A Study of Parameter Efficient Fine-tuning by Learning to Efficiently Fine-Tune

Taha Ceritli[1], Savas Ozkan[1], Jeongwon Min[2], Eunchung Noh[2], Jung Min Cho[2], Mete Ozay[1]

[1]Samsung R&D Institute UK (SRUK), [2]Samsung Electronics Korea

Correspondence: {t.ceritli, m.ozay} @samsung.com

Abstract

The growing size of large language models (LLMs) requires parameter-efficient fine-tuning (PEFT) methods for their adaptation to new tasks. Existing methods, such as Low-Rank Adaptation (LoRA), typically involve model adaptation by training the PEFT parameters. One open problem required to be solved to effectively employ these methods is the identification of PEFT parameters. More precisely, related works identify PEFT parameters by projecting high dimensional parameters of LLMs onto low dimensional parameter manifolds with predefined projections, or identifying PEFT parameters as projections themselves. To study this problem, we propose a new approach called Learning to Efficiently Fine-tune (LEFT) where we aim to learn spaces of PEFT parameters from data. In order to learn how to generate the PEFT parameters on a learned parameter space while fine-tuning the LLMs, we propose the Parameter Generation (PG) method. In the experimental analyses, we examine the effectiveness of our solutions exploring accuracy of fine-tuned LLMs and characteristics of PEFT parameters on benchmark GLUE tasks.

1 Introduction

Most natural language processing applications today rely on pre-trained large language models (LLMs), which are designed for general use. Even so, one can enhance their adaptation to a new specific task through a fine-tuning process in which their parameters are updated with fine-tuning data. However, fine-tuning all the parameters of modern LLMs is highly computationally complex due to the need for increased computational resources.

The problem of efficient fine-tuning of models has been studied (Aghajanyan et al., 2021) by exploring their objective (Li et al., 2018), or in general, optimization landscape (Ozay, 2019). For training a model with D dimensional parameters, Li et al. (2018) and Ozay (2019) suggest

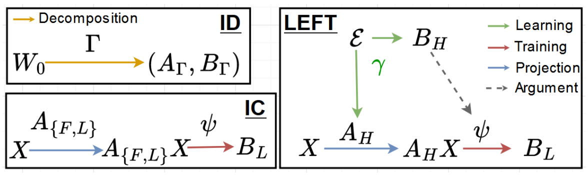


Figure 1: A comparison of different approaches for identifying PEFT parameters. Unlike identification by decomposition (ID) and construction (IC), our approach (LEFT) learns A (A_H) and B (B_H) parameters with a hypernetwork γ from the embeddings $\mathcal{E}$. The learned B_H can be further trained using projected data A_HX with a task objective. In ID, W_0 is decomposed to (A_Γ, B_Γ) with a deterministic function Γ. In IC, a random projection A_F or trained A_L is used to project input data X (or the gradients $\nabla_{W_0}L$) to low-dimensional manifolds to train B_L.

performing optimization on a lower dimensional ($\hat{D} << D$) manifold of the parameters, where the solutions to the model training problem first appear. Aghajanyan et al. (2021) proposed that measuring intrinsic dimension can provide information on the number of free parameters which are required to closely approximate the optimization problem that is solved while fine-tuning BERT models.

In the context of LLMs, this approach has been utilized for developing parameter efficient fine-tuning (PEFT) methods (Hu et al., 2022; Xu et al., 2023; Lialin et al., 2023a) by optimizing only a small subset of parameters while keeping the remaining parameters W_0 of the LLMs frozen. Inspired from Li et al. (2018); Aghajanyan et al. (2021), Hu et al. (2022) hypothesized that the change in parameters during fine-tuning also has a low intrinsic dimension. Following this hypothesis, they proposed a method named low-rank adaptation (LoRA) to compute low-rank approximation of the change ΔW. To employ LoRA methods effectively, their fine-tuning parameters should be well-designed.

The characteristics of fine-tuning parameters: Despite the recent efforts (Fan et al., 2024; Hao et al., 2024), the characteristics of these parameters (e.g. their roles in adaptation process, impact on outcome of fine-tuning, and their relationship) are still not fully understood. For instance, LoRA methods constrain fine-tuning of a pre-trained model with a parameter matrix $W_0 \in \mathbb{R}^{D \times K}$ by a low-rank decomposition $W_0 + \Delta W = W_0 + BA$ with $A \in \mathbb{R}^{R \times K}$ and $B \in \mathbb{R}^{D \times R}$. Inspired from the intrinsic dimension of manifolds, $R << \min(D, K)$ is described as the rank of A and B.

In the literature, various methods and configurations have been proposed to define characteristics of A and B under two approaches. These configurations are illustrated in Figure 1.

(1) Identification by decomposition (ID) where A and B are mapped onto low-dimensional manifolds by projection: Babakniya et al. (2023) applied a decomposition method, such as Singular Value Decomposition (SVD), to assure that the A and B matrices reside on low-dimensional manifolds (e.g. the Stiefel manifold (Ozay, 2019)). However, performing SVD for LLMs can be computationally complex. To address the complexity problem, model compression methods such as quantization can be applied (Guo et al., 2024). However, accuracy of the solutions decreases as more aggressive quantization schemes are applied.

(2) Identification by construction (IC) where A and B are projections mapping high dimensional input onto low-dimensional manifolds: The parameters are identified as matrices with special structures by construction

Zhu et al. (2024) observed that A applies a random projection on features. Then, B is utilized to learn features on low-dimensional manifolds of projected features. Furthermore, the authors reveal their asymmetric behavior by demonstrating that fine-tuning B is inherently more effective than fine-tuning A, and that a random untrained A should perform nearly as well as a fine-tuned one. Note that fixing randomly initialized A and training only B can also help to avoid mismatch issues when merging LoRA parameters (Sun et al., 2024).

Conversely, Hao et al. (2024) explained the role of A through random projection of ΔB when small learning rates are utilized. When the random A matrices are frozen and only B parameters are updated using gradients with upper-bounded norms, the A matrices project $\nabla_{W_0} L$ onto lower-dimensional manifolds. Following these results, Hao et al. (2024) proposed optimizing parameters W_0 on low-dimensional manifolds of rank-deficient gradients as an alternative approach to the LoRA.

In this work, we address the PEFT problem for LLMs in computationally constrained settings such as for mobile and edge devices. Therefore, we pose the problem of PEFT for LLMs using frozen W_0 different from that investigated by the ID. Similar to the ID, we aim to update low-dimensional approximation of original parameters instead of W_0. More precisely, we aim to learn a low-dimensional representation of newly introduced data during fine-tuning, which is then integrated to W_0 for inference. Following this constraint, we identify A and B parameters by construction as in the IC instead of decomposing W_0. Thereby, we consider their approximation BA as a representation learned during fine-tuning, and we aim to *learn* these representations efficiently. For this purpose, we first study the following question (Q1):

Q1: How should we identify the structure of A and B for PEFT of LLMs?

Learning to Efficiently Fine-tune: To elucidate this question, we introduce a new approach named Learning to Efficiently Fine-tune (LEFT). In the LEFT, similar to the ID, we consider studying the characteristics of BA on low-dimensional parameter manifolds. However, we aim to learn these manifolds by parameter generation using generative models, such as hypernetworks. In other words, we target *learning to generate PEFT parameters* residing on low-dimensional parameter manifolds for learning new representations unlike ID and IC as depicted in Figure 1.

To this end, we propose a method called Parameter Generation (PG) to learn how to generate PEFT parameters in LEFT. Deutsch et al. (2019) showed that a hypernetwork can learn to generate a distribution of parameters of a deep neural network on a non-trivial manifold, and the hypernetwork can be considered as the coordinate map of this low-dimensional manifold (Shamsian et al., 2021). Therefore, during fine-tuning, we train hypernetworks to learn how to generate the PEFT parameters on low-dimensional manifolds.

Similar to the IC, we explore random structures of matrices identifying A. Our PG enables us to control how to utilize different random and deterministic structures for identifying A and B by parameter generation. In the experiments, we consider several configurations where the parameters are randomly generated (as in the related work), optimized (as utilized by LoRA), and generated by conditioning on network properties (e.g. layers and embeddings). However, designing PEFT parameters according to network properties is another understudied open problem as addressed next.

Shareability of PEFT parameters among the layers of LLMs: Existing works applying PEFT parameters to LLMs, including the studies focusing on the role of A with random projections (Fan et al., 2024; Hao et al., 2024), follow the standard practice of defining separate matrices for each target layer. Such applications do not explicitly consider the hierarchy of the layers and their roles, which are investigated in various studies.

For instance, Durrani et al. (2023) show that basic lexical details such as suffixation or word structure are concentrated at the lower layers of a pre-trained model, whereas non-local dependencies are primarily captured at the higher layers. This may also be connected to why "easy" tasks activate the neurons of LLMs at shallower layers while "hard" ones at deeper layers, as inspected by Fan et al. (2024). Motivated by these observations, we explore the next follow-up question:

Q2: How does shareability and dependency of representations learned at different layers guide designing PEFT parameters and affect their accuracy?

In order to investigate this question, we first analyze distribution of parameters trained by LoRA and that of parameters generated by our PG. Then, we examine the accuracy of fine-tuned LLMs with parameters generated by sharing hypernetworks among different layers of the LLMs.

Our contributions can be summarized as follows:

- We propose a new approach, called LEFT, to study the fine-tuning parameter identification problem. To employ this approach for fine-tuning LLMs in resource constraints setups, we propose the PG method which implements hypernetworks for generating PEFT parameters.
- In the experimental analyses of the question Q1, PG improves the accuracy of LoRA methods that identify A parameters as random projections by up to 16% using just 65K more trainable parameters for the benchmark GLUE tasks. Moreover, PG achieves similar accuracy using 30K less parameters compared to these LoRA methods.
- Experimental analyses of the question Q2 reveal different shareability patterns for LLMs fine-tuned for different tasks. For instance, we observe that shareability of hypernetworks among layers of the LLMs can consistently improve accuracy for the MNLI, MRPC, QQP, RTE and STS-B tasks. However, the accuracy fluctuates or does not change remarkably for the other tasks.

Organization Our paper is organized as follows. We introduce our Parameter Generation (PG) method in Section 2, which is followed by our experimental analyses (Section 3). Section 4 concludes the paper.

2 Parameter Generation for LEFT

Suppose that W_0 represents the parameters of a pre-trained LLM kept frozen during fine-tuning. Parameter Generation (PG) produces the fine-tuned model parameters W through a non-linear transformation function (NTF) f by

$$W = f(W_0, \gamma(\mathcal{E}; \Theta_\gamma)), \tag{1}$$

where $\mathcal{E}$ is a set of embeddings assigned to the layers of a pre-trained LLM, and γ is a hypernetwork parametrized by Θ_γ. In Eq. (1), we compute $f(W_0, \gamma(\mathcal{E}; \Theta_\gamma)) = W_0 + BA$, where A and B are obtained from the hypernetwork γ by $(A, B) = \gamma(\mathcal{E}; \Theta_\gamma)$. Thereby, the nonlinearity of f is attributed to that of γ. Notice that PG offers a flexible approach for PEFT, that can be designed in different ways based on the choices of f, γ, and $\mathcal{E}$.

Figure 2 presents an overview of our method where W_0^l denotes the fixed parameters at the l^{th} layer of a pre-trained LLM. The corresponding parameters W_a^l are obtained by passing the embeddings $\{e^l, p^l\} \in \mathcal{E}$ through the hypernetwork γ. Finally, the function f combines these parameters to produce the final fine-tuned parameters W. Thereby, the function f *learns how to fine-tune the parameters* by training the hypernetwork γ which generates fine-tuning parameters, and integrating the generated parameters with W_0.

In this work, we use our PG to generate a separate set of PEFT parameters for each layer of a pre-trained LLM, as shown in Figure 3. Unlike LoRA where the parameters A and B are trained during fine-tuning, we train a generative model to learn how to generate and integrate them by updating the NTF f, which is described next.

2.1 The non-linear transformation function

The main block of PG is the non-linear transformation function (NTF) f. Given a set of embeddings

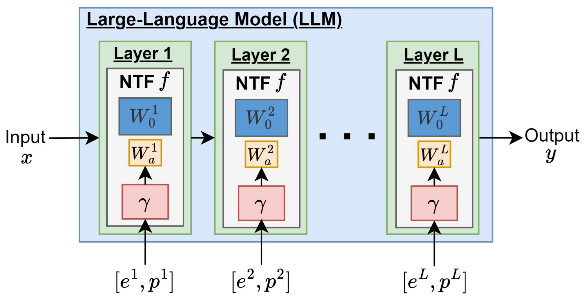


Figure 2: **A diagram describing employment of the PG method.** W_0^l denotes the fixed parameters at the l^{th} layer of a pre-trained LLM. The corresponding parameters W_a^l are obtained by passing the embeddings $\{e^l, p^l\} \in \mathcal{E}$ through the hypernetwork γ. Finally, the function f combines these parameters to produce the final fine-tuned parameters W using the Eq. (1).

$\mathcal{E}$ and a hypernetwork γ, it aims to produce the fine-tuned model parameters W by feeding the embeddings into the hypernetwork. Below, we describe these components and their roles.

Layer and positional embeddings ($\mathcal{E}$): The goal of the embeddings is to incorporate the sequential order of the layers so that the transformation function f can produce distinct layer-wise fine-tuning parameters. In this work, we develop two types of embeddings which are (i) layer embeddings (fixed or trainable) and (ii) positional embeddings (fixed).

Layer embeddings (e^l): Perhaps the simplest approach to characterize the sequential order of the layers is to represent a layer of an LLM by a randomly generated embedding vector, i.e., $e^l \in \mathbb{R}^{h_e}$ for the l^{th} layer where h_e denotes its dimension (note that the dimension h_e depends on the design of hypernetwork γ, as the embeddings form the inputs to the hypernetwork). Unfortunately, randomly generated embeddings, whether trained or not, do not necessarily reflect the order of layers.

Positional embeddings (p^l): We complement layer embeddings e^l with positional embeddings $p^l \in \mathbb{R}^{h_p}$, which are carefully chosen to follow the sequential order of the layers. Inspired from transformers (Vaswani et al., 2017), the idea behind using positional embeddings is to describe the location of a layer, similarly to describing the position of a word in a sequence in transformers.

Let l be the location of a layer in a pre-trained LLM with L layers. Then, the h_p dimensional positional embedding $p^l \in \mathbb{R}^{h_p}$ is defined by

$$p^l = \left[\sin(w_i \cdot l), \cos(w_i \cdot l)\right]_{i=1}^{h_p/2}, \qquad (2)$$

where $\sin(\cdot)$ and $\cos(\cdot)$ are respectively the sine and cosine functions, with $w_k = \frac{1}{10000^{2k/h_p}}, \forall k$.

Hypernetworks (γ): A hypernetwork (Ha et al., 2017) is a neural network that generates the parameters of another, typically larger, neural network performing a target task. The inputs of a hypernetwork can be chosen to generate personalized, task-specific or layer-specific parameters, making them applicable in various domains such as language modeling (Suarez, 2017), computer vision (Klocek et al., 2019), continual learning (von Oswald et al., 2020), multi-tasking (Tay et al., 2021), and personalized federated learning (Shamsian et al., 2021).

Our motivation behind using hypernetworks is to facilitate information sharing across the layers of a pre-trained LLM, following their sequential order which are represented by the embeddings $\mathcal{E}$. Passing the embeddings through the same hypernetwork enables its parameters Θ_γ to be shared while generating the layer-wise parameters $W_a^l = \{A^l, B^l\}, \forall l \in [L]$.

A few studies utilize hypernetworks to generate LoRA parameters such as HyperDreamBooth (Ruiz et al., 2024) for personalized text-to-image generation, PIHLoRA (Majumdar et al., 2023) for solving partial differential equations, and HyperTuning (Phang et al., 2023) for multi-task fine-tuning of language models. These methods show the generalization capabilities of hypernetwork-based LoRA for a diverse set of applications. However, the proposed hypernetworks rely only on data or its extracted features by discarding the relationship across layers. Conversely, our approach considers layer indices and their positional information using different random and deterministic structures.

Our hypernetwork architecture: We implement the hypernetwork γ based on fully connected networks parametrized by Θ_γ. Specifically, we build the hypernetwork γ with two linear layers where the weights are initialized either using Kaiming uniform distribution or zero initialization.

2.2 Conditional Parameter Generation

We generate parameters conditioned on three types of embeddings described above. To this end, we develop three PG schemes to utilize embeddings at each l^{th} layer of a pre-trained LLM to better understand their impact on fine-tuning performance:

PG-Fixed initializes the layer embeddings e^l using the standard Gaussian distribution $\mathcal{N}(0, 1)$, which are kept frozen during fine-tuning.

PG-Trainable is implemented similarly to PG-

Fixed except that the layer embeddings e^l are trained during fine-tuning.

PG-Pos extends the PG-Trainable by concatenating fixed positional embedding p^l with the layer embedding e^l defined earlier.

2.3 Layer-wise Parameter Generation

Once the components f, $\mathcal{E}$, and γ are specified, PG lets us produce layer-wise PEFT parameters, as shown in Figure 3. The function f utilizes both W_0^l and W_a^l by transforming the embedding e^l and $p^l, \forall l$. In LoRA, the parameters W_a^l are optimized during fine-tuning. Instead, PG enables us to generate the parameters A^l, B^l or both for all layers during fine-tuning. Figure 3 illustrates how PG can generate both A^l and B^l at the l^{th} layer of an LLM by optimizing the parameters Θ_γ during fine-tuning to learn how to generate A^l and B^l. In different configurations, the generated parameters can be further fine-tuned or identified as random matrices as considered in the ID and IC. In the experimental analyses, we explore the effect of different configurations on accuracy.

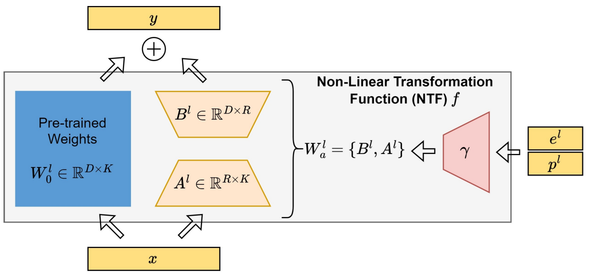

y
$\oplus$
Non-Linear Transformation Function (NTF) f
Pre-trained Weights
$W_0^l \in \mathbb{R}^{D\times K}$
$B^l \in \mathbb{R}^{D\times R}$
$A^l \in \mathbb{R}^{R\times K}$
$W_a^l = \{B^l, A^l\}$
γ
e^l
p^l
x

Figure 3: **Generating layer-wise PEFT parameters** through the non-linear transformation function f. W_0^l denotes the fixed parameters at the l^{th} layer of a pre-trained model. The corresponding fine-tuning parameters W_a^l are generated by the hypernetwork γ conditioned on the layer e^l and positional p^l embeddings.

2.4 Computational Complexity of Parameter Generation

PEFT methods such as LoRA reduce the memory usage during fine-tuning by decreasing the number of trainable parameters. However, the independent construction of LoRA parameters scales the number of trainable parameters linearly by the number of layers in an LLM, which is not desirable. PG allows us to avoid this issue by using hypernetworks.

Table 1 presents the number of trainable parameters. For a given pre-trained LLM with L layers, the independent application of LoRA with rank R leads to a total number of trainable parameters $L \times R \times (D + K)$ where $\{D, K\}$ denote the dimensions of a frozen layer.

Method	Number of Trainable Parameters
LoRA	$L \times R \times (D + K)$
PG-Fixed	$h_2 \times R \times (D + K) + h_1 \times h_2$
PG-Trainable	$h_2 \times R \times (D + K) + h_1 \times h_2 + L \times h_1$
PG-Pos	$h_2 \times R \times (D + K) + h_1 \times h_2 + L \times h_1 \times 2$

Table 1: **Parameter size of LoRA and three PG variants** for a pre-trained LLM with L layers where R denotes the rank, while h_1 and h_2 denote the hidden dimensions of the layers of a hypernetwork.

Let us first consider PG-Fixed where the layer embeddings e^l are kept frozen for each layer of an LLM. If the first layer of the model γ has dimensions $\{h_1, h_2\}$, then its second layer would have the dimensions $\{h_2, R \times (D + K)\}$.Therefore, the number of trainable parameters becomes $h_1 \times h_2 + h_2 \times R \times (D + K)$. For PG-Trainable, we have an additional term of $L \times h_1$ as the layer embeddings e^l are now updated during fine-tuning. Similarly, PG-Pos has an extra term for the positional embeddings p^l. Even though the positional embeddings are kept frozen, they are concatenated with the layer embeddings e^l, resulting in additional weights in the first linear layer of the hypernetwork γ.

Table 1 provides two main insights into the complexity of PG. First, the complexity of our approach largely depends on the dimension h_2 rather than h_1. Second, choosing h_2 lower than L (i.e., $h_2 < L$) reduces the parameter size.

3 Experiments

We give a description of the setup used throughout the experiments in Section 3.1, which is followed by a motivating example for our approach (Section 3.2). We then explore the questions Q1 and Q2 (Sections 3.3 and 3.4, respectively), and present ablation studies to illustrate the properties of our proposed approach (Section 3.5).

3.1 Environmental Setup

All experiments are conducted using the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018) and Tiny Llama with 1.1B parameters (Zhang et al., 2024), due to its suitability to on-device applications.

Data: The labels are not publicly available for the test sets of the GLUE benchmark. Therefore, following Chen et al. (2022), we use the development set of each task as the test set, and split the training set into training and development sets with a 9:1

ratio (see Appendix C for a brief description of the tasks and the number of samples for each split).

Methods: We compare PG with LoRA under various setups which are denoted by subscripts: A_F indicates that the parameter A is fixed during fine-tuning. Learning the parameter A using PG is denoted by A_H (B_H for learning B using PG). Similarly, A_L refers to training the parameter A using LoRA (B_L for training B using LoRA).

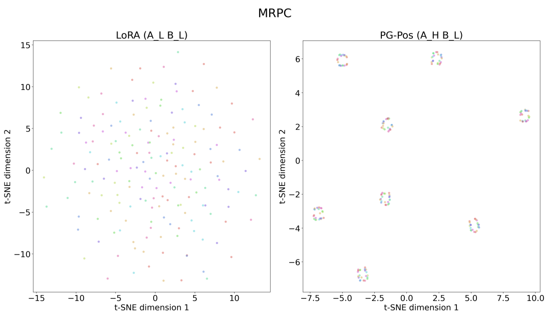
Metrics: We report Matthews correlation coefficient for CoLA, Pearson correlation coefficient for STS-B, and accuracy for the remaining tasks.

Implementation Details: As proposed by Hu et al. (2022), we apply the PEFT parameters to query and value projection matrices in self-attention modules of the LLM, utilizing the AdamW optimizer in the training phase. As proposed by Ding et al. (2023), we fine-tune and evaluate the models 5 times with randomly selected seeds, and report the average of the results. The hypernetwork architecture is composed of two fully connected linear layers with bias terms and one non-linear activation function (GeLU) used between these linear layers. This architecture provides us a lightweight computationally efficient model. Please see Appendix D for the additional details.

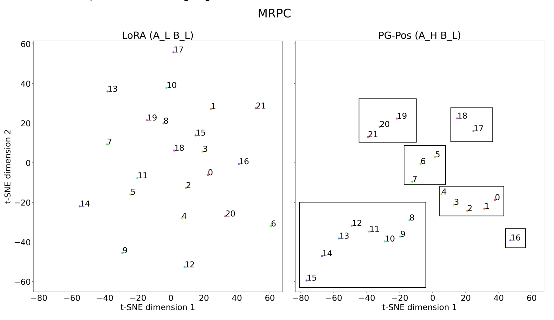
3.2 Motivating Examples

Goal: This example tests our assumption on the benefits of relying on low-dimensional parameter manifolds for learning PEFT parameters. We test this hypothesis by comparing LoRA and three PG variants using the MRPC dataset with a rank of 8 and perform two analyses on two sample sets: (i) each sample $s_i \in \mathcal{S}$ corresponds to the c^{th} column vector $A^l_c \in \mathbb{R}^D, \forall c \in [R]$, of the A^l matrix at the l^{th} layer, $\forall l \in [L]$ (to analyze the similarities of projections), (ii) each sample s_i corresponds to the $A^l \in \mathbb{R}^{DR}$, $\forall l \in [L]$ with the rank $R = 8$, $D = 2048$ and $L = 22$ (to analyze the relationships across the layers). The samples are visualized on two-dimensional manifolds using t-SNE in Figure 4 for LoRA and PG-Pos. The results for different configurations are given in Appendix F.

Results: Figure 4 (a) shows that the vectors A^l_c obtained from LoRA are scattered across the lower-dimensional manifold. Conversely, PG-Pos provides a clustering of these vectors. A closer inspection of the clusters reveal that they are homogeneous in that each cluster contains one vector per layer. Therefore, the c^{th} cluster C_c corresponds to c^{th} component of the A parameters, i.e., $C_c = \{A^l_c\}^L_{l=1}$. This confirms that the A parameters share similar characteristics across the layers. Figure 4 (b) indicates that LoRA produces randomly distributed layers, whereas PG-Pos captures the sequential order among the layers. We also observe that some layers can be grouped together, which reflect their similarities within the group and dissimilarities across the other groups.



(a) Analysis of $\mathcal{S} = \{s_i\}^{RL}_{i=1}$ where each sample s_i corresponds to the c^{th} column vector $A^l_c \in \mathbb{R}^D, \forall c \in [R]$ at the l^{th} layer, $\forall l \in [L]$.



(b) Analysis for $\mathcal{S} = \{s_i\}^L_{i=1}$ where each sample s_i corresponds to the $A^l \in \mathbb{R}^{DR}$ at the l^{th} layer, $\forall l \in [L]$.

Figure 4: **A visualization of the A parameters provided by LoRA and PG-Pos on a two-dimensional manifold.** We obtain the parameters associated with the query matrices using the MRPC dataset with the rank $R = 8$, $D = 2048$ and $L = 22$. The parameters are then mapped to a two-dimensional manifold using t-SNE. a) shows that the A parameters share similarities across the layers. b) shows that our method learns the sequential order of the layers and their groupings.

3.3 Comparison with fixed A identified by a random matrix

Goal: The goal of these experiments is to analyze the impact of information-sharing among the A parameters (denoted by $A_H B_L$), as opposed to no explicit information-sharing while keeping A frozen (denoted by $A_F B_L$). Note that we evaluate using hypernetworks with two different sizes for $A_H B_L$ to make a fair comparison. Additionally, we report the impact of fixing A when learning B using the PG (denoted by $A_F B_H$) for completeness.

Method	Config	Rank	h_2	# Trainable Params	GLUE MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
LoRA	$A_F B_L$		-	405,504	83.00$_{\pm 0.3}$	93.66$_{\pm 0.6}$	72.12$_{\pm 1.2}$	42.54$_{\pm 3.3}$	88.50$_{\pm 0.5}$	86.72$_{\pm 0.1}$	58.75$_{\pm 3.8}$	72.04$_{\pm 2.5}$	74.7
PG-Fixed				387,492	79.09$_{\pm 0.4}$	92.57$_{\pm 0.4}$	69.67$_{\pm 2.2}$	26.65$_{\pm 8.2}$	85.08$_{\pm 0.7}$	85.03$_{\pm 0.2}$	56.95$_{\pm 2.8}$	51.04$_{\pm 7.7}$	68.2
PG-Trainable	$A_F B_H$		20	387,932	80.49$_{\pm 0.3}$	92.92$_{\pm 0.5}$	69.79$_{\pm 2.3}$	27.29$_{\pm 7.9}$	85.86$_{\pm 0.8}$	85.54$_{\pm 0.2}$	57.22$_{\pm 2.7}$	52.63$_{\pm 7.6}$	69.0
PG-Pos				388,332	82.00$_{\pm 0.3}$	93.06$_{\pm 0.7}$	72.55$_{\pm 3.5}$	42.22$_{\pm 3.7}$	87.31$_{\pm 1.0}$	86.47$_{\pm 0.1}$	58.21$_{\pm 2.7}$	73.19$_{\pm 2.4}$	74.4
PG-Fixed		8		471,061	86.11$_{\pm 0.2}$	**94.63$_{\pm 0.3}$**	77.55$_{\pm 0.9}$	55.25$_{\pm 1.1}$	91.06$_{\pm 0.3}$	88.32$_{\pm 0.1}$	74.30$_{\pm 2.2}$	84.78$_{\pm 1.3}$	81.5
PG-Trainable	$A_H B_L$		1	471,501	86.13$_{\pm 0.3}$	94.61$_{\pm 0.3}$	77.60$_{\pm 1.0}$	55.26$_{\pm 1.1}$	91.06$_{\pm 0.3}$	88.33$_{\pm 0.1}$	**74.30$_{\pm 2.0}$**	84.80$_{\pm 1.3}$	81.5
PG-Pos				471,521	86.32$_{\pm 0.1}$	94.22$_{\pm 0.5}$	77.79$_{\pm 0.9}$	55.36$_{\pm 1.2}$	91.06$_{\pm 0.4}$	88.49$_{\pm 0.1}$	74.15$_{\pm 1.3}$	85.32$_{\pm 1.2}$	81.6
PG-Fixed				1,094,052	86.72$_{\pm 0.1}$	94.54$_{\pm 0.3}$	77.40$_{\pm 1.1}$	55.50$_{\pm 2.1}$	91.26$_{\pm 0.4}$	88.53$_{\pm 0.1}$	62.64$_{\pm 2.7}$	85.42$_{\pm 1.2}$	80.2
PG-Trainable	$A_H B_L$		20	1,094,492	86.76$_{\pm 0.1}$	94.47$_{\pm 0.2}$	77.45$_{\pm 2.3}$	55.71$_{\pm 2.2}$	91.25$_{\pm 0.4}$	88.54$_{\pm 0.1}$	62.82$_{\pm 2.6}$	85.55$_{\pm 1.2}$	80.3
PG-Pos				1,077,720	**87.38$_{\pm 0.2}$**	94.54$_{\pm 0.3}$	**80.39$_{\pm 0.7}$**	**56.62$_{\pm 1.8}$**	**91.29$_{\pm 0.3}$**	**89.01$_{\pm 0.0}$**	70.85$_{\pm 1.7}$	**87.43$_{\pm 0.7}$**	**82.2**
LoRA	$A_F B_L$		-	811,008	83.09$_{\pm 0.4}$	94.04$_{\pm 0.3}$	73.35$_{\pm 1.7}$	36.63$_{\pm 6.8}$	87.80$_{\pm 0.9}$	86.74$_{\pm 0.1}$	58.48$_{\pm 3.6}$	72.15$_{\pm 3.8}$	74.0
PG-Fixed				774,564	80.25$_{\pm 0.3}$	93.32$_{\pm 0.4}$	70.59$_{\pm 1.5}$	19.19$_{\pm 6.1}$	86.15$_{\pm 1.2}$	85.46$_{\pm 0.1}$	53.25$_{\pm 2.3}$	54.57$_{\pm 10.2}$	67.9
PG-Trainable	$A_F B_H$		20	775,404	81.95$_{\pm 0.3}$	93.61$_{\pm 0.3}$	70.65$_{\pm 1.4}$	20.01$_{\pm 5.6}$	86.99$_{\pm 1.1}$	86.03$_{\pm 0.1}$	53.34$_{\pm 2.6}$	56.44$_{\pm 9.5}$	68.6
PG-Pos				775,404	83.11$_{\pm 0.3}$	93.29$_{\pm 0.4}$	72.61$_{\pm 1.4}$	35.96$_{\pm 6.6}$	88.44$_{\pm 0.8}$	86.99$_{\pm 0.0}$	55.32$_{\pm 2.4}$	75.76$_{\pm 3.4}$	73.9
PG-Fixed		16		942,101	86.10$_{\pm 0.2}$	94.43$_{\pm 0.4}$	76.91$_{\pm 1.1}$	54.50$_{\pm 2.8}$	90.96$_{\pm 0.4}$	88.35$_{\pm 0.1}$	73.21$_{\pm 2.6}$	84.67$_{\pm 1.3}$	81.1
PG-Trainable	$A_H B_L$		1	942,541	86.14$_{\pm 0.2}$	94.43$_{\pm 0.3}$	76.86$_{\pm 1.1}$	54.31$_{\pm 2.7}$	90.99$_{\pm 0.4}$	88.35$_{\pm 0.1}$	73.21$_{\pm 2.6}$	84.69$_{\pm 1.3}$	81.1
PG-Pos				942,561	86.48$_{\pm 0.2}$	94.29$_{\pm 0.5}$	77.99$_{\pm 0.7}$	55.38$_{\pm 1.4}$	91.07$_{\pm 0.3}$	88.52$_{\pm 0.1}$	**73.57$_{\pm 2.4}$**	85.14$_{\pm 1.2}$	81.6
PG-Fixed				2,187,684	86.80$_{\pm 0.1}$	94.61$_{\pm 0.3}$	77.65$_{\pm 0.9}$	55.80$_{\pm 2.0}$	91.29$_{\pm 0.3}$	88.54$_{\pm 0.1}$	63.63$_{\pm 2.9}$	85.52$_{\pm 1.2}$	80.5
PG-Trainable	$A_H B_L$		20	2,188,124	86.84$_{\pm 0.2}$	**94.68$_{\pm 0.2}$**	78.09$_{\pm 0.9}$	56.11$_{\pm 1.8}$	**91.34$_{\pm 0.4}$**	88.56$_{\pm 0.1}$	63.54$_{\pm 3.0}$	85.66$_{\pm 1.1}$	80.6
PG-Pos				2,188,524	**87.54$_{\pm 0.2}$**	94.40$_{\pm 0.2}$	**80.00$_{\pm 0.4}$**	**56.88$_{\pm 1.4}$**	91.26$_{\pm 0.4}$	**89.06$_{\pm 0.0}$**	70.58$_{\pm 1.4}$	**87.55$_{\pm 0.6}$**	**82.2**

Table 2: **Performance of the methods when the A parameter is kept frozen during fine-tuning**. The experiments are based on all the GLUE datasets over 5 randomly selected seeds. We report Matthews correlation coefficient for CoLA, Pearson correlation coefficient for STS-B, and accuracy for the remaining tasks, with the standard deviations given in the subscript. The best performance per rank is highlighted by bold.

Results: In Table 2, we obtain the best performance using PG-Pos under the $A_H B_L$ setting, which consistently improves over the performance of $A_F B_L$ by a large margin. These results reflect the benefit of learning A parameters with information-sharing. Conversely, fixing A leads to similar results for PG and LoRA (with $A_F B_L$ performing better than $A_F B_H$ on two (for $R = 8$) and three tasks (for $R = 16$). With these findings, we form the following answer to $Q1$:

A1: Learning PEFT parameters can help improving the performance of LoRA when A is characterized as a random projection for both trained and learned B.

3.4 Comparison with trained A

Goal: We first compare our approach with LoRA when A is trained, leading to two settings which are: (i) $A_L B_L$ where both parameters are learned using LoRA and (ii) $A_H B_L$ where A_H is learned using PG while B_L is learned using LoRA. Second, we explore the shareability of the layers by using different number of hypernetworks for PG-Pos. In addition to using 1 hypernetwork for all layers, we create (i) 1 hypernetwork per layer and (ii) 1 hypernetwork per 11 layers (in this case, we group lower layers and upper layers). Additionally, we compare our method with two other PEFT baselines, namely HiRA (Authors, 2024) and ReLoRA (Lialin et al., 2023b).

Results: Table 3 shows that PG-Pos (1 hypernetwork in total) consistently improves over the performance of LoRA across all the tasks, except for QNLI when the rank is 8 and for CoLA when the rank is 16. Moreover, LoRA achieves better results than the best performing PG variant only by 0.04% on QNLI and by 0.14% on CoLA. The performance gap is larger for MRPC, CoLA, RTE, and STS-B than it is for the remaining datasets for $R = 8$. Similarly, the RTE and STS-B tasks lead to a larger gap than the other tasks when the rank is 16. For the remaining tasks, the gap is usually in the favor of PG-Pos. These differences can also be seen in the convergence plots (e.g., see Figure 5 where PG-Pos converges faster than LoRA on the MRPC dataset, achieving higher final accuracy. The results for additional datasets are provided in Appendix F).

The results also show that although the loss converges to a lower value for LoRA for $R = 16$, PG provides higher accuracy. Moreover, the accuracy for the PG generated parameters continues increasing for larger number of epochs. This observation suggests that the LoRA may cause over-fitting where PG can resolve this over-fitting problem.

Our method outperforms HiRA and ReLoRA on the GLUE benchmark. In particular, even when the parameter size is doubled for HiRa, our method still achieves performance comparable to that of the PEFT baselines.

We obtain comparable results under different number of hypernetworks. PG-Pos with 1 hypernetwork per layer achieves the best performance, which may partially be due to the increased

Method	Config	Rank	# Layers per Hypernet	# Trainable Params	GLUE MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
LoRA	$A_L B_L$	8	-	1,126,400	$87.09_{\pm 0.2}$	$94.52_{\pm 0.5}$	$79.66_{\pm 1.3}$	$55.68_{\pm 3.1}$	$91.33_{\pm 0.1}$	$88.72_{\pm 0.1}$	$70.40_{\pm 2.8}$	$85.45_{\pm 0.8}$	81.6
ReLoRA	$sA_L B_L$		-	1,126,401	$86.23_{\pm 0.1}$	$93.90_{\pm 0.3}$	$78.82_{\pm 1.1}$	$57.41_{\pm 1.8}$	$91.31_{\pm 0.2}$	$88.41_{\pm 0.1}$	$65.13_{\pm 4.3}$	$84.46_{\pm 1.0}$	80.7
HiRA	$C_L D_L W + A_L B_L$		-	2,252,800	$86.69_{\pm 0.1}$	$94.31_{\pm 0.4}$	$80.24_{\pm 0.9}$	$57.24_{\pm 0.8}$	$91.40_{\pm 0.1}$	$88.85_{\pm 0.1}$	$70.54_{\pm 2.7}$	$86.35_{\pm 0.8}$	81.9
PG-Pos	$A_H B_L$		22	1,094,892	$\mathbf{87.38}_{\pm \mathbf{0.2}}$	$\mathbf{94.54}_{\pm \mathbf{0.3}}$	$80.39_{\pm 0.7}$	$56.62_{\pm 1.8}$	$91.29_{\pm 0.3}$	$89.01_{\pm 0.0}$	$70.85_{\pm 1.7}$	$87.43_{\pm 0.7}$	82.2
PG-Pos	$A_H B_L$		11	1,784,280	$86.94_{\pm 0.3}$	$93.90_{\pm 0.2}$	$81.13_{\pm 1.1}$	$56.92_{\pm 1.3}$	$91.13_{\pm 0.1}$	$89.13_{\pm 0.1}$	$69.17_{\pm 3.0}$	$87.41_{\pm 0.7}$	82.0
PG-Pos	$A_H B_L$		1	15,572,040	$87.15_{\pm 0.2}$	$94.15_{\pm 0.4}$	$\mathbf{82.50}_{\pm \mathbf{1.7}}$	$\mathbf{57.46}_{\pm \mathbf{2.8}}$	$\mathbf{91.47}_{\pm \mathbf{0.3}}$	$\mathbf{89.40}_{\pm \mathbf{0.1}}$	$\mathbf{73.07}_{\pm \mathbf{1.5}}$	$\mathbf{87.68}_{\pm \mathbf{0.2}}$	**82.9**
LoRA	$A_L B_L$	16	-	2,252,800	$86.91_{\pm 0.2}$	$94.24_{\pm 0.3}$	$79.75_{\pm 1.8}$	$57.02_{\pm 1.7}$	$91.11_{\pm 0.2}$	$88.70_{\pm 0.1}$	$68.32_{\pm 1.8}$	$85.81_{\pm 1.6}$	81.5
ReLoRA	$sA_L B_L$		-	2,252,801	$87.00_{\pm 0.1}$	$94.31_{\pm 0.1}$	$79.95_{\pm 1.9}$	$58.05_{\pm 1.1}$	$91.35_{\pm 0.2}$	$88.93_{\pm 0.1}$	$71.91_{\pm 1.3}$	$86.46_{\pm 0.2}$	82.2
HiRA	$C_L D_L W + A_L B_L$		-	4,505,600	$87.03_{\pm 0.1}$	$94.35_{\pm 0.2}$	$80.00_{\pm 1.7}$	$58.08_{\pm 1.3}$	$91.37_{\pm 0.1}$	$88.96_{\pm 0.1}$	$72.06_{\pm 1.9}$	$86.38_{\pm 1.0}$	82.2
PG-Pos	$A_H B_L$		22	2,188,524	$\mathbf{87.54}_{\pm \mathbf{0.2}}$	$\mathbf{94.40}_{\pm \mathbf{0.2}}$	$80.00_{\pm 0.4}$	$56.88_{\pm 1.4}$	$91.26_{\pm 0.4}$	$89.06_{\pm 0.0}$	$70.58_{\pm 1.4}$	$87.55_{\pm 0.6}$	82.2
PG-Pos	$A_H B_L$		11	3,566,040	$86.93_{\pm 0.3}$	$94.04_{\pm 0.4}$	$80.98_{\pm 1.2}$	$56.47_{\pm 1.7}$	$91.18_{\pm 0.2}$	$89.19_{\pm 0.1}$	$69.68_{\pm 3.5}$	$87.41_{\pm 0.8}$	82.0
PG-Pos	$A_H B_L$		1	31,116,360	$87.20_{\pm 0.2}$	$94.38_{\pm 0.3}$	$\mathbf{83.04}_{\pm \mathbf{1.3}}$	$\mathbf{58.23}_{\pm \mathbf{2.0}}$	$\mathbf{91.47}_{\pm \mathbf{0.2}}$	$\mathbf{89.45}_{\pm \mathbf{0.1}}$	$\mathbf{72.56}_{\pm \mathbf{1.2}}$	$\mathbf{87.62}_{\pm \mathbf{0.2}}$	**83.0**

Table 3: **Evaluation of the methods under different groupings of the layers** We run the experiments on all the GLUE datasets and average the results over 5 randomly selected seeds. We report Matthews correlation coefficient for CoLA, Pearson correlation coefficient for STS-B, and accuracy for the remaining tasks, with the standard deviations given in the subscript. The best performance per rank is highlighted by bold.

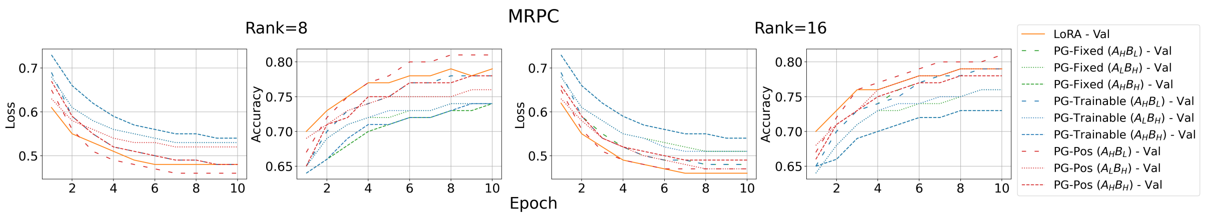


Figure 5: **Convergence of the methods on the validation split of the MRPC dataset.** The figures display the loss and the corresponding performance metric at each epoch and the ranks of 8 and 16.

number of trainable parameters. Based on these findings, we form the following answer to Q2:

A2: Learning the projection matrices A with information shared across the layers of an LLM improves the fine-tuning performance. The performance gain can be relatively smaller when one hypernetwork is utilized per layer depending on the task.

3.5 Ablation Studies

We carry out ablation studies to: (i) show the impact of combining LoRA and the PG methods using different configurations, (ii) illustrate the impact of hypernetwork size on the performance and (iii) investigate whether the benefits of PG over LoRA generalize to different LLM architectures and sizes.

Hypernetwork configurations: We design two additional configurations to characterize the impact of information-sharing across the B parameters (denoted by $A_L B_H$) and both the A and B parameters (denoted by $A_H B_H$),

Table 4 displays a declining trend in the performance in the order of $A_H B_L$, $A_L B_H$ and $A_H B_H$. Information-sharing is more effective on the A parameters than it is on the B parameters, especially for the MRPC, CoLA, RTE and STS-B tasks. We conjecture that projecting the inputs onto a lower-dimensional manifold makes it difficult to incorporate the layer hierarchy into the B parameters through information sharing. Moreover, $A_H B_H$ leads to the worst performance which might be due to the difficulty of jointly learning the A and B parameters using the same hypernetwork. To address this issue, one can use separate hypernetworks for these parameters; however, we avoid doing so, not to optimize on the test sets.

Lastly, we observe a trend in that the best performing PG variant is PG-Pos, which is respectively followed by PG-Trainable and PG-Fixed. These results are perhaps not surprising as PG-Pos is utilizing the positional embeddings in addition to the layer embeddings.

Hypernetwork size: We conduct experiments under varying hypernetwork sizes using the MRPC, CoLA, RTE and STS-B datasets. Given that the hypernetwork size largely depends on the dimension h_2, we sweep across $h_2 \in \{1, 5, 10, 20\}$ while keeping $h_1 = 20$. The results given in Figure 6 indicate that using a higher number of trainable parameters often improves the performance, with exceptions for the RTE dataset which may be explained by overfitting. Please see Appendix E for results on all of the datasets and ranks.

Generalization of the results: Here, we investigate whether the results generalize to other LLM architectures and sizes. Specifically, we compare LoRA and PG using Pythia-1B, Llama2-7B, Pythia-

Method	Config	Rank	# Trainable Params	GLUE								
				MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
PG-Fixed	$A_L B_H$	8	1,108,388	$85.72_{\pm0.2}$	$93.81_{\pm0.3}$	$75.54_{\pm1.9}$	$50.64_{\pm2.4}$	$90.05_{\pm0.2}$	$88.10_{\pm0.1}$	$61.91_{\pm2.9}$	$81.44_{\pm1.3}$	78.4
	$A_H B_L$		1,094,052	$86.72_{\pm0.1}$	**$94.54_{\pm0.3}$**	$77.40_{\pm1.1}$	$55.50_{\pm2.1}$	$91.26_{\pm0.4}$	$88.53_{\pm0.1}$	$62.64_{\pm2.7}$	$85.42_{\pm1.2}$	80.2
	$A_H B_H$		1,076,040	$84.69_{\pm0.2}$	$93.12_{\pm0.5}$	$72.55_{\pm2.6}$	$45.50_{\pm5.0}$	$88.47_{\pm0.3}$	$87.08_{\pm0.2}$	$52.98_{\pm1.8}$	$77.69_{\pm2.2}$	75.3
PG-Trainable	$A_L B_H$		1,108,828	$85.79_{\pm0.2}$	$93.90_{\pm0.3}$	$75.59_{\pm1.8}$	$51.10_{\pm2.1}$	$90.13_{\pm0.2}$	$88.12_{\pm0.1}$	$61.91_{\pm2.9}$	$80.86_{\pm1.5}$	78.4
	$A_H B_L$		1,094,492	$86.76_{\pm0.1}$	$94.47_{\pm0.2}$	$77.45_{\pm1.1}$	$55.71_{\pm2.2}$	$91.25_{\pm0.4}$	$88.54_{\pm0.1}$	$62.82_{\pm2.6}$	$85.55_{\pm1.2}$	80.3
	$A_H B_H$		1,076,920	$84.97_{\pm0.1}$	$93.35_{\pm0.4}$	$72.70_{\pm2.7}$	$46.40_{\pm4.9}$	$88.81_{\pm0.2}$	$87.24_{\pm0.2}$	$53.16_{\pm1.6}$	$78.22_{\pm2.2}$	75.6
PG-Pos	$A_L B_H$		1,109,228	$86.35_{\pm0.2}$	$94.11_{\pm0.6}$	$77.70_{\pm1.7}$	$53.55_{\pm0.6}$	$90.30_{\pm0.5}$	$88.58_{\pm0.1}$	$65.07_{\pm4.3}$	$84.75_{\pm0.4}$	80.0
	$A_H B_L$		1,094,892	**$87.38_{\pm0.2}$**	**$94.54_{\pm0.3}$**	**$80.39_{\pm0.7}$**	**$56.62_{\pm1.8}$**	**$91.29_{\pm0.3}$**	**$89.01_{\pm0.0}$**	**$70.85_{\pm1.7}$**	**$87.43_{\pm0.7}$**	**82.2**
	$A_H B_H$		1,077,720	$86.33_{\pm0.2}$	$93.90_{\pm0.4}$	$77.01_{\pm2.2}$	$53.45_{\pm1.8}$	$90.49_{\pm0.2}$	$88.39_{\pm0.1}$	$65.34_{\pm3.0}$	$85.52_{\pm1.1}$	80.0
PG-Fixed	$A_L B_H$	16	2,216,356	$86.22_{\pm0.2}$	$94.11_{\pm0.5}$	$74.71_{\pm1.2}$	$51.30_{\pm2.0}$	$90.44_{\pm0.2}$	$88.26_{\pm0.1}$	$58.21_{\pm2.1}$	$82.98_{\pm0.9}$	78.3
	$A_H B_L$		2,187,684	$86.80_{\pm0.1}$	$94.61_{\pm0.3}$	$77.65_{\pm0.9}$	$55.80_{\pm2.0}$	$91.29_{\pm0.3}$	$88.54_{\pm0.1}$	$63.63_{\pm2.9}$	$85.52_{\pm1.2}$	80.5
	$A_H B_H$		2,151,240	$84.85_{\pm0.1}$	$93.30_{\pm0.3}$	$72.06_{\pm3.1}$	$46.42_{\pm4.5}$	$88.83_{\pm0.3}$	$87.20_{\pm0.1}$	$53.25_{\pm0.7}$	$78.17_{\pm2.4}$	75.5
PG-Trainable	$A_L B_H$		2,216,796	$86.33_{\pm0.2}$	$94.08_{\pm0.5}$	$74.80_{\pm1.3}$	$51.66_{\pm2.1}$	$90.49_{\pm0.1}$	$88.31_{\pm0.1}$	$58.12_{\pm2.3}$	$83.13_{\pm0.9}$	78.4
	$A_H B_L$		2,188,124	$86.84_{\pm0.2}$	**$94.68_{\pm0.2}$**	$78.09_{\pm0.6}$	$56.11_{\pm1.8}$	**$91.34_{\pm0.4}$**	$88.56_{\pm0.1}$	$63.54_{\pm3.0}$	$85.66_{\pm1.1}$	80.6
	$A_H B_H$		2,152,120	$85.14_{\pm0.1}$	$93.42_{\pm0.4}$	$72.11_{\pm3.1}$	$47.04_{\pm4.8}$	$89.02_{\pm0.2}$	$87.32_{\pm0.0}$	$53.16_{\pm0.5}$	$78.86_{\pm2.3}$	75.8
PG-Pos	$A_L B_H$		2,217,196	$86.90_{\pm0.2}$	$94.11_{\pm0.1}$	$78.43_{\pm0.5}$	$54.08_{\pm2.3}$	$90.88_{\pm0.1}$	$88.95_{\pm0.1}$	$64.08_{\pm2.7}$	$85.60_{\pm0.6}$	80.4
	$A_H B_L$		2,188,524	**$87.54_{\pm0.2}$**	$94.40_{\pm0.2}$	**$80.00_{\pm0.4}$**	**$56.88_{\pm1.4}$**	$91.26_{\pm0.4}$	**$89.06_{\pm0.0}$**	**$70.58_{\pm1.4}$**	**$87.55_{\pm0.6}$**	**82.2**
	$A_H B_H$		2,152,920	$86.77_{\pm0.1}$	$94.06_{\pm0.5}$	$77.70_{\pm1.3}$	$54.30_{\pm1.4}$	$90.84_{\pm0.2}$	$88.59_{\pm0.1}$	$64.26_{\pm2.6}$	$85.68_{\pm0.9}$	80.3

Table 4: **Evaluation of the PG variants in terms of embeddings** We run the methods on GLUE benchmark and average the results over 5 randomly selected seeds. We report Matthews correlation coefficient for CoLA, Pearson correlation coefficient for STS-B, and accuracy for the remaining tasks, with the standard deviations given in the subscript. The best performance per rank is highlighted by bold.

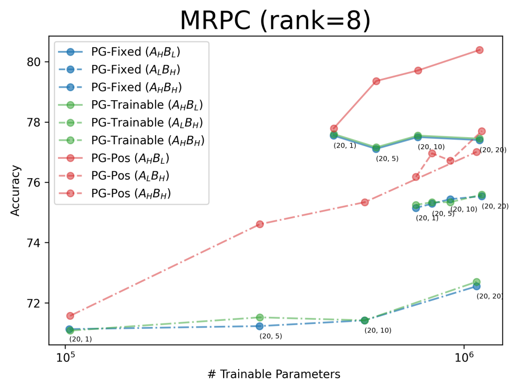


Figure 6: **Accuracy of the methods with respect to the number of trainable parameters** for the MRPC.

6.9B, Pythia-12B, Llama2-13B, and Gemma2-27B. The results are presented in Appendix E. We observe that PG-Pos improves over the performance of LoRA in all settings, except for the MRPC task using Pythia-1B and Pythia-12B for both ranks and Gemma-2-27B for rank 8.

4 Conclusion

Parameter-efficient fine-tuning (PEFT) methods play a critical role in adapting large language models (LLMs) to new tasks. Existing PEFT methods typically employ direct optimization on the PEFT parameters during fine-tuning. In this paper, we propose a new approach called Learning to Efficiently Fine-tune (LEFT) where the focus is on modelling the generative process of these parameters. To this end, we develop the Parameter Generation (PG) method where we utilize hypernetworks conditioned on the properties of a pre-trained LLM. Our experimental analyses on the GLUE benchmark confirm the effectiveness of PG. Future work includes its application to text-generation tasks and its extension to data-dependent model adaptation.

5 Potential Risks

Since our work is based on large-language models (LLMs), it has two main potential risks: environmental risk and fairness considerations. Although our work aims for parameter-efficient model fine-tuning, training foundational LMs consumes huge computational resources and electricity that can be harmful for the environment. Additionally, fairness for different concepts is not adequately considered in our experiments.

6 Limitations

Despite the encouraging results demonstrated by LEFT, there are certain limitations in our current study that are worth acknowledging. This paper only evaluates the effectiveness of PG on traditional natural language processing tasks. However, recent studies demonstrate that parameter-efficient methods could be applied to cross-modal or instruction-tuning scenarios. We believe that our proposed LEFT and PG methods can be used in more general cross-modal and multi-modal tasks as well. In addition, we used hypernetworks for implementing the PG methods. Exploring other generative models, such as Bayesian Neural Networks and GANs would be a valuable future direction.

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 (Volume 1: Long Papers)*, pages 7319–7328. Association for Computational Linguistics.

Anonymous Authors. 2024. HiRA: Parameter-efficient Hadamard high-rank adaptation for large language models. In *ACL ARR 2024 June Submission*. `https://openreview.net/forum?id=UHAFkcfNdL` (available on 30 July 2024).

Sara Babakniya, Ahmed Elkordy, Yahya Ezzeldin, Qingfeng Liu, Kee-Bong Song, Mostafa EL-Khamy, and Salman Avestimehr. 2023. SLoRA: Federated parameter efficient fine-tuning of language models. In *International Workshop on Federated Learning in the Age of Foundation Models in Conjunction with NeurIPS 2023*.

Guanzheng Chen, Fangyu Liu, Zaiqiao Meng, and Shangsong Liang. 2022. Revisiting parameter-efficient tuning: Are we really there yet? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2612–2626. Association for Computational Linguistics.

Lior Deutsch, Erik Nijkamp, and Yu Yang. 2019. A generative model for sampling high-performance and diverse weights for neural networks. *Preprint*, arXiv:1905.02898.

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. 2023. Sparse low-rank adaptation of pre-trained language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4133–4145. Association for Computational Linguistics.

Nadir Durrani, Fahim Dalvi, and Hassan Sajjad. 2023. Discovering Salient Neurons in deep NLP models. *Journal of Machine Learning Research*, 24(362):1–40.

Siqi Fan, Xin Jiang, Xiang Li, Xuying Meng, Peng Han, Shuo Shang, Aixin Sun, Yequan Wang, and Zhongyuan Wang. 2024. Not all layers of LLMs are necessary during inference. *arXiv preprint arXiv:2403.02181*.

Han Guo, Philip Greengard, Eric Xing, and Yoon Kim. 2024. LQ-loRA: Low-rank plus quantized matrix decomposition for efficient language model finetuning. In *The Twelfth International Conference on Learning Representations*.

David Ha, Andrew M. Dai, and Quoc V. Le. 2017. Hypernetworks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Yongchang Hao, Yanshuai Cao, and Lili Mou. 2024. FLORA: Low-rank adapters are secretly gradient compressors. In *Proceedings of the 41st International Conference on Machine Learning*.

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 *International Conference on Learning Representations*.

Sylwester Klocek, Łukasz Maziarka, Maciej Wołczyk, Jacek Tabor, Jakub Nowak, and Marek Śmieja. 2019. Hypernetwork functional image representation. In *International Conference on Artificial Neural Networks*, pages 496–510. Springer.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. 2018. Measuring the intrinsic dimension of objective landscapes. In *International Conference on Learning Representations*.

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

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. 2023b. Relora: High-rank training through low-rank updates. In *The Twelfth International Conference on Learning Representations*.

Ritam Majumdar, Vishal Sudam Jadhav, Anirudh Deodhar, Shirish Karande, Lovekesh Vig, and Venkataramana Runkana. 2023. PIHLoRA: Physics-informed hypernetworks for low-ranked adaptation. In *AI for Accelerated Materials Design-NeurIPS 2023 Workshop*.

Mete Ozay. 2019. Fine-grained optimization of deep neural networks. In *Advances in Neural Information Processing Systems*, volume 32.

Jason Phang, Yi Mao, Pengcheng He, and Weizhu Chen. 2023. HyperTuning: Toward adapting large language models without back-propagation. In *Proceedings of the 40th International Conference on Machine Learning*, pages 27854–27875. PMLR.

Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Wei Wei, Tingbo Hou, Yael Pritch, Neal Wadhwa, Michael Rubinstein, and Kfir Aberman. 2024. HyperDreamBooth: HyperNetworks for fast personalization of text-to-image models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6527–6536.

Aviv Shamsian, Aviv Navon, Ethan Fetaya, and Gal Chechik. 2021. Personalized federated learning using hypernetworks. In *International Conference on Machine Learning*, pages 9489–9502. PMLR.

Joseph Suarez. 2017. Language modeling with recurrent highway hypernetworks. In *Advances in Neural Information Processing Systems*, volume 30.

Youbang Sun, Zitao Li, Yaliang Li, and Bolin Ding. 2024. Improving LoRA in privacy-preserving federated learning. In *International Conference on Learning Representations*.

Yi Tay, Zhe Zhao, Dara Bahri, Donald Metzler, and Da-Cheng Juan. 2021. HyperGrid Transformers: Towards a single model for multiple tasks. In *International Conference on Learning Representations*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, volume 30.

Johannes von Oswald, Christian Henning, Benjamin F. Grewe, and João Sacramento. 2020. Continual learning with hypernetworks. In *International Conference on Learning Representations*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355. Association for Computational Linguistics.

Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. 2023. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *arXiv preprint arXiv:2312.12148*.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model. *Preprint*, arXiv:2401.02385.

Jiacheng Zhu, Kristjan Greenewald, Kimia Nadjahi, Haitz Sáez de Ocáriz Borde, Rickard Brüel Gabrielsson, Leshem Choshen, Marzyeh Ghassemi, Mikhail Yurochkin, and Justin Solomon. 2024. Asymmetry in low-rank adapters of foundation models. *arXiv preprint arXiv:2402.16842*.

A Appendix

B Background

This section provides a brief description of LoRA.

Parameter efficient fine-tuning methods inject a relatively small number of trainable parameters into LLMs while keeping the original parameters frozen. In particular, Low-Rank Adapdation (LoRA, Hu et al. 2022) introduces two low-dimensional trainable parameters, whose multiplication approximates the weight updates during fine-tuning.

Denoting the weights of a pre-trained LLM by $W_0 \in \mathbb{R}^{D\times K}$, the forward pass of the vanilla LoRA yields the output y as follows:

$$y = W_0 x + \Delta W x = W_0 x + BAx, \qquad (3)$$

where $B \in \mathbb{R}^{D\times R}$ and $A \in \mathbb{R}^{R\times K}$. In the vanilla LoRA methods, these parameters are identified by the trainable LoRA matrices. More recent works suggest defining A by random matrices as discussed in the Introduction section. By choosing a low value for R, LoRA reduces the number of trainable parameters from $D \times K$ to $R \times (D + K)$.

We can apply LoRA to any subset of layers in a deep neural network. For transformers-based LLMs, LoRA can be applied to the self-attention and the multilayer perceptron (MLP) modules. However, Hu et al. (2022) show that applying LoRA to the query and value projection parameters in the self-attention module gives the best performance overall.

C Additional Information about the Datasets

Table 5 presents a summary of the datasets in the GLUE benchmark. Note that the MNLI dataset is shared with the ONAC license, whereas the QNLI dataset is licensed with CC-BY-SA 4. No license specified for the remaining tasks.

D Implementation Details

Our experimental setup partly follows the evaluation setup used by Ding et al. (2023). For instance, we fine-tune and evaluate the methods 5 times with randomly selected seeds, and report the average of the results. We use the learning rate and the number of training epochs reported in Table 6. Note that the remaining hyper-parameters are the default parameters of the huggingface library used for AdamW optimizer. Lastly, we also use standard initialization across models for the MRPC, RTE and STS-B

Corpus	Task	# Train	# Dev	# Test	# Labels	Evaluation Metrics
			Single Sentence Tasks			
CoLA	Grammatical Acceptability	7,695	856	1,043	2	Matthews corr. coef.
SST-2	Sentiment	60,614	6,735	872	2	Accuracy
			Pairwise Sentences Tasks			
MNLI	Entailment	353,431	39,271	19,647	3	Accuracy
MRPC	Semantic Equivalence	3,301	367	408	2	Accuracy
QNLI	Question Answering	94,268	10,475	5,463	2	Accuracy
QQP	Semantic Equivalence	327,461	36,385	40,430	2	Accuracy
RTE	Entailment	2,241	249	277	2	Accuracy
STS-B	Sentence Similarity	5,174	575	1,500	1	Pearson corr. coef.

Table 5: Summary of the GLUE benchmark datasets.

datasets rather than initializing using the model checkpoint on the MNLI dataset. We use NVIDIA A100-SXM4-80GB with 8 GPUs. Each fine-tuning process utilizes all the GPUs, taking approximately 12 GB memory per GPU. We also initiate multiple runs at the same time to make a better use of the memory.

We implement a simple hypernetwork γ with two linear layers using PyTorch. The weights are initialized using Kaiming uniform distribution for $A_H B_H$ and $A_H B_L$, and zero initialization for $A_L B_H$. The training is done using Transformers library. We utilize the evaluate library for calculating the metrics.

E Ablation Studies

We conduct experiments using LLMs other than Tiny Llama to investigate whether our method brings similar improvements for larger models and different model architectures. The results are presented in Table 7 for Pythia-1B, Table 8 for Llama2-7B, Table 9 for Pythia-6.9B, Table 10 for Pythia-12B, Table 11 for Llama2-13B, and Table 12 for Gemma2-27B.

F Additional Results

Table 13 gives the average training times. We also provide the convergence plots in Figure 9 for the rank $R = 8$ and Figure 10 for the rank $R = 16$.

	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B
Optimizer	AdamW							
Weight Decay	0.01							
β_1	0.9							
β_2	0.999							
Accumulation Steps	1							
Warmup Ratio	0.0							
Warmup Steps	0							
LR Schedule	Linear							
Batch Size per GPU	32	8	8	8	32	32	8	8
# Epochs	5	10	10	10	10	5	10	10
Learning Rate	5e-5							
Rank	{8, 16}							
α	16							
Max Seq. Len.	128							
Dropout Prob.	0.0							
Random Seeds	{100, 482, 584, 721, 770}							

Table 6: The hyper-parameters used for employing Tiny-Llama on the GLUE benchmark.

Method	Config	Rank	h_2	# Trainable Params	GLUE						
					MNLI	SST-2	MRPC	QQP	RTE	STS-B	Avg.
LoRA	$A_F B_L$		-	786,432	$78.77_{\pm0.2}$	$90.48_{\pm0.6}$	$72.01_{\pm2.8}$	$85.06_{\pm0.2}$	$60.79_{\pm2.0}$	$71.94_{\pm1.9}$	76.5
LoRA	$A_L B_L$	8	-	1,048,576	$82.42_{\pm0.3}$	$92.80_{\pm0.2}$	**$77.79_{\pm1.7}$**	$87.49_{\pm0.2}$	$65.70_{\pm1.8}$	$85.09_{\pm0.5}$	81.9
PG-Pos	$A_H B_L$		14	1,032,822	**$82.70_{\pm0.2}$**	**$92.96_{\pm0.4}$**	$76.81_{\pm1.1}$	**$87.77_{\pm0.1}$**	**$66.21_{\pm1.5}$**	**$86.49_{\pm0.4}$**	**82.2**
LoRA	$A_F B_L$		-	1,572,864	$77.87_{\pm0.4}$	$90.50_{\pm0.7}$	$71.86_{\pm2.5}$	$84.92_{\pm0.1}$	$59.49_{\pm2.5}$	$72.75_{\pm1.1}$	76.2
LoRA	$A_L B_L$	16	-	2,097,152	$82.41_{\pm0.2}$	$92.80_{\pm0.3}$	**$77.99_{\pm2.0}$**	$87.49_{\pm0.2}$	$65.20_{\pm2.0}$	$85.22_{\pm0.5}$	81.9
PG-Pos	$A_H B_L$		14	2,065,014	**$82.74_{\pm0.1}$**	**$92.91_{\pm0.4}$**	$76.52_{\pm1.2}$	**$87.81_{\pm0.1}$**	**$66.50_{\pm2.0}$**	**$86.47_{\pm0.4}$**	**84.3**

Table 7: **Performance of the methods using Pythia-1B**. The experiments are based on all the GLUE datasets over 5 randomly selected seeds. We report Matthews correlation coefficient for CoLA, Pearson correlation coefficient for STS-B, and accuracy for the remaining tasks, with the standard deviations given in the subscript. The best performance per rank is highlighted by bold.

Method	Config	Rank	h_2	# Trainable Params	GLUE					
					SST-2	MRPC	CoLA	RTE	STS-B	Avg.
LoRA	$A_F B_L$		-	2,097,152	$96.12_{\pm0.2}$	$72.89_{\pm1.5}$	$46.72_{\pm8.7}$	$56.97_{\pm2.0}$	$58.31_{\pm1.9}$	66.2
LoRA	$A_L B_L$	8	-	4,194,304	$96.26_{\pm0.3}$	$83.38_{\pm1.1}$	$64.54_{\pm0.7}$	$79.35_{\pm2.2}$	$84.47_{\pm0.9}$	81.6
PG-Pos	$A_H B_L$		28	4,000,188	**$96.31_{\pm0.3}$**	**$83.87_{\pm1.6}$**	**$66.95_{\pm2.1}$**	**$80.36_{\pm3.5}$**	**$87.95_{\pm1.9}$**	**83.0**
LoRA	$A_F B_L$		-	4,194,304	$95.99_{\pm0.4}$	$72.25_{\pm1.7}$	$36.17_{\pm14.6}$	$57.91_{\pm3.8}$	$57.24_{\pm3.7}$	63.9
LoRA	$A_L B_L$	16	-	8,388,608	$96.24_{\pm0.4}$	$83.33_{\pm1.2}$	$63.79_{\pm1.2}$	$77.33_{\pm1.3}$	$84.59_{\pm1.0}$	81.1
PG-Pos	$A_H B_L$		28	7,997,884	**$96.26_{\pm0.2}$**	**$84.26_{\pm1.3}$**	**$67.74_{\pm1.3}$**	**$79.71_{\pm3.4}$**	**$88.21_{\pm1.6}$**	**83.2**

Table 8: **Performance of the methods using Llama2-7B**. The experiments are based on all the GLUE datasets over 5 randomly selected seeds. We report Matthews correlation coefficient for CoLA, Pearson correlation coefficient for STS-B, and accuracy for the remaining tasks, with the standard deviations given in the subscript. The best performance per rank is highlighted by bold.

Method	Config	Rank	h_2	# Trainable Params	GLUE MRPC	CoLA	RTE	STS-B	Avg.
LoRA	$A_F B_L$		-	3,145,728	76.76$_{\pm 0.8}$	52.51$_{\pm 1.6}$	61.52$_{\pm 3.3}$	76.31$_{\pm 3.0}$	66.8
LoRA	$A_L B_L$	8	-	4,194,304	81.91$_{\pm 1.6}$	56.02$_{\pm 1.0}$	75.09$_{\pm 2.0}$	87.50$_{\pm 0.6}$	75.1
PG-Pos	$A_H B_L$		28	4,098,492	**82.30**$_{\pm 1.4}$	**58.39**$_{\pm 0.9}$	**77.98**$_{\pm 3.5}$	**89.19**$_{\pm 0.3}$	**77.0**
LoRA	$A_F B_L$		-	6,291,456	75.98$_{\pm 0.6}$	52.97$_{\pm 0.7}$	61.81$_{\pm 3.1}$	76.25$_{\pm 2.8}$	66.8
LoRA	$A_L B_L$	16	-	8,388,608	82.21$_{\pm 0.8}$	56.58$_{\pm 1.2}$	74.51$_{\pm 2.0}$	87.51$_{\pm 0.7}$	75.2
PG-Pos	$A_H B_L$		28	8,194,492	**82.25**$_{\pm 1.3}$	**57.55**$_{\pm 1.1}$	**76.75**$_{\pm 3.3}$	**89.07**$_{\pm 0.5}$	**76.4**

Table 9: **Performance of the methods using Pythia-6.9B**. The experiments are based on all the GLUE datasets over 5 randomly selected seeds. We report Matthews correlation coefficient for CoLA, Pearson correlation coefficient for STS-B, and accuracy for the remaining tasks, with the standard deviations given in the subscript. The best performance per rank is highlighted by bold.

Method	Config	Rank	h_2	# Trainable Params	GLUE MRPC	CoLA	RTE	STS-B	Avg.
LoRA	$A_F B_L$		-	4,423,680	67.30$_{\pm 3.0}$	51.89$_{\pm 2.3}$	58.12$_{\pm 3.3}$	66.67$_{\pm 4.6}$	61.0
LoRA	$A_L B_L$	8	-	5,898,240	**70.88**$_{\pm 2.5}$	60.70$_{\pm 1.7}$	68.09$_{\pm 2.9}$	79.74$_{\pm 4.2}$	69.9
PG-Pos	$A_H B_L$		32	5,778,592	69.17$_{\pm 2.3}$	**62.85**$_{\pm 2.3}$	**70.83**$_{\pm 2.0}$	**87.01**$_{\pm 0.9}$	**72.5**
LoRA	$A_F B_L$		-	8,847,360	67.45$_{\pm 1.0}$	51.01$_{\pm 2.2}$	57.33$_{\pm 3.1}$	65.86$_{\pm 3.5}$	60.4
LoRA	$A_L B_L$	16	-	11,796,480	69.26$_{\pm 2.5}$	60.61$_{\pm 2.0}$	66.21$_{\pm 2.6}$	83.01$_{\pm 2.9}$	69.8
PG-Pos	$A_H B_L$		32	11,553,952	**70.39**$_{\pm 4.9}$	**62.69**$_{\pm 1.8}$	**71.34**$_{\pm 2.5}$	**86.81**$_{\pm 1.0}$	**72.8**

Table 10: **Performance of the methods using Pythia-12B**. The experiments are based on all the GLUE datasets over 5 randomly selected seeds. We report Matthews correlation coefficient for CoLA, Pearson correlation coefficient for STS-B, and accuracy for the remaining tasks, with the standard deviations given in the subscript. The best performance per rank is highlighted by bold.

Method	Config	Rank	h_2	# Trainable Params	GLUE MRPC	CoLA	RTE	STS-B	Avg.
LoRA	$A_F B_L$		-	3,276,800	69.07$_{\pm 0.9}$	37.30$_{\pm 5.2}$	51.84$_{\pm 1.6}$	30.43$_{\pm 11.7}$	47.2
LoRA	$A_L B_L$	8	-	6,553,600	78.58$_{\pm 0.8}$	67.06$_{\pm 1.1}$	58.27$_{\pm 1.9}$	86.09$_{\pm 0.9}$	72.5
PG-Pos	$A_H B_L$		36	6,311,908	**82.99**$_{\pm 0.6}$	**67.53**$_{\pm 1.1}$	**62.74**$_{\pm 2.8}$	**88.88**$_{\pm 1.1}$	**75.5**
LoRA	$A_F B_L$		-	6,553,600	68.68$_{\pm 1.1}$	34.87$_{\pm 10.3}$	53.00$_{\pm 2.3}$	31.98$_{\pm 9.5}$	47.1
LoRA	$A_L B_L$	16	-	13,107,200	80.83$_{\pm 1.2}$	67.29$_{\pm 0.7}$	60.14$_{\pm 3.6}$	87.08$_{\pm 0.7}$	73.8
PG-Pos	$A_H B_L$		36	12,619,748	**83.48**$_{\pm 1.1}$	**68.56**$_{\pm 0.6}$	**62.60**$_{\pm 3.2}$	**88.66**$_{\pm 0.9}$	**75.8**

Table 11: **Performance of the methods using Llama2-13B**. The experiments are based on all the GLUE datasets over 5 randomly selected seeds. We report Matthews correlation coefficient for CoLA, Pearson correlation coefficient for STS-B, and accuracy for the remaining tasks, with the standard deviations given in the subscript. The best performance per rank is highlighted by bold.

Method	Config	Rank	h_2	# Trainable Params	GLUE SST-2	MRPC	CoLA	QNLI	RTE	STS-B	Avg.
LoRA	$A_F B_L$		-	2,260,992	96.95$_{\pm 0.2}$	78.53$_{\pm 1.0}$	64.18$_{\pm 1.1}$	95.32$_{\pm 0.1}$	70.11$_{\pm 1.7}$	75.90$_{\pm 1.9}$	80.1
LoRA	$A_L B_L$	8	-	5,652,480	97.09$_{\pm 0.2}$	**86.32**$_{\pm 1.2}$	69.04$_{\pm 1.7}$	96.10$_{\pm 0.1}$	84.12$_{\pm 0.5}$	86.53$_{\pm 1.2}$	86.5
PG-Pos	$A_H B_L$		40	5,288,920	**97.18**$_{\pm 0.1}$	86.03$_{\pm 1.3}$	**69.83**$_{\pm 1.6}$	**96.22**$_{\pm 0.1}$	**86.57**$_{\pm 1.9}$	**87.66**$_{\pm 1.3}$	**87.2**
LoRA	$A_F B_L$		-	4,521,984	96.86$_{\pm 0.2}$	78.77$_{\pm 0.9}$	63.91$_{\pm 1.6}$	95.29$_{\pm 0.1}$	70.40$_{\pm 2.1}$	75.97$_{\pm 2.1}$	80.2
LoRA	$A_L B_L$	16	-	11,304,960	97.09$_{\pm 0.2}$	85.54$_{\pm 1.1}$	68.60$_{\pm 1.8}$	96.07$_{\pm 0.1}$	83.90$_{\pm 1.5}$	86.74$_{\pm 1.0}$	86.3
PG-Pos	$A_H B_L$		40	10,572,760	**97.11**$_{\pm 0.2}$	**86.42**$_{\pm 0.8}$	**70.39**$_{\pm 1.5}$	**96.18**$_{\pm 0.1}$	**85.99**$_{\pm 2.6}$	**87.68**$_{\pm 0.8}$	**87.3**

Table 12: **Performance of the methods using Gemma2-27B-it**. The experiments are based on all the GLUE datasets over 5 randomly selected seeds. We report Matthews correlation coefficient for CoLA, Pearson correlation coefficient for STS-B, and accuracy for the remaining tasks, with the standard deviations given in the subscript. The best performance per rank is highlighted by bold.

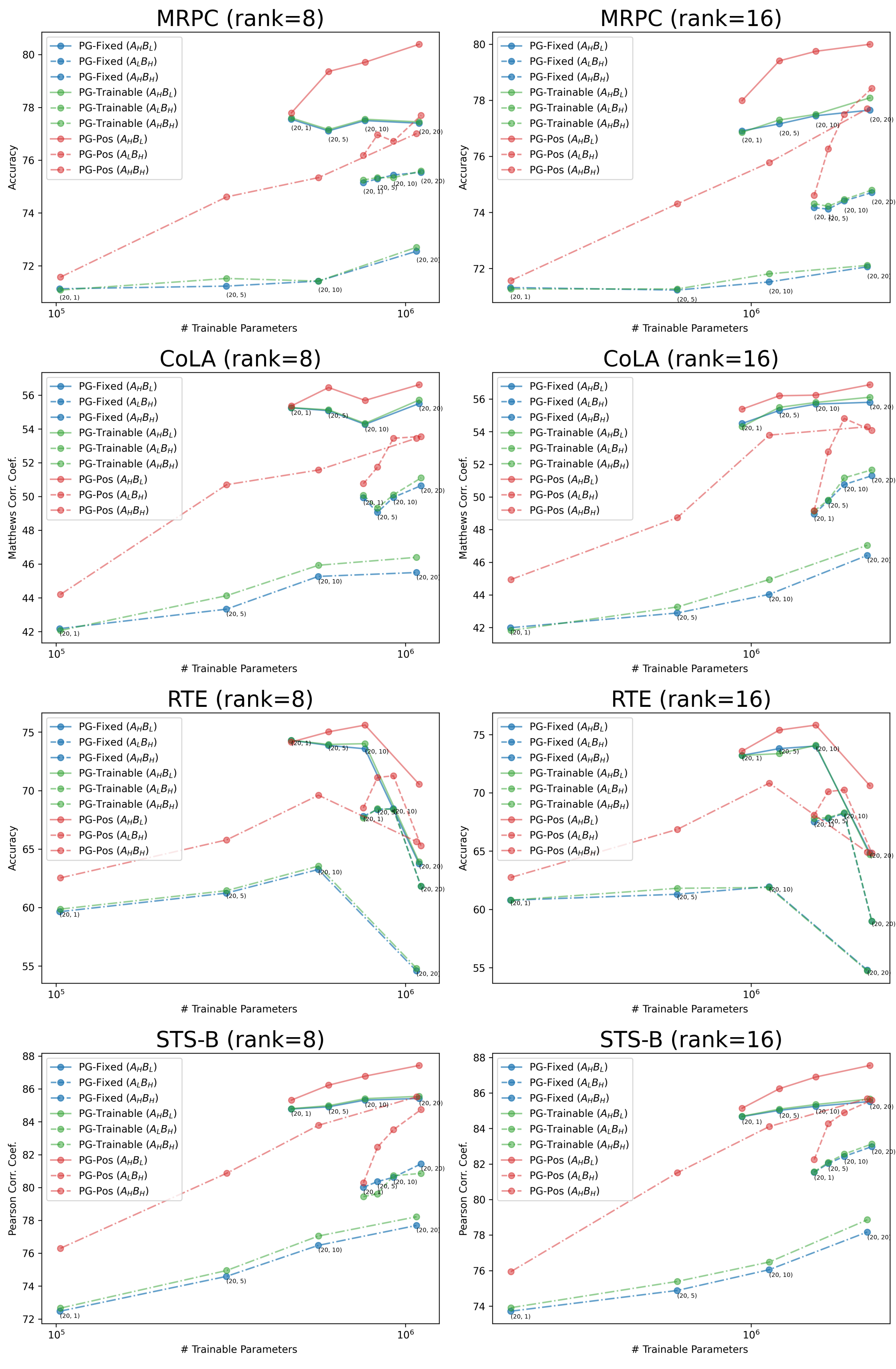
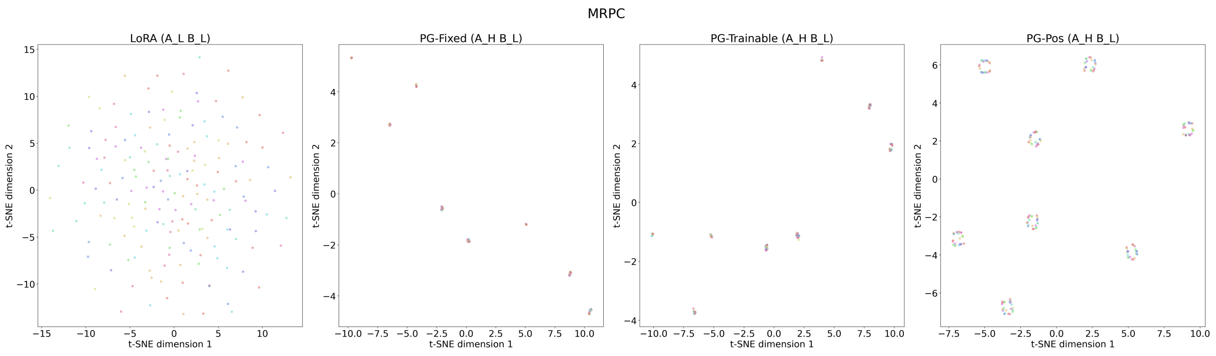
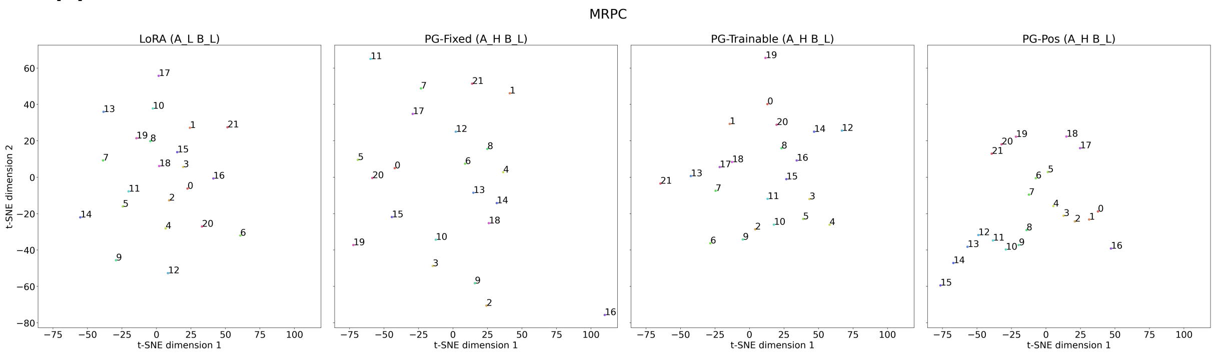


Figure 7: **Performance of the methods with respect to the number of trainable parameters** We conduct these experiments using the MRPC, CoLA, RTE and STS-B datasets. Here, we denote the hidden dimensions of hypernetworks by (h_1, h_2).



(a) Analysis of $\mathcal{S} = \{s_i\}_{i=1}^{RL}$ where each sample s_i corresponds to the c^{th} column vector $A_c^l \in \mathbb{R}^D, \forall c \in [R]$ at the l^{th} layer, $\forall l \in [L]$.



(b) Analysis for $\mathcal{S} = \{s_i\}_{i=1}^{L}$ where each sample s_i corresponds to the $A^l \in \mathbb{R}^{DR}$ at the l^{th} layer, $\forall l \in [L]$.

Figure 8: **A visualization of the A parameters provided by LoRA, PG-Fixed, PG-Trainable and PG-Pos on a two-dimensional manifold.** We obtain the parameters associated with the query matrices using the MRPC dataset with the rank $R = 8$, $D = 2048$ and $L = 22$. The parameters are then mapped to a two-dimensional manifold using t-SNE. a) shows that the A parameters share similarities across the layers. b) shows that our method learns the sequential order of the layers and their groupings.

Method	Config	Rank	# Trainable Params	GLUE								
				MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
LoRA	$A_L B_L$		1,126,400	2.12	0.65	0.08	0.15	0.98	1.91	0.08	0.11	0.76
	$A_L B_H$		1,108,388	1.93	0.57	0.08	0.15	0.87	1.78	0.08	0.11	**0.70**
PG-Fixed	$A_H B_L$		1,094,052	2.54	0.78	0.10	0.20	1.18	2.31	0.10	0.15	0.92
	$A_H B_H$		1,076,040	1.89	0.58	0.11	0.19	0.88	1.77	0.08	0.14	**0.70**
	$A_L B_H$		1,108,828	1.93	0.57	0.08	0.15	0.88	1.78	0.08	0.11	**0.70**
PG-Trainable	$A_H B_L$	8	1,094,492	2.53	0.78	0.10	0.20	1.18	2.32	0.10	0.15	0.92
	$A_H B_H$		1,076,920	1.90	0.58	0.11	0.20	0.88	1.79	0.08	0.15	0.71
	$A_L B_H$		1,109,228	1.92	0.57	0.08	0.15	0.88	1.78	0.08	0.11	**0.70**
PG-Pos	$A_H B_L$		1,094,892	2.54	0.78	0.10	0.20	1.18	2.32	0.10	0.15	0.92
	$A_H B_H$		1,077,720	1.90	0.58	0.11	0.20	0.88	1.79	0.08	0.14	0.71
LoRA	$A_L B_L$		2,252,800	2.09	0.65	0.09	0.15	0.98	1.93	0.08	0.11	0.76
	$A_L B_H$		2,216,356	1.93	0.58	0.08	0.15	0.90	1.80	0.08	0.11	**0.70**
PG-Fixed	$A_H B_L$		2,187,684	2.52	0.79	0.11	0.20	1.18	2.34	0.10	0.15	0.92
	$A_H B_H$		2,151,240	1.92	0.58	0.11	0.20	0.90	1.80	0.08	0.14	0.72
	$A_L B_H$		2,216,796	1.93	0.58	0.08	0.15	0.90	1.81	0.08	0.11	0.70
PG-Trainable	$A_H B_L$	16	2,188,124	2.52	0.79	0.11	0.20	1.19	2.34	0.10	0.15	0.92
	$A_H B_H$		2,152,120	1.92	0.58	0.11	0.20	0.90	1.80	0.08	0.15	0.72
	$A_L B_H$		2,217,196	1.94	0.58	0.08	0.15	0.90	1.81	0.08	0.11	0.71
PG-Pos	$A_H B_L$		2,188,524	2.52	0.79	0.11	0.20	1.19	2.34	0.10	0.15	0.92
	$A_H B_H$		2,152,920	1.93	0.58	0.11	0.20	0.90	1.79	0.08	0.15	0.72

Table 13: **Average training runtimes (in hours)** We run the experiments on all the GLUE datasets and average the results over 5 randomly selected seeds.

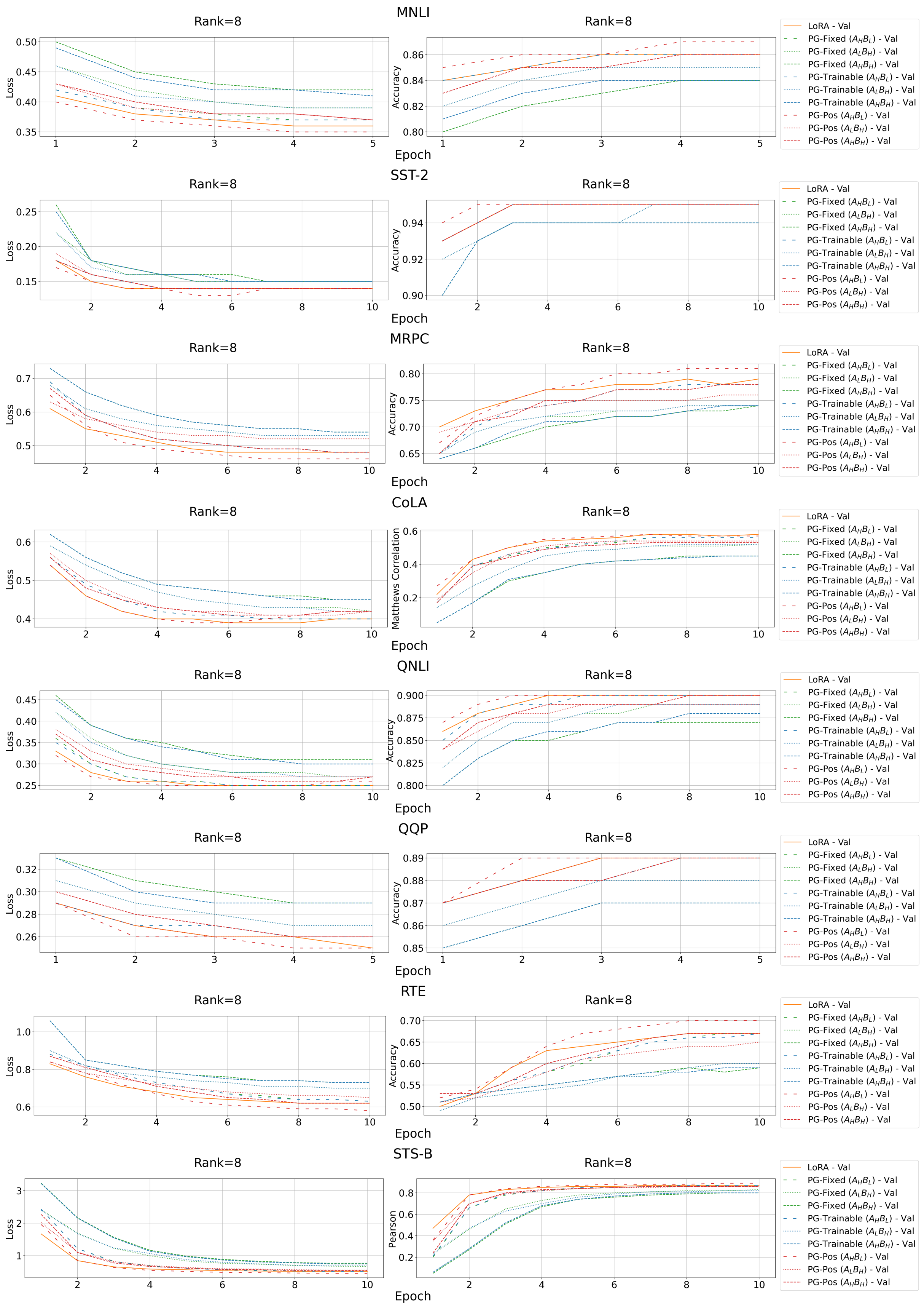


Figure 9: **Convergence of the methods on the validation datasets** The figures display the loss and the corresponding performance metric at each epoch for all the GLUE datasets and a ranks of 8.

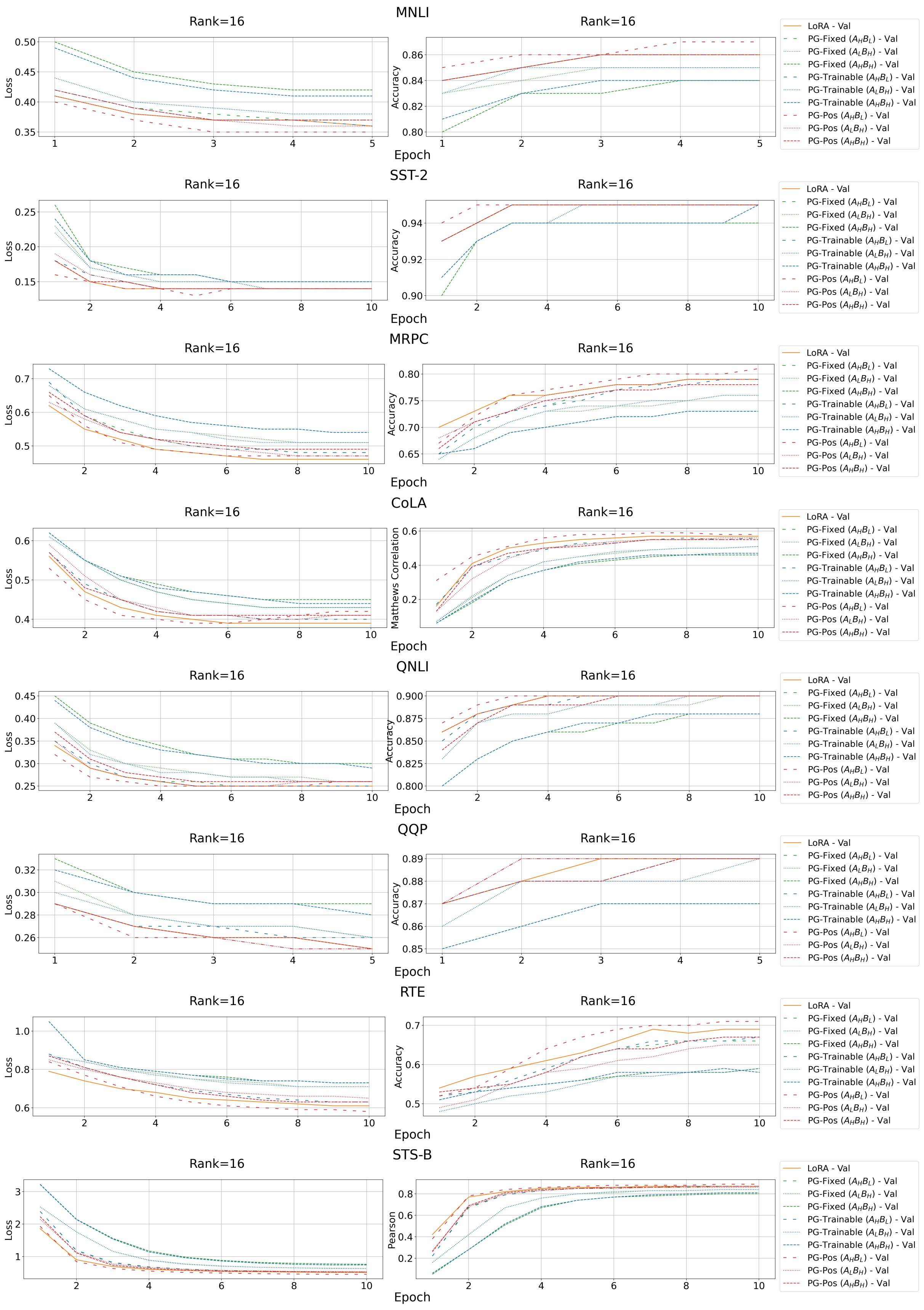


Figure 10: **Convergence of the methods on the validation datasets** The figures display the loss and the corresponding performance metric at each epoch for all the GLUE datasets and a rank of 16.