Inducer-tuning: Connecting Prefix-tuning and Adapter-tuning

Yifan Chen[1*] **Devamanyu Hazarika**[2*] **Mahdi Namazifar**[2]
Yang Liu[2] **Di Jin**[2†] **Dilek Hakkani-Tur**[2]
[1]University of Illinois Urbana-Champaign [2]Amazon Alexa AI

Abstract

Prefix-tuning, or more generally continuous prompt tuning, has become an essential paradigm of parameter-efficient transfer learning. Using a large pre-trained language model (PLM), prefix-tuning can obtain strong performance by training only a small portion of parameters. In this paper, we propose to understand and further develop prefix-tuning through the kernel lens. Specifically, we make an analogy between *prefixes* and *inducing variables* in kernel methods and hypothesize that *prefixes* serving as *inducing variables* would improve their overall mechanism. From the kernel estimator perspective, we suggest a new variant of prefix-tuning—*inducer-tuning*, which shares the exact mechanism as prefix-tuning while leveraging the residual form found in adapter-tuning. This mitigates the initialization issue in prefix-tuning. Through comprehensive empirical experiments on natural language understanding and generation tasks, we demonstrate that inducer-tuning can close the performance gap between prefix-tuning and fine-tuning.

1 Introduction

Transfer learning from large pre-trained language models (PLMs) has been the de-facto method to tackle downstream natural language processing (NLP) tasks with proven performance and scalability (Peters et al., 2018). Among all the adaption techniques, fine-tuning (Howard and Ruder, 2018; Kale and Rastogi, 2020) is predominant for PLMs and maintains the models' architecture while updating all the parameters within. Though powerful, fine-tuning is considered parameter-inefficient since it results in separate copies of model parameters for each task/client after training.

With the sizes of PLMs increasing to hundreds of millions (Brown et al., 2020) or even up to trillion (Fedus et al., 2021) parameters, the trend motivates a range of parameter-efficient adaptation techniques, including *adapter-tuning* and *prompting*, as promising lightweight alternatives to fine-tuning to reduce computational consumption and storage space. Adapter-tuning inserts bottlenecked Multi-layer Perception (MLP) modules between the pre-trained layers of PLMs and tunes only these new parameters for task adaptation (Houlsby et al., 2019; Pfeiffer et al., 2020a). Prompting, instead, aims to adapt the general-purpose PLMs through prompts, whose effectiveness has been shown on a frozen GPT-3 model (Brown et al., 2020).

An implicit drawback of the prompt-based adaptation is the difficulty of searching for the proper prompt. To avoid manually designing the prompts, Shin et al. (2020) propose a search algorithm to find the effective prompt over discrete space of vocabularies; prefix-tuning (Li and Liang, 2021) and other concurrent methods (Lester et al., 2021; Liu et al., 2021b,a) further extend the discrete search to continuous prompts, attaining performance close to fine-tuning in some tasks. Despite the effort, there is still a performance gap between "prefix-tuning" and "fine-tuning" in many tasks, especially when the model size is small (Lester et al., 2021; He et al., 2021a). In addition, the mechanism of prefix-tuning is still poorly understood and under-explored. Prefix-tuning is also similar to adapter-tuning, since they both insert additional modules into each transformer layer (classical prompt-based methods (Lester et al., 2021; Liu et al., 2021b) only add prompts to the embedding layer).

Scrutinizing the evolution of prompt-based methods, we can observe they have gradually deviated from the concept of "prompts". Compared to the manually designed prompts, the discrete search usually results in counter-intuitive prompt tokens, which vaguely match the topic but are not as sensible as the manual one; for continuous prompt tuning, it even breaks the limit of the existing vo-

[*]Equal contribution. This work was performed while the first author was interning at Amazon Alexa AI.

[†]Correspondence to: Di Jin <djinamzn@amazon.com>

cabulary. All these pieces imply that the mechanism behind prompt-based tuning might be more complicated than guiding the output through hint prompts. To open the black box of "prompts", in this work, we propose to consider the prompts (either hard or soft) as "inducing variables" in kernel methods (Titsias, 2009). This analogy is justified due to the close connection between attention modules in PLMs and kernel estimators (Choromanski et al., 2020; Chen et al., 2021; Tsai et al., 2019). This kernel perspective explains the potential mechanism of prefix-tuning and motivates a new method, *inducer-tuning*. Specifically, inducer-tuning freezes all the original parameters in the PLMs as other prompt-based methods; when computing the attention output for a certain input token in each layer, inducer-tuning utilizes a point close to the query vector as the "inducer". This unique "soft prompt" eases the search for appropriate prompts and builds a new connection between "prompting" and "adapter-tuning".

In summary, the contribution of this work is three-fold: ① We explain the underlying mechanism of prefix-tuning as the inducing variables in kernel learning. ② We propose a new parameter-efficient adaptation technique, inducer-tuning, to further improve prefix-tuning. ③ Through comprehensive empirical studies, we verify our proposed method can close the gap between "prefix-tuning" and "fine-tuning" on relatively small PLMs, and provide a tighter lower bound on the potential of continuous prompt tuning.

2 Related Work

In this section, we briefly introduce the classical form of adapter-tuning and mainly focus on the different variants of prompting.

Adapter-tuning. Compared to fine-tuning all the parameters in the PLMs, Houlsby et al. (2019), Pfeiffer et al. (2020a) propose to modulate the output of a transformer layer through inserting additional small-bottleneck MLP layers (adapters) (Houlsby et al., 2019)[1]:

$$\text{Adapter}(\boldsymbol{h}) = \boldsymbol{h} + \text{ReLU}(\boldsymbol{h}\boldsymbol{W}_1)\boldsymbol{W}_2, \quad (1)$$

where $\boldsymbol{h}$ is the dimension-d hidden state in the transformer and $\boldsymbol{W}_1, \boldsymbol{W}_2$ are d-by-r and r-by-d projection matrices. Adapters have a residual form similar to skip connection, while only $\boldsymbol{W}_1, \boldsymbol{W}_2$ will be trained, greatly decreasing the size of tunable parameters. Up to now, the adapter-based method has been widely used for multiple NLP tasks (Stickland and Murray, 2019; Pfeiffer et al., 2020a; Wang et al., 2020; Pfeiffer et al., 2020b; Üstün et al., 2020; Vidoni et al., 2020; Pfeiffer et al., 2021; He et al., 2021b; Xu et al., 2021; Rücklé et al., 2020; Karimi Mahabadi et al., 2021), and adapters are also intrinsically connected to many other parameter-efficient adaptation techniques, as detailed in He et al. (2021a).

Prompting. Prompting prepends task-specific instructions to the task input and was originally demonstrated in Brown et al. (2020). As manual prompts rely on trial and error, Jiang et al. (2020), Shin et al. (2020) suggests search algorithms to specify the prompts among all the tokens in the vocabulary. Prompt-tuning (Lester et al., 2021) and P-tuning (Liu et al., 2021b) remove the vocabulary restriction on prompts by using trainable "soft prompts". The prompts in the aforementioned methods are only inserted into the bottom embedding layer of PLMs, while Prefix-tuning (Li and Liang, 2021; Liu et al., 2021a) adds soft prompts to all the transformer layers to further increase the capacity of prompting.

Though effective, proper initialization of the soft prompts remains challenging. To mitigate the issue, Li and Liang (2021) used an extra MLP to re-parameterize the prompts in each layer, thus adding more parameters that need training; SPoT (Vu et al., 2021) suggests performing pre-training for soft prompts using a wide range of NLP tasks, which requires additional computational resources. In contrast, though adapters have a similar expression form to prefix-tuning (He et al., 2021a), adapter-tuning only requires regular initialization. We speculate that the residual form of adapters mitigates the initialization issue since the output of each layer in the new model would be centered around the output in the frozen PLMs, and the residual form contributes to gradient back-propagation as in skip connection. We rely on this intuition and utilize the above-mentioned advantages of adapters to guide the design of our proposed inducer-tuning.

3 Preliminaries: Transformer Layers

Before discussing the mechanism of prompt-tuning, we introduce the structure of transformer layers and necessary notations in this section.

A general transformer-based PLM is mainly

[1] We ignored layer normalization and bias terms here for brevity.

composed of L stacked layers. Each layer contains a multi-headed self-attention and a fully connected feed-forward network (FFN) sub-layer, both followed by an "Add & Norm" module (Vaswani et al., 2017). [2] Hereon, we shall focus on the structure of the attention sub-layer since prefix-tuning directly works on this sub-layer.

Passing a length-n input sequence $\boldsymbol{X} \in \mathbb{R}^{n \times N_h p}$ to an attention sub-layer (assuming N_h heads and dimension size p for each head), we first perform linear transforms to the input $\boldsymbol{X}$ and obtain the query matrix ($\boldsymbol{Q}$), the key matrix ($\boldsymbol{K}$), and the value matrix ($\boldsymbol{V}$) as:

$$\boldsymbol{Q}/\boldsymbol{K}/\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_{[q/k/v]} + \mathbf{1}\boldsymbol{b}_{[q/k/v]}^T, \quad (2)$$

where $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{n \times N_h p}$ are the query/ key/ value matrix; $\boldsymbol{W}_{[q/k/v]} \in \mathbb{R}^{N_h p \times N_h p}$ are the weight matrices, and $\boldsymbol{b}_{[q/k/v]} \in \mathbb{R}^{N_h p}$ are the bias terms in the corresponding transformations. [3]

To increase the model capacity, the three components $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ are respectively divided into N_h blocks, contributing to the attention output in each head of the multi-headed self-attention module. For instance, we represent $\boldsymbol{Q}$ as $\boldsymbol{Q} = \left(\boldsymbol{Q}^{(1)}, \cdots, \boldsymbol{Q}^{(N_h)}\right)$, where each block $\boldsymbol{Q}^{(h)} = \boldsymbol{X}\boldsymbol{W}_q^{(h)} + \mathbf{1}(\boldsymbol{b}_q^{(h)})^T$ is an n-by-p matrix, and $\boldsymbol{W}_q^{(h)}, \boldsymbol{b}_q^{(h)}$ are the corresponding parts in $\boldsymbol{W}_q, \boldsymbol{b}_q$. The attention output for the h^{th} head is:

$$\begin{aligned} \boldsymbol{L}^{(h)}\boldsymbol{V}^{(h)} &:= \text{softmax}(\boldsymbol{Q}^{(h)}(\boldsymbol{K}^{(h)})^T/\sqrt{p})\boldsymbol{V}^{(h)} \\ &= (\boldsymbol{D}^{(h)})^{-1}\boldsymbol{M}^{(h)}\boldsymbol{V}^{(h)}, \end{aligned} \quad (3)$$

where $\boldsymbol{M}^{(h)} := \exp\left(\boldsymbol{Q}^{(h)}(\boldsymbol{K}^{(h)})^T/\sqrt{p}\right)$ and $\boldsymbol{D}^{(h)}$ is a diagonal matrix in which $\boldsymbol{D}_{ii}^{(h)}$ is the sum of the i-th row in $\boldsymbol{M}^{(h)}$, serving as the normalization procedure in softmax. The attention outputs in each head are then concatenated as $\boldsymbol{L} := (\boldsymbol{L}^{(1)}\boldsymbol{V}^{(1)}, \ldots, \boldsymbol{L}^{(N_h)}\boldsymbol{V}^{(N_h)})$.

After concatenating the heads, there is a linear transform following the output

$$\boldsymbol{L}\boldsymbol{W}_o + \mathbf{1}\boldsymbol{b}_o^T, \quad (4)$$

where $\boldsymbol{W}_o$ and $\boldsymbol{b}_o$ are similarly sized as the other matrices in Equation (2). This is the overall output of the attention sub-layer, which we shall revisit in § 4.4.

[2] For simplicity, we omit the cross-attention module in transformer-based encoder-decoder models.

[3] To ease the notations we adopt the practical setting where $\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ have the same shape.

4 Parameter-Efficient Inducer-Tuning

We describe the motivation and the mechanism of inducer-tuning in this section. We first revisit the connection between self-attention and kernel estimators in § 4.1, which interprets attention from another perspective by considering query, key, and value matrices as three separate sets of vectors rather than the related representations of the same input sequence. This special perspective motivates and justifies the inducer-tuning we propose in § 4.3.

4.1 Attention as Kernel Estimators

Traditionally, attention operation (Equation (3)) is viewed as a transformation $g(\cdot)$ of the input sequence $\boldsymbol{X}$. However, in prefix-tuning, parameters within PLMs are frozen, which implies that given the input $\boldsymbol{X}$, the represenattions $\boldsymbol{Q}, \boldsymbol{K}$, and $\boldsymbol{V}$ are invariant.[4] This observation allows us to reinterpret attention as a kernel estimator $f(\cdot)$ with $\boldsymbol{Q}$ as its input. Specifically, we denote the i-th input vector $\boldsymbol{X}_i$'s attention operation as $f(\boldsymbol{Q}_i) := g(\boldsymbol{X}_i)$. This attention representation can be seen as modifying the input query vector $\boldsymbol{Q}_i$ to $f(\boldsymbol{Q}_i)$ via *supporting points* $\{\boldsymbol{K}_j\}_{j=1}^n$ (Choromanski et al., 2020; Peng et al., 2020; Chen et al., 2021), which can be considered as a Nadaraya–Watson kernel estimator (Wasserman, 2006, Definition 5.39):

$$\text{row-normalize}\left(\kappa\left(\boldsymbol{Q}, \boldsymbol{K}\right)\right)\boldsymbol{V},$$

where $\kappa(\cdot, \cdot)$ is a kernel function. (Refer to Appendix C for more details on this claim.)

4.2 Prefix-Tuning and Inducing Variables

Prefix-tuning (Li and Liang, 2021) alters the attention output in each layer. Concretely, it prepends length-l prefix vectors $\boldsymbol{P}_k, \boldsymbol{P}_v \in \mathbb{R}^{l \times p}$ to $\boldsymbol{K}$ and $\boldsymbol{V}$, respectively; for a certain query token $\boldsymbol{Q}_i$ (the i-th row of the query matrix $\boldsymbol{Q}$), its attention output $f(\boldsymbol{Q}_i) := \text{Attn}(\boldsymbol{Q}_i, \boldsymbol{K}, \boldsymbol{V})$ is updated as a weighted sum of $f(\boldsymbol{Q}_i)$ and $\text{Attn}(\boldsymbol{Q}_i, \boldsymbol{P}_k, \boldsymbol{P}_v)$ (He et al., 2021a, Equation (7)).

Remark. From the kernel estimator perspective, the two categories of virtual tokens play different roles. The virtual key vectors $\boldsymbol{P}_k$ apply to the empirical kernel matrix part and can alter the attention scores (and thus the weights for $\text{Attn}(\boldsymbol{Q}_i, \boldsymbol{P}_k, \boldsymbol{P}_v)$); whereas $\boldsymbol{P}_v$ takes effect in the value part. It might not be optimal for prefix-tuning to model the two

[4] While our discussion is for a single attention head, we omit the superscript (h) for brevity.

categories of virtual tokens similarly. In § 4.3 we will show how *inducer-tuning* addresses the two parts through different residual forms.

We suggest that the mechanism of prefix-tuning can be further understood through the concept of *inducing variables* in kernel learning literature (Titsias, 2009). Many computational methods in kernel learning utilize a small set of support points (inducing variables) to improve the inference performance (Musco and Musco, 2017; Chen and Yang, 2021). Snelson and Ghahramani (2005) specifically consider the inducing variables as auxiliary pseudo-inputs and infer them using continuous optimization, which is similar to prefix-tuning. We emphasize that from the first sight the main character of inducing-point methods is representing a vast amount of training examples through a small number of points, so as to reduce the computational cost; however, here we instead aim to leverage the mechanism of inducing variables to well-steer the estimation: the goal we try to attain is to strengthen prefix-tuning by making the prefixes better modulate the attention output. We introduce and analyze the mechanism as follows.

Mechanism for well-steering inference outputs in inducing-point methods. Conceptually, inducing variables help the inference because they can represent the distribution of the query inputs and steer the kernel methods without changing the kernel in use. In particular, we consider the distribution pattern of unconstrained inducing points X_M (Snelson and Ghahramani, 2005, Figure 1). We observe that most of them are *close to the testing examples* X^*, and in the new estimation (Snelson and Ghahramani, 2005, Equation (8)) the inducers X_M will receive great weights through the weights assignment mechanism in kernel methods (we recall kernel methods can assign the weights of samples as attention (Choromanski et al., 2020; Chen et al., 2021; Tsai et al., 2019); for inducing variables close to the query, they would automatically receive more attention), and thus effectively modulate the output.

From this mechanism, we draw an inductive bias "the prefix should be close to the query" (which is not enforced in the method of prefix-tuning) and accordingly propose inducer-tuning. We remark since we are not pursuing the original goal, reducing computational cost, of inducing variables, it is ordinary that the concrete design in the next subsection is different from the usual form of inducing points, a small number of samples.

We speculate prefix-tuning partially benefits from the above mechanism as well. Furthermore, some indirect evidence is stated as follows. As discussed in previous studies, to make the full potential of prompting, the manually designed prompts are expected to be related to the topic of the input sequence (Brown et al., 2020) (close to the query); even for the soft prompts they are recommended to be initialized with the token relevant to the specific tasks (Li and Liang, 2021), which also requires the prompts to be close to the query to provide effective adaptation. With this belief, we propose *inducer-tuning* to exploit further the mechanism of inducing variables and improve upon prefix-tuning.

4.3 Method

Inducer-tuning follows the same design principle as prefix-tuning, which modulates the attention output through inserting virtual tokens (vectors). However, unlike prefix-tuning, our virtual tokens are not shared among the input sequences. Inducer-tuning also incorporates the benefits of residual forms to ease the initialization and remove the reparametrization trick in prefix-tuning. Specifically, we suggest the following modifications: ① The "inducers" are adaptive to and customized for each input token to strengthen the expressiveness of the new attention output. ② We propose to model the virtual vectors in a residual form as an adapter, which makes the final attention output be in a residual form as well. We now dive into discussing the intuitions behind the modifications in detail.

Adaptive *inducers*. There is an important difference between language models and kernel methods, making fixed prefixes less effective than inducing variables in kernel methods. In language models, the distribution of the input queries keeps changing, and for some inputs, the fixed prefixes fail to be qualified as "inducing variables". Even worse, for a long input, there probably exists some query vectors away (regarding ℓ_2 distance) from all the virtual vectors in the fixed prefixes, which are thus unable to modulate the attention output well. The phenomenon that prefix-tuning has a relatively poorer performance on tasks with longer inputs can be observed in our experiments (§ 6).

To alleviate the above issue, we propose adaptive modeling of the virtual key vectors. For a query $\boldsymbol{Q}_i$, we suggest taking a vector close to $\boldsymbol{Q}_i$ itself as the corresponding virtual key vector (the length of

the new prefix is thus 1), in the hope of leading to better inference.

As for the virtual value vectors, we relate them to the corresponding virtual key vectors. The motivation comes from traditional (non-self-)attention, whose mechanism coincides with a kernel estimator: the value $\boldsymbol{V}$ is independent of the query sequence $\boldsymbol{Q}$ and related to the supporting points $\boldsymbol{K}$. Specifically, considering our design above that the virtual key vectors are close to $\boldsymbol{Q}_i$ (we take the virtual key vectors as transforms of the input query vectors $\boldsymbol{Q}_i$'s), we propose to accordingly model the virtual value vectors as a map of $\boldsymbol{Q}_i$ as well, which implies the virtual value vectors are also adaptive to the input query vectors.

Adapter Structures. To stabilize the training procedure, we propose incorporating the adapter structures into modeling the virtual key/value vectors. Specifically, for the i-th token $\boldsymbol{Q}_i$ (in a certain head), we represent the corresponding virtual key/value vectors respectively as

$$\boldsymbol{P}_{k,i} = \boldsymbol{Q}_i + \text{MLP}_k(\boldsymbol{Q}_i) \tag{5}$$

$$\boldsymbol{P}_{v,i} = f(\boldsymbol{Q}_i) + \text{MLP}_v(\boldsymbol{Q}_i), \tag{6}$$

where $\text{MLP}_{k/v}$ will both return a vector of the same dimension as the input $\boldsymbol{Q}_i$. [5]

It is natural to model $\boldsymbol{P}_{k,i}$ in a residual form as in Equation (1), considering $\boldsymbol{P}_{k,i}$ is expected to center around $\boldsymbol{Q}_i$; as for $\boldsymbol{P}_{v,i}$, we claim the specific form in Equation (6) allows the complete expression of inducer-tuning to be adapter-like, and the justification is stated as the following derivation.

To derive the expression for inducer-tuning, we denote the new key matrix and value matrix (specific to the input query vector $\boldsymbol{Q}_i$) as

$$\widetilde{\boldsymbol{K}}^{(i)} = \begin{bmatrix} \boldsymbol{P}_{k,i}^T \\ \boldsymbol{K} \end{bmatrix}, \quad \widetilde{\boldsymbol{V}}^{(i)} = \begin{bmatrix} \boldsymbol{P}_{v,i}^T \\ \boldsymbol{V}^T \end{bmatrix}.$$

The new attention output $\tilde{f}(\boldsymbol{Q}_i)$ for the query $\boldsymbol{Q}_i$ is thus (omitting the factor $1/\sqrt{p}$ for clarity)

$$\begin{aligned} &\text{Attn}(\boldsymbol{Q}_i, \widetilde{\boldsymbol{K}}^{(i)}, \widetilde{\boldsymbol{V}}^{(i)}) \\ =&\frac{\exp(\langle \boldsymbol{Q}_i, \boldsymbol{P}_{k,i} \rangle) \boldsymbol{P}_{v,i} + \sum_j \exp(\langle \boldsymbol{Q}_i, \boldsymbol{K}_j \rangle) \boldsymbol{V}_j}{\exp(\langle \boldsymbol{Q}_i, \boldsymbol{P}_{k,i} \rangle) + \sum_j \exp(\langle \boldsymbol{Q}_i, \boldsymbol{K}_j \rangle)} \\ =&\lambda_i \boldsymbol{P}_{v,i} + (1 - \lambda_i) f(\boldsymbol{Q}_i) \end{aligned} \tag{7}$$

[5] Note that these virtual vectors can be applied to causal attention in auto-regressive decoders since they do not utilize any future token information.

where we define the weight λ_i as,

$$\frac{\exp(\langle \boldsymbol{Q}_i, \boldsymbol{P}_{k,i} \rangle)}{\exp(\langle \boldsymbol{Q}_i, \boldsymbol{P}_{k,i} \rangle) + \sum_j \exp(\langle \boldsymbol{Q}_i, \boldsymbol{K}_j \rangle)}.$$

Combining the pieces, we state the complete equation for the new attention output $\tilde{f}(\boldsymbol{Q}_i)$ as,

$$\begin{aligned} &\lambda_i \boldsymbol{P}_{v,i} + (1 - \lambda_i) f(\boldsymbol{Q}_i) \\ =&\lambda_i \left(f(\boldsymbol{Q}_i) + \text{MLP}_v(\boldsymbol{Q}_i) \right) + (1 - \lambda_i) f(\boldsymbol{Q}_i) \\ =&f(\boldsymbol{Q}_i) + \lambda_i \text{MLP}_v(\boldsymbol{Q}_i). \end{aligned} \tag{8}$$

We observe inducer-tuning now perturbs the output $f(\boldsymbol{Q}_i)$ in a residual form, which therefore *connects* prefix-tuning and adapter-tuning.

The procedure of inducer-tuning is summarized in Figure 1, and § 6.2 shows the residual form greatly impacts the model performance.

4.4 Extending the Scope of Value

Besides the representation of the virtual vectors, we propose another improvement via the self-attention decomposition proposed by Hou et al. (2020).

Considering the linear transform right after the attention module, we can accordingly rewrite the attention sub-layer as (ignoring the bias term in the linear transform)

$$\sum_{h=1}^{N_h} \text{softmax}(\boldsymbol{Q}^{(h)} (\boldsymbol{K}^{(h)})^T / \sqrt{p}) \boldsymbol{V}^{(h)} \boldsymbol{W}_o^{(h)},$$

where $\boldsymbol{W}_o^{(h)}$ is the h-th row block in $\boldsymbol{W}_o$. Notably, $\boldsymbol{W}_o^{(h)}$ is attached to the value matrix $\boldsymbol{V}^{(h)}$, suggesting that $\boldsymbol{W}_o^{(h)}$'s should be counted into the complete kernel structure of a head. We therefore define the complete attention output $\bar{f}(\boldsymbol{Q}^{(h)})$ as

$$\text{softmax}\left(\boldsymbol{Q}^{(h)} (\boldsymbol{K}^{(h)})^T / \sqrt{p} \right) \boldsymbol{V}^{(h)} \boldsymbol{W}_o^{(h)}, \tag{9}$$

and align the prefix vectors $\boldsymbol{P}_{v,i}$'s with the rows in $\boldsymbol{V}^{(h)} \boldsymbol{W}_o^{(h)}$, instead of solely $\boldsymbol{V}^{(h)}$ as in prefix-tuning. The detailed implementation of the extended $\boldsymbol{P}_{v,i}$ is provided in Appendix B.3. We can verify the improvement by this extension through the ablation studies in § 6.2.

4.5 A Potential Limitation of Prompting

A potential limitation of prompt-based methods comes from the frozen weight matrices $\boldsymbol{W}_q$ and $\boldsymbol{W}_k$. For all the $n(n + l)$ pairs of query / key vectors in a head, most of the pairs (corresponding to the elements within $\boldsymbol{Q}\boldsymbol{K}^T$) have invariant

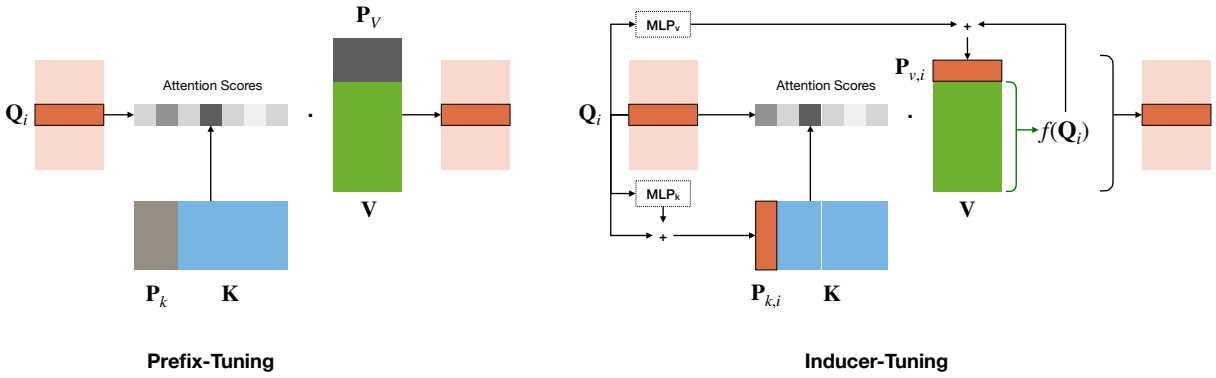

Figure 1: The mechanisms of prefix-tuning (left) and inducer-tuning (right) in inference (the MLP module for reparameterization in prefix-tuning is dropped). For prefix-tuning, the virtual tokens ($\boldsymbol{P}_k, \boldsymbol{P}_v$) are shared among all the query vectors; inducer-tuning instead prepends customized inducers ($\boldsymbol{P}_{k,i}, \boldsymbol{P}_{v,i}$) for a certain vector $\boldsymbol{Q}_i$.

pairwise positional interactions due to the frozen weight matrices $\boldsymbol{W}_q$ and $\boldsymbol{W}_k$. However, on downstream tasks, there can be a mismatch between $\boldsymbol{W}_q$ and $\boldsymbol{W}_k$ maintained from pre-training: the distribution of $\boldsymbol{Q}, \boldsymbol{K}$ will substantially change due to the distinct task-specific datasets as well as the virtual tokens added in the previous layers. There is no adaptation to ensure the positional interactions between $\boldsymbol{Q}, \boldsymbol{K}$ still contribute to the proper representation $f(\boldsymbol{Q}_i)$.

To resolve the potential issue of prefix-tuning, we suggest applying low-rank adaptation (LoRA) (Hu et al., 2021) to $\boldsymbol{W}_q$ as a complement to prompt-based methods, including inducer-tuning. Specifically, before we compute the attention output in each layer, $\boldsymbol{W}_q$ will be updated as

$$\boldsymbol{W}_q \leftarrow \boldsymbol{W}_q + \boldsymbol{B}\boldsymbol{A}, \tag{10}$$

where $\boldsymbol{W}_q$ is kept frozen and $\boldsymbol{B} \in \mathbb{R}^{N_h p \times r}$, $\boldsymbol{A} \in \mathbb{R}^{r \times N_h p}$ will be tunable in training. We report in § 6 that combining both inducer-tuning and LoRA outperforms their individual counterparts.

Final Model. Our final proposed model does the inferencex as follows: ① in each layer, we first apply Equation (10) to update $\boldsymbol{W}_q$ before obtaining $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$; ② construct the inducer matrices $\boldsymbol{P}_k = \boldsymbol{Q} + \text{MLP}_k(\boldsymbol{Q})$, and compute the vector $\boldsymbol{a}$ with the i-th component $\boldsymbol{a}_i = \langle \boldsymbol{Q}_i, \boldsymbol{P}_{k,i} \rangle$; ③ compute the matrix product $[\boldsymbol{a}; \boldsymbol{Q}\boldsymbol{K}^T]/\sqrt{p}$ and then perform softmax over the product—the first column (denoted as $\boldsymbol{p}$) is the weights λ_i's in Equation (7); ④ obtain $\bar{f}(\boldsymbol{Q})$ as in Equation (9), and return $\bar{f}(\boldsymbol{Q}) + \text{diag}(\boldsymbol{p})\overline{\text{MLP}}_v(\boldsymbol{Q})$ (corresponding to Equation (8)) as the complete attention output.

5 Experiments

While prefix-tuning has been shown comparable to fine-tuning on some natural language understanding (NLU) tasks (Liu et al., 2021a), there is still a performance gap between prefix-tuning and fine-tuning on natural language generation (NLG) tasks, especially for those tasks with long input sequences. Complete settings of the experiments below can be found in Appendix A and Appendix B. The code for our algorithms is publicly available at https://github.com/ychen-stat-ml/kernel-adapters.

5.1 Sketch of the Tasks

We test the performance of our methods on both NLU and NLG tasks. For NLU tasks, we follow (He et al., 2021a) to use $\text{RoBERTa}_{\text{BASE}}$ (Liu et al., 2019) on MNLI (Williams et al., 2018) and SST2 (Socher et al., 2013) from the GLUE benchmark (Wang et al., 2019); in SST2, the models predict the two-way sentiment (positive/negative) of a given sentence, and the MNLI task is to decide, given a premise and a hypothesis, whether there is entailment, contradiction, or neither. We use $\text{GPT-2}_{\text{SMALL}}$ (Radford et al., 2019) for NLG tasks: WebNLG-challenge (Gardent et al., 2017) focuses on table-to-text tasks, in which the language models generate some relatively long and sensible sentences based on the triples with solely a few words; in contrast, CoQA (Reddy et al., 2019) provides the data for conversational question answering [6], which requires the language model to return short answers to questions based on long

[6] The official validation set is taken as the test set in our experiments, while we randomly choose 500 instances from the training set as the new validation set.

	Parameters to		WebNLG									CoQA	
			BLEU			MET			TER ↓			EM	F1
	train	store	S	U	A	S	U	A	S	U	A		
Fine-tuning	100.00%	100.00%	59.8	28.7	46.1	0.43	0.29	0.36	0.38	0.68	0.51	59.0	67.4
Adapter-108	1.62%	1.62%	59.5	34.1	48.2	0.42	0.32	**0.38**	0.38	0.61	0.49	57.7	66.4
LoRA-54	1.61%	1.61%	54.8	36.9	46.7	0.40	0.33	0.37	0.41	**0.55**	0.47	57.2	65.7
Prefix-tuning-108	7.98%	1.60%	56.1	37.2	47.6	0.40	0.33	0.37	0.40	**0.55**	0.47	51.8	60.3
MAM-adapter	1.61%	1.61%	58.9	36.2	48.7	0.42	0.33	0.38	0.38	0.59	0.47	56.4	65.0
Inducer-tuning	1.61%	1.61%	59.4	36.8	49.2	0.42	0.33	0.38	0.38	0.59	0.47	57.7	66.1
+ LoRA	1.61%	1.61%	59.8	**37.5**	**49.7***	0.43	**0.34***	**0.38**	**0.37***	0.57	**0.46***	58.7	67.1
MAM inducer-tuning	1.61%	1.61%	**59.9***	36.8	49.5	**0.43**	0.33	**0.38**	**0.37***	0.57	0.47	**59.9***	**68.4***

Table 1: Performance (%) on WebNLG [a] and CoQA. All the parameter-efficient methods have similar sizes of parameters to store. The best scores (among parameter-efficient methods) under different metrics are **boldfaced** (for TER, the lower the metric is, the better the performance is). Significance tests are performed between our methods and the other baselines for each metric (5 runs for WebNLG and 3 runs for the others), and a superscript * is added if the test p-value < 0.05.

[a] For the metrics on WebNLG, the notations are same as in prefix-tuning (Li and Liang, 2021, Table 1) that S, U, and A denote SEEN, UNSEEN, and ALL respectively; in training only the examples from the SEEN categories are used; the examples from the UNSEEN categories only appear in the test set; and ALL consists of all the categories.

Method (# params)	MNLI	SST2
Fine-tuning (100%)	$87.6_{\pm.4}$	$94.6_{\pm.4}$
Bitfit (0.1%)	84.7	93.7
Prefix-tuning (0.5%)	$86.3_{\pm.4}$	$94.0_{\pm.1}$
LoRA (0.5%)	$87.2_{\pm.4}$	$94.2_{\pm.2}$
Adapter (0.5%)	$87.2_{\pm.2}$	$94.2_{\pm.1}$
MAM-Adapter (0.5%)	$\mathbf{87.4}_{\pm.3}$	$94.2_{\pm.3}$
Inducer-tuning (0.5%)	$86.6_{\pm.2}$	$94.1_{\pm.3}$
Inducer-tuning + LoRA (0.5%)	$86.8_{\pm.5}$	$94.7_{\pm.3}$
MAM inducer-tuning (0.5%)	$\mathbf{87.4}_{\pm.04}$	$\mathbf{94.8}_{\pm.3}$

Table 2: Accuracy (%) on MNLI and SST2. Baseline numbers and settings are from He et al. (2021a).

conversational materials. More details about the datasets (including the average sequence length) and the evaluation metrics used are provided in Appendix A.

5.2 Baselines

We compare our method with other representative methods: Fine-Tuning (Howard and Ruder, 2018), Adapters (Houlsby et al., 2019) used by Lin et al. (2020), Prefix-Tuning (Li and Liang, 2021), and LoRA (Hu et al., 2021) [7]; we also follow the strategy in Mix-And-Match (MAM) adapters (He et al., 2021a) to combine inducer-tuning (in self-attention) with adapters in FFN sub-layers, and study how the combination compares to the original MAM adapter (prefix-tuning + adapters in FFN). In Table 1 the suffixes after adapters / prefix-tuning / LoRA indicate the corresponding bottleneck size / prefix length / rank of updates, respectively.

We differentiate the number of parameters to store and tune, as for prefix-tuning, the two numbers are inconsistent due to a re-parametrization trick (Li and Liang, 2021) to mitigate the initialization issue. Instead of directly setting up an embedding matrix for virtual tokens, an additional MLP module in each layer is used in prefix-tuning to model the representation for those virtual tokens; after the fine-tuning stage, the additional MLP modules are dropped and only the output embedding for virtual tokens needs storing, which leads to a regular number of parameters to store. For the proposed inducer-tuning, we adopt the residual form to address the initialization issue and avoid the usage of the extra MLP, which makes inducer-tuning have the same number of parameters to store as to train and behave more like a regular adapter.

To make a fair comparison, we intentionally choose the number of parameters to **store** in prefix-tuning roughly the same as its adapter counterpart by adjusting the prefix length. Detailed settings are available in Appendix B.4.

6 Results

6.1 Main Results

We conclude our experimental results in Tables 1 and 2, comparing the proposed inducer-tuning (§ 4.3), or inducer-tuning with LoRA (§ 4.5), against other baselines. The benefit of using Mix-And-Match (MAM) techniques (He et al., 2021a)

[7] LoRA in our experiments follows the recommended setting by Hu et al. (2021), adjusting both $\boldsymbol{W}_q$ and $\boldsymbol{W}_v$.

	Parameters to		WebNLG									CoQA	
			BLEU			MET			TER ↓			EM	F1
	train	store	S	U	A	S	U	A	S	U	A		
Prefix-tuning-108	7.98%	1.60%	56.1	37.2	47.6	0.40	0.33	0.37	0.40	0.55	0.47	51.8	60.3
Adaptive	1.62%	1.62%	57.5	36.9	48.2	0.41	0.33	0.37	0.39	0.57	0.47	55.4	63.9
Extension	1.55%	1.55%	59.1	36.7	49.0	0.42	0.33	0.38	0.38	0.58	0.47	57.2	65.7
Gating	1.61%	1.61%	57.1	29.2	44.6	0.41	0.286	0.35	0.40	0.65	0.51	49.1	58.5
Inducer-tuning	1.61%	1.61%	59.4	36.8	49.2	0.42	0.33	0.38	0.38	0.59	0.47	57.7	66.1
+ LoRA	1.61%	1.61%	59.8	37.5	49.7	0.43	0.34	0.38	0.37	0.57	0.46	58.7	67.1

Table 3: Compare several variants of inducer-tuning against Prefix-tuning-108 and our proposed methods (copied from Table 1). The exact settings of the methods listed are illustrated in § 6.2 and Appendix B.4.

is also explored and stated as follows.

Performance of our proposed methods. Our proposed methods generally improve the performance of prompt-based methods. The average accuracy of MAM inducer-tuning on MNLI and SST2 is increased by 0.8% compared to Prefix-tuning. The benefits are clearer on NLG tasks: on WebNLG, inducer-tuning with LoRA provides a 1.5% increase in BLEU score compared to Adapter-108 and a 2%+ increase compared to LoRA-54 and Prefix-tuning-108; on CoQA, all the previous parameter-efficient methods cannot attain a close performance to fine-tuning on this harder task, while inducer-tuning with LoRA closes this gap, which shrinks to 0.3% with solely 1.61% tunable parameters of GPT-2.

The MAM technique benefits inducer-tuning. As remarked by He et al. (2021a), the "Mix-And-Match" of adapters in both self-attention and FFN sub-layers can better exploit parameter-efficient transfer learning than only modulating a single sub-layer. We obtain a similar conclusion by replacing prefix-tuning with inducer-tuning (+ LoRA) in self-attention sub-layers. The combination (MAM inducer-tuning) performs well on most of the tasks; especially on the tasks with relatively longer sequences, MNLI and CoQA, MAM inducer-tuning attains respectively 0.6% and 1.2% performance improvement over vanilla inducer-tuning + LoRA.

Long inputs deteriorate prefix-tuning. Notably, the performance of prefix-tuning is sensitive to the input length (c.f. § 4.3). For WebNLG with short inputs, prefix-tuning attains comparable performance with fine-tuning and other parameter-efficient methods. On CoQA, however, prefix-tuning has a substantially lower exact-match / F1 score than others (e.g., over 7% decrease in F1 score compared with fine-tuning). The similar pattern can be observed on the two NLU tasks as well: the performance gap between prefix-tuning and other candidate methods is much smaller on SST2, whose mean sequence length is shorter than MNLI. We remark our proposed adaptive inducers somewhat resolve the issue: both variants of inducer-tuning in Table 1 obtain a 5%+ improvement on CoQA.

Enhance inducer-tuning through adapting pairwise positional interactions. In § 4.5, we speculate the prompt-based methods can benefit from adapting pairwise positional interactions, and we investigate it on both NLU and NLG tasks. With the same parameter budgets, the inducer-tuning + LoRA outperforms the pure inducer-tuning on all tasks. The improvement is more evident in CoQA, the more challenging generation task with longer input sequences. We remark that inducer-tuning more effectively exploits the tunable parameters than LoRA-54 for the value part, as the combination variant also performs better than pure LoRA.

6.2 Ablation Studies

We perform ablation studies on generation tasks to analyze the efficacy of the different components in our proposed method. We recall there are four different features in inducer-tuning compared to prefix-tuning, including the usage of adaptive inducers, the extension of virtual value vectors, the residual form of $\boldsymbol{P}_k$, and the design for $\boldsymbol{P}_{v,i}$ to concentrate around attention output.

Accordingly, we implement three other variants of inducer-tuning to help ablate the effects of the above-mentioned components. Among them, *Adaptive* directly takes $\boldsymbol{Q}_i$ as $\boldsymbol{P}_{k,i}$ but still models $\boldsymbol{P}_{v,i}$ as $f(\boldsymbol{Q}_i) + \text{MLP}_v(\boldsymbol{Q}_i)$; upon Adaptive, *Extension* changes $\boldsymbol{P}_{v,i}$ to $\bar{f}(\boldsymbol{Q}_i) + \overline{\text{MLP}}_v(\boldsymbol{Q}_i)$; compared to Extension, *Inducer-tuning* just mod-

ifies $\boldsymbol{P}_{k,i}$ to $\boldsymbol{Q}_i + \text{MLP}_k(\boldsymbol{Q}_i)$; to justify the design that $\boldsymbol{P}_{v,i}$ centers around the attention output, *Gating* models $\boldsymbol{P}_{v,i}$ simply as $\overline{\text{MLP}}_v(\boldsymbol{Q}_i)$, and the new complete attention output thus becomes $(1-\lambda_i)\bar{f}(\boldsymbol{Q}_i) + \lambda_i \overline{\text{MLP}}_v(\boldsymbol{Q}_i)$. The concrete setting of each variant is deferred to Appendix B.4 due to limited space.

The usage of adaptive inducers. To demonstrate the benefits of adaptive inducers, we compare Prefix-tuning-108 with the basic counterpart—Adaptive. Table 3 shows Adaptive attains close performance to Prefix-tuning-108 on WebNLG while obtaining a substantial improvement on CoQA, which has longer inputs.

The extension of virtual value vectors. We observe an obvious improvement attributed to extending the scope of virtual value vectors by comparing the performance of Adaptive and Extension. For almost all the metrics, Extension obtains better performance than Adaptive, with the same number of tunable parameters.

The residual form of $\boldsymbol{P}_k$. A natural design for $\boldsymbol{P}_k$ is to directly model it as $\boldsymbol{Q}$, which would automatically be the closest vectors to the ones in $\boldsymbol{Q}$. To ablate the usage of MLP_k, we compare Inducer-tuning against Extension, which follows the natural design to model $\boldsymbol{P}_k$. Through the empirical results, we find assigning parameters to MLP_k can still slightly help the performance of inducer-tuning.

$\boldsymbol{P}_{v,i}$ centers around $\bar{f}(\boldsymbol{Q}_i)$. Lastly, to show the benefits of modeling $\boldsymbol{P}_{v,i}$ as centering around $\bar{f}(\boldsymbol{Q}_i)$, we compare the variant Gating against Inducer-tuning. While Gating has a weighted sum form similar to prefix-tuning, it suffers from a great performance drop on both tasks, which justifies the effectiveness of our design for $\boldsymbol{P}_{v,i}$'s.

7 Conclusion

In this work, we connect attention modules to Nadaraya-Watson kernel estimators and review prefix-tuning from a kernel estimator perspective. We speculate that continuous prompt tuning prompts serve as inducing variables in kernel methods. Following this principle, we propose inducer-tuning, which customizes an "inducer" for each query vector from the input sequence and adopts a residual form to resolve the initialization issue in prefix-tuning. In addition, the perspective implies a potential limitation of prompt-based methods: the positional interactions in attention cannot adapt to the new tasks in prompt tuning. We empirically demonstrate that our proposed method performs better on NLU and NLG tasks.

Limitations

In this section, we start with a common limitation of the current parameter-efficient techniques, and further, discuss the specific limitation of our methods.

The shared limitation of parameter-efficient techniques is that they are not computation-efficient; These methods choose to directly inherit the pre-trained weights of the backbone model and add some extra modules, which increases the computational cost of these methods during inference.

Additionally, our method, which relies on the unique structure of self-attention, is only applicable to attention modules. And thus, not as generic as LoRA and adapters.

Ethics Statement

As an efficient method for NLP, we consider our work to have a low ethical risk since the outcomes of the algorithm mainly depend on the downstream applications. The usage of the method would be the same as some previous methods, i.e., the practical deployment for some applications. Also, our method doesn't assume any specific structure of the input and thus doesn't leverage biases in the data. We conclude that our work will not likely have a negative ethical impact.

References

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

Yifan Chen and Yun Yang. 2021. Fast statistical leverage score approximation in kernel ridge regression. In *International Conference on Artificial Intelligence and Statistics*, pages 2935–2943. PMLR.

Yifan Chen, Qi Zeng, Heng Ji, and Yun Yang. 2021. Skyformer: Remodel self-attention with gaussian kernel and nystr\" om method. *Advances in Neural Information Processing Systems*, 34.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2020. Rethinking attention with performers. In *International Conference on Learning Representations*.

William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. Creating training corpora for NLG micro-planners. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 179–188, Vancouver, Canada. Association for Computational Linguistics.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021a. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

Ruidan He, Linlin Liu, Hai Ye, Qingyu Tan, Bosheng Ding, Liying Cheng, Jia-Wei Low, Lidong Bing, and Luo Si. 2021b. On the effectiveness of adapter-based tuning for pretrained language model adaptation. *arXiv preprint arXiv:2106.03164*.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. *Advances in Neural Information Processing Systems*, 33:9782–9793.

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

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339.

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

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438.

Mihir Kale and Abhinav Rastogi. 2020. Text-to-text pre-training for data-to-text tasks. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 97–102, Dublin, Ireland. Association for Computational Linguistics.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. *arXiv e-prints*, pages arXiv–2106.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *ICLR (Poster)*.

Alon Lavie and Abhaya Agarwal. 2007. Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. In *Proceedings of the second workshop on statistical machine translation*, pages 228–231.

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

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, abs/2101.00190.

Zhaojiang Lin, Andrea Madotto, and Pascale Fung. 2020. Exploring versatile generative language model via parameter-efficient transfer learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 441–459.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021a. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *CoRR*, abs/2110.07602.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2021b. Gpt understands, too. *arXiv:2103.10385*.

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. 2018. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Cameron Musco and Christopher Musco. 2017. Recursive sampling for the nystrom method. In *NIPS*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. 2020. Random feature attention. In *International Conference on Learning Representations*.

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.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition

for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online. Association for Computational Linguistics.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020a. MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online. Association for Computational Linguistics.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020b. Unks everywhere: Adapting multilingual language models to new scripts. *arXiv preprint arXiv:2012.15562*.

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.

Siva Reddy, Danqi Chen, and Christopher D Manning. 2019. Coqa: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2020. Adapterdrop: On the efficiency of adapters in transformers. *arXiv preprint arXiv:2010.11918*.

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 *Empirical Methods in Natural Language Processing (EMNLP)*.

Edward Snelson and Zoubin Ghahramani. 2005. Sparse gaussian processes using pseudo-inputs. In *NIPS*.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pages 223–231.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Asa Cooper Stickland and Iain Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*, pages 5986–5995. PMLR.

Michalis Titsias. 2009. Variational learning of inducing variables in sparse gaussian processes. In *Artificial intelligence and statistics*, pages 567–574. PMLR.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. 2019. Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4344–4353.

Ahmet Üstün, Arianna Bisazza, Gosse Bouma, and Gertjan van Noord. 2020. UDapter: Language adaptation for truly Universal Dependency parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2302–2315, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Marko Vidoni, Ivan Vulić, and Goran Glavaš. 2020. Orthogonal language and task adapters in zero-shot cross-lingual transfer. *arXiv preprint arXiv:2012.06460*.

Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou, and Daniel Cer. 2021. Spot: Better frozen model adaptation through soft prompt transfer. *arXiv preprint arXiv:2110.07904*.

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 the Proceedings of ICLR.

Ruize Wang, Duyu Tang, Nan Duan, Zhongyu Wei, Xuanjing Huang, Guihong Cao, Daxin Jiang, Ming Zhou, et al. 2020. K-adapter: Infusing knowledge into pre-trained models with adapters. *arXiv preprint arXiv:2002.01808*.

Larry Wasserman. 2006. *All of nonparametric statistics*. Springer Science & Business Media.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

Yan Xu, Etsuko Ishii, Zihan Liu, Genta Indra Winata, Dan Su, Andrea Madotto, and Pascale Fung. 2021. Retrieval-free knowledge-grounded dialogue response generation with adapters. *arXiv preprint arXiv:2105.06232*.

A Dataset Details

- The Multi-Genre Natural Language Inference Corpus (Williams et al., 2018, **MNLI**) involves 433k sentence pairs of premises and hypotheses, labeled with textual entailment annotations. The premise sentences include ten distinct genres, and the classification can be performed on both the matched (in-domain) and mismatched (cross-domain) sections. Concatenating premises and hypothesis as the inputs, we obtain the sequence lengths are on average 39.9 and max 444. For the results reported in Table 2, we follow Hu et al. (2021) and take mismatched accuracy as the metric.

- The Stanford Sentiment Treebank (Socher et al., 2013, **SST2**) is a corpus of movie reviews and human annotations of their sentiment. This task is incorporated into the GLUE benchmark (Wang et al., 2019), and the dataset split assigns $67k$ sentences to the training set and $0.9k$ to the dev set. In SST2, the sequence lengths are on average 13.3 and max 66, much shorter than in MNLI. As specified in the GLUE benchmark, we test the accuracy metric on whether the sentiment of a review sentence is positive or negative.

- The instances in **WebNLG** dataset are the mapping set of RDF triples to text. They are Data/Text pairs, where the "Data" is in a format of (subject, property, object) triples. For the train and the validation set, they involve nine categories which are extracted from DBpedia; while in the test set, there are five extra unseen categories, which can partially reflect the generalization of the adaptation methods. The input sequences in the training set consist of 1 to 7 triples, and the lengths of most sequences are bounded by 50 (as each triple only includes three short phrases). The official evaluation script is used in our experiments, and we report BLEU (Papineni et al., 2002), METEOR, (Lavie and Agarwal, 2007) and TER (Snover et al., 2006) as the metrics.

- **CoQA** is a large-scale dataset, mainly for conversational question answering. It collects more than 8K conversations over text passages, involving over 127K questions with answers in 5 domains. The average conversation length is 15 turns (each turn consists of a question and an answer). The task requires the language model to generate answers to the given questions based on related conversation histories and documents in the dataset. The average passage length in CoQA is 271 (Reddy et al., 2019, Table 3). We simply follow the evaluation script provided on the official website, reporting both the macro-average F1 score of word overlap and the exact-match metric (Reddy et al., 2019).

B Training Details

We mainly implement our methods based on the GitHub repositories provided by Lin et al. (2020) and He et al. (2021a). Our code will be made public after the review procedure.

B.1 General Training Settings

For the NLU tasks, we exactly follow the experimental setup used by He et al. (2021a), and more details can be found in Appendix B.2.

For the two NLG tasks, we mainly follow the experimental setting adopted by Lin et al. (2020), and specifically, keep using "task embeddings" in our experiments, as they are also applied in the original GPT-2 model. These task embeddings are specialized segment embeddings used to indicate the different components of the text input (e.g., the three components of a triple in WebNLG, questions, and answers in CoQA, etc.). [8]

We list the task embedding used in each NLG task: for CoQA, we follow the task embedding suggested by Lin et al. (2020); for WebNLG, we simply use the special tokens to indicate the different components in the triples. The details of the special tokens in each task are summarized in Table 4. Notably, the parameter budget for task embedding is much smaller than the number of tunable parameters in the aforementioned parameter-efficient adaptation methods (around 2M).

B.2 Hyper-parameters for Training

For NLU tasks, we train the models with Adam (Kingma and Ba, 2015) optimizer and use a polynomial learning rate scheduler to make the learning rate linearly decay; specifically, the learning rate is linearly warmed up from 0 for the first 6For NLG tasks, an AdamW (Loshchilov and Hutter, 2018) optimizer is applied to train the models, and a linear learning rate scheduler with a 500-step warmup duration is used.

[8] The task embedding for the special tokens will also be updated during training, though we do not count them in Table 1.

Datasets	Special tokens	# of trainable parameters for task embedding
WebNLG	<bos_webnlg>, <eos_webnlg>, <subject>, <property>, <object>, <target_webnlg>	$6 * 768 = 4608$
CoQA	<bos_qa>, <eos_qa>, <question>, <answer>, <document>	$5 * 768 = 3840$

Table 4: The special tokens used in different tasks and the corresponding size of trainable parameters.

For the evaluation of NLG tasks, we follow the script provided by Lin et al. (2020) to generate the texts through a greedy search for both WebNLG and CoQA. As for the number of epochs and the argument for weight decay, we mainly follow the setting used by Lin et al. (2020); Hu et al. (2021): for WebNLG, we train the model for 10 epochs; for CoQA, we train the model for 5 epochs.

For the model-specific hyper-parameters, namely batch size (gradient accumulation is used if necessary) and learning rate, we decide them for different methods based on the loss on the validation set. For the proposed method inducer-tuning with LoRA and MAM inducer-tuning in Table 1, we set the learning rate as 0.00125, and the batch size is 16 for WebNLG; for CoQA, the learning rate we use is 0.001, and the batch size is 16. On MNLI, we set the learning rate as 0.0002 for both two methods and the batch size as 32 for inducer-tuning with LoRA and 16 for MAM inducer-tuning; on SST2, the learning rate is similarly set as 0.0002, the batch size for inducer-tuning with LoRA is 16, and for MAM inducer-tuning 64.

To reduce the random variability in the results, all the methods reported are trained for multiple independent runs. In particular, for WebNLG, we train models over 5 runs, and for CoQA, MNLI, and SST2 3 runs. The reported numbers in the cells in Tables 1 and 2 are the mean value averaged over the runs, and the significance tests in Table 1 are also based on the replicates.

B.3 Training Efficiency and Implementation Details

All the models in this work are implemented by PyTorch. For the training runtime, if we perform the training using 1 Tesla V100 16GB GPU, on WebNLG, it will take inducer-tuning and its variants around 8 minutes to finish one epoch; on CoQA, the time cost is around 160 minute/epoch. On MNLI and SST2, the runtime is 120 and 25 minute/epoch, respectively. We remark all the parameter-efficient tuning methods have a similar time cost, while indeed, they solely slightly save training time compared to fine-tuning. The same phenomenon is also observed by Lin et al. (2020); Rücklé et al. (2020).

For the implementation of $\overline{\text{MLP}}_v$, we provide the exact expression for $\overline{\text{MLP}}_v^{(h)}(\boldsymbol{Q}^{(h)})$ in head h as follows:

$$\sigma\left(\boldsymbol{Q}^{(h)}\boldsymbol{W}_1^{(h)} + \mathbf{1}(\boldsymbol{b}_1^{(h)})^T\right)\boldsymbol{W}_2^{(h)} + \mathbf{1}\boldsymbol{b}_2^T, \quad (11)$$

where σ is the activation function. As the superscript suggests, $\boldsymbol{W}_1^{(h)} \in \mathbb{R}^{p\times r}$, $\boldsymbol{b}_1^{(h)} \in \mathbb{R}^r$, and $\boldsymbol{W}_2^{(h)} \in \mathbb{R}^{r\times N_h p}$ are specific to the head h, while $\boldsymbol{b}_2 \in \mathbb{R}^{N_h p}$ are shared among all the heads, which is the same case as in Equation (4) (in the original attention sub-layer, the bias term $\boldsymbol{b}_o$ applies to all the heads as well).

B.4 Specific settings for baseline methods

In this subsection, we provide the detailed setting for the methods in Tables 1, 2, and 3 that need further specification.

In Table 1, the settings for Adapter-108 and Prefix-tuning-108 are clear, as the only arguments are the bottleneck size / prefix length; for LoRA-54, we apply rank-54 updates for both $\boldsymbol{W}_q$ and $\boldsymbol{W}_v$, as suggested by Hu et al. (2021); for MAM adapter, we mimic the parameter assignment scheme (bottleneck size 512 for FFN and prefix length 30) by He et al. (2021a), and use the ratio $102 : 6$ to implement MAM adapters with 1.61% tunable parameters.

For the variants of inducer tuning, their settings are summarized in Table 5. In this table, the numbers in column MLP_k and MLP_v are the bottleneck sizes used for computing $\boldsymbol{P}_k$ and $\boldsymbol{P}_v$; notice for Adaptive, the scope of virtual value tokens is not extended and thus has a larger bottleneck size than others. (Recall for $\overline{\text{MLP}}_v$, the size of $\boldsymbol{W}_2^{(h)}$ in Equation (11) is larger than the counterparts in MLP_v. For the numbers in column *LoRA*, they are

the rank of the update used in the LoRA component to adjust $\boldsymbol{W}_q$; only for our proposed method inducer-tuning with LoRA, the number will be nonzero.

C Attention as Kernels

To justify the claim that attention is a kernel operation, we construct a Nadaraya–Watson kernel estimator (Wasserman, 2006, Definition 5.39) of a query vector $\boldsymbol{Q}_i$ (taking $\{\boldsymbol{K}_j\}_{j=1}^n$ as the supporting points) as follows:

$$f(\boldsymbol{Q}_i) = \sum_{j=1}^{n} \ell_j(\boldsymbol{Q}_i)\boldsymbol{C}_j, \tag{12}$$
$$\text{where} \quad \ell_j(\boldsymbol{Q}_i) := \frac{\kappa(\boldsymbol{Q}_i, \boldsymbol{K}_j)}{\sum_{k=1}^{n} \kappa(\boldsymbol{Q}_i, \boldsymbol{K}_j)}.$$

$\kappa(\cdot,\cdot)$ is a kernel function, and $\boldsymbol{C}_j$'s are the coefficients corresponding to the rows $\boldsymbol{V}_j$'s in the value matrix $\boldsymbol{V}$.

Take kernel function $\kappa(x, y) = \exp\left(\langle x, y\rangle / \sqrt{p}\right)$. We slightly abuse the notation $\kappa(\boldsymbol{Q}, \boldsymbol{K})$ to represent the n-by-n empirical kernel matrix $\boldsymbol{M}$, in which the i-th row and the j-th column is $\kappa(\boldsymbol{Q}_i, \boldsymbol{K}_j), \forall i \in [n], j \in [N]$. With these notations, the output of the kernel estimator will be,

$$\boldsymbol{D}^{-1}\boldsymbol{M}\boldsymbol{C}, \tag{13}$$

where $\boldsymbol{D}$ is a diagonal matrix serving as the row normalization in Equation (12), and $\boldsymbol{C}$ is an n-by-p matrix with $\boldsymbol{C}_j$ as its j-th row. We observe an obvious correspondence between Equation (13) and the standard attention in Equation (3). The correspondence implies a finer division of the attention module: the empirical kernel matrix $\boldsymbol{M}$ ($\boldsymbol{D}$ is decided by $\kappa(\boldsymbol{Q}, \boldsymbol{K})$) and the value part $\boldsymbol{C}$. (In Section 4.4, we show that $\boldsymbol{C}$ includes but is *not* limited to the *value* matrix in attention.)

D Example

We provide an example answer generated by fine-tuning and our inducer-tuning with LoRA on CoQA in Table 6.

NLG tasks	MLP_k	MLP_v	LoRA	FFN-adapter
Adaptive	0	108	0	0
Extension	0	16	0	0
Gating	10	15	0	0
Inducer-tuning	10	15	0	0
$\sim$ w/ LoRA	5	12	24	0
MAM inducer-tuning	3	7	16	42
NLU tasks	MLP_k	MLP_v	LoRA	FFN-adapter
Inducer-tuning	6	4	0	0
Inducer-tuning + LoRA	2	4	4	0
MAM inducer-tuning	2	2	4	12

Table 5: The exact parameter assignment settings for variants of inducer-tuning in Tables 1, 2, and 3.

Documents	Kendra and Quinton travel to and from school every day. Kendra lives further from the bus stop than Quinton does, stops every morning at Quinton's house to join him to walk to the bus stop. Every afternoon, after school, when walking home from the bus stop they go in for cookies and milk that Quinton's mother has ready and waiting for them. Quinton can't eat cheese or cake so they had the same snack every day. They both work together on their homework and when they are done they play together. Kendra always makes sure to leave in time to get home for dinner. She doesn't want to miss story time which was right before bedtime. One morning Kendra walked up to Quinton's house, **she thought something might be wrong** because normally Quinton was waiting outside for her and on this morning **he was not to be found**. Kendra went up to the door and knocked. She waited and waited and yet **no one answered**. She saw that Quinton's mother's car wasn't in their driveway which was weird. She waited for a few bit looking up and down the block and getting worried when Quinton was nowhere to be found. Kendra didn't want to miss the bus to school and hurried off to make it in time. The bus driver saw that she was upset and that Quinton was not with her that morning. She told him what happened and he said that he was sure that everything would be okay. Kendra got to school, ran to her teacher and told him what happened that morning. The teacher smiled and told her not to worry, Quinton's mother had called and he was going to the dentist and would be at school after lunch and that she would see him at the bus stop like normal tomorrow. Q4: What happened when Kendra knocked on Quinton's door? Reference: no one answered
Fine-tuning	she thought something might be wrong
Inducer-tuning w/ LoRA	he was not to be found

Table 6: A qualitative example from CoQA. In particular fine-tuning generates an answer farther away from the correct answer than inducer-tuning with LoRA.