LaMDA: Large Model Fine-Tuning via Spectrally Decomposed Low-Dimensional Adaptation

Seyedarmin Azizi[1]*, **Souvik Kundu**[2], **Massoud Pedram**[1]

[1]University of Southern California, USA [2]Intel Labs, San Diego, USA

Abstract

Low-rank adaptation (LoRA) has become the default approach to fine-tune large language models (LLMs) due to its significant reduction in trainable parameters. However, trainable parameter demand for LoRA increases with increasing model embedding dimensions, leading to high compute costs. Additionally, its backward updates require storing high-dimensional intermediate activations and optimizer states, demanding high peak GPU memory. In this paper, we introduce *LaMDA*, a novel approach to fine-tuning large language models, which leverages low-dimensional adaptation to achieve significant reductions in trainable parameters and peak GPU memory footprint. LaMDA freezes a first projection matrix (PMA) in the adaptation path while introducing a low-dimensional trainable square matrix, resulting in substantial reductions in trainable parameters and peak GPU memory usage. LaMDA gradually freezes a second projection matrix (PMB) during the early fine-tuning stages, reducing the compute cost associated with weight updates to enhance parameter efficiency further. We also present an enhancement, LaMDA++, incorporating a "lite-weight" adaptive rank allocation for the LoRA path via normalized spectrum analysis of pre-trained model weights. We evaluate LaMDA/LaMDA++ across various tasks, including natural language understanding with the GLUE benchmark, text summarization, natural language generation, and complex reasoning on different LLMs. Results show that LaMDA matches or surpasses the performance of existing alternatives while requiring up to **17.7**× fewer parameter updates and up to **1.32**× lower peak GPU memory usage during fine-tuning. Code will be publicly available at https://github.com/ArminAzizi98/LaMDA.

*Correspondence author (seyedarm@usc.edu)

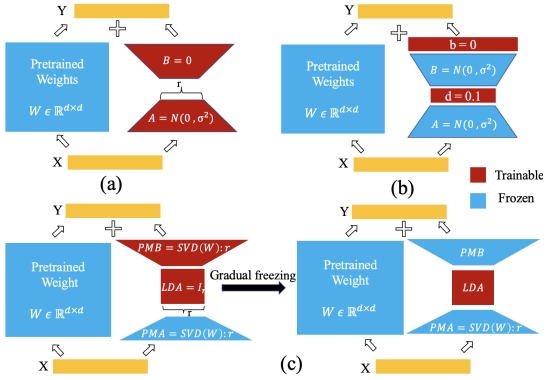

Figure 1: (a) LoRA (Hu et al., 2022). (b) VERA (Kopiczko et al., 2024). (c) **LaMDA**. At the beginning, PMB is trainable and gradually freezes based on the singular values. After t_i iterations, PMB is completely frozen, and only the LDA is fine-tuned.

1 Introduction

Large language models (LLMs) have demonstrated remarkable performance in addressing a variety of natural language processing (NLP) tasks due to their generalization ability upon training on large corpus of data (Brown et al., 2020; Touvron et al., 2023a). To fully harness the capabilities of LLMs, fine-tuning has become the standard approach to serve various downstream tasks. However, full fine-tuning of LLMs can be prohibitively costly, making fine-tuning at the edge hardly possible. For example, even the smaller variants of LLMs with 7B parameters may ask for $\sim$ 60 GB memory to perform full fine-tuning (Pan et al., 2024a). Additionally, such approach is prone to causing overfitting and catastrophic forgetting in the over-parameterized regime (Luo et al., 2023; Doering et al., 2024).

As a solution to these challenges, parameter-efficient fine-tuning (PEFT) techniques were proposed in which either a small portion of model parameters are updated, including the weight adapters (Houlsby et al., 2019; Hu et al., 2023), or task-specific soft prompts are trained (Lester et al., 2021). Among these, low-rank adaptation (LoRA)

Findings of the Association for Computational Linguistics: EMNLP 2024, pages 9635–9646

(Hu et al., 2022) has gained significant popularity. It assumes that the changes in the pre-trained weight reside in a low-rank space and thus adds two trainable low-rank adapters, named the projection matrix $\boldsymbol{A}$ (PMA) and the projection matrix $\boldsymbol{B}$ (PMB) as $\boldsymbol{BA}$ in parallel to the frozen main path of the model weight $\boldsymbol{W}$ (refer to Fig. 1(a)). LoRA fine-tuning can reduce GPU memory demand and trainable parameter-count since it only fine-tunes the $\boldsymbol{BA}$ which is much smaller in parameter count compared to $\boldsymbol{W}$. However, the number of trainable-parameters in LoRA may still be potentially larger than the low intrinsic dimension of a pre-trained LLM (Aghajanyan et al., 2021). Moreover, as evident in Fig 1(a), the input activations $\boldsymbol{X}$ that must be stored for backpropagation reside in a d-dimensional space, where d denotes the embedding dimension of the model. Subsequently, the activation's GPU memory increases linearly with the embedding size, and LoRA does not provide any merit in activation memory saving. Notably, few contemporary works (Liu et al., 2024b; Kopiczko et al., 2024) have presented solutions of $\boldsymbol{BA}$ freezing. However, they still suffer from increased activation storage and often demand high rank to compensate for the accuracy drop due to freezing (Kopiczko et al., 2024).

To address these issues, in this work we present, **La**rge **M**odel Fine-tuning via Spectrally **D**ecomposed Low-Dimensional **A**daptation (LaMDA). LaMDA as demonstrated in Fig. 1(c) employs a trainable **low-dimensional adapter** (LDA), which is a square matrix in the r-dimensional space. We keep the PMA frozen throughout the fine-tuning, while allow PMB to freeze gradually only after few epochs based on relative magnitude of the singular values. We only keep the LDA trainable throughout. This allows the trainable parameters to be independent of d and the activations that are saved for backward pass remain in the r-dimensional space ($r \ll d$). Thus LaMDA can significantly reduce the trainable parameter, activation, and optimizer state memory. We summarize our contributions as follows:

- We introduce LaMDA, a novel framework to fine-tune LLMs that significantly reduces both parameter count and activation memory, resulting in lower computational costs and GPU memory usage.
- We then present LaMDA++, a novel enhancement of LaMDA that uses adaptive rank across different layers to fine-tune the model. Precisely, we use the pre-trained weight tensors to present a "lite-weight" normalized **energy-score**[1] based layer ranking to adaptively assign rank to the LDA of each layer allowing more optimal distribution of trainable parameters. Table 1 compares the different PEFT methods and their benefits and limitations in the context of different memory footprint and adaptive rank allocation policy.
- We evaluate the performance of LaMDA and LaMDA++ fine-tuning on encoder-only (DeBERTa-V3 (He et al., 2021)), encoder-decoder (BART-Large (Lewis et al., 2020)), and decoder-only (LLaMA2-7B (Touvron et al., 2023b)) models across various tasks including the GLUE benchmark for natural language understanding, text summarization, and complex reasoning. Our experiments show that LaMDA fine-tuned models consistently yield similar or improved performance with up to **17.7**× fewer trainable parameters, at reduced activation storage while providing peak GPU memory saving of up to **1.32**×.

Table 1: Comparison of different important metrics associated to different fine-tuning techniques.

Method	Memory			Adaptive rank
	Optimizer	Gradient	Activation	
Full FT	✗	✗	✗	✗
LoRA	✓	✓	✗	✗
AdaLoRA	✓	✓	✗	✓
LoRA-FA	✓	✓	✓	✗
LISA	✓	✓	✗	✗
VERA	✓	✓✓	✗	✗
AFLoRA	✓	✓✓	✗	✗
LaMDA (Ours)	✓	✓✓	✓	✗
LaMDA++ (Ours)	✓	✓✓	✓	✓✓

2 Background

Transformer-based models. Each module of an L layer transformer model (Vaswani et al., 2017) usually consists of two sub-blocks: the multi-head self-attention (MHSA) sub-block and the feed-forward network (FFN). Each MHSA takes input token embedding $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ and applies the following:

$$\boldsymbol{Q}^{(i)} = \boldsymbol{X}\boldsymbol{W}_Q^{(i)}, \boldsymbol{K}^{(i)} = \boldsymbol{X}\boldsymbol{W}_K^{(i)}, \boldsymbol{V}^{(i)} = \boldsymbol{X}\boldsymbol{W}_V^{(i)} \quad (1)$$

$$\boldsymbol{H}^{(i)} = [\text{Softmax}(\boldsymbol{Q}^{(i)}\boldsymbol{K}^{(i)T}/\sqrt{d_h})\boldsymbol{V}^{(i)}] \quad (2)$$

$$\text{MHSA}(\boldsymbol{X}) = \text{Concat}[\boldsymbol{H}^{(1)}, .., \boldsymbol{H}^{(i)}, .., \boldsymbol{H}^{(h)}]\boldsymbol{W}_o \quad (3)$$

[1] Energy-score of a matrix is defined as the summation of the square of its singular values.

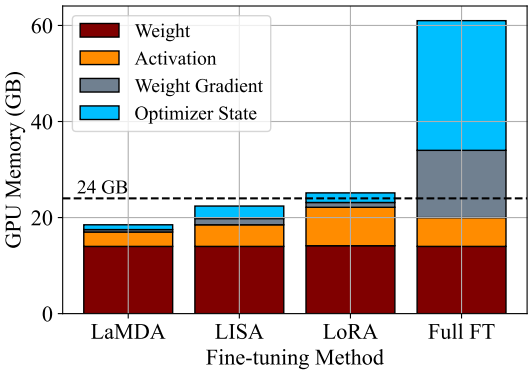

Figure 2: GPU memory usage of LLaMA2-7B on different fine-tuning methods including ours (LaMDA).

where $\boldsymbol{W}_O \in \mathbb{R}^{d\times d}$, $\boldsymbol{W}_{Q,K,V}$ (all $\in \mathbb{R}^{d\times d_h}$) are output projection, query, key, and value matrices with d_h as model embedding dimension per head. The FFN includes two linear transformation layers ($\boldsymbol{W}_{FFN1}$ and $\boldsymbol{W}_{FFN2}$) with a non-linear activation function σ in the middle: $\text{FFN}(\boldsymbol{X}) = \sigma(\boldsymbol{X}\boldsymbol{W}_{FFN1})\boldsymbol{W}_{FFN2}$. With MHSA and FFN as the two sub-blocks, the output of the transformer block is computed as:

$$\boldsymbol{X}' = \text{LayerNorm}(\boldsymbol{X} + \text{MHSA}(\boldsymbol{X})) \quad (4)$$

$$\boldsymbol{Y} = \text{LayerNorm}(\boldsymbol{X}' + \text{FFN}(\boldsymbol{X}')) \quad (5)$$

LayerNorm is the layer normalization module, and Y is the output of the transformer block.

Low-rank adaptation. LoRA adds a trainable adaptation path (through the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrix) parallel to the frozen main path of the respective module (i.e., $\boldsymbol{W}$). This results in a significant reduction in the number of trainable parameters, the optimizer's state memory, and the required gradient memory:

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{W} + \alpha\boldsymbol{X}\boldsymbol{A}\boldsymbol{B}. \quad (6)$$

α serves as a fine-tuning hyper-parameter.

Further numerous variants of LoRA have been introduced to enhance its performance. DoRA (Liu et al., 2024a) proposed the decomposition of weights into magnitude and direction components and used LoRA for directional updates to enhance the learning capacity and training stability of LoRA. Earlier works (Zhang et al., 2023b; Pan et al., 2024a) explored different per-layer rank allocation and layer-importance sampling strategies to better utilize the fine-tuning budget across the model layers. The approach learns the adapters' ranks dynamically by analyzing the singular values of the adapters, allowing for more effective utilization of the fine-tuning budget. More recently, (Zhang et al., 2023a) addresses GPU memory savings by freezing the PMA matrix in the adaptation path, thereby reducing the size of the stored activations during fine-tuning. However, this method still requires fine-tuning $d \times r$ parameters per linear layer and compromises accuracy. VERA (Kopiczko et al., 2024) takes a different approach by randomly initializing and freezing PMA and PMB with a large r dimension, focusing on fine-tuning two feature transformation vectors instead (Fig. 1). While this method reduces parameter count, add significant compute and activation memory overhead. To address VERA's computational inefficiencies, AFLoRA (Liu et al., 2024b) was proposed. However, it still suffers from high activation storage overhead.

LaMDA, on the contrary, offers two key benefits: 1) it significantly reduces trainable parameter, activation, and optimizer storage to enhances memory savings compared to LoRA; 2) it greatly reduces computational cost in the forward pass during fine-tuning. Table 1 compares various state-of-the-art (SOTA) fine-tuning methods regarding their memory requirement and rank adaptation. Notice that **LaMDA(++) is the only method that can simultaneously reduce gradient, optimizer, and activation memory while also yielding adaptive ranks based on a notion of layers' energy-score**.

3 Methodology

This section provides a detailed explanation of LaMDA and LaMDA++ as novel parameter-efficient fine-tuning methods.

3.1 Low-Dimensional Adapter (LDA)

One of the essential components of the LaMDA method is a square r-dimensional matrix $\boldsymbol{S}$, as depicted in Figure 1(c). Integrating this module into the adapter path yields the following formulation:

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{W} + \alpha\boldsymbol{X}\boldsymbol{A}\boldsymbol{S}\boldsymbol{B}, \quad (7)$$

where $\boldsymbol{A} \in \mathbb{R}^{d\times r}$, $\boldsymbol{S} \in \mathbb{R}^{r\times r}$, and $\boldsymbol{B} \in \mathbb{R}^{r\times d}$ represent PMA, LDA, and PMB, respectively. By freezing $\boldsymbol{A}$ and $\boldsymbol{B}$ and keeping $\boldsymbol{S}$ trainable, we significantly reduce the number of trainable parameters, which is reduced to $r^2 \ll 2 \times d \times r$ of LoRA, and is independent of the increasing model d. This reduction in the number of trainable parameters offers a two-fold advantage. Firstly, it effectively constrains the parameter count, thereby reducing the risk of overfitting and enhancing the model's generalization capabilities. This is particularly advantageous considering that $2\times d\times r\times L$ potentially

exceeds the intrinsic dimension of large language models (Aghajanyan et al., 2021). Secondly, fine-tuning fewer parameters requires less computation in the backward pass as fewer gradient-based updates and optimizer states computations happen accordingly. This alleviates the overall computational and optimizer storage overhead of fine-tuning. Additionally, employing the low-dimensional adapter while freezing $\boldsymbol{A}$ reduces activation memory usage during fine-tuning. Assuming a fine-tuning batch size of b and an input sequence length of n, in LoRA, the input X in Equation 6 is represented as a $\boldsymbol{B} \times n \times d$ tensor. This tensor must be stored in GPU memory, as it is essential for computing the gradient for the trainable matrix $\boldsymbol{A}$. Consequently, the required GPU memory for storing the activations is a function of the embedding dimension d. However, when utilizing the low-dimensional adapter $\boldsymbol{S}$ in Equation 7, and since $\boldsymbol{A}$ is not being trained, the required activation in the backward pass is of dimension $\boldsymbol{B} \times n \times r$. This leads to a significant reduction in the number of stored elements and GPU memory usage.

Figure 2 reports the peak GPU memory usage of different fine-tuning methods for the LLaMA2-7B model with a batch size one. As the figure shows, compared to LoRA, most GPU memory saving is achieved by the required activation memory reduction. Furthermore, LaMDA surpasses the current SOTA fine-tuning approach of LISA (Pan et al., 2024b). Having established the benefits of our low-dimensional adapter, we delve into the details of the LaMDA fine-tuning process in the next section.

3.2 LaMDA

Building upon the idea of the low-dimensional adapter, we now disclose the LaMDA in detail. Considering Figure 1(c), a natural implementation of the idea of the low-dimensional adapter would be to keep the $\boldsymbol{A}$ and $\boldsymbol{B}$ frozen and train the matrix S until convergence. This achieves the benefits discussed in section 3.1. One critical issue will be the initialization of the fixed adapters PMA and PMB. VERA (Kopiczko et al., 2024) kept them frozen by initializing via Kaiming normal distribution. Despite frozen $\boldsymbol{BA}$, the downside was that it required the rank r to potentially converge to good accuracy, thus costing substantial compute for the forward and the backward pass. We on the contrary, propose to initialize PMA and PMB with the singular vectors (SVs) corresponding to the most significant singular values of the pre-trained weight. This is accomplished by applying singular value decomposition (SVD) to the pre-trained weight and extracting its **spectrum**, then initializing $\boldsymbol{A}$ and $\boldsymbol{B}$ with the SVs:

$$\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V} = SVD(\boldsymbol{W}) \tag{8}$$

$$\boldsymbol{A} = \boldsymbol{U}[:, :r]\boldsymbol{\Sigma}[:r, :r], \quad \boldsymbol{B} = \boldsymbol{V}[:, :r]^{\mathrm{T}}. \tag{9}$$

Since $\boldsymbol{B}$ forms a basis for $\mathbb{R}^r$, learning matrix $\boldsymbol{S}$ in Equation 7 can be interpreted as learning a **basis change matrix**. Previous studies (Hu et al., 2022; Li et al., 2023) have emphasized the importance of ensuring that the combined effect of the main path and the adapters approximates the pre-trained weights at the onset of fine-tuning. Accordingly, based on Equations 8 and 9, we initialize $\boldsymbol{S}$ (LDA) with the identity matrix I_r and set the main path with the last $d - r$ components of spectrum of $\boldsymbol{W}$. We note, a contemporary work (Meng et al., 2024) has suggested similar initialization of the adapters. However, our approach largely differ in primary objective, as we intend to find suitable initialization to freeze by allowing the LDA to learn. (Meng et al., 2024), on the contrary, focuses primarily on the impact of adapter initialization and does not yield any memory or compute saving compared to that with LoRA.

Having initialized all the parameters in equation 7, we perform fine-tuning by keeping the PMA always frozen and LDA always learnable. For the PMB, we present a gradual freezing strategy, to be discussed next. We hypothesize that having only a trainable LDA for simpler tasks (e.g. GLUE benchmark) would be sufficient potentially due to much lower intrinsic dimensions of the pre-trained weights, thus not necessitating any need of high trainable parameters. However, for relatively complex tasks, like summarizing, complex reasoning, we believe intrinsic dimensionality of the weight may not be very low. To tackle this challenge, we adapt a gradual freezing strategy of the PMB allowing the fine-tuned model to perform better while keeping all the benefits of LaMDA. Further analysis on the relation of task difficulty to model intrinsic low-dimensionality may be an interesting future research.

To enhance LaMDA's expressiveness while maintaining the benefits of having low parameter count and minimal activation memory, we introduce the mechanism of **gradual freezing** of PMB, as illustrated in Figure 1(c). The concept involves keeping PMB trainable during the initial iterations

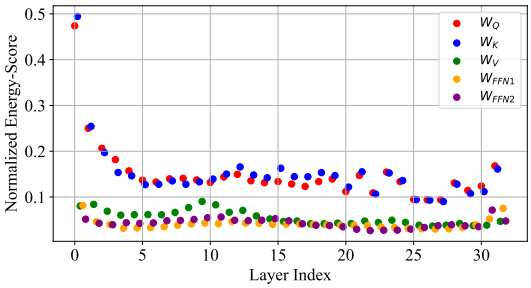

Figure 3: Layer-wise energy-score of the first 32 ranks of each linear module, normalized over the total energy-score of the same module, evaluated on a pre-trained LLaMA2-7B.

of fine-tuning and then progressively freeze PMB row by row. Previous work by (Liu et al., 2024b) has suggested gradual freezing based on fixing the scores computed during fine-tuning. In contrast, we employ a simpler heuristic to circumvent the additional computational and memory overhead associated with calculating and storing these scores. As argued in (Meng et al., 2024), learning the SVs corresponding to the most significant singular values is the most effective approach for parameter-efficient fine-tuning. Consequently, a reasonable criterion is to freeze the rows of $\boldsymbol{B}$ sequentially from the last row to the first, given that the first row represents the highest-energy component of the spectrum of PMB. So, we propose a linear schedule for the number of trainable rows in PMB as below:

$$r(t) = \begin{cases} \text{int}(r - \frac{t}{t_i}), & t \leq t_i \\ 0 & t \geq t_i \end{cases}$$

where t_i is set to be 30% of the total iterations in our experiments. Since $\boldsymbol{B}$ is an $r \times d$ matrix, the input activation that needs to be stored for back-propagation is again in the $r-$dimensional space, so the memory-saving arguments still hold. In section 4.5, we do an ablation study on the correct order of freezing the rows of PMB.

3.3 LaMDA++

This section presents LaMDA++, an enhanced version of LaMDA that incorporates the option of varying ranks across different network layers. Previous works on adaptive rank for LLMs (Zhang et al., 2023b) have introduced multiple hyperparameters, that can lead to increased training time. Furthermore, changing the rank of the matrices dynamically could result in more complex training. Conversely, We rely on a "lite-weight" static analysis for fine-tuning with adaptive rank. In specific, we analyze the normalized energy-score of the pre-trained model weights to simplify implementation and adoption.

To motivate this approach, Figure 3 reports the normalized energy-score of the first 32 singular vectors (E_r^l with $r = 32$) for each trainable linear module l of a LLaMA2-7B across all layers. The normalization factor is the total energy-score computed over all the singular vectors (E_T^l) of the corresponding module l. As the figure indicates, some modules capture significantly higher normalized energy-score than others when applying SVD to the weights. This observation suggests that, to achieve a similar normalized energy-score across all layers, the $\boldsymbol{W}_Q$ and $\boldsymbol{W}_K$ modules may require a lower rank. In contrast, the $\boldsymbol{W}_{FFN1}$, $\boldsymbol{W}_{FFN2}$, and $\boldsymbol{W}_V$ modules might necessitate a higher rank to reach an equivalent level of normalized energy-score. This can be of great importance, as previous works (Hu et al., 2023) have shown that all linear layers of the LLMs (including the attention weights) are essential to be fine-tuned.

To implement this heuristic while maintaining the same number of trainable parameters, LaMDA++ employs a pre-processing step to select the ranks of each LoRA path. Intuitively, ranks should be reduced from the budget of layers less affected by rank reduction and reallocated to layers that capture the least normalized energy-scores. Firstly, we define a rank budget set R_S, containing S potential candidate ranks, $R_S = \{r_1, ..., r_S\}$, with $r_1 < r_2 < r_S$, to be selected for a LoRA path. We ensure that the summation of all the different ranks selected for the layers gets averaged to the target rank r_T. Additionally, for a module at layer l, LaMDA++ assigns a candidacy score ν_l as,

$$\nu_l = \frac{E_{r_S}^l - E_{r_1}^l}{E_{r_T}^l} \tag{10}$$

LaMDA++ then sorts the linear modules based on the ascending order of ν_l. The initial elements of this sorted array are the layers that potentially require higher ranks to yield better energy-scores. In contrast, the later elements can potentially sacrifice rank reduction without losing significant energy. Based on this ranking, and to maintain simplicity, LaMDA++ assigns r_S to the first $\frac{1}{S}$th quantile of the sorted array, r_{S-1} to the second $\frac{1}{S}$th quantile, and so on. This heuristic approach favors allocating higher rank to modules that would benefit most and lower rank to modules that would suffer the least from rank reduction.

4 Experiments

This section evaluates LaMDA and LaMDA++ on NLU, NLG, and reasoning tasks.

4.1 Experimental Setup

Our experiments encompass a broad range of models and datasets. For NLU, we utilize DeBERTa-V3 (He et al., 2023) and conduct evaluations on the GLUE benchmark (Wang et al., 2019). For NLG, we employ BART-large (Lewis et al., 2020) and assess performance on the XSUM (Narayan et al., 2018) and CNN/DailyMail (Hermann et al., 2015) datasets. Additionally, we evaluate the LLaMA2 series (Touvron et al., 2023b) on GSM8K (Cobbe et al., 2021), Wikitext-2 (Merity et al., 2017), and a collection of commonsense reasoning datasets. Following prior works on LoRA variants (Hu et al., 2022; Zhang et al., 2023b; Meng et al., 2024), we freeze the main path of the model while treating the LoRA path according to the LaMDA methodology. LaMDA is applied to the MHSA and FFN blocks of all models, encompassing the $\boldsymbol{W}_Q$, $\boldsymbol{W}_K$, $\boldsymbol{W}_V$, $\boldsymbol{W}_{FFN1}$, and $\boldsymbol{W}_{FFN2}$ linear modules. As baselines, we compare LaMDA with full fine-tuning, LoRA, LoRA-FA (Zhang et al., 2023a), AFLoRA (Liu et al., 2024b), and VERA (Kopiczko et al., 2024). Our implementation of LaMDA is based on HuggingFace's Transformers library (Wolf et al., 2019), and all experiments are conducted on a single NVIDIA A6000 GPU.

4.2 Encoder-only Model: DeBERTa-V3

We fine-tuned DeBERTa-V3 (He et al., 2023) using LaMDA and LaMDA++ on the GLUE NLU benchmark. For LaMDA, the rank of the adapter path is set to 32, and in LaMDA++, the target rank r_T is set to 32 as well. For LaMDA++, the set of potential candidate ranks is $R_S = \{16, 24, 32, 40, 48\}$. For further details on experimental hyperparameters please refer to Appendix A. Table 2 presents the performance and the number of trainable parameters for LaMDA, LaMDA++, and SOTA PEFT methods. As shown in the Table, LaMDA achieves performance close to LoRA with a 17.7× reduction in the number of trainable parameters. Similarly, LaMDA achieves reductions of 17×, 2.1×, and 1.8× compared to AdaLoRA, VERA, and AFLoRA, respectively. Furthermore, LaMDA++ achieves SOTA performance with only a negligible increase in the parameter count. Please note, here we trained the LDA only while keeping the PMA, PMB frozen throughout the fine-tuning period.

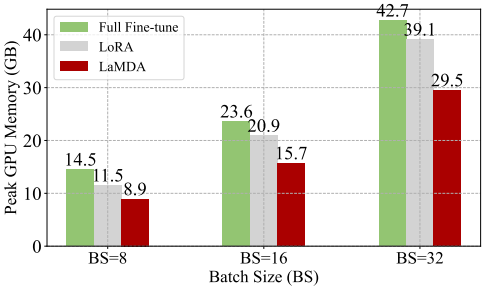

Figure 4: Peak GPU memory usage during fine-tuning BART-large on XSUM dataset.

4.3 Encoder-Decoder Model: BART-large

For the text summarization tasks, we utilize the BART-large model (Lewis et al., 2020) and fine-tune it on the XSUM (Narayan et al., 2018) and CNN/DailyMail (Hermann et al., 2015) datasets using LaMDA. The selected rank and the set R_S are the same as those used for DeBERTa-V3. The low-rank path is added parallel to the math path of the MHSA and FFN blocks of the encoder and decoder across all model layers. As mentioned in section 3.2, here we freeze PMA, keep LDA trainable, and gradually freeze PMB. The hyperparameter t_i is set to be 30% of the total training iterations. For evaluation, we report the ROUGE-1, ROUGE-2, and ROUGE-L scores (R1/2/L) (Lin, 2004). Table 3 showcases the number of trainable parameters and the performance of LaMDA and LaMDA++. Compared to LoRA, LaMDA achieves comparable performance while requiring 10× fewer parameter updates. LaMDA++ surpasses LoRA on the XSUM dataset and performs similarly to it on CNN/DailyMail. The hyperparameters used for fine-tuning are provided in appendix A.

To better understand the memory saving of LaMDA, we profile the total memory usage of fine-tuning BART-large on the XSUM dataset for full fine-tuning, LoRA, and LaMDA across different batch sizes. Figure 4 shows the peak GPU memory usage for various methods. In specific, LaMDA provides a peak memory saving of up to 1.32× to fine-tune the BART-large, profiled for different batch-sizes. This saving is primarily due to reduced memory required for activations. Such system-level benefit allows us to fine-tune larger models with larger batch sizes.

4.4 Decoder-only Model: LLaMA2

We fine-tune and evaluate LLaMA2-7B (Touvron et al., 2023b) on complex reasoning task GSM8K

Table 2: Comparison of different fine-tuning methods for DeBERTa-V3 on GLUE benchmark.

Method	#Params.	CoLA ↑	SST-2 ↑	MRPC ↑	QNLI ↑	STS-B ↑	RTE ↑	MNLI ↑	QQP ↑	Avg. ↑
FFT	184M	69.21	95.64	89.22	93.78	91.59	82.49	89.98	92.05/89.31	87.82
LoRA (r = 8)	1.33M	69.73	95.57	89.71	93.76	**91.86**	85.32	90.46	91.95/89.26	88.38
AdaLoRA	1.27M	70.86	95.95	90.22	94.28	91.39	87.36	90.30	92.13/88.41	88.83
VERA	0.16M	70.74	95.18	90.93	93.58	91.08	87.36	90.22	90.69/87.63	88.53
AFLoRA (r = 4)	0.14M	72.01	96.22	**91.91**	**94.42**	91.84	**88.09**	90.17	90.81/87.77	89.23
LaMDA (r = 32)	**0.075M**	71.60	95.70	90.44	93.72	91.30	87.50	90.05	90.70/87.70	88.87
LaMDA++ (r_T = 32)	**0.078M**	**72.12**	**96.25**	91.65	94.30	91.55	88.01	**90.56**	90.80/87.75	**89.28**

Table 3: Comparison of fine-tuning methods for Bart-large. NR denotes not reported. The three values in the last column correspond to R1/R2/RL scores.

Method	#Params(M)	XSUM	CNN/DailyMail
Full fine-tuning	415	**45.14/22.27/37.25**	44.16/21.28/40.90
LoRA	8.6	43.95/20.72/35.68	**45.03**/21.84/42.15
AdaLoRA	8.6	44.72/21.46/36.46	45.00/**21.89/42.16**
AFLoRA	5.1	NR	43.96/21.06/NR
LaMDA (LDA-only)	**0.20**	40.64/18.11/33.20	40.92/17.53/38.1
LaMDA (r=32)	0.85	43.92/20.68/35.21	44.12/21.16/40.45
LaMDA++ (r_T=32)	0.92	44.32/21.08/36.10	45.01/21.85/42.15

(Cobbe et al., 2021) and token generative task Wikitext-2 (Merity et al., 2017) using LaMDA and LaMDA++. The low-rank path is incorporated into the $\boldsymbol{W_Q}$, $\boldsymbol{W_K}$, $\boldsymbol{W_V}$, $\boldsymbol{W}_{FFN1}$, and $\boldsymbol{W}_{FFN2}$ matrices in all layers of the model. The hyperparameter t_i is set to 30% of the total fine-tuning iterations. For the LoRA and LaMDA experiments, the rank r is set to 16 and 32, respectively, while the set of potential ranks in LaMDA++ is $R_S = \{16, 24, 32, 40, 48\}$. We report accuracy for GSM8K and perplexity for Wikitext-2. The results are reported in Table 4; LaMDA and LaMDA++ both surpass LoRA on GSM8K complex reasoning task. And for the Wikitext-2, LaMDA achieves a very close perplexity to that of LoRA, and LaMDA++ outperforms LoRA, while fine-tuning with 5.5× fewer trainable parameters. This clearly shows the efficacy of LaMDA in yielding improved performance even for complex generative tasks.

Table 4: Comparison of fine-tuning results for LLaMA2-7B on GSM8K and Wikitext-2.

Method	#Params(M)	GSM8K ↑	Wikitext-2 ↓
LoRA ($r = 16$)	28	36.9	5.43
LaMDA (r=32)	**4.37**	37.9	5.45
LaMDA++ (r_T=32)	5.12	**38.2**	**5.41**

We also evaluate the performance of LaMDA on commonsense reasoning. We follow the settings in (Hu et al., 2023) and use the Commonsense170K dataset as a combination of training examples of various tasks. Then we evaluate the fine-tuned model on the validation set of each task separately. The collection includes samples of the following datasets: BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), the HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2020), ARC-e and ARC-c (Clark et al., 2018), and OBQA (Mihaylov et al., 2018). For this experiment, the set R_S of LaMDA++ is $\{32, 48, 64, 80, 96\}$. The fine-tuning results are shown in Table 8. LaMDA achieves a higher average accuracy than LoRA, while fine-tuning $\sim 11.5\times$ less parameters.

4.5 Ablations and Discussions

Impact of initialization choices. A primary step in LaMDA involves initializing the PMA and PMB with the singular vectors (SVs) of the pre-trained weight $\boldsymbol{W}$. LaMDA utilizes the SVs corresponding to the most significant singular values because, according to SVD theory, these vectors capture the highest proportion of the matrix's total energy-score compared to any other set of r SVs. Consequently, fine-tuning these vectors has the most significant impact on adaptation. To verify this hypothesis and validate the findings of (Meng et al., 2024), we also initialize PMA and PMB with the set of SVs associated with the smallest singular values.

Conversely, VERA (Kopiczko et al., 2024) initializes the adapters randomly and keeps them frozen. An insightful ablation study would examine the performance of LaMDA when PMA and PMB are initialized randomly, with PMA frozen at the beginning and PMB gradually frozen over time. In this scenario, LDA is initialized to a zero matrix instead of I_r, ensuring that the combined effect of the main path and the adapter path equals the main path at the onset of fine-tuning.

We fine-tune LLaMA2-7B on GSM8K and Wikitext-2 using the three discussed initialization methods and report the results in Table 6. For random initialization, we perform Kaiming normal

Table 5: Commonsense reasoning results for LLaMA2-7B

Method	#Params.(M)	BoolQ ↑	PIQA ↑	SIQA ↑	HellaSwag ↑	WinoGrande ↑	ARC-e ↑	ARC-c ↑	OBQA ↑	Avg. ↑
LoRA (r=32)	56	69.8	79.9	79.5	83.6	82.6	79.8	64.7	81.0	77.6
LaMDA (r=64)	**4.85**	71.6	80.3	79.1	**84.0**	82.4	**81.5**	65.8	79.6	78.0
LaMDA++ (r_T=64)	5.65	**71.8**	**80.6**	**79.5**	**84.0**	**82.7**	**81.5**	**66.0**	**80.6**	**78.3**

initialization for both PMA and PMB. The remaining hyperparameters are consistent with those in Section 4.4. The Table shows that LaMDA initialized with the first r SVs outperforms the random initialization when using the same r. Additionally, random initialization surpasses the model initialized with the last r SVs. The r is set to 32 for this ablation study. The result underscores the critical impact of fine-tuning the high-energy components of the model.

Table 6: Effect of the initialization in LaMDA.

Initialization	#Params(M)	GSM8K ↑	Wikitext-2 ↓
First r SV	4.37	**37.9**	**5.45**
Last r SV	4.37	35.8	5.55
Kaiming normal	4.37	37.1	5.49

Number of iterations t_i in gradual freezing. LaMDA freezes PMB in t_i first iterations of fine-tuning based on linear schedule. Adjusting this hyperparameter (t_i) significantly alters the effective number of trainable parameters (#Params), as the size of PMB ($r \times d$) is considerably larger than that of LDA ($r \times r$). To investigate the impact of this hyperparameter, we conducted the GSM8K experiment using LLaMA2-7B with various values of t_i. We present the resulting #Params and accuracy in Table 7. By comparing these results with those in Table 4, we observe that allocating a sufficient number of iterations to training PMB is crucial for surpassing LoRA. Specifically, LaMDA with t_i set to 10% of the total iterations fails to outperform LoRA, whereas allocating 20% and 30% of the iterations to PMB training results in superior performance relative to LoRA. In the appendix B, we explain how to count the effective number of trainable parameters (#Params).

Effect of the LaMDA++ ranking. As explained in Section 3.3, LaMDA++ generates a sorted list of all linear modules based on the candidacy score ν. We conduct an essential study to validate the effectiveness of such sorting. First, we allocate ranks according to the list generated by LaMDA++, assigning more ranks to layers with smaller scores. Subsequently, we conduct another

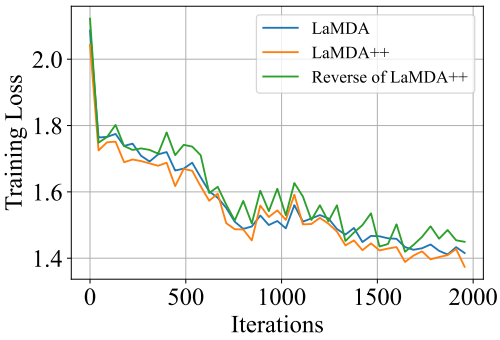

Figure 5: Training Curve of LLaMA2-7B on Wikitext-2.

experiment where ranks are allocated in the reverse order of LaMDA++, assigning more ranks to layers with higher scores. The training curves for this experiment are shown in Figure 5. The results indicate that LaMDA++ with reverse ordering exhibits noisier training behavior and ends with a higher loss value, translating into higher perplexity on Wikitext-2. Among the three approaches presented in the figure, LaMDA++ demonstrates the lowest training loss, attributable to its appropriate allocation of the rank budget.

Table 7: Effect of the fine-tuning iteration % before freezing PMB.

t_i	#Params(M)	Accuracy ↑
10% of iterations	1.56	36.1
20% of iterations	2.97	37.0
30% of iterations	4.37	**37.9**

5 Conclusion

In this work, we proposed LaMDA, a novel framework for fine-tuning large language models. LaMDA employs a low-dimensional adapter, significantly reducing the number of trainable parameters and conserving activation memory. The methodology involves freezing the projection matrix $\boldsymbol{A}$ from the outset and gradually freezing the projection matrix $\boldsymbol{B}$. We further enhanced LaMDA by incorporating the flexibility of varying ranks across layers, allocating ranks to adapters based on the energy components of the pre-trained weights. Both LaMDA and LaMDA++ demonstrate the capability to facilitate the fine-tuning of larger models on commercial GPUs, offering an efficient and scalable approach to model adaptation.

6 Limitations

This study has a few limitations. Firstly, the largest model we tested was LLaMA2-7B. Due to time constraints associated with the paper's deadline, we could not extend our experiments to larger models, which could provide further insights into the scalability and effectiveness of LaMDA. Our methodology, LaMDA, has not yet been tested on instruction-following tasks. While the current results are promising, evaluating the performance of LaMDA in these specific tasks is essential to fully understanding its potential and versatility. We plan to address these limitations in future work by conducting experiments on larger models and a broader range of tasks. We are also eager to test the applicability of our method to vision-language models, which was not explored in this paper.

References

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 7319–7328. Association for Computational Linguistics.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. PIQA: reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 7432–7439. AAAI Press.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *CoRR*, abs/2005.14165.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *Preprint*, arXiv:1905.10044.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *Preprint*, arXiv:1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Nigel Doering, Cyril Gorlla, Trevor Tuttle, and Adhvaith Vijay. 2024. Empirical analysis of efficient fine-tuning methods for large pre-trained language models. *CoRR*, abs/2401.04051.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2023. A framework for few-shot language model evaluation.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021. Debertav3: Improving deberta using electra-style pretraining with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2023. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 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 *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. 2023. Llm-adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 5254–5276. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki Markus Asano. 2024. Vera: Vector-based random matrix adaptation. *CoRR*, abs/2310.11454.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 3045–3059. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7871–7880. Association for Computational Linguistics.

Yixiao Li, Yifan Yu, Chen Liang, Pengcheng He, Nikos Karampatziakis, Weizhu Chen, and Tuo Zhao. 2023. Loftq: Lora-fine-tuning-aware quantization for large language models. *CoRR*, abs/2310.08659.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024a. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*.

Zeyu Liu, Souvik Kundu, Anni Li, Junrui Wan, Lianghao Jiang, and Peter Anthony Beerel. 2024b. Aflora: Adaptive freezing of low rank adaptation in parameter efficient fine-tuning of large models. *CoRR*, abs/2403.13269.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2023. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *CoRR*, abs/2308.08747.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. 2024. Pissa: Principal singular values and singular vectors adaptation of large language models. *Preprint*, arXiv:2404.02948.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? A new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 2381–2391. Association for Computational Linguistics.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1797–1807. Association for Computational Linguistics.

Rui Pan, Xiang Liu, Shizhe Diao, Renjie Pi, Jipeng Zhang, Chi Han, and Tong Zhang. 2024a. LISA: layerwise importance sampling for memory-efficient large language model fine-tuning. *CoRR*, abs/2403.17919.

Rui Pan, Xiang Liu, Shizhe Diao, Renjie Pi, Jipeng Zhang, Chi Han, and Tong Zhang. 2024b. Lisa: Layerwise importance sampling for memory-efficient large language model fine-tuning. *Preprint*, arXiv:2403.17919.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8732–8740. AAAI Press.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. 2019. Socialiqa: Commonsense reasoning about social interactions. *CoRR*, abs/1904.09728.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

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 *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4791–4800. Association for Computational Linguistics.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. 2023a. Lora-fa: Memory-efficient low-rank adaptation for large language models fine-tuning. *CoRR*, abs/2308.03303.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023b. Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*. Openreview.

A Training Details

Table 8: Hyperparameters for fine-tuning DeBERTa-V3 on GLUE benchmark

Hyperparameter	CoLA	SST-2	MRPC	QNLI	STS-B	RTE	MNLI	QQP
Learning rate	1e-2	4e-3	8e-2	4e-3	2e-2	4e-2	4e-3	4e-3
#Epochs	20	10	20	10	20	20	10	10
Max Seq. Len.	512	512	512	512	512	512	512	512

Table 9: Hyperparameters for fine-tuning LLaMA2-7B

Hyperparameters	GSM8K	Wikitext-2	Commonsense170k
Learning rate	3e-4	3e-4	3e-4
#Epochs	6	2	3
Batch size	16	16	16

Table 10: Hyperparameters for fine-tuning BART-large

Hyperparameters	XSUM	CNN/DailyMail
Learning rate	2e-4	2e-4
#Epochs	25	15
Batch size	32	64

Here we provide the implementation details and the hyperparameters used for training. In all experiments, we used the PyTorch framework and ADAM (Kingma and Ba, 2015) optimizer.

A.1 DeBERTa-V3

To fine-tune DeBERTa-V3 on the GLUE benchmark, we use a batch size of 32 and use the following setup for the learning rate and number of epochs, which are similar to what (Li et al., 2023) used.

A.2 BART-large

For fine-tuning BART-large on XSUM and CNN/DailyMail we set the maximum input sequence to 1024 and the maximum target sequence to 128. Learning rate, number of epochs, and batch size are shown in the Table 10, which are similar to what (Li et al., 2023) used.

A.3 LLaMA2-7B

We follow the setting of (Li et al., 2023) to fine-tune LLaMA2-7B on GSM8K and Wikitext-2 datasets. Moreover, we adopt the Commonsense170K dataset from (Hu et al., 2023) and use the default setup to fine-tune LLaMA2-7B for commonsense reasoning. For evaluation, we use lm-evaluation-harness library (Gao et al., 2023). The hyperparameters are provided in Table 9.

B Effective number of trainable parameters (#Params) in LaMDA

Assuming L trainable linear modules in the model, t_i initial iteration for gradual freezing, and T total iterations, the effective number of trainable parameters can be computed as below:

$$\#\text{Params} = \sum_{l=1}^{L}[\frac{t_i}{T} \times \frac{\text{NP}(PMB_l)}{2} + \text{NP}(LDA_l)] \tag{11}$$

where $\text{NP}(\boldsymbol{X})$ is a function that counts the number of trainable elements in the matrix $\boldsymbol{X}$; PMB_l and LDA_l are the projection matrix B and low-dimensional adapter in the linear module l.