Plug-and-Play Knowledge Injection for Pre-trained Language Models

Anonymous ACL submission

Abstract

Injecting external knowledge can improve the performance of pre-trained language models (PLMs) in various downstream NLP tasks. However, current knowledge injection methods usually require knowledge-aware pre-training or fine-tuning, which makes the knowledge-enhanced models strongly coupled to some specific knowledge bases. Toward flexible knowledge injection, we explore a new paradigm, *plug-and-play knowledge injection*, which decouples models from knowledge bases. Correspondingly, we propose a plug-and-play injection method, *map-tuning*, which trains a mapping of knowledge embeddings to enrich model inputs with mapped embeddings while keeping PLMs frozen. Experimental results on two typical knowledge-driven NLP tasks show that map-tuning effectively improves the performance of PLMs with little computational cost. Specifically, one mapping network can be plugged on various downstream tasks without any additional training. And, one downstream model can work with multiple mapping networks of different knowledge bases in order to adapt to different domains. We will release all the code and models of this paper.

1 Introduction

Recent years have witnessed rapid development in enhancing pre-trained language models (PLMs) with various external knowledge, including linguistic knowledge (Levine et al., 2020; Zhou et al., 2020), factual knowledge (Zhang et al., 2019; Peters et al., 2019), and commonsense knowledge (Bosselut et al., 2019; Guan et al., 2020). Knowledge can improve PLMs on a wide range of tasks, including language modeling (Chen et al., 2020), information extraction (Liu et al., 2020a; Wang et al., 2021b), and question answering (Xiong et al., 2020; Wang et al., 2021a).

Existing knowledge injection methods usually introduce knowledge through pre-training, such as KnowBERT (Peters et al., 2019) and LUKE (Yamada et al., 2020), or fine-tuning, such as K-BERT (Liu et al., 2020a) and K-Adapter (Wang et al., 2021a), which makes the knowledge-enhanced models *strongly coupled* to some specific knowledge bases and requires retraining the whole model for a new knowledge base. However, knowledge bases are continuously updated and the numbers of pre-trained and fine-tuned models are growing rapidly, leading to the need for *flexible* knowledge injection methods.

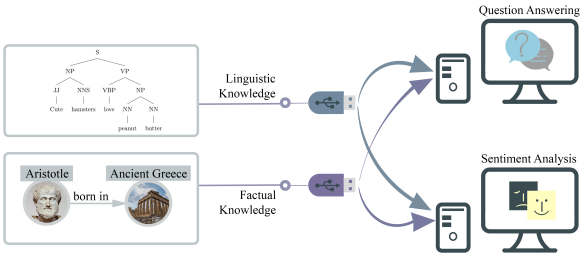

Figure 1: Illustration of plug-and-play knowledge injection, where models are decoupled from knowledge.

Toward flexible injection, we explore a new paradigm, named *plug-and-play knowledge injection*. As shown in Figure 1, we treat knowledge from multiple sources as a series of flash disks and treat existing models as computers. In this paradigm, knowledge and models are decoupled. With model parameters being fixed, one knowledge base can be plugged into multiple models and one model can utilize multiple knowledge bases.

In this work, we target downstream models that have already adapted to downstream tasks. There are two advantages. First, it avoids additional knowledge-aware pre-training or fine-tuning, which reduces corresponding computational costs, especially for large-scale PLMs. Second, there are amounts of open-sourced downstream models on the Internet, such as Hugging Face Models[1], which could be used in this paradigm. These two advantages make plug-and-play knowledge injection

[1] https://huggingface.co/models

meaningful in real-world scenarios.

Toward plug-and-play knowledge injection, we propose a parameter-efficient injection method, named *map-tuning*. Specifically, we freeze original model parameters and train a light-weight mapping network to transform knowledge representations into the representation space of token embeddings. Then, we append mapped knowledge representations to the sequences of token embeddings to construct new input sequences.

There are two ways for map-tuning, *general map-tuning* and *task-specific map-tuning* respectively. For general map-tuning, we train a mapping network based on a PLM with the language modeling objective. Then, we use this mapping network to map knowledge representations for all downstream models trained from this PLM. In this way, one mapping network is plugged into all downstream models, which significantly reduces the computational cost of knowledge-aware training. For task-specific map-tuning, we train a mapping network based on a downstream model with the task objective. Although we need to train a mapping network for each model, the memory cost of map-tuning is much lower than fine-tuning. Besides, one downstream model can utilize multiple mapping networks, enabling the model to adapt to different domains with different knowledge bases.

Experimental results on two kinds of knowledge-driven NLP tasks, relation classification and entity typing, show that map-tuning performs well in both general and task-specific settings. (1) In almost all cases, general map-tuning improves the performance of downstream models with different adaptation methods, including full-model fine-tuning (Devlin et al., 2019), LoRA (Hu et al., 2021), Adapter (Houlsby et al., 2019), and BitFit (Zaken et al., 2021). Moreover, the training time of general map-tuning is extremely faster (100x) than knowledge-aware pre-training. (2) Task-specific map-tuning enables a frozen relation classification model to adapt to both Wikipedia and PubMed domains with different knowledge bases. Besides, in the traditional setting of knowledge injection during fine-tuning, map-tuning is still better than previous methods, including E-BERT (Poerner et al., 2020) and PELT (Ye et al., 2022).

2 Related Work

Paradigms of Knowledge Injection. To enhance PLMs with external knowledge, there are two mainstream paradigms: injection **before** fine-tuning and injection **during** fine-tuning (Yin et al., 2022).

For injection before fine-tuning, researchers usually construct new knowledge-aware objectives, such as entity prediction (Zhang et al., 2019; Xu et al., 2021), entity discrimination (Xiong et al., 2020), entity and relation discrimination (Qin et al., 2021), and link prediction (Wang et al., 2021b). In this way, knowledge will be implicitly stored in model parameters. However, the training cost of these methods is expensive, e.g., LUKE and KEPLER take more than 3,000 GPU hours (Yamada et al., 2020; Wang et al., 2021b; Ye et al., 2022).

For injection during fine-tuning, researchers usually augment original text inputs with external knowledge (Zhou et al., 2019; Lin et al., 2019; Liu et al., 2020b; Cheng et al., 2021). When using unstructured knowledge, such as texts, current methods usually extract informative spans from knowledge bases and augment the original text with these textual spans (Karpukhin et al., 2020; Liu et al., 2020a). When using structured knowledge, such as knowledge graphs, current methods usually apply knowledge representation methods to these graphs to obtain knowledge embeddings (Bordes et al., 2013; Lin et al., 2015), and then fuse them with token embedding using knowledge networks (Sun et al., 2020; Su et al., 2021; Yasunaga et al., 2021).

In this work, we study a new paradigm, *plug-and-play knowledge injection*, where we inject knowledge **after** fine-tuning[2]. Compared to the previous two paradigms, injection after fine-tuning is much more inexpensive and flexible. First, due to the original parameters being fixed, the memory footprint of injection is significantly reduced. Second, we could inject knowledge into downstream models without any training.

Implementation of Knowledge Networks. When explicitly injecting knowledge, we need knowledge networks to handle the heterogeneous representation space of token embeddings and knowledge embeddings. ERNIE (Zhang et al., 2019) and KnowBERT (Peters et al., 2019) add intermediate layers in Transformers to fuse token and knowledge embeddings. However, these methods are not suitable for injection after fine-tuning because they will significantly change the forward process and hence require additional fine-tuning. K-Adapter (Wang et al., 2021a) freezes the orig-

[2]Note that fine-tuning includes full-model fine-tuning and parameter-efficient fine-tuning.

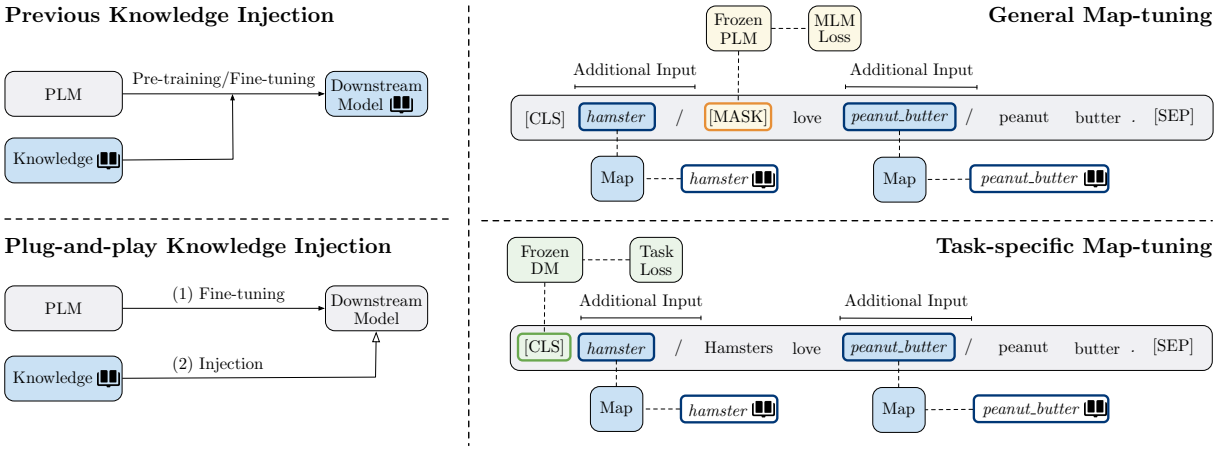

Figure 2: Left: Comparisons between previous paradigms and our proposed plug-and-play paradigm. Right: Two ways for map-tuning. The input text is "Hamsters love peanut butter.". "DM" refers to "downstream model".

inal models and computes knowledge representations based on the outputs of the original models with adapters. To use these knowledge representations, K-Adapter requires fine-tuning, which is also not suitable for injection after fine-tuning. E-BERT (Poerner et al., 2020) and PELT (Ye et al., 2022) directly build an entity embedding lookup table in the representation space of token embeddings and combine entity embedding with token embeddings to construct input embeddings. These methods are suitable for injection after fine-tuning because they can reuse downstream models seamlessly. However, it is unclear whether they work well in this paradigm.

3 Method

In this section, we will first introduce the paradigm of plug-and-play knowledge injection, including general plug-and-play injection and task-specific plug-and-play injection. Then, we will present the overall framework of map-tuning. Finally, we will show how to learn a mapping network for general injection and task-specific injection, respectively.

3.1 Plug-and-Play Knowledge Injection

Given a downstream model $\mathcal{D}$ trained from a PLM $\mathcal{P}$ on a downstream task, we intend to improve its performance on this task by incorporating an extra knowledge base with $\mathcal{D}$'s parameters being fixed.

The knowledge base consists of a set of entities and structured or unstructured knowledge about these entities. To utilize the knowledge base, we first train a knowledge representation model $\mathcal{K}$, which assigns each entity e an entity embedding $\mathbf{e} \in \mathbb{R}^{d_{\text{KE}}}$, where d_{KE} is the dimension of entity embeddings. Note that the knowledge representation model could be a knowledge graph embedding model, e.g., TransE (Bordes et al., 2013), for structured knowledge containing amounts of triples, or a sentence embedding model, e.g., KEPLER (Wang et al., 2021b), for unstructured knowledge containing entity descriptions.

As shown in Figure 2, plug-and-play knowledge injection decouples models from knowledge bases by separately conducting fine-tuning and injection, which is different from previous methods. Concretely, plug-and-play knowledge injection includes general plug-and-play injection and task-specific plug-and-play injection. For general plug-and-play injection, we first train a general knowledge network based on $\mathcal{P}$ and directly plug the general knowledge network into different downstream models, $\mathcal{D}_1, \mathcal{D}_2, \ldots$, without any additional training. For task-specific plug-and-play injection, we train a task-specific knowledge module for each $\mathcal{D}$, and then plug the knowledge network.

3.2 Overall Framework

To fill the gap brought by plug-and-play injection, we map knowledge representations into the space of token embeddings and use the mapped representations as additional inputs, which is also adopted by Poerner et al. (2020); Ye et al. (2022).

Specifically, given an input text, we first match the entity mentions in the text with the entities in the knowledge base. The input text is denoted by $\{w_1, w_2, \ldots, w_n\}$, where w_i is the i-th token and n is the number of tokens in the input text. We use a triple (e, l, r) to represent a mention span, where e is the matched entity, l and r are the left

and right token indices of the mention span. The corresponding mention span is $\{w_l, w_{l+1}, \ldots, w_r\}$. Assume there are m entities in the text, $(e_1, l_1, r_1), (e_2, l_2, r_2), \ldots, (e_m, l_m, r_m)$, where $1 \leq l_1 \leq r_1 < l_2 \leq r_2 < \cdots < l_m \leq r_m \leq n$. The original sequence of input embeddings are $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n\}$, where $\mathbf{w}_i \in \mathbb{R}^{d_{\text{PLM}}}$ is the i-th token embedding and d_{PLM} is the dimension of token embeddings. Then, we map each entity embedding $\mathbf{e}_i$ to $\mathcal{M}(\mathbf{e}_i) \in \mathbb{R}^{d_{\text{PLM}}}$ by a mapping network $\mathcal{M}$. Finally, we replace $\{\mathbf{w}_{l_i}, \mathbf{w}_{l_i+1}, \ldots, \mathbf{w}_{r_i}\}$ with $\{\mathcal{M}(\mathbf{e}_i), /, \mathbf{w}_{l_i}, \ldots, \mathbf{w}_{r_i}\}$ for every (e_i, l_i, r_i) to construct a new input sequence. Note that / is the token embedding of "/".

In the following subsections, we will introduce how to learn the mapping network $\mathcal{M}$ in two different settings, general map-tuning and task-specific map-tuning respectively.

3.3 General Map-tuning

Given a PLM $\mathcal{P}$, general map-tuning aims to learn a mapping network $\mathcal{M}$ based on $\mathcal{P}$ and plug $\mathcal{M}$ into all downstream models trained from $\mathcal{P}$. It requires $\mathcal{M}$ to be general enough to handle different downstream tasks. Hence, it is intuitive to learn the mapping network with language modeling, which has been shown to contain various task abilities (Radford et al., 2019). To reduce the gap between the general mapping network and downstream models, we freeze the parameters of $\mathcal{P}$ and only train the mapping network.

To adapt the mapping network for $\mathcal{P}$, we design a new variant of Masked Language Model (MLM) (Devlin et al., 2019), named Mention-Masked Language Modeling (MMLM). Specifically, instead of randomly masking tokens, we only randomly mask entity mentions in the input text as shown in Figure 2. According to our observation in the experiments, random masking usually leads to insufficient training for the mapping network because most token prediction does not require external knowledge, such as some stop words. On the contrary, predicting entity mentions usually relies on external knowledge, which ensures the mapping network is trained sufficiently. Note that, when masking an entity mention, we will mask all of its tokens. And, the MMLM loss is the same as the original MLM loss (Devlin et al., 2019).

After general map-tuning, $\mathcal{M}$ we get can be used in general plug-and-play injection. Although the mapping network $\mathcal{M}$ did not train with any $\mathcal{D}$ before, we directly plug $\mathcal{M}$ into each $\mathcal{D}$.

3.4 Task-specific Map-tuning

Task-specific map-tuning aims to learn a mapping network $\mathcal{M}$ for a given downstream model $\mathcal{D}$. We freeze the parameters of $\mathcal{D}$ and train the mapping network $\mathcal{M}$ on the downstream task, of which the procedure is shown in Figure 2. The training objective is identical to the original objective of this task. If the knowledge representations provide essential information for this task, the mapping network will learn to extract this kind of information by gradient descent. Note that the mapping network can be trained from scratch, or be initialized with a mapping network from general map-tuning, which could provide a good starting point.

4 Experiments

4.1 Experimental Setups

Training Methods of Downstream Models. We use $\text{BERT}_{\text{base}}$ (Devlin et al., 2019) as the backbone PLM in the experiments. And, we consider four training methods for the adaptation on downstream tasks. (1) **Fine-tuning** optimizes all the parameters of a PLM with the task objective following the original BERT. (2) **LoRA** (Hu et al., 2021) freezes the PLM parameters and injects trainable rank-decomposition matrices as additional parameters. (3) **Adapter** (Houlsby et al., 2019) injects additional adapter networks with the PLM parameters frozen. (4) **BitFit** (Zaken et al., 2021) only optimizes the parameters of bias vectors and freeze the rest parameters. All of the last three methods are parameter-efficient tuning methods. The hyperparameters are reported in the Appendix.

Downstream Tasks. We evaluate our proposed map-tuning on two kinds of knowledge-driven NLP tasks including relation classification and entity typing. For relation classification, which requires models to classify the relation between two entities given a context, we experiment on both few-shot and full-data settings. In the few-shot setting, we aim to evaluate model performance on long-tail relations whose training instances are not sufficient. Specifically, we use FewRel 1.0 (Han et al., 2018) and FewRel 2.0 (Gao et al., 2019). Note that we randomly select 5000 instances of each setting of FewRel 1.0 for fast evaluation, and we experiment with full test data on the official leaderboard in Section 4.4. In the full-data setting, we evaluate models on Wiki80, which contains 80 relation types

from Wikidata, and follow the data split of Zhang et al. (2019). For entity typing, which requires models to classify the type of an entity given a context, we evaluate models on Wiki-ET (Xin et al., 2018) containing 74 entity types from Freebase. All of these datasets provide the annotation of entity linking, which can be used to introduce knowledge through entities. We report accuracy on relation classification and F1 score on entity typing.

Knowledge Bases. We use Wikidata5M (Wang et al., 2021b) and UMLS[3] as our external knowledge bases for the Wikipedia domain and PubMed[4] domain, respectively. To avoid information leakage in relation classification tasks, we remove the triples appearing in the datasets from these knowledge bases. We adopt TransE (Bordes et al., 2013) as our knowledge representation model and the dimension of knowledge embeddings is set to 128.

Baselines. We mainly compare map-tuning with the following baselines: (1) **E-BERT** (Poerner et al., 2020) also learns a mapping network to transform knowledge embeddings into the space of token embeddings. Different from map-tuning, E-BERT builds the relationship between the PLM vocabulary and entities by string matching. Since most entities consist of multiple tokens, the matched entities are a small part. Based on the matching results, E-BERT optimizes the mapping network to make the mapped knowledge embeddings similar to their corresponding token embeddings. In this work, E-BERT and map-tuning use the same TransE embeddings. (2) **PELT** (Ye et al., 2022) aggregates the output representations of a specific entity in multiple contexts to build the entity representation. Then, the entity representation can be appended to the model input without any mapping because the input space and output space are the same for most PLMs. The entity-related context can be treated as an external textual knowledge base. (3) **Retrieval Augmentation (RA)** is to augment input texts with additional retrieved unstructured knowledge, such as RAG (Lewis et al., 2020) and REALM (Guu et al., 2020). In this work, we retrieve the entity descriptions from Wikidata5M and append them to the input texts. (4) **K-Adapter** (Wang et al., 2021a) implicitly stores knowledge in the parameters of adapter networks. For general injection, we follow the original procedure of K-Adapter while keeping the parameters of PLMs and adapters frozen. Note that this procedure still requires the training of the final fully connected layer, which does not strictly meet the setting of general plug-and-play Injection.

Details of Map-tuning. The architecture of the mapping network is simply an affine transformation $\mathbf{We} + \mathbf{b}$, where $\mathbf{W} \in \mathbb{R}^{d_{\text{PLM}} \times d_{\text{KE}}}$ and $\mathbf{b} \in \mathbb{R}^{d_{\text{PLM}}}$. In this work, the parameter amount of the mapping network is $768 \times 128 + 768 < 0.1$M. For Mention-Masked Language Modeling, we sample part of the Wikipedia corpus, where we obtain the annotations of entity linking through hyperlinks. The total size is around 300MB, much smaller than conventional pre-training corpora. Since map-tuning only aims to learn how to adapt the map for a PLM, it does not require much training data. We train the mapping network for 5 epochs, which costs only 12 hours on an NVIDIA Tesla V100. Other hyper-parameters used in the training of mapping networks are reported in the Appendix.

4.2 General Plug-and-Play Injection

In this subsection, we evaluate general map-tuning in the setting of general plug-and-play injection, where we directly combine downstream models and knowledge networks without any training. The results are reported in Table 1.

From this table, we have four observations: (1) All of the four baselines cannot consistently improve the performance of downstream models. In many cases, injecting these knowledge models even degrades the performance. It indicates that **the setting of general plug-and-play injection is challenging** and these four methods are not suitable in this setting. The knowledge provided by these methods cannot be directly used by downstream models in some cases and thus require further training. (2) Our proposed **general map-tuning achieves consistent improvement** on almost all downstream models, suggesting that the mapping network effectively transforms knowledge embeddings into the space of token embeddings and the mapped embeddings can be directly used by downstream models. We highlight the importance of Mention-Masked Language Modeling, which provides sufficient training instances for general map-tuning, while the matched entity-token pairs for E-BERT are insufficient for training the mapping network. (3) Intuitively, parameter-efficient tuning methods, such as Adapter, may work better with the general map-tuning than full-model fine-

[3]UMLS represents the Unified Medical Language System, which is the trademark of U.S. National Library of Medicine.

[4]https://pubmed.ncbi.nlm.nih.gov/

Method	Injection	FewRel 1.0 5-1	FewRel 1.0 5-5	FewRel 1.0 10-1	FewRel 1.0 10-5	Wiki80	Wiki-ET
Fine-tuning	–	91.0	95.1	85.4	90.8	86.1	77.5
	E-BERT	91.0 (+0.0)	95.0 (−0.1)	86.5 (+1.1)	90.5 (−0.3)	85.4 (−0.7)	77.0 (−0.5)
	PELT	90.5 (−0.5)	94.8 (−0.3)	85.3 (−0.1)	89.8 (−1.0)	85.0 (−1.1)	76.8 (−0.7)
	RA	91.5 (+0.5)	95.5 (+0.4)	85.8 (+0.4)	**91.7** (+**0.9**)	85.9 (−0.2)	76.7 (−0.8)
	K-Adapter	88.6 (−2.4)	94.5 (−0.6)	82.3 (−3.1)	89.9 (−0.9)	86.0 (−0.1)	**77.8** (+**0.3**)
	Map-tuning	**92.6** (+**1.6**)	**95.6** (+**0.5**)	**88.1** (+**2.7**)	91.2 (+0.4)	**86.7** (+**0.6**)	76.6 (−0.9)
LoRA	–	90.7	95.1	84.9	91.2	85.3	77.5
	E-BERT	90.7 (+0.0)	95.2 (+0.1)	85.4 (+0.5)	90.4 (−0.8)	83.7 (−1.6)	77.6 (+0.1)
	PELT	89.9 (−0.8)	94.8 (−0.3)	84.6 (−0.3)	89.8 (−1.4)	83.1 (−2.2)	77.5 (+0.0)
	RA	91.3 (+0.6)	95.8 (+0.7)	85.0 (+0.1)	**92.5** (+**1.3**)	83.8 (−1.5)	76.8 (−0.7)
	K-Adapter	90.0 (−0.7)	94.8 (−0.3)	83.4 (−1.5)	89.1 (−2.1)	85.0 (−0.3)	77.3 (−0.2)
	Map-tuning	**92.3** (+**1.6**)	**96.0** (+**0.9**)	**87.4** (+**2.5**)	91.9 (+0.7)	**85.8** (+**0.5**)	**78.3** (+**0.8**)
Adapter	–	91.2	95.2	86.2	91.1	85.7	77.5
	E-BERT	91.3 (+0.1)	95.4 (+0.2)	86.9 (+0.7)	91.6 (+0.5)	84.4 (−1.3)	78.4 (+0.9)
	PELT	91.0 (−0.2)	95.4 (+0.2)	86.3 (+0.1)	91.3 (+0.2)	84.3 (−1.4)	77.9 (+0.4)
	RA	91.7 (+0.5)	95.5 (+0.3)	85.8 (−0.4)	**92.3** (+**1.2**)	85.0 (−0.7)	76.8 (−0.7)
	K-Adapter	89.9 (−1.3)	94.7 (−0.5)	83.6 (−2.6)	90.0 (−1.1)	**85.9** (+**0.2**)	77.7 (+0.2)
	Map-tuning	**92.6** (+**1.4**)	**95.8** (+**0.6**)	**88.2** (+**2.0**)	91.8 (+0.7)	**85.9** (+**0.2**)	**79.2** (+**1.7**)
BitFit	–	89.2	94.8	83.0	90.0	82.7	77.1
	E-BERT	88.7 (−0.5)	94.5 (−0.3)	83.5 (+0.5)	89.6 (−0.4)	81.3 (−1.4)	77.2 (+0.1)
	PELT	88.2 (−1.0)	94.3 (−0.5)	80.9 (−2.1)	88.3 (−1.7)	80.3 (−2.4)	77.6 (+0.5)
	RA	89.5 (+0.3)	95.2 (+0.4)	82.7 (−0.3)	**91.1** (+**1.1**)	81.8 (−0.9)	74.0 (−3.1)
	K-Adapter	86.4 (−2.8)	93.7 (−1.1)	78.8 (−4.2)	87.5 (−2.5)	81.5 (−1.2)	77.2 (+0.1)
	Map-tuning	**90.4** (+**1.2**)	**95.5** (+**0.7**)	**85.2** (+**2.2**)	90.8 (+0.8)	**83.7** (+**1.0**)	**78.0** (+**0.9**)

Table 1: Results of general plug-and-play injection. We adapt $\text{BERT}_{\text{base}}$ to these datasets with four different training methods. There are five different injection methods. E-BERT, PELT, and Map-tuning utilize entity representations. RA utilizes entity descriptions as additional text input. K-Adapter utilizes the knowledge implicitly stored in the adapter network. Note that downstream models and injection models are trained separately. N-K indicates the N-way K-shot configuration. We boldface the best result for each training method.

tuning does because they change fewer parameters from the PLM and general map-tuning is based on the PLM. In fact, the injection performance of full-model fine-tuning is comparable to that of these parameter-efficient tuning methods. It demonstrates that map-tuning is a promising method for all training methods. (4) Map-tuning significantly improves the model performance on the one-shot setting of FewRel 1.0, showing that knowledge is important when the training instances are extremely insufficient.

4.3 Task-specific Plug-and-Play Injection

In this subsection, we evaluate task-specific map-tuning in the setting of task-specific plug-and-play injection, where we train mapping networks based on downstream models with task objectives.

We first evaluate task-specific map-tuning on Wiki80 and Wiki-ET. If we have already conducted general map-tuning on a PLM, we can initialize the network with the general mapping network. Otherwise, we have to train the network from scratch. The results are reported in Table 3.

From the table, we have two observations: (1) Task-specific map-tuning achieves better performance on these two datasets than general map-tuning. It indicates that the mapping network extracts more informative knowledge for the specific task by task-specific training than the general one does. (2) If the general mapping network is available, it is recommended to use it to initialize the mapping network, which further improves the model performance.

Then, we evaluate task-specific map-tuning on a more challenging setting, domain adaptation. In this setting, we conduct task-specific map-tuning based on a downstream model, in order to adapt the model to a new domain. Here, we use the relation classification datasets on Wikipedia domain (FewRel 1.0) and PubMed domain (FewRel 2.0). FewRel 1.0 is the source domain. FewRel 2.0 is the target domain. Since the original FewRel 2.0 does not provide training instances, we rearrange FewRel 2.0 and have the following data split. FewRel 2.0 has 25 relations. We separate 15 relations for training and development and the rest 10 relations are used for testing. The results are reported in Table 2.

Training Data	Map-tuning	Source Domain 5-1	5-5	10-1	10-5	Target Domain 5-1	5-5	10-1	10-5
Target Domain	–	65.4	80.8	56.9	73.8	78.6	88.6	71.4	79.7
Multiple Domains	–	90.3	94.6	84.9	90.4	**84.8**	**92.0**	**79.0**	**86.8**
Source Domain	–	91.0	95.1	85.4	90.8	76.7	88.2	69.1	81.5
	✓	**92.9**	**95.6**	**88.2**	**91.1**	81.2	89.8	72.6	83.3

Table 2: Results of domain adaptation. The source domain is Wikipedia from FewRel 1.0. The target domain is PubMed from FewRel 2.0. The training data of multiple domains consists of both source and target domains.

Method	Wiki80	Wiki-ET
Fine-tuning	86.1	77.5
+ General Map-tuning	86.7	76.6
+ Task-specific Map-tuning		
Train from Scratch	87.2	78.8
Train from the General Map	**87.8**	**78.9**

Table 3: Results of task-specific map-tuning. We train the mapping network from scratch or initialize the mapping network with the general mapping network.

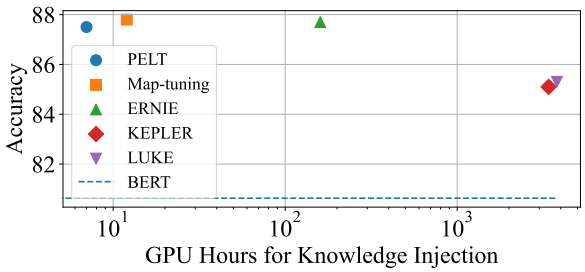

Figure 3: Time cost of different knowledge injection methods on an NVIDIA Tesla V100 GPU.

From this table, we have two observations: (1) For the domain adaptation from Wikipedia domain to PubMed domain, map-tuning significantly improves the model performance (e.g., from 76.7 to 81.2 in 5-1) and achieves better performance than the model fine-tuned on PubMed domain (e.g., from 78.6 to 81.2 in 5-1). It suggests that it is promising to use map-tuning to introduce external knowledge for domain adaptation. (2) Multi-domain training degrades the performance on Wikipedia domain and maintains the performance on PubMed domain while map-tuning does not degrade the performance on each domain. It indicates that map-tuning supports the adaptation of multiple target domains without any performance degradation on the source domains.

4.4 Computational Efficiency

In this subsection, we compare our proposed map-tuning with previous knowledge injection methods, including injection during pre-training and injection during fine-tuning.

Map-tuning	PELT	ERNIE	KEPLER	LUKE
0.1M	123M	114M	123M	274M

Table 4: Number of parameters optimized in knowledge injection. These methods are based on backbone PLMs with around 100 million parameters.

First, we compare the time cost for knowledge injection. We evaluate the training time on an NVIDIA Tesla V100 and compare the model performance on the 10-way 1-shot setting of FewRel 1.0. ERNIE (Zhang et al., 2019), KEPLER (Wang et al., 2021b), and LUKE (Yamada et al., 2020) inject knowledge during pre-training. PELT (Ye et al., 2022) injects knowledge during fine-tuning. The results of ERNIE, KEPLER, LUKE, PELT are taken from Ye et al. (2022). Map-tuning refers to general map-tuning, which is the most challenging setting. The results are shown in Figure 3. From this figure, we observe that the training time of map-tuning is much shorter than those injection-during-pre-training methods and comparable to PELT. Besides, general map-tuning does not require additional training while PELT requires knowledge-aware fine-tuning. Moreover, the performance of map-tuning is also competitive with previous knowledge injection methods.

Second, compared to previous knowledge injection methods, map-tuning is a parameter-efficient method. The numbers of optimized parameters for different knowledge injection methods are shown in Table 4. In order to introduce external knowledge, previous methods usually optimize all parameters during pre-training and fine-tuning while map-tuning only optimizes additional 0.1% of parameters and freezes the original model, which makes it flexible to use mapping networks for different inputs with the same models.

Method	Map-tuning Corpus	FewRel 1.0 5-1	FewRel 1.0 5-5	FewRel 1.0 10-1	FewRel 1.0 10-5	Wiki80	Wiki-ET
Fine-tuning	–	91.0	95.1	85.4	90.8	86.1	77.5
+ E-BERT	–	92.3	95.6	87.6	91.4	87.8	79.0
+ PELT	–	91.2	95.8	86.1	91.6	88.2	79.6
+ General Map	Wikipedia Corpus	**93.7**	**96.2**	**89.6**	**92.4**	88.8	79.9
	Downstream Data	93.2	**96.2**	88.2	92.0	**89.1**	**81.0**

Table 5: Results of knowledge injection during fine-tuning. For general map-tuning, we can use the Wikipedia corpus mentioned in previous section or use the data of downstream tasks.

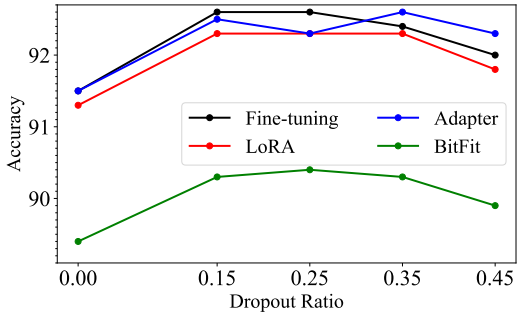

Figure 4: Effect of dropout ratios on the performance of general map-tuning.

5 Analysis

5.1 How Do We Ensure the Generality of Map-tuning?

In the setting of general plug-and-play injection, we train a general mapping network based on a PLM and directly plug it into various downstream models during inference. There exists a gap between the general map-tuning procedure and the inference on downstream tasks, i.e., the PLM used for map-tuning is different from downstream models. To reduce this gap, we use dropout (Hinton et al., 2012) in the attention probabilities and all fully connected layers of the PLM during general map-tuning. Intuitively, dropout simulates different variants of the PLM and makes the mapping network have better generality for different downstream models trained from the PLM. We explore five different dropout ratios. The results on the 5-way 1-shot of FewRel 1.0 are chosen as the representative and shown in Figure 4.

From this figure, we have two observations: (1) Training without dropout leads to the worst performance, which indicates that the generality of the mapping network is not good enough and downstream models cannot utilize the knowledge. (2) Large dropout ratios are also not optimal. Empirically, the dropout ratio of 0.25 is a good choice.

5.2 Is Map-tuning also Competitive in the Traditional Paradigm?

It is natural to use the general mapping network in the traditional injection during fine-tuning paradigm, as the general network essentially builds an entity embedding lookup table. We freeze the parameters of the mapping network and fine-tune the PLM on downstream tasks, during which we augment model inputs with mapped knowledge representations. Intuitively, the models learn to effectively extract information from mapped knowledge representations during fine-tuning. Inspired by ULMFiT (Howard and Ruder, 2018), we also experiment on the setting where we use the task's training data as the corpus for general map-tuning. Our results are shown in Table 5.

From this table, we have two observations: (1) map-tuning consistently outperforms E-BERT and PELT in the traditional paradigm. Considering that E-BERT and map-tuning use the same knowledge embedding, we suggest that map-tuning provides more useful knowledge representations for BERT than E-BERT. (2) General map-tuning on downstream data achieves comparable performance to that on large-scale unsupervised corpus. It indicates that general map-tuning does not necessitate a large amount of training data for a specific task.

6 Conclusion

In this work, we propose the paradigm of plug-and-play knowledge injection toward flexible knowledge injection. Correspondingly, we propose map-tuning to train a mapping of knowledge embeddings. In the experiments, we show that a general mapping network can be plugged into different downstream models without any additional training and one downstream model can work with multiple knowledge bases by different mapping networks. Besides, the computation efficiency of map-tuning is also better than previous methods that require knowledge-aware pre-training and fine-tuning.

References

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Proceedings of NeurIPS*, pages 2787–2795.

Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Celikyilmaz, and Yejin Choi. 2019. COMET: commonsense transformers for automatic knowledge graph construction. In *Proceedings of ACL*, pages 4762–4779.

Wenhu Chen, Yu Su, Xifeng Yan, and William Yang Wang. 2020. KGPT: knowledge-grounded pre-training for data-to-text generation. In *Proceedings of EMNLP*, pages 8635–8648.

Hao Cheng, Yelong Shen, Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2021. Unitedqa: A hybrid approach for open domain question answering. In *Proceedings of ACL*, pages 3080–3090.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186.

Tianyu Gao, Xu Han, Hao Zhu, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2019. Fewrel 2.0: Towards more challenging few-shot relation classification. In *Proceedings of EMNLP*, pages 6249–6254.

Jian Guan, Fei Huang, Minlie Huang, Zhihao Zhao, and Xiaoyan Zhu. 2020. A knowledge-enhanced pretraining model for commonsense story generation. *TACL*, 8:93–108.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. REALM: retrieval-augmented language model pre-training. *arXiv preprint 2002.08909*.

Xu Han, Hao Zhu, Pengfei Yu, Ziyun Wang, Yuan Yao, Zhiyuan Liu, and Maosong Sun. 2018. Fewrel: A large-scale supervised few-shot relation classification dataset with state-of-the-art evaluation. In *Proceedings of EMNLP*, pages 4803–4809.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

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 *Proceedings of ICML*, pages 2790–2799.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of ACL*, pages 328–339.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint 2106.09685*.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of EMNLP*, pages 6769–6781.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of ICLR*.

Yoav Levine, Barak Lenz, Or Dagan, Ori Ram, Dan Padnos, Or Sharir, Shai Shalev-Shwartz, Amnon Shashua, and Yoav Shoham. 2020. Sensebert: Driving some sense into BERT. In *Proceedings of ACL*, pages 4656–4667.

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of NeurIPS*.

Bill Yuchen Lin, Xinyue Chen, Jamin Chen, and Xiang Ren. 2019. Kagnet: Knowledge-aware graph networks for commonsense reasoning. In *Proceedings of EMNLP*, pages 2829–2839.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of AAAI*, pages 2181–2187.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Qi Ju, Haotang Deng, and Ping Wang. 2020a. K-BERT: enabling language representation with knowledge graph. In *Proceedings of AAAI*, pages 2901–2908.

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

Zhenghao Liu, Chenyan Xiong, Maosong Sun, and Zhiyuan Liu. 2020b. Fine-grained fact verification with kernel graph attention network. In *Proceedings of ACL*, pages 7342–7351.

Matthew E. Peters, Mark Neumann, Robert L. Logan IV, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A. Smith. 2019. Knowledge enhanced contextual word representations. In *Proceedings of EMNLP-IJCNLP*, pages 43–54.

Nina Poerner, Ulli Waltinger, and Hinrich Schütze. 2020. E-BERT: Efficient-yet-effective entity embeddings for BERT. In *Findings of EMNLP*, pages 803–818.

Yujia Qin, Yankai Lin, Ryuichi Takanobu, Zhiyuan Liu, Peng Li, Heng Ji, Minlie Huang, Maosong Sun, and Jie Zhou. 2021. ERICA: Improving entity and relation understanding for pre-trained language models via contrastive learning. In *Proceedings of ACL*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*.

Yusheng Su, Xu Han, Zhengyan Zhang, Yankai Lin, Peng Li, Zhiyuan Liu, Jie Zhou, and Maosong Sun. 2021. CokeBERT: Contextual knowledge selection and embedding towards enhanced pre-trained language models. *AI Open*, 2:127–134.

Tianxiang Sun, Yunfan Shao, Xipeng Qiu, Qipeng Guo, Yaru Hu, Xuanjing Huang, and Zheng Zhang. 2020. CoLAKE: Contextualized language and knowledge embedding. In *Proceedings of COLING*, pages 3660–3670.

Ruize Wang, Duyu Tang, Nan Duan, Zhongyu Wei, Xuanjing Huang, Jianshu Ji, Guihong Cao, Daxin Jiang, and Ming Zhou. 2021a. K-adapter: Infusing knowledge into pre-trained models with adapters. In *Findings of ACL/IJCNLP*, pages 1405–1418.

Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. 2021b. KEPLER: A unified model for knowledge embedding and pre-trained language representation. *TACL*, 9:176–194.

Ji Xin, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2018. Improving neural fine-grained entity typing with knowledge attention. In *Proceedings of AAAI*, pages 5997–6004.

Wenhan Xiong, Jingfei Du, William Yang Wang, and Veselin Stoyanov. 2020. Pretrained encyclopedia: Weakly supervised knowledge-pretrained language model. In *Proceedings of ICLR*.

Song Xu, Haoran Li, Peng Yuan, Yujia Wang, Youzheng Wu, Xiaodong He, Ying Liu, and Bowen Zhou. 2021. K-PLUG: knowledge-injected pre-trained language model for natural language understanding and generation in e-commerce. In *Findings of EMNLP*, pages 1–17.

Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. 2020. LUKE: deep contextualized entity representations with entity-aware self-attention. In *Proceedings of EMNLP*, pages 6442–6454.

Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. 2021. QA-GNN: reasoning with language models and knowledge graphs for question answering. In *Proceedings of NAACL-HLT*, pages 535–546.

Deming Ye, Yankai Lin, Peng Li, Maosong Sun, and Zhiyuan Liu. 2022. A simple but effective pluggable entity lookup table for pre-trained language models. In *Proceedings of ACL*.

Da Yin, Li Dong, Hao Cheng, Xiaodong Liu, Kai-Wei Chang, Furu Wei, and Jianfeng Gao. 2022. A survey of knowledge-intensive nlp with pre-trained language models. *arXiv preprint arXiv:2202.08772*.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint 2106.10199*.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. 2019. ERNIE: enhanced language representation with informative entities. In *Proceedings of ACL*, pages 1441–1451.

Jie Zhou, Xu Han, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2019. GEAR: graph-based evidence aggregating and reasoning for fact verification. In *Proceedings of ACL*, pages 892–901.

Junru Zhou, Zhuosheng Zhang, Hai Zhao, and Shuailiang Zhang. 2020. LIMIT-BERT : Linguistics informed multi-task BERT. In *Findings EMNLP*, pages 4450–4461.

Method	Hyper-parameters	FewRel	Wiki80	Wiki-ET
-	Sequence Length	512	128	64
Fine-tuning	Learning Rate	2E-5	5E-5	1E-5
	Batch Size	4	32	64
	Training Step/Epoch	3000	15	2
LoRA	Learning Rate	8E-4	2E-3	1E-3
	Batch Size	4	64	64
	Training Step/Epoch	3000	60	2
	Rank	32	32	4
Adapter	Learning Rate	5E-4	2E-3	1E-3
	Batch Size	4	64	64
	Training Step/Epoch	3000	60	2
	Hidden Size	32	32	32
BitFit	Learning Rate	8E-4	2E-3	1E-3
	Batch Size	4	64	64
	Training Step/Epoch	3000	60	2

Table 6: Hyper-parameters for four training methods. We report the training steps of FewRel and the training epochs of Wiki80 and Wiki-ET.

Method		FewRel	Wiki80	Wiki-ET
Fine-tuning	Dropout	0.25	0.25	0.25
	Epoch	3	5	3
LoRA	Dropout	0.35	0.25	0.35
	Epoch	5	5	4
Adapter	Dropout	0.35	0.35	0.15
	Epoch	5	5	5
BitFit	Dropout	0.25	0.35	0.35
	Epoch	5	4	4

Table 7: Hyper-parameters for general map-tuning.

A Hyper-parameters

A.1 Fine-tuning

We experiment with four training methods for the adaptation of PLMs on downstream tasks, which are Full-model fine-tuning, LoRA, Adapter, and BitFit. We train all the models using AdamW with 10% warming-up steps. We list our hyper-parameters in Table 6.

A.2 General Map-tuning

For general map-tuning, we search the dropout ratio in {0.15, 0.25, 0.35, 0.45}. We train all the mapping networks using Adam (Kingma and Ba, 2015). The learning rate is 3E-5 and the batch size is 64. We train the mapping network on the Wikipedia corpus for 5 epochs. The hyper-parameters of the best mapping network in all cases are listed in Table 7.

A.3 Task-specifc Map-tuning

We report hyper-parameters for task-specific map-tuning in Table 8. We train all mapping networks using Adam with 10% warming-up steps

Hyper-parameters	FewRel	Wiki80	Wiki-ET
Learning Rate	2E-5	4E-4	5E-5
Batch Size	4	64	128
Training Step/Epoch	3000	30	2

Table 8: Hyper-parameters for task-specific map-tuning.

	FewRel	Wiki80	Wiki-ET
Training Epoch	5	2	2

Table 9: Hyper-parameters for map-tuning on the Wikipedia corpus, after which we fine-tune BERT on downstream tasks with the mapping network plugged.

Regarding the results reported in Table 3, during task-specific map-tuning, we use dropout in the attention probabilities and all fully connected layers of the PLM. The dropout ratio is 0.30 and 0.20 for Wiki80 and Wiki-ET, respectively. Regarding the results reported in Table 2, when using training data from source domain for task-specific map-tuning, the dropout ratio is 0.35. In these cases, the training data for task-specific map-tuning are identical to those for fine-tuning the downstream models. We search the dropout ratio in {0.00, 0.15, 0.20, 0.25, 0.30, 0.35}. When using training data from target domain for task-specific map-tuning, we do not use dropout.

The hyper-parameters for experiments with RoBERTa are identical to those with BERT.

A.4 Fine-tuning with the Mapping Network

Regarding the results reported in Table 5, the hyper-parameters for fine-tuning BERT are identical to those in Table 6. We train all mapping networks using Adam without dropout, and the batch size is 64. For map-tuning on the Wikipedia corpus, the learning rate is 1E-5. We report other hyper-parameters for map-tuning on the Wikipedia corpus in Table 9, and those for map-tuning on downstream data in Table 10.

	FewRel	Wiki80	Wiki-ET
Training Epoch	3	12	2
Learning Rate	2E-4	2E-4	1E-5

Table 10: Hyper-parameters for map-tuning on downstream data, after which we fine-tune BERT on downstream tasks with the mapping network plugged.

	FewRel	Wiki80	Wiki-ET
Learning Rate	2E-5	5E-5	5E-5
Bacth Size	4	32	64
Training Step/Epoch	3000	15	2

Table 11: Hyper-parameters for tuning the final fully connected layer, during which we plug K-Adapter into frozen downstream models.

Training Data	Map	5-1	5-5	10-1	10-5
Target Domain	–	81.9	91.0	74.2	84.0
Multiple Domains	–	80.9	92.2	75.4	87.8
Source Domain	–	72.5	89.2	65.2	83.3
	✓	**91.6**	**96.6**	**88.1**	**94.5**

Table 12: Results of domain adaptation using RoBERTa. We report the performance on the target domain.

A.5 The Details of K-Adapter

We use the open-source implementation of K-Adapter[5], and we only consider facAdapter (Factual Adapter). The $\text{BERT}_{\text{base}}$ layers where adapter layers plug in are $\{5, 10\}$. The hyper-parameters for pre-training facAdapter are identical to those reported in Wang et al. (2021a).

In order to plug K-Adapter into frozen downstream models in the setting of general plug-and-play injection, we tune the final fully connected layer on downstream data. We use Adam with 10% warming-up steps, and other hyper-parameters are listed in Table 11.

B Dose Map-tuning Work with Other PLMs?

In this section, we experiment map-tuning with RoBERTa (Liu et al., 2019), another representative PLM, on the domain transfer setting using task-specific map-tuning. The setting is identical to that in Section 4.3. The results are shown in Table 12. From this table, we observe that task-specific map-tuning significantly improves the performance of the model trained on the source domain by introducing the knowledge of the target domain. Moreover, the model plugged with map-tuning is even much better than the model trained on multiple domains. It indicates that map-tuning is a universal knowledge injection method for different PLMs.

C Case Study

We present a qualitative analysis of map-tuning in Table 13. In the first case, the original downstream model does not understand that "flying officer" is a military rank and wrongly predicts the relation as "occupation". With the general mapping network, which enriches the meaning of "flying officer", the model correctly predicts the relation.

The general mapping network, however, may be misleading in some cases. In the second case, it is easy for the original downstream model to recognize "Wasp" as a member of "Avengers" without any external knowledge since this fact could be inferred by the word "other". Compared to the external knowledge provided by the task-specific mapping network, coarse-grained is that provided by the general mapping network, because there is no additional training before the inference. As a result, the model wrongly recognizes "Avengers" as comic books instead of the fictional superhero team, and thus changes the correct model prediction. Task-specific map-tuning, which is further adapted to the task, corrects the prediction.

[5] https://github.com/microsoft/k-adapter

Input	True label	Injection	Predicted label	Logits
Ernest Russell Lyon was a flying officer in 234 Squadron of the Royal Air Force during the Second World War.	military_rank	-	occupation, military_rank, field_of_work	8.0, 4.7, 3.3
		General Map-tuning	military_rank, field_of_work, occupation	6.3, 6.2, 3.9
He later enslaved Thor, then captured the Wasp and the other Avengers.	member_of	-	member_of, parts, characters	8.8, 5.0, 4.4
		General Map-tuning	characters, member_of, parts	6.9, 6.6, 4.7
		Task-specifc Map-tuning	member_of, parts, characters	8.4, 5.7, 4.6

Table 13: A case study for map-tuning on Wiki80. Underlines and wave lines highlight head entities and tail entities respectively. We report the top 3 ranked predictions of different methods.