MITSUBISHI ELECTRIC RESEARCH LABORATORIES
https://www.merl.com

# LoDA: Low-Dimensional Adaptation of Large Language Models

Liu, Jing; Koike-Akino, Toshiaki; Wang, Pu; Brand, Matthew; Parsons, Kieran; Wang, Ye

TR2025-130 September 03, 2025

**Abstract**

Parameter-Efficient Fine-Tuning (PEFT) has recently garnered significant attention, due to the enormous size of Large Language Models (LLMs). Among various PEFT methods, Low-Rank Adaptation (LoRA) demonstrates comparable performance to full fine-tuning, despite having significantly fewer trainable parameters. In this work, we first generalize LoRA from a low-rank linear adaptation/mapping to low-dimensional, non-linear adaptation/mapping, which we have named Low- Dimensional Adaptation (LoDA). We also propose LoDA+, which further improves the expressiveness of the non-linear adaptation, while still using nearly the same number of tunable parameters as LoRA. Both LoDA and LoDA+ include LoRA as a special case. To improve computational efficiency at inference, we further propose R-LoDA(+) and S-LoDA(+), by replacing the pre-trained weight matrix with its low-rank or sparse approximation, which is frozen during fine-tuning. Empirical evaluations on Natural Language Generation tasks demonstrate that variants of LoDA outperform LoRA and other baselines.

*Springer Book 2025*



Mitsubishi Electric Research Laboratories, Inc.
201 Broadway, Cambridge, Massachusetts 02139

# LoDA: Low-Dimensional Adaptation of Large Language Models

Jing Liu, Toshiaki Koike-Akino, Pu (Perry) Wang, Matthew Brand, Kieran Parsons, and Ye Wang

**Abstract** Parameter-Efficient Fine-Tuning (PEFT) has recently garnered significant attention, due to the enormous size of Large Language Models (LLMs). Among various PEFT methods, **Lo**w-**R**ank **A**daptation (LoRA) demonstrates comparable performance to full fine-tuning, despite having significantly fewer trainable parameters. In this work, we first generalize LoRA from a low-rank linear adaptation/mapping to low-dimensional, non-linear adaptation/mapping, which we have named **Lo**w-**D**imensional **A**daptation (LoDA). We also propose LoDA+, which further improves the expressiveness of the non-linear adaptation, while still using nearly the same number of tunable parameters as LoRA. Both LoDA and LoDA+ include LoRA as a special case. To improve computational efficiency at inference, we further propose R-LoDA(+) and S-LoDA(+), by replacing the pre-trained weight matrix with its low-rank or sparse approximation, which is frozen during fine-tuning. Empirical evaluations on Natural Language Generation tasks demonstrate that variants of LoDA outperform LoRA and other baselines.

## 1 Introduction

Large language models (LLMs), such as ChatGPT [1], Gemini [2], Claude 3 [3], and LLaMA2 [4], have shown great promise in generating human-like text and have sparked excitement about their potential applications across various industries. The sizes of large language models (LLMs) have grown at an unprecedented rate, with current models boasting parameter counts in the hundreds of billions or even trillions, necessitating massive amounts of computational resources for training and inference. Recent studies show that the performance of a Pre-trained Language Model (PLM) can be significantly improved by further fine-tuning on domain-specific data [5]. Therefore, fine-tuning PLMs for domain-specific tasks has become the *de*

Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA 02139, USA
e-mail: {jiliu,koike,pwang,brand,parsons,yewang}@merl.com

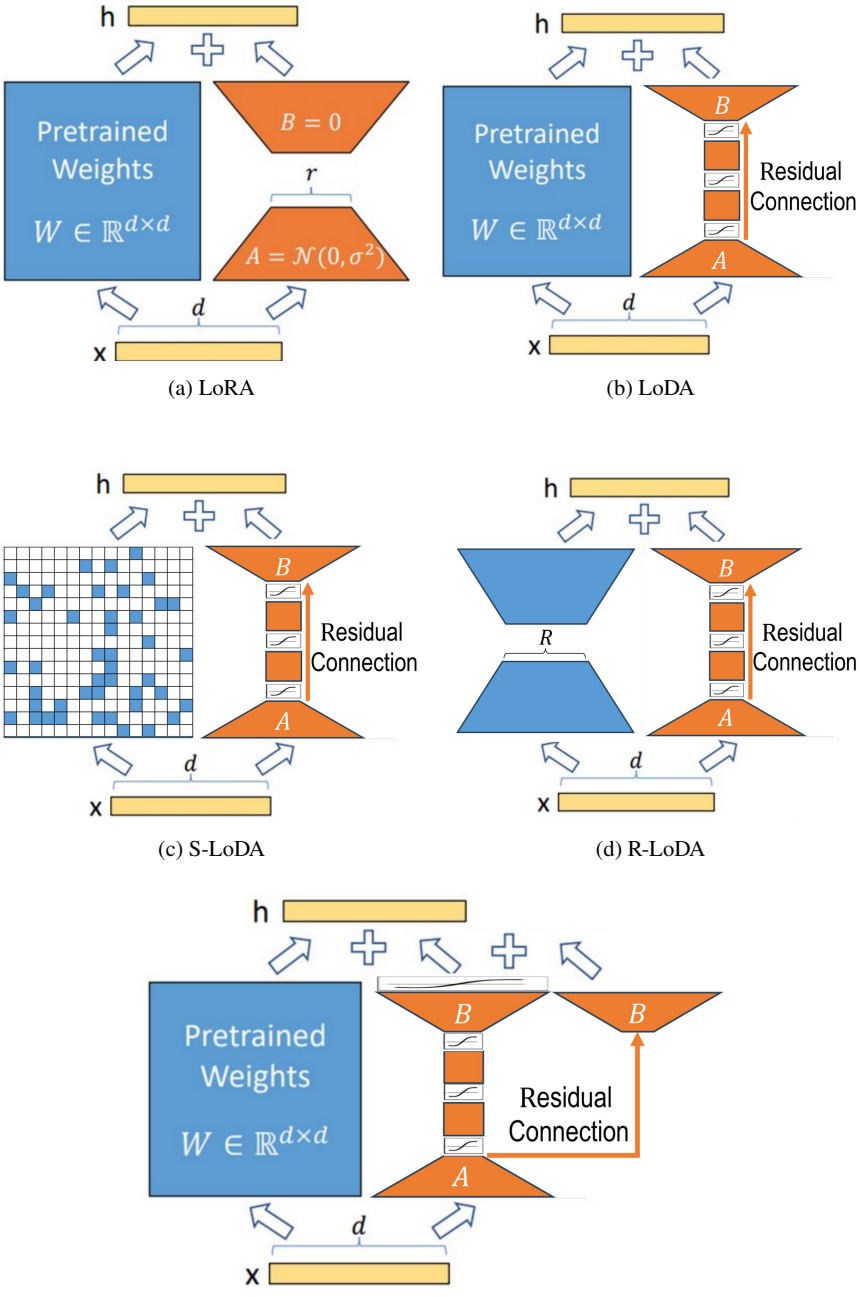
*facto* procedure. However, full fine-tuning of such LLMs is still very expensive. For instance, fine-tuning a 65 billion-parameter model requires more than 780 GB of memory [6]. Parameter-Efficient Fine-Tuning (PEFT) fine-tunes only a small set of parameters, which may be a subset of the existing model parameters or a set of newly added parameters, thereby greatly reducing the computational and memory costs. Another advantage of PEFT is that, in addition to the pre-trained model, only a small number of (extra) model parameters need to be stored for each fine-tuned task. While for multiple downstream tasks, PEFT greatly saves the storage, while full fine-tuning needs to generate a new large model for each downstream task.[1]. Besides parameter savings, PEFT makes it possible to quickly adapt to new tasks without catastrophic forgetting [7], which has often been observed during the full fine-tuning of LLMs. PEFT approaches have also been shown to be better than full fine-tuning in low-data regimes [8, 5]

Therefore, many PEFT methods have been proposed. For example, prefix tuning [8] and prompt tuning [9] prepend some tunable prefix tokens to the input or hidden layers, and only train these soft prompts during fine-tuning. Several adapter tuning methods [10, 11, 7, 12] insert (and tune) small neural modules called adapters to some layers of the PLM. More recently, [5] propose to use low-rank decomposition matrices to approximate the parameter update of the weight matrix of a dense layer. In particular, they propose to update the Query and Value projection matrices in the Transformer architecture, which shows promising performance and has become a popular PEFT tool for LLMs in modern libraries, e.g., Hugging Face PEFT library [13]. For a comprehensive review and comparison, we refer the interested readers to recent surveys [14, 15, 16, 17].

LoRA is motivated by the hypothesis that the change in the model (to adapt to a related downstream task) is intrinsically low-dimensional [18]. LoRA constrains the change in weights during model adaptation to be low-rank, leading to the **Lo**w-**R**ank **A**daptation (LoRA) approach [5]. For a dense layer of the PLM, its original weight parameters, e.g., $W \in \mathbb{R}^{d\times d}$ (blue part of Figure 1a) is frozen. During fine-tuning, LoRA uses low-rank decomposition matrices $A \in \mathbb{R}^{d\times r}$ and $B \in \mathbb{R}^{r\times d}$ to constrain the weight update $\Delta W = AB$ (orange part of Figure 1a). As the rank $r$ is typically set to be very small, the number of parameters in $A$ and $B$ are significantly less than the original $W$.

We view the neural network as a function, and the change in the neural network during the task adaptation can be more generally viewed as the change in the functional mapping. Let the input to the dense layer be denoted by $x$, and the output of the pre-trained dense layer denoted by $h_0 = xW$. After LoRA fine-tuning of that dense layer, the new output $h'_{\text{LoRA}} = h_0 + \Delta h_{\text{LoRA}}$, where $\Delta h_{\text{LoRA}} = xAB$. Thus, the mapping from input $x$ to the update $\Delta h_{\text{LoRA}} = xAB$ by LoRA is a low-rank (i.e., $r$-dimensional) linear mapping.

[1] For example, in [5], a GPT-3 175B model, of size 350GB, is fine-tuned with a rank-4 LoRA adapter that requires only 35MB for each downstream task. Storing 100 adapted models requires only 350GB + 35MB $\times$ 100 $\approx$ 354GB as opposed to 350GB $\times$ 100 $\approx$ 35TB for full fine-tuning over 100 tasks.




(a) LoRA (b) LoDA

(c) S-LoDA (d) R-LoDA

(e) LoDA+

Fig. 1: Overview of (a) LoRA; (b) LoDA; (c) S-LoDA; (d) R-LoDA; (e) LoDA+. The blue part is frozen during fine-tuning, and only the orange part is trained. In LoDA+, there is essentially only one matrix $B$, but the non-linear part has additional non-linear operations after $B$ (e.g., LeakyReLU).

Although the update of the mapping $x \rightarrow \Delta h$ likely has an intrinsic low dimension for task adaptation, the linear low-rank constraint imposed by LoRA might be too restrictive, and therefore we aim to relax it to a more general low-dimensional, non-linear constraint. [19] also argue that the linear adaptation may limit the learning capacity of LoRA. Consequently, we propose to extend LoRA to a more general **Lo**w-**D**imensional **A**daptation (LoDA), which will be detailed in the next section.

**Notation** We follow conventional terminologies for the Transformer architecture, where $d_{\text{model}}$ denotes the input/output dimension of a Transformer block. We use $W_{\text{q}}$, $W_{\text{k}}$, and $W_{\text{v}}$ to respectively refer to the query, key, and value projection matrices of a self-attention module.

## 2 Proposed Methods

One key question is how to design and realize a more general low-dimensional, non-linear constraint than the linear low-rank constraint of LoRA, while keeping LoRA as a special case. We propose a deep neural network architecture for LoDA as illustrated in Figure 1b. The low-dimensional non-linear constraint is realized by a multi-layer neural network within the bottleneck structure (to maintain parameter efficiency) and a residual connection between matrices $A$ and $B$. It can be considered as a non-linear version of LoRA with the non-linear mapping $x \rightarrow \Delta h_{\text{LoDA}} \doteq f_{\text{LoDA}}(x)$.

Mathematically, with our proposed residual connection architecture of LoDA in Figure 1b, we have

$$\Delta h_{\text{LoDA}} = f_{\text{LoDA}}(x) = xAB + f_1(xA)B, \tag{1}$$

where $f_1(\cdot)$ is a non-linear function between matrix $A$ and matrix $B$, which consists of a series of linear layers and non-linear operations (e.g., LeakyReLU activation, layer-normalization). Note that Figure 1b is merely an example of a LoDA structure. For instance, the non-linear part between matrix $A$ and matrix $B$ could have more layers than illustrated, and may use non-square matrices. LoRA is a special case of LoDA if $f_1(xA)B$ is zero (e.g., a hidden layer's weights in LoDA are zero) or if $f_1(\cdot)$ is linear, for example.

### 2.1 Extension to LoDA+

Although LoDA generalizes LoRA from a low-rank linear mapping/adaptation to a low-dimensional, non-linear mapping/adaptation, and keeps LoRA as a special case, the image of such a non-linear mapping still lies in a low-dimensional linear subspace (i.e., the range of matrix $B$). *Is it possible to further generalize that to a low-dimensional (non-linear) manifold, while keeping LoRA as a special case, and using almost the same number of tunable parameters as LoRA?*

We further propose the following mapping for LoDA+, illustrated in Figure 1e, that affirms the possibility:

$$\Delta h_{\text{LoDA+}} = f_{\text{LoDA+}}(x) = xAB + f_2(f_1(xA)B) \tag{2}$$

Note that the key difference between LoDA and LoDA+ is the additional non-linear function $f_2(\cdot)$, e.g., non-linear activation and/or layer-normalization. With this additional non-linear function, the image of the mapping $f_{\text{LoDA+}}$ becomes the combination of a linear subspace (the first term in Eq. 2) and a non-linear manifold (the second term in Eq. 2). For convenience, we will use LoDA(+) to represent both 'LoDA and LoDA+'.

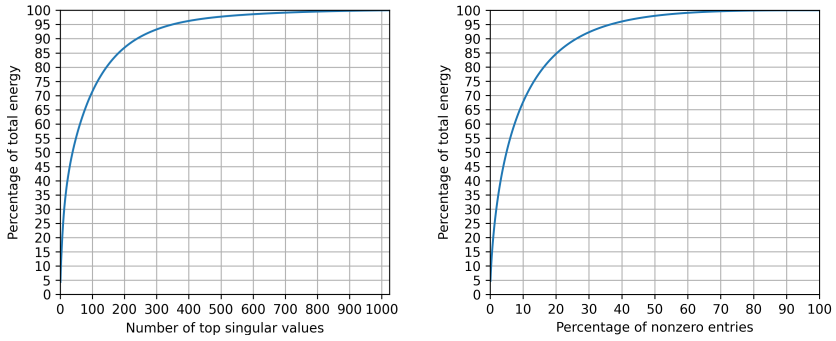
**Viewing LoDA(+) as Deep Parallel Adapters:** [20] viewed LoRA as a parallel adapter. Similarly, the proposed LoDA can be viewed as a *deep* parallel adapter. Recent work [20, 21] propose to use the traditional shallow adapter in a parallel fashion instead of the usual sequential fashion. The shallow adapter there only has a down-projection layer, followed by a non-linear activation function (typically ReLU), then an up-projection layer, and the adapter attaches to the input and output of the Attention module or the Feed-Forward Network module of a Transformer block in an LLM. We refer the interested readers to [22, Figure 1] and [20, Table 1] for more details. In contrast, the proposed LoDA, which aims at learning a low-dimensional, non-linear mapping, has a *deep* structure to capture the underlying nonlinearity. Further, in LLMs, LoDA and LoRA are attached to $W_{\text{q}}$ and $W_{\text{v}}$, but not attached to the whole Attention module nor to the Feed-Forward Network module of the Transformer block. Also, it is interesting to note that LoDA has a Residual Connection *inside*, that is between the output of matrix $A$ and the input of matrix $B$ (see Figure 1b), which is different from the existing adapters. More interestingly, LoDA+ can be viewed as a deep+shallow dual parallel adapter, where the shallow and deep parts correspond to the first and second terms in Eq. 2 respectively.

## 2.2 S-LoDA(+) and R-LoDA(+)

As the dimension of the non-linear layers in LoDA(+) is restricted to a very small value $r$, the additional computational cost during the inference is very small (see Section 3.2 for more details). Further, as LoDA(+) runs in parallel with the pre-trained weight matrix $W$, it will not introduce apparent delay in the overall inference with parallelization, unlike the sequential adapters in the literature. With LoDA(+), the main computational bottleneck is still $W$ of the PLM. *Is it possible to further improve the computational efficiency of a LoDA(+) fine-tuned model, and even significantly better than the pre-trained model?*

We observe that the combined projection matrix $W_{\text{proj}} = [W_{\text{q}}, W_{\text{k}}, W_{\text{v}}]$, inside the Attention module of GPT2-medium (with size $1024 \times 3072$), can be well-approximated by a relatively low-rank matrix or a relatively sparse matrix. More specifically, Figure 2a shows the percentage of total energy (i.e., $\sum_{i=1}^{R} \sigma_i^2 / \sum_{i=1}^{1024} \sigma_i^2$)

with respect to the number of top singular values $R$ of $W_{\text{proj}}$ in the first Transformer block of GPT2-medium. We see that using only the top 300 singular value components of $W_{\text{proj}}$ preserves over 93% of its total energy. Figure 2b shows the percentage of total energy w.r.t. the percentage of nonzero entries of $W_{\text{proj}}$ (by zeroing out smaller magnitude weights). We see that keeping 40% of the larger magnitude entries of $W_{\text{proj}}$ can preserve over 96% of its total energy.




(a) (b)

Fig. 2: (a) Percentage of total energy w.r.t. the number of top singular values of $W_{\text{proj}}$. (b) Percentage of total energy w.r.t. the percentage of nonzero entries of $W_{\text{proj}}$ (by zeroing out smaller magnitude weights).

The aforementioned questions and observations motivate us to further propose R-LoDA(+) and S-LoDA(+), which are LoDA(+) combined with the low-**R**ank or **S**parsified approximations of $W$, which are frozen during fine-tuning, while the adapter is trained/fine-tuned for this approximate $W$. See Figure 1c and Figure 1d for illustrations[2]. Our empirical investigation shows that even when the pre-trained projection matrix $W_{\text{proj}}$ is low-rank approximated or sparsified, combining with LoDA(+) can still achieve competitive performance. More importantly, the total inference cost of the R-LoDA(+) or S-LoDA(+) fine-tuned model can be significantly lower than the pre-trained model. The analysis of the computational costs can be found in Section 3.2, and Table 2 in Section 4 also demonstrates the significant computational savings for the GPT2-medium model.

[2] If applying R-LoDA (or S-LoDA) on $W_{\text{q}}$ and $W_{\text{v}}$, one could approximate $W_{\text{q}}$ and $W_{\text{v}}$ separately, but we directly approximate the whole $W_{\text{proj}} = [W_{\text{q}}, W_{\text{k}}, W_{\text{v}}]$ to make the model inference more efficient.

# 3 Discussion

## 3.1 Number of Fine-Tuning Parameters

The number of trainable parameters of LoRA and the proposed methods are determined by the bottleneck dimension $r$ and the shape of the original weights. More specifically, in Figure 1, the matrices $A$ and $B$ across all methods have dimensions $d$ times $r$. The proposed methods have two additional $r$ by $r$ bottleneck matrices. Their non-linear activation function is LeakyReLU with a fixed slope of 0.8, and their layer-normalization is not trainable. So, the total number of trainable parameters for LoRA is $2rdL$, and for all proposed methods is $2(rd + r^2)L$, where $L$ is the number of weight matrices that we apply LoRA/LoDA(+)/S-LoDA(+)/R-LoDA(+) to. Note that they are almost the same when $r \ll d$. For example, in GPT2-medium, $d = d_{\text{model}} = 1024$ and $r = 4$ are used for all methods by default, $r^2$ is negligible compared to $rd$. As in LoRA, we only apply the proposed methods to $W_{\text{q}}$ and $W_{\text{v}}$ (of shape $d_{\text{model}} \times d_{\text{model}}$) in the self-attention module.

## 3.2 Computational Efficiency During Inference

In Figure 1, let the input embeddings be $X \in \mathbb{R}^{n\times d}$, where $n$ is sequence length. For the LoDA(+) part, recall that $A \in \mathbb{R}^{d\times r}$, $B \in \mathbb{R}^{r\times d}$, and the two bottleneck matrices in Figure 1b-Figure 1e are $r$ by $r$ square matrices, and there are some non-linear activations and/or layer-normalization layers. The computational complexity of a LoDA(+) adapter during the inference is $O(rdn + dn + r^2n + rn)$, where the dominant part is $O(rdn)$, which is much lower than computing $XW_{\text{q}}$ (or $XW_{\text{v}}$), which costs $O(d^2n)$, since $r \ll d$ (recall that in GPT2-medium, $d = d_{\text{model}} = 1024$ and $r = 4$ are used for all methods by default).

For R-LoDA and S-LoDA, as mentioned earlier, if applying them on $W_{\text{q}}$ and $W_{\text{v}}$, one could low-rank approximate (or sparsify) $W_{\text{q}}$ and $W_{\text{v}}$ separately. To make the model inference more efficient, we directly approximate the whole $W_{\text{proj}} = [W_{\text{q}}, W_{\text{k}}, W_{\text{v}}]$ instead. More specifically, for $W_{\text{proj}} \in \mathbb{R}^{d\times 3d}$, we approximate it using the product of matrix $W_A \in \mathbb{R}^{d\times R}$ and matrix $W_B \in \mathbb{R}^{R\times 3d}$, i.e., $W_AW_B$. The computational cost for $XW_{\text{proj}}$ is $3d^2n$ MACs (Multiply-Accumulate Operations), while the computational cost for the low-rank version $(XW_A)W_B$ is $ndR + nR3d = 4Rdn$ MACs, which is lower than the former as long as $R < 3d/4$ (e.g., in GPT2-medium, $d = 1024$, so we only need $R < 768$). One can calculate that the R-LoDA(+) fine-tuned model is computationally more efficient than the pre-trained model during inference, even setting $R$ as high as 700 in our experimental settings.

Similarly, for S-LoDA(+), the computational cost for the sparsified version $XW_{\text{Sparse}}$ is $s \times 3d^2n$ MACs, where $s$ is the fraction of nonzero entries in $W_{\text{proj}}$. While the added computational cost of the LoDA+ adapters on $W_{\text{q}}$ and $W_{\text{v}}$ is $2(2rd + 2r^2 + 4r)n$ MACs, which is relatively negligible since $r \ll d$, and the total computational saving

| Method | Approx $W_{\text{proj}}$ | # Trainable Parameters | E2E NLG Challenge BLEU | NIST | MET | ROUGE-L | CIDEr |
|---|---|---|---|---|---|---|---|
| FT* | No | 354.92M | 68.2 | 8.62 | 46.2 | 71.0 | 2.47 |
| Adapter$^{\text{L}}$* | No | 0.37M | 66.3 | 8.41 | 45.0 | 69.8 | 2.40 |
| Adapter$^{\text{L}}$* | No | 11.09M | 68.9 | 8.71 | 46.1 | 71.3 | 2.47 |
| Adapter$^{\text{H}}$* | No | 11.09M | $67.3_{\pm.6}$ | $8.50_{\pm.07}$ | $46.0_{\pm.2}$ | $70.7_{\pm.2}$ | $2.44_{\pm.01}$ |
| PreLayer* | No | 0.35M | 69.7 | 8.81 | 46.1 | 71.4 | 2.49 |
| FT$^{\text{Top2}}$* | No | 25.19M | 68.1 | 8.59 | 46.0 | 70.8 | 2.41 |
| FT$^{W_{\text{q}},W_{\text{v}}}$ | No | 48.00M | $69.4_{\pm.1}$ | $8.74_{\pm.02}$ | $46.0_{\pm.0}$ | $71.0_{\pm.1}$ | $2.48_{\pm.01}$ |
| One-shot | No | - | 14.6 | 2.86 | 27.5 | 40.3 | 0.82 |
| LoRA | No | 0.38M | $69.0_{\pm.7}$ | $8.69_{\pm.07}$ | $46.5_{\pm.2}$ | $71.3_{\pm.4}$ | $2.51_{\pm.00}$ |
| **LoDA** | No | 0.38M | $\mathbf{70.2}_{\pm.3}$ | $\mathbf{8.83}_{\pm.03}$ | $\mathbf{46.6}_{\pm.1}$ | $\mathbf{71.6}_{\pm.1}$ | $\mathbf{2.53}_{\pm.01}$ |
| **S-LoDA** | **Keep 40%** | 0.38M | $\mathbf{70.2}_{\pm.3}$ | $\mathbf{8.83}_{\pm.03}$ | $\mathbf{46.6}_{\pm.1}$ | $\mathbf{71.6}_{\pm.1}$ | $\mathbf{2.53}_{\pm.01}$ |
| **R-LoDA** | **Rank300** | 0.38M | $\mathbf{69.7}_{\pm.2}$ | $8.79_{\pm.03}$ | $\mathbf{46.7}_{\pm.0}$ | $\mathbf{71.5}_{\pm.3}$ | $\mathbf{2.52}_{\pm.00}$ |
| **LoDA+** | No | 0.38M | $\mathbf{69.9}_{\pm.3}$ | $\mathbf{8.81}_{\pm.04}$ | $\mathbf{46.5}_{\pm.0}$ | $\mathbf{71.4}_{\pm.0}$ | $\mathbf{2.52}_{\pm.00}$ |
| **S-LoDA+** | **Keep 40%** | 0.38M | $69.6_{\pm.5}$ | $8.77_{\pm.06}$ | $\mathbf{46.7}_{\pm.1}$ | $\mathbf{71.6}_{\pm.2}$ | $2.50_{\pm.01}$ |
| **R-LoDA+** | **Rank300** | 0.38M | $\mathbf{70.1}_{\pm.4}$ | $\mathbf{8.81}_{\pm.05}$ | $46.4_{\pm.1}$ | $\mathbf{71.6}_{\pm.3}$ | $\mathbf{2.52}_{\pm.01}$ |

Table 1: GPT2-medium with different adaptation methods on E2E NLG Challenge. For all metrics, higher is better. * indicates numbers published in prior works, as compiled by [5].

is then approximately $(1-s)\times 3d^2n$ MACs. Reducing the computational complexity with R-LoDA(+) and S-LoDA(+) can directly decrease total power consumption in inference.

## 4 Empirical Studies

We focus on Natural Language Generation (NLG) tasks, and we follow the setup of [5, 8] on GPT2-medium [23] for a direct comparison. We compare the downstream task performance of our proposed methods with LoRA, adapter tuning methods by [10] (Adapter$^{\text{H}}$) and [24] (Adapter$^{\text{L}}$), prefix-layer tuning (PreLayer), full fine-tuning (FT), and fine-tuning the top-2 layers (FT$^{\text{Top2}}$), similar to [5]. We also compare direct fine-tuning of the projection matrices $W_{\text{q}}$ and $W_{\text{v}}$ (denoted as FT$^{W_{\text{q}},W_{\text{v}}}$). As a reference, we also tested the one-shot in-context learning performance of the pre-trained GPT2-medium model, by providing the task description and one example in the prompt.

For LoDA(+), we simply set the hyper-parameters (e.g., bottleneck dimension $r = 4$, learning rate, etc.) to the same as that used by LoRA (indicated in Table 11

of [5][3]) without tuning, which may favor LoRA. For R-LoDA(+) and S-LoDA(+), since the pre-trained $W_{\text{proj}}$ is approximated, training a few more epochs may be needed to recover some details that are potentially lost during approximation. Thus, we train R-LoDA(+) and S-LoDA(+) up to 10 epochs and choose the best result from epoch 5 and epoch 10. The experiments were run on an NVIDIA A40 GPU with 48 GB memory, and the implementations will be available on GitHub.

We first evaluated on the E2E NLG Challenge [25] dataset, which is a dataset for training end-to-end NLG systems and is commonly used for data-to-text evaluation. The dataset consists of approximately 42K training, 4.6K validation, and 4.6K testing examples from the restaurant domain. It is released under the Creative Commons BY-NC-SA 4.0 license. Table 1 compares performance of different methods on this dataset. One-shot in-context learning[4] performs much worse than other methods which fine-tune the model. LoDA, LoDA+, and S-LoDA outperform the baselines (including Fine-Tuning methods) on all 5 evaluation metrics. Other variants R-LoDA(+) and S-LoDA+ perform better than or at least on-par with LoRA and other baselines.

We also perform experiments on the DART dataset [26] following the setup of [5, 8]. This open-domain data-to-text dataset has a total of 82K examples. DART presents a significantly larger and more complex data-to-text task compared to the E2E NLG Challenge dataset [25]. This dataset is released under the MIT license. We evaluate with the BLEU [27], METEOR [28], and TER [29] metrics, similar to [5], but with slightly higher precision. The evaluation code is based on the source code of LoRA [5], which is available on GitHub.[5] The results are shown in Table 2, which also include the number of MACs per token for computing Query/Key/Value during inference in the fine-tuned model. Note that LoRA fine-tuned model has the same computational cost as the pre-trained model if the adapter is merged into the pre-trained weight, otherwise its computational cost is almost the same as LoDA. Also note that the number of trainable parameters in $\text{FT}^{W_{\text{q}},W_{\text{v}}}$ accounts for nearly $1/7$ of the total model parameters, and is 126 times more than that of LoRA and proposed methods. LoDA and especially LoDA+ again outperform LoRA and $\text{FT}^{W_{\text{q}},W_{\text{v}}}$, and their computational cost during inference is only slightly higher than the pre-trained model.

We notice that higher rank and less sparseness to approximate $W_{\text{proj}}$, respectively, in R-LoDA(+) and S-LoDA(+), are needed for DART, as it is a more complex task than the E2E NLG Challenge. Nevertheless, even with pruning 40% of the entries in $W_{\text{proj}}$, S-LoDA(+) can outperform LoRA and $\text{FT}^{W_{\text{q}},W_{\text{v}}}$. For R-LoDA(+), approximating $W_{\text{proj}}$ with the 300 top singular value components seems insufficient on DART,

[3] We do not know the random seeds used in [5]. So we run LoDA(+) and LoRA with the same random seeds for fair comparisons. On DART, we cannot reproduce the results of LoRA using the default of 5 epochs, and we run 10 epochs instead to obtain results similar to that reported in [5].

[4] The task description and one example that we provided in the prompt are as follows: "Generate a restaurant description from the table. Here is an example: name : The Eagle | Type : coffee shop | food : Japanese | price : less than £20 | customer rating : low | area : riverside | family friendly : yes | near : Burger King. The generated description is like: The Eagle is a low rated coffee shop near Burger King and the riverside that is family friendly and is less than £20 for Japanese food. Now generate description for this table: ".

[5] `https://github.com/microsoft/LoRA`

while it is sufficient on E2E NLG Challenge in Table 1 (as R-LoDA and R-LoDA+ outperform baselines on E2E NLG Challenge with Rank = 300). This is likely because the E2E NLG Challenge is a relatively easier downstream task, so a rough low-rank approximation of the pre-trained weights combined with LoDA(+) fine-tuning is sufficient. R-LoDA+ with Rank = 500 shows reasonable performance. We observe a trade-off between efficiency and accuracy in R-LoDA(+) and S-LoDA(+). This may shed light on how to choose the Rank and Sparsity in R-LoDA(+) and S-LoDA(+), which depends on the downstream task as well as its relation to the pre-trained tasks. Such (downstream) task-dependent auto-configuration of Rank and Sparsity is a topic for our future work.

Most notably, the number of MACs per token for computing Query/Key/Value during inference, in both the S-LoDA(+) and R-LoDA(+) fine-tuned models, is significantly lower than the pre-trained model. For example, S-LoDA+ that prunes 40% of the entries in $W_{\text{proj}}$ can significantly reduce the number of MACs, while having slightly better performance than LoRA and $\text{FT}^{W_\text{q},W_\text{v}}$.

### 4.1 Why LoDA(+) can Outperform $\text{FT}^{W_\text{q},W_\text{v}}$?

From Table 1 and Table 2, one can see that LoDA and LoDA+ consistently outperform $\text{FT}^{W_\text{q},W_\text{v}}$ on all evaluation metrics. Recall that LoDA(+) are applied to projection matrices $W_\text{q}$ and $W_\text{v}$, while $\text{FT}^{W_\text{q},W_\text{v}}$ directly fine tunes the whole matrices $W_\text{q}$ and $W_\text{v}$. Naturally, one may question why LoDA(+) does better.

It is important to note that directly fine tuning the weight matrix $W$ of a dense layer still retains a linear mapping. Let the input to that dense layer be denoted by $x$, and the output of the pre-trained dense layer be denoted by $h_0 = xW$. After directly fine-tuning that dense layer, we have $W' = W + \Delta W$, and the new output $h'_{\text{FT}^W} = xW' = xW + x\Delta W = h_0 + \Delta h_{\text{FT}^W}$, where $\Delta h_{\text{FT}^W} = x\Delta W$. So the mapping from input $x$ to the update $\Delta h_{\text{FT}^W}$ is still a linear mapping, though this mapping is generally not low-rank.[6]

In contrast, the mappings of LoDA and LoDA+ in Eq. 1 and Eq. 2 are non-linear, and cannot be expressed in the form of $\Delta h = x\Delta W$. This is in line with the observation in Eq. 5 of [19], when the authors discuss the limited learning capacity of LoRA. From that perspective, our proposed LoDA(+) can be viewed as expanding the learning capacity of LoRA, which further explains why LoDA(+) performs better.

### 4.2 Effect of the Bottleneck Dimension

We further study the effect of the bottleneck dimension $r$ of LoDA(+) adapters in GPT2-medium using the E2E NLG Challenge dataset, and also include LoRA (under

[6] The definition of the rank of a linear mapping can be found at `https://en.wikipedia.org/wiki/Linear_map`

| Method | Approx $W_{\text{proj}}$ | MACs/token of compute Q/K/V | # Trainable Parameters | DART BLEU↑ | DART MET↑ | DART TER↓ |
|---|---|---|---|---|---|---|
| $\text{FT}^{W_q,W_v}$ | No | 75.5M | 48.00M | $47.1_{\pm.1}$ | $\mathbf{36.0}_{\pm.0}$ | $0.480_{\pm.000}$ |
| LoRA | No | 75.5/75.9M | 0.38M | $47.2_{\pm.1}$ | $\mathbf{36.0}_{\pm.0}$ | $0.480_{\pm.000}$ |
| **LoDA** | No | 75.9M | 0.38M | $\mathbf{47.3}_{\pm.1}$ | $\mathbf{36.0}_{\pm.0}$ | $0.480_{\pm.000}$ |
| **S-LoDA** | **Keep 60%** | 45.7M | 0.38M | $\mathbf{47.3}_{\pm.2}$ | $\mathbf{36.0}_{\pm.0}$ | $\mathbf{0.477}_{\pm.006}$ |
| **S-LoDA** | **Keep 50%** | 38.1M | 0.38M | $47.1_{\pm.2}$ | $\mathbf{36.0}_{\pm.0}$ | $0.480_{\pm.000}$ |
| **R-LoDA** | **Rank500** | 49.5M | 0.38M | $46.8_{\pm.7}$ | $35.9_{\pm.1}$ | $0.483_{\pm.006}$ |
| **R-LoDA** | **Rank400** | 39.7M | 0.38M | $46.6_{\pm.2}$ | $35.9_{\pm.1}$ | $0.483_{\pm.006}$ |
| **R-LoDA** | **Rank300** | 29.9M | 0.38M | $46.5_{\pm.2}$ | $35.5_{\pm.4}$ | $0.487_{\pm.006}$ |
| **LoDA+** | No | 76.1M | 0.38M | $\mathbf{47.3}_{\pm.2}$ | $\mathbf{36.0}_{\pm.0}$ | $\mathbf{0.477}_{\pm.006}$ |
| **S-LoDA+** | **Keep 60%** | 45.9M | 0.38M | $\mathbf{47.3}_{\pm.1}$ | $\mathbf{36.0}_{\pm.0}$ | $\mathbf{0.473}_{\pm.006}$ |
| **S-LoDA+** | **Keep 50%** | 38.3M | 0.38M | $47.1_{\pm.2}$ | $\mathbf{36.0}_{\pm.0}$ | $0.480_{\pm.000}$ |
| **R-LoDA+** | **Rank500** | 49.7M | 0.38M | $47.1_{\pm.5}$ | $35.9_{\pm.1}$ | $0.480_{\pm.000}$ |
| **R-LoDA+** | **Rank400** | 39.9M | 0.38M | $46.7_{\pm.2}$ | $35.9_{\pm.1}$ | $0.483_{\pm.006}$ |
| **R-LoDA+** | **Rank300** | 30.1M | 0.38M | $46.3_{\pm.6}$ | $35.6_{\pm.5}$ | $0.483_{\pm.006}$ |

Table 2: GPT2-medium with different adaptation methods on DART. For TER metric, lower is better. The number of MACs per token of computing Query/Key/Value during inference in LoRA fine-tuned model is 75.5 million if the LoRA adapter is merged into the pre-trained weight $W$, otherwise it is 75.9 million.

the same random seeds) as a baseline. The hyper-parameters (e.g., learning rate) of LoDA(+) are set to be the same as that used by LoRA (indicated in Table 11 of [5]) without tuning, which may favor LoRA, but the main purpose here is to study the effect of the bottleneck dimension in LoDA(+). Table 3 shows the performances of each adapter under different bottleneck dimension $r$. LoDA+ and LoRA achieve their best performance roughly around $r = 128$, while further increasing $r$ does not show apparent improvement. Interestingly, LoDA achieves favorable performance at bottleneck dimensions of both $r = 4$ and $r = 256$.

## 5 Conclusion and Future Work

We have generalized LoRA to the framework of LoDA(+), where LoRA is a special case, and have demonstrated their very promising performance. We also extended LoDA(+) to R-LoDA(+) and S-LoDA(+), by applying low-rank and sparse approximation, which achieves similar performance, while drastically improving computational efficiency. One future direction is to approximate $W$ with other structured matrices, e.g., block-sparse matrix, Monarch matrix [30], or with quantization of the pre-trained model, such as in QLoRA [6].

| Method | Bottleneck Dimension $r$ | E2E NLG Challenge BLEU | NIST | MET | ROUGE-L | CIDEr |
|---|---|---|---|---|---|---|
| LoRA | 2 | $69.4_{\pm.5}$ | $8.74_{\pm.07}$ | $46.5_{\pm.1}$ | $71.4_{\pm.3}$ | $2.49_{\pm.03}$ |
| LoRA | 4 | $69.0_{\pm.7}$ | $8.69_{\pm.07}$ | $46.5_{\pm.2}$ | $71.3_{\pm.4}$ | $2.51_{\pm.00}$ |
| LoRA | 8 | $69.4_{\pm.6}$ | $8.74_{\pm.06}$ | $\mathbf{46.6_{\pm.1}}$ | $71.7_{\pm.3}$ | $\mathbf{2.52_{\pm.01}}$ |
| LoRA | 16 | $68.6_{\pm.5}$ | $8.65_{\pm.06}$ | $46.4_{\pm.2}$ | $71.4_{\pm.3}$ | $2.50_{\pm.01}$ |
| LoRA | 32 | $69.4_{\pm.8}$ | $8.74_{\pm.10}$ | $\mathbf{46.6_{\pm.2}}$ | $71.6_{\pm.1}$ | $2.51_{\pm.01}$ |
| LoRA | 64 | $69.7_{\pm.1}$ | $\mathbf{8.77_{\pm.02}}$ | $46.5_{\pm.1}$ | $\mathbf{71.8_{\pm.1}}$ | $2.51_{\pm.01}$ |
| LoRA | 128 | $\mathbf{69.8_{\pm.4}}$ | $\mathbf{8.77_{\pm.03}}$ | $\mathbf{46.6_{\pm.1}}$ | $\mathbf{71.8_{\pm.2}}$ | $2.51_{\pm.01}$ |
| LoRA | 256 | $69.0_{\pm.4}$ | $8.68_{\pm.04}$ | $\mathbf{46.6_{\pm.2}}$ | $71.7_{\pm.2}$ | $2.50_{\pm.01}$ |
| LoRA | 512 | $69.4_{\pm.5}$ | $8.72_{\pm.05}$ | $\mathbf{46.6_{\pm.1}}$ | $71.6_{\pm.1}$ | $2.51_{\pm.01}$ |
| LoRA | 1024 | $69.4_{\pm.4}$ | $8.72_{\pm.04}$ | $\mathbf{46.6_{\pm.1}}$ | $71.6_{\pm.1}$ | $2.51_{\pm.01}$ |
| **LoDA** | 2 | $67.7_{\pm1.0}$ | $8.62_{\pm.14}$ | $44.9_{\pm.5}$ | $69.5_{\pm.7}$ | $2.32_{\pm.04}$ |
| **LoDA** | 4 | $\mathbf{70.2_{\pm.3}}$ | $\mathbf{8.83_{\pm.03}}$ | $46.6_{\pm.1}$ | $71.6_{\pm.1}$ | $\mathbf{2.53_{\pm.01}}$ |
| **LoDA** | 8 | $69.4_{\pm.2}$ | $8.74_{\pm.04}$ | $46.5_{\pm.2}$ | $71.1_{\pm.2}$ | $2.52_{\pm.01}$ |
| **LoDA** | 16 | $69.6_{\pm.4}$ | $8.77_{\pm.05}$ | $46.6_{\pm.1}$ | $71.4_{\pm.2}$ | $2.51_{\pm.01}$ |
| **LoDA** | 32 | $68.3_{\pm1.0}$ | $8.63_{\pm.12}$ | $46.3_{\pm.2}$ | $71.0_{\pm.3}$ | $2.49_{\pm.03}$ |
| **LoDA** | 64 | $68.2_{\pm.7}$ | $8.62_{\pm.09}$ | $46.2_{\pm.2}$ | $70.8_{\pm.4}$ | $2.49_{\pm.02}$ |
| **LoDA** | 128 | $69.9_{\pm.0}$ | $8.81_{\pm.02}$ | $46.6_{\pm.1}$ | $71.6_{\pm.2}$ | $\mathbf{2.53_{\pm.01}}$ |
| **LoDA** | 256 | $\mathbf{70.2_{\pm.8}}$ | $8.82_{\pm.09}$ | $\mathbf{46.8_{\pm.1}}$ | $\mathbf{71.8_{\pm.3}}$ | $\mathbf{2.53_{\pm.02}}$ |
| **LoDA** | 512 | $68.9_{\pm.8}$ | $8.69_{\pm.08}$ | $46.4_{\pm.4}$ | $71.5_{\pm.5}$ | $2.50_{\pm.02}$ |
| **LoDA** | 1024 | $68.9_{\pm.3}$ | $8.70_{\pm.04}$ | $46.5_{\pm.2}$ | $71.5_{\pm.2}$ | $2.51_{\pm.01}$ |
| **LoDA+** | 2 | $66.9_{\pm.9}$ | $8.52_{\pm.16}$ | $44.7_{\pm.1}$ | $69.3_{\pm.1}$ | $2.35_{\pm.06}$ |
| **LoDA+** | 4 | $69.9_{\pm.3}$ | $8.81_{\pm.04}$ | $46.5_{\pm.0}$ | $71.4_{\pm.0}$ | $2.52_{\pm.00}$ |
| **LoDA+** | 8 | $70.0_{\pm.6}$ | $8.82_{\pm.06}$ | $46.6_{\pm.1}$ | $71.4_{\pm.3}$ | $\mathbf{2.54_{\pm.02}}$ |
| **LoDA+** | 16 | $69.6_{\pm.3}$ | $8.77_{\pm.05}$ | $46.6_{\pm.1}$ | $71.4_{\pm.3}$ | $2.52_{\pm.01}$ |
| **LoDA+** | 32 | $69.9_{\pm.3}$ | $8.79_{\pm.04}$ | $46.7_{\pm.1}$ | $71.7_{\pm.0}$ | $2.53_{\pm.01}$ |
| **LoDA+** | 64 | $69.9_{\pm.6}$ | $8.81_{\pm.06}$ | $46.7_{\pm.1}$ | $71.6_{\pm.5}$ | $2.52_{\pm.02}$ |
| **LoDA+** | 128 | $\mathbf{70.2_{\pm.6}}$ | $\mathbf{8.83_{\pm.06}}$ | $\mathbf{46.8_{\pm.1}}$ | $71.9_{\pm.2}$ | $2.53_{\pm.01}$ |
| **LoDA+** | 256 | $69.9_{\pm.4}$ | $8.79_{\pm.03}$ | $46.6_{\pm.1}$ | $71.7_{\pm.4}$ | $2.52_{\pm.01}$ |
| **LoDA+** | 512 | $70.1_{\pm.5}$ | $8.81_{\pm.06}$ | $\mathbf{46.8_{\pm.1}}$ | $\mathbf{72.0_{\pm.2}}$ | $2.53_{\pm.02}$ |
| **LoDA+** | 1024 | $69.6_{\pm.8}$ | $8.77_{\pm.11}$ | $46.7_{\pm.1}$ | $71.7_{\pm.2}$ | $2.52_{\pm.03}$ |

Table 3: GPT2-medium with different adaptation methods and corresponding bottleneck dimensions $r$, on E2E NLG Challenge. For all metrics, higher is better. For each adapter, bold fonts indicate its best metric score among tested bottleneck dimensions.

## References


[1] OpenAI. GPT-4 technical report, 2023.

[2] Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[3] Anthropic. The claude 3 model family: Opus, sonnet, haiku. *Claude 3 technical report*, 2024. URL https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf.

[4] Hugo Touvron, Louis Martin, and Kevin Stone et al. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

[5] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

[6] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36, 2023.

[7] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning, 2021.

[8] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *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)*, pages 4582–4597, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353.

[9] Brian Lester, Rami Al-Rfou, and Noah Constant. The Power of Scale for Parameter-Efficient Prompt Tuning. *arXiv:2104.08691 [cs]*, April 2021. URL http://arxiv.org/abs/2104.08691. arXiv: 2104.08691.

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

[11] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 506–516, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[12] Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. Adapterdrop: On the efficiency of adapters in transformers, 2020.

[13] Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, and Sayak Paul. PEFT: state-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft, 2022.

[14] Jonas Pfeiffer, Sebastian Ruder, Ivan Vulić, and Edoardo Maria Ponti. Modular deep learning. *arXiv preprint arXiv:2302.11529*, 2023.

[15] Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647*, 2023.

[16] Mohammed Sabry and Anya Belz. Peft-ref: A modular reference architecture and typology for parameter-efficient finetuning techniques. *arXiv preprint arXiv:2304.12410*, 2023.

[17] Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235, 2023.

[18] Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *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)*, pages 7319–7328, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.568. URL https://aclanthology.org/2021.acl-long.568.

[19] Baohao Liao, Yan Meng, and Christof Monz. Parameter-efficient fine-tuning without introducing new latency. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4242–4260, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.233. URL https://aclanthology.org/2023.acl-long.233.

[20] Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*, 2022.

[21] Yaoming Zhu, Jiangtao Feng, Chengqi Zhao, Mingxuan Wang, and Lei Li. Serial or parallel? plug-able adapter for multilingual machine translation. *arXiv preprint arXiv:2104.08154*, 6(3), 2021.

[22] Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Lee. LLM-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5254–5276, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.319. URL https://aclanthology.org/2023.emnlp-main.319.

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

[24] Zhaojiang Lin, Andrea Madotto, and Pascale Fung. Exploring versatile generative language model via parameter-efficient transfer learning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 441–459, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.41. URL https://aclanthology.org/2020.findings-emnlp.41.

[25] Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. The E2E dataset: New challenges for end-to-end generation. In Kristiina Jokinen, Manfred Stede, David DeVault, and Annie Louis, editors, *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 201–206, Saarbrücken, Germany, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-5525. URL https://aclanthology.org/W17-5525.

[26] Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, Yangxiaokang Liu, Nadia Irwanto, Jessica Pan, Faiaz Rahman, Ahmad Zaidi, Mutethia Mutuma, Yasin Tarabar, Ankit Gupta, Tao Yu, Yi Chern Tan, Xi Victoria Lin, Caiming Xiong, Richard Socher, and Nazneen Fatema Rajani. DART: Open-domain structured data record to text generation. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 432–447, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.37. URL https://aclanthology.org/2021.naacl-main.37.

[27] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 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, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL https://aclanthology.org/P02-1040.

[28] Alon Lavie and Abhaya Agarwal. 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, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL https://aclanthology.org/W07-0734.

[29] Matthew Snover, Bonnie Dorr, Rich Schwartz, Linnea Micciulla, and John Makhoul. 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, Cambridge, Massachusetts, USA, August 8-12 2006. URL https://aclanthology.org/2006.amta-papers.25.

[30] Tri Dao, Beidi Chen, Nimit S Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, pages 4690–4721. PMLR, 2022.