# LoDA: Low-Dimensional Adaptation of Large Language Models

**Jing Liu** **Toshiaki Koike-Akino** **Pu Wang**
**Matthew Brand** **Ye Wang** **Kieran Parsons**
Mitsubishi Electric Research Laboratories
Cambridge, MA 02139, USA
{jiliu, koike, pwang, brand, yewang, parsons}@merl.com

## Abstract

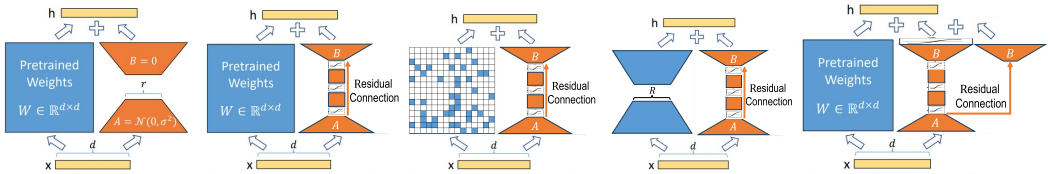
Parameter-Efficient Fine-Tuning (PEFT) has recently garnered significant attention, due to the enormous size of Large Language Models (LLM). Among various PEFT methods, **Lo**w-**R**ank **A**daptation (LoRA) demonstrates comparable performance to full fine-tuning, despite having significantly fewer trainable parameters. In this work, we first generalize LoRA from a low-rank linear adaptation/mapping to low-dimensional, non-linear adaptation/mapping, called **Lo**w-**D**imensional **A**daptation (LoDA). We further propose LoDA+, which further improves the expressiveness of the non-linear adaptation and still uses almost 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(+), replacing the pre-trained weight matrix by its low-rank or sparse approximation, which is frozen during fine-tuning. Empirical evaluations on Natural Language Generation tasks show that LoDA(+) and some variants outperform LoRA as well as other baselines. We will release a package that facilitates the integration of LoDA(+) and their variants with PyTorch models.

## 1 Introduction

Large language models (LLMs), e.g., ChatGPT (OpenAI, 2023), PALM2 (Anil et al., 2023), and LLaMA2 (Touvron et al., 2023) 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 been growing 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. For domain-specific tasks, recent studies show that the performance of the Pre-trained Languge Model (PLM) can be significantly improved by further fine-tuning on domain-specific data. Therefore, fine-tuning PLMs for domain-specific tasks has become the *de 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 GPU memory (Dettmers et al., 2023). Parameter-Efficient Fine-Tuning (PEFT) only fine-tunes 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. 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 (Pfeiffer et al., 2021), which has been observed during the full fine-tuning of LLMs. PEFT

[1]An example from Hu et al. (2022) is fine-tuning GPT-3 175B uses LoRA with rank 4. The model is 350GB, and LoRA adapter size is only 35MB for each downstream task. Storing 100 adapted models only requires 350GB + 35MB $\times$ 100 $\approx$ 354GB as opposed to 350GB $\times$ 100 $\approx$ 35TB of the full fine-tuning for 100 tasks.

NeurIPS 2023 workshop on Efficient Natural Language and Speech Processing (ENLSP).




(a) LoRA (b) LoDA (c) S-LoDA (d) R-LoDA (e) LoDA+

Figure 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).

approaches have also been shown to be better than full fine-tuning in the low-data regimes (Li & Liang, 2021; Hu et al., 2022). Therefore, many PEFT methods have been proposed. For example, prefix tuning (Li & Liang, 2021) and prompt tuning (Lester et al., 2021) prepend some tunable prefix tokens to the input or hidden layers and only train these soft prompts during fine-tuning. Several adapter tuning methods (Houlsby et al., 2019; Rebuffi et al., 2017; Pfeiffer et al., 2021; Rücklé et al., 2020) insert (and tune) small neural modules called adapters to some layers of the PLM. More recently, Hu et al. (2022) propose to use low-rank decomposition matrices to approximate the parameter update of the weight matrix of a dense layer, and in particular, they propose to update the Query and Value projection matrices in the Transformer architecture, which shows very promising performance and has become a popular PEFT tool for LLMs in modern libraries, e.g., Hugging Face PEFT library (Mangrulkar et al., 2022). For a comprehensive review and comparison, we refer the interested readers to recent surveys (Pfeiffer et al., 2023; Lialin et al., 2023; Sabry & Belz, 2023; Ding et al., 2023).

LoRA is motivated by the 'intrinsic low-dimensional task adaptation' hypothesis of Aghajanyan et al. (2020). LoRA assumes that the change in weights during model adaptation has a low 'intrinsic rank', leading to the **Lo**w-**R**ank **A**daptation (LoRA) approach (Hu et al., 2022). 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 the low-rank decomposition matrices $A \in \mathbb{R}^{d\times r}$ and $B \in \mathbb{R}^{r\times d}$ to constrain the weight update $\Delta W = AB$ (see 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 that of the original $W$.

Further, let the input to that dense layer be $x$, the output of pre-trained dense layer is $h_0 = xW$. 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.

Though the update of mapping $x \to \Delta h$ likely has an intrinsic low dimension for task adaptation, it may not be well captured by the linear low-rank adaptation $xAB$ in LoRA, but rather a more general low-dimensional adaptation $f(x)$. Therefore, we propose the **Lo**w-**D**imensional **A**daptation (LoDA) approach and its variants, which will be detailed in next section.

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

## 2 Proposed Method

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

Mathematically, with our proposed residual connection architecture of LoDA in Figure 1b, we have $\Delta h_{\text{LoDA}} = f_{\text{LoDA}}(x) = xAB + f_1(xA)B$, 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 just an example of a LoDA structure, e.g., 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.

**Extension to LoDA+** Though LoDA generalizes LoRA from a low-rank linear mapping/adaptation to low-dimensional, non-linear mapping/adaptation, and keeps LoRA as a special case, the image of such 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 following mapping for LoDA+, illustrated in Figure 1e, that gives a positive answer:

$$\Delta h_{\mathrm{LoDA+}} = f_{\mathrm{LoDA+}}(x) = xAB + f_2(f_1(xA)B). \tag{1}$$

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 $f_{\mathrm{LoDA+}}$ becomes the combination of a linear subspace (the first term in Eq. 1) and a non-linear manifold (the second term in Eq. 1). For convenience, we will use LoDA(+) to stand for 'LoDA and LoDA+'.

**LoDA(+) viewed as deep parallel adapters** He et al. (2022) viewed LoRA as a parallel adapter. Similarly, the proposed LoDA can be viewed as a *deep* parallel adapter. Recent work He et al. (2022); Zhu et al. (2021) 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 is attached 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 Hu et al. (2023, Figure 1) and He et al. (2022, 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 $W_{\mathrm{q}}$ and $W_{\mathrm{v}}$, not attached to the whole Attention module nor Feed-Forward Network module of 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. 1 respectively.

**S-LoDA(+) and R-LoDA(+)** As the dimension of the non-linear layers in LoDA(+) is restricted to a very small $r$, the additional computation cost during the inference is very small (see Appendix A.4 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 bottleneck of the computation is still on $W$. *Is it possible to further improve the computation efficiency of a LoDA(+) fine-tuned model, and even better than the pre-trained model?*

We also observe that the combined projection matrix $W_{\mathrm{proj}} = [W_{\mathrm{q}}, W_{\mathrm{k}}, W_{\mathrm{v}}]$, 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 details can be found in Appendix A.1.

The above questions and observations motivate us to further propose R-LoDA(+) and S-LoDA(+), which are LoDA(+) combined with the low-rank or sparsified approximations of $W$, which are frozen during the fine-tuning, and the adapter part is learned/fine-tuned given such approximated $W$. See Figure 1c and Figure 1d for illustrations[2]. They can be computationally more efficient than the pre-trained model during inference, more details can be found in Section A.4. Our empirical investigation shows that even when the pre-trained projection matrix $W_{\mathrm{proj}}$ is low-rank approximated or sparsified, combining with LoDA(+) can still achieve competitive performance.

**Number of fine-tuning parameters** The number of trainable parameters of LoRA and 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 $B$ in all methods are of shape $d$ by $r$. The proposed methods have additional two $r$ by $r$ bottleneck matrices. Their non-linear activation function is LeakyReLU with fixed slope $0.8$, and their layer-normalization is not trainable. So, the total number

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

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 we apply LoRA/LoDA(+)/S-LoDA(+)/R-LoDA(+) to. Note that they are almost the same when $r \ll d$. For example, in GPT-2 medium, $d = d_{\text{model}} = 1024$ and $r = 4$ are used for all methods, $r^2$ 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 Empirical Studies

We focus on Natural Language Generation (NLG) tasks, and we follow the setup of Hu et al. (2022); Li & Liang (2021) on GPT-2 medium (Radford et al.) for a direct comparison. We compare the downstream task performance of proposed methods with LoRA, adapter tuning methods by Houlsby et al. (2019) (Adapter$^{\text{H}}$) and Lin et al. (2020) (Adapter$^{\text{L}}$), prefix-layer tuning (PreLayer), full fine-tuning (FT), fine-tuning the top-2 layers (FT$^{\text{Top2}}$), same as in Hu et al. (2022). We additionally compare with directly fine-tuning the projection matrices $W_{\text{q}}$ and $W_{\text{v}}$ (denoted as FT$^{W_{\text{q}},W_{\text{v}}}$).

The benchmark datasets we evaluated on are E2E NLG Challenge (Novikova et al., 2017) and DART (Nan et al., 2020). We notice that the model performs very poorly if not adapted. For LoDA(+), we simply set their hyper-parameters (e.g., learning rate) the same as that used by LoRA (indicated in Table 11 of Hu et al. (2022)[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 pickup some details that are potentially lost during approximation. Therefore, we train R-LoDA(+) and S-LoDA(+) up to 10 epochs and choose the best result from epoch 5 and epoch 10.

Table 1 compares performance on the E2E NLG Challenge. 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 repeat our experiments on DART (Nan et al., 2020) following the setup of Hu et al. (2022); Li & Liang (2021). The results, where LoDA, LoDA+ and S-LoDA(+) outperform LoRA, are shown in Table 2 of Appendix A.2, due to space constraints, with further discussion also found in Appendix A.3.

Table 1: GPT-2 medium with different adaptation methods on E2E NLG Challenge. For all metrics, higher is better. * indicates numbers published in prior works, as compiled by Hu et al. (2022).

| 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 | $69.4_{\pm.1}$ | $8.74_{\pm.02}$ | $46.0_{\pm.0}$ | $71.0_{\pm.1}$ | $2.48_{\pm.01}$ |
| 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 | $\mathbf{70.1}_{\pm.4}$ | $\mathbf{8.81}_{\pm.05}$ | $46.4_{\pm.1}$ | $\mathbf{71.6}_{\pm.3}$ | $\mathbf{2.52}_{\pm.01}$ |

## 4 Conclusion and Future Work

We generalized LoRA to the framework of LoDA(+), where LoRA is a special case, and have shown their very promising performance. To improve computation efficiency, we extended LoDA(+) to R-LoDA(+) and S-LoDA(+), which show promising performance. One future direction is to approximate $W$ with other structured matrices, e.g., block-sparse matrix, Monarch matrix (Dao et al., 2022), or with quantization of the pre-trained model, such as in QLoRA (Dettmers et al., 2023).

[3]We do not know the random seeds used in Hu et al. (2022). So we run LoDA(+) and LoRA under the same random seeds for fair comparisons. On DART dataset, we can not reproduce the results of LoRA using the default 5 epochs, and we run 10 epochs instead to obtain results similar to that reported in Hu et al. (2022).

## References


Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic Dimensionality Explains the Effectiveness of Language Model Fine-Tuning. *arXiv:2012.13255 [cs]*, December 2020. URL http://arxiv.org/abs/2012.13255.

Rohan Anil, Andrew M. Dai, and Orhan Firat et al. PaLM 2 technical report, 2023. URL https://arxiv.org/abs/2305.10403.

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*, pp. 4690–4721. PMLR, 2022.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. *arXiv preprint arXiv:2305.14314*, 2023.

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.

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.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-Efficient Transfer Learning for NLP. *arXiv:1902.00751 [cs, stat]*, June 2019. URL http://arxiv.org/abs/1902.00751.

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.

Zhiqiang Hu, Yihuai Lan, Lei Wang, Wanyu Xu, Ee-Peng Lim, Roy Ka-Wei Lee, Lidong Bing, and Soujanya Poria. LLM-Adapters: An adapter family for parameter-efficient fine-tuning of large language models. *arXiv preprint arXiv:2304.01933*, 2023.

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*, pp. 228–231, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL https://aclanthology.org/W07-0734.

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.

Xiang Lisa Li and Percy Liang. Prefix-Tuning: Optimizing Continuous Prompts for Generation. *arXiv:2101.00190 [cs]*, January 2021. URL http://arxiv.org/abs/2101.00190.

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.

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)*, pp. 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.

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*, pp. 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.

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.

Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, et al. DART: Open-domain structured data record to text generation. *arXiv preprint arXiv:2007.02871*, 2020.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. The E2E dataset: New challenges for end-to-end generation. *arXiv preprint arXiv:1706.09254*, 2017.

OpenAI. GPT-4 technical report, 2023.

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*, pp. 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073135. URL https://aclanthology.org/P02-1040.

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

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

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. pp. 24.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. *arXiv:1705.08045 [cs, stat]*, November 2017. URL http://arxiv.org/abs/1705.08045. arXiv: 1705.08045.

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

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

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*, pp. 223–231, Cambridge, Massachusetts, USA, August 8-12 2006. URL https://aclanthology.org/2006.amta-papers.25.

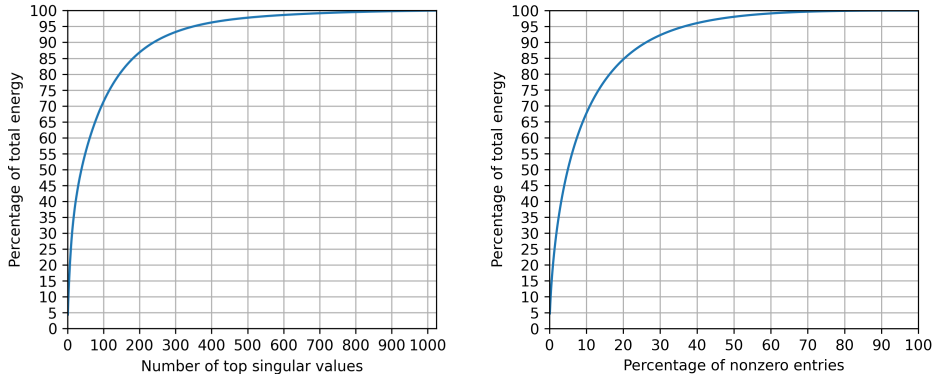
Hugo Touvron, Louis Martin, and Kevin Stone et al. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

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.

# A Appendix

## A.1 Low-rank or sparse approximation

We observe that the combined projection matrices $W_{\text{proj}} = [W_{\text{q}}, W_{\text{k}}, W_{\text{v}}]$ (of size $1024 \times 3072$), within the Attention module of GPT2-medium, can be well-approximated by a relative low-rank matrix with rank $< 500$ or a relatively sparse matrix with more than half of the entries equal to zero. More specifically, Figure 2a shows the percentage of total energy (i.e., $\sum_{i=1}^{R} \sigma_i^2 / \sum_{i=1}^{1024} \sigma_i^2$) w.r.t. the number of top singular values $R$ of $W_{\text{proj}}$ in the first Transformer block of GPT2-medium. We can see that, even using only the top 300 singular value components of $W_{\text{proj}}$ preserves over $93\%$ of its total energy. Similarly, 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 $40\%$ larger magnitude entries of $W_{\text{proj}}$ can preserve over $96\%$ of its total energy.




(a) (b)

Figure 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).

### A.2 Additional experiments and discussions

We also perform experiments on the DART dataset (Nan et al., 2020) following the setup of Hu et al. (2022); Li & Liang (2021). This open-domain data-to-text dataset has a total of $82K$ examples. DART is a significantly larger and more complex data-to-text task compared to the E2E NLG Challenge dataset (Novikova et al., 2017). We evaluate with the BLEU (Papineni et al., 2002), METEOR (Lavie & Agarwal, 2007), and TER (Snover et al., 2006) metrics, similar to Hu et al. (2022), but with slightly higher precision, with the results shown in Table 2. 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 $\text{FT}^{W_{\text{q}},W_{\text{v}}}$. We notice that higher rank and less sparseness, 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, 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 relative 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.

### A.3 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. Mathematically, let the input to that dense layer be $x$, the output of pre-trained dense layer is $h_0 = xW$. 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[4].

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

In contrast, recall that for LoDA and LoDA+, we have

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

$$\Delta h_{\mathrm{LoDA+}} = f_{\mathrm{LoDA+}}(x) = xAB + f_2(f_1(xA)B). \tag{3}$$

These mappings 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 Liao et al. (2023), 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(+) can do better.

Table 2: GPT-2 medium with different adaptation methods on DART. For TER metric, lower is better.

| Method | Approx $W_{\mathrm{proj}}$ | # Trainable Parameters | DART BLEU↑ | DART MET↑ | DART TER↓ |
|---|---|---|---|---|---|
| FT$^{W_{\mathrm{q}},W_{\mathrm{v}}}$ | No | 48.00M | $47.1_{\pm.1}$ | $\mathbf{36.0}_{\pm.0}$ | $0.480_{\pm.000}$ |
| LoRA | No | 0.38M | $47.2_{\pm.1}$ | $\mathbf{36.0}_{\pm.0}$ | $0.480_{\pm.000}$ |
| **LoDA** | No | 0.38M | $\mathbf{47.3}_{\pm.1}$ | $\mathbf{36.0}_{\pm.0}$ | $0.480_{\pm.000}$ |
| **S-LoDA** | **Keep 60%** | 0.38M | $\mathbf{47.3}_{\pm.2}$ | $\mathbf{36.0}_{\pm.0}$ | $\mathbf{0.477}_{\pm.006}$ |
| **S-LoDA** | **Keep 50%** | 0.38M | $47.1_{\pm.2}$ | $\mathbf{36.0}_{\pm.0}$ | $0.480_{\pm.000}$ |
| **R-LoDA** | **Rank500** | 0.38M | $46.8_{\pm.7}$ | $35.9_{\pm.1}$ | $0.483_{\pm.006}$ |
| **R-LoDA** | **Rank400** | 0.38M | $46.6_{\pm.2}$ | $35.9_{\pm.1}$ | $0.483_{\pm.006}$ |
| **R-LoDA** | **Rank300** | 0.38M | $46.5_{\pm.2}$ | $35.5_{\pm.4}$ | $0.487_{\pm.006}$ |
| **LoDA+** | No | 0.38M | $\mathbf{47.3}_{\pm.2}$ | $\mathbf{36.0}_{\pm.0}$ | $\mathbf{0.477}_{\pm.006}$ |
| **S-LoDA+** | **Keep 60%** | 0.38M | $\mathbf{47.3}_{\pm.1}$ | $\mathbf{36.0}_{\pm.0}$ | $\mathbf{0.473}_{\pm.006}$ |
| **S-LoDA+** | **Keep 50%** | 0.38M | $47.1_{\pm.2}$ | $\mathbf{36.0}_{\pm.0}$ | $0.480_{\pm.000}$ |
| **R-LoDA+** | **Rank500** | 0.38M | $47.1_{\pm.5}$ | $35.9_{\pm.1}$ | $0.480_{\pm.000}$ |
| **R-LoDA+** | **Rank400** | 0.38M | $46.7_{\pm.2}$ | $35.9_{\pm.1}$ | $0.483_{\pm.006}$ |
| **R-LoDA+** | **Rank300** | 0.38M | $46.3_{\pm.6}$ | $35.6_{\pm.5}$ | $0.483_{\pm.006}$ |

### A.4 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 activation and/or layer-normalization layers. The computation 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 $XW_{\mathrm{q}}$ (or $XW_{\mathrm{v}}$), which costs $O(d^2n)$, since $r \ll d$ (recall that $r = 4$ and $d = 1024$ in GPT2-medium).

For R-LoDA and S-LoDA, as mentioned earlier, if applying them on $W_{\mathrm{q}}$ and $W_{\mathrm{v}}$, one could low-rank approximate (or sparsify) $W_{\mathrm{q}}$ and $W_{\mathrm{v}}$ separately. To make the model inference more efficient, we directly approximate the whole $W_{\mathrm{proj}} = [W_{\mathrm{q}}, W_{\mathrm{k}}, W_{\mathrm{v}}]$ instead. More specifically, for $W_{\mathrm{proj}} \in \mathbb{R}^{d\times 3d}$, we can 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 computation complexity for $XW_{\mathrm{proj}}$ is $3d^2n$ MACs (Multiply-Accumulate Operations); while the computation complexity 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, the S-LoDA(+) fine-tuned model is computationally more efficient than the pre-trained model, since $r \ll d$.