] \\ &= \mathbf{agg}^{+}_q \cdot \mathbf{agg}^{+}_p. \end{aligned} \quad (5)$$

Note that the approximation error in Eq. (5) partially comes from *term misalignment*:

$$\mathbf{id}_q[n] \neq \mathbf{id}_p[n], \quad (6)$$

where the values in $\mathbf{agg}^{+}_q[n]$ and $\mathbf{agg}^{+}_p[n]$ do not represent the same term. Alternatively, this can be explained as fuzzy matching between two lexical representations since the two different wordpiece tokens may interact and contribute to the dot product. Term misalignment increases as d becomes smaller with respect to $|\text{V}_{\text{BERT}}|$; thus, the error increases as well, which we show in Section 5.4.

[1] Slice *mean* pooling is less effective in our experiment.

To mitigate this error, we distribute the semi-aggregated lexical representation to the negative orthants to form what we call the fully aggregated lexical representation, distributed over the entire d-dimensional space.

$$\mathbf{agg}^{\star}_q[n] = \begin{cases} \mathbf{agg}^{+}_q[n] & \text{if } \mathbf{id}_q[n] \in S^{+}_n \\ -\mathbf{agg}^{+}_q[n] & \text{if } \mathbf{id}_q[n] \in S^{-}_n, \end{cases} \tag{7}$$

where S^{+}_n and S^{-}_n are disjoint subsets of S_n (i.e., $S^{+}_n \cup S^{-}_n = S_n$ and $S^{+}_n \cap S^{-}_n = \emptyset$). That is, we evenly distribute the elements in S_n to S^{+}_n and S^{-}_n.

The dot product between two fully aggregated lexical representations then becomes:

$$\begin{aligned} \text{sim}_{\text{agg}}(q,p) &\triangleq \mathbf{agg}^{\star}_q \cdot \mathbf{agg}^{\star}_p \\ &= \sum_{n=1}^{d} \begin{cases} -\mathbf{agg}^{+}_q[n] \cdot \mathbf{agg}^{+}_p[n] & \text{if case (a) or (b)} \\ \mathbf{agg}^{+}_q[n] \cdot \mathbf{agg}^{+}_p[n] & \text{otherwise,} \end{cases} \end{aligned} \tag{8}$$

where the cases are:

(a) $\mathbf{id}_q[n] \in S^{+}_n; \mathbf{id}_p[n] \in S^{-}_n$;

(b) $\mathbf{id}_q[n] \in S^{-}_n; \mathbf{id}_p[n] \in S^{+}_n$.

That is, the dot product of $\mathbf{agg}^{\star}$ in Eq. (8) avoids interactions between misaligned terms in the above cases (with 50% of probability), which $\mathbf{agg}^{+}$ in Eq. (5) does not consider. Note that we do not store the vectors $\mathbf{id}_p$ and $\mathbf{id}_q$ to compute Eq. (8). Figure 3(b) illustrates the difference between $\mathbf{agg}^{+}_q$ and $\mathbf{agg}^{\star}_q$ with $d = 5$, $|S_n| = 2$ and $|S^{-}_n| = |S^{+}_n| = 1$ for simplicity.

3.3 Fine-Tuning and Retrieval

Although $\mathbf{agg}^{\star}$ can mitigate the issue of term misalignment, the approximation error cannot be completely eliminated unless $d = |\text{V}_{\text{BERT}}|$. To enhance retrieval effectiveness, we concatenate the $\mathbf{agg}^{\star}$ vector with the `[CLS]` vector since they are pre-trained to capture textual representations in different ways, focusing on the lexical and semantic, respectively.

In our Aggretriever model, the scoring function is the dot product of the concatenated vectors:

$$\text{sim}(q,p) \triangleq (\mathbf{e}_{q_{\texttt{[CLS]}}} \oplus \mathbf{agg}^{\star}_q) \cdot (\mathbf{e}_{p_{\texttt{[CLS]}}} \oplus \mathbf{agg}^{\star}_p),$$

where $\oplus$ means vector concatenation. The vector $\mathbf{e}_{q_{\texttt{[CLS]}}} \oplus \mathbf{agg}^{\star}_q$ captures representations pre-trained from both NSP and MLM.

	MARCO	NQ	TQA
# passages	8,841,823	21,015,325	
# training queries	532,761	58,880	60,413
# test queries	**Dev / DL19 / 20**	**Test**	**Test**
	6,980 / 43 / 53	3,610	11,313

Table 1: Dataset statistics.

During fine-tuning, we minimize the negative log-likelihood of a relevant query–passage pair. Specifically, given a query q, its relevant passage p^{+}, and a set of negative passages $\{p^{-}_1, p^{-}_2, \cdots, p^{-}_{bs}\}$, we train our model by minimizing the negative log-likelihood (NLL) of the positive $\{q, p^{+}\}$ pair over all the passages, i.e., $\mathcal{L}$ is

$$-\log \frac{\exp(\text{sim}(q,p^{+}))}{\exp(\text{sim}(q,p^{+})) + \sum_{j=1}^{bs} \exp(\text{sim}(q,p^{-}_j))}.$$

Following Karpukhin et al. (2020), we include the positive and negative passages from the other queries in the same batch as the negatives. In addition, we also use the same NLL loss, $\mathcal{L}_{\text{agg}}$ and $\mathcal{L}_{\texttt{[CLS]}}$, to optimize sim_{agg} and $\text{sim}_{\texttt{[CLS]}}$ separately. The final loss is as follows:

$$\mathcal{L} + \lambda_1 \cdot \mathcal{L}_{\text{agg}} + \lambda_2 \cdot \mathcal{L}_{\texttt{[CLS]}}.$$

We set λ_1 and λ_2 to 0.5 in all our experiments. While conducting end-to-end retrieval, we use Flat IP in Faiss (Johnson et al., 2021) to index the passage vectors. Note that in our main experiments, we project $\mathbf{e}_{q_{\texttt{[CLS]}}}$ and $\mathbf{e}_{p_{\texttt{[CLS]}}}$ to 128 dimensions through a linear layer and set $d = 640$ for $\mathbf{agg}^{\star}$ so that the dimensionality is 768.

4 Experimental Setup

4.1 Datasets

In-Domain Evaluations. We evaluate in-domain retrieval effectiveness on web search and open-domain question answering. Table 1 provides statistics of the datasets.

For web search, we use the MS MARCO passage ranking dataset introduced by Bajaj et al. (2016), comprising a corpus with 8.8M passages and around 500K training queries. We evaluate model effectiveness on the following query sets: (1) MARCO Dev, 6980 queries from the development set with one relevant passage per query

on average. Following the established procedure, we report RR@10 and R@1000 as the evaluation metrics. (2) TREC DL (Craswell et al., 2019, 2020), created by the organizers of the 2019 (2020) Deep Learning Tracks at the Text REtrieval Conferences (TRECs), where 43 (53) queries with graded relevance labels are released. We report nDCG@10, used by the organizers as the main metric.

For open-domain question answering, we use the Wikipedia corpus released by Karpukhin et al. (2020) and conduct experiments on two query sets, Natural Questions (NQ; Kwiatkowski et al., 2019) and Trivia QA (TQA; Joshi et al., 2017). We directly use the training and test sets released by Karpukhin et al. (2020) for training and evaluation, respectively. For this task, we report hit accuracy at cutoffs 5, 20, and 100, denoted R@5/20/100.

Zero-Shot Evaluations. We evaluate zero-shot retrieval effectiveness on open-domain QA with two query sets, SQuAD (Rajpurkar et al., 2016) and EntityQuestions (EntityQs; Sciavolino et al., 2021), which are challenging for dense retrieval models. We report hit accuracy at cutoffs 20 and 100 (R@20/100). In addition, we use BEIR (Thakur et al., 2021), consisting of 18 distinct IR datasets spanning diverse domains and tasks, including retrieval, question answering, fact checking, question paraphrasing, and citation prediction. We conduct zero-shot retrieval on 14 of the 18 datasets that are publicly available.[2] We report nDCG@10 averaged over the 14 datasets.

4.2 Models

Since our approach to text aggregation can be applied to any existing pre-trained encoder-only model, we test the effectiveness of Aggretriever on two pre-trained LM models and two further pre-trained models: (1) BERT (Devlin et al., 2018); (2) DistilBERT (Sanh et al., 2019), a 6-layer transformer distilled from BERT; (3) Condenser (Gao and Callan, 2021), a BERT model further pre-trained with the tasks of auto-encoding and skip-connection MLM; and (4) coCondenser (Gao and Callan, 2022), a corpus-aware Condenser combining the tasks of skip-connection MLM and an ICT variant that comes in two separate flavors, further pre-trained on the MS MARCO and Wikipedia corpora, respectively. All model checkpoints can be downloaded from the HuggingFace Model Hub.[3] We compare models fine-tuned using only the `[CLS]` vector and based on our approach with the subscripts ''CLS'' and ''AGG'', respectively, e.g., BERT_{CLS} and BERT_{AGG}. In addition, we also report the effectiveness of BM25 as a reference point; these results come from the Pyserini IR toolkit (Lin et al., 2021a).

For implementation details, we refer readers to Appendix A.1. It is worth emphasizing that in our main experiments, we do not leverage any expensive fine-tuning strategies such as hard negative mining or knowledge distillation. Thus, we fine-tune all the DPR models under the same settings for a fair comparison. Additional detailed comparisons are provided in Appendix A.2.

5 Results

5.1 In-Domain Evaluations

Fine-Tuning with Full Training Data. Table 2 compares in-domain retrieval effectiveness across the various models. We observe that our approach consistently improves on DistilBERT and BERT across all datasets, especially for metrics that emphasize top rankings. For example, $\text{DistilBERT}_{\text{AGG}}$ sees a three-point and five-point improvement over $\text{DistilBERT}_{\text{CLS}}$ on RR@10 and nDCG@10 for MS MARCO Dev and TREC DL, respectively, and over two points on R@5 for both NQ and TQA (row 2 vs 1). Similar trends can be observed on BERT (row 4 vs 3).

For the further pre-trained models, we observe that both $\text{Condenser}_{\text{AGG}}$ and $\text{coCondenser}_{\text{AGG}}$ yield effectiveness gains on MS MARCO and TQA (rows 6 and 8), which suggests that our approach is orthogonal and additive to further pre-training methods. We observe that in some cases, Aggretriever using pre-trained BERT as the backbone can obtain better retrieval effectiveness than further pre-trained models that are fine-tuned only on the `[CLS]` vector. For example, BERT_{AGG} outperforms $\text{Condenser}_{\text{CLS}}$ for MS MARCO and TQA (row 4 vs 5). This indicates that existing language models pre-trained on MLM can serve as an effective single-vector

[2]We exclude BioASQ, Signal-1M, TREC-NEWS, and Robust04.

[3]`https://huggingface.co/models`.

	MARCO Dev		DL19 / 20	NQ Test			TQA Test		
Model	RR@10	R@1K	nDCG@10	R@5	R@20	R@100	R@5	R@20	R@100
(a) BM25	0.188	0.858	0.506 / 0.475	0.438	0.629	0.783	0.663	0.764	0.832
(1) DistilBERT$_{CLS}$	0.308	0.940	0.633 / 0.629	0.660	0.785	0.860	0.698	0.790	0.849
(2) DistilBERT$_{AGG}$	0.341	0.960	**0.682** / 0.674	0.681	0.805	0.869	0.729	0.808	0.857
(3) BERT$_{CLS}$	0.314	0.942	0.612 / 0.643	0.677	0.799	0.863	0.710	0.796	0.852
(4) BERT$_{AGG}$	0.343	0.962	0.677 / 0.666	0.696	0.805	0.867	0.735	0.813	0.860
(5) Condenser$_{CLS}$	0.335	0.954	0.663 / 0.666	0.701	0.814	0.872	0.732	0.812	0.858
(6) Condenser$_{AGG}$	0.356	0.966	0.674 / **0.697**	0.699	0.810	0.873	0.747	0.821	0.864
(7) coCondenser$_{CLS}$	0.352	**0.973**	0.674 / 0.684	**0.707**	**0.818**	**0.878**	0.745	0.819	**0.867**
(8) coCondenser$_{AGG}$	**0.363**	**0.973**	0.678 / **0.697**	0.699	0.812	0.875	**0.751**	**0.823**	**0.867**

Table 2: In-domain retrieval effectiveness comparisons. All models are fine-tuned with negatives from BM25. Bold denotes the best model for that metric.

	MARCO Dev			
Model	RR@10		R@1K	
(a) BM25	0.188		0.858	
Train Size	1K	10K	1K	10K
(1) DistilBERT$_{CLS}$	0.145	0.222	0.754	0.865
(2) DistilBERT$_{AGG}$	0.207	0.260	0.868	0.905
(3) BERT$_{CLS}$	0.153	0.230	0.778	0.866
(4) BERT$_{AGG}$	0.207	0.258	0.871	0.906
(5) Condenser$_{CLS}$	0.191	0.259	0.841	0.903
(6) Condenser$_{AGG}$	0.211	0.258	0.873	0.899
(7) coCondenser$_{CLS}$	**0.234**	**0.287**	**0.935**	**0.948**
(8) coCondenser$_{AGG}$	0.209	0.280	0.880	0.914

Table 3: In-domain retrieval effectiveness while fine-tuning models using limited training data.

dense retriever, without further pre-training, using our proposed methods. Without corpus-aware further pre-training, Condenser$_{AGG}$ is competitive with coCondenser$_{CLS}$ on MS MARCO and TQA (row 6 vs 7).

Fine-Tuning with Limited Data. Table 3 reports retrieval effectiveness when the models are fine-tuned on subsets of the MS MARCO training data. Specifically, we randomly sample 1K and 10K queries from the training queries and fine-tune the models on each set for 40 epochs. We first observe that with only 1K training queries, both DistilBERT$_{CLS}$ and BERT$_{CLS}$ underperform BM25 (rows 1, 3 vs a), while both DistilBERT$_{AGG}$ and BERT$_{AGG}$ surpass BM25 (rows 2, 4 vs a) and are on par with Condenser$_{CLS}$ (row 5), indicating that our approach successfully aggregates text information into a single vector without any further pre-training. We observe similar trends when fine-tuning models with 10K training queries.

Finally, we find that coCondenser$_{CLS}$ performs the best when fine-tuning with limited training data. This is probably because coCondenser's further pre-training is designed for the `[CLS]` vector to learn corpus-aware signals from pseudo relevance in addition to skip-connection MLM. Thus, the `[CLS]` vector is more ''ready'' for retrieval with small training data.

5.2 Zero-Shot Evaluations

Near-Domain Retrieval Effectiveness. In these experiments, we examine robustness in a zero-shot retrieval setting. We first consider transfer to ''near-domain'' (Wikipedia) datasets, reported in Table 4. Specifically, we perform retrieval on test queries from SQuAD and EntityQs using models fine-tuned on NQ or TQA.

We see that Aggretriever with any backbone yields sizable gains over its `[CLS]` counterpart, with the exception that Condenser$_{AGG}$ (and coCondenser$_{AGG}$) underperforms Condenser$_{CLS}$ (and coCondenser$_{CLS}$) in SQuAD using NQ as the source (e.g., row 6 vs 5). It is worth mentioning that using TQA as the source, Aggretriever with any backbone is competitive with BM25 while the other `[CLS]` models still lag behind BM25 on the EntityQs test queries. Finally, we observe that models fine-tuned on TQA have

Target (Source)	SQuAD (NQ)		EntityQs (NQ)		SQuAD (TQA)		EntityQs (TQA)	
Model	R@20	R@100	R@20	R@100	R@20	R@100	R@20	R@100
(a) BM25	0.712	0.820	0.714	0.800	0.712	0.820	0.714	0.800
(1) DistilBERT$_{CLS}$	0.514	0.670	0.518	0.650	0.573	0.725	0.640	0.751
(2) DistilBERT$_{AGG}$	0.529	0.688	0.564	0.683	0.648	0.775	0.713	0.797
(3) BERT$_{CLS}$	0.512	0.671	0.534	0.664	0.581	0.722	0.637	0.747
(4) BERT$_{AGG}$	0.539	0.692	0.562	0.681	**0.651**	**0.779**	0.716	0.798
(5) Condenser$_{CLS}$	0.559	0.705	0.567	0.692	0.605	0.742	0.671	0.775
(6) Condenser$_{AGG}$	0.541	0.692	0.564	0.684	0.643	0.772	0.716	0.800
(7) coCondenser$_{CLS}$	**0.567**	**0.715**	0.556	0.684	0.629	0.762	0.695	0.791
(8) coCondenser$_{AGG}$	0.535	0.696	**0.584**	**0.701**	0.646	0.777	**0.724**	**0.804**

Table 4: Near-domain zero-shot retrieval effectiveness comparisons using NQ or TQA for fine-tuning. Bold denotes the best model for that metric.

Model	**BEIR** (nDCG@10)		
(a) BM25	0.430		
Source	MARCO	NQ	TQA
(1) DistilBERT$_{CLS}$	0.364	0.262	0.266
(2) DistilBERT$_{AGG}$	**0.450**	0.277	0.386
(3) BERT$_{CLS}$	0.382	0.283	0.305
(4) BERT$_{AGG}$	0.449	**0.299**	**0.394**
(5) Condenser$_{CLS}$	0.393	0.286	0.314
(6) Condenser$_{AGG}$	0.447	0.295	0.385
(7) coCondenser$_{CLS}$	0.414	0.277	0.307
(8) coCondenser$_{AGG}$	0.446	0.280	0.376

Table 5: Multi-domain zero-shot retrieval effectiveness comparisons using various sources for fine-tuning. Bold denotes the best model for that metric.

better zero-shot retrieval effectiveness in near-domain datasets compared to those fine-tuned on NQ, which is also observed by Ram et al. (2022).

Multi-Domain Retrieval Effectiveness. In addition, we evaluate zero-shot retrieval effectiveness on the multi-domain BEIR dataset, reported in Table 5. We evaluate the models fine-tuned on three different sources: MS MARCO, NQ, and TQA. Similarly, Aggretriever shows better zero-shot retrieval effectiveness compared to its `[CLS]` counterpart with any backbone. For example, our model consistently and substantially outperforms the comparable baselines using MS MARCO and TQA as the source dataset for fine-tuning. Although models fine-tuned on NQ show the worst zero-shot retrieval capability, Aggretriever with any backbone still slightly outperforms its `[CLS]` counterpart. It is also worth mentioning that Aggretriever with any backbone fine-tuned on MS MARCO outperforms the strong BM25 baseline.

5.3 Fine-Tuning with Noisy Hard Negatives

In this experiment, we use DistilBERT$_{AGG}$ to examine Aggretriever's robustness to fine-tuning with noisy hard negatives. Following TCT (Lin et al., 2021b) and RocketQA (Qu et al., 2021), for each query in the MS MARCO training set, we retrieve the top-200 candidates using DistilBERT$_{AGG}$ and further fine-tune the model by randomly sampling the candidates as negatives for two additional epochs using the same settings as the previous fine-tuning setup.

The results are listed in Table 6; we directly copy the numbers of TCT and RocketQA from the original papers. We notice that hard negatives reduce the effectiveness of both TCT and RocketQA since there are many false negatives in the candidates, as noted by Qu et al. (2021). They address this issue using expensive training strategies: knowledge distillation, denoising, and cross-batch negative sampling. On the other hand, DistilBERT$_{AGG}$ obtains competitive retrieval effectiveness without any expensive training strategies. This experiment demonstrates

Model	batch size	MARCO Dev RR@10	MARCO Dev R@1K
RocketQA (Qu et al., 2021)			
BM25 Neg.	8K	0.333	–
+ Hard Neg.	4K	0.260	–
+ Denoise	4K	0.364	–
+ Data Aug.	4K	0.370	0.979
TCT (Lin et al., 2021b)			
BM25 Neg. + KD	96	0.344	0.967
+ Hard Neg.	96	0.237	0.929
+ KD	96	0.359	0.970
DistilBERT$_{\text{AGG}}$			
BM25 Neg.	64	0.341	0.960
+ Hard Neg.	64	0.360	0.967

Table 6: Fine-tuning with noisy hard negatives.

that Aggretriever is robust and able to extract useful information when fine-tuned with hard negatives.

5.4 Ablation Study

In this experiment, we use DistilBERT$_{\text{AGG}}$ fine-tuned on the MS MARCO dataset to conduct an ablation study. In addition to MARCO Dev, to understand the zero-shot effectiveness of each condition, we conduct retrieval on a subset of BEIR (denoted BEIR small), consisting of five datasets from different domains: NFCorpus, FiQA, ArguAna, SCIDOCS, and SciFact. We report nDCG@10 averaged over these five datasets.

Dimensionality Ablation. We first study the effects of dimensionality on the `[CLS]` and **agg*** vectors in Table 7. We find that `[CLS]` alone slightly outperforms **agg*** alone (row 1 vs 4) on in-domain evaluation while the reverse trend is seen on zero-shot evaluation. This observation indicates that the `[CLS]` and **agg*** vectors encode text in different ways and that combining them further improves retrieval effectiveness (row 5). Compared to `[CLS]` alone and **agg*** alone, we still see a slight improvement for in-domain evaluation at 256 dimensions (row 6 vs 1 and 4). Holding the number of dimensions constant (rows 1–4), the best condition (row 3) indicates that the **agg*** vector requires more space than the `[CLS]` vector.

	Dim.		MARCO Dev		BEIR small
	`[CLS]`	**agg***	RR@10	R@1K	nDCG@10
(1)	768	0	0.308	0.940	0.259
(2)	640	128	0.327	0.954	0.307
(3)	128	640	0.341	0.960	0.355
(4)	0	768	0.307	0.926	0.328
(5)	768	768	0.350	0.966	0.358
(6)	128	128	0.320	0.946	0.300
(7)	0	30522	0.345	0.956	0.363

Table 7: DistilBERT$_{\text{AGG}}$ dimensionality ablation.

Finally, we report the retrieval effectiveness of the original wordpiece lexical representations before pruning (row 7), which can be considered the effectiveness upper bound of **agg***. Although **agg*** with 768 dimensions has lower effectiveness (row 4 vs 7), combined with `[CLS]`, Aggretriever reduces the gap (rows 3, 5 vs 7), with better retrieval efficiency in terms of smaller index size and lower retrieval latency. For example, on the MS MARCO dataset, representing each passage as a 768-dimensional vector in a Faiss Flat index with 32 (16) bits requires 26 (13) GB and 100 ms/q retrieval latency on a single V100 GPU, while the 30522-dimensional vectors (without pruning) require around 40 times more index storage and are not practical for end-to-end retrieval.

Pooling Stage Ablation. In the second ablation experiment, we fix `[CLS]` and **agg*** to 128 and 640 dimensions, respectively, and compare different designs of the pooling stage to form **agg***, as discussed in Section 3.1. The results are reported in the first main block of Table 8; row 1 is our default condition. In row 2, we remove the term importance component and assign a term weight of one for weighted max pooling. A substantial drop in retrieval effectiveness can be observed. In row 3, we remove MLM projection and represent each query (or passage) token with the 30522-dimensional indicator vector in Eq. (2); that is, $\mathbf{p}_{q_i} = x_j \in \{0,1\}^{|\text{V}_{\text{BERT}}|}$ for $j \in \{\text{token\_id}(q_i)\}$. We notice that skipping the MLM projector modestly harms retrieval effectiveness. This means that most textual information can be captured without the MLM projector, but it *does* help. This is sensible since

	Pooling		Pruning	MARCO Dev		BEIR small
	MLM	Weight		RR@10	R@1K	nDCG@10
(1)	✓	✓	full aggregation	0.341	0.960	0.355
(2)	✓	✗	full aggregation	0.308	0.937	0.308
(3)	✗	✓	full aggregation	0.332	0.953	0.355
(4)	✓	✓	semi aggregation	0.341	0.960	0.322
(5)	✓	✓	`linear`($\lvert V_{\text{BERT}}\rvert \rightarrow 640$)	0.327	0.959	0.313
(6)	AVERAGE		`linear`($768 \rightarrow 640$)	0.300	0.933	0.270
(7)	RepBERT (Zhan et al., 2020)		–	0.306	0.942	0.264

Table 8: DistilBERT$_{\text{AGG}}$ text aggregation ablation. We project `[CLS]` to 128 dimensions and concatenate with a 640-dimensional embedding pooled and pruned using different strategies. AVERAGE denotes average pooling over all 768-dimensional contextualized token embeddings other than `[CLS]`.

the 30522-dimensional indicator vector still retains each original query (or passage) term. A comparison of row 2 and row 3 shows that learned term weights for each token are more important than the term semantic distribution (projected by MLM) over the wordpiece vocabulary.

Pruning Stage Ablation. In the second main block of Table 8, we study the effects of pruning wordpiece lexical representations on Aggretriever. For example, we semi-aggregate (linearly project) the lexical representations into 640-dimensional dense vectors, as shown in row 4 (5). We observe that our non-parametric pruning approaches are better than the learned ones (rows 1, 4 vs 5). Although **agg**$^{+}$ shows the same retrieval effectiveness as **agg*** on in-domain evaluation, a substantial drop can be observed on out-of-domain evaluation (row 1 vs 4). This result demonstrates that our fully aggregated representations better preserve information from lexical representations and appear to be more robust to domain shifts.

We observe that directly projecting averaged contextualized embedding (excluding the `[CLS]`), denoted AVERAGE, into 640 dimensions, and then concatenating with `[CLS]` (row 6), does not perform well, indicating that projecting contextualized token embeddings into the high-dimensional wordpiece lexical space before pooling is key to preserving lexical information. Finally, we also try average pooling over all contextualized embeddings (including the `[CLS]`), which corresponds to RepBERT (Zhan et al., 2020). This yields negligible effectiveness difference from AVERAGE concatenated with `[CLS]` (row 7 vs 6); i.e., 0.306 (RR@10) and 0.264 (nDCG@10) on MARCO dev and BEIR small, respectively.

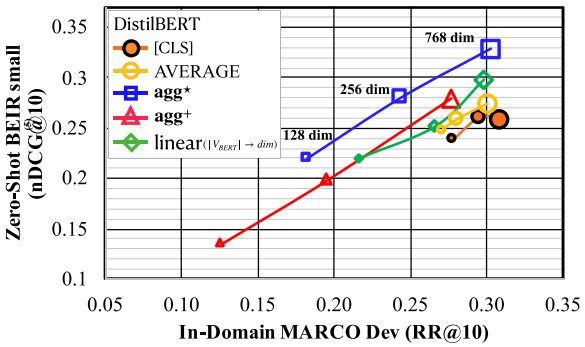


Figure 4: In-domain versus zero-shot effectiveness comparisons between textual representations under different numbers of dimensions.

To further understand the differences between pruned lexical representations (rows 1, 4, 5 in Table 8), we fine-tune DistilBERT using each representation alone (without using `[CLS]`) with 128, 256, and 768 dimensions on the MS MARCO dataset and compare their retrieval effectiveness on MS MARCO Dev and BEIR small in Figure 4. We observe that **agg*** performs better than **agg**$^{+}$ under all conditions, demonstrating that distributing representations to the full vector space can mitigate the problem of term misalignment (rectangles vs triangles) mentioned in Section 3.2, especially when the number of dimensions is small. Although the linearly projected lexical representations (diamonds) show better in-domain retrieval effectiveness than our non-parametric

pruning approaches (**agg**$^+$ and **agg***) with 128 and 256 dimensions, **agg*** still exhibits better zero-shot retrieval effectiveness. This indicates that the learned linear projector helps compress textual information into low-dimensional space in a way that is biased toward the training data.

In addition, in Figure 4, we also show the retrieval effectiveness of [CLS] and AVERAGE (solid and hollow circles) as comparisons. We observe that although all 768-dimensional textual representations reach similar in-domain retrieval effectiveness, [CLS] and AVERAGE show poor zero-shot retrieval effectiveness on BEIR small compared to the other models pruned from 30K-dimensional lexical representations. We hypothesize that [CLS] and AVERAGE capture textual information in a different manner than our lexical representations. This explains why fusing [CLS] with pruned lexical representations performs better than AVERAGE (rows 1, 4, 5 vs 6 in Table 8).

However, [CLS] and AVERAGE do not exhibit much retrieval effectiveness drop on both in-domain and zero-shot evaluations when reducing the number of dimensions. This is probably because lexical representations contain fine-grained textual information in 30K-dimensional lexical space while [CLS] and AVERAGE embeddings capture high-level textual information in low-dimensional semantic space. This result also explains the optimal balance in Table 7, where **agg*** requires more space than [CLS] when restricting the total vector dimension to 768.

5.5 Query Encoding Latency

Although different single-vector dense retrievers with the same vector dimensionality have similar retrieval latency under the same software and environment when performing top-k retrieval, query encoding latency is also an important component to consider. In this experiment, we compare the query encoding latency of DistilBERT$_{\text{AGG}}$ and DistilBERT$_{\text{CLS}}$. We measure the time required to encode the 6980 queries from MARCO Dev with batch size one on the CPU and GPU, using one thread on a Linux machine with a 2.2 GHz Intel Xeon Silver 4210 CPU and a single Tesla V100 GPU (32GB), respectively. We report the latency at 1$^{\text{st}}$, 50$^{\text{th}}$, and 99$^{\text{th}}$ percentiles in Table 9.

We observe that query encoding with Aggretriever is slightly slower than its [CLS] counterpart on the GPU (row 2 vs 1). On the CPU, the gap is much larger, especially for tail queries. However, from row 3 (the same condition as row 3 in Table 8), we see that skipping the MLM head projection step reduces the query encoding latency with only a small retrieval effectiveness loss. For a real-world application, this might be a sensible option, bringing query encoding latency roughly in line with the [CLS]-only model.

		latency (1$^{\text{st}}$ / 50$^{\text{th}}$ / 99$^{\text{th}}$ perc.)	
		CPU	GPU
(1)	DistilBERT$_{\text{CLS}}$	93 / 103 / 122 ms	15 / 16 / 18 ms
(2)	DistilBERT$_{\text{AGG}}$	155 / 163 / 191 ms	18 / 19 / 24 ms
(3)	w/o MLM	103 / 109 / 138 ms	16 / 19 / 20 ms

Table 9: Query encoding latency comparisons.

5.6 Comparison with Sparse Retrievers

In our final set of experiments, we compare Aggretriever and sparse retrievers since we borrow ideas from existing learned sparse retrieval models such as SPLADE-max (Formal et al., 2021a), which uses a different activation function after the MLM projector and adds sparsity regularization to generate sparse lexical representations for inverted indexes. For comparison to a sparse retriever without MLM projection, we use uniCOIL without expansions from T5 (Nogueira and Lin, 2019). Both models are fine-tuned on MS MARCO with BM25 negatives; thus, they represent reasonably fair comparisons to DistilBERT$_{\text{AGG}}$ and its variant without MLM, respectively (although uniCOIL uses BERT as a backbone). We index and evaluate SPLADE-max and uniCOIL using the code provided by Formal et al. (2021a)[4] and Pyserini (Lin et al., 2021a), respectively.[5]

Results are shown in Table 10. We first observe that DistilBERT$_{\text{CLS}}$ shows competitive in-domain retrieval effectiveness but underperforms sparse retrievers on out-of-domain evaluations (row 1 vs 5). This indicates that sparse retrieval using lexical matching has better generalization across retrieval tasks than dense retrieval with [CLS] alone. On the other hand, DistilBERT$_{\text{AGG}}$ and its variant show equally good generalization capability compared to the sparse retrievers (rows 2, 3

[4] https://github.com/naver/splade.

[5] Note that the BEIR figures for SPLADE-max reported in Formal et al. (2021a) do not include CQADupStack and use Tóuche-2020 (v1) instead of Tóuche-2020 (v2).

	MARCO Dev		BEIR
	RR@10	R@1K	nDCG@10
(1) DistilBERT$_{CLS}$	0.308	0.940	0.364
(2) DistilBERT$_{AGG}$	0.341	0.960	0.450
(3) w/o MLM	0.332	0.953	0.445
(4) SPLADE-max	0.340	0.965	0.447
(5) w/o MLM*	0.315	0.924	0.441

* uniCOIL w/o expansion (Lin and Ma, 2021) can be considered a variant of SPLADE-max w/o MLM.

Table 10: Comparison with sparse retrievers.

vs 4, 5). We attribute the transferability of Aggretriever to **agg***, which effectively aggregates and preserves information from wordpiece lexical representations.

Finally, we observe that without the MLM projector, the effectiveness of the sparse retrievers degrades, especially on in-domain evaluation (row 4 vs 5), while **agg*** only sees a slight degradation (row 2 vs 3). We hypothesize that the MLM projector helps sparse retrievers learn semantic matching as well as exact term matching. In contrast, Aggretriever can still learn semantic matching, even without the MLM projector, because it benefits from fusion with the `[CLS]` vector.

6 Related Work

Dense Retrieval. The most related line of research to our own work is the literature on how to effectively fine-tune a single-vector dense retriever. On the one hand, some researchers propose computationally expensive fine-tuning techniques such as hard negative mining strategies (Xiong et al., 2021; Zhan et al., 2021b), knowledge distillation (Lin et al., 2021b; Hofstätter et al., 2021), or their combination (Qu et al., 2021). On the other hand, others leverage further pre-training to improve the subsequent fine-tuning (Lee et al., 2019; Gao et al., 2021b; Lu et al., 2021; Gao and Callan, 2021; Izacard et al., 2021; Gao and Callan, 2022; Liu and Shao, 2022). As far as we are aware, our work is the first to discuss how to fine-tune dense retrieval models to effectively aggregate textual information from the pre-trained MLM head rather than directly using the `[CLS]` vector or contextualized embeddings from max or average pooling (Reimers and Gurevych, 2019).

Sparse Retrieval. Previous work (Bai et al., 2020; Mallia et al., 2021; Formal et al., 2021b; Lin and Ma, 2021) has demonstrated that projecting contextualized token embeddings into a high-dimensional vector in the wordpiece vocabulary space is an effective way to represent token-level information from transformers for lexical matching. These models directly feed the high-dimensional vectors into an inverted index for retrieval. Thus, sparsity control for effectiveness–efficiency tradeoffs involves additional considerations (Mackenzie et al., 2021). In contrast, our approach converts high-dimensional vectors into low-dimensional ones where top-k retrieval can be performed directly using ANN search libraries (Guo et al., 2020; Johnson et al., 2021).

Hybrid Retrieval. Our work can be characterized as hybrid since we ‘‘fuse’’ semantic and lexical representations into a single dense vector. Recent work (Gao et al., 2021a; Hofstätter et al., 2022; Shen et al., 2022; Lin and Lin, 2022) proposes to jointly train `[CLS]` and token-level representations for semantic and lexical matching, respectively. The two kinds of representations require different implementations for top-k retrieval, so multiple software stacks are required to perform retrieval. In contrast, our representations retain the best of semantic and lexical matching, but entirely as dense vectors. Thus, retrieval can be performed in a simple execution environment.

7 Conclusion and Future Work

In this paper, we present Aggretriever, a single-vector dense retrieval model that exploits all contextualized token embeddings from the input to BERT. We introduce a simple approach to aggregate the contextualized token embeddings into a dense vector, **agg***. Experiments show that **agg*** combined with the standard `[CLS]` vector achieves better retrieval effectiveness than using the `[CLS]` vector alone for both in-domain and zero-shot evaluations. Our work demonstrates that MLM pre-trained transformers can be fine-tuned into effective dense retrievers without further pre-training or expensive fine-tuning strategies.

Our work leads to a few open questions for future research: (1) Since we have demonstrated that Aggretriever still benefits from further pre-training, can we design additional pre-training tasks tailored directly to our model? The design

of these tasks, of course, needs to be mindful of the computational costs. (2) Can we apply current state-of-the-art compression techniques to Aggretriever? Zhan et al. (2021a, 2022) has shown that 768-dimensional dense representations can be effectively compressed into much smaller vectors. However, it is still unknown if these techniques can be applied to Aggretriever to retain both in-domain and zero-shot retrieval effectiveness. (3) Finally, can we apply Aggretriever to multi-lingual retrieval? Because, in a multi-lingual BERT model, the MLM head can project into tokens in multiple languages, we can envision a natural extension. However, as shown in Section 5.5, MLM projection is expensive, and the issue becomes worse when using a pre-trained multi-lingual model since the vocabulary size is usually even larger.

Acknowledgments

This research was supported in part by the Canada First Research Excellence Fund and the Natural Sciences and Engineering Research Council (NSERC) of Canada. We thank the anonymous referees who provided useful feedback to improve this work.

A Appendix

A.1 Implementation Details

We implement our models using Tevatron (Gao et al., 2022) and apply its default training settings in most tasks. For MS MARCO, we train models for three epochs with a learning rate $5e-6$, and for each batch, we include 8 queries. Each of the queries is paired with a randomly sampled positive passage and 7 negative passages mined using BM25. The maximum query and passage lengths are set to 32 and 128, respectively. Note that we use the official training set and corpus[6] instead of the ones in Tevatron, which are further processed by Qu et al. (2021). For open-domain QA, we follow the original settings used by Karpukhin et al. (2020) except for two modifications: (1) we use shared instead of independent weights between the query and passage encoders; (2) we set the maximum query and passage lengths to 32 and 156 for faster fine-tuning and inference.

[6] https://microsoft.github.io/msmarco/TREC-Deep-Learning-2019.

	w/o pre-training			w/ pre-training			
	DistilBERT$_{\text{AGG}}$	TAS-B	CL-DRD	coCondenser$_{\text{AGG}}$	coCondenser	Contriever	GTR-Base
KD	✗	✓	✓	✗	✗	✗	✓
HNM	✗	✗	✓	✗	✓	✓	✓
batch size >1K	✗	✗	✗	✗	✗	✓	✓
MARCO				RR@10			
Dev	0.341	0.344	0.381	0.363	0.382*	0.341	0.366
BEIR				nDCG@10			
TREC-COVID	0.661	0.481	0.584	0.751	0.712	0.596	0.539
NFCorpus	0.297	0.319	0.315	0.323	0.325	0.328	0.308
NQ	0.474	0.463	0.500	0.490	0.487	0.498	0.495
HotpotQA	0.616	0.584	0.589	0.609	0.563	0.638	0.535
FiQA-2018	0.292	0.300	0.308	0.305	0.276	0.329	0.349
ArguAna	0.417	0.429	0.413	0.438	0.299	0.446	0.511
Tóuche-2020 (v2)	0.263	0.162	0.203	0.213	0.191	0.230	0.205
Quora	0.834	0.835	0.826	0.851	0.856	0.865	0.881
DBPedia	0.362	0.384	0.381	0.380	0.363	0.413	0.347
SCIDOCS	0.138	0.149	0.146	0.143	0.137	0.165	0.149
FEVER	0.781	0.700	0.734	0.600	0.495	0.758	0.660
Climate-FEVER	0.210	0.228	0.204	0.155	0.144	0.237	0.241
SciFact	0.630	0.643	0.621	0.650	0.615	0.677	0.600
CQADupStack	0.318	0.314	0.325	0.338	0.320	0.345	0.357
Avg.nDCG@10	0.450	0.428	0.439	0.446	0.413	0.466	0.441

* These numbers are not comparable due to the use of a ''non-standard'' MS MARCO passage corpus that has been augmented with title.

Table 11: Comparisons with existing DPR models.

Note that we use one and four Tesla V100 GPUs (32GB) for model fine-tuning on MS MARCO and open-domain QA, respectively. For BEIR evaluation, we use the APIs provided by Thakur et al. (2021) and set maximum query and passage input lengths to 512.[7]

A.2 Comparison with Existing DPR Models

Table 11 compares Aggretriever with existing dense retrievers fine-tuned with more expensive strategies; i.e., cross-encoder knowledge distillation (KD), hard negative mining (HNM), and large in-batch negatives, on both in-domain and out-of-domain evaluations. The two baseline models without further pre-training are: (1) TAS-B (Hofstätter et al., 2021), which distills ColBERT and a cross-encoder to DPR with an efficient topic-aware sampling strategy; (2) CL-DRD (Zeng et al., 2022), which further improves TAS-B by combining curriculum learning, HNM, and cross-encoder KD. Three models with further pre-training are included: (1) coCondenser (Gao and Callan, 2022), already discussed in Section 4.2; (2) Contriever (Izacard et al., 2021), which leverages pre-training by combining advanced contrastive learning techniques with an Inverse Cloze Task (ICT) variant; (3) GTR-Base

[7] https://github.com/beir-cellar/beir.

(Ni et al., 2021), which trains a T5-Base encoder model that combines pre-training, KD, and HNM. For TAS-B, Contriever, and GTR-Base, we directly copy numbers from Izacard et al. (2021) and Ni et al. (2021), respectively. For CL-DRD[8] and coCondenser,[9] we use the models provided by the authors to conduct in-domain and out-of-domain evaluations ourselves. Note that the coCondenser model provided by the authors is fine-tuned in another round with self-mined hard negatives. Furthermore, they use a ''non-standard'' MS MARCO corpus where each passage is concatenated with a title; thus, the MS MARCO Dev results are different from the values for $\text{coCondenser}_{\text{CLS}}$ reported in Table 2.

First, we observe that $\text{DistilBERT}_{\text{AGG}}$ is not only competitive with TAS-B on in-domain evaluation but also outperforms both TAS-B and CL-DRD on out-of-domain evaluation, without needing supervision from an expensive cross-encoder teacher. Secondly, Contriever yields the best out-of-domain results at the cost of in-domain effectiveness. On the other hand, $\text{coCondenser}_{\text{AGG}}$ reaches the same level of retrieval effectiveness as GTR-Base without leveraging any expensive fine-tuning strategies. Fine-tuning Aggretriever with KD, HNM, and large batch size is possible to further improve retrieval effectiveness, but these techniques are orthogonal to our proposed model.

References

Yang Bai, Xiaoguang Li, Gang Wang, Chaoliang Zhang, Lifeng Shang, Jun Xu, Zhaowei Wang, Fangshan Wang, and Qun Liu. 2020. SparTerm: Learning term-based sparse representation for fast text retrieval. *arXiv:2010.00768*.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. 2016. MS MARCO: A human generated machine reading comprehension dataset. *arXiv: 1611.09268*.

Wei-Cheng Chang, Felix X. Yu, Yin-Wen Chang, Yiming Yang, and Sanjiv Kumar. 2020. Pre-training tasks for embedding-based large-scale retrieval. In *Proceedings of ICLR*.

Nick Craswell, Bhaskar Mitra, and Daniel Campos. 2019. Overview of the TREC 2019 deep learning track. In *Proceedings of TREC*.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2020. Overview of the TREC 2020 deep learning track. In *Proceedings of TREC*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*.

Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2021a. SPLADE v2: Sparse lexical and expansion model for information retrieval. *arXiv:2109.10086*. https://doi.org/10.1145/3404835.3463098

Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021b. SPLADE: Sparse lexical and expansion model for first stage ranking. In *Proceedings of SIGIR*, pages 2288–2292. https://doi.org/10.1145/3404835.3463098

Luyu Gao and Jamie Callan. 2021. Condenser: A pre-training architecture for dense retrieval. In *Proceedings of EMNLP*, pages 981–993. https://doi.org/10.18653/v1/2021.emnlp-main.75

Luyu Gao and Jamie Callan. 2022. Unsupervised corpus aware language model pre-training for dense passage retrieval. In *Proceedings of ACL*, pages 2843–2853. https://doi.org/10.18653/v1/2022.acl-long.203

Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021a. COIL: Revisit exact lexical match in information retrieval with contextualized inverted list. In *Proceedings of NAACL*, pages 3030–3042. https://doi.org/10.18653/v1/2021.naacl-main.241

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2022. Tevatron: An efficient and flexible toolkit for dense retrieval. *arxiv.2203.05765*.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021b. SimCSE: Simple contrastive learning

[8] https://github.com/HansiZeng/CL-DRD.

[9] https://huggingface.co/Luyu/co-condenser-marco-retriever.

of sentence embeddings. In *Proceedings of EMNLP*, pages 6894–6910.

Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *Proceedings of ICML*.

Sebastian Hofstätter, Sheng-Chieh Lin, Jheng-Hong Yang, Jimmy Lin, and Allan Hanbury. 2021. Efficiently teaching an effective dense retriever with balanced topic aware sampling. In *Proceedings of SIGIR*, pages 113–122. https://doi.org/10.1145/3404835.3462891

Sebastian Hofstätter, Omar Khattab, Sophia Althammer, Mete Sertkan, and Allan Hanbury. 2022. Introducing neural bag of whole-words with ColBERTer: Contextualized late interactions using enhanced reduction. *arXiv:2203.13088*. https://doi.org/10.1145/3511808.3557367

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised dense information retrieval with contrastive learning. *arXiv:2112.09118*.

Kyoung-Rok Jang, Junmo Kang, Giwon Hong, Sung-Hyon Myaeng, Joohee Park, Taewon Yoon, and Heecheol Seo. 2021. Ultra-high dimensional sparse representations with binarization for efficient text retrieval. In *Proceedings of EMNLP*, pages 1016–1029.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, pages 535–547. https://doi.org/10.1109/TBDATA.2019.2921572

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of ACL*, pages 1601–1611. https://doi.org/10.18653/v1/P17-1147

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of EMNLP*, pages 6769–6781. https://doi.org/10.18653/v1/2020.emnlp-main.550

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural Questions: A benchmark for question answering research. *Transactions of the Association of Computational Linguistics*, 7:452–466. https://doi.org/10.1162/tacl_a_00276

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. 2019. Latent retrieval for weakly supervised open domain question answering. In *Proceedings of ACL*, pages 6086–6096.

Jimmy Lin and Xueguang Ma. 2021. A few brief notes on DeepImpact, COIL, and a conceptual framework for information retrieval techniques. *arXiv:2106.14807*.

Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021a. Pyserini: A Python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of SIGIR*, pages 2356–2362. https://doi.org/10.1145/3404835.3463238

Sheng-Chieh Lin and Jimmy Lin. 2022. A dense representation framework for lexical and semantic matching. *arXiv:2206.09912*.

Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. 2021b. In-batch negatives for knowledge distillation with tightly-coupled teachers for dense retrieval. In *Proceedings of RepL4NLP*, pages 163–173.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv:1907.11692*.

Zheng Liu and Yingxia Shao. 2022. RetroMAE: Pre-training retrieval-oriented transformers via masked auto-encoder. *arXiv:2205.12035*.

Shuqi Lu, Di He, Chenyan Xiong, Guolin Ke, Waleed Malik, Zhicheng Dou, Paul Bennett, Tie-Yan Liu, and Arnold Overwijk. 2021. Less is more: Pretrain a strong Siamese encoder for dense text retrieval using a weak decoder. In *Proceedings of EMNLP*, pages 2780–2791. https://doi.org/10.18653/v1/2021.emnlp-main.220

Joel Mackenzie, Andrew Trotman, and Jimmy Lin. 2021. Wacky weights in learned sparse representations and the revenge of score-at-a-time query evaluation. *arXiv:2110.11540*.

Antonio Mallia, Omar Khattab, Torsten Suel, and Nicola Tonellotto. 2021. Learning passage impacts for inverted indexes. In *Proceedings of SIGIR*, pages 1723–1727. https://doi.org/10.1145/3404835.3463030

Jianmo Ni, Chen Qu, Jing Lu, Zhuyun Dai, Gustavo Hernández Ábrego, Ji Ma, Vincent Y. Zhao, Yi Luan, Keith B. Hall, Ming-Wei Chang, and Yinfei Yang. 2021. Large dual encoders are generalizable retrievers. *arXiv: 2112.07899*.

Rodrigo Nogueira and Jimmy Lin. 2019. From doc2query to docTTTTTquery. https://api.semanticscholar.org/CorpusID:208612557.

Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. 2021. RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering. In *Proceedings of NAACL*, pages 5835–5847. https://doi.org/10.18653/v1/2021.naacl-main.466

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of EMNLP*, pages 2383–2392. https://doi.org/10.18653/v1/D16-1264

Ori Ram, Gal Shachaf, Omer Levy, Jonathan Berant, and Amir Globerson. 2022. Learning to retrieve passages without supervision. In *Proceedings of NAACL*, pages 2687–2700. https://doi.org/10.18653/v1/2022.naacl-main.193

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of EMNLP*, pages 3982–3992. https://doi.org/10.18653/v1/D19-1410

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *arXiv:1910.01108*.

Christopher Sciavolino, Zexuan Zhong, Jinhyuk Lee, and Danqi Chen. 2021. Simple entity-centric questions challenge dense retrievers. In *Proceedings of EMNLP*. https://doi.org/10.18653/v1/2021.emnlp-main.496

Tao Shen, Xiubo Geng, Chongyang Tao, Can Xu, Kai Zhang, and Daxin Jiang. 2022. Unifier: A unified retriever for large-scale retrieval. *arXiv:2205.11194*.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Proceedings of NIPS*.

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

Jheng-Hong Yang, Xueguang Ma, and Jimmy Lin. 2021. Sparsifying sparse representations for passage retrieval by top-k masking. *arXiv: 2112.09628*.

Hansi Zeng, Hamed Zamani, and Vishwa Vinay. 2022. Curriculum learning for dense retrieval distillation. In *Proceedings of SIGIR*, pages 1979–1983. https://doi.org/10.1145/3477495.3531791

Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2021a.

Jointly optimizing query encoder and product quantization to improve retrieval performance. In *Proceedings of CIKM*, pages 2487–2496. https://doi.org/10.1145/3459637.3482358

Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2021b. Optimizing dense retrieval model training with hard negatives. In *Proceedings of SIGIR*, pages 1503–1512. https://doi.org/10.1145/3404835.3462880

Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2022. Learning discrete representations via constrained clustering for effective and efficient dense retrieval. In *Proceedings of WSDM*, pages 1328–1336. https://doi.org/10.1145/3488560.3498443

Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. 2020. RepBERT: Contextualized text embeddings for first-stage retrieval. *arXiv:2006.15498*.