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

Zhengyan Zhang[1*], **Zhiyuan Zeng**[1*], **Yankai Lin**[2,3], **Huadong Wang**[1], **Deming Ye**[1]
Chaojun Xiao[1], **Xu Han**[1†], **Zhiyuan Liu**[1,4,5†], **Peng Li**[6], **Maosong Sun**[1,4], **Jie Zhou**[7]

[1]NLP Group, DCST, IAI, BNRIST, Tsinghua University, Beijing
[2]Gaoling School of Artificial Intelligence, Renmin University of China, Beijing
[3] Beijing Key Laboratory of Big Data Management and Analysis Methods
[4]International Innovation Center of Tsinghua University, Shanghai [5] Quan Cheng Laboratory
[6] Institute for AI Industry Research (AIR), Tsinghua University, China
[7] Pattern Recognition Center, WeChat AI, Tencent Inc
{zy-z19,zengzy20}@mails.tsinghua.edu.cn {hanxu2022,liuzy}@tsinghua.edu.cn

Abstract

Injecting external knowledge can improve the performance of pre-trained language models (PLMs) on various downstream NLP tasks. However, massive retraining is required to deploy new knowledge injection methods or knowledge bases for downstream tasks. In this work, we are the first to study how to improve the flexibility and efficiency of knowledge injection by reusing existing downstream models. To this end, we explore a new paradigm *plug-and-play knowledge injection*, where knowledge bases are injected into frozen existing downstream models by a *knowledge plugin*. 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 model parameters frozen. Experimental results on three knowledge-driven NLP tasks show that existing injection methods are not suitable for the new paradigm, while map-tuning effectively improves the performance of downstream models. Moreover, we show that a frozen downstream model can be well adapted to different domains with different mapping networks of domain knowledge. Our code and models are available at https://github.com/THUNLP/Knowledge-Plugin.

1 Introduction

Recent years have witnessed rapid development in enhancing pre-trained language models (PLMs) with various external knowledge bases, i.e., knowledge injection for PLMs (Levine et al., 2020; Zhou et al., 2020; Zhang et al., 2019; Peters et al., 2019; Bosselut et al., 2019; Guan et al., 2020). Knowledge injection improves the performance of PLMs

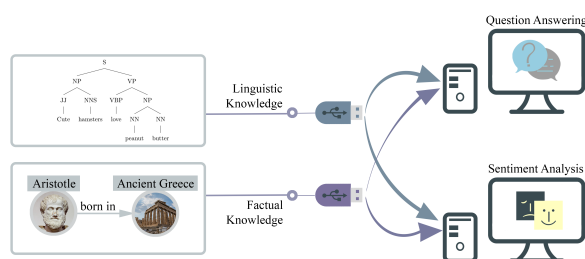* Equal contribution
† Corresponding authors

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

on a wide range of tasks such as information extraction (Liu et al., 2020a; Wang et al., 2021b), question answering (Xiong et al., 2020; Wang et al., 2021a), and text generation (Chen et al., 2020).

Existing injection methods commonly inject knowledge by knowledge-aware pre-training or fine-tuning (Peters et al., 2019; Yamada et al., 2020; Liu et al., 2020a; Wang et al., 2021a). However, rarely studied is how to inject knowledge into a downstream model that is already adapted to a specific task. If we want to apply a new knowledge injection method to enhance models on a specific task, we have to discard task-specific downstream models and retrain them. In addition, one downstream model working with multiple knowledge bases requires retraining itself to inject each knowledge base. Retraining models is time-consuming and resource-intensive, leading to the need for a **flexible and efficient** injection paradigm.

Toward flexible and efficient injection, we explore a novel paradigm *plug-and-play knowledge injection*, where knowledge bases are injected into frozen existing downstream models by knowledge modules. The knowledge module bridges the knowledge base and the model, and we call it a plugin vividly. Under this paradigm, a downstream model would have multiple plugins, each

corresponding to a combination of an injection method and a knowledge base, which ensures flexibility. Moreover, knowledge plugins should be small enough to ensure efficiency. Intuitively, as shown in Figure 1, we treat models and knowledge bases as computers and flash disks, respectively.

In this work, we study two settings for the plug-and-play knowledge injection paradigm. The first is *general plug-and-play knowledge injection*, aiming to inject knowledge into all downstream models (trained from a particular PLM) by a general plugin without any task-specific training. In this setting, all downstream models share exactly one plugin for one combination of an injection method and a knowledge base. The second is *task-specific plug-and-play knowledge injection*, where knowledge plugins are trained to better adapt to downstream tasks while keeping downstream models frozen.

By our pilot study, we find that existing methods (Poerner et al., 2020; Ye et al., 2022; Wang et al., 2021a; Lewis et al., 2020) that can be used directly can not be well applied to the plug-and-play injection paradigm. To this end, we propose *map-tuning*, a preliminary exploration of learning knowledge plugins. Specifically, we train a lightweight mapping network that augments model inputs with mapped knowledge representations, e.g., TransE (Bordes et al., 2013). To meet the general and task-specific injection requirements, we design general map-tuning and task-specific map-tuning, respectively. General map-tuning adopts language modeling as its objective to learn knowledge plugins and seeks better generalizability. Task-specific map-tuning adopts task targets for plugin learning and seeks better task adaptation.

We use three typical knowledge-driven NLP tasks to evaluate our plug-and-play knowledge injection, including relation classification (Han et al., 2018), entity typing (Xin et al., 2018), and question answering (Sciavolino et al., 2021). The experimental results show that: (1) after adapting PLMs to downstream tasks through full-parameter fine-tuning or parameter-efficient tuning, also known as delta tuning (Liu et al., 2021; Ding et al., 2022), injecting knowledge into these downstream models by general map-tuning leads to performance improvements in almost all cases; (2) using task-specific map-tuning to inject domain knowledge further enables a frozen downstream model to work well in different domains. We hope our contribution can draw more attention to the plug-and-play

knowledge injection paradigm and inspire more future research.

2 Plug-and-Play Knowledge Injection

Paradigm Description. Given a downstream model $\mathcal{D}$ trained on a downstream task with a PLM $\mathcal{P}$ as the backbone, we intend to improve its performance on this task by incorporating an extra knowledge base $\mathcal{B}$ and freezing $\mathcal{D}$'s parameters, for which we need to train a knowledge plugin $\mathcal{M}$. Note that neither pre-training nor fine-tuning trains the model $\mathcal{D}$ to cooperate with $\mathcal{B}$ or $\mathcal{M}$.

Two Injection Settings. As shown in Figure 2(a), plug-and-play knowledge injection decouples knowledge injection from model training, which is different from existing paradigms. For general plug-and-play knowledge injection, $\mathcal{M}$ is obtained based on only $\mathcal{P}$ and $\mathcal{B}$, and then it is directly plugged into all downstream models, $\mathcal{D}_1, \mathcal{D}_2, \ldots$, without any additional training. For task-specific plug-and-play knowledge injection, it is allowed to train $\mathcal{M}_1, \mathcal{M}_2, \ldots$ for $\mathcal{D}_1, \mathcal{D}_2, \ldots$ respectively while keeping $\mathcal{D}_1, \mathcal{D}_2, \ldots$ frozen.

Challenges. The general plug-and-play knowledge injection poses serious challenges to methods designed for it. $\mathcal{M}$ is expected to improve the performance of $\mathcal{D}$, yet $\mathcal{D}$ has never seen $\mathcal{M}$ or been seen by $\mathcal{M}$ during training. The only prior condition is that $\mathcal{P}$ and $\mathcal{B}$ are visible during training $\mathcal{M}$. Therefore, the designed methods for general injection need to endow $\mathcal{M}$ with enough generalizability such that $\mathcal{M}$ can adapt to unknown $\mathcal{D}_1, \mathcal{D}_2, \ldots$ well. Even though the knowledge base $\mathcal{B}$ may have rich knowledge, without a good adaptation of $\mathcal{M}$, useful information brought to D will be less than disruptive noise.

The task-specific plug-and-play knowledge injection relaxes the restrictions, where $\mathcal{M}_i$ is allowed to be trained with frozen $\mathcal{D}_i$. Compared to injection during fine-tuning, the training of $\mathcal{M}_i$ should be fast and the parameter number of $\mathcal{M}_i$ should be small compared to that of $\mathcal{D}_i$. Otherwise, the methods would be meaningless. Hence, it requires simple and efficient architecture designs and informative training objectives for $\mathcal{M}_i$.

Potentiality of Using Existing Methods. Few existing knowledge injection methods can be directly used for general plug-and-play knowledge injection. We summarize the existing knowledge
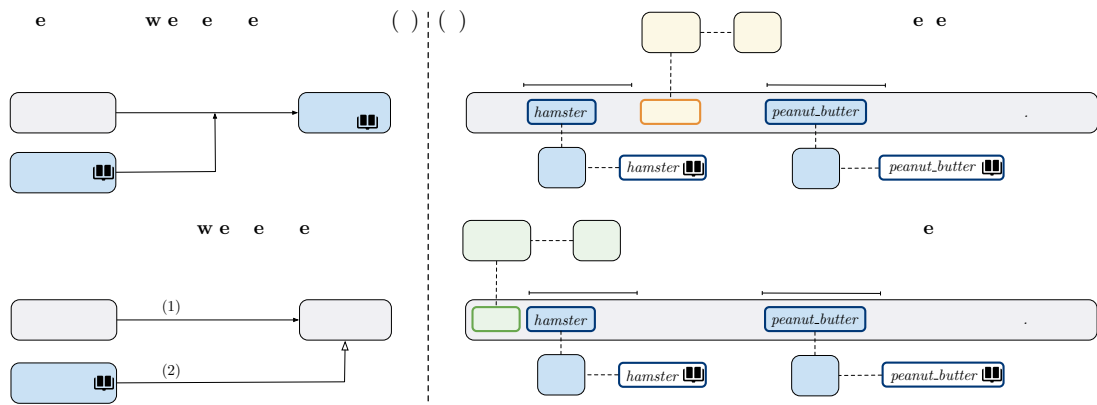
Figure 2: Left: Comparisons between previous paradigms and our proposed plug-and-play paradigm. Right: Two ways for map-tuning. The input text is "Hamsters love peanut butter.". "DM" refers to "downstream model".

injection methods[1] that have the possibility to be used for general plug-and-play knowledge injection as follows. (1) Embedding-based methods: E-BERT (Poerner et al., 2020) and PELT (Ye et al., 2022) build an entity embedding lookup table in the representation space of token embeddings and combine entity embeddings with token embeddings to construct input embeddings. (2) Retrieval-based methods: RAG (Lewis et al., 2020) retrieves plain text from knowledge bases and augments the original input text with the plain text as injected knowledge. (3) Adapter-based methods: K-Adapter (Wang et al., 2021a) computes knowledgeable representations based on the outputs of the downstream models accompanied by knowledgeable adapters, which are trained with frozen PLMs and plugged into all downstream models. Even though these methods may bring knowledge without training PLMs, it is unclear whether they work well in the plug-and-play knowledge injection paradigm, i.e., whether the knowledge brought by them is utilizable for downstream models that have never learned how to use these methods.

3 Map-Tuning

In this section, we first present the overall framework of map-tuning, which is designed for plug-and-play knowledge injection. Then, we show how to use it for general injection and task-specific injection, where the methods are called general map-tuning and task-specific map-tuning respectively.

3.1 Overall Framework

We target knowledge bases consisting of a set of entities and structured or unstructured knowledge

about these entities. To utilize such a knowledge base $\mathcal{B}$, we assume a knowledge representation model $\mathcal{K}$ to assign each entity e an entity embedding $\mathbf{e} \in \mathbb{R}^{d_{\mathrm{KE}}}$, where d_{KE} is the dimension of entity embeddings. Map-tuning injects knowledge by mapping knowledge representations into the space of token embeddings and using 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 $\mathcal{B}$. The input text is denoted by $\{w_1, w_2, \dots, w_n\}$, 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 $\{w_l, w_{l+1}, \dots, w_r\}$. Assume there are m entities in the text, $(e_1, l_1, r_1), (e_2, l_2, r_2), \dots, (e_m, l_m, r_m)$, where $1 \le l_1 \le r_1 < l_2 \le r_2 < \dots < l_m \le r_m \le n$. The original sequence of input embeddings are $\{\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_n\}$, where $\mathbf{w}_i \in \mathbb{R}^{d_{\mathrm{PLM}}}$ 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 $\mathcal{M}(\mathbf{e}_i) \in \mathbb{R}^{d_{\mathrm{PLM}}}$ by a mapping network $\mathcal{M}$. Finally, we replace $\{\mathbf{w}_{l_i}, \mathbf{w}_{l_i+1}, \dots, \mathbf{w}_{r_i}\}$ with $\{\mathcal{M}(\mathbf{e}_i), /, \mathbf{w}_{l_i}, \dots, \mathbf{w}_{r_i}\}$ for every (e_i, l_i, r_i) to construct a new input sequence. Note that $/$ is the token embedding of "/".

3.2 General Map-tuning

General map-tuning aims to train a mapping network $\mathcal{M}$ based on $\mathcal{P}$ and $\mathcal{K}$. It requires $\mathcal{M}$ to have enough generalizability to handle different downstream tasks because $\mathcal{M}$ will be plugged into all downstream models. Hence, we train $\mathcal{M}$ with a

---

[1]These methods are originally designed for injecting knowledge during pre-training or fine-tuning.

general pre-training task while plugging it into $\mathcal{P}$, such as language modeling, which has been shown to be an unsupervised multi-task learning (Radford et al., 2019). We freeze the parameters of $\mathcal{P}$ and only train the mapping network $\mathcal{M}$ to meet the requirement of plug-and-play knowledge injection.

We adopt a variant of Masked Language Model (MLM) (Devlin et al., 2019), named Mention-Masked Language Modeling (MMLM), as the task for training $\mathcal{M}$. According to our observation in the preliminary experiments, the prediction of most tokens requires only language ability instead of external knowledge, such as that of some stop words, while the prediction of entity mentions relies on external knowledge more often. Hence, as shown in Figure 2(b), we randomly mask only entity mentions[2] in the input text to ensure that the mapping network is trained sufficiently and the mapped embeddings are well utilized in the PLM. In this way, the ability of PLMs to predict masked entity mentions is enhanced by the mapped embeddings of both the masked entity and other entities in the context. We mask all tokens of a masked entity mention, and the MMLM loss is the same as the original MLM loss (Devlin et al., 2019).

After general map-tuning, $\mathcal{M}$ can be used for the general plug-and-play injection. Although the mapping network $\mathcal{M}$ was not trained with any downstream model $\mathcal{D}$ before, we can directly plug $\mathcal{M}$ into each $\mathcal{D}$.

3.3 Task-specific Map-tuning

Task-specific map-tuning aims to adapt 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, whose procedure is shown in Figure 2(b). The training objective is identical to the original objective of this task. If the knowledge representations provide useful information for this task, the mapping network will learn to extract this information and to make it recognizable to the downstream model $\mathcal{D}$. Note that the mapping network can not only be trained from scratch, but can also be initialized with a mapping network learned with general map-tuning, which could provide a good starting point.

4 Experiments

4.1 Experimental Setups

Training Methods of Downstream Models. We adopt $BERT_{base}$ (Devlin et al., 2019) as the backbone PLM in the experiments and consider four training methods for its adaptation to downstream tasks. Besides vanilla full-model fine-tuning, we also consider three parameter-efficient tuning (PET) methods, which have been becoming increasingly important in the era of large-scale PLMs (Liu et al., 2021). As resource-saving are both plug-and-play knowledge injection and PET, it is meaningful to apply this paradigm to downstream models trained by PET methods in the resource-limited scenario. (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 most PLM parameters and represents the weight update during model training with a low-rank decomposition. (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 freezes the rest parameters. The hyper-parameters are reported in Appendix A.

Downstream Tasks. We evaluate methods under the plug-and-play knowledge injection paradigm on three kinds of knowledge-driven NLP tasks including relation classification, entity typing, and question answering. 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).[3] In the full-data setting, we evaluate models on Wiki80 (Han et al., 2019), 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 68 entity types from Freebase. For question answering, we evaluate models on EntityQuestions (Sciavolino

---

[2]If the number of entity mentions is small, we can choose to cover all the masking combinations.

[3]We randomly sample 5000 instances from test data of FewRel 1.0 and FewRel 2.0 respectively for fast evaluation. Note that the test data is not publicly released and we get the data from the authors. We experiment with full test data of FewRel 1.0 on the official leaderboard in Section 4.4.

et al., 2021), an open-domain QA dataset consisting of entity-centric questions. We use knowledge-enhanced models to directly answer questions without retrieving related documents. We report accuracy on relation classification and question answering, and F1 score on entity typing.

Knowledge Bases. We use Wikidata5M (Wang et al., 2021b) and UMLS[4] as our external knowledge bases for the Wikipedia domain and PubMed[5] domain, respectively. To avoid information leakage in the relation classification task, 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.

Evaluated Existing Methods. We evaluate existing methods that can be applied to general plug-and-play knowledge injection. (1) **E-BERT** (Poerner et al., 2020) also obtains a mapping network to transform knowledge embeddings. Different from map-tuning, E-BERT builds the connection between the vocabulary and entities by string matching, and then make the mapped knowledge embeddings close to their corresponding token embeddings. In this work, E-BERT uses the same TransE embeddings as map-tuning instead of wikipedia2vec for fair comparisons. (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. We follow the original procedure of K-Adapter while keeping the parameters of PLMs and adapters frozen.[6]

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 $768 \times 128 + 768 < 0.1\text{M}$. For Mention-Masked Language Modeling, we use the raw texts of Wiki20M (Gao et al., 2021), which is sampled from the Wikipedia corpus and provides the annotations of entity linking. The total size is around 300MB, much smaller than common pre-training corpora. Since map-tuning only aims to adapt the mapping network 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. General map-tuning essentially builds an entity embedding lookup table. To evaluate its quality, we evaluate it in the traditional injection during fine-tuning paradigm as a preliminary experiment. To be more specific, we fine-tune the PLMs on downstream tasks, during which the mapping network is plugged into them. The details are in Appendix E. We find that map-tuning consistently outperforms E-BERT and PELT in the traditional paradigm, which also builds entity embedding lookup tables.

4.2 General Plug-and-Play Injection

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

From this table, we have four observations: (1) All of the four existing methods can not consistently improve the performance of downstream models. In most cases, injecting these knowledge modules degrades the model performance, often to a large degree. It empirically proves 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 can not be directly used, so they are basically disruptive noise to the downstream models. (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,

---

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

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

[6]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.

Method	Injection	FewRel 1.0				Wiki80	Wiki-ET	EntityQuestions
		5-1	5-5	10-1	10-5			
Fine-tuning	—	91.0	95.1	85.4	90.8	86.1	77.5	41.7
	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)	42.9 (+1.2)
	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)	46.8 (+5.1)
	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)	**69.5 (+27.8)**
	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)**	39.2 (−2.5)
	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)	49.0 (+7.3)
LoRA	—	90.7	95.1	84.9	91.2	85.3	77.5	42.4
	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)	44.0 (+1.6)
	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)	47.7 (+5.3)
	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)	47.7 (+5.3)
	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)	41.1 (−1.3)
	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)**	**49.6 (+7.2)**
Adapter	—	91.2	95.2	86.2	91.1	85.7	77.5	43.6
	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)	45.1 (+1.5)
	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)	48.4 (+4.8)
	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)	42.9 (−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)	41.5 (−2.1)
	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)**	**50.8 (+7.2)**
BitFit	—	89.2	94.8	83.0	90.0	82.7	77.1	41.3
	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)	42.3 (+1.0)
	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)	46.7 (+5.4)
	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)	33.9 (−7.4)
	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)	40.7 (−0.6)
	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)**	**48.4 (+7.1)**

Table 1: Results of general plug-and-play injection. We adapt BERT$_{base}$ 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.

while the matched entity-token pairs for E-BERT are insufficient for training the mapping network. (3) Intuitively, general map-tuning may work better with PET methods than with full-model fine-tuning because PET methods change much fewer parameters from the PLM and general map-tuning is trained based on the PLM. In fact, the performance improvement brought to models trained by full-model fine-tuning is comparable to that of PET methods. It demonstrates that map-tuning is a promising method regardless of the training methods of downstream models. (4) Remarkably high is the performance improvement brought by RA to fine-tuned BERT on EntityQuestions. We observe that the retrieved entity description contains the exact answer as a substring for 62.19% of instances in the test set, and we remove these instances and report the result in Table 16. We find that RA still gets a slightly higher performance than map-tuning does for fine-tuned BERT, but brings a significant performance drop to other downstream models, while map-tuning brings consistent performance improvement to all downstream models. It suggests that fine-tuned BERT has the surprising generalization ability to extract a substring in the additional context as the answer, and even to reveal the answer hidden in the additional context without string matches. On the contrary, other downstream

Method	Wiki80	Wiki-ET	EntityQuestions
Fine-tuning	86.1	77.5	41.7
+ General Map-tuning	86.7	76.6	49.0
+ Task-specific Map-tuning			
Train from Scratch	87.2	78.8	57.7
Train from the General Map	**87.8**	**78.9**	**58.9**

Table 2: Results of task-specific map-tuning. We train the mapping network from scratch or initialize the mapping network with the general mapping network.

models are not able to reveal the hidden answer. Thus, it is worth investigating RA with pluggable knowledge modules to stably provide information for different downstream models, rather than directly appending unstructured text to model inputs.

4.3 Task-specific Plug-and-Play Injection

Since map-tuning achieves the best performance in the general plug-and-play injection setting, we further evaluate it in the setting of task-specific plug-and-play injection, where we train mapping networks based on downstream models with task objectives. 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.

We first evaluate task-specific map-tuning on Wiki80 and Wiki-ET. The results are reported in Table 2. From the table, we have two observa-

tions: (1) Task-specific map-tuning achieves better performance on these two datasets than general map-tuning does. 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 in domain adaptation, which is a more challenging setting. In this setting, we aim to plug multiple knowledge bases into a single downstream model. Specifically, a downstream model is trained on a source domain, and then we plug the knowledge modules of the target domain into it for domain adaptation. Here, we use the relation classification datasets on the Wikipedia domain (FewRel 1.0) and the PubMed domain (FewRel 2.0). FewRel 1.0 is the source domain. FewRel 2.0 is the target domain. The knowledge base for FewRel 2.0 is UMLS. Since the original FewRel 2.0 does not provide training instances, we rearrange FewRel 2.0 and have the following data split. As FewRel 2.0 has 25 relations, we separate 15 relations for training and development and the rest 10 relations are used for testing.

From Table 3, we have two observations: (1) For the domain adaptation from Wikipedia to PubMed, 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 model performance on the Wikipedia domain and maintains its performance on the PubMed domain while map-tuning does not degrade the performance on each domain. It indicates that the pluggable mapping networks are suitable for continual domain adaptation.

4.4 Computational Efficiency

We compare our proposed plug-and-play knowledge injection paradigm with previous knowledge injection paradigms on the time cost. 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
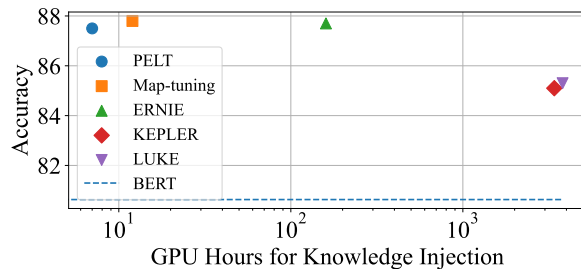

Figure 3: Time cost of different knowledge injection methods on an NVIDIA Tesla V100 GPU.

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, and PELT are taken from Ye et al. (2022). Map-tuning injects knowledge after fine-tuning.

The results are shown in Figure 3. From this figure, we observe that the training time of map-tuning is much shorter than those methods under the paradigm of injecting during pre-training, and it is comparable to PELT. Besides, the performance of map-tuning is also competitive compared to previous knowledge injection methods. Moreover, map-tuning only optimizes additional 0.1% of parameters and we report the number of parameters optimized for different knowledge injection methods in Appendix G. Plug-and-play knowledge injection has great potential to be comparable to previous paradigms w.r.t. task performance, while maintaining its innate flexibility and efficiency.

4.5 Case Study

We present a qualitative analysis of map-tuning in Table 4. 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

Training Data	Map-tuning	Source Domain				Target Domain			
		5-1	5-5	10-1	10-5	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 3: Results of domain adaptation. The source domain is Wikipedia from FewRel 1.0. The target domain is PubMed from FewRel 2.0. We compare task-specific map-tuning with fine-tuning on the target domain and multiple domains consisting of both source and target domains.

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 4: 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.

team, and thus changes the correct model prediction. Task-specific map-tuning, which is further adapted to the task, corrects the prediction.

5 Related Work

To enhance PLMs with external knowledge, there are two mainstream paradigms: injection during pre-training and injection during fine-tuning (Yin et al., 2022). For injection during pre-training, researchers usually construct new knowledge-aware objectives, such as entity prediction (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 the parameters of PLMs. Injection knowledge during pre-training can simultaneously improve performance on a range of downstream knowledge-driven tasks. However, the training cost of this paradigm is expensive. Taking the typical knowledge-enhanced PLMs LUKE (Yamada et al., 2020) and KEPLER (Wang et al., 2021b) as an example, it takes more than 3,000 GPU hours to train them.

Injection knowledge during fine-tuning is a relatively lightweight paradigm, where external knowledge is often used to augment model inputs for specific tasks (Zhou et al., 2019; Lin et al., 2019; Liu et al., 2020b; Cheng et al., 2021; Kang et al.,

2022). When injecting unstructured textual knowledge, some methods retrieve task-related information from external corpora to augment the original input text (Karpukhin et al., 2020; Liu et al., 2020a). When using structured knowledge, such as knowledge graphs, existing methods usually apply knowledge representation learning methods (Bordes et al., 2013; Lin et al., 2015) to encode structured knowledge into embeddings, and then fuse these knowledge embeddings with input token embeddings using knowledge injection methods (Sun et al., 2020; Su et al., 2021; Yasunaga et al., 2021).

In general, existing knowledge injection methods mainly target PLMs and adopt paradigms where knowledge and models are highly coupled. Toward flexible and efficient injection, we study a new paradigm, plug-and-play knowledge injection, where we decouple models and knowledge sources, and then inject knowledge into downstream models without retraining the models. This work is also related to parameter-efficient tuning (Liu et al., 2021; Ding et al., 2022) and plugins for large language models (Xiao et al., 2023; Dathathri et al., 2020; Lauscher et al., 2021; Chronopoulou et al., 2022; Yu et al., 2023; Xu et al., 2023; Alayrac et al., 2022) while we are the first to study knowledge injection in a parameter-efficient and plug-and-play way.

6 Conclusion

In this work, we propose a new paradigm of injection toward flexible and efficient knowledge injection. In this paradigm, downstream models can be enhanced with little computational cost, which benefits large amounts of models. We first systematically evaluate existing knowledge injection methods and find that they are not suitable for plug-and-play injection. Then, we propose map-tuning for this paradigm, which effectively injects knowledge into downstream models to enhance them.

There are four promising directions for future investigation into plug-and-play knowledge injection. (1) How can we reduce the performance gap between methods for this novel paradigm and those for the previous injection paradigms, while maintaining superior flexibility and efficiency? (2) Besides factual knowledge, how can we effectively plug diverse knowledge bases, such as text corpora, voice, images, and even other PLMs? (3) After injecting the knowledge in a plug-and-play way, how can the PLMs do various types of complex reasoning based on the injected knowledge (Onoe et al., 2023)? (4) Can the plug-and-play knowledge injection methods for these sources be unified, so we can plug a combination of multiple sources? We hope this work can attract attention to and inspire research on these problems.

Limitations

In this paper, we present a novel knowledge injection paradigm *plug-and-play knowledge injection* for PLMs. We show existing methods can not be well applied to the new paradigm and propose *map-tuning* as a preliminary exploration of methods.

The paradigm *plug-and-play knowledge injection* has a limitation in terms of its assumption. It assumes that a PLM should be fine-tuned for downstream tasks. However, very large-scale PLMs can perform zero-shot learning or in-context learning on downstream tasks without being fine-tuned. Future work may extend the definition of the proposed paradigm to make it meaningful in these scenes.

The method *map-tuning* has three limitations in terms of its applicability. Firstly, we did not evaluate map-tuning for PLMs pre-trained by other language modeling objectives (e.g., casual language modeling) besides MLM. As its spirit can be easily generalized to various language modeling objectives, we leave this evaluation as future work. Secondly, we did not evaluate whether the PLM can do complex reasoning (e.g., multi-hop reasoning) based on the knowledge injected by map-tuning. Thirdly, map-tuning is designed to plug structural fact knowledge. It is also meaningful to plug other diverse knowledge bases, including text corpora, voice, images, and even other PLMs, which are not covered by our work.

References

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob L. Menick, Sebastian Borgeaud, Andy Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karén Simonyan. 2022. Flamingo: a visual language model for few-shot learning. In *Proceedings of NeurIPS*.

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.

Alexandra Chronopoulou, Matthew E. Peters, and Jesse Dodge. 2022. Efficient hierarchical domain adaptation for pretrained language models. In *Proceedings of NAACL-HLT*, pages 1336–1351.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2020. Plug and play language models: A simple approach to controlled text generation. In *Proceedings of ICLR*.

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.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, Jing Yi, Weilin Zhao, Xiaozhi Wang, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. 2022. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint 2203.06904*.

Tianyu Gao, Xu Han, Yuzhuo Bai, Keyue Qiu, Zhiyu Xie, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2021. Manual evaluation matters: Reviewing test protocols of distantly supervised relation extraction. In *Findings of ACL/IJCNLP 2021*, pages 1306–1318.

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, Tianyu Gao, Yuan Yao, Deming Ye, Zhiyuan Liu, and Maosong Sun. 2019. OpenNRE: An open and extensible toolkit for neural relation extraction. In *Proceedings of EMNLP-IJCNLP*, pages 169–174.

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*.

Minki Kang, Jinheon Baek, and Sung Ju Hwang. 2022. KALA: knowledge-augmented language model adaptation. In *Proceedings of NAACL-HLT*, pages 5144–5167.

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*.

Anne Lauscher, Tobias Lüken, and Goran Glavas. 2021. Sustainable modular debiasing of language models. In *Findings of ACL: EMNLP*, pages 4782–4797.

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.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2021. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586*.

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.

Yasumasa Onoe, Michael J. Q. Zhang, Shankar Padmanabhan, Greg Durrett, and Eunsol Choi. 2023. Can lms learn new entities from descriptions? challenges in propagating injected knowledge. *arXiv preprint arXiv:2305.01651*.

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*.

Christopher Sciavolino, Zexuan Zhong, Jinhyuk Lee, and Danqi Chen. 2021. Simple entity-centric questions challenge dense retrievers. In *Proceedings of EMNLP*, pages 6138–6148.

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.

Chaojun Xiao, Zhengyan Zhang, Xu Han, Chi-Min Chan, Yankai Lin, Zhiyuan Liu, Xiangyang Li, Zhonghua Li, Zhao Cao, and Maosong Sun. 2023. Plug-and-play document modules for pre-trained models. In *Proceedings of ACL*.

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*.

Canwen Xu, Yichong Xu, Shuohang Wang, Yang Liu, Chenguang Zhu, and Julian McAuley. 2023. Small models are valuable plug-ins for large language models. *arXiv preprint 2305.08848*.

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*.

Zichun Yu, Chenyan Xiong, Shi Yu, and Zhiyuan Liu. 2023. Augmentation-adapted retriever improves generalization of language models as a zero-shot plug-in. In *Proceedings of ACL*.

10651

Elad Ben Zaken, Shauli Ravfogel, and Yoav Gold-
berg. 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 rea-
soning for fact verification. In *Proceedings of ACL*,
pages 892–901.

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

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

Table 5: 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	EntityQuestions
Fine-tuning	Dropout	0.25	0.25	0.25	0.35
	Epoch	3	5	3	3
LoRA	Dropout	0.35	0.25	0.35	0.25
	Epoch	5	5	4	5
Adapter	Dropout	0.35	0.35	0.15	0.25
	Epoch	5	5	5	5
BitFit	Dropout	0.25	0.35	0.35	0.35
	Epoch	5	4	4	5

Table 6: Hyper-parameters for general map-tuning.

A   Hyper-parameters

A.1   Fine-tuning downstream PLMs

We experiment with four training methods for the
adaptation of PLMs on downstream tasks, which
are Full-model fine-tuning, LoRA, Adapter, and
BitFit. The embedding layer is frozen during
training. We train all the models using AdamW
with 10% warming-up steps. We list our hyper-
parameters in Table 5.

A.2   General Map-tuning

For general map-tuning, we search the dropout rate
in {0.15, 0.25, 0.35, 0.45}. We train all the map-
ping 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 Ta-
ble 6. When we evaluate RA on these datasets, we
set the sequence length to 512.

A.3   Task-specifc Map-tuning

We report hyper-parameters for task-specific map-
tuning in Table 7. We train all mapping networks
using Adam with 10% warming-up steps

Regarding the results reported in Table 2, during
task-specific map-tuning, we use dropout in the at-
tention probabilities and all fully connected layers

Hyper-parameters	FewRel	Wiki80	Wiki-ET	EntityQuestions
Learning Rate	2E-5	4E-4	5E-5	2E-3
Batch Size	4	64	128	64
Training Step/Epoch	3000	30	2	5

Table 7: Hyper-parameters for task-specific map-tuning.

	FewRel	Wiki80	Wiki-ET	EntityQuestions
Training Epoch	5	2	2	4

Table 8: Hyper-parameters for map-tuning on the Wikipedia corpus, after which we fine-tune BERT on downstream tasks with the mapping network plugged.

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

Table 9: 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	EntityQuestions
Learning Rate	2E-5	5E-5	5E-5	5E-3
Bacth Size	4	32	64	64
Training Step/Epoch	3000	15	2	20

Table 10: Hyper-parameters for tuning the final fully connected layer, during which we plug frozen K-Adapter into frozen downstream models.

of the PLM. The dropout rate is 0.30, 0.20, and 0.00 for Wiki80, Wiki-ET, and EntityQuestions, respectively. Regarding the results reported in Table 3, when using training data from the source domain for task-specific map-tuning, the dropout rate 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 rate in {0.00, 0.15, 0.20, 0.25, 0.30, 0.35}. When using training data from the 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 14, the hyper-parameters for fine-tuning BERT are identical to those in Table 5. 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 8, and those for map-tuning on downstream data in Table 9.

A.5 Details of K-Adapter

We use the open-source implementation of K-Adapter[7], and we only consider facAdapter (Factual Adapter). The BERT$_{base}$ 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 10.

A.6 Details of Data Preprocessing

For FewRel and Wiki80, we mark the subject and object spans by # and $ tokens respectively. For WikiET and EntityQuestions, we mark the entity span by $ token.

To evaluate encoder PLMs on EntityQuestions, we append the [MASK] token to the question, and only keep the instances whose answers are in the PLM token vocabulary. We train the model to fill in the [MASK] token. It is a classification task, where all tokens in the vocabulary are choices. Only when the answer token is ranked as the top 1 result is the model considered to give a correct prediction. We further remove the instances whose entity is not in the database. Finally, we have 37800 training instances, 4693 validation instances, and 4731 test instances.

FewRel, Wiki80, and WikiET provide the annotation of entity linking, and for EntityQuestions we do entity liking by string matching.

B Stability of Map-tuning

We evaluate the stability of map-tuning in general plug-and-play knowledge injection. Training the PLMs on downstream tasks with three different seeds (one of which is used in all main experiments), for each task, we have three different downstream models, into which we plug the mapping network. The mean and standard deviation of performance improvement brought by map-tuning is shown in Table 11. From this table, we observe that map-tuning is not sensitive to downstream models overall, showing its decent stability.

Method	FewRel 1.0				Wiki80	Wiki-ET	EntityQuestions
	5-1	5-5	10-1	10-5			
Fine-tuning	1.300±0.300	0.800±0.436	2.033±0.577	0.533±0.231	0.600±0.200	−0.567±0.306	6.967±0.850
LoRA	1.633±0.153	0.833±0.115	2.800±0.361	0.833±0.115	0.600±0.100	1.000±0.200	7.000±0.173
Adapter	1.367±0.058	0.733±0.115	2.067±0.208	0.833±0.153	0.267±0.306	1.100±0.529	6.967±0.252
BitFit	1.367±0.208	0.500±0.265	2.333±0.153	0.867±0.058	0.700±0.300	0.700±0.173	7.233±0.153

Table 11: The mean and standard deviation of performance improvement brought by map-tuning. We train PLMs on each downstream task with three different seeds.

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.

Map-Tuning	Evaluation Setting	Loss on Test Set
−	No-Plug	7.246
✓	No-Perturbation	5.316
	Self-Perturbation	6.347
	Other-Perturbation	5.501
	All-Perturbation	6.613
✗	No-Perturbation	7.179
	Self-Perturbation	7.237
	Other-Perturbation	7.268
	All-Perturbation	7.355

Table 13: The MMLM loss on the test set in different evaluation settings.

C How Map-tuning Works 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.

D Empirical Analysis of MMLM

We conduct an empirical analysis of what MMLM trains the mapping network to learn. Concretely, we split the general map-tuning corpus into a training set and a test set. During training on the training set, we plug $\mathcal{M}(\mathbf{e_1})$ and $\mathcal{M}(\mathbf{e_2})$ before two entity mentions e_1 and e_2 for each instance, and mask only the mention span of e_1. During inference on the test set, we evaluate the MMLM loss in four settings. (1) **No-Perturbation** plugs the $\mathcal{M}(\mathbf{e_1})$ and $\mathcal{M}(\mathbf{e_2})$, which is identical to the setting of training. (2) **Self-Perturbation** replaces $\mathcal{M}(\mathbf{e_1})$ with $\mathcal{M}(\mathbf{e_i})$, where e_i is a random entity. (3) **Other-Perturbation** replaces $\mathcal{M}(\mathbf{e_2})$ with $\mathcal{M}(\mathbf{e_i})$. (4) **All-Perturbation** replaces both $\mathcal{M}(\mathbf{e_1})$ and $\mathcal{M}(\mathbf{e_2})$ with random ones. We also evaluate these settings with a randomly-initialized mapping network without map-tuning. For analysis, we report the result in the setting **No-Plug** where there is no plugged embedding.

The result is shown in Table 13. From this table, we have three observations. (1) With map-tuning, the loss in Self-Perturbation is significantly larger than that in No-Perturbation, even close to that in All-Perturbation. It proves that MMLM trains the mapping network to extract the entity information stored in the knowledge embedding so that PLMs can utilize the information. (2) The loss in Other-Perturbation is also larger than that in No-Perturbation, which indicates that the mapping network learns to extract the connections between different entities and to feed such information into PLMs. (3) Interestingly, the loss in All-Perturbation with map-tuning is smaller than that in No-Plug, and the loss in settings without map-tuning is close to the latter. The trained mapping network may be able to convert an arbitrary knowledge embedding to an embedding that can activate the PLM's own memory of some factual knowledge. In conclusion, the three mentioned abilities of mapping networks trained by MMLM enable PLMs to know new knowledge or better recall their own knowledge. Future work may improve MMLM to get stronger mapping networks.

Method	Map-tuning Corpus	FewRel 1.0				Wiki80	Wiki-ET	EntityQuestions
		5-1	5-5	10-1	10-5			
Fine-tuning	–	91.0	95.1	85.4	90.8	86.1	77.5	41.7
+ E-BERT	–	92.3	95.6	87.6	91.4	87.8	79.0	61.3
+ PELT	–	91.2	95.8	86.1	91.6	88.2	79.6	**62.9**
+ General Map	Wikipedia Corpus	**93.7**	**96.2**	**89.6**	**92.4**	88.8	79.9	**62.9**
	Downstream Data	93.2	**96.2**	88.2	92.0	**89.1**	**81.0**	62.0

Table 14: Results of knowledge injection during fine-tuning. For general map-tuning, we can use the Wikipedia corpus mentioned in the previous section or use the data of downstream tasks.

E Is Map-tuning 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 14.

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 the large-scale unsupervised corpus. It indicates that general map-tuning does not necessitate a large amount of training data for a specific task.

F 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. In-
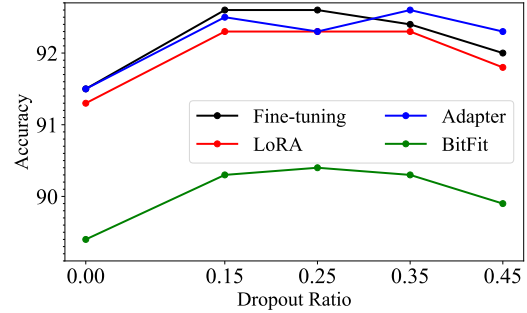

Figure 4: Effect of dropout rates on the performance of general map-tuning.

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

Table 15: Number of parameters optimized in knowledge injection. These methods are based on backbone PLMs with around 100 million parameters.

tuitively, 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 rates. 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 can not utilize the knowledge. (2) Large dropout rates are also not optimal. Empirically, the dropout rate of 0.25 is a good choice.

G Numbers of Optimized Parameters

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 15. In order to introduce external knowl-

	Fine-tuning	LoRA	Adapter	BitFit
-	35.2	36.7	38.1	35.6
E-BERT	36.9 (+1.7)	38.4 (+1.7)	39.2 (+1.1)	35.8 (+0.2)
PELT	38.8 (+3.6)	40.6 (+3.9)	41.6 (+3.5)	38.5 (+2.9)
RA	**42.7** (+**7.5**)	29.0 (−7.7)	25.0 (−13.1)	17.4 (−18.2)
K-Adapter	32.3 (−2.9)	35.8 (−0.9)	35.8 (−2.3)	35.7 (+0.1)
Map-tuning	41.9 (+6.7)	**42.8** (+**6.1**)	**44.4** (+**6.3**)	**41.1** (+**5.5**)

Table 16: Performance on filtered EntityQuestions.

edge, 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.

H Performance on EntityQuestions

We report the performance on filtered EntityQuestions in Table 16.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Left blank.

☐ A2. Did you discuss any potential risks of your work?
Not applicable. Left blank.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Left blank.

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?
4

☑ B1. Did you cite the creators of artifacts you used?
Left blank.

☒ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
These datasets are publicly available and free of use for research purposes.

☒ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
The use is consistent with their intended use.

☒ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
We use established public datasets, which should not cause privacy issues.

☑ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
4

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
4

C ☑ Did you run computational experiments?
4

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
4

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?

A

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?

A

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?

4

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.