Empowering Parameter-Efficient Transfer Learning by Recognizing the Kernel Structure in Attention

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

The massive amount of trainable parameters in the pre-trained language models (PLMs) makes them hard to be deployed to multiple downstream tasks. To address this issue, parameter-efficient transfer learning methods have been proposed to tune only a few parameters during fine-tuning while freezing the rest. This paper looks at existing methods along this line through the *kernel lens*. Motivated by the connection between self-attention in transformer-based PLMs and kernel learning, we propose *kernel-wise adapters*, namely *Kernel-mix*, that utilize the kernel structure in self-attention to guide the assignment of the tunable parameters. These adapters use guidelines found in classical kernel learning and enable separate parameter tuning for each attention head. Our empirical results, over a diverse set of natural language generation and understanding tasks, show that our proposed adapters can attain or improve the strong performance of existing baselines.

1 Introduction

Transfer learning using large-scale transformer-based pre-trained language models (PLMs) (Radford et al., 2019) has become the standard scheme for various natural language processing (NLP) tasks. Among many strategies, fine-tuning these PLMs emerges as the predominant strategy to adapt the generic models to a specific task (Howard and Ruder, 2018). However, deploying these models is a challenge as curating customized models across a wide variety of tasks would lead to scalability issues. It requires one to store (and sometimes move) multiple copies of the PLM parameters for different tasks, which is inefficient.

A popular approach to tackling such scalability issues is to make the PLM-based transfer learning more parameter-efficient. This can be done by freezing most of the PLM parameters and inserting small trainable modules into the PLM. Adapters (Houlsby et al., 2019; Pfeiffer et al., 2020; Mahabadi et al., 2021; Hu et al., 2021) and Prefix-/Prompt-tuning (Shin et al., 2020; Li and Liang, 2021; Lester et al., 2021; Liu et al., 2021c,b) have emerged as the prominent approaches under this paradigm. These methods are incredibly parameter-efficient and have comparable performance to full fine-tuning models on many common NLP tasks (mainly in Natural Language Understanding) by tuning only 0.1-3% task-specific parameters of the original PLMs.

However, most of these studies take the PLMs as a black box, i.e., these methods are not customized to transformers. This raises whether parameter-efficient fine-tuning has fully utilized the transformer structure in PLMs. Therefore in this work, we propose *kernel-wise adaptation*, which recognizes and utilizes the kernel structure within self-attention—the core component in a transformer. Specifically, we take inspiration from recent work that connects self-attention to kernel learning (Choromanski et al., 2020; Chen et al., 2021; Tsai et al., 2019) to treat the different attention heads in a transformer's attention sub-layer as separate kernel estimators. We hypothesize that parameter-efficient tuning can benefit from some useful guidelines in classical kernel learning literature and incorporate them into our proposed methods. These include:

1. By interpreting attention heads as kernel estimators, we design the adaptation to be head-specific;
2. We assign more budgets of tunable parameters to learn the *value* components in the attention mechanism, which correspond to *coefficients* in kernel methods.

We discuss these guidelines in detail in § 4.1. We also evaluate our hypotheses through rigorous

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

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

Findings of the Association for Computational Linguistics: NAACL 2022, pages 1375 - 1388

empirical evaluation. First, we test the effectiveness of the two guidelines above by comparing the default LoRA (Hu et al., 2021)—a state-of-the-art approach for efficient adaptation—and two of its variants that implement the two guidelines, respectively. Next, we evaluate our variant of kernel-wise adaptation on three Natural Language Generation (NLG) benchmarks and two Natural Language Understanding (NLU) tasks using the GPT-2 architecture. While parameter-efficient work has extensively covered NLU tasks, it is unknown how well the results transfer to NLG tasks. As language generation typically requires more expressive models, we put more emphasis on multiple NLG tasks that include data-to-text, free-form question answering (QA), and summarization. The empirical results in § 6 demonstrate that with the same parameter budgets, our proposed method can attain better generation quality and classification accuracy than previous techniques, and in many settings, it is close to or even outperforms the full parameter fine-tuning.

2 Related Work

The literature on parameter-efficient adaptation can be broadly categorized as follows:

Adapters. Originally proposed by Houlsby et al. (2019); Pfeiffer et al. (2020), adapters modulate the output of a transformer layer by inserting small Multi-layer Perception (MLP) bottleneck layers. Recent work has proposed many variants of the original adapters, including dropping adapters across several layers (Rücklé et al., 2020) or constraining adapters to be low-rank operators (Mahabadi et al., 2021).

A recent line of work focused on identifying the important subset of parameters within the PLMs. Ben Zaken et al. (2021) proposed to only tune the bias terms in the PLMs. MPOP (Liu et al., 2021a) suggested decomposing the weight matrices in PLMs through matrix product operators (MPO) and only trained the matrices of small size (freezing the large matrices) obtained from the decomposition, which implicitly recognizes the small matrices as the important subset.

Low-rank adaptation (LoRA) (Hu et al., 2021) directly assumes that the update of the weight matrices during training can be approximate low-rank, and accordingly proposed to re-parameterize the original weight matrix $\boldsymbol{W}$ by $\boldsymbol{W} + \boldsymbol{B}\boldsymbol{A}$, where $\boldsymbol{W}$ is frozen to its pre-trained weights whereas $\boldsymbol{A} \in \mathbb{R}^{r\times N_h p}$, $\boldsymbol{B} \in \mathbb{R}^{N_h p\times r}$ are updated in training [1]. We note that LoRA too introduces new weights $\boldsymbol{A}$ and $\boldsymbol{B}$ similar to adapters, but they are used only to re-parameterize the existing weights and do not add extra sandwiched layers that modify the original model architecture.

Prefix-tuning. Originally shown in GPT-3 (Brown et al., 2020), prompts are extra tokens that help in the task adaptation of PLMs. Transitioning from the manual design of prompts, Shin et al. (2020) searched for the prompts over the discrete space of tokens based on the task-specific training data; Li and Liang (2021); Lester et al. (2021); Liu et al. (2021c,b) further extended the search space to continuous prompts and tuned the prompts through back-propagating the error in training. Prompt-based methods have been shown to be similar to adapters by (He et al., 2021).

3 Preliminaries

We start by providing a brief introduction to the transformer architecture (§ 3.1) and then revisit the connection between attention and kernel estimators (§ 3.2), building on which we propose the kernel-wise adapter in § 4.

3.1 Transformer Architecture

Transformers (Vaswani et al., 2017) are composed of L stacked layers, where each layer comprises of a multi-headed attention and a fully connected feed-forward network (FFN) sub-layer.[2] The attention sub-layer, assuming N_h heads and dimension size p for each head, first maps an input $\boldsymbol{X} \in \mathbb{R}^{n\times N_h p}$ into the query ($\boldsymbol{Q}$), key ($\boldsymbol{K}$), and value ($\boldsymbol{V}$) matrices through the following affine transformations:

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

where $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{n\times N_h p}$, $\boldsymbol{W}_q, \boldsymbol{W}_k, \boldsymbol{W}_v$ are $N_h p$-by-$N_h p$ weight matrices, and $\boldsymbol{b}_q, \boldsymbol{b}_k, \boldsymbol{b}_v \in \mathbb{R}^{N_h p}$ are the bias terms[3]. After the transformation, the three components $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ are split into N_h blocks corresponding to different heads. For example, $\boldsymbol{Q}$ is re-written 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

[1] In implementation, LoRA also trains the bias terms in the linear transform besides the matrices $\boldsymbol{A}, \boldsymbol{B}$, while for brevity, the bias terms are omitted throughout the paper.

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

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

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 then computed as:

$$\begin{aligned}\boldsymbol{L}^{(h)}\boldsymbol{V}^{(h)} &:= \text{softmax}(\boldsymbol{Q}^{(h)}(\boldsymbol{K}^{(h)})^T/\sqrt{p})\boldsymbol{V}^{(h)} \\ &= (\boldsymbol{D}^{(h)})^{-1}\boldsymbol{M}^{(h)}\boldsymbol{V}^{(h)}, \end{aligned} \tag{2}$$

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 $\boldsymbol{M}^{(h)}$, corresponding to the normalization part in softmax.

After we obtain the outputs in each head, they are concatenated as,

$$\boldsymbol{L} := (\boldsymbol{L}^{(1)}\boldsymbol{V}^{(1)}, \ldots, \boldsymbol{L}^{(N_h)}\boldsymbol{V}^{(N_h)}),$$

followed by the overall output,

$$\boldsymbol{L}\boldsymbol{W}_o + \mathbf{1}\boldsymbol{b}_o^T, \tag{3}$$

where $\boldsymbol{W}_o$ and $\boldsymbol{b}_o$ are similarly sized as the other matrices in Equation (1).

3.2 Attention as Kernel Estimators

For each head in the attention module, we have given the expression of attention output in Equation 2. In this subsection, we will re-write attention as a kernel estimator to show the connection.

In computing the attention output (of a single head), we have a length-n input sequence $\{x_i\}_{i=1}^n$ (the rows in $\boldsymbol{X}$) and accordingly we can obtain N [4] key vectors $\{k_j\}_{j=1}^N \subset \mathbb{R}^p$ (from the *key* matrix $\boldsymbol{K}$) and query vectors $\{q_i\}_{i=1}^n \subset \mathbb{R}^p$ (from $\boldsymbol{Q}$).[5] The original goal of self-attention is to obtain the representation of each input token x_i: $g(x_i)$. By denotation exchange: $q_i := x_i$ and $f(q_i) := g(x_i)$, we can also understand the aforementioned self-attention module as returning the representation $f(q_i)$ of the input query vector q_i through $\{k_j\}_{j=1}^n$, which behaves as a kernel estimator (Choromanski et al., 2020; Peng et al., 2020; Chen et al., 2021). Specifically, for a single query vector q_i, a Nadaraya–Watson kernel estimator (Wasserman, 2006, Definition 5.39) models its representation as,

$$f(q_i) = \sum_{j=1}^{n} \ell_j(q_i)c_j, \tag{4}$$

$$\text{where} \quad \ell_j(q_i) := \frac{\kappa(q_i, k_j)}{\sum_{k=1}^N \kappa(q_i, k_j)}.$$

Here, $\kappa(\cdot,\cdot)$ is a kernel function, and c_j's are the coefficients (c_j can either be a scalar or a vector in different applications) that are learned during training. In this estimator, $\{k_j\}_{j=1}^n$ serve as the *supporting points* which help construct the representation for an input q_i.

For 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 an n-by-N empirical kernel matrix, whose element in the i-th row and the j-th column is $\kappa(q_i, k_j), \forall i \in [n], j \in [N]$. With these notations, the representation of the whole sequence $\boldsymbol{Q}$ will be,

$$\boldsymbol{D}^{-1}\kappa(\boldsymbol{Q}, \boldsymbol{K})\boldsymbol{C}, \tag{5}$$

where $\boldsymbol{D}$ is a diagonal matrix for row normalization in Eq. (4), and $\boldsymbol{C}$ is an N-by-p matrix whose j-th row is c_j. Considering the correspondence between Equation (5) and the standard softmax attention in Equation (2), we can have a finer division of the attention module: the empirical kernel matrix $\kappa(\boldsymbol{Q}, \boldsymbol{K})$ ($\boldsymbol{D}$ is decided by $\kappa(\boldsymbol{Q}, \boldsymbol{K})$) and the coefficient part $\boldsymbol{C}$, which includes but is *not* limited to *value* matrices in attention (see § 4). In what follows, we will discuss how we build adapters for these two parts.

4 Method

We introduce our *Kernel-mix* method in this section, which builds upon the proposed *Kernel-wise* adaptation. To explain the principle behind Kernel-wise, we first illustrate the guidelines we aim to adopt in our adapter design and show that existing methods fail to satisfy them (§ 4.1). With these details, we finally propose our method in § 4.2 and § 4.3.

4.1 *Guidelines* Motivated by Kernel Learning

Given the analogy between attention in PLMs and kernel estimators, we hypothesize that parameter-efficient adaptation should be aware of this connection in transformer-based PLMs and utilize desirable guidelines emerging from the literature on kernel learning. Here we discuss the guidelines introduced in § 1 in further detail.

Guideline-1 suggests that the adaptation should be head-specific. Conceptually, different heads correspond to different empirical kernel matrices (distinct distribution of attention scores), and it will be beneficial to adapt the attention module in a head-specific manner. The effect of head-specific

[4] Note that N may not always equal n, such as in cross attention ($N \neq n$) or in prefix-tuning ($N > n$ due to the prefix prepended to the key matrix) (Li and Liang, 2021).

[5] In this subsection we omit the superscript (h) for simplicity since the discussion is limited within a single head

adaptation is also observed by other work, e.g., (He et al., 2021) that mentioned multi-head influence can make methods such as prefix-tuning more expressive.

Guideline-2 is that we should assign more parameter budgets to the coefficient (or *value*) part of attention compared to the empirical kernel matrix part (*query* and *key*). This guideline comes from the classical optimization procedure in kernel learning (Wasserman, 2006, Definition 5.29) where we fix the kernel in use and only perform the unconstrained optimization for the coefficients (c_j's in Equation (4)). This practice in kernel learning can be justified by Representer Theorem (Schölkopf et al., 2001) that the minimizer f^* of some certain empirical risks admits a representation of the form:

$$f^*(\cdot) = \sum_{j=1}^{N} \alpha_j \kappa(\cdot, k_j),$$

where α_j's are the free parameters to optimize. Representer Theorem indicates that the target estimator $f^*(\cdot)$ is simply a linear combination of $\kappa(\cdot, k_j)$'s, and therefore many kernel methods focus on optimizing the coefficients α_j's. Revisiting Equation (4), we find $\alpha_j = \frac{c_j}{\sum_{j'=1}^{N} \kappa(q_i, k_{j'})}$ under the setting of transformers, which motivates us to apply Guideline-2 to better model the tunable coefficient part c_j's in attention.

In addition, kernel learning theory concludes that the sample efficiency of a Nadaraya-Watson kernel estimator is mainly influenced by its bandwidth (the scaling factor, corresponding to the factor $\frac{1}{\sqrt{p}}$ in Equation (2)), rather than the concrete form of the empirical kernel matrix $\kappa(\boldsymbol{Q}, \boldsymbol{K})$ (Wasserman, 2006, Section 5.4). This implies that the adaptation to the empirical kernel matrix can be conservative. This is also similar to the conjecture by (He et al., 2021), which mentions "attention learns pairwise positional interactions which do not require large capacity for adapting to new tasks."

Do Existing Adapters Satisfy the Guidelines? Adapters are designed to modify the hidden states in a certain step in a transformer, and their mechanism can be stated as,

$$\boldsymbol{H} \leftarrow \boldsymbol{H} + \Delta \boldsymbol{H},$$

where $\boldsymbol{H}$ is the "hidden state" in a certain step, and $\Delta \boldsymbol{H}$ is the update given by the adapter. As shown in (He et al., 2021), this definition embodies most of the recent proposals for efficient adaptation, such as adapters, prefix-tuning, LoRA, and similar variants.

The original MLP-based adapters, which only adjust the output of a particular layer (Houlsby et al., 2019), do not modify the empirical kernel matrix in the attention sub-layer.

Prefix-tuning satisfies the first guideline as it is head-specific by nature (it prepends trainable continuous prefixes to both key and value matrices in each head). However, it fails to satisfy the second guideline as it enforces an equal assignment of tunable parameters to both the kernel and the coefficient parts (since the prefixes for *key* matrices and *value* matrices correspond to each other).

As for weight-updating adapters, such as LoRA (Hu et al., 2021), its proposed setup disobeys both guidelines. The original LoRA updates the whole weight matrix, which is not head-specific. To explain this, consider the weight matrix $\boldsymbol{W}_q$ as an example [6]. In each training step, LoRA updates $\boldsymbol{W}_q$ with a low-rank matrix $\boldsymbol{BA}$ of the same size as $\boldsymbol{W}_q$. However, if we denote the r-by-$N_h p$ matrix $\boldsymbol{A}$ as $(\boldsymbol{A}^{(1)}, \boldsymbol{A}^{(2)}, \ldots, \boldsymbol{A}^{(N_h)})$, $\boldsymbol{A}^{(h)} \in \mathbb{R}^{r \times p}, \forall i \in [N_h]$, the modification to the weight matrix $\boldsymbol{W}_q^{(h)}$ for each head would be $\boldsymbol{BA}^{(h)}$'s, $\forall i \in [N_h]$. We observe that the updates for all the heads share the same column space spanned by $\boldsymbol{B}$. In the extreme case of rank-1 $\boldsymbol{B}$ (for example), the updates for each column in the weight matrix will be in the same direction, which is not ideal for adapting all the heads. Further, as suggested by Hu et al. (2021), LoRA evenly assigns the parameter budgets to the weight matrices for $\boldsymbol{Q}$ and $\boldsymbol{V}$, which deviates from the second guideline.

4.2 Kernel-wise Adaptation

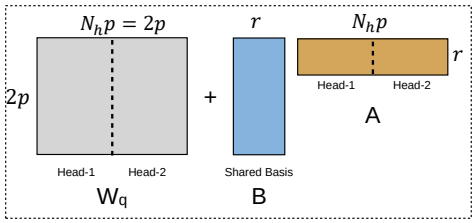
We choose LoRA as our primary base to develop our method because of its flexibility in assigning parameters to different weight matrices—both in empirical kernel matrix and coefficient components.

4.2.1 Guideline-1: Head-specific Adaptation

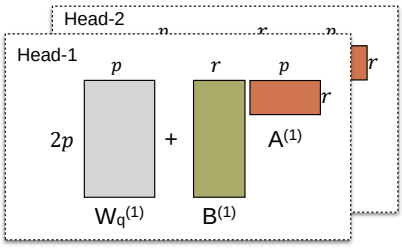
To incorporate Guideline-1, we extend the framework of LoRA and propose **Kernel-wise** [7] that sat-

[6] The case of $\boldsymbol{W}_v$ is similar to $\boldsymbol{W}_q$, while for $\boldsymbol{W}_o$, the role of $\boldsymbol{A}$ and $\boldsymbol{B}$ would be exchanged. For simplicity, we will always assume we are modifying $\boldsymbol{W}_q$ and discuss how to make $\boldsymbol{B}$ head-specific throughout the paper.

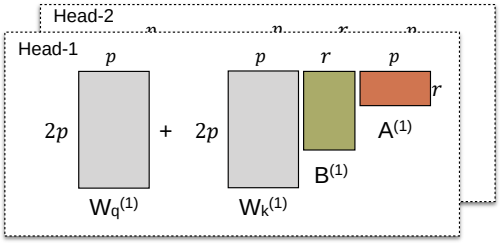
[7] Kernel-wise is a concrete scheme to adjust a specific weight matrix. It will have multiple variants modifying different weight matrices.



(a) LoRA



(b) Kernel-wise



(c) Kernel-wise-lite

Figure 1: (a) Original LoRa with two heads. (b) Kernel-wise variant of LoRa that satisfies Guideline-1 (c) lightweight version of (b) that uses $\boldsymbol{W}_k^{(h)}$ to reduce the trainable parameters in $\boldsymbol{B}^{(h)}$. Note that colored matrices are tunable, whereas gray-scale matrices are fixed (Best viewed in color).

isfies the first guideline.

Here, we allow the low-rank weight matrix updates for each head to have customized column spaces, by training distinct $\boldsymbol{B}^{(h)} \in \mathbb{R}^{N_h p \times r}$ for head-h, $\forall h \in [N_h]$. In this case, the weight matrix $\boldsymbol{W}^{(h)}$ in head-h would be updated by

$$\boldsymbol{W}^{(h)} \leftarrow \boldsymbol{W}^{(h)} + \boldsymbol{B}^{(h)}\boldsymbol{A}^{(h)},$$

and is expected to be more expressive due to the non-shared column spaces (see Figure 1(b)).

On the downside, this design suffers from inflexibility with a small parameter budget. For all the $\boldsymbol{B}^{(h)}$'s, to provide rank-r updates in each head, the new adaptation takes around $N_h^2 pr$ parameters. However, if, for instance, only $4N_h p$ parameters are assigned to modulate a weight matrix, we can still implement the original LoRA by using rank-2 $\boldsymbol{A}, \boldsymbol{B}$, while the construction of Kernel-wise would be prohibited since even rank-1 updates in each head will require more parameters than the budget.

To resolve the issue, we provide a lightweight alternative to the head-specific adaptation above, which we call **Kernel-wise-lite**. In this version, we propose to use the frozen $\boldsymbol{W}_k^{(h)} \in \mathbb{R}^{N_h p \times p}$ as the head-specific basis for head-h ($\boldsymbol{W}_k^{(h)}$ is the h-th block in the weight matrix $\boldsymbol{W}_k$, c.f. § 3.1). Therefore any target weight matrix $\boldsymbol{W}^{(h)}$ would be updated by

$$\boldsymbol{W}^{(h)} \leftarrow \boldsymbol{W}^{(h)} + \boldsymbol{W}_k^{(h)}\boldsymbol{B}_k^{(h)}\boldsymbol{A}^{(h)},$$

where $\boldsymbol{B}_k^{(h)}$ is a p-by-r matrix (see Figure 1(c)).

Utilizing this $\boldsymbol{W}_k^{(h)}$ with a smaller $\boldsymbol{B}_k^{(h)}$ allows the adaptation to be head-specific while containing the same number of trainable parameters as the original LoRA, which is $2N_h pr$. This comes at the cost of restricting the basis spaces of updates for each head from the unconstrained $\boldsymbol{B}^{(h)}$ to $\boldsymbol{W}_k^{(h)}\boldsymbol{B}_k^{(h)}$, where $\boldsymbol{W}_k^{(h)}$ is fixed.

But why should we choose $\boldsymbol{W}_k$ for the lightweight updates? The motivation comes from results in kernel learning that encourage adapting of the *coefficient* part using the basis spaces spanned by *key* matrices. In a kernel estimator, the coefficient $\boldsymbol{C}$ is independent of the query sequence $\boldsymbol{Q}$ as it is trained solely by the supporting points $\boldsymbol{K}$—asymptotically, c_j, the j-th row in $\boldsymbol{C}$, is only decided by $k_j, j \in [N]$ (Yang et al., 2017). Concretely, given the loss function and the kernel function, c_j is in general influenced by k_j and $\boldsymbol{K}_{-j}$ (all the points except k_j), while when $N \to \infty$, $\boldsymbol{K}_{-j}$ can be fully specified by a fixed distribution. This implies that in attention, compared to *queries*, *keys* are more related to *values*, and following which, we turn to $\boldsymbol{W}_k$ to form the basis for the low-rank updates for the conceptual motivation.

Combining LoRA with Kernel-wise. Our proposed adaptation can make fine-grained adjustments to each attention head and thus improve the representations. However, increased representation power might at times trade-off with lower-rank updates. As such, we propose our main variant—**Kernel-mix**, to combine the original LoRA and Kernel-wise, to attain the best of both worlds—larger basis (therefore higher rank updates) and specific adaptation to each head. Its update expression is as follows,

$$\boldsymbol{W}^{(h)} \leftarrow \boldsymbol{W}^{(h)} + (\boldsymbol{B}_{LoRA}, \boldsymbol{B}^{(h)}) \begin{pmatrix} \boldsymbol{A}_{LoRA}^{(h)} \\ \boldsymbol{A}^{(h)} \end{pmatrix},$$

where $\boldsymbol{B}_{LoRA}$ is shared among all the heads, while $\boldsymbol{B}^{(h)}$'s are head-specific. We remark Kernel-mix also has a lightweight alternative, **Kernel-mix-lite**, which is the combination of the original LoRA and Kernel-wise-lite. We compare different variants through experiments in § 6.

4.2.2 Guideline-2: Making Coefficients more Expressive

To incorporate Guideline-2, we propose to make coefficients more expressive by allowing the modification of both $\boldsymbol{W}_v$ and $\boldsymbol{W}_o$ for the coefficient part, compared to only updating $\boldsymbol{W}_v$ in the original LoRA. We achieve this by re-writing the attention sub-layer under the kernel estimator perspective, which extends the scope of attention by including $\boldsymbol{W}_o$ in its head-specific computation as well. If we represent $\boldsymbol{W}_o$ as,

$$\begin{pmatrix} \boldsymbol{W}_o^{(1)} \\ \vdots \\ \boldsymbol{W}_o^{(N_h)} \end{pmatrix},$$

where, the i-th block $\boldsymbol{W}_o^{(h)} \in \mathbb{R}^{p \times N_h p}$ corresponds to head-h (i.e., $\boldsymbol{L}^{(h)}\boldsymbol{V}^{(h)}$) in the attention output matrix, we can re-write the attention sub-layer as,

$$\sum_{h=1}^{N_h} \boldsymbol{L}^{(h)}\boldsymbol{V}^{(h)}\boldsymbol{W}_o^{(h)}, \tag{6}$$

and propose to take each summand as the complete form of a head (kernel estimator). We thus extend the coefficient part from the value matrices to the matrix products $\boldsymbol{V}^{(h)}\boldsymbol{W}_o^{(h)}$'s, which naturally results in N-by-$N_h p$ coefficients (with rank-p).

4.3 Final Model

Combining the two pieces together, we report the concrete adaptation scheme under two settings:

- With very limited parameter budgets (less than 0.2% of the total PLM parameters), similar to LoRA, we modify $\boldsymbol{W}_q$ and $\boldsymbol{W}_v$ with equal parameter budgets using Kernel-mix-lite(qv). (The suffix (qv) means the method will adjust $\boldsymbol{W}_q$ and $\boldsymbol{W}_v$.) In this case, we omit Guideline-2 and only incorporate Guideline-1 to apply head-specific updates.

- With intermediate parameter budgets (around 1.6% of the total parameters in the PLM), we suggest using Kernel-mix(qvo), which instead modifies $\boldsymbol{W}_q$, $\boldsymbol{W}_v$, and $\boldsymbol{W}_o$, assigning more budgets to the coefficient part (Guideline-2). Given the increased parameter budgets, we allow Kernel-mix scheme for $\boldsymbol{W}_q$ and $\boldsymbol{W}_o$, while continue to utilize Kernel-mix-lite scheme for $\boldsymbol{W}_v$. Kernel-mix(qvo) incorporates both the guidelines.

5 Experiments

While parameter-efficient tuning methods have been extensively studied for NLU tasks, their applicability towards NLG tasks is not well-known. This section performs empirical experiments of our proposed methods on three NLG tasks. To show the consistent effectiveness of our methods, we provide results on two NLU tasks as well. [8]

5.1 Experimental Setup

We mainly evaluate the performance of our methods on NLG tasks, in which there is still a gap between fine-tuning and most parameter-efficient adaptation techniques (He et al., 2021). Specifically, we conduct our experiments on the following datasets: WebNLG-challenge (Gardent et al., 2017) for table-to-text tasks, CoQA (Reddy et al., 2019) for conversational question answering, and CNN/Daily-Mail (CNN/DM) (Hermann et al., 2015) for summarization (SUM). CNN/DM has only one domain, while CoQA has five domains [9], and WebNLG has 14 domains. Descriptions of the datasets and evaluation metrics are provided in Appendix A.

Our experiments mainly follow the setting used by Lin et al. (2020), which takes GPT-2_{SMALL} (124M parameters) (Wolf et al., 2019) as the backbone for all the NLG tasks. We choose the smaller model size as compared to the larger models used in other related studies it is generally difficult for smaller models to attain the same performance as full-model fine-tuning (Lester et al., 2021; Liu et al., 2021b). This creates a challenging testbed to evaluate our proposed approaches.

In addition to the NLG experiments, we also study two NLU tasks: MNLI (Williams et al., 2018) and SST2 (Socher et al., 2013), to show the performance of our methods on encoder-only transformers. The Multi-Genre Natural Language Inference (MNLI) Corpus (sentence pairs of hypotheses and premises with entailment annotations) will be given, and the task is to predict whether

[8] The code release info will be available in this page.

[9] We use the official dev set as the test set and randomly select 500 examples from the train set as the new dev set.

	Parameters to train	Parameters to store	WebNLG BLEU ↑ S	BLEU ↑ U	BLEU ↑ A	WebNLG MET ↑ S	MET ↑ U	MET ↑ A	WebNLG TER ↓ S	TER ↓ U	TER ↓ A	CoQA EM	CoQA F1	SUM ROUGE-2
Full Budget = 100%														
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	15.72
Tiny Budget < 0.1%														
Adapter-4	0.08%	0.08%	51.7	35.6	44.4	0.38	**0.32**	**0.35**	0.44	0.58	0.51	49.9	58.8	14.18
Compacter	0.08%	0.08%	**53.3**	35.0	**45.0**	**0.39**	0.31	**0.35**	**0.43**	0.58	0.50	51.4	59.9	14.23
Bitfit	0.08%	0.08%	49.0	34.9	42.6	0.37	**0.32**	0.34	0.45	0.57	0.51	51.2	59.8	13.66
Kernel-mix-lite(qv)	0.07%	0.07%	51.0	**36.7**	44.5	0.38	**0.32**	**0.35**	**0.43**	**0.54**	**0.48**	**53.1**	**61.6**	**14.27**
Small Budget < 0.2%														
Adapter-8	0.14%	0.14%	**54.8**	36.4	**46.5**	**0.40**	**0.33**	**0.36**	0.42	0.58	0.49	51.9	60.7	14.42
Prefix-tuning-8	7.92%	0.12%	49.2	35.6	43.0	0.37	0.31	0.34	0.45	0.56	0.50	50.1	58.6	14.18
LoRA-4	0.13%	0.13%	52.8	37.1	45.8	0.39	**0.33**	**0.36**	0.42	0.55	**0.48**	**56.1**	**64.7**	14.42
Kernel-mix-lite(qv)	0.13%	0.13%	53.8	**37.2**	46.3	0.39	**0.33**	**0.36**	**0.41**	**0.54**	**0.48**	55.4	63.9	**14.52**
Intermediate Budget < 2%														
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	15.22
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	14.81
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	15.29
Kernel-mix(qvo)	1.61%	1.61%	**59.8***	36.7	**49.3***	**0.43***	**0.33**	**0.38**	**0.37***	0.57	**0.46***	**59.9***	**68.4***	**15.34***

Table 1: Performance (%) on NLG tasks [a]. The methods are divided into 4 groups based on the number of parameters to store, and the methods in the same group have similar sizes. We **boldface** the best score in each group for different metrics. In the group of intermediate budget, we further conduct the significance tests between Kernel-wise and the best baseline for each metric (* means the test p-value < 0.05).

[a] We follow the notations in prefix-tuning (Li and Liang, 2021) that S, U, A represent SEEN, UNSEEN, and ALL respectively; SEEN categories are used in training; UNSEEN categories only appear in the test set; and ALL consists of all the categories.

the premise entails, contradicts, or is neutral to the hypothesis; the Stanford Sentiment Treebank (SST2) is composed of movie reviews and corresponding human-annotated sentiment and specifies a task to predict the sentiment of a review sentence (positive/negative). We implement the backbone under the setting used by He et al. (2021) and use RoBERTa$_{\text{BASE}}$ (125M parameters) (Liu et al., 2019) for both MNLI and SST2.

5.2 Baselines

We compare our method with several other representative methods: fine-tuning (Howard and Ruder, 2018), adapters (Houlsby et al., 2019) used by Lin et al. (2020), Compacter (Mahabadi et al., 2021), Bitfit (Ben Zaken et al., 2021), prefix-tuning (Li and Liang, 2021), LoRA (Hu et al., 2021), and MAM-adapter (He et al., 2021). In Table 1 we use a postfix of adapters / LoRA / prefix-tuning to indicate their bottleneck size / rank of updates / prefix length, respectively. For instance, Adapter-4 means that the bottleneck size of the two-layer MLP in the inserted adapter is 4.

For some adaptation techniques, the number of parameters to train is not flexible to tune. For instance, Bitfit proposes to tune all the bias terms within the PLMs, and, as a result, the maximum parameters to tune are limited; as for Compacter, the weight matrices in the adapter modules are constructed through the Kronecker product, and the parameter complexity is $\mathcal{O}(\frac{L}{n} + n^3)$ (Mahabadi et al., 2021), where n is the size of a square matrix used in the Kronecker product and L is the number of layers. We choose a particular setting to make the number of trainable parameters in Compacter close to Bitfit and a tiny size adapter. The parameter size of Compacter is not further increased since a larger n would significantly retard the training.

For prefix-tuning, Li and Liang (2021) suggest to utilize a re-parametrization trick to mitigate its initialization issue, and therefore, the number of parameters to train will be much larger than the actual number of parameters to store, while these two numbers are the same for all other methods. In deciding the model size, we manage to make the number of parameters to **store** in prefix-tuning roughly the same as its adapter counterpart by adjusting the prefix length.

6 Results

6.1 Main Results

Table 1 compares our proposed methods against other baselines on the aforementioned generation tasks. The performance of our proposed Kernel-mix method on text classification is reported in Table 2. We summarize our observations as follows.

Tasks with long input. As shown in Table 1, all previous parameter-efficient methods fail to attain

comparable performance to fine-tuning on CoQA and CNN/DM tasks which have a longer input than the table-to-text generation task WebNLG. This indicates that current parameter-efficient methods still fall behind fine-tuning in those more challenging generation tasks with longer sequences. However, Kernel-mix, encloses this gap and even outperforms fine-tuning in some NLG tasks, e.g. the CoQA task, with solely 1.61% tunable parameters of GPT-2_{SMALL}.

Impact of parameter sizes. The results in Table 1 show that overall, as the tunable parameter size increases, the performance of various parameter-efficient methods also increases, getting closer to fully fine-tuning. This indicates that a large enough parameter budget is still a prerequisite for the excellent performance of parameter-efficient adaptation methods. This finding can help us explain why the performance of Compacter can be better than Adapter-4 over all the tasks when the parameter budget is tiny (*Tiny Budget* $< 0.1\%$ in the second group in Table 1), considering that the Compacter can construct a larger MLP than the adapter with the same parameter budget due to the usage of Kronecker product. Inspired by this finding, we can also clearly see the limitation of Bitfit and Compacter as their parameter budgets are constrained to be small and cannot be elevated of free will. As for the original LoRA, the impact of parameter sizes is somewhat tricky—LoRA-4 shows competitive performance while LoRA-54 is not improved as greatly as other methods on WebNLG and CoQA. A similar phenomenon on different datasets is also observed by Hu et al. (2021); He et al. (2021).

Performance of our proposed Kernel-mix(qvo). Our proposed Kernel-mix(qvo) can generally improve the performance on all three NLG datasets. On WebNLG, Kernel-mix(qvo) provides a 1.1 increase in BLEU score compared to Adapter-108 and a 2.6% increase compared to LoRA-54; on CoQA, our method is even more greatly better than LoRA-54, obtaining 2.7 exact-match and F1 improvement, and even outperforms fine-tuning by around 1% in both of the metrics; On CNN/DM, all the parameter-efficient methods have close performance, while through a test, we show our method Kernel-mix has a significantly higher Rouge-2 score than the best baseline, LoRA-54. Overall, Table 1 demonstrates that kernel-mix adaptation better exploits the attention structure in PLMs and improves the overall generation quality under all three kinds of parameter budgets.

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%)	$87.4_{\pm.3}$	$94.2_{\pm.3}$
Kernel-mix(qvo) (0.5%)	$87.4_{\pm.2}$	$94.3_{\pm.3}$

Table 2: Accuracy on the dev set of MNLI and SST2. Bitfit numbers are copied from Ben Zaken et al. (2021), and all the other results (except for Kernel-mix) are from He et al. (2021, Table 2).

Performance on NLU tasks. Table 2 shows the performance of Kernel-mix(qvo) when it is extended to encoder-only transformers. For a fair comparison, we specify a new parameter budget (0.5%) for Kernel-mix(qvo), different from the previous settings in Table 1. With the new budget, Kernel-mix(qvo) attains close accuracy to the other parameter-efficient methods on both MNLI and SST2. We remark that the parameter budget used here is slightly tight for Kernel-mix(qvo), as the ranks assigned for head-wise adaptation (1 for W_q and 2 for W_v, W_o) are limited (see Table B.5).

6.2 Ablation Studies

Besides the main results in Table 1, we also perform ablation studies to verify the effectiveness of our propositions. We additionally implement four variants of Kernel-wise to help ablate the effects of our proposed guidelines. Among the new variants, Kernel-wise-lite(qv), Kernel-wise(mq), and Kernel-wise(mv) only adjust $\boldsymbol{W}_q$, $\boldsymbol{W}_v$; Kernel-wise-lite(qv) takes the strategy in LoRA-4 to evenly assign parameters to $\boldsymbol{W}_q$ and $\boldsymbol{W}_v$; Kernel-wise(mq) leaves more budget to $\boldsymbol{W}_q$ than $\boldsymbol{W}_v$ with a ratio of 3:1, while Kernel-wise(mv) is set up in the reversed way. In contrast, Kernel-wise(qvo) simultaneously adjusts $\boldsymbol{W}_q$, $\boldsymbol{W}_v$, and $\boldsymbol{W}_o$ (with a budget ratio of 5:1:10). The experimental results are summarized in Table 3. The settings of the variants designed for ablation are described in Appendix B.4.

Head-specific adaptation (Guideline-1). For "LoRA-4", the setting recommended by Hu et al. (2021), we compare it with its head-specific

	Parameters to train	Parameters to store	WebNLG BLEU ↑ S	U	A	MET ↑ S	U	A	TER ↓ S	U	A	CoQA EM	CoQA F1	SUM ROUGE-2
Budget $< 0.2\%$														
LoRA-4	0.13%	0.13%	52.8	37.1	45.8	0.39	0.33	0.36	0.42	0.55	0.48	56.1	64.7	14.42
Kernel-wise-lite(qv)	0.13%	0.13%	**55.1**	36.9	**46.9**	**0.40**	0.33	**0.37**	**0.40**	0.55	**0.47**	55.0	63.6	**14.48**
Budget $< 2\%$														
Kernel-wise(mq)	1.56%	1.56%	58.9	36.7	48.9	0.42	0.33	0.38	0.37	0.57	0.46	57.4	66.3	15.22
Kernel-wise(mv)	1.56%	1.56%	59.4	37.2	49.3	0.42	0.33	0.38	0.37	0.57	0.46	58.0	67.0	15.24
Kernel-wise(qvo)	1.61%	1.61%	59.5	36.1	49.0	0.43	0.33	0.38	0.37	0.58	0.47	59.1	68.0	15.28
Kernel-mix(qvo)	1.61%	1.61%	59.8	36.7	49.3	0.43	0.33	0.38	0.37	0.57	0.46	59.9	68.4	15.34

Table 3: Performance on new variants of Kernel-wise compared to LoRA-4 and our proposed Kernel-mix(qvo) (both copied from Table 1). The methods with similar budgets of tunable parameters are grouped. The exact settings of the methods to compare are illustrated in § 6.2 and Appendix B.4.

counterpart—Kernel-wise-lite(qv). In almost all the tasks, Kernel-wise-lite(qv) can attain better performance with the same number of parameters.

More parameters for the coefficient part (Guideline-2). Hu et al. (2021) (Section 7.1) indeed have already done some preliminary exploration to find the relatively more important weight matrices in transformers. Their experimental results (copied as Table C.6 in Appendix C) clearly show that "putting all the parameters in $\Delta \boldsymbol{W}_q$ or $\Delta \boldsymbol{W}_k$ results in significantly lower performance". In this work, we additionally show that by simply moving some trainable parameters from $\boldsymbol{Q}$, the empirical kernel matrix part, to $\boldsymbol{V}$, the coefficient part, Kernel-wise(mv) can improve the performance upon Kernel-wise(mq) as well.

Extending the scope of attention. To show the benefits of both adjusting $\boldsymbol{W}_v$ and $\boldsymbol{W}_o$, we compare the new variant Kernel-wise(mv) against Kernel-wise(qvo). They both assign more budgets to the coefficient part; Kernel-wise(qvo) would update both $\boldsymbol{W}_q$, $\boldsymbol{W}_v$ and $\boldsymbol{W}_o$, while Kernel-wise(mv) only adjusts $\boldsymbol{W}_q$, $\boldsymbol{W}_v$. We can observe that Kernel-wise has better performance in most tasks.

Combining the shared and the head-specific basis. Lastly, we find that Kernel-mix(qvo) (our proposed method) outperforms Kernel-wise(qvo), which justifies combining the two types of basis, as opposed to pure head-specific adaptation.

7 Conclusion and Future Work

In this work, we revisit the connection between the attention module and kernel estimators, and accordingly propose kernel-wise adaptation, which adopts the guidelines from kernel learning to strengthen the low-rank adaptation (LoRA). We verify that with the same parameter budgets, our proposed adaptation techniques can have better performance on three generation tasks than the existing parameter-efficient methods, including adapters, prefix-tuning, and LoRA, and attain close accuracy on two classification tasks as well.

One possible extension of our work is combining our proposed method with other adapters in feed-forward sub-layers. In MAM-adapter, He et al. (2021) suggest applying prefix-tuning to adapt the parameters in self-attention sub-layers and assigning budgets to feed-forward sublayers as well; it can be beneficial to replace prefix-tuning with Kernel-mix for adaptation in the attention part.

Another direction of future research is the extension of our method to the feed-forward sub-layers, which are interpreted as key-value memories in recent work and behave like attention blocks (Geva et al., 2021). It will be interesting to study if kernel-specific guidelines could help design better adapters employed in the feed-forward layers.

Acknowledgements

We appreciate all the valuable feedback from the anonymous reviewers.

References

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv e-prints*, pages arXiv–2106.

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, 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*.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. The webnlg challenge: Generating text from rdf data. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 124–133.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495.

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

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28:1693–1701.

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*.

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

Dan Klein and Christopher D Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st annual meeting of the association for computational linguistics*, pages 423–430.

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.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

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.

Peiyu Liu, Ze-Feng Gao, Wayne Xin Zhao, Zhi-Yuan Xie, Zhong-Yi Lu, and Ji-Rong Wen. 2021a. Enabling lightweight fine-tuning for pre-trained language model compression based on matrix product operators. *arXiv preprint arXiv:2106.02205*.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021b. 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. 2021c. 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*.

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. 2021. Compacter: Efficient low-rank hypercomplex adapter layers. *arXiv preprint arXiv:2106.04647*.

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*.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020. 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.

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*.

Bernhard Schölkopf, Ralf Herbrich, and Alex J Smola. 2001. A generalized representer theorem. In *International conference on computational learning theory*, pages 416–426. Springer.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. Eliciting knowledge from language models using automatically generated prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235.

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.

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.

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.

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.

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.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Yun Yang, Anirban Bhattacharya, and Debdeep Pati. 2017. Frequentist coverage and sup-norm convergence rate in gaussian process regression. *arXiv preprint arXiv:1708.04753*.

A Dataset Details

- The **WebNLG** dataset consists of mapping sets of RDF triples to text. The training data are Data/Text pairs where the data is a set of (subject, property, object) triples. There are 9 categories extracted from DBpedia in the train and the development (dev) set, while the test set contains 5 more unseen categories, which can be used to evaluate the generalization of the adaptation methods. We adopt the official evaluation script and reports BLEU (Papineni et al., 2002), METEOR, (Lavie and Agarwal, 2007) and TER (Snover et al., 2006) [10].

- **CoQA** is a large-scale conversational question answering dataset. It contains over 127K questions with answers collected from more than 8K conversations. The problem involves generating answers to the questions based on related conversation histories and documents. We follow the official evaluation script and use the macro-average F1 score of word overlap as the main evaluation metric (Reddy et al., 2019).

- **CNN/DM** is a benchmark for text summarization, involving more than 300K news articles provided by CNN and the Daily Mail. We report ROUGE-2 scores (Lin, 2004) as evaluation metrics.

- **MNLI** The Multi-Genre Natural Language Inference Corpus (Williams et al., 2018) provides sentence pairs of hypotheses and premises with entailment annotations. There are $393k$ pairs in the training set, $10k$ in the dev set, and another $10k$ pairs in the test set. (Only the dev set is used in Table 2.) The premise sentences come from ten different sources, and the model performance can be evaluated on both the matched (in-domain) and mismatched (cross-domain) sections. In Table 2, we follow the setting used by Hu et al. (2021) and report mismatched accuracy as the metric.

- **SST2** The Stanford Sentiment Treebank (Socher et al., 2013) is a corpus with fully labeled parse trees. In this corpus, $11,855$ single sentences extracted from movie reviews were parsed with the Stanford parser (Klein and Manning, 2003), generating $215,154$ unique phrases from those parse trees (3 human judges annotate each phrase). Wang et al. (2019) incorporated the task into the GLUE benchmark, with $67k$ sentences in the training set, $0.9k$ in the dev set, and $1.8k$ instances in the dev set. (Only the dev set is used in Table 2.) The metric is the accuracy of the decision whether the sentiment of a review sentence is positive or negative.

B Training Details

B.1 General Training Settings

We avoid the sentence-level knowledge distillation trick used in (Lin et al., 2020), as it might interfere in analyzing our hypotheses. However, we retain the usage of "task embeddings" as they are required by the original GPT-2 model. These task embeddings act as specialized segment embeddings that indicate the different components of the text input (e.g., the three components of a triple in NLG, questions and answers in CoQA, etc.). [11]

We state the specific task embedding used in each task. For CoQA and CNN/DM, we follow the task embedding suggested by Lin et al. (2020); for WebNLG, we similarly set up the special tokens for the different components in the triples. We conclude the details of the special tokens in each dataset in Table B.4. Notably, the parameter budget for task embedding is neglectable compared to the size of the aforementioned parameter-efficient adaptation techniques.

B.2 Hyper-parameters

We apply an AdamW optimizer and a linear learning rate scheduler with a 500-step warmup duration in training. At generation time, we use a greedy search for all the tasks, the same as Lin et al. (2020). For the choice of some hyperparameters, we mainly follow the setting used by Lin et al. (2020); Hu et al. (2021) and He et al. (2021), including the number of epochs and the argument for weight decay. Specifically, for WebNLG, CNN/DM, MNLI, and SST2, we train the model for 10 epochs; for CoQA, we train the model for 5 epochs. For the other important hyper-parameters, such as batch size and learning rate, we tune the hyper-parameters for different methods according to the loss on the validation set. For our Kernel-mix methods in Table 1, they share the same batch size and learning rate in each task. Specifically, for WebNLG, the learning

[10] For TER, the lower the metric is, the better the performance is.

[11] The task embedding for the special tokens will also be updated during training, while 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>	$5 * 768 = 3840$
CNN/DM	<bos_sm>, <eos_sm>, <source_sm>, <target_sm>	$4 * 768 = 3072$

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

rate we use is 0.00125, and the batch size is 16; for CoQA, the learning rate we use is 0.005, and the batch size is 8; for CNN/DM, the learning rate we use is 0.001, and the batch size is 16; for MNLI, the learning rate we use is 0.0002, and the batch size is 32; for SST2, the learning rate we use is 0.0001, and the batch size is 16;

We train each variant for multiple independent runs to account for variability. In particular, for WebNLG, we train models over 5 runs, for CoQA 3 runs, for CNN/DM 2 runs, for MNLI 3 runs, and for SST2 3 runs.[12] The reported numbers in Tables 1 and 2 are the mean value averaged over the runs.

B.3 Implementation and Training Efficiency

All the models in this work are implemented by PyTorch. For the devices, we perform the distributed training using 8 Tesla V100 16GB GPUs. On WebNLG, it will take our method around 1 minute to finish one epoch; on CoQA, the time cost is around 20 minute / epoch; on CNN/DM, the training time per epoch will be 30 minutes. For the NLU tasks, the task implementation by He et al. (2021) cannot be adapted to distributed training (can only be trained with one graphic card), and thus the training time is longer: it will take our method around 2 hours to finish one epoch in MNLI, and 25 minutes in SST2.

We additionally report there is actually an implementation trick in Kernel-wise-lite. We can simply associate the h^{th} head's *key* matrix $\boldsymbol{K}^{(h)}$ to the computation of any weight matrix, say $\boldsymbol{Q}^{(h)}$, as follows,

$$\begin{aligned}\boldsymbol{Q}^{(h)} &= \boldsymbol{X}\boldsymbol{W}_q^{(h)} + \boldsymbol{X}\boldsymbol{W}_k^{(h)}\boldsymbol{B}_k^{(h)}\boldsymbol{A}^{(h)} \\ &= \boldsymbol{X}\boldsymbol{W}_q^{(h)} + \boldsymbol{K}^{(h)}\boldsymbol{B}_k^{(h)}\boldsymbol{A}^{(h)}.\end{aligned}$$

In that case we can reuse the given $\boldsymbol{K}$ to save the computation of the product $\boldsymbol{X}(\boldsymbol{W}_k^{(h)}\boldsymbol{B}_k^{(h)}\boldsymbol{A}^{(h)})$.

[12] We reduce the number of runs for larger datasets given computation budget.

B.4 Specific settings for each method

We report the exact setting for the methods that need further explanation in this subsection. For Compacter, the bottleneck size of the adapter is 192, and the number of components is 4, as suggested by Karimi Mahabadi et al. (2021); for the original LoRA and the variants of our proposed methods, we summarize their settings in Table B.5. In this table, the numbers in columns Q_wise, V_wise, and O_wise are the rank used for Kernel-wise; if the number is followed by "(lite)", we apply Kernel-wise-lite with the listed rank to adjust the corresponding weight matrices. The numbers in columns Q_LoRA, V_LoRA, and O_LoRA are the rank of the update used as in the original LoRA. For Kernel-mix methods, the numbers in columns Q_wise and Q_LoRA (for example) will be non-zero.

C Partial Experimental Results Reported in LoRA (Hu et al., 2021)

For ease of reading, we copy Table 5 from the paper (Hu et al., 2021) as a piece of evidence to show "putting all the parameters in $\Delta\boldsymbol{W}_q$ or $\Delta\boldsymbol{W}_k$ results in significantly lower performance".

	Budget	Q_wise	Q_LoRA	V_wise	V_LoRA	O_wise	O_LoRA
LoRA-4	small	0	4	0	4	0	0
LoRA-54	intermediate	0	4	0	4	0	0
Kernel-mix-lite(qv)	tiny	1 (lite)	1	1 (lite)	1	0	0
Kernel-mix-lite(qv)	small	2 (lite)	2	1 (lite)	2	0	0
Kernel-mix(qvo)	intermediate	3	12	8 (lite)	8	8	8
Kernel-wise-lite(qv)	small	4 (lite)	0	4 (lite)	0	0	0
Kernel-wise(mq)	intermediate	12	0	4	0	0	0
Kernel-wise(mv)	intermediate	4	0	12	0	0	0
Kernel-wise(qvo)	intermediate	5	0	10 (lite)	0	10	0
Kernel-mix(qvo)	0.5% in Table 2	1	2	2 (lite)	4	2	4

Table B.5: The exact settings for the original LoRA and the variants of our proposed methods.

	# of Trainable Parameters $= 18\text{M}$						
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
Rank r	8	8	8	8	4	4	2
WikiSQL $(\pm 0.5\%)$	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI $(\pm 0.1\%)$	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

Table C.6: Validation accuracy provided by Hu et al. (2021, Table 5) on WikiSQL and MultiNLI.