Semi-Siamese Bi-encoder Neural Ranking Model Using Lightweight Fine-Tuning

Euna Jung∗
GSCT
Seoul National University
Seoul, Korea
xlpczv@snu.ac.kr

Jaekeol Choi∗
Seoul National University
& Naver Corp.
Seoul, Korea
jaekeol.choi@snu.ac.kr

Wonjong Rhee
GSCT, GSAI, AII
Seoul National University
Seoul, Korea
wrhee@snu.ac.kr

ABSTRACT
A BERT-based Neural Ranking Model (NRM) can be either a cross-encoder or a bi-encoder. Between the two, bi-encoder is highly efficient because all the documents can be pre-processed before the actual query time. In this work, we show two approaches for improving the performance of BERT-based bi-encoders. The first approach is to replace the full fine-tuning step with a lightweight fine-tuning. We examine lightweight fine-tuning methods that are adapter-based, prompt-based, and hybrid of the two. The second approach is to develop semi-Siamese models where queries and documents are handled with a limited amount of difference. The limited difference is realized by learning two lightweight fine-tuning modules, where the main language model of BERT is kept common for both query and document. We provide extensive experiment results for monoBERT, TwinBERT, and CoBiBERT where three performance metrics are evaluated over Robust04, ClueWeb09b, and MS-MARCO datasets. The results confirm that both lightweight fine-tuning and semi-Siamese are considerably helpful for improving BERT-based bi-encoders. In fact, lightweight fine-tuning is helpful for cross-encoder, too.†

CCS CONCEPTS
• Information systems → Language models;
  • Computing methodologies → Neural networks.

KEYWORDS
Information retrieval; neural ranking model; bi-encoder; lightweight fine-tuning; prefix-tuning; LoRA;

ACM Reference Format:
Euna Jung, Jaekeol Choi, and Wonjong Rhee. 2022. Semi-Siamese Bi-encoder Neural Ranking Model Using Lightweight Fine-Tuning. In Proceedings of the ACM Web Conference 2022 (WWW ’22), April 25–29, 2022, Lyon, France. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3485447.3511978

1 INTRODUCTION
Since the advent of large-scale language models, BERT-based Neural Ranking Models (NRMs) [8, 16, 19] have been developed and shown to achieve state-of-the-art performance. A BERT-based NRM can be classified either as a cross-encoder or as a bi-encoder. Although cross-encoders generally outperform bi-encoders, bi-encoders are superior in terms of computational efficiency because they allow one-time pre-processing of the long documents. Therefore, bi-encoders tend to receive more attention from industrial practitioners. For processing both queries and documents, a bi-encoder uses a common BERT model with a fixed set of weight values. This has been considered to be a mandatory requirement, because the underlying language model is desired to be the same for handling both queries and documents and because heterogeneous models indeed show very poor performance. Therefore, all of the existing bi-encoder models are Siamese models.

Learning a ranking model is a special task because of the involvement of query and document. In particular, query and document can have distinct characteristics. Table 1 summarizes the length information of query and document for three popular Information Retrieval (IR) datasets. It can be immediately noticed that the length difference between query and document is remarkably large. Robust04 and ClueWeb09b contain very short queries while MS-MARCO has relatively longer queries. Even though not shown in the table, many of ClueWeb09b queries consist of only one word per query. On the other hand, a document usually contains more

| Dataset       | Query count     | Document word count |
|---------------|-----------------|---------------------|
| Robust04      | 2.66 (±0.69)    | 912 (±2.114)        |
| ClueWeb09b    | 2.40 (±0.98)    | 2346 (±2.2128)      |
| MS-MARCO      | 6.00 (±2.58)    | 2454 (±4.761)       |

Table 2: Query examples - Robust04 and ClueWeb09b mainly contain short and keyword-based queries, but MS-MARCO mainly contains long and descriptive queries.

| Dataset | Query examples                                      |
|---------|-----------------------------------------------------|
| Robust04| new fuel sources most dangerous vehicles            |
| ClueWeb09b| dinosaurs air travel                               |
| MS-MARCO| Vitamin D deficiency and skin lesions What happens when blood goes through the lungs? |
than 1,000 words for all three datasets. Table 2 shows examples of the queries. It can be seen that Robust04 and ClueWeb09b have keyword-based queries while MS-MARCO has descriptive queries in full or almost full sentence format. Because the documents of the three datasets are usually in full sentence format, it can be concluded that Robust04 and ClueWeb09b have different sentence formats for query and document while MS-MARCO has the same format. In this work, we hypothesize that a high-performance bi-encoder should process query and document with two different networks, because they tend to have distinct characteristics.

Because heterogeneous networks do not perform well\(^2\), we propose Semi-Siamese (SS) bi-encoder neural ranking models that can properly reflect the different characteristics of query and document. Our semi-Siamese networks are based on a common pre-trained BERT model that is not fine-tuned at all. Instead, a mild differentiation between the query network and the document network is implemented through a lightweight fine-tuning method including prompt-tuning\([11]\), prefix-tuning\([12]\), and LoRA\([6]\). The resulting semi-Siamese networks have less than 1% difference in terms of the number of parameters that are different. We also introduce LoRA+ that allows a small additional differentiation and also consider two sequential hybrids of prefix-tuning and LoRA.

While semi-Siamese learning for bi-encoders is our main focus, we also investigate the benefits of lightweight fine-tuning for Siamese cross-encoders and Siamese bi-encoders. With our best knowledge, we are the first to apply lightweight fine-tuning for improving NRMs. Originally, lightweight fine-tuning methods were developed to reduce task-specific parameter memory/storage and computational cost. But, we will show that they can also provide performance enhancement through their regularizing effect of NRMs. Compared to a full fine-tuning, a lightweight fine-tuning of NRM allows only a limited amount of parameters to be modified and we obtain performance improvements by choosing adequate lightweight fine-tuning methods. Our method improves bi-encoders that are practical in the real web search environment.

Our contributions can be summarized as below.

- For cross-encoder, we show that adapter-based lightweight fine-tuning methods (LoRA and LoRA+) can improve the performance by 0.85%-5.29%.
- For bi-encoder, we show that prefix-tuning performs well for Robust04 and ClueWeb09b that have short queries. The improvement can be very large for TwinBERT, and a modest gain of 0.12%-3.90% is achieved for ColBERT.
- For bi-encoder, we show that semi-Siamese learning is effective where the overall gain of 1.46%-16.23% is achieved for ColBERT.

### 2 RELATED WORKS

#### 2.1 BERT-based NRMs

With the advent of large-scale language models such as BERT and GPT, fine-tuning of such a pre-trained language model has become a standard approach for handling document ranking tasks. For neural ranking models, a variety of BERT-based NRMs have been developed. For instance, monoBERT\([19]\) is often considered to be a powerful baseline that takes a pair of query and document as the input to the BERT model. Such cross-encoder models, however, are computationally demanding because the BERT's output representation needs to be calculated for every combination of query and document. If we have \(N_q\) queries to evaluate for \(N_d\) documents, this means BERT representations need to be evaluated \(N_q \cdot N_d\) times. In contrast, bi-encoder models are computationally efficient because their BERT models do not jointly process each pair of query and document. With a bi-encoder, all of the \(N_d\) documents are pre-processed only once, and their BERT representations are pre-stored. For each query, the query's BERT representation is calculated and used together with each of the pre-stored \(N_d\) document representations for the relevance score calculation. Because the documents are much longer than the queries (see Table 1), the pre-processing of documents makes bi-encoders extremely efficient when compared to the cross-encoders. TwinBERT\([16]\) and ColBERT\([8]\) are two of the popular bi-encoder models. TwinBERT aggregates CLS vectors of the query and the document for the relevance score estimation. ColBERT utilizes the interaction between the BERT representations instead.

#### 2.2 Lightweight Fine-Tuning (LFT)

Traditionally, fine-tuning refers to updating all of the pre-trained weights of a neural network. Because all the weights are updated, we refer to it as full Fine-Tuning (FT) in this work. A full FT of BERT is not always necessary for achieving a high performance. Instead, partial fine-tuning methods can be adopted for reducing task-specific parameter storage and computational cost. Lee et al.\([10]\) investigated a partial fine-tuning strategy where only the final layers are fine-tuned and showed that only a fourth of the final layers need to be fine-tuned to achieve 90% of the original downstream task quality. Radiya-Dixit and Wang\([20]\) showed that it sufficed to fine-tune only the most critical layers. Similar results can be found in\([15, 22]\). While a partial FT can be advantageous over a full FT, they both require the pre-trained BERT to be modified. This is not desirable especially when multiple downstream tasks need to be handled. Recently, another type of fine-tuning called Lightweight Fine-Tuning (LFT) has emerged. With an LFT, all of the BERT parameters are kept frozen. To implement the effect of fine-tuning, LFT instead augments the BERT model with small trainable elements. Two types of LFT have been shown to be extremely useful. In our work, the first type is addressed as adapter-based LFT and it augments a language model like BERT with small weight modules. The second type is addressed as prompt-based LFT and it augments a language model's representation vectors with a small number of input embedding vectors or mid-layer activation vectors. With our best knowledge, LFT has never been applied to neural ranking models before and we are the first to demonstrate its effectiveness for NRMs.

#### 2.2.1 Adapter-based LFT

Houlsby et al.\([5]\) proposed an alternative of full FT where task-specific adapters were inserted between the layers of the pre-trained language model. The adapter’s effectiveness was demonstrated over 26 diverse text classification tasks, where near full fine-tuning performance was achieved with only

\(^2\)We confirmed this by fine-tuning query and document models independently. For example, heterogeneous full fine-tuning of ColBERT on Robust04 resulted in 0.3233 (P@20) while Siamese full fine-tuning resulted in 0.3355 (P@20) where the p-value was 0.014.
3.6% of additional parameters per task. Later, He et al. [4] showed that the adapter FT could mitigate forgetting issues of full FT because of the smaller weight deviations. LoRA [6] is another example of adapter-based LFT where augmentation is implemented with a different design of small weight modules. Instead of inserting layers, LoRA expands the query and value projection weight matrices with linear low-rank residual matrices. Because of its simplicity and small size, we investigate only LoRA among the adapter-based LFTs.

2.2.2 Prompt-based LFT. GPT-3 [1] is an enormously large model, and it is known for its capability of performing few-shot or zero-shot learning merely with text prompts. A survey of prompt-based learning can be found in [14]. Following the idea of prompt-based learning, Lester et al. [11] introduced prompt-tuning where soft prompts were learned through back-propagation. Soft prompts are fine-tuned embeddings, and they do not correspond to discrete text embeddings. Li and Liang [12] extended the concept beyond the input layer to introduce prefix-tuning, where it was different from prompt-tuning in that it learned prefix activation vectors for all the layers including the input layer. In our work, we investigate both prompt-tuning and prefix-tuning for enhancing neural ranking models.

Adapter-based LFTs are fundamentally different from prompt-based LFTs. Adapter-based LFTs do not augment the representations at all, and prompt-based LFTs do not augment weights at all. This difference has a strong implication to NRMs as we will show in the experiment section. Despite the difference, both are fine-tuning methods because their augmentations are task-specific, i.e. not dependent on input examples, where the trainable augmentation elements are fine-tuned for a specific downstream task.

2.3 Semi-Siamese (SS) Models

All of the existing bi-encoder NRMs are Siamese models where a single version of fine-tuned BERT is used for processing both of the queries and the documents. Learning a heterogeneous bi-encoder model is also possible where two different versions of fine-tuned BERT models can be learned, one for processing queries and the other for processing documents. Such a heterogeneous bi-encoder, however, is known to suffer from a large performance degradation due to the deviation of the two language models for handling queries and documents. But when queries are short or when queries have significantly different characteristics compared to the documents, it makes sense to allow a certain degree of deviation to handle the queries better. A possible solution for learning is to introduce Semi-Siamese (SS) models. Semi-Siamese models have never been used for information retrieval tasks, but they have been already adopted in image, video, and recommendation domains. Du et al. [3] used a semi-Siamese model to prevent over-fitting in a face recognition task with a small number of data examples. The model has two structurally identical networks that are trained simultaneously using different inputs. Zhang and Duan [24] proposed a semi-Siamese CNN network for vocal imitation search. To encode vocal imitation and real sound, two CNN networks are needed and they share lower layers. Li et al. [13] used a semi-Siamese model to make directional recommendations. First, two identical networks are trained with undirected data, and then the two networks are trained differently with directed data which makes them semi-Siamese. In the previous works, a semi-Siamese network is utilized when there is a need to train two slightly different networks. In our case of bi-encoder NRMs, we focus on the different characteristics between query and document and design semi-Siamese models to enhance the performance.

3 METHODOLOGY

3.1 Document Re-ranking

Document re-ranking is to rank a set of pre-selected documents for a given query according to the relevance score estimates, where a relevance score is estimated for each pair of query and document. Let $Q$ be a query consisting of tokens $q_1, q_2, ..., q_{|Q|}$ and let $D$ be a document consisting of tokens $d_1, d_2, ..., d_{|D|}$. A positive pair of a query and a document $X_{pos}$ is comprised of relevant $Q$ and $D$ whereas a negative pair $X_{neg}$ is comprised of irrelevant $Q$ and $D$. BERT-based NRMs are composed of two parts, BERT and the ranker. BERT processes query and document pairs $X$ to output contextualized representation vectors $Z$. The ranker is a function $f: Z \rightarrow s \in \mathbb{R}$ that estimates the relevance score $s$ using the BERT output. $s_{pos}$ and $s_{neg}$ indicate the relevance scores of positive pair and negative pair, respectively. NRMs are trained by minimizing the hinge loss of triplet data:

$$J(W) = \mathbb{E}(1 - \frac{e^{s_{pos}}}{e^{s_{pos}} + e^{s_{neg}}})$$

To use the knowledge learned from pre-training tasks, we initialize BERT with the pre-trained weights and randomly initialize the ranker. During the training, we perform gradient updates on the trainable parameters for reducing the loss above. When performing a full fine-tuning, we update the entire set of BERT and ranker weights at the same time.

3.2 Lightweight Fine-Tuning (LFT)

Unlike the full fine-tuning that trains all of the BERT weights for a downstream task, a lightweight fine-tuning trains only the augmented elements. While training only a small portion of parameters (1% or less of BERT weights), some of the LFT methods have been proven to perform well, especially for NLG (Natural Language Generation) tasks. In this section, we address prefix-tuning and LoRA and explain how we apply these LFT methods to BERT-based NRMs.

3.2.1 Prefix-tuning. Prefix-tuning [12] trains the network to generate prefix activation vectors that are prepended to the normal activation vectors of the transformers. Different from prompt-tuning that prepends prompts in front of the input word embeddings, prefix-tuning inserts prefixes to all layers including the input layer. Where to prepend the prefix is a design choice that can affect the performance. Li and Liang [12] prepended the prefix to the key and value representations, but we chose to prepend the prefix to the representations right before the self-attention projection because we experimented with both options and found that our modified approach provided a better performance. As shown in Figure 1(a), prefix embedding vectors $P_{θ}$ that are generated from the source $P_{θ}'$ are prepended to the normal activations of the layers. The activation of an $i^{th}$ token $h_i$ in each layer becomes $P_{θ}[i, :]$ if $i \in P_{idx}$ where $P_{idx}$ denotes the sequence of prefix indices and $P_{θ}[i, :]$ is
computed as the MLP output of the source vectors $P_\theta^i[i,:]$ in low dimension (for simplicity, we omit layer indexes):

$$h_i = P_\theta^i[i,:] = MLP_\theta(P_\theta^i[i,:]) \text{ if } i \in P_{idx}$$

(2)

We train both of the source $P_\theta^i$ and the parameters of $MLP_\theta$ during the prefix-tuning. We set the length of the prefix as 10 and the source dimension as 768. $MLP_\theta$ consists of two linear layers between which a ReLU layer exists. The first linear layer down-projects the source vector corresponding to each index $P_\theta^i[i,:]$ into the space of 256 dimensions and the second linear layer up-projects the vectors back into the space of 768 dimensions. The inference overhead induced by the prefix is less than 0.5%.

When applying prefix-tuning on Siamese NRMs, we prepend the same prefixes to both models of query and document:

$$h_{q,i} = h_{d,i} = P_\theta^i[i,:].$$

(3)

For semi-Siamese NRMs, $h_{q,i}$ and $h_{d,i}$ are not constrained to be the same.

3.2.2 LoRA. LoRA stands for Low-Rank Adaptation [6], and it is an LFT method that freezes the pre-trained weights $W_0$ and trains only the rank decomposition matrix part of $\Delta W = BA$. As shown in Figure 1(b), the representation $h$ is computed as $h = (W_0 + \Delta W)x = (W_0 + BA)x$ where $x$ is the previous layer’s representation vector. $W_0 \in \mathbb{R}^{d \times k}, B \in \mathbb{R}^{d \times r},$ and $A \in \mathbb{R}^{r \times k}$ are the weight matrices, and $r \ll \min(d,k)$. Because $r$ is much smaller than $d$ and $k$, the number of learnable parameters is significantly reduced from $d \times k$ to $r(d+k)$ for each projection process. Because LoRA is applied only to query and value matrices, the number of learnable parameters is decreased even further. Besides, we can sum up the additional LoRA weights to the original weights upon the completion of the training, and therefore the inference overhead can be forced to be zero.

3.2.3 LoRA+. NRM is a complicated task, and it can be desirable to increase the number of trainable parameters. After investigating several options, we have designed LoRA+ that is the same as LoRA except for additionally applying rank decomposition matrix to the dense layer $W_d$. Dense layer here refers to the layer following the self-attention layer in transformers. LoRA+ allows the model an additional room for fine-tuning.

3.2.4 Sequential hybrid. We additionally propose a new LFT method that combines prefix-tuning and LoRA. We devised our method based on the hypothesis that prefix-tuning and LoRA can complement each other. Note that they are involved in different parts of BERT. Prefix-tuning inserts task-specific information to the model by prepending activation vectors. On the other hand, LoRA fine-tunes the model by modifying projection weights through residual connections. There are many possibilities for combining the two, but we have chosen to sequentially combine them such that their learning dynamics are not mixed. After fine-tuning with one of the two LFT modules for $m$ epochs, we freeze the module. Then, we train the other LFT module for $n$ epochs. For robust04 and ClueWeb09b, we used $m = 30$ and $n = 10$. For MS-MARCO, we used $m = 10$ and $n = 3$. Depending on which of prefix-tuning and LoRA is first fine-tuned, we end up with two different sequential hybrid LFTs. We present the Prefix-tuning $\rightarrow$ LoRA in Algorithm 1, and LoRA $\rightarrow$ Prefix-tuning is the same except for the order.

### Algorithm 1 Sequential Hybrid: Prefix-tuning $\rightarrow$ LoRA

1. Train the prefix-tuning parameters for $m$ epochs and save the prefixes for the epoch with the best validation performance.
2. Freeze the prefixes with the saved prefixes.
3. Train the LoRA parameters for $n$ epochs and save the LoRA weights for the epoch with the best validation performance.
4. Freeze the LoRA weights with the saved LoRA weights.

3.3 Semi-Siamese Neural Ranking Model

We propose three types of semi-Siamese LFT for bi-encoder NRMs.

3.3.1 SS prefix-tuning. To allow bi-encoder models to effectively process query-specific and document-specific information, we devised semi-Siamese prefix-tuning. As shown in Figure 2(a), we generate SS prefixes by summing up common prefixes with query-specific or document-specific prefixes:

$$h_{q,i} = P_\theta^i[i,:] + P_{q,i}^x[i,:] \text{ if } i \in P_{idx}$$

(4)

$$h_{d,i} = P_\theta^i[i,:] + P_{q,i}^x[i,:] \text{ if } i \in P_{idx}$$

(5)
We explored a few options for SS prefix-tuning. They are discussed without any tuning and have found that they achieved comparable where $W$ weights for query projection and to use heterogeneous LoRA weights where $P$ value weights. On the other hand, $q$ in the subscript represents the type of input to NRMs.

$P_\theta[i, :]$ means the common prefix, $P_{\theta_q}[i, :]$ is the prefix for query, and $P_{\theta_d}[i, :]$ is the prefix for document. With this method, prefixes of query and document share the information through common prefixes while keeping their own characteristics in specific prefixes. $P'_\theta$, the source input for MLP, is shared for $\theta$, $\theta_q$, and $\theta_d$.

We explored a few options for SS prefix-tuning. They are discussed in Appendix A, and the best performing option is presented here.

### 3.3.2 SS LoRA
We also devised semi-Siamese LoRA as illustrated in Figure 2(b). Because there are two types of LoRA weights, one for query and the other for value, we chose to use common LoRA weights for query projection and to use heterogeneous LoRA weights for value projection. In other words, we train $W_{q}^\theta$, $W_{q}^V$, and $W^q$ where $W_{q}^\theta$ and $W_{q}^V$ are self-attention’s value projection matrix for query and document respectively, and $W^q$ is the query projection matrix for both query and document.

We also explored a few options for SS LoRA. They are discussed in Appendix B, and the best performing option is presented here.

### 3.3.3 SS sequential hybrid
The sequential hybrid can be modified for learning semi-Siamese networks. Because we have two individual LFTs in the sequential hybrid method, we have three options for applying semi-Siamese: apply semi-Siamese to prefix-tuning only, LoRA only, or both. We have lightly investigated all three options without any tuning and have found that they achieved comparable performances. In our experiment section, we show the results for applying SS to LoRA only. Therefore, evaluation results for SS LoRA → Prefix-tuning and Prefix-tuning → SS LoRA are provided.

## 4 EXPERIMENT

### 4.1 Experimental Setup

#### 4.1.1 Datasets and metrics
We conduct our experiments on Robust04 [21], WebTrack 2009 (ClueWeb09b) [2], and MS-MARCO [18] datasets as in [17]. Following Huston and Croft [7], we divide each of Robust04 and ClueWeb09b into five folds and use three folds for training, one for validation, and the remaining one for test. For MS-MARCO, we used the pre-assigned datasets for training, validation, and test. We use document collections from TREC discs 4 and 5 of Robust04 and from ClueWeb09b of WebTrack 2009. We also use MS-MARCO document collections. For evaluation metrics, we used P@20, nDCG@5, and nDCG@20.

#### 4.1.2 Baseline models
We implement three BERT-based NRMs, monoBERT [19], ColBERT [8], and TwinBERT [16]. monoBERT is a cross-encoder model, and the other two are bi-encoder models. We consider the full fine-tuning performance of these models as the baseline performance. We compare LFT and full FT by inspecting the improvements of LFT over full FT.

#### 4.1.3 Training and optimization
When fine-tuning BERT-based models on down-stream tasks, the convention of training only three epochs was shown to be insufficient [23]. We set the maximum epoch as 30 for Robust04 and ClueWeb09b and as 10 for MS-MARCO. We select the checkpoint whose validation score is highest among the checkpoints of all epochs. For all experiments, we used an Adam [9] optimizer. For each method, we used appropriately selected hyper-parameters. For fine-tuning, we set the learning rate ($lr$) for the ranker as 1e-4 and $lr$ for BERT as 2e-5. For prefix-tuning, we used $lr$ of 1e-4 for prefix parameters and ranker weights. For LoRA, we used $lr$ of 1e-4 for both LoRA weights and ranker weights. The detailed setting of hyper-parameters are showed in Table 8 in Appendix C. We repeated each experiment three times with three different random seeds. The results in the tables are calculated by averaging the test scores of all folds of the three experiments. For statistical analysis, we performed one-tailed t-test under the assumption of homoscedasticity. We compared the three performance values of each LFT method with the three performance values of the corresponding full fine-tuning.

#### 4.1.4 Implementation
Our experiments are implemented with Python 3 and PyTorch 1. We use a popular transformer library for

---

1. $q$ in the superscript represents the projection matrix type of self-attention. On the other hand, $q$ in the subscript represents the type of input to NRMs.
2. MS-MARCO, 520K documents, 7.5K triplet data samples, https://trec.nist.gov/data-diskx.html
3. 50M web pages, 4.5K triplet data samples, https://lemurproject.org/clueweb09/
4. 22G documents, 372K triplet data samples, https://microsoft.github.io/msmarco/TREC-Deep-Learning-2019
5. https://github.com/huggingface/transformers
the pre-trained BERT model. We used 10 RTX3090 GPUs each of which has 25.6G memory.

### 4.2 LFT Results for Cross-encoder

Table 3 shows the evaluation results for monoBERT. Here, we compare the performances of full fine-tuning and lightweight fine-tuning. We also show the number of fine-tuning parameters of each fine-tuning method. Prompt-tuning trains the least amount of parameters of 8K, and LoRA+ trains the largest amount of 0.9M. All LFT methods train less than 1M of parameters and thus less than 1% of the BERT’s 110M weight parameters.

4.2.1 Prompt-tuning and Prefix-tuning. Prompt-tuning simply appends the prefix to the input by training only 8K parameters, and the performance is inferior to the baseline of full fine-tuning by about 10%. Lester et al. [11] showed that prompt-tuning could achieve a comparable performance to full fine-tuning on SuperGLUE tasks, but we can observe that training only prompts is not sufficient for NRMIs. As for the prefix-tuning, it degrades the performance of full fine-tuning by about 3% for MS-MARCO but it achieves comparable performance for the other two datasets. From the results, we can observe that prefix-tuning achieves comparable performance to full fine-tuning for Robust04 and ClueWeb09b that have short queries, but not for MS-MARCO.

4.2.2 LoRA and LoRA+. For cross-encoder, LoRA and LoRA+ perform significantly better than full fine-tuning for all three datasets with up to a 5% of improvement. In particular, LoRA+ achieves the best performance for 7 out of 9 evaluation cases. We can explain this result with two reasons. First, because LoRA methods freeze the pre-trained BERT and train only a small amount of the augmented weights, they act as a regularizer with a better generalization. Second, we can infer that the number of trainable parameters is not large enough for LoRA. As we can see in the Table 3, the performance tends to increase as the number of parameters increases. LoRA has more trainable parameters than prefix-tuning and LoRA+ has 1.5 times more trainable parameters than LoRA.

4.2.3 Hybrid LFT. Prefix-tuning and LoRA can be used together because they train parameters in representation dimension and weight dimension, respectively. We combined the two LFT methods by adopting a sequential augmentation of BERT. Overall, hybrid LFT methods show better performance than prefix-tuning, but do not outperform LoRA+.

### 4.3 LFT Results for Bi-encoders

Table 4 shows the evaluation results of LFT for bi-encoders, TwinBERT and ColBERT. The effectiveness of LFT methods exhibits a quite different pattern compared to the case of cross-encoder, and the results and their explanations are provided below.

4.3.1 Prompt-tuning and Prefix-tuning. As in the cross-encoder, prompt-tuning shows degenerate results compared to the full fine-tuning baseline. The only exception is the case of TwinBERT on ClueWeb09b, but perhaps it is due to the poor baseline performance. Therefore, we confirm that prompt-tuning is not adequate for the document ranking tasks and we exclude it from the Semi-Siamese experiments in the next section.

In contrast to the cross-encoder results, prefix-tuning achieves significant improvements over the full fine-tuning for the datasets of Robust04 and ClueWeb09b that have short queries. It also achieves a comparable performance to full fine-tuning for MS-MARCO. We first focus on the results of Robust04 and ClueWeb09b that consist of short and keyword-based queries. Prefix-tuning shows the best performance for most of the cases with improvements over full fine-tuning by large margins of up to 76.65%. Prefix-tuning also outperforms LoRA and LoRA+. We attribute this behavior to the way of bi-encoder’s encoding and the characteristics of the datasets. As shown in Table 1, queries in Robust04 and ClueWeb09b are relatively short and keyword-based. Since bi-encoder models encode query and document separately, the model needs to extract the contextual information from either a query or a document. For query, however, it can be very short and thus BERT can fail to extract any meaningful contextual information. In this case, prefix-tuning has the advantage of adding task-specific and meaningful contextual information into the representation embeddings. For the dataset of MS-MARCO, however, queries are relatively long and the benefit of prefix-tuning becomes smaller. We discuss this result further in Section 5.

4.3.2 LoRA and LoRA+. Unlike in the case of cross-encoder, LoRA and LoRA+ do not dominantly outperform the other methods. LoRA and LoRA+ show a varying performance according to the characteristics of dataset. In the result of the cross-encoder, LoRA+ shows the best performance for Robust04 and ClueWeb09b. For bi-encoder, however, the performance of LoRA and LoRA+ is relatively inferior when compared to prefix-tuning. For MS-MARCO that has long queries, LoRA or LoRA+ can perform better than full fine-tuning and prefix-tuning. Interestingly, LoRA+ performs better than LoRA for cross-encoder, but LoRA performs better than LoRA+ for bi-encoders. LoRA+ uses 50% more weight than LoRA for learning,
indicating that this difference worked positively for cross-encoder and negatively for bi-encoders. Since cross-encoder models do self-attention between query and document, learning is likely to be more complex than in the bi-encoder models that do not use self-attention between query and document. Therefore, it can be assumed that it is advantageous to use more parameters for the cross-encoder models.

4.3.3 Hybrid LFT. Since prefix-tuning shows outstanding performance, and LoRA also performs better than full fine-tuning, the combination of two LFT methods might lead to an additional performance improvement. In Table 4, we included the performance of hybrid LFT methods. The results show the possibility of hybrid methods to improve the performance of single LFT methods.

### 4.4 Semi-Siamese LFT Results for Bi-encoders

LFT methods easily outperform full fine-tuning on document ranking. We further improve LFT methods by applying semi-Siamese networks. SS LFT can handle query and document slightly differently to reflect their distinct characteristics while maintaining the original BERT without any modification to enable consistent encoding over query and document. Table 5 shows that SS LFT methods can improve the best performing LFT or full fine-tuning in most cases.

#### 4.4.1 SS Prefix-tuning

SS prefix-tuning performs significantly better than the other methods when adopted for ColBERT on Robust04 and ClueWeb09b datasets whose queries are short. The gain is very large for ClueWeb09b whose queries are the shortest among the three datasets. Note: *p ≤ 0.05, **p ≤ 0.01 (1-tailed).

#### 4.4.2 SS LoRA

As shown in the table 5, SS LoRA outperforms LoRA in most cases, indicating that semi-Siamese networks allow LoRA for bi-encoder models to better estimate relevance scores. We infer that using different LoRA weights for the query and document representations induces a performance improvement by giving query and document more capacity to focus on query-specific or document-specific information.
Figure 3: For ClueWeb09b, the P@20 performance ratio between ambiguous queries and non-ambiguous queries is shown. Semi-Siamese was not applied. For bi-encoders, prefix-tuning clearly shows relatively better performance for ambiguous queries.

4.4.3 SS Hybrid LFT. We have shown that SS LoRA and SS Prefix-tuning can improve the document ranking performance. We have also shown that prefix-tuning and LoRA can complement each other, increasing the performance when used together. Therefore, we combined prefix-tuning and SS LoRA by sequentially training parameters of each method. From the Table 5, we can see that our hybrid methods perform best in many cases bringing up to 85.63% of improvement over full fine-tuning. SS could have been applied to prefix-tuning as well, but we have found that SS LoRA is sufficient for hybrid LFT.

5 DISCUSSION

5.1 Cross-encoder vs. Bi-encoder

For the cross-encoder results shown in Table 3, clearly LoRA based models outperform prefix-tuning based models. In fact, LoRA+ is the dominant LFT method that achieves the best performance for 7 out of 9 evaluations. For the bi-encoder results shown in Table 4 and Table 5, however, we can observe that prefix-tuning based models clearly outperform LoRA based models at least for Robust04 and ClueWeb09b. In fact, SS prefix-tuning is the best performing model for 6 out of 6 cases for ColBERT in Table 5.

We provide a possible explanation for the phenomenon. First, we can consider the cross-encoder case. For cross-encoder, query and document are used together as a single input to BERT. Because the pair is used together, the direct relationship or contextual information between the query and the document can be evaluated by the language model. In this case, the value of prepending with task-specific prefixes can become insignificant, and probably it suffices to fine-tune the weight-related parameters such as LoRA weights. Second, we can consider the bi-encoder case. For Robust04 and ClueWeb09b, queries are short. Therefore, bi-encoder might not be able to create effective contextual representations for the query part. Note that no document is visible to BERT when processing a query. Then, it can be more important to provide task-specific information by processing the entire training dataset, such that at least task information can be used for creating contextual query representation that is effective. This can be achieved with the prefixes.

As a further analysis, we have analyzed ClueWeb09b dataset. We divided one-word queries of ClueWeb09b, that has the shortest queries, into ambiguous queries and non-ambiguous queries using pywsd library⁸. We only included one-word queries because a high-confidence ambiguity classification is possible for one-word queries. The results are shown in Figure 3. We can see that prefix-tuning is effective for improving the performance of ambiguous queries while full FT and LoRA are not effective.

5.2 Hybrid: Concurrent Learning vs. Sequential Learning

When combining two LFT methods to create a hybrid method, we have chosen to sequentially train the two LFT modules. The other option is to concurrently train the two LFT modules. The reason for choosing the sequential training can be explained with Figure 4. In the example, prefix-tuning performs well and LoRA does not perform well. If we choose concurrent learning, we end up with a performance that is worse than prefix-tuning only. This indicates that concurrent learning can hinder a proper learning of individual LFTs, especially when the performance gap between the two individual methods is large. To avoid this problem, we have chosen to train them in a sequential way. By considering the two possible orders of sequential training, we are likely to preserve the larger gain of the two individual gains. We didn’t necessarily obtain improved performance for all the cases by adopting sequential hybrid learning, but quite often we were able to obtain improvements over the individual approaches in the case of bi-encoders.

6 CONCLUSION

We have shown the effectiveness of adopting lightweight fine-tuning methods such as prefix-tuning and LoRA to replace the full fine-tuning of the existing bi-encoder NRMs. We have also shown how semi-Siamese networks can be used to achieve a significant performance improvement when the queries are very short. Our semi-Siamese architecture is also efficient in terms of storage and memory requirements thanks to the use of lightweight fine-tuning for creating two slightly different networks.

ACKNOWLEDGMENTS

This work was supported by Naver corporation (Development of an Improved Neural Ranking Model.), National Research Foundation

⁸https://pypi.org/project/pywsd/
REFERENCES

[1] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. arXiv preprint arXiv:2005.14165 (2020).

[2] Jamie Callan, Mark Hov, Changkuk Yoo, and Le Zhao. 2009. Clueweb09 data set.

[3] Hang Du, Hailin Shi, Yuchi Liu, Jun Wang, Zhen Lei, Dan Zeng, and Tao Mei. 2020. Semi-siamese training for shallow face learning. In European Conference on Computer Vision. Springer, 36–53.

[4] Ruidan He, Linlin Liu, Hai Ye, Qingyu Tan, Bosheng Ding, Living Cheng, Jia-Wei Low, Lidong Bing, and Luo Si. 2021. On the Effectiveness of Adapter-based Tuning for Pretrained Language Model Adaptation. arXiv preprint arXiv:2106.03164 (2021).

[5] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In International Conference on Machine Learning. PMLR, 2790–2799.

[6] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Lu, Liang Huang, Zongxian Li, Sheng Li, Lantian Xue, and Yonghong Tian. 2019. Semi-Siamese Bi-encoder Neural Ranking Model Using Lightweight Fine-Tuning WWW '22, April 25–29, 2022, Lyon, France.

[7] Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. (2021). arXiv preprint arXiv:2106.10199 (2021).

[8] Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage retrieval with CLIR model. arXiv preprint arXiv:2006.05987 (2020).

[9] Ruidan He, Linlin Liu, Hai Ye, Qingyu Tan, Bosheng Ding, Living Cheng, Jia-Wei Low, Lidong Bing, and Luo Si. 2021. Semi-Siamese training for shallow face learning. In European Conference on Computer Vision. Springer, 36–53.

[10] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. arXiv preprint arXiv:2005.14165 (2020).

[11] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for NLP. arXiv preprint arXiv:2106.09685 (2021).

[12] Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage retrieval with CLIR model. arXiv preprint arXiv:2006.05987 (2020).

A VARIANTS OF SS PREFIX-TUNING

We experimented with three SS Prefix-tuning methods and compared them with Siamese Prefix-tuning. As the first method, we generated prefixes by summing up common prefixes with query-specific or document-specific prefixes as in the Equations 4 and 5. This method is SS Prefix-tuning in Table 5. As the second method, we generated k common prefixes for query and document and concatenate them with query-specific or document-specific prefixes as below equations where \( \theta_q \) and \( \theta_d \) indicate parameters for query and document respectively, \( \theta \) is the parameter common to both query and document, and \( k \) stands for the number of query-specific and document-specific prefixes.

\[
\begin{align*}
h_{q,i} &= \begin{cases} P_{\theta_q}[i:] \text{ if } i \leq k \text{ and } i \in P_{\text{dx}} \\ P_{\theta_d}[i:] \text{ if } i > k \text{ and } i \in P_{\text{dx}} \end{cases} \\
h_{d,i} &= \begin{cases} P_{\theta_d}[i:] \text{ if } i \leq k \text{ and } i \in P_{\text{dx}} \\ P_{\theta_q}[i:] \text{ if } i > k \text{ and } i \in P_{\text{dx}} \end{cases}
\end{align*}
\]

As the third method, we generated prefixes that did not share information between query and document as in Equations 8 and 9. In this case, prefixes for query and document are generated using different weight parameters while sharing the source vectors.

\[
\begin{align*}
h_{q,i} &= P_{\theta_q}[i:] \text{ if } i \in P_{\text{dx}} \\
h_{d,i} &= P_{\theta_d}[i:] \text{ if } i \in P_{\text{dx}}
\end{align*}
\]

In Table 6, we compare the performance of prefix-tuning variants. From this result, we confirm that the semi-Siamese methods perform generally well when the information is appropriately shared between query and document.

B VARIANTS OF SS LORA

Because LoRA trains the LoRA weights for query and value, \( W_q^q \) and \( W_v^q \), we devised three SS LoRA methods. As the first method, we used heterogeneous LoRA query weights for both query and document while using the same LoRA value weights for query and document. In other words, we trained \( W_q^q, W_v^q, W_q^d, \) and \( W_v^d \). This method is written as SS LoRA (hetero query) in Table 7. As the second method, we tried a different method where trained common LoRA query weights for query and document while training heterogeneous LoRA value weights for query and document. Here, we trained \( W_q^q, W_v^q, W_q^d \), and \( W_v^d \). This method is written as SS LoRA (hetero value) in Table 7. As the third method, we trained different LoRA weights for query and document, letting \( W_q^q, W_q^d, W_v^q, \) and \( W_v^d \) be trained. From the results in Table 7, we can observe that SS LoRA variants generally perform well. We infer that sharing information between query and document through LoRA weights is important for a better performance. When comparing SS LoRA variants, the method that shares LoRA query weights performs slightly better. We can infer that learning query-specific or document-specific value weights are more important in estimating relevance rather than using different query weights.

C HYPER-PARAMETER SETTING

We have lightly tuned the hyper-parameters of each method, and the values are shown in Table 8. Additional explanations on LoRA alpha and LoRA dropout can be found in [6].
Table 6: Variants of SS prefix-tuning. We compare performances of prefix-tuning and three SS prefix-tuning methods. Among SS prefix-tuning methods, the method that shares information between query and document by averaging common prefixes is our suggested method and it is used for obtaining the results in Table 5.

| Model       | Prefix sharing (query-document) | Robust04 P@20 | NDCG@5 | NDCG@20 | Clueweb09b P@20 | NDCG@5 | NDCG@20 | MS-MARCO P@20 | NDCG@5 | NDCG@20 |
|-------------|----------------------------------|---------------|--------|---------|-----------------|--------|---------|----------------|--------|---------|
| TwinBERT    | All                              | 0.3117        | 0.3647 | 0.3496  | 0.2437          | 0.2065 | 0.2168  | 0.5132         | 0.5630 | 0.5014  |
|             | Average*                         | 0.3109        | 0.3678 | 0.3492  | 0.2223          | 0.1721 | 0.1942  | 0.5155         | 0.5229 | 0.4918  |
|             | Concatenation                     | 0.3101        | 0.3685 | 0.3505  | 0.2105          | 0.1638 | 0.1836  | 0.5174         | 0.5347 | 0.4970  |
|             | None                             | 0.3085        | 0.3597 | 0.3453  | 0.2036          | 0.1554 | 0.1746  | 0.5015         | 0.4955 | 0.4716  |
| ColBERT     | All                              | 0.3429        | 0.4084 | 0.3865  | 0.2695          | 0.2571 | 0.2544  | 0.5577         | 0.6221 | 0.5556  |
|             | Average*                         | 0.3442        | 0.3472 | 0.3459  | 0.2950          | 0.2914 | 0.2835  | 0.5554         | 0.6125 | 0.5526  |
|             | Concatenation                     | 0.3373        | 0.4071 | 0.3813  | 0.2807          | 0.2755 | 0.2694  | 0.5605         | 0.6221 | 0.5588  |
|             | None                             | 0.3384        | 0.4057 | 0.3823  | 0.2774          | 0.2682 | 0.2663  | 0.5546         | 0.5920 | 0.5392  |

Table 7: Variants of SS LoRA. We compare performances of LoRA and three SS LoRA methods. Among the SS LoRA methods, the method that shares LoRA weights for the value projection with the asteroid mark is our suggested method and used for obtaining the results in Table 5.

| Model       | LoRA weight sharing (query-document) | Robust04 P@20 | NDCG@5 | NDCG@20 | Clueweb09b P@20 | NDCG@5 | NDCG@20 | MS-MARCO P@20 | NDCG@5 | NDCG@20 |
|-------------|--------------------------------------|---------------|--------|---------|-----------------|--------|---------|----------------|--------|---------|
| TwinBERT    | All                                  | 0.3090        | 0.3639 | 0.3484  | 0.1851          | 0.1224 | 0.1530  | 0.5326         | 0.5458 | 0.5154  |
|             | Value (hetero query)                 | 0.3099        | 0.3666 | 0.3496  | 0.1882          | 0.1286 | 0.1543  | 0.5294         | 0.5816 | 0.5241  |
|             | Query (hetero value)*               | 0.3117        | 0.3713 | 0.3511  | 0.1926          | 0.1341 | 0.1603  | 0.5240         | 0.5778 | 0.5129  |
|             | None                                 | 0.3116        | 0.3790 | 0.3535  | 0.1870          | 0.1309 | 0.1553  | 0.5310         | 0.5630 | 0.5127  |
| ColBERT     | All                                  | 0.3386        | 0.4021 | 0.3818  | 0.2644          | 0.2373 | 0.2498  | 0.5601         | 0.6360 | 0.5566  |
|             | Value (hetero query)                 | 0.3398        | 0.4071 | 0.3840  | 0.2611          | 0.2319 | 0.2455  | 0.5655         | 0.6263 | 0.5602  |
|             | Query (hetero value)*               | 0.3406        | 0.4042 | 0.3823  | 0.2615          | 0.2470 | 0.2500  | 0.5647         | 0.6289 | 0.5587  |
|             | None                                 | 0.3404        | 0.3947 | 0.3791  | 0.2595          | 0.2347 | 0.2435  | 0.5566         | 0.5998 | 0.5461  |

Table 8: Hyper-parameter settings of each dataset and fine-tuning methods.

| Method          | Hyper-parameter | MS-MARCO | Robust04 | Clueweb09b |
|-----------------|-----------------|----------|----------|------------|
| Common HP       | Optimizer       | Adam     | 0.0001   |            |
|                 | Ranker lr       |          |          |            |
| Full Fine-tuning| Prefix length   | 10       | 30       | 30         |
|                 | Source dimension| 256      |          |            |
|                 | Prefix lr       | 0.0001   |          |            |
|                 | Common prefix length (SS Prefix-tuning (concat)) | 5 | | |
| LoRA            | LoRA rank       | 16       |          |            |
|                 | LoRA alpha      | 32       |          |            |
|                 | LoRA dropout    | 0.1      |          |            |
|                 | LoRA lr         | 0.0001   |          |            |