Decoder Tuning: Efficient Language Understanding as Decoding

Ganqu Cui[1], **Wentao Li**[1], **Ning Ding**[1], **Longtao Huang**[2], **Zhiyuan Liu**[1,3]*, **Maosong Sun**[1,3]*
[1] NLP Group, DCST, IAI, BNRIST, Tsinghua University, Beijing
[2] Alibaba Group [3] IICTUS, Shanghai
cgq22@mails.tsinghua.edu.cn

Abstract

With the evergrowing sizes of pre-trained models (PTMs), it has been an emerging practice to only provide the inference APIs for users, namely model-as-a-service (MaaS) setting. To adapt PTMs with model parameters frozen, most current approaches focus on the input side, seeking for powerful prompts to stimulate models for correct answers. However, we argue that input-side adaptation could be arduous due to the lack of gradient signals and they usually require thousands of API queries, resulting in high computation and time costs. In light of this, we present Decoder Tuning (DecT), which in contrast optimizes task-specific decoder networks on the output side. Specifically, DecT first extracts prompt-stimulated output scores for initial predictions. On top of that, we train an additional decoder network on the output representations to incorporate posterior data knowledge. By gradient-based optimization, DecT can be trained within several seconds and requires only one PTM query per sample. Empirically, we conduct extensive natural language understanding experiments and show that DecT significantly outperforms state-of-the-art algorithms with a 200× speed-up. Our codes are available at https://github.com/thunlp/DecT.

1 Introduction

Recent advances in pre-trained models (PTMs) demonstrate the power of the "pre-training-fine-tuning" paradigm, which empowers broad downstream NLP tasks with a single backbone model (Devlin et al., 2019; Raffel et al., 2020; Radford et al., 2019). Given the million even billion-scale models, model-as-a-service (MaaS) has become an emerging practice in deploying massive PTMs, where users can only get access to model inference APIs (Brown et al., 2020; Sun et al., 2022b). Under such a scenario, PTMs' parameters are frozen, and users cannot fine-tune the model on downstream tasks for adaptation. To find an alternative way, researchers have studied MaaS PTM adaptation methods extensively.

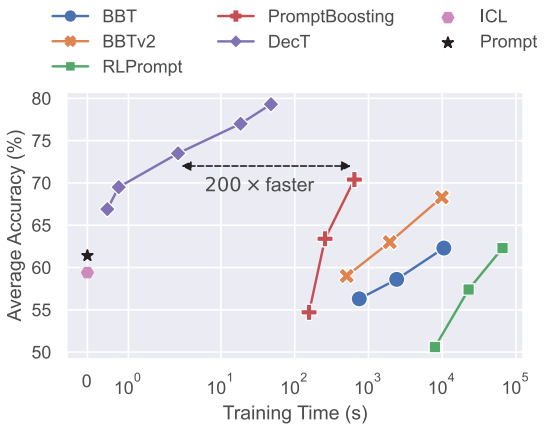

Figure 1: Accuracy v.s. training time for MaaS adaptation methods under different training shots. We plot $\{1, 4, 16, 64, 256\}$-shot DecT, 0-shot Prompt, 1-shot ICL and $\{1, 4, 16\}$-shot for other algorithms. DecT outperforms all baselines by a large margin with a 200×speed-up.

Most existing approaches in this line are based on *prompts*, which modify inputs with specific patterns. By wrapping inputs into cloze-style questions or prepending inputs with a few demonstrative examples, PTMs could produce the right outputs directly and show strong "in-context" learning abilities (Petroni et al., 2019; Brown et al., 2020) without any parameter update. Besides heuristic prompt design, some recent works try to optimize the input prompts without gradients. Among them, Black-box Tuning (BBT) (Sun et al., 2022b) and BBTv2 (Sun et al., 2022a) apply evolutionary algorithm (Hansen and Ostermeier, 2001) on continuous prompt tokens, while RLPrompt (Deng et al., 2022) adopts reinforcement learning to find discrete prompt tokens. Nevertheless, gradient-free optimization is rather difficult and these input-

*Corresponding Author.

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 15072–15087

side methods need to query the PTMs thousands of times for optimization, which leads to huge inference costs in terms of time and computation resources. Moreover, their final performance is not satisfying as well.

Given the flaws of *input-side* adaptation, we turn to *output-side* adaptation, which builds tunable decoder networks on model outputs. Comparatively, output-side adaptation enjoys two major advantages: (1) We can directly tune decoder networks on top of model outputs with back-propagation rather than arduous alternatives. (2) We can reduce thousands of model queries to only once per sample. However, designing decoder networks is not straightforward. Past studies have shown that merely tuning an MLP or LSTM (Hochreiter and Schmidhuber, 1997) over output features cannot provide satisfying results (Sun et al., 2022a,b), leaving this path underexplored.

In this work, we aim to solve the performance issue for output-side adaptation, and we argue that there are two critical reasons behind it: (1) Simply utilizing PTMs as feature extractors ignores the infilling ability of PTMs, which is a strong prior for adaptation. (2) MLP and LSTM are not proper networks especially when training data is not sufficient.

Based on these findings, we present Decoder Tuning (DecT), an enhanced output-side adaptation method. Specifically, DecT has two crucial design choices to address the above issues. First, DecT queries the PTM with prompts and adopts model output scores as the initial predictions, which takes advantage of internal model knowledge. Second, on top of the output representations, we select a Prototypical Network (ProtoNet) (Snell et al., 2017) as the decoder network and train it to fit the training data, which is more suitable for few-shot learning. In this way, DecT modifies the initial model scores with subsequent training data, thus achieving better performance.

Through few-shot learning experiments on ten language understanding datasets, we highlight three advantages of DecT (see Figure 1). (1) DecT achieves over 3% absolute accuracy improvement on average, greatly outperforming previous works. (2) DecT is highly efficient. Compared with major prompt engineering baselines, DecT dramatically reduces the average adaptation time from over 9,800 seconds (BBTv2) to 3 seconds. (3) DecT only requires one PTM query for each example, while other input-side optimization methods need about 10^4 calls. This advantage is vital when PTM calls are not for free. In addition, we conduct extensive ablation studies and validate the impact of each component of DecT.

2 Preliminaries

Given a set of training data $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{N}$ and PTM $\mathcal{M}$, we need to predict the label $y \in \{1, \ldots, K\}$ for sample x, where K is the number of classes. We assume that each class has the same amount of n training samples.

In the MaaS setting, $\mathcal{M}$ is a black-box inference API with fixed parameters. Therefore, we can only query the model with input x and get corresponding outputs. To better utilize the PTMs, it has been a common practice to wrap input samples into prompts. Specifically, we enclose each input x into a template $\mathcal{T}$ with a `[MASK]` token (here we assume using a masked language model). Then, we query $\mathcal{M}$ with $\mathcal{T}(x)$ and get the final layer hidden states $\mathbf{h}$ at the `[MASK]` position and scores $\mathbf{s} = S_{\mathcal{M}}(\mathcal{T}(x)) \in \mathbb{R}^K$ over label words $\mathcal{V}$. Take sentiment analysis as an example, we can use

$$\mathcal{T}(x) = x \text{ In summary, it was } \texttt{[MASK]}.$$

as the template with $\mathcal{V} = \{\text{bad}, \text{great}\}$ as label words for negative and positive sentiment respectively. The output scores on these label words further correspond to the classes.

3 Methodology

In this section, we elaborate on our proposed Decoder Tuning (DecT) method for the classification task. We start with reviewing current input-side adaptation methods, then give an overview of DecT and finally detail it step-by-step.

3.1 Input-side Adaptation

Previous MaaS adaptation methods seek for optimal prompts that stimulate PTMs to output correct answers[1]. Without loss of generality, we formulate these methods with a transformation function $f(\cdot)$ which pre-processes the input x. $f(\cdot)$ can be specialized by adding demonstrations (Brown et al., 2020), discrete prompt tokens (Deng et al., 2022) or soft ones (Sun et al., 2022a,b). Denote the final score as $q(x)$ and probability as

[1] BBTv2 (Sun et al., 2022a) further optimizes prompt tokens in the intermediate layers, but we omit this here.

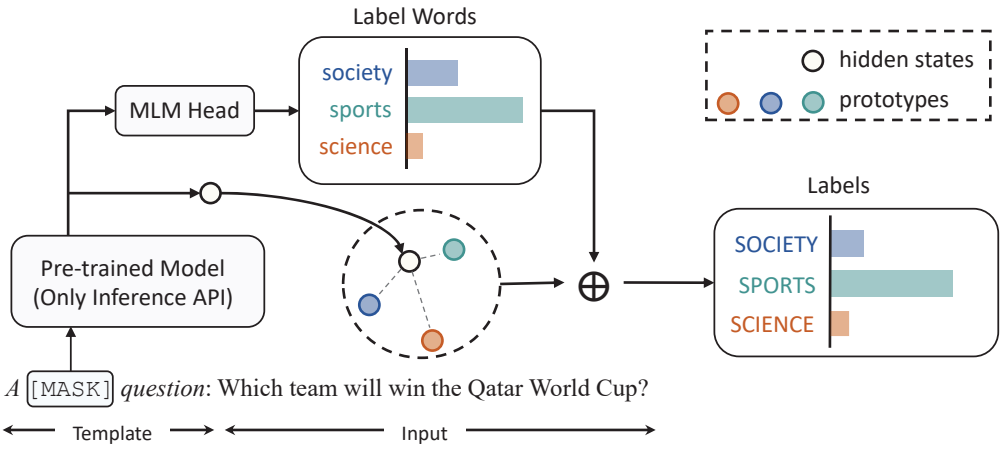

Figure 2: Pipeline of DecT. We feed the PTM with prompts and collect model output scores over a set of label words (**Top**) and hidden states at `[MASK]` position. The hidden states are used to train a ProtoNet to fit training data (**Bottom**). We make final predictions by combining model and ProtoNet scores.

$P(y|x) = \text{Softmax}(q(x))$, these methods define $q(x) = S_{\mathcal{M}}(f(x))$ and optimize $f(\cdot)$ for correct predictions. Although optimizing $f(\cdot)$ without model gradients is possible, we argue that it is highly burdensome. Forwarding through a large "black box" model $\mathcal{M}$, it is rather challenging to find corresponding inputs for specific outputs without the guidance of gradient signals. As a result, users may get suboptimal performance with expensive query costs. We empirically validate it in experiments.

3.2 Overview of DecT

For more effective and efficient PTM adaptation, we turn to output-side adaptation rather than input-side. Overall, output-side adaptation can be viewed as a post-processing of model outputs which uses another function $g(\cdot)$ to process the model outputs, and get the final scores $q(x) = g(S_{\mathcal{M}}(\mathcal{T}(x)))$. Different from input-side ones, output-side adaptation is easy-to-optimize with gradient descent, and for each sample, we only need to query the PTM once.

For DecT, as shown in Figure 2, we model the post-processing as decoding, which refers to a post-modification to the initial model predictions. Specifically, we first query the PTM with prompt-enclosed inputs to get model outputs, including the scores for each class and hidden states. Intuitively, output scores contain prior knowledge inside the PTM, so we retain them as part of the final scores.Then, we tune an additional decoder function on the hidden states to fit the training data and make final predictions. Next, we describe how we query the model and then specify the implementation of the score function.

3.3 Querying with Prompts

To get model outputs, we simply follow the procedure in Section 2 and query the model with manual template-wrapped inputs. We then process the scores by calibration.

Calibration. As stated in Zhao et al. (2021), PTMs tend to assign higher probabilities on those frequent label words, leading to biased output scores. To eliminate the prediction bias, we further calibrate the output scores with empty input $x_c =$"" following (Zhao et al., 2021). Querying the model with x_c, we can obtain the calibaration scores $\mathbf{s}_c$ and normalize them by $\mathbf{s}_c/\text{mean}(\mathbf{s}_c)$. Then we calibrate $\mathbf{s}$ by

$$\hat{\mathbf{s}} = \text{diag}(\mathbf{s}_c/\text{mean}(\mathbf{s}_c))^{-1}\mathbf{s}. \quad (1)$$

After that, the calibrated scores $\hat{\mathbf{s}}$ are balanced over classes.

3.4 Tuning the Outputs

After getting the hidden states and calibrated scores, we perform DecT outside the PTM to modify the output scores fitting the training data. Denote the final score on class k as $q(x, k)$, we calculate it by the following function:

$$q(x, k) = \text{Dec}(\mathbf{h}, k) + \lambda\hat{\mathbf{s}}_k, \quad (2)$$

where $\text{Dec}(\cdot)$ is a trainable decoder function, λ is a hyperparameter controlling the weight of PTM scores and $\hat{\mathbf{s}}_k$ is the k-th logit in $\hat{\mathbf{s}}$. By tuning $\text{Dec}(\cdot)$, the final predictions incorporate training

data on top of PTM outputs, which combine both knowledge effectively.

The design choice of $\mathrm{Dec}(\cdot)$ is fairly flexible. In practice, we select Prototypical Networks (ProtoNet) (Snell et al., 2017) due to their simplicity and remarkable performance in few-shot learning and prompt-based tuning (Cui et al., 2022). For this, we project the hidden states with a linear layer parameterized by $\mathbf{W}$ and get sample representation

$$\mathbf{v} = \mathbf{W}\mathbf{h}. \tag{3}$$

On prototypes, classical approaches model them as points in the embedding space, which overlook the different class characteristics. Inspired by Ding et al. (2022a), we model prototypes as hyperspheres with an additional radius parameter. Concretely, the prototype for class k contains two parameters, center position vector $\mathbf{z}_k$ and radius scalar r_k. We randomly initialize $\mathbf{z}_k$ and initialize r_k as the average distance between $\mathbf{z}_k$ and instances in class k:

$$r_k = \frac{1}{N_k} \sum_{i}^{y_i=k} \|\mathbf{v}_i - \mathbf{z}_k\|_2. \tag{4}$$

As for the score function, we calculate the Euclidean distances between instances and prototypes.

$$\mathrm{Dec}(\mathbf{h}, k) = -\|\mathbf{W}\mathbf{h} - \mathbf{z}_k\|_2 + r_k. \tag{5}$$

According to Eq. 2, the final logit is

$$q(x, k) = -\|\mathbf{W}\mathbf{h} - \mathbf{z}_k\|_2 + r_k + \lambda \hat{\mathbf{s}}_k. \tag{6}$$

From a geometric view, the score function calculates the distance from instance x to the "surface" of the prototype, where $r_k + \lambda \hat{\mathbf{s}}_k$ is the whole radius acting like the bias term. With the scores, we can calculate the predicted probability by the Softmax function:

$$P(y = k|x) = \frac{\exp(q(x, k))}{\sum_{k'=1}^{K} \exp(q(x, k'))}, \tag{7}$$

and we can optimize $\mathbf{W}$ and r_k by the cross-entropy loss

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \log P(y_i|x_i). \tag{8}$$

4 Experiments

In this section, we first introduce the experimental settings (Section 4.1), then discuss the results for few-shot experiments (Section 4.2), efficiency comparison (Section 4.3), and experiment results for more training data (Section 4.4).

4.1 Experimental Settings

Datasets. We conduct experiments on four typical natural language understanding tasks. For sentiment analysis, we select SST2 (Socher et al., 2013), Yelp P. (Zhang et al., 2015) and IMDB (Maas et al., 2011). For text classification, we use AG's News, Yahoo (Zhang et al., 2015) and DBPedia (Lehmann et al., 2015). For natural language inference (NLI), we adopt RTE (Dagan et al., 2005; Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009), SNLI (Bowman et al., 2015) and MNLI (Williams et al., 2018). For entity typing, we experiment on FewNERD (Ding et al., 2021b). We report dataset statistics in Appendix A.1.

Splits. We randomly sample $n = 1, 4, 16$ data instances for each class from the training set for few-shot learning, and sample same amount data for validation. For datasets in GLUE (Wang et al., 2019) (SST2, RTE, MNLI) and SNLI, we use the original validation sets as test sets following Zhang et al. (2021). For other datasets, we evaluate on their original test sets.

Baselines. We compare with representative MaaS PTM adaptation methods. **Prompt** refers to directly performing zero-shot classification with template-wrapped examples. **In-context learning (ICL)** (Brown et al., 2020) further concatenates some exemplars before the test samples. **BBT** (Sun et al., 2022b) optimizes soft prompt tokens with an evolutionary algorithm, and **BBTv2** (Sun et al., 2022a) further inserts deep prompts to intermediate layers for better performance. **RLPrompt** (Deng et al., 2022) is another recent algorithm that optimizes discrete prompts with reinforcement learning. **PromptBoosting** (Hou et al., 2022) is a concurrent work that applies boosting algorithm for prompt ensembling. We report the details of baselines in Appendix A.2.

Environments. For all experiments, we use NVIDIA A100 and RTX 2080 Ti GPUs. We implement DecT with PyTorch (Paszke et al., 2019), HuggingFace Tansformers (Wolf et al., 2020), and OpenPrompt (Ding et al., 2022b).

Implementation Details. For all methods, we use the same $\text{RoBERTa}_{\text{LARGE}}$ (Liu et al., 2019) as the backbone model. For DecT, we set the representation dimension to 128 and optimize the parameters for 30 epochs with Adam optimizer (Kingma and Ba, 2015). The learning rate is 0.01. On the

n	Method	SST2	IMDB	Yelp	AG	DB	Yahoo	RTE	SNLI	MNLI-m/mm	NERD	Avg.
0	**Prompt**	83.3	89.4	87.1	80.9	68.4	49.9	52.4	40.7	50.8/51.7	21.1	61.4
1	**ICL**	$81.5_{3.7}$	$65.6_{11.4}$	$81.1_{10.6}$	$66.7_{4.8}$	$71.7_{2.6}$	$53.2_{6.2}$	$45.0_{4.7}$	$46.1_{5.3}$	$\mathbf{53.6_{0.5}}/\mathbf{53.9_{0.8}}$	$34.7_{3.3}$	$59.4_{4.9}$
	BBT	$83.4_{1.3}$	$89.0_{0.1}$	$89.7_{0.1}$	$75.4_{0.8}$	$59.1_{1.7}$	$31.2_{2.7}$	$52.3_{1.4}$	$38.5_{0.8}$	$43.4_{2.5}/42.9_{3.3}$	$14.1_{2.3}$	$56.3_{1.5}$
	BBTv2	$83.3_{2.5}$	$89.0_{0.2}$	$89.9_{0.2}$	$74.3_{3.2}$	$74.2_{5.2}$	$34.0_{3.5}$	$48.2_{5.7}$	$38.6_{4.0}$	$44.2_{3.2}/44.3_{4.5}$	$29.0_{0.8}$	$59.0_{3.0}$
	RLPrompt	$63.5_{6.3}$	$65.0_{6.5}$	$66.3_{6.9}$	$72.5_{4.5}$	$65.6_{5.5}$	$38.1_{5.8}$	$53.8_{5.3}$	$36.5_{3.0}$	$40.3_{2.0}/41.0_{2.1}$	$14.5_{1.8}$	$50.6_{4.7}$
	PromptBoosting	$86.7_{2.6}$	$82.4_{6.1}$	$88.7_{2.5}$	$58.7_{11.8}$	$73.0_{4.8}$	$23.7_{7.0}$	$50.0_{5.9}$	$43.5_{6.1}$	$36.8_{1.6}/36.3_{2.3}$	$22.0_{0.8}$	$54.7_{4.6}$
	DecT	$\mathbf{90.8_{0.2}}$	$\mathbf{91.2_{0.3}}$	$\mathbf{94.8_{0.1}}$	$\mathbf{79.9_{1.1}}$	$\mathbf{78.8_{0.9}}$	$\mathbf{55.2_{0.8}}$	$\mathbf{56.0_{2.7}}$	$\mathbf{47.7_{4.1}}$	$52.2_{2.7}/53.3_{3.0}$	$\mathbf{35.7_{1.5}}$	$\mathbf{66.9_{1.6}}$
4	**ICL**	$60.3_{9.8}$	$80.4_{6.6}$	$77.4_{14.6}$	$65.1_{5.4}$	$71.7_{6.5}$	$49.9_{9.9}$	$42.7_{3.9}$	$42.1_{3.2}$	$44.7_{5.9}/45.2_{6.0}$	$31.7_{4.8}$	$55.6_{7.0}$
	BBT	$84.5_{1.2}$	$\mathbf{89.8_{0.9}}$	$90.2_{0.6}$	$79.0_{2.1}$	$67.7_{3.5}$	$42.9_{0.6}$	$48.4_{4.0}$	$40.5_{1.3}$	$41.2_{1.7}/40.7_{2.0}$	$19.4_{1.5}$	$58.6_{1.8}$
	BBTv2	$86.6_{2.2}$	$89.4_{0.6}$	$90.3_{0.5}$	$79.1_{2.1}$	$89.0_{1.7}$	$46.0_{1.4}$	$46.2_{2.3}$	$40.8_{4.3}$	$44.0_{0.9}/44.8_{1.6}$	$31.9_{1.4}$	$62.6_{1.7}$
	RLPrompt	$80.7_{7.5}$	$75.8_{10.1}$	$78.8_{7.3}$	$76.1_{4.8}$	$76.3_{5.9}$	$45.0_{3.1}$	$53.5_{2.9}$	$36.3_{2.6}$	$44.4_{2.9}/45.5_{3.8}$	$16.7_{2.4}$	$57.4_{4.8}$
	PromptBoosting	$\mathbf{88.9_{2.3}}$	$83.0_{5.2}$	$92.3_{2.1}$	$78.2_{6.8}$	$\mathbf{90.1_{0.7}}$	$36.4_{5.1}$	$53.5_{5.9}$	$\mathbf{53.4_{3.4}}$	$39.8_{4.5}/40.3_{5.7}$	$40.9_{2.5}$	$63.4_{4.0}$
	DecT	$87.6_{1.6}$	$89.6_{0.9}$	$\mathbf{94.8_{0.7}}$	$\mathbf{81.9_{2.6}}$	$89.1_{0.6}$	$\mathbf{59.9_{2.1}}$	$\mathbf{56.7_{2.7}}$	$53.2_{2.9}$	$\mathbf{52.2_{2.3}}/\mathbf{53.4_{2.4}}$	$\mathbf{46.7_{1.7}}$	$\mathbf{69.5_{1.9}}$
16	**ICL**	$71.5_{15.8}$	$80.6_{6.0}$	$73.7_{14.5}$	$64.4_{6.0}$	$71.8_{9.1}$	$52.6_{5.7}$	$43.8_{7.0}$	$42.0_{6.3}$	$51.4_{3.0}/52.1_{3.3}$	$35.1_{2.6}$	$58.1_{7.2}$
	BBT	$89.6^{b}_{0.3}$	$89.3_{0.4}$	$91.5^{b}_{0.2}$	$81.5^{b}_{0.8}$	$87.8^{b}_{3.0}$	$48.3_{1.4}$	$52.6^{b}_{2.2}$	$46.6^{b}_{1.3}$	$40.0_{2.6}/39.9_{2.9}$	$17.8_{1.4}$	$62.3_{1.5}$
	BBTv2	$90.3^{a}_{1.7}$	$88.6_{2.1}$	$92.9^{a}_{0.6}$	$85.3^{a}_{0.5}$	$93.6^{a}_{0.7}$	$52.0_{1.4}$	$56.7^{a}_{3.3}$	$57.3^{a}_{2.3}$	$50.1_{2.4}/51.7_{3.2}$	$33.3_{1.0}$	$68.3_{1.7}$
	RLPrompt	$87.0_{2.6}$	$87.6_{2.4}$	$95.1^{c}_{1.0}$	$80.2^{c}_{0.7}$	$80.8_{3.3}$	$48.1_{2.2}$	$54.3_{2.8}$	$41.1_{5.0}$	$43.3_{3.9}/44.3_{4.5}$	$17.5_{1.4}$	$61.8_{2.7}$
	PromptBoosting	$87.6^{d}_{3.0}$	$86.2_{3.1}$	$94.7_{1.0}$	$85.2^{d}_{0.9}$	$\mathbf{95.0_{0.5}}$	$46.6_{2.4}$	$\mathbf{60.0^{d}_{5.5}}$	$\mathbf{61.3^{d}_{3.5}}$	$52.5^{d}_{1.5}/50.4_{5.1}$	$52.1_{2.6}$	$70.1_{2.6}$
	DecT	$\mathbf{91.0_{0.5}}$	$\mathbf{91.0_{0.9}}$	$\mathbf{95.4_{0.3}}$	$\mathbf{86.4_{0.4}}$	$94.6_{0.5}$	$\mathbf{64.2_{0.7}}$	$59.7_{1.8}$	$60.5_{0.8}$	$\mathbf{55.3_{1.3}}/\mathbf{56.8_{1.5}}$	$\mathbf{53.5_{1.8}}$	$\mathbf{73.5_{1.0}}$

Table 1: Experiment results for MaaS adaptation methods. Some baseline results are taken from corresponding papers ([a]Sun et al. (2022a), [b]Sun et al. (2022b), [c]Deng et al. (2022), [d]Hou et al. (2022)). We run other experiments over 5 random seeds and report average accuracy and standard deviation (%). Best results are in **bold**.

selection of λ, we directly set $\lambda = 1/n$ for most datasets based on the intuition that λ should decrease as the amount of training data increases. On MNLI and FewNERD, we tune λ on the validation set and select $\lambda = 1$ and $\lambda = 1/16$ respectively. We give the templates and label words in Appendix A.3.

4.2 Main Results

Table 1 presents the main few-shot learning results. From the results, we have these observations:

Overall, DecT outperforms the state-of-the-art baseline methods by a large margin (more than 3% on average), especially under extreme data scarcity, showing its superior performance. Across different tasks, DecT and baselines obtain similar results on some easy sentiment analysis and topic classification tasks, but we highlight that DecT is much more favorable on difficult datasets, such as Yahoo and FewNERD. While other baseline methods struggle to optimize well, DecT surpasses them significantly (about 10% on Yahoo and 20% on FewNERD under 16-shot setting compared with BBTv2 and ICL).

On stability, DecT also has consistently low variance and some baselines (ICL, RLPrompt and PromptBoosting) are unstable. Given the difficulty of few-shot PTM adaptation, it is of great significance that the adaptation method is robust to random seeds.

On baselines, optimization-free methods, i.e. zero-shot prompt and ICL are strong baselines. However, as shown in the table, ICL gives the best results in the 1-shot setting, and it can hardly improve with more training data due to the input length restriction. To compare, merely optimizing the input prompts (BBT and RLPrompt) can hardly outperform them, showing the limitation of input-side prompt optimization. In contrast, two other baselines, BBTv2 and PromptBoosting, are more powerful because they either inserts additional learnable prompt tokens inside the PTM or ensembles the outputs of different prompts. With the superior results of DecT, we argue that output-side optimization is a promising way for MaaS PTM adaptation.

Method	Tr. Time (s)	# Query	# Param. (K)
ICL	0	0	0
BBT	10,512	8,000	0.5
BBTv2	9,856	8,000	12
RLPrompt	65,579	12,000	3,100
PromptBoosting	644	10	0.4
DecT	3	1	130

Table 2: Efficiency comparison of MaaS adaptation methods. Training time is the average wall clock time measured in the 16-shot setting. "Tr." stands for Training and "Param." stands for Parameter.

4.3 Efficiency Comparison

Despite the superior performance, another major advantage of DecT is its high efficiency. In Fig-

n	Method	SST2	IMDB	Yelp	AG	DB	Yahoo	RTE	SNLI	MNLI-m/mm	NERD	Avg.
64	**Fine-tuning**†	$92.5_{1.9}$	$86.3_{3.8}$	$94.5_{1.4}$	$87.4_{0.6}$	$98.2_{0.2}$	$69.0_{0.7}$	$67.7_{3.2}$	$66.6_{6.4}$	$65.6_{2.9}/67.7_{4.0}$	$67.6_{0.8}$	$78.5_{2.4}$
	DecT	$92.4_{0.5}$	$91.3_{0.5}$	$94.9_{0.5}$	$89.2_{0.3}$	$97.0_{0.1}$	$69.3_{0.4}$	$65.7_{1.7}$	$67.2_{1.0}$	$62.0_{1.4}/63.3_{1.3}$	$56.1_{0.8}$	$77.1_{0.8}$
256	**Fine-tuning**†	$92.0_{0.9}$	$92.1_{0.2}$	$94.3_{0.3}$	$89.6_{0.3}$	$98.5_{0.2}$	$70.2_{0.4}$	$79.8_{1.0}$	$84.4_{0.4}$	$77.2_{0.2}/78.7_{0.3}$	$71.4_{0.5}$	$84.4_{0.4}$
	DecT	$92.7_{0.2}$	$92.1_{0.1}$	$95.6_{0.1}$	$90.3_{0.1}$	$97.4_{0.1}$	$71.3_{0.1}$	$69.2_{1.0}$	$69.7_{0.4}$	$68.0_{0.3}/69.4_{0.3}$	$56.2_{0.3}$	$79.3_{0.3}$

Table 3: Experiment results for more training data. We run all experiments over 5 random seeds and report the average accuracy and standard deviation (%). †: Update model parameters.

ure 1, we plot average accuracy versus training time for each method under different shots. We also provide detailed statistics of training time, query numbers, and parameter numbers for 16-shot experiments in Table 2.

From Figure 1 and Table 2, we clearly see that DecT can be optimized quickly and only requires one model query per training sample, **which is about $200\times$faster and queries $10\times$fewer than all prompt optimization methods**. For BBT, BBTv2, and RLPrompt, users have to query the model near 10^4 times and spend several hours for sufficient optimization even in the few-shot scenario. When the inference API is not for free such as OpenAI API [2], using these methods would be expensive, and this further burdens their usage in the scenarios of rich data and large models.

In terms of tunable parameters, DecT demands 130K additional parameters for the linear projection layer, which is less than 0.04% of RoBERTa$_{\text{LARGE}}$ (355M) that largely saves storage space. Although some other methods (BBT, BBTv2 and PromptBoosting) require fewer parameters, DecT is much easier to optimize.

4.4 Beyond Few-shot

As shown in Section 4.3, the simple architecture and high efficiency enable DecT to scale on more training data, while baseline methods struggle to finish training within acceptable time limits. To explore the scalability of DecT beyond the few-shot setting, we conduct experiments with increased training data ($n = 64$ and 256). For reference, we compare DecT with fine-tuning, the strongest baseline which update full model parameters.

The detailed results are presented in Figure 1 and Table 3 and we have the following conclusions. (1) DecT continually improves its performance on more training data at a low cost. The average accuracy gains 6% from 16-shot to 256-shot while the average training time is less than 100 seconds. (2) Compared with fine-tuning, DecT is even on par with it in the 64-shot scenario and gradually falls behind in the 256-shot setting, which is reasonable as we only tune a small portion of parameters outside the model. Through further task-level observation, we find DecT still performs well on sentiment analysis and topic classification, but cannot catch up with fine-tuning on NLI and entity typing, which are identified as harder tasks as they require complex reasoning or fine-grained semantic understanding. (3) In experiments, we find fine-tuning is more sensitive to random seeds in the few-shot setting due to the huge amount of trainable parameters and relatively few loss signals, which is evidenced by the high variance in the 64-shot setting. In such scenario, DecT has lower variances due to most parameters are frozen. Therefore, the stability advantage of DecT has been verified again.

To conclude, we take the first step to applying MaaS methods beyond few-shot learning. **The results show that DecT is competitive against fine-tuning on regular classification tasks, but is limited on difficult tasks.** How to adapt PTMs on challenging tasks without parameter updates still needs further exploration.

5 Analysis

In addition to main experiments, we further provide more analytical experiments for understanding DecT. We conduct ablation study on several components in Section 5.1. Then we evaluate the scaling effect (Section 5.2), the impact of hyperparameter λ (Section 5.3) and templates (Section 5.4) respectively. We further conduct transferability experiments in Appendix B.

5.1 Ablation Study

To validate each component of our proposed DecT, especially the effect of model scores $\mathbf{s}$, radius parameter r, and ProtoNet, we conduct extensive ablation studies. We present results in Table 4 and Figure 4.

[2] https://openai.com/api/

s	r	Average Accuracy		
		1	4	16
✗	✗	$54.0_{6.3}$	$66.3_{2.5}$	$73.0_{1.2}$
✓	✗	$64.8_{2.6}$	$69.3_{1.7}$	$\mathbf{73.5_{1.0}}$
✗	✓	$54.0_{6.2}$	$67.2_{2.0}$	$73.0_{1.1}$
✓	✓	$\mathbf{66.9_{1.6}}$	$\mathbf{69.5_{1.9}}$	$\mathbf{73.5_{1.0}}$

Table 4: Ablation study of model scores s and radius parameter r. We run each experiment over 5 random seeds and report average accuracy and standard deviation (%). Best results are in **bold**.

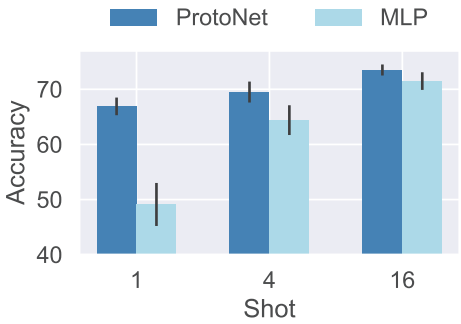

Figure 3: Comparison between ProtoNet and MLP. We report the average accuracy (%) and standard deviation.

Ablating model scores. Apparently, model scores contribute largely to the few-shot performance of DecT, especially when the training data is extremely scarce (1-shot), which illustrates that model scores contain beneficial prior model knowledge for language understanding. Also, incorporating training data reduces the variance. When there are more training data, model scores bring less enhancement, which is reasonable as the relative weights of model and ProtoNet scores change.

Ablating radius. Meanwhile, the radius is also helpful under low-shot scenarios, which characterizes the difference across classes. But as the number of training data increases, ProtoNet dominates model predictions and the impact of r is weakened as well.

Ablating decoder. As stated previously, the design choice of the decoder function is flexible. We replace ProtoNet with a two-layer MLP and evaluate the performance. In Figure 3 we can see that ProtoNet significantly outperforms MLP in the 1-shot setting, which matches the advantages of ProtoNet in the few-shot setting. In 4-shot and 16-shot experiments, ProtoNet still gets higher scores, but with smaller margins. On stability, ProtoNet achieves consistently lower standard deviation scores, which serve as another advantage. Overall, we find ProtoNet is a vital component in DecT, and simply replacing it with MLP would worsen the performance.

5.2 Model Scaling

In this section, we explore how DecT applies to PTMs with different architecture and scales.

We select T5 (Raffel et al., 2020), an encoder-decoder PTM, at different scales, from $\text{T5}_{\texttt{Small}}$, $\text{T5}_{\texttt{Base}}$, $\text{T5}_{\texttt{Large}}$ to $\text{T5}_{\texttt{3B}}$. Table 5 presents the full results of T5 models. First of all, DecT has been successfully deployed on T5, a generative language model, which verifies its transferability across PTMs. Furthermore, we can observe an apparent trend of the scaling effect where larger models consistently perform better.

We also evaluate the DecT on CPM-Bee[3], which is a bilingual generative pre-trained language model with 10B parameters. Table 6 presents the results of CPM-Bee in different settings. The results show that DecT strongly enhances the adaptation of large PLM on downstream tasks. Moreover, CPM-Bee achieves great performance on NLI tasks, which flags that DecT could deal with more difficult tasks with powerful backbone models.

5.3 Impact of λ

As a hyperparameter, λ controls the relative importance of model scores and prototype scores. Here we examine its impact on AG's News and SST2. In Figure 4, we can observe that: (1) λ largely affects DecT in the 1-shot settings. As λ increases, DecT gradually performs better and stabler, which illustrates the importance of model knowledge in this case. (2) With the shot number increases, the impact of varying λ is weakened, and the best practices become smaller. These observations validate our selection strategy in Section 4.1, which effectively balances model and data knowledge.

5.4 Impact of Templates

Although DecT is an output-side adaptation method, the choice of templates also affects the final performance. To assess the influence of templates, we conduct experiments on AG's News and SST2 and show results in Table 7. Overall, DecT does not rely much on templates. While different templates may induce fluctuant zero-shot perfor-

[3] https://live.openbmb.org/models/bee

Model	SST2	IMDB	Yelp	AG	DB	Yahoo	RTE	SNLI	MNLI-m/mm	NERD	Avg.
$\text{T5}_{\texttt{Small}}$	$73.4_{1.8}$	$68.8_{2.1}$	$79.5_{1.4}$	$79.1_{0.6}$	$76.8_{0.7}$	$57.5_{0.7}$	$51.9_{0.8}$	$38.7_{2.1}$	$38.6_{0.4}/39.0_{0.3}$	$35.1_{1.9}$	$58.0_{1.2}$
$\text{T5}_{\texttt{Base}}$	$83.8_{1.1}$	$86.3_{0.9}$	$91.5_{0.6}$	$84.3_{0.6}$	$93.5_{0.4}$	$61.1_{0.8}$	$54.0_{1.6}$	$44.9_{1.6}$	$47.8_{0.6}/49.4_{0.7}$	$50.2_{3.0}$	$67.9_{1.1}$
$\text{T5}_{\texttt{Large}}$	$92.3_{0.4}$	$92.0_{0.4}$	$94.4_{1.3}$	$85.5_{0.8}$	$94.9_{0.3}$	$63.6_{0.9}$	$55.9_{2.3}$	$49.5_{1.9}$	$49.7_{1.4}/50.8_{1.9}$	$53.2_{1.6}$	$71.1_{1.2}$
$\text{T5}_{\texttt{3B}}$	$89.9_{0.4}$	$92.7_{0.7}$	$94.9_{2.0}$	$87.7_{0.8}$	$96.2_{0.3}$	$66.5_{0.7}$	$55.8_{2.2}$	$52.0_{1.9}$	$52.8_{1.6}/52.2_{2.1}$	$51.9_{1.4}$	$72.1_{1.3}$

Table 5: Experiment results (16-shot) for our method on different versions of T5 (Raffel et al., 2020). We run each experiment over 5 random seeds and report average accuracy and standard deviation (%).

Shot	SST2	IMDB	Yelp	AG	DB	Yahoo	RTE	SNLI	MNLI-m/mm	NERD	Avg.
0	80.5	89.1	96.6	74.6	71.3	46.7	84.1	45.4	45.6/45.6	1.6	61.9
4	91.2	96.5	97.8	80.5	81.1	56.5	82.2	77.8	66.0/66.5	52.9	77.2
16	92.7	96.2	97.5	85.5	89.8	65.2	86.0	86.4	76.3/76.3	54.6	82.4
64	94.3	96.5	98.3	88.5	93.5	68.7	87.1	88.9	78.0/79.0	59.8	84.8
256	94.5	96.7	98.4	89.7	94.2	69.9	87.7	89.4	81.7/80.6	59.1	85.6

Table 6: Experiment results for our method on CPM-Bee. We run each experiment over 5 random seeds and report average accuracy (%).

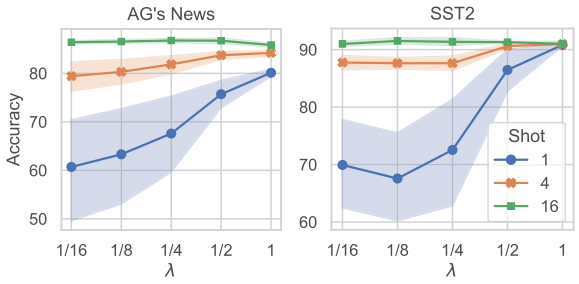

Figure 4: Accuracy (%) with 85% confidence interval on 5 runs for different λ on AG's News and SST2.

Template	Prompt	DecT
SST2		
x In summary, it was `[MASK]`.	83.3	91.0
x It was `[MASK]`.	73.3	88.4
x AgentMediaGrade Officials Grade `[MASK]`.^{c}	90.4	92.2
AG's News		
[Topic : `[MASK]`] x	80.9	86.4
[Category : `[MASK]`] x	78.6	86.8
`[MASK]` Alert Blog Dialogue Diary Accountability x ^{c}	78.8	86.0

Table 7: Accuracy of prompt (zero-shot) and DecT (16-shot) across different templates. Templates marked with ^{c} are taken from Deng et al. (2022).

mance, DecT largely moderates the gaps between them. Additionally, we try two templates searched from RLPrompt (Deng et al., 2022) and they both achieve satisfying results. On SST2, the template from RLPrompt is even better than manually designed ones. **Therefore, we highlight that DecT is complementary with input-side adaptation algorithms, and they can work together for better performance.**

6 Related Work

Our work explores how to efficiently adapt large PTMs. In this section, we review three lines of research for *prompt-based tuning* (data efficiency), *parameter-efficient tuning* (parameter efficiency), and MaaS adaptation methods respectively.

6.1 Prompt-based Tuning

Prompt-based tuning aims to bridge the gap between pre-training and downstream tasks for data-efficient model adaptation. The major practice for prompt-based tuning (Liu et al., 2021) is wrapping text pieces into human-designed templates. By this means, prompt-based tuning converts downstream tasks to pre-training tasks (e.g. masked language modeling) and greatly enhances the zero/few-shot learning ability of PTMs. Prompt-based tuning is first applied in knowledge probing (Trinh and Le, 2018; Petroni et al., 2019), and it has been adopted broadly in NLP (Schick and Schütze, 2021; Hu et al., 2022b; Ding et al., 2021a; Han et al., 2021; Cui et al., 2022). Despite manually designed prompts, other works also investigate automatic and learnable prompts (Shin et al., 2020; Gao et al., 2021; Hambardzumyan et al., 2021; Schick et al.,

2020; Lester et al., 2021) to alleviate the prompt engineering efforts. However, the optimization of prompts is a non-trivial problem (Ding et al., 2022c; Lester et al., 2021) which sometimes leads to more computation costs and suboptimal performance. Thus in our work, we adopt manual prompts to stimulate model knowledge and help data-efficient model adaptation.

6.2 Parameter-efficient Tuning

Another line of work explores tuning a small fraction of model parameters to reduce computation and storage budgets, namely parameter-efficient tuning (PET) (Ding et al., 2022c). Typical PET methods include inserting tunable modules (Houlsby et al., 2019; Li and Liang, 2021; Hu et al., 2022a), adding soft prompt tokens (Lester et al., 2021) and specifying certain parameters (Zaken et al., 2022). Although PET methods achieve remarkable performance with few parameter updates, they still require model gradients, which are unavailable in the MaaS setting.

6.3 MaaS Adaptation

With inference-only APIs, there are also works that adapt models without tuning any model parameters. Brown et al. (2020) present in-context learning, which concatenates test inputs with several exemplars. Further improvements focus on reliving the instability issues caused by model biases (Zhao et al., 2021; Han et al., 2022) and examplar orders (Lu et al., 2022). PromptBoosting (Hou et al., 2022) employs boosting algorithm for prompt ensembling, giving strong performance. Other approaches try to optimize prompts with either black-box optimization methods (Sun et al., 2022a,b) or reinforcement learning (Deng et al., 2022). However, due to the lack of gradient signals, they need thousands of model queries, resulting in high costs when the model is large and API calls are not for free. Different from the abovementioned methods, we adapt models at the output side, which need not optimize the distant input prompts. We demand only one API call for each training sample and achieve better results across tasks.

7 Conclusion

In this paper, we present DecT, which performs both data and parameter-efficient adaptation with off-shelf PTMs. By fusing prior model knowledge and posterior data knowledge, DecT achieves superior performance on ten language understanding tasks. Meanwhile, DecT exceeds existing baselines by three orders of magnitude in terms of training time and the number of queries, highlighting its advantages in real-world deployment. In future works, we are eager to explore how to combine input and output-side adaptation methods for better PTM adaptation, and how to extend this line of research to more challenging scenarios.

Limitation

DecT explores how to adapt black-box PTMs on downstream tasks. As we show in Section 4.4, our method is not comparable to fine-tuning on hard tasks with increased data points. Moreover, we only focus on classification tasks in this work and do not testify DecT on free-form generation tasks. In the future, we will work toward more general MaaS adaptation strategies across tasks.

Ethical Statement

As large language models are getting more and more popular in NLP research and application, DecT provides a cost-efficient way to adapt these large models. However, we need also to be cautious about the improper adaptation of large language models, such as generating toxic and biased speeches.

Acknowledgements

This work is supported by the National Key R&D Program of China (No.2022ZD0116312), Institute Guo Qiang at Tsinghua University and sponsored by Tsinghua-Toyota Joint Research Fund.

Ganqu Cui made the original research proposal and wrote the paper. Ganqu Cui and Wentao Li conducted experiments. Ning Ding revised the paper and participated in the discussion. Longtao Huang, Zhiyuan Liu and Maosong Sun advised the project.

References

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. In *TAC*.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of EMNLP*.

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*.

Ganqu Cui, Shengding Hu, Ning Ding, Longtao Huang, and Zhiyuan Liu. 2022. Prototypical verbalizer for prompt-based few-shot tuning. In *Proceedings of ACL*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Machine learning challenges workshop*.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P Xing, and Zhiting Hu. 2022. Rlprompt: Optimizing discrete text prompts with reinforcement learning. In *Proceedings of EMNLP*.

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*.

Ning Ding, Yulin Chen, Ganqu Cui, Xiaobin Wang, Hai-Tao Zheng, Zhiyuan Liu, and Pengjun Xie. 2022a. Few-shot classification with hypersphere modeling of prototypes. *arXiv preprint arXiv:2211.05319*.

Ning Ding, Yulin Chen, Xu Han, Guangwei Xu, Pengjun Xie, Hai-Tao Zheng, Zhiyuan Liu, Juanzi Li, and Hong-Gee Kim. 2021a. Prompt-learning for fine-grained entity typing. *arXiv preprint arXiv:2108.10604*.

Ning Ding, Shengding Hu, Weilin Zhao, Yulin Chen, Zhiyuan Liu, Hai-Tao Zheng, and Maosong Sun. 2022b. Openprompt: An open-source framework for prompt-learning. In *Proceedings of ACL*.

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, Zhiyuan Liu, Hai-Tao Zheng, Jianfei Chen, Yang Liu, Jie Tang, Juanzi Li, and Maosong Sun. 2022c. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. In *arXiv*.

Ning Ding, Guangwei Xu, Yulin Chen, Xiaobin Wang, Xu Han, Pengjun Xie, Haitao Zheng, and Zhiyuan Liu. 2021b. Few-nerd: A few-shot named entity recognition dataset. In *Proceedings of ACL*, pages 3198–3213.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making pre-trained language models better few-shot learners. In *Proceedings of ACL*, pages 3816–3830.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and William B Dolan. 2007. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*.

R Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second pascal recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, volume 7.

Karen Hambardzumyan, Hrant Khachatrian, and Jonathan May. 2021. WARP: word-level adversarial reprogramming. In *Proceedings of ACL*, pages 4921–4933.

Xu Han, Weilin Zhao, Ning Ding, Zhiyuan Liu, and Maosong Sun. 2021. Ptr: Prompt tuning with rules for text classification. *arXiv preprint arXiv:2105.11259*.

Zhixiong Han, Yaru Hao, Li Dong, Yutao Sun, and Furu Wei. 2022. Prototypical calibration for few-shot learning of language models. *arXiv preprint arXiv:2205.10183*.

Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Bairu Hou, Joe O'Connor, Jacob Andreas, Shiyu Chang, and Yang Zhang. 2022. Promptboosting: Black-box text classification with ten forward passes. *arXiv preprint arXiv:2212.09257*.

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*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022a. Lora: Low-rank adaptation of large language models. In *Proceedings of ICLR*.

Shengding Hu, Ning Ding, Huadong Wang, Zhiyuan Liu, Juanzi Li, and Maosong Sun. 2022b. Knowledgeable prompt-tuning: Incorporating knowledge into prompt verbalizer for text classification. In *Proceedings of ACL*.

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

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef,

Sören Auer, et al. 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*.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. 2021. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*.

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

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. *ACM Computing Surveys*.

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*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *Proceedings of ICLR*.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2022. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of ACL*.

Andrew 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*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of NeurIPS*, pages 8024–8035.

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*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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

Timo Schick, Helmut Schmid, and Hinrich Schütze. 2020. Automatically identifying words that can serve as labels for few-shot text classification. In *Proceedings of COLING*.

Timo Schick and Hinrich Schütze. 2021. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of EACL*.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of EMNLP*.

Jake Snell, Kevin Swersky, and Richard S. Zemel. 2017. Prototypical networks for few-shot learning. In *Proceedings of NIPS*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*.

Tianxiang Sun, Zhengfu He, Hong Qian, Yunhua Zhou, Xuanjing Huang, and Xipeng Qiu. 2022a. BBTv2: Towards a gradient-free future with large language models. In *Proceedings of EMNLP*.

Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. 2022b. Black-box tuning for language-model-as-a-service. In *Proceedings of ICML*.

Trieu H Trinh and Quoc V Le. 2018. A simple method for commonsense reasoning. *arXiv preprint arXiv:1806.02847*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of ICLR*.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of NAACL-HLT*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of EMNLP*, pages 38–45, Online.

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*.

Tianyi Zhang, Felix Wu, Arzoo Katiyar, Kilian Q Weinberger, and Yoav Artzi. 2021. Revisiting few-sample bert fine-tuning. In *Proceedings of ICLR*.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Proceedings of NIPS*.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. In *Proceedings of ICML*.

A Experiment Details

A.1 Dataset Statistics

We provide dataset statistics in Table 8. We obtain AG's News, Yahoo, and Yelp P. from https://github.com/zhangxiangxiao/Crepe under the BSD-3-Clause license. We get FewNERD from https://ningding97.github.io/fewnerd/ with CC BY-SA 4.0 license. Other datasets are downloaded using Huggingface Datasets (Lhoest et al., 2021).

Task	Dataset	# Class	# Test
Sentiment Analysis	SST2	2	872
	Yelp	2	38,000
	IMDB	2	25,000
Topic Classification	AG's News	4	7,600
	Yahoo	10	60,000
	DBPedia	14	70,000
NLI	RTE	2	277
	SNLI	3	9,842
	MNLI-m/mm	3	9,815/9,832
Entity Typing	FewNERD	66	96,853

Table 8: The dataset information of our experiments. # Class column represents the number of classes, # Test column represents the number of examples for testing.

A.2 Baselines

In-context Learning (ICL). To guarantee meaningful results, we randomly permute the demonstrations and prepend them before input prompts. Due to the input length limit, we truncate the demonstrations which exceed input length.

BBT and BBTv2. We reproduce BBT and BBTv2 using the official codes[4]. For datasets adopted in their work, we follow the original implementations including templates, label words, and hyperparameters. For other datasets, we reproduce with our templates, label words, and default hyperparameters. We take existing 16-shot experiment results from the paper and run other experiments with 5 random seeds for a fair comparison.

RLPrompt. We also use the official codes[5] for reproduction and take some experiment results from their original paper. It is worth noting that RLPrompt adopts the test set of SST2 and we use the validation set, so we report the reproduced results on the SST2 validation set.

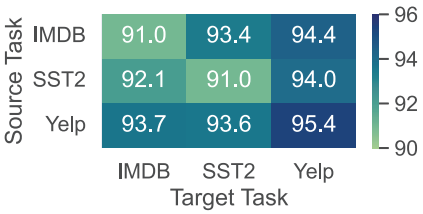

Figure 5: Accuracy heatmap of DecT transferred across different sentiment analysis tasks under 16-shot setting. The backbone model is RoBERTa$_{\text{LARGE}}$

PromptBoosting. We follow the official codes[6] for reproduction. Since the number of additional parameters is related to number of classes, we compute the average numbers across datasets.

Fine-tuning. We adopt prompt-based fine-tuning which uses the same templates and label words with DecT. We tune the whole model for 5 epochs with AdamW optimizer (Loshchilov and Hutter, 2019) using a 3×10^{-5} learning rate.

A.3 Templates and Label Words

We report the used prompt templates and label words in Table 9. Most of them are taken from OpenPrompt (Ding et al., 2022b).

B Transferability

We conduct transferability experiments on sentiment analysis and present the results in Figure 5. We see that DecT is highly transferable across datasets. On SST2 and IMDB, DecT trained on other datasets even surpasses the original performance. More surprisingly, we find DecT trained on Yelp, a restaurant review dataset, performs best on SST2 and IMDB, which are two movie review datasets. This result further shows the great domain generalization ability of DecT.

[4] https://github.com/txsun1997/Black-Box-Tuning

[5] https://github.com/mingkaid/rl-prompt

[6] https://github.com/UCSB-NLP-Chang/PromptBoosting

Dataset	Template	Label Words
SST2 Yelp IMDB	x In summary, it was `[MASK]`.	bad, great
AG's News	[Topic : `[MASK]`] x	politics, sports, business, technology
Yahoo	[Topic : `[MASK]`] x	society, science, health, education, computers, sports, business, entertainment, family, politics
DBPedia	x_1 x_2 The category of x_1 is `[MASK]`.	company, school, artist, athlete, politics, transportation, building, river, village, animal, plant, album, film, book
RTE SNLI MNLI-m/mm	x_1 ? `[MASK]`, x_2	No, Yes No, Maybe, Yes No, Maybe, Yes
FewNERD	x `[ENT]` is `[MASK]`.	actor, director, artist, athlete, politician, scholar, soldier, person, show, religion, company, team, school, government, media, party, sports, organization, geopolitical, road, water, park, mountain, island, location, software, food, game, ship, train, plane, car, weapon, product, theater, facility, airport, hospital, library, hotel, restaurant, building, championships, attack, disaster, election, protest, event, music, written, film, painting, broadcast, art, biology, chemical, living, astronomy, god, law, award, disease, medical, language, currency, educational

Table 9: The templates and label words used in our experiments. For each dataset, x, x_1, and x_2 represent the input sentences or sentence pairs. `[ENT]` copies the entity mentioned in the input sentence.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
In the last section

☐ 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?

In experiments we use datasets

☑ B1. Did you cite the creators of artifacts you used?
Section 4.1

☑ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Appendix A.1

☑ 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)?
Section 4.1

☐ 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?
Not applicable. Left blank.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Not applicable. Left blank.

☑ 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.
Appendix A.1

C ☑ Did you run computational experiments?

Section 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?
Section 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?
Section 4.1

☑ 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?
Section 4

☑ 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.)?
Section 4.1

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.