Plug-and-Play Representation Learning of Documents for Pre-trained Models

Anonymous ACL submission

Abstract

Recently, inserting task-specific plugins such as adapters and prompts into a unified pre-trained model (PTM) to handle multiple tasks has become an efficient paradigm for NLP. In this paper, we explore to extend this paradigm from task adaptation to document representation. Specifically, we introduce plug-and-play representation learning of documents (named PlugD), which aims to represent each document as a unified task-agnostic plugins. By inserting document plugins as well as task plugins into the PTM for downstream tasks, we can encode a document one time to handle different tasks, which is more efficient than conventional methods that learn task-specific encoders to represent documents. Extensive experiments on 7 datasets of 5 typical NLP tasks show that PlugD enables models to encode documents once and for all with a unified PTM as basis, resulting in a 3.2× tuning and inference speedup while achieving comparable or even better performance. Besides, we also find that plugins can serve as an effective way to inject external knowledge into task-specific models, improving model performance without any additional model training. Our code and plugins will be released to advance future work.

1 Introduction

In recent years, fine-tuning pre-trained models (PTMs) (Radford et al., 2018; Devlin et al., 2019; Raffel et al., 2020) has been widely used in various NLP tasks and achieved breakthrough performance. But the paradigm of tuning all parameters of PTMs for ever-increasing tasks results in low computational efficiency. For this problem, many researchers are committed to exploring parameter-efficient tuning (Houlsby et al., 2019; Ding et al., 2022), aiming to freeze PTMs and learn additional task-specific plugins for PTMs, such as adapters (Houlsby et al., 2019) and prompts (Liu et al., 2021). Sufficient empirical results show that we can obtain performance comparable to tuning all parameters, by inserting these additional plugins into a frozen PTM. As a result, processing multiple tasks with a unified PTM as the backbone has gradually become a common paradigm for NLP.

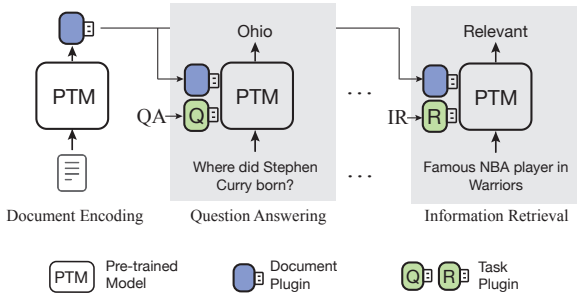

Figure 1: Illustration of plug-and-play representation learning. The document representation is decoupled from tasks. By plugging document plugins as well as task-specific functional plugins into a unified PTM, we can handle multiple tasks such as question answering and information retrieval.

Despite taking a unified model backbone across tasks, existing methods still need to build task-specific encoders to generate non-uniform document representation for specific tasks, i.e., a document has to be encoded multiple times for different tasks. Hence, a natural question is raised: *based on a unified PTM, can we further learn unified document representations for multiple tasks, encoding documents once and for all and with guaranteed transferability?* If we can obtain unified document representations based on a unified backbone model, this can form a new learning paradigm with better transferability and less computational overhead.

Inspired by parameter-efficient tuning (Ding et al., 2022) to learn task plugins, we propose to represent documents as task-agnostic plug-and-play modules around a unified PTM. Different from feature-based representation learning (Bengio et al., 2013; Dai et al., 2015), document plugins are not only features but also tiny neural layers that can drive the backbone PTM. In Figure 1, documents are encoded only once before task adaptation, and

document representations are all pluggable plugins. By plugging document plugins, the semantics and knowledge of documents can be injected into the backbone PTM to serve various downstream tasks. Due to plug-and-play representation learning, during task adaptation and inference, the model is only required to process task-specific inputs conditioned on document knowledge, with increasing minimal computational requirements and no redundant document encoding process.

To represent documents as efficient plugins and generalize these plugins to various tasks, we represent documents as additional prefix tokens (Li and Liang, 2021) for the attention layer of Transformer (Vaswani et al., 2017). For the task adaptation, we follow delta tuning (Ding et al., 2022) to learn task plugins such as adapters (Houlsby et al., 2019). Since task plugins mainly act on the model weights while prefix tokens act on the hidden states, both document and task plugins are decoupled and have good composition, enabling document plugins to be task-agnostic. To enable document plugins with sufficient semantics and knowledge, we adopt two self-supervised tasks to tune plugin parameters, including masked query prediction and inverse context prediction. Specifically, by plugging a document plugin into the PTM, both two tasks require the model to acquire information from the plugged module to predict the results.

To fulfill various requirements of different tasks, we propose two plugging strategies: *plugging during tuning* and *plugging after tuning* [1]. For plugging during tuning, document plugins are used in both tuning and inference stages, and task plugins are trained when document plugins are plugged into the PTM. This setting is suitable for tasks that require documents as input context, such as question answering. In this way, the document do not need to be encoded multiple times, which can significantly reduce the computational cost. For plugging after tuning, document plugins are only used in the inference stage. This setting can be applied in tasks with external knowledge requirements, such as relation classification. Different from previous knowledge-aware methods (Zhang et al., 2019; Wang et al., 2021a), which require to re-train the models to inject knowledge, document plugins can be directly injected into task-specific models without additional model training.

[1]Here tuning refers to downstream tuning, such as full-model fine-tuning and parameter-efficient tuning.

To verify the effectiveness of our plug-and-play framework, we conduct experiments on 7 datasets of 5 typical NLP tasks. The results show that we can generate document plugins once and successfully adapt them to various downstream tasks. Compared to competitive baselines that encode documents and task-specific inputs simultaneously, our plugin-based method can achieve more than $3.2\times$ tuning and inference speedup with comparable or even superior performance. Besides, utilizing document plugins can effectively introduce the knowledge contained in documents into the downstream models. Specifically, via representing textual knowledge as plugins and injecting them into downstream models, we achieve significant performance improvements without any plugin adaptation on both relation classification (1.22 accuracy improvements) and entity typing (2.60 F1 improvements). We argue that with the size of PTMs increasing, it is an interesting and promising direction to learn plug-and-play document representations, which can be an effective and efficient foundation to support various NLP tasks.

2 Related Work

2.1 Parameter-efficient Tuning

Recent pre-trained language models (PTMs) have shown to be effective in transfering the pre-trained parameters to downstream tasks for language representation (Devlin et al., 2019; Liu et al., 2019; Raffel et al., 2020; Radford et al., 2018; Brown et al., 2020; Han et al., 2021; Chowdhery et al., 2022). However, training and tuning large-scale pre-trained models for ever-increasing tasks is expensive in computation and storage. To address this issue, parameter-efficient tuning, which is also known as delta tuning, is proposed to perform task adaptation by fine-tuning only small amounts of parameters and keeping other parameters fixed (Zaken et al., 2022; Houlsby et al., 2019; Lester et al., 2021; Liu et al., 2021; Hu et al., 2021; Ding et al., 2022). The task-specific modules possess play-and-play characteristics and can effectively inject task knowledge into PTMs. In results, PTMs and parameter-efficient learning raise a paradigm shift: adopting a unified model to handle multiple tasks. Based on this, we explore to unify document representation across different tasks and attempts to represent documents as plug-and-play document modules. Delta tuning methods are suitable for PlugD to serve as the document representation.

2.2 Language Representation Learning

Language representation learning is a fundamental NLP task (Bengio et al., 2013; Devlin et al., 2019; Radford et al., 2018) that aims to effectively represent rich semantics distributed in text and benefit various downstream tasks. Previous efforts attempt to map the language inputs into intermediate distributed features, such as word embeddings (Mikolov et al., 2013; Kiros et al., 2015; Pennington et al., 2014; Peters et al., 2018), sentence embeddings (Conneau et al., 2017; Reimers and Gurevych, 2019; Gao et al., 2021), and document embeddings (Dai et al., 2015; Wu et al., 2018). These feature-based representations can be further used as inputs of downstream task-specific models for task adaptation. In recent years, after the emergence of powerful PTMs, many efforts have been devoted to exploring the use of PTMs to encode document semantics (Beltagy et al., 2020; Zaheer et al., 2020; Zhang et al., 2021; Mehta et al., 2022), and achieve processing multiple tasks with a unified model. However, these methods still rely on tuning PTMs into task-specific encoders to encode the same documents multiple times for different tasks, leading to expensive computational cost. Different from previous methods, we explore task-agnostic plug-and-play representation learning, aiming to achieve a paradigm based on both a unified model and unified document representations to handle various different tasks.

3 Methodology

In this section, we will present the overall framework of PlugD, and introduce how to conduct plugin representation learning with a unified PTM backbone. Then we show two strategies about how to utilize plug-and-play document representations.

3.1 Preliminary

We adopt parameter-efficient tuning as our basis to obtain both document plugins and task plugins. Therefore, in this section, we will briefly introduce two typical methods of parameter-efficient tuning, including prefix-tuning (Li and Liang, 2021) and adapter-tuning (Houlsby et al., 2019), to facilitate the introduction of PlugD.

Prefix-tuning adds several continuous virtual tokens (i.e. trainable vectors) to the input of the multi-head attention layer in each transformer block. Then, the hidden vectors of original input tokens can attend to these virtual tokens to compute outputs. Specifically, we denote the prefix tokens as $\boldsymbol{P_i}$, the original inputs of the i-th transformer layer as $\boldsymbol{x_i}$. The prefix tokens are concatenated with the key and value of the multi-head attention layer to compute the output hidden states $\mathbf{H}$ as follows,

$$\mathbf{H} = \text{Attn}(\boldsymbol{x_i}\boldsymbol{W_q}, \text{cat}(\boldsymbol{P_i}, \boldsymbol{x_i})\boldsymbol{W_k}, \text{cat}(\boldsymbol{P_i}, \boldsymbol{x_i})\boldsymbol{W_v}) \quad (1)$$

where $\boldsymbol{W_q}$, $\boldsymbol{W_k}$, and $\boldsymbol{W_v}$ are the parameters of i-th attention layer, $\text{cat}(\cdot)$ and $\text{Attn}(\cdot)$ respectively refer to the concatenation and attention function.

Adapter-tuning proposes to insert some small trainable neural layers into PTMs, named adapter layers. Each adapter layer consists of a down projection layer and an up projection layer. Given the hidden vector $\boldsymbol{h} \in \mathbb{R}^d$, where d is the hidden size, the output of adapter layer is calculated as:

$$\boldsymbol{h}_{out} = \boldsymbol{h} + \phi(\boldsymbol{h}\boldsymbol{W}_{\text{down}})\boldsymbol{W}_{\text{up}}, \quad (2)$$

where $\boldsymbol{W}_{\text{down}} \in \mathbb{R}^{d\times r}$, $\boldsymbol{W}_{\text{up}} \in \mathbb{R}^{r\times d}$, and $r \ll d$ refer to the bottleneck dimension. Generally, adapter layers can be inserted after both the output of multi-head attention layers and the output of FFN layers (Houlsby et al., 2019), or only after the output of FFN layers (Pfeiffer et al., 2021).

3.2 Overall Framework

Our primary goal is to design a framework for plug-and-play representation learning where both task model and documents are pluggable modules of a backbone PTM. As shown in Figure 2, we design PlugD, which consists of three components: PTM backbone, document plugins that contain document knowledge, and task plugins that can drive the backbone PTM to handle specific tasks. We will present these components below.

PTM Backbone PTMs have been proven effective in a wide range of downstream tasks, and raise a paradigm shift to solve multiple tasks with one unified model (Bommasani et al., 2021; Brown et al., 2020; Chowdhery et al., 2022). In view of this, we further explore unifying document representations across tasks in this paper. PlugD relies on a large-scale PTM, which can serve as a fundamental infrastructure to perform task adaptation with task-specific plugins. Note that, for our framework, any PTM with a large parameter scale can be used as the backbone.

Document Plugin Document plugins store document knowledge and are obtained before utilizing these documents for specific tasks. Inspired by recent progress in model interpretation (Petroni et al.,

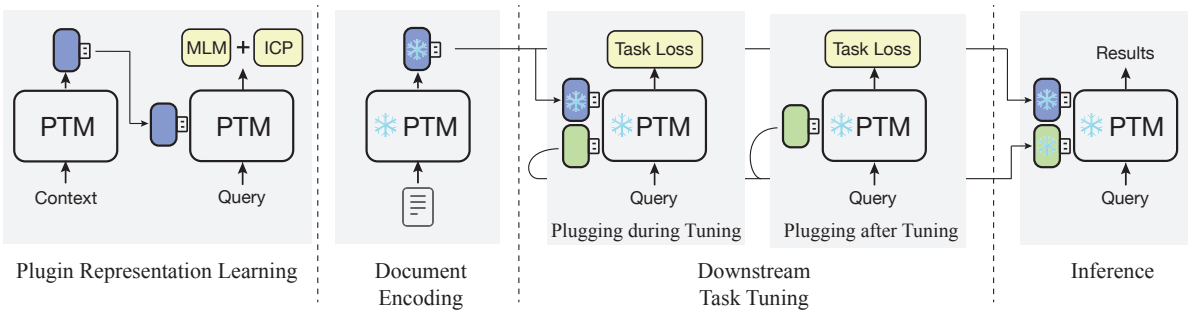

Figure 2: The illustration of PlugD in different stages. The modules with the snowflake symbol are frozen. The document plugins are generated before fine-tuning and inference and can be injected into PTMs to provide document knowledge. Here modules in yellow refer to training tasks.

2019; Jiang et al., 2020; Roberts et al., 2020; Dai et al., 2022; Mitchell et al., 2022), which claims that the parameters of PTMs store vast amounts of knowledge, we propose to encode the semantics and knowledge of documents into pluggable parameters. In this way, when the document plugin of a specific document is inserted into the backbone PTM, the PTM is empowered with the corresponding document knowledge.

Specifically, we represent documents as the prefix tokens used in Eq. (1). Given a document d with L tokens, we first encode the document with the backbone PTM to get the raw document representation $\boldsymbol{H}_d = \{\boldsymbol{h}_1, ..., \boldsymbol{h}_L\}$. Then, to map the raw representation into the prefix space, we adopt a mapping network to project the representation vectors into prefix tokens: $\boldsymbol{P}_d = \{\boldsymbol{p}_1, ..., \boldsymbol{p}_L\}$, where $\boldsymbol{p}_i = \boldsymbol{h}_i + \text{MLP}(\boldsymbol{h}_i)$. Similar to prefix-tuning, we concatenate these prefix tokens with the original hidden vectors for each attention layer.

Different from encoding documents during task adaptation or inference, prefix tokens do not involve the computation of FFN layers in Transformer. Therefore, these document plugins in the form of prefix tokens only increase limited computation requirements, whereas PlugD can achieve a significant computational speedup as a result. Due to the high storage requirement of adding different prefix tokens to different attention layers, we share $\boldsymbol{P}_d$ for all attention layers. To better integrate the semantics of documents and queries for handling tasks, document plugins are only inserted in the near-top layers of the backbone PTM. Note that, we can also utilize other model structures, such as bias parameter (Zaken et al., 2022) and LoRA (Hu et al., 2021), to represent documents in PlugD, which we leave for future work.

Task Plugin Task plugins store task-specific knowledge that can help the unified model to handle specific tasks. Task plugins are randomly initialized and tuned on the task data. During tuning task plugins for downstream tasks, we freeze the parameters of the backbone PTM and the inserted document plugins. Only the task plugin and the mapping network of the document plugin are trainable so that the document plugins can be reused across different tasks. Since parameter-efficient tuning has been widely studied in NLP, more details of learning task plugins for specific tasks can be found in Ding et al. (2022).

3.3 Plugin Representation Learning

The original PTM does not involve document plugins and is not straightforward for plug-and-play representation learning. We further conduct self-supervised learning to empower the PTM to generate task-agnostic document plugins and utilize their knowledge. In this section, we will detail the training procedure.

The document plugins are required to provide document knowledge for query understanding. Therefore, we design two self-supervised tasks, requiring the model to integrate information from both queries and documents. Specifically, given a document with n sentences, $d = \{s_1, ..., s_n\}$, we first randomly select k sentences as queries $q = \{s_1^q, ..., s_k^q\}$ and utilize the remaining sentences as context, $c = \{s_1^c, ..., s_{n-k}^c\}$, to generate the document plugin, $\boldsymbol{P}_c$. Then we require the model to perform the following tasks based on the query and document plugin.

Masked query prediction. Like masked language model (MLM) (Devlin et al., 2019; Raffel et al., 2020), we randomly replace spans from the query as special mask tokens, and require the model to recover the original spans. And the loss is same

as original MLM loss. Here, a high mask rate is required. If the mask rate is low, the model can predict the masked spans solely based on the query content and does not need to utilize knowledge from the document plugin.

Inverse context prediction. The task requires the model to predict the relation between queries and contexts. Given the query q, and the document plugin $\boldsymbol{P}_{c'}$ generated from context c', the model is required to predict whether q and c' comes from the same document. We concatenate the prompts "Do the following sentences come from the document?" with the query, and require the model to output "yes" or "no". We adopt the cross-entropy classification loss for this task.

Besides, to avoid catastrophic forgetting for tasks which do not involve documents, we also adopt the vallina masked language model objective to train model. The model is trained in a multi-task fashion, and the final training loss is the sum of three tasks. During plugin representation learning, the document plugins are generated in real-time for different documents. All parameters are tuned for plugin representation learning. After that, the document representations can be calculated and stored for further downstream fine-tuning and inference.

3.4 Plugging Strategy

As shown in Figure 2, to fulfill requirements of various tasks, we explore two plugging strategies to utilize document plugins:

Plugging during tuning aims to adopt document plugins during tuning for tasks that require documents as a necessary part of inputs, such as question answering. In this way, given an instance with the query and document as inputs, we first insert the corresponding document plugin D, computed before fine-tuning, into the backbone PTM. Then we learn task plugins with the task-specific objectives. The task plugins will be trained to capture context from the document plugins. The documents are encoded only once before downstream tuning, reducing computational costs.

Plugging after tuning aims to integrate the knowledge of document plugins into the downstream models after tuning. The setting is suitable for tasks that do not require documents as inputs, but the document knowledge can help improve the model performance. In this setting, document plugins can directly collaborate with existing task plugins. During inference, given an instance, we directly insert related document plugins into the PTM to achieve knowledge injection. This setting does not require additional training for task plugins and can be used to inject various knowledge.

Datasets	Train	Test	Task
SQuAD2.0	130.3k	118.7k	QA
RACE	87.9k	4.9k	QA
IMDB	250.0k	250.0k	TC
DBPedia	560.0k	70.0k	TC
MSMarco	532.8k	59.3k	Rank
Wiki80	8.0k	16.0k	RE
Wiki-ET	860.0k	68.2k	ET

Table 1: The statistics of evaluation datasets. Here QA, TC, RE, ET refer to question answering, text classification, relation extraction, entity typing, respectively.

4 Experiments

4.1 Evaluation Datasets

As shown in Table 1, we evaluate PlugD on 7 datasets of 5 typical NLP tasks, including 3 tasks, which require documents as necessary inputs, for plugging during tuning setting, and 2 tasks, where documents can serve as additional knowledge, for plugging after tuning setting. For plugging during tuning, we adopt three typical document-level tasks for evaluation: question answering, text classification, and document ranking. For question answering, we adopt widely used datasets, SQuAD2.0 (Rajpurkar et al., 2018), and RACE (Lai et al., 2017). We use F1 scores, exact match scores (EM), and no answer F1 scores (NA_F1) for SQuAD2.0 and accuracy for RACE as metrics. For text classification, we adopt a sentiment classification dataset, IMDB (Maas et al., 2011), and a topic classification dataset, DBPedia (Lehmann et al., 2015) with accuracy as the metric. For document ranking, we adopt a large-scale passage reranking dataset, MS-Marco (Nguyen et al., 2016) with mean reciprocal ranking (MRR@10) as the metric. We also compute the average performance scores on these 5 datasets to evaluate the overall performance.

For plugging after tuning, we adopt two knowledge-intensive tasks, relation extraction, and entity typing, to evaluate PlugD. For relation classification, we utilize Wiki80 (Han et al., 2018) with accuracy as metric for evaluation, which contains 80 relation types. For entity typing, we utilize Wiki-ET (Xin et al., 2018) for evaluation, which contains 68 fine-grained entity types. We adopt precision (P), recall (R), and F1 scores as metrics for entity typing. Both two tasks are entity-oriented. We

Setting	Models	SQuAD2.0 EM	SQuAD2.0 F1	SQuAD2.0 NA_F1	RACE Acc.	IMDB Acc.	DBPedia Acc.	MSMarco MRR@10	Average
Task-Specific	ED2LM	74.30	77.78	61.58	73.81	95.71	99.24	36.89	76.69
	LateInter	73.00	76.44	61.19	74.43	93.11	99.31	33.71	75.40
Task-Agnostic	ED2LM-T	68.20	72.35	60.84	71.73	95.36	99.21	35.37	74.80
	LateInter-T	72.92	76.13	61.62	73.67	93.94	99.30	32.80	75.17
	All-Hidden	71.34	76.33	61.92	76.29	95.33	99.24	35.35	76.51
	PlugD	**76.06**	**79.89**	**62.33**	**77.82**	**95.58**	**99.25**	**35.98**	**77.70**

Table 2: The main results of our proposed PlugD and baselines for plugging during tuning.

adopt entity information as knowledge base to improve performance. We encode entity description in Wikidata5M (Wang et al., 2021b) for document plugins to provide knowledge after tuning.

4.2 Training Details

We utilize the widely used T5-large (Raffel et al., 2020), as our PTM backbone and adapters as our task plugins. We conduct plugin representation learning on a large-scale dataset, C4 (Raffel et al., 2020) for 28k steps. We set the mask rate of masked query prediction task as 0.5. We train and tune models with half precision floating point on NVIDIA A100 GPUs. We set the learning rate as 10^{-4} and batch size as 1024. We use Adam to optimize our models. All the downstream models are tuned with adapters. The bottleneck dimension of adapters is set as 32. Please refer to the Appendix for details.

4.3 Baselines

Plugging during tuning. Here we compare PlugD with two representative dual models, which encode queries and documents separately to generate query-agnostic but task-specific document representation. 1) **ED2LM** (Hui et al., 2022) utilize the encoder-decoder architecture, where the documents are inputted into the encoder and queries are inputted into the decoder. 2) **LateInter** (Khattab and Zaharia, 2020) first independently encodes the queries and documents using the first several layers of pre-trained model, and then fuses the information together with the last two layers. 3) To make ED2LM and LateInter task-agnostic, we freeze the parameters of document encoders and only tune parameters in the decoder. In this way, the generated document representation can be used across tasks. We denote these two models for task-agnostic representation as **ED2LM-T** and **LateInter-T**. 4) PlugD regards the generated document representation as prefix tokens. To verify the effectiveness, we also compare our method with a competitive baseline, which preserves all hidden vectors of documents as document plugins and inserts the vectors into the PTM layer-by-layer to fuse the query-document information. We denote the method as **All-Hidden**.

Plugging after tuning. We attempt to inject unstructured textual knowledge into PTMs without additional model tuning. Existing methods mainly focus on enhancing PTMs with structural knowledge during pre-training or fine-tuning (Zhang et al., 2019; Wang et al., 2021a; Bosselut et al., 2019). These methods require retraining the downstream models to achieve knowledge injection, which thus cannot be easily adopted in this setting. Therefore, we compare PlugD with the following baseline models: 1) Original models (**Adapter**) do not inject knowledge for inference. 2) **Concat** trains the models without knowledge injection. After tuning, Concat directly concatenates the sentence and entity description together as inputs. 3) **All-Hidden** preserves the hidden vectors of the entity description from all layers, and inserts these vectors into the self-attention layers.

4.4 Plugging during Tuning

We present the comparison results between baseline models and PlugD in Table 2. From this table, we can observe that: (1) Our proposed PlugD can significantly outperform baseline models for task-agnostic representation on almost all downstream tasks. Besides, PlugD can also achieve comparable or even superior results with task-specific models, while reducing lots of computational costs for fine-tuning. The results suggest that PlugD can effectively capture document semantics and inject them into the PTM to provide context. (2) Both ED2LM and LateInter degrade performance significantly in task-agnostic setting. It indicates that plug-and-play representation learning is quite challenging, and the document information provided by these two methods can not be well utilized by the PTM. (3) All-Hidden can achieve competitive results in the task-agnostic setting and outperform other base-

Datasets	Wiki80	Wiki-ET		
	Acc.	P	R	F1
Adapter	86.32	79.24	72.22	75.55
Concat	86.78	80.44	69.70	74.69
All-Hidden	86.66	**81.46**	72.22	76.56
PlugD	**87.54**	80.92	**75.57**	**78.15**

Table 3: The results for plugging after tuning.

Datasets	SQuAD2.0	RACE	MSMarco
PlugD	79.89	77.82	35.98
w/o ICP	79.78	77.84	35.28
w/ None	78.19	76.41	34.85

Table 4: The results of ablation study.

lines. It shows that representing documents as plugins to provide context is effective, even though the model has not been trained to generate plugins. However, as All-Hidden has to store hidden vectors from all layers, it requires $24\times$ storage cost than PlugD and thus is unsuitable for real-world application. Besides, we highlight the importance of plugin representation learning tasks, which enable PlugD to outperform All-Hidden significantly. (4) For some tasks which do not require complex reasoning, including text classification, all models can achieve comparable results. But on challenging tasks, including question answering and document reranking, our proposed PlugD cannot consistently achieve superior results to task-specific methods, suggesting that there still require future efforts for plug-and-play document representation.

4.5 Plugging after Tuning

The comparison results are shown in Table 3. From the results, we can observe that: 1) The baseline models cannot achieve consistent improvement on two tasks. The plugging after tuning setting requires to inject knowledge after downstream tuning, which leads to the gap between training and evaluation. Thus, this setting is challenging and straightforward methods are not suitable for this setting. 2) PlugD can achieve significant improvement on both two datasets (1.22 accuracy improvements on Wiki80, and 2.60 F1 improvements on Wiki-ET). It indicates that the document plugins can capture useful knowledge from the entity information and inject it into the PTM. It is worthy of note that we can insert various textual knowledge into the downstream models via PlugD.

4.6 Ablation Study

In this section, we conduct an ablation study to verify the effectiveness of our proposed plugin representation learning tasks. We show the results of the models, which are trained without inverse context prediction (**w/o ICP**), or are not further trained (**w/ None**). As there is no significant performance difference on text classification, we only present the results on question answering and document reranking tasks.

The results are shown in Table 4. We can find that 1) PlugD with no further trainig leads to a significant performance drop, which further indicates that the proposed training task can help the PTM to generate and acquire knowledge from the document plugins. 2) PlugD with no further trainig can still achieve comparable results with the All-Hidden, the best performing baseline model, while reducing the storage cost. The results prove that the proposed framework is effective in plug-and-play representation. Both the training task and framework empower PlugD to generate document plugins with sufficient semantics. 3) PlugD without ICP leads to performance drop on document reranking task, while has limited impact on question answering tasks. As the ICP task is similar to the reranking task, and thus ICP can serve as a method for data augmentation. We encourage future researchers to propose more effective tasks to improve the performance of plug-and-play document representation.

4.7 Transferability Analysis

In this section, we want to explore the effectiveness of training tasks on document representation transferability. Here we present the results of ED2LM, which can outperform other baselines. Specifically, we train the task-specific document encoder on a source task, and then reuse the encoder on other target tasks to continually train the rest of the model. The results are shown in Figure 3.

From the results, we can observe that 1) The non-diagonal values of the matrix are consistently smaller than the diagonal values. It suggests that training the document encoder with existing supervised tasks can hardly benefit other target tasks. PlugD trained with two self-supervised objectives can provide transferable document representation and achieve superior results, which indicates the effectiveness of our proposed plugin representation learning tasks. 2) The document encoders trained on the question answering task perform better than

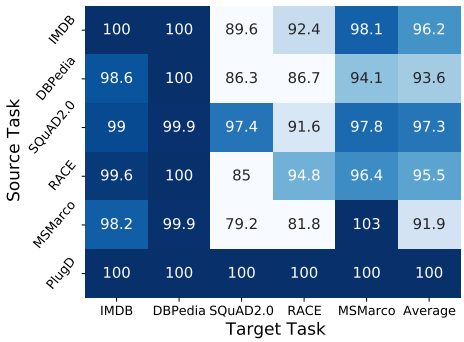

Figure 3: Relative transfer performance (transfer performance / PlugD's performance)(%).

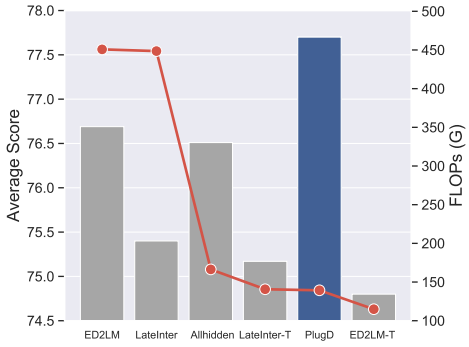

Figure 4: The average scores and FLOPs for each method. The models are arranged in reverse order of computational cost.

encoders trained on other tasks. It indicates that training with challenging tasks may lead to better performance, which we leave for future work.

4.8 Computational Cost

In this section, we compare the computational cost of the methods. Here, we present the floating point operations (FLOPs) required to process one data in downstream task tuning for each method. We assume that the document, query, and answer contain 512, 48, and 32 tokens, respectively. These methods adopt T5-large model as the backbone.

The results are shown in Figure 4. From this figure, we observe that: 1) The methods for task-agnostic representation require much less computational cost than methods for task-specific representation. That is because methods for task-agnostic representation can generate document representation before fine-tuning and do not require to encode documents multiple times. Especially, our method PlugD can achieve 3.23× speed up (139.3 GFLOPs vs. 450.7 GFLOPs). 2) The methods for task-agnostic representation generally are inferior to task-specific methods. Our proposed PlugD can achieve better average scores than all baseline

Datasets	RACE Acc.	Tuning FLOPs	Inference FLOPs
ED2LM	73.81	450.8	114.9
PlugD	77.82	139.3	139.3
T5-base	73.87	131.6	131.6
T5-large	81.16	453.1	453.1

Table 5: Comparison between our method with query-specific methods.

(77.70 vs. 76.69) and preserve low computational cost. It indicates that PlugD can effectively capture document knowledge and further inject it into PTM to perform downstream adaptation.

4.9 Comparison with Query-Specific Model

In this section, we compare PlugD with the query-specific models, which concatenate the query and document together as the model inputs, which even need to encode the documents multiple times for different queries. We present the accuracy on the challenging question-answering dataset, RACE. The results are shown in Table 5. From the results, we can find that 1) Query-agnostic models, including ED2LM and PlugD can achieve significant speedup for inference than corresponding query-specific models with the same model size. And our task-agnostic model can further reduce the computational costs of downstream task tuning. 2) Though there is a gap between the performance of PlugD and the query-specific model with the same parameter size, PlugD can significantly outperform the query-specific model with similar computational cost. It indicates that training large-scale PTMs under plug-and-play representation learning is a promising direction, and can effectively and efficiently handle document-level tasks.

5 Conclusion

In this paper, we explore a new paradigm, named plug-and-play representation learning, which aims to represent documents as pluggable modules for PTMs. In plug-and-play representation learning, we can get rid of encoding the same document multiple times for different tasks and achieve using a unified model and unified representation for different tasks. The extensive experiments prove that our proposed PlugD can significantly reduce the computational cost and effectively inject document knowledge into PTMs to improve performance. In the future, we will explore more effective plugin representation learning tasks and frameworks for plug-and-play representation learning.

References

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *CoRR*, abs/2004.05150.

Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen Creel, Jared Quincy Davis, Dorottya Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, and et al. 2021. On the opportunities and risks of foundation models. *CoRR*, abs/2108.07258.

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 the ACL*, pages 4762–4779.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Proceedings of NeurIPS*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways. *CoRR*, abs/2204.02311.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of EMNLP*, pages 670–680.

Andrew M. Dai, Christopher Olah, and Quoc V. Le. 2015. Document embedding with paragraph vectors. *CoRR*, abs/1507.07998.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. 2022. Knowledge neurons in pretrained transformers. In *Proceedings of ACL*, pages 8493–8502.

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. *CoRR*, abs/2203.06904.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. Simcse: Simple contrastive learning of sentence embeddings. In *Proceedings of EMNLP*, pages 6894–6910.

Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, Wentao Han, Minlie Huang, Qin Jin, Yanyan Lan, Yang Liu, Zhiyuan Liu, Zhiwu Lu, Xipeng Qiu, Ruihua Song, Jie Tang, Ji-Rong Wen, Jinhui Yuan, Wayne Xin Zhao, and Jun Zhu. 2021. Pre-trained models: Past, present and future. *AI Open*, 2:225–250.

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.

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*, volume 97, pages 2790–2799.

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. *CoRR*, abs/2106.09685.

Kai Hui, Honglei Zhuang, Tao Chen, Zhen Qin, Jing Lu, Dara Bahri, Ji Ma, Jai Prakash Gupta, Cícero Nogueira dos Santos, Yi Tay, and Donald Metzler. 2022. ED2LM: encoder-decoder to language model for faster document re-ranking inference. In *Findings of ACL*, pages 3747–3758.

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know. *TACL*, 8:423–438.

Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over BERT. In *Proceedings of SIGIR*, pages 39–48.

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. In *Proceedings of NeurIPS*, pages 3294–3302.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard H. Hovy. 2017. RACE: large-scale reading comprehension dataset from examinations. In *Proceedings of EMNLP*, pages 785–794.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. 2015. Dbpedia - A large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of EMNLP*, pages 3045–3059.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of ACL-IJCNLP*, pages 4582–4597.

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. *CoRR*, abs/2107.13586.

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. *CoRR*, abs/1907.11692.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of ACL-HLT*, pages 142–150.

Harsh Mehta, Ankit Gupta, Ashok Cutkosky, and Behnam Neyshabur. 2022. Long range language modeling via gated state spaces. *CoRR*, abs/2206.13947.

Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of NeurIPS*, pages 3111–3119.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. 2022. Fast model editing at scale. In *Proceedings of ICLR*.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A human generated machine reading comprehension dataset. In *Proceedings of NeurIPS (Workshop)*, volume 1773.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of EMNLP*, pages 1532–1543.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, pages 2227–2237.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick S. H. Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander H. Miller. 2019. Language models as knowledge bases? In *Proceedings of EMNLP-IJCNLP*, pages 2463–2473.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. Adapterfusion: Non-destructive task composition for transfer learning. In *Proceedings of EACL*, pages 487–503.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding with unsupervised learning.

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

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. In *Proceedings of ACL*, pages 784–789.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of EMNLP-IJCNLP*, pages 3980–3990.

Adam Roberts, Colin Raffel, and Noam Shazeer. 2020. How much knowledge can you pack into the parameters of a language model? In *Proceedings of EMNLP*, pages 5418–5426.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

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*, volume ACL/IJCNLP 2021, 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.

Lingfei Wu, Ian En-Hsu Yen, Kun Xu, Fangli Xu, Avinash Balakrishnan, Pin-Yu Chen, Pradeep Ravikumar, and Michael J. Witbrock. 2018. Word mover's embedding: From word2vec to document embedding. In *Proceedings of EMNLP*, pages 4524–4534.

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.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2020. Big bird: Transformers for longer sequences. In *Proceedings of NeurIPS*.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of ACL*, pages 1–9.

Hang Zhang, Yeyun Gong, Yelong Shen, Weisheng Li, Jiancheng Lv, Nan Duan, and Weizhu Chen. 2021. Poolingformer: Long document modeling with pooling attention. In *Proceedings of ICML*, volume 139, pages 12437–12446.

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.

A Implementation Details

In this section, we introduce the implementation details for downstream task tuning. We adopt adapters (Houlsby et al., 2019) to perform task adaptation. Specifically, following Pfeiffer et al. (2021), we add the adapters after the layernorm operation of self-attention layers and feed-forward layers. The parameters of adapters are initialized following a zero-mean Gaussian distribution with standard deviation as 10^{-2}. The learning rate is selected from $\{10^{-4}, 2 \times 10^{-4}, 10^{-3}\}$. The batch size for downsteram tasks is set as 64. We report the results of checkpoints which achieve best performance on the validation set. And for datasets without the validation set, we directly use the test data for validation. We implement the models with ModelCenter [2], a efficient implementation for big models.

SQuAD2.0 We train the model on SQuAD2.0 following a sequence-to-sequence paradigm, where the inputs are documents and questions, and the outputs are the answers. We set the maximum length of documents as 512, the maximum length of questions as 48, and the maximum length of answers as 20. During evaluation, we adopt the greedy decoding strategy. Besides, we will make the generated answers lowercase, and remove punctuation, articles and extra blank character for evaluation.

RACE RACE is a multi-choice question answering dataset. And we inputs the documents, questions, options into models, and use the decoding scores of "right" token as the option scores. Then the cross-entropy function is adopted on the logit of four options to predict the final answers.

IMDB and DBPedia IMDB and DBPedia are two text classification datasets. For IMDB, we adopt the decoding scores of "positive" and "negative" tokens as the scores for sentiment classification. For DBPedia, which contains 14 types of documents, we use the decoding scores of the first token in the type name as scores for classification.

MSMarco MSMarco is a large-scale reranking dataset. In this paper, we adopt the passage ranking set for evaluation. Following previous work (Hui et al., 2022), we use BM25 to generate the top1000 candidates for each query, and we utilize the checkpoints which are trained for $10,000$ steps for eval-

[2] https://github.com/OpenBMB/ModelCenter

uation. We train the model with a binary classification objective. The maximum length of documents is set as 256.

Wiki80 Wiki80 is a large-scale relation extraction dataset, which contains 80 relation types. We follow the data split of (Zhang et al., 2019). Similar to DBPedia, we use the decoding scores of the first token in the relation name as the classification scores. And we directly concatenate the description of the head entity and tail entity to generate document plugs.

Wiki-ET Wiki-ET is a fine-grained entity typing dataset, which contains 68 entity types. We formulate the task as a multi-label text classification problem. We adopt binary cross-entropy loss to optimize the models.

B Limitation

In this paper, we explore a new paradigm, play-and-play representation learning, which aims to represent documents as plugins for large-scale pre-trained language models. However, there are some limitation of our proposed method. Though our method can outperform other query-agnostic methods, but can not outperform the query-specific method with the same model size, which still need future exploration. Besides, in this paper, we implement the model with Adapter as the task plugin and prefix as the document plugin. It is worthy of exploration that how do other delta tuning method perform in plug-and-plug representation learning. In the future, we will still devote effort to plug-and-plug representation learning to promote the future progress in large-scale pre-trained language models.