

Parameter-Efficient Sparse Retrievers and Rerankers using Adapters

Vaishali Pal^{1,2}[0000-0002-1493-3659], Carlos Lassance²[0000-0002-7754-6656],
Hervé Déjean²[0000-0002-9837-5358], and Stéphane
Clinchant²[0000-0003-2367-8837]

¹ IRLab, University of Amsterdam, Amsterdam, Netherlands

² Naver Labs Europe, Meylan, France

Abstract. Parameter-Efficient transfer learning with Adapters have been studied in Natural Language Processing (NLP) as an alternative to full fine-tuning. Adapters are memory-efficient and scale well with downstream tasks by training small bottle-neck layers added between transformer layers while keeping the large pretrained language model (PLMs) frozen. In spite of showing promising results in NLP, these methods are under-explored in Information Retrieval. While previous studies have only experimented with dense retriever or in a cross lingual retrieval scenario, in this paper we aim to complete the picture on the use of adapters in IR. First, we study adapters for SPLADE, a sparse retriever, for which adapters not only retain the efficiency and effectiveness otherwise achieved by finetuning, but are memory-efficient and orders of magnitude lighter to train. We observe that Adapters-SPLADE not only optimizes just 2% of training parameters, but outperforms fully finetuned counterpart and existing parameter-efficient dense IR models on IR benchmark datasets. Secondly, we address domain adaptation of neural retrieval thanks to adapters on cross-domain BEIR datasets and TripClick. Finally, we also consider knowledge sharing between rerankers and first stage rankers. Overall, our study complete the examination of adapters for neural IR.³

Keywords: Adapters · Information Retrieval · Sparse Neural Retriever

1 Introduction

Information Retrieval (IR) systems often aim to return a ranked list of documents ordered with respect to their relevance to a user query. In modern web search engines, there is, in fact, not a single retrieval model but several ones specialized in diverse information needs such as different search verticals. To add to this complexity, multi-stage retrieval considers effectiveness-efficiency trade-off where first stage retrievers are essential for fast retrieval of potentially relevant candidate documents from a large corpus. Further down the pipeline, rerankers are added focusing on effectiveness.

³ The code can be found at: <https://github.com/naver/splade/tree/adapters-splade>

With the advent of large Pretrained Language Models (PLM), recent neural retrieval models have millions of parameters. Training, updating and adapting such models implies significant computing and storage cost calling for efficient methods. Moreover, generalizability across out-of-domain datasets is critical and even when effectively adapted to new domains, full finetuning often comes at the expense of large storage and catastrophic forgetting. Fortunately, such research questions have already been studied in the NLP literature [1,2,9,10] with parameter-efficient tuning. In spite of very recent work exploring parameter-efficient techniques for neural retrieval, the use of adapters in IR has been overlooked. Previous work on dense retriever had mixed results [11] and successful adaptation was achieved for cross lingual retrieval [17]. Our study aims to complete the examination of adapters for neural IR and investigates it with neural sparse retrievers. We study ablation of adapter layers to analyze whether all layers contribute equally. We examine how adapter-tuned neural sparse retriever SPLADE [5] fares on benchmark IR datasets MS MARCO [21], TREC DL 2019 and 2020 [3] and out-of-domain BEIR datasets [30]. We explore whether generalizability of SPLADE can be further improved with adapter-tuning on BEIR and out-of-domain dataset such as TripClick [26]. In addition, we examine knowledge transfer between first stage retrievers and rerankers with full fine-tuning and adapter-tuning. To the best of our knowledge, this is the first work which studies adapters on sparse retrievers, focuses on sparse models’ generalizability and explores knowledge transfer between retrievers in different stages of the retrieval pipeline. In summary, we address the following research questions:

1. RQ1: What is the efficiency-accuracy trade-off of parameter-efficient fine-tuning with adapters on the sparse retriever model SPLADE?
2. RQ2: How does each adapter layer ablation affect retrieval effectiveness?
3. RQ3: Are adapters effective for adapting neural sparse neural retrieval in a new domain?
4. RQ4: Could adapters be used to share knowledge between rerankers and first stage rankers?

2 Background and Related Work

Parameter efficient transfer learning techniques aim to adapt large pretrained models to downstream tasks using a fraction of training parameters, achieving comparable effectiveness to full fine-tuning. Such methods [9,15,10,25,28] are memory efficient and scale well to numerous downstream tasks due to the massive reduction in task specific trainable parameters. This makes them an attractive solution for efficient storage and deployment compared to fully fine-tuned instances. Such methods have been successfully applied to language translation [25], natural language generation [16], Tabular Question Answering [22], and on the GLUE benchmark [7,28]. In spite of all its advantages and a large research footprint in NLP, parameter-efficient methods remain under-explored in IR.

A recent comprehensive study [4] categorises parameter efficient transfer learning into 3 categories: 1) Addition based 2) Specification based 3) Reparam-

eterization based. Addition based methods insert intermediate modules into the pretrained model. The newly added modules are adapted to the downstream task while keeping the rest of the pretrained model frozen. The modules can be added vertically by increasing the model depth as observed in Houlsby Adapters [9] and Pfeiffer Adapters [25]. Houlsby Adapters insert small bottle-neck layers after both the multi-head attention and feed-forward layer of the each transformer layer which are optimized for NLP tasks on GLUE benchmark. Pfeiffer Adapter inserts the bottle-neck layer after only the feed-forward layer and has shown comparable effectiveness to fine-tuning on various NLP tasks. Prompt-based adapter methods such as Prefix-tuning [15] prepend continuous task-specific vectors to the input sequence which are optimized as free-parameters. Compacter [20] hypothesizes that the model can be optimized by learning transformations of the bottle-neck layer in a low-rank subspace leading to less parameters.

Specification based methods fine-tune only a subset of pretrained model parameters to the task-at-hand while keeping the rest of the model frozen. The fine-tuned model parameters can be only the bias terms as observed in BitFit [2], or only cross-attention weights as in the case of Seq2Seq models with X-Attention [6]. Re-parameterization methods transform the pretrained weights into parameter efficient form during training. This is observed in LoRA [10] which optimises rank decomposition matrices of pretrained layer while keeping the original layer frozen.

Recent studies exploring parameter efficient transfer learning for Information Retrieval show promising results of such techniques for dense retrieval models [11,17,19,29]. [11] studies parameter efficient prefix-tuning, [15] and LoRA [10] on bi-encoder and cross-encoder dense models. Additionally, they combine the two methods by sequentially optimizing one method for m epochs, freezing it and optimizing the other for n epochs. Their studies show that while cross-encoders with LoRA and LoRA+(50% more parameters compared to LoRA) outperform fine-tuning with TwinBERT [18] and ColBERT [13], parameter-efficient methods *do not outperform fine-tuning* for bi-encoders across all datasets. [17] uses parameter-efficient techniques such as Sparse Fine-Tuning Masks and Adapters for multilingual and cross-lingual retrieval tasks with rerankers. They train language adapters with Masked Language Modeling (MLM hereafter) task and then task-specific retrieval adapters. This enables the fusion of reranking adapter trained with source language data together with the language adapter of the target language. Concurrent to our work, [29] studies parameter-efficient prompt tuning techniques such as Prefix tuning and P-tuning v2, specification based methods such as BitFit and adapter-tuning with Pfeiffer Adapters on late interaction bi-encoder models such as Dense Passage Retrieval [12] and ColBERT. They are motivated by cross-domain generalization of dense retrievals and achieve better results with P-tuning compared to fine-tuning on the BEIR benchmark. [19] studies various parameter-efficient tuning procedures at both retrieval and re-ranking stages. They conduct a comprehensive study of parameter-efficient techniques such as BitFit, Prefix-tuning, Adapters, LoRA, MAM adapters with dense bi-encoders and cross-encoders with BERT-base as the backbone model.

Their parameter-efficient techniques achieve comparable effectiveness to fine-tuning on top-20 retrieval accuracy and marginal gains on top-100 retrieval accuracy.

Compared to prior works, our experiments first study the use of adapters for state of the art sparse models such as SPLADE, contrary to previous work that studied dense bi-encoder models⁴. Furthermore, our results show improvements compared to the previous studies. We also studied the case of using distinct adapters for query and document encoders in a “bi-adapter” setting where the same pretrained backbone model is used by both the query and the document encoder but different adapters are trained for the queries and documents. Secondly, we address another research questions ignored by previous work, which is efficient domain adaptation⁵ for neural first stage rankers. We start from a trained neural ranker and study adaptation with adapters on a different domain, such as the ones present in the BEIR benchmark. Finally, we also study parameters sharing between rerankers and first stage rankers using adapters, which to our knowledge has not been studied yet.

3 Parameter-Efficient Retrieval with Adapters

In this section, we first present the self-attention used in transformers and how the adapters we use for our experiments interact with them. We then introduce the models used for first stage ranking and reranking.

3.1 Self-Attention Transformer Layers

Large pretrained language models are based on the transformer architecture composed of N stacked transformer layers. Each transformer layer comprises of a fully connected feed-forward module and a multi-headed self attention module. Each attention layer has a function of query matrix ($Q \in R^{n \times d_k}$), a key matrix and a value matrix. The attention can be formally written as:

$$A(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

where the query Q , key K and value V are parameterized by weight matrices $W_q \in R^{n \times d_k}$, $W_k \in R^{n \times d_k}$, and $W_v \in R^{n \times d_v}$, as $Q = XW_q$, $K = XW_k$ and $V = XW_v$. Each of the N heads has its respective Q_i , V_i and K_i weights and its corresponding attention A_i . The feed-forward layer takes as input a transformation of the concatenation of the N attentions as:

$$FFN(x) = \sigma(XW_1 + b_1)W_2 + b_2 \quad (2)$$

where $\sigma(\cdot)$ is the activation function. A residual connection is further added after each attention layer and feed-forward layer.

⁴ To the best of our knowledge the only work involving SPLADE and adapters/freezing layers is [32], which found that freezing the embeddings improves effectiveness.

⁵ Here we use adaptation as further finetuning on the target domain.

3.2 Adapters

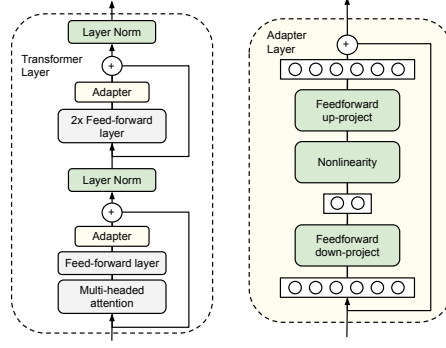


Fig. 1. Houdsby Adapter, image from the original paper [9]

In this paper, we focus on the Houdsby adapter [9], which as described in Section 3 can be considered an additive adapter and is depicted in Figure 1. An additive adapter inserts trainable parameters in addition to the aforementioned transformer layers. The added modules form a bottle-neck architecture with a down-projection, an up-projection and a non-linear transformation. The size of the bottle-neck controls the number of training parameters in an adapter layer. Additionally, a residual connection is applied across each adapter layers. Finally, a layer normalization is added after each transformer sublayer. Formally, this is defined as:

$$x = f(hW_{down})W_{up} + x \quad (3)$$

where $x \in R^d$ is the input to the adapter layer, $W_{down} \in R^{d \times r}$ is the down projection matrix transforming input x into bottle-neck dimension d , $W_{up} \in R^{r \times d}$ is the up projection matrix transforming the bottle-neck representation back to the d -dimensional space. Each adapter layer is initialized with a near-identity weights to enable stable training.

3.3 Neural Sparse First Stage Retrievers

Neural sparse first stage retrievers learn contextualized representations of documents and queries in a sparse high-dimensional latent space. In this work, we focus on SPLADE sparse retriever [5,14], which uses both L_1 and $FLOPS$ regularizations to force sparsity. We freeze the pretrained language model while training the adapter layers. SPLADE predicts term weights of each vocabulary token j with respect to an input token i as:

$$w_{ij} = transform(h_i)^T E_j + b_j \quad j \in 1, \dots, |V| \quad (4)$$

where E_j is the j^{th} vocabulary token embedding, b_j is it's bias, h_i is i^{th} input token embedding, $transform(.)$ is a linear transformation followed by GeLU activation and LayerNorm. The final term importance for each vocabulary term j is obtained by taking the maximum predicted weights over the entire input sequence of length n , after applying a log-saturation effect:

$$w_j = \max_n \log(1 + ReLU(w_{ij})) \quad (5)$$

Given a query q_i , the ranking score s of a document d is defined by the degree to which it is relevant to q obtained as a dot product $s(q, d) = w(q).w(d)$. The learning objective is to discriminate representations obtained from Equation 5 of a relevant document d^+ and non-relevant hard-negatives d^- obtained from BM25 and in-batch negatives $d_{i,j}^-$ by minimizing the contrastive loss:

$$L = -\log \frac{e^{s(q_i, d^+)}}{e^{s(q_i, d_i^+)} + e^{s(q_i, d_i^-)} + \sum_j e^{s(q_i, d_{i,j}^-)}} \quad (6)$$

SPLADE can be further improved with distillation. The learning objective here is to minimize the MarginMSE [5] loss: mean-squared-error between the positive negative margins of a cross-encoder teacher and the student:

$$L = MSE(M_s(q_i, d^+) - M_s(q_i, d^-), M_t(q_i, d^+) - M_t(q_i, d^-)) \quad (7)$$

where MSE is mean-squared error, M_t is the teacher's margin and M_s is the student's margin. The final objective optimizes either of the objective in Equation 6 or 7 with regularization losses:

$$\mathcal{L}_{SPLADE} = L + \lambda_q \mathcal{L}_1 + \lambda_d \mathcal{L}_{FLOPS} \quad (8)$$

$$\text{where } \mathcal{L}_{FLOPS} = \sum_{j \in V} \hat{a}_j^2 = \sum_{j \in V} \left(\frac{1}{N} \sum_{i=1}^N w_j^{d_i} \right) \quad (9)$$

The Flops regularizer is a smooth relaxation of the average number of floating-point operations necessary to compute the score of a document, and hence directly related to the retrieval time. It is defined using as a continuous relaxation of the activation (i.e. the term has a non zero weight) probability a_j for token j , and estimated for documents d in a batch of size N by \hat{a}_j^2 .

Retrieval Flops: SPLADE also reports the retrieval flops (noted R-FLOPS), i.e., the number of floating point operations on the inverted index to return the list of documents for a given query. The R-FLOPS metric is defined by an estimation of the average number of floating-point operations between a query and a document which is defined as the expectation $\mathbb{E}_{q,d} \left[\sum_{j \in V} p_j^{(q)} p_j^{(d)} \right]$ where p_j is the activation probability for token j in a document d or a query q . It is empirically estimated from a set of approximately 100k development queries, on the MS MARCO collection. It is thus an indication of the inverted index sparsity and of the computational cost for a sparse model (which is different from the inference ie forward cost of the model)

3.4 Cross-Encoding Rerankers

Another way to use PLMs for neural retrieval is to use what is called “cross-encoding” [33]. In this case, both query and document are concatenated before being provided to the network and the score is directly computed by the network. The cross-encoding procedure allows for networks that are much more effective, but this effectiveness comes with a cost on efficiency as the retrieval procedure now has to go through the entire network for each query document pair, instead of being able to precompute document representations and only go through the network for the query representation. The models are trained with a contrastive loss as seen in equation (6) that aims to maximize the score of the true query/document pair compared to a BM25 negative query/document pair, without using in-batch negatives.

4 Experimental Setting and Results

We use the SPLADE github repository⁶ to implement our modifications and followed the standard procedure to train SPLADE models. We implement our SPLADE models using an L_1 regularization for the query, and $FLOPS$ regularization for the document following [14]. Unless otherwise stated, the document regularization weight λ_d is set to $9e-5$ and the query regularization weight λ_q to $5e-4$ to train all variants of Adapters-SPLADE. In order to mitigate the contribution of the regularizer at the early stages of training, we follow [23] and use a scheduler for λ , quadratically increasing λ at each training iteration, until the $50k$ step. We use a learning rate of $8e-5$, a batch size of 128, a linear scheduler and warmup step of 6000. We set the maximum sequence length to 256. We train for 300k iterations and keep the best checkpoint using MRR@10 on the validation set. We use a bottle-neck reduction factor of 16 (i.e. 16 times smaller) for all adapter layers. We use PyTorch [24], Huggingface Transformers [31] and AdapterHub [1] to train all models on 4 Tesla V100 GPUs with 32GB memory. We compute statistical significance with $p \leq 0.05$ using the Student’s t-test and use superscripts to identify statistical significance for almost all measures safe for metrics related to BEIR⁷.

4.1 RQ1: Adapters-SPLADE

We study 2 different settings of encoding with adapters. The first called **adapter**, is a mono-encoder setup where the query and document shares a single encoder. The adapter layers are optimized with both the input sequences keeping the PLM frozen. The second setting inspired by the work on [14], is a bi-encoder setup which separates query and document encoders by training distinct query and document adapters on a shared frozen PLM. We call this setting **bi-adapter**.

⁶ <https://github.com/naver/splade>

⁷ due to lack of standard procedure.

This setting not only benefits from optimizing exclusive adapters for input sequence type (different lengths of query/document, etc.), it is also possible to use smaller PLMs for the queries instead of sharing PLM weights. We explore different backbone PLMs: DistilBERT and CC+MLM FLOPs, a pretrained PLM of cocondenser trained on the masked language model (MLM) task using the FLOPS regularization in order to make it easier to work with SPLADE, introduced in [14]. We trained and evaluated Adapter-SPLADE models on the MS MARCO passage ranking dataset [21] in full ranking setting. The results for finetuning with BM25 triplets are available in Table 1, whereas in Table 2 we make available the results of training models with distillation. For distillation, we use hard-negatives and scores generated by a cross-encoder reranker⁸ and the MarginMSE loss as described in [5] and set λ_d to 1e-2 and λ_q to 9e-2.

Table 1. Finetuning and adapter-tuning comparison using BM25 triplets for training.

Model	#	Method	MS MARCO dev		TREC DL 2019	TREC DL 2020	R-Flops	Training Params
			MRR@10	R@1000	NDCG@10	NDCG@10		
DistilBERT	a	finetuning	0.346	0.963	0.692	0.677	1.43	100%
	b	adapter	0.351	0.968 ^a	0.711	0.676	1.44	2.23%
	c	bi-adapter	0.352	0.967 ^a	0.690	0.666	0.74	2.23%
CC + MLM FLOPS	d	finetuning	0.366 ^{abc}	0.977 ^{abc}	0.712	0.684	1.09	100%
	e	adapter	0.376^{abcd}	0.980^{abcdf}	0.712	0.688	0.8	2.23%
	f	bi-adapter	0.372 ^{abc}	0.976 ^{abc}	0.701	0.700	0.37	2.23%

Table 2. Finetuning and adapter-tuning comparison using distillation training.

Model	#	Method	MS MARCO dev		TREC DL 2019	TREC DL 2020	R-Flops	Training Params
			MRR@10	R@1000	NDCG@10	NDCG@10		
DistilBERT	a	finetuning	0.371	0.979 ^b	0.727	0.711	3.93	100%
	b	adapter	0.373	0.975	0.728	0.716	1.86	2.16%
CC + MLM FLOPS	c	finetuning	0.388 ^{ab}	0.982 ^{ab}	0.734	0.732	4.38	100%
	d	adapter	0.390^{ab}	0.983^{ab}	0.740	0.729	2.34	2.16%

To study efficiency-effectiveness trade-off of Adapters-SPLADE, we compare effectiveness, R-FLOPS size and number of training parameters of adapter-tuned models with their baseline finetuned counterparts having the same backbone PLM. [23] first showed that R-FLOPs reduction is a reasonable measure of retrieval speed. R-FLOPS measure the average number of floating-point operations needed to compute a document score during retrieval. A sparse embedding and subsequently lower FLOP achieves a retrieval speedup of the order of $1/p^2$ over an inverted index where p is the probability of each document embedding dimension being non-zero.

⁸ <https://huggingface.co/cross-encoder/ms-marco-MiniLM-L-6-v2>

Overall, we observe, from Table 1 and 2, all variants of adapter-tuned SPLADE outperform all baseline fine-tuned counterparts on MS MARCO and TREC DL 2019. The distilled cocondenser with MLM mono-encoder model is the highest performing with an MRR@10 score of 0.390 and R@100 of 0.983. The difference in effectiveness between the mono-encoder and bi-encoder adapter-tuning is marginal and depends on the PLM. Most noteworthy, we also observe that the R-FLOPS are lower for adapter-tuned models indicating sparser representation than the fine-tuned counterparts. This is more pronounced in the adapter-tuned models with distillation. Finally, the **bi-adapter** models have even lower R-FLOPS than the mono-encoder settings, which shows that for the same effectiveness the bi-adapters models are more efficient and sparse. We also observe that the number of training parameters is only 2.23% of the total model parameters for *triplets* training (1.5M/67M for mono-adapter DistilBERT, 3M/135M for bi-adapter DistilBERT, 2M/111M for CC + MLM FLOPS) and 2.16% for the distillation process (1.5M/67M for mono-adapter DistilBERT, 2M/111M for CC + MLM FLOPS). This has direct consequence in low-hardware setting where adapters with lower number of number of training parameters and gradients can be trained on a smaller GPU(such as 24GB P40) but full finetuning is infeasible. Overall, there is a clear advantage in using Adapter-SPLADE over finetuning, which differs from the previous results on dense adapters [11].

We also evaluate with the full BEIR benchmark [41] comprising of 18 different datasets to measure generalizability of IR models with zero-shot effectiveness on out-of-domain data. The results are listed in Table 3. We observe from that in the mono-adapter Triplets training, **adapter** outperforms finetuning on mean nDCG@10 with the highest gap in arguana. With CC+MLM FLOPS as the backbone model, finetuning and adapter-tuning performs similarly. However, adapter scores drop on models trained with distillation. This can be attributed to the adapter representations being sparser compared to the finetuned models. As depicted by the R-FLOPS in Table 1, adapter-tuned DistilBERT has less than half the number of R-FLOPS than its finetuned counterpart whereas CC+MLM FLOPS finetuned model has approximately 1.87 times the number of R-FLOPS of the adapter-tuned model. This reflects in model representation capacity in 0-shot setting in Table 3. However, as discussed in Section 4.3, adapters are well suited for domain adaptation when trained on out-of-domain datasets keeping the backbone retriever intact and free from catastrophic forgetting.

4.2 RQ2: Adapter Layer Ablation

Furthermore, we perform extensive adapter layer ablation by progressively removing adapter layers from the early layers of the encoder. Doing so results in n separate models for each layer ablation setting. The frozen pretrained model for our ablation studies is DistilBERT in a mono-encoder setting where the same instance of the encoder is used to encode both the document and the query, which is the same configuration as the adapter method in Table 1. This results in a total of 6 configurations for the ablation study corresponding to the 6 adapter

Table 3. nDCG@10 score comparison on the BEIR zero-shot evaluation

Datasets	Triplets training				Distillation training			
	DistilBERT		CC + MLM FLOPS		DistilBERT		CC + MLM FLOPS	
	finetuning	adapter	finetuning	adapter	finetuning	adapter	finetuning	adapter
arguana	0.298	0.364	0.427	0.388	0.513	0.443	0.463	0.433
climate-fever	0.167	0.172	0.180	0.187	0.202	0.197	0.229	0.202
dbpedia-entity	0.379	0.392	0.388	0.401	0.419	0.417	0.438	0.432
fever	0.730	0.734	0.724	0.722	0.773	0.757	0.792	0.773
fiqa	0.295	0.289	0.317	0.320	0.332	0.314	0.342	0.337
hotpotqa	0.626	0.647	0.650	0.603	0.687	0.670	0.687	0.629
nfcopus	0.318	0.321	0.331	0.333	0.335	0.335	0.340	0.344
nq	0.481	0.482	0.506	0.523	0.522	0.508	0.539	0.544
quora	0.819	0.810	0.821	0.806	0.825	0.722	0.841	0.552
scidocs	0.143	0.150	0.151	0.153	0.154	0.147	0.152	0.153
scifact	0.614	0.611	0.658	0.669	0.687	0.658	0.690	0.673
trec-covid	0.694	0.684	0.668	0.689	0.703	0.728	0.700	0.713
webis-touche2020	0.270	0.255	0.277	0.274	0.260	0.258	0.294	0.290
mean	0.449	0.455	0.469	0.467	0.493	0.473	0.500	0.467

Table 4. Adapter layer Ablation with **adapters** on DistilBERT PLM.

#	Adapters Removed	MS MARCO dev		TREC DL 2019 NDCG@10	TREC DL 2020 NDCG@10	BEIR NDCG@10	R-Flops	Training Params	Training Time(Hrs)
		MRR@10	R@1000						
a	None	0.351 ^{cdefg}	0.968 ^{defg}	0.711 ^{fg}	0.676 ^g	0.455	1.44	2.23%	34.42
b	0	0.348 ^{defg}	0.967 ^{efg}	0.708 ^{fg}	0.674 ^g	0.458	1.27	2.01%	32.23
c	0-1	0.344 ^{efg}	0.968 ^{efg}	0.709 ^{fg}	0.699 ^{abdefg}	0.459	1.34	1.80%	28.55
d	0-2	0.341 ^{efg}	0.966 ^{efg}	0.703 ^{fg}	0.665 ^g	0.459	1.36	1.59%	26.70
e	0-3	0.325 ^{fg}	0.962 ^{fg}	0.689	0.660 ^g	0.455	1.50	1.37%	24.18
f	0-4	0.318 ^g	0.956	0.659	0.663 ^g	0.455	1.27	1.15%	22.51
g	0-5	0.312	0.955	0.660	0.617	0.449	2.78	0.90%	21.35

layers after each pretrained transformer layer. The final experimental setting removes all 6 adapter layers (0 – 5) and fine-tunes only the language model head.

We note that such an experiment (dropping adapter layers from transformer models) has been studied in NLP [28] and was shown to improve both training and inference time while retaining comparable effectiveness. We report the effectiveness of each adapter ablation setting on MS MARCO, TREC DL 2019 and TREC DL 2020 in Table 4. We actually observe gradual performance drop for MS MARCO and TREC DL datasets as the training parameters decrease with the progressive removal of adapter layers as shown in Table 4. The drop is significantly higher (a drop of 0.25 MRR score) when layers are removed from the second half of the model ($\geq 0 - 3$). This phenomenon is consistent with studies in NLP [22,28] that task-specific information is stored in the later layers of the adapters. For the BEIR datasets, this effectiveness drop is not as evident until all adapters but the language model head is removed (configuration 0 – 5). The last configuration also has less sparsity as observed from the R-FLOPS size of 2.78 compared to the other configurations. We also observe that the training time drops proportional to the drop in adapter layers. The training time for

adapter-tune without any drop in adapter layers is 34.42 hours on 4 Tesla V100 GPUS for 150,000 iterations, and it drops to 26.70 hours with only 1% drop in MRR with the first 0 – 2 adapter layers dropped. The lowest training time is 21.35 hours with a drop of 3.2% in MRR for the configuration with all adapters dropped but the language model head.

4.3 RQ3: Out-of-Domain Dataset Adaptation

For the next research question, we want to check how adapters compare to full finetuning when adapting a model trained on MSMARCO on a smaller out-of-domain dataset. We evaluate this question under two scenarios: i) BEIR and ii) TripClick.

BEIR: On the beir benchmark we use 3 datasets (FEVER, FiQA and NFCorpus) that have training, development and test sets and aim for very different domains and tasks (fact checking , financial QA and bio-medical IR). We start from a pre-finetuned SPLADE model called “splade-cocondenser-ensembledistil” made available in [5]. We verify the effectiveness of the models in zero shot and get a first set of hard negatives. These hard negatives are then used to train either via finetuning of all parameters or via the introduction of adapters. The networks are trained for either 10 (FEVER) or 100 epochs (FiQA and NFCorpus), and at the end of each epoch we compute the development set effectiveness. We use the models with the best development set to compute the 1st round test set effectiveness and generate hard negatives that are used for another round of training that we call 2nd round (which repeats the 1st round, starting from the best network of the 1st round and using negatives from the 1st round).

Results are available in Table 5. While finetuning is not always able to improve the results over the zero-shot, mostly due to overfitting on the training/dev sets. For example, on fever fine-tuning first makes all representations as it can easily overfit to the training even without using many words and only on the second round of training started using more dimensions. On the other hand, adapter tuning is able to consistently improve the effectiveness over the zero shot and first rounds (even if it does not always perform the best, as is the case on NFCorpus). Overall, we conclude that adapters are more stable than finetuning when finetuning on these specific domains.

Table 5. Domain adaptation comparison on BEIR Datasets

Dataset	Training	Zero Shot		1st Round		2nd Round	
		NDCG@10	Recall@100	NDCG@10	Recall@100	NDCG@10	Recall@100
Fever	finetuning			0.692	0.866	0.851	0.959
	adapter	0.793	0.954	0.841	0.960	0.881	0.964
FiQA	finetuning			0.371	0.678	0.356	0.694
	adapter	0.348	0.632	0.373	0.675	0.393	0.711
NFCorpus	finetuning			0.384	0.466	0.403	0.484
	adapter	0.348	0.285	0.362	0.435	0.371	0.428

TripClick: Given that in the BEIR benchmark the adapters underperformed finetuning on bio-medical data, we decided to further experiment on a larger bio-medical dataset called TripClick. The TripClick collection [27] contains approximately 1.5 millions MEDLINE documents (title and abstract), and 692,000 queries. The test set is divided into three categories of queries: Head, Torso and Tail (according to their decreasing frequency), which contain 1,175 queries each. For the Head queries, a DCTR click model was employed to create relevance signals, otherwise raw clicks were used. We use the triplets released by [8]. Similarly to the BEIR experiments, we start from the “splade-cocondenser-ensembledistil” SPLADE model and fine-tune or adapt-tune it over 100,000 iterations (batch size equal to 100). As shown in Table 6, adapter-tuning shows very competitive results, on par with finetuning for head categories (frequent queries), and achieving even better results for the less frequent queries (torso and tail).

Table 6. Performance of mono-encoder on out-of-domain Tripclick Dataset

#	Training	HEAD (dctr)		HEAD		Torso		Tail	
		NDCG@10	Recall@100	NDCG@10	Recall@100	NDCG@10	Recall@100	NDCG@10	Recall@100
a	Finetuning	0.218	0.579	0.302	0.523	0.219	0.679	0.238	0.722
b	Adapter	0.219	0.578	0.299	0.526	0.229^a	0.679	0.253^a	0.720

4.4 RQ4: Knowledge Sharing between Rerankers and First stage Rankers

The final research question explores sharing knowledge between rerankers and first-stage rankers. We explore this with transforming first stage rankers into rerankers. First, we tune the pretrained DistilBERT for reranking task as a baseline for both finetuning and adapter-tuning. We then test transforming both sparse (splade-cocondenser) and dense (tct_colbert-v2-msmarco) first stage rankers into rerankers, using either fine-tuning or adapter-tuning. To be clear, the cross-encoder is initialized with the weights of the aforementioned first stage models, but the reranker classification head on the CLS token is randomly initialized. Also note that we rerank the top-1k returned from “splade-cocondenser-ensembledistil” (represented by “first stage” on table).

We compare adapter-tuning with finetuning and display the results in Table 7. We observe that finetuning the baseline model (DistilBERT) is better than adapter-tuning. When using first stage rankers, results are varied. Dense first stage rerankers were able to learn similarly with both adapter and fine-tuning. However, this was not the case for sparse first stage rankers (splade-cocondenser-ensembledistil). We posit that this may come from two different reasons: i) The SPLADE model does not focus on the CLS representations, but on the MLM head representations of all tokens, thus needing more flexibility; ii) The model has been trained multiple times (initial BERT

Table 7. Knowledge Sharing between first stage rankers and rerankers comparison between finetuning and adapter-tuning.

Base Model	#	Training	MS MARCO dev MRR@10	TREC DL 2019 NDCG@10	TREC DL 2020 NDCG@10
First Stage	a	None	0.383 ^e	0.732	0.721
DistilBERT	b	finetune	0.396 ^{ace}	0.764^e	0.736
	c	adapter	0.388 ^e	0.737	0.727
SPLADE++	d	finetune	0.408^{abceg}	0.753	0.743
	e	adapter	0.358	0.723	0.707
TCT Colbert v2	f	finetune	0.404 ^{abce}	0.749	0.731
	g	adapter	0.400 ^{ace}	0.740	0.739

training, then condenser, then cocondenser and finally SPLADE), while not always using the same precision (fp16 or fp32), which under preliminary analysis seems to have made some parts of the model unusable for cross-encoding without full finetuning. Overall, there is slight gain in using the first stage model for the reranker. However, there’s no increase in effectiveness of using adapters, we actually see worse effectiveness on all settings.

5 Conclusion

Retrieval models, based on PLM, require finetuning millions of parameters which makes them memory inefficient and non-scalable for out-of-domain adaptation. This motivates the need for efficient methods to adapt them to information retrieval tasks. In this paper, we examine adapters for sparse retrieval models. We show that with approximately 2% of training parameters, adapters can be successfully employed for SPLADE models with comparable or even better effectiveness on benchmark IR datasets such as MS MARCO and TREC. We further analyze adapter layer ablation and see a further reduction in training parameters to 1.8% retains effectiveness of full finetuning. For domain adaptation, adapters are more stable and outperform finetuning, which is prone to overfitting. On Tripclick dataset, adapters outperform on precision metrics Torso and Tail queries and performs comparably on Head queries. We explore knowledge transfer between first stage rankers and rerankers as a final study. Adapters underperform full finetuning when trying to reuse sparse model to rerankers. Dense first stage rankers perform similarly for adapters and finetuning while sparse first stage rankers is less effective compared to finetuning. We leave this as future work. As memory-efficient adapters are effective for Splade, we leave for future studying larger sparse models and their generalizability. Finally, an interesting scenario could also be to tackle unsupervised domain adaptation with adapters.

References

1. Beck, T., Bohlender, B., Viehmann, C., Hane, V., Adamson, Y., Khuri, J., Brossmann, J., Pfeiffer, J., Gurevych, I.: Adapterhub playground: Simple and flexible few-shot learning with adapters. In: ACL (2022)
2. Ben Zaken, E., Goldberg, Y., Ravfogel, S.: BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). pp. 1–9. Association for Computational Linguistics, Dublin, Ireland (May 2022). <https://doi.org/10.18653/v1/2022.acl-short.1>, <https://aclanthology.org/2022.acl-short.1>
3. Craswell, N., Mitra, B., Yilmaz, E., Campos, D., Voorhees, E.M., Soboroff, I.: Trec deep learning track: reusable test collections in the large data regime. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 2369–2375 (2021)
4. Ding, N., Qin, Y., Yang, G., Wei, F., Yang, Z., Su, Y., Hu, S., Chen, Y., Chan, C.M., Chen, W., Yi, J., Zhao, W., Wang, X., Liu, Z., Zheng, H., Chen, J., Liu, Y., Tang, J., Li, J., Sun, M.: Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. ArXiv [abs/2203.06904](https://arxiv.org/abs/2203.06904) (2022)
5. Formal, T., Lassance, C., Piwowarski, B., Clinchant, S.: From distillation to hard negative sampling: Making sparse neural ir models more effective. In: Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval. p. 2353–2359. SIGIR ’22, Association for Computing Machinery, New York, NY, USA (2022). <https://doi.org/10.1145/3477495.3531857>, <https://doi.org/10.1145/3477495.3531857>
6. Gheini, M., Ren, X., May, J.: Cross-attention is all you need: Adapting pre-trained Transformers for machine translation. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. pp. 1754–1765. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic (Nov 2021). <https://doi.org/10.18653/v1/2021.emnlp-main.132>, <https://aclanthology.org/2021.emnlp-main.132>
7. Han, W., Pang, B., Wu, Y.: Robust transfer learning with pretrained language models through adapters (2021). <https://doi.org/10.48550/ARXIV.2108.02340>, <https://arxiv.org/abs/2108.02340>
8. Hofstätter, S., Althammer, S., Sertkan, M., Hanbury, A.: Establishing strong baselines for tripclick health retrieval (2022)
9. Houlsby, N., Giurghi, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., Gelly, S.: Parameter-efficient transfer learning for nlp. In: Chaudhuri, K., Salakhutdinov, R. (eds.) ICML. Proceedings of Machine Learning Research, vol. 97, pp. 2790–2799. PMLR (2019)
10. Hu, E., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, L., Chen, W.: Lora: Low-rank adaptation of large language models (2021)
11. Jung, E., Choi, J., Rhee, W.: Semi-siamese bi-encoder neural ranking model using lightweight fine-tuning. In: Proceedings of the ACM Web Conference 2022. p. 502–511. WWW ’22, Association for Computing Machinery, New York, NY, USA (2022). <https://doi.org/10.1145/3485447.3511978>, <https://doi.org/10.1145/3485447.3511978>
12. Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., Yih, W.t.: Dense passage retrieval for open-domain question answering. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language

- Processing (EMNLP). pp. 6769–6781. Association for Computational Linguistics, Online (Nov 2020). <https://doi.org/10.18653/v1/2020.emnlp-main.550>, <https://aclanthology.org/2020.emnlp-main.550>
13. Khattab, O., Zaharia, M.: Colbert: Efficient and effective passage search via contextualized late interaction over bert. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. p. 39–48. SIGIR '20, Association for Computing Machinery, New York, NY, USA (2020). <https://doi.org/10.1145/3397271.3401075>, <https://doi.org/10.1145/3397271.3401075>
 14. Lassance, C., Clinchant, S.: An efficiency study for splade models. In: Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 2220–2226 (2022)
 15. Li, X.L., Liang, P.: Prefix-tuning: Optimizing continuous prompts for generation. In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). pp. 4582–4597. Association for Computational Linguistics, Online (Aug 2021). <https://doi.org/10.18653/v1/2021.acl-long.353>, <https://aclanthology.org/2021.acl-long.353>
 16. Lin, Z., Madotto, A., Fung, P.: Exploring versatile generative language model via parameter-efficient transfer learning. In: Findings of the Association for Computational Linguistics: EMNLP 2020. pp. 441–459. Association for Computational Linguistics, Online (Nov 2020). <https://doi.org/10.18653/v1/2020.findings-emnlp.41>, <https://aclanthology.org/2020.findings-emnlp.41>
 17. Litschko, R., Vulic, I., Glavas, G.: Parameter-efficient neural reranking for cross-lingual and multilingual retrieval. CoRR **abs/2204.02292** (2022). <https://doi.org/10.48550/arXiv.2204.02292>, <https://doi.org/10.48550/arXiv.2204.02292>
 18. Lu, W., Jiao, J., Zhang, R.: Twinbert: Distilling knowledge to twin-structured compressed bert models for large-scale retrieval. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management. p. 2645–2652. CIKM '20, Association for Computing Machinery, New York, NY, USA (2020). <https://doi.org/10.1145/3340531.3412747>, <https://doi.org/10.1145/3340531.3412747>
 19. Ma, X., Guo, J., Zhang, R., Fan, Y., Cheng, X.: Scattered or connected? an optimized parameter-efficient tuning approach for information retrieval. CoRR **abs/2208.09847** (2022). <https://doi.org/10.48550/arXiv.2208.09847>, <https://doi.org/10.48550/arXiv.2208.09847>
 20. Mahabadi, R.K., Ruder, S., Dehghani, M., Henderson, J.: Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In: ACL (2021)
 21. Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., Deng, L.: Ms marco: A human generated machine reading comprehension dataset. In: CoCo@ NIPs (2016)
 22. Pal, V., Kanoulas, E., Rijke, M.: Parameter-efficient abstractive question answering over tables or text. In: Proceedings of the Second DialDoc Workshop on Document-grounded Dialogue and Conversational Question Answering. pp. 41–53. Association for Computational Linguistics, Dublin, Ireland (May 2022). <https://doi.org/10.18653/v1/2022.dialdoc-1.5>, <https://aclanthology.org/2022.dialdoc-1.5>
 23. Paria, B., Yeh, C.K., Yen, I.E., Xu, N., Ravikumar, P., Póczos, B.: Minimizing flops to learn efficient sparse representations. In: International Conference on Learning Representations (2019)

24. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* **32** (2019)
25. Pfeiffer, J., Vulic, I., Gurevych, I., Ruder, S.: Mad-x: An adapter-based framework for multi-task cross-lingual transfer. In: *EMNLP* (2020)
26. Rekabsaz, N., Lesota, O., Schedl, M., Brassey, J., Eickhoff, C.: Tripclick: the log files of a large health web search engine. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. pp. 2507–2513 (2021)
27. Rekabsaz, N., Lesota, O., Schedl, M., Brassey, J., Eickhoff, C.: Tripclick: The log files of a large health web search engine. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. pp. 2507–2513 (2021). <https://doi.org/10.1145/3404835.3463242>
28. Rücklé, A., Geigle, G., Glockner, M., Beck, T., Pfeiffer, J., Reimers, N., Gurevych, I.: AdapterDrop: On the efficiency of adapters in transformers. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. pp. 7930–7946. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic (Nov 2021). <https://doi.org/10.18653/v1/2021.emnlp-main.626>, <https://aclanthology.org/2021.emnlp-main.626>
29. Tam, W.L., Liu, X., Ji, K., Xue, L., Zhang, X., Dong, Y., Liu, J., Hu, M., Tang, J.: Parameter-efficient prompt tuning makes generalized and calibrated neural text retrievers (2022). <https://doi.org/10.48550/ARXIV.2207.07087>, <https://arxiv.org/abs/2207.07087>
30. Thakur, N., Reimers, N., Rücklé, A., Srivastava, A., Gurevych, I.: Beir: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)* (2021)
31. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al.: Transformers: State-of-the-art natural language processing. In: *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*. pp. 38–45 (2020)
32. Yang, J.H., Ma, X., Lin, J.: Sparsifying sparse representations for passage retrieval by top- k masking. *arXiv preprint arXiv:2112.09628* (2021)
33. Yates, A., Nogueira, R., Lin, J.: Pretrained transformers for text ranking: Bert and beyond. In: *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. pp. 1154–1156 (2021)