STEP: Staged Parameter-Efficient Pre-training for Large Language Models

Kazuki Yano[1] **Takumi Ito**[1,2] **Jun Suzuki**[1,3]
[1]Tohoku University [2]Langsmith Inc. [3]RIKEN
yano.kazuki.s4@dc.tohoku.ac.jp
{t-ito, jun.suzuki}@tohoku.ac.jp

Abstract

Pre-training large language models faces significant memory challenges due to the large size of model weights. We propose STaged parameter-Efficient Pre-training (STEP), which combines ideas from parameter-efficient tuning and staged training. We conduct experiments on pre-training models of various sizes and demonstrate that STEP can achieve up to a 40.4% reduction in maximum memory requirement compared to vanilla pre-training while maintaining comparable performance.

1 Introduction

Large Language Models (LLMs) have become a fundamental technology in artificial intelligence. One challenge we aim to address in the research on LLMs is the vast amount of computational resources needed for pre-training, e.g., LLaMA (Touvron et al., 2023). This requirement for enormous computational resources is a significant obstacle to the research of LLMs.

To tackle this challenge, methods for reducing computational costs during pre-training have been actively studied. For example, ReLoRA (Lialin et al., 2024) reduces the computational cost by repeatedly applying low-rank adaptations while freezing the original parameters during pre-training. However, ReLoRA often degrades performance compared to vanilla pre-training under fair conditions (Lialin et al., 2024; Zhao et al., 2024); there is still considerable room to improve in this line of studies. From this background, this paper attempts to develop a method for pre-training LLMs that can achieve comparable performance at the same computational cost as vanilla pre-training while reducing the maximum memory requirements.

For this goal, we propose a method that combines ideas of Parameter-Efficient Tuning (PET) (He et al., 2022) and staged training (Shen et al., 2022). The basic concept is that by incorporating the idea of staged training, we can reduce the maximum memory requirement by (1) pre-training a model with a smaller size in the first stage and (2) freezing the parameters already pre-trained in the previous stages and instead introducing much smaller additional training parameters following the PET technique in the remaining stages. Hereafter, we refer to our method as STaged parameter Efficient Pre-training (STEP). Figure 1 illustrates this concept.

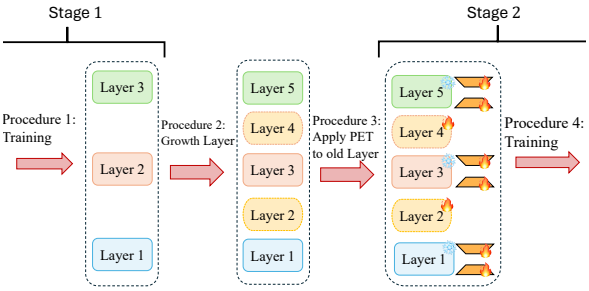

Figure 1: Overview of the STEP method. STEP performs a standard pre-training on a model with a much smaller size (Procedure 1). The stage switches and the model grows (Procedure 2); PET is applied to the layers that originally existed in the previous stage (Procedure 3). The new layers are trained with full parameters, while the weights of the originally existing layers are frozen, and only the smaller parameters (orange parts) introduced by PET are trained (Procedure 4).

We explore the effectiveness of STEP in pre-training experiments by comparing the baseline (Vannila pre-training) and conventional method (ReLoRA) under the same computational cost. We demonstrate that STEP achieves up to a 40.4% reduction in maximum memory requirements compared to vanilla pre-training while maintaining comparable validation es.

2 Related Work

Memory Efficient Training for LLMs Several memory-efficient training approaches have been actively developed in the literature of training LLMs (Rajbhandari et al., 2020; Korthikanti et al.,

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop), pages 607–614

2023). Among these, one major approach is training parameter reduction methods. One approach is PET (He et al., 2022), such as Adapter (Houlsby et al., 2019) and LoRA (Hu et al., 2022). PET techniques have mostly been developed for fine-tuning LLMs. There are only a few studies applying the PET techniques to the LLM pre-training (Lialin et al., 2024; Zhao et al., 2024). ReLoRA is a representative method designed for pre-training LLMs using LoRA (Hu et al., 2022). However, to achieve comparable performance to vanilla pre-training, ReLoRA needs to train the model in the vanilla setting for the first several steps. This implies that ReRoLA is currently unable to reduce the maximum memory requirement, as it requires the same amount of memory as vanilla pre-training.

Staged Training (Shen et al., 2022) The core idea behind Staged Training is based on the observation that while small-scale models are advantageous in the initial stages of learning from a computational efficiency perspective, large-scale models eventually achieve lower (Kaplan et al., 2020). Staged Training leverages this observation by training a small-scale model with high computational efficiency and applying an expansion operation called the Growth Operator during training. This operation expands the dimensions of Transformer layers and adds new layers. Regarding memory usage in staged training, since most existing studies train the full parameters of the model, this approach does not reduce the maximum memory requirements.

3 STEP: Staged Efficient Parameter Training

As briefly described in Section 1, our goal is to develop a method for pre-training LLMs that can achieve comparable performance at the same computational cost while reducing the maximum memory requirements during pre-training.

3.1 Procedure

The following four procedures are an overview of STEP and how it efficiently trains LLMs;

(Procedure 1) STEP performs a vanilla pre-training on a model with a much smaller size than the target model size as an initial model.

(Procedure 2) STEP expands the layers of the initial model to increase its size.

(Procedure 3) STEP also introduces the PET parameters given by the parameter-efficient adaptors for the layers trained in Procedure 1.

(Procedure 4) STEP continues to pre-train the parameters in layers newly added in Procedure 2 and the adaptors added in Procedure 3 while freezing those in layers trained in Procedure 1.

Note that The first to fourth red right-arrow in Figure 1 corresponds to Procedures 1 to 4, respectively. After finishing Procedure 4, we obtain the pre-trained model.[1]

3.2 Growth Layer Operator

This section explains how we expand the layers in Procedure 2. Given a model with n layers, the Growth Layer Operator modifies the structure of the model's layers. We use Interpolation (Chang et al., 2018; Dong et al., 2020; Li et al., 2022), which adds new layers between existing layers and initialize them with the lower layer weights, namely $\phi_{2i}^{\text{new}} = \phi_{2i-1}^{\text{new}} = \phi_i$.

We further extend it by incorporating an idea of a fusing method that averages the parameters of the two layers (O'Neill et al., 2021), namely, $\phi_{2i}^{\text{new}} = (\phi_i + \phi_{i+1})/2$, which we call Interpolation-Mean. The validity of using the average will be verified through experiments (Section 4.4). We discuss more detailed initialization in Appendix A.

3.3 Incorporating PET parameters

This section provides additional information about Procedure 3, which introduces PET parameters by the adaptors. We specifically focus on the low-rank adaptation method (Hu et al., 2022; Lialin et al., 2024) for this part.

3.4 Maximum memory requirement of STEP

We assume that the maximum memory requirement during the pre-training can be estimated by the size of model states, which include the parameters of the model itself, the gradients of the model parameters being trained, and the optimizer state.[2] Moreover, we assume that we use a typical Transformer model and the Adam optimizer (Kingma and Ba, 2014), which are a commonly used configuration for pre-training LLMs. Additionally, we assume that all parameters are represented as 32-bit

[1] Note that we have the option to continue growing the layers by repeating Procedures 2 to 4.

[2] Other memory usages, such as activations, can be reduced using methods like Activation Recomputation (Korthikanti et al., 2023).

floating-point numbers. Consequently, when the number of parameters in one layer of the Transformer is P_{layer} and the number of layers in the model is n, the memory usage of the model state, expressed in bytes, is given by

$$\begin{aligned} P_{\text{trn}} &= 4n(\underbrace{P_{\text{layer}}}_{\text{model}} + \underbrace{P_{\text{layer}}}_{\text{gradient}} + \underbrace{P_{\text{layer}} + P_{\text{layer}}}_{\text{optimizer}}) \\ &= 16nP_{\text{layer}}, \end{aligned} \tag{1}$$

where the optimizer state of Adam consists of two parts; the gradient momentum and variance.

Regarding the maximum memory requirement for STEP, let n_i be the number of layers increased in the i-th stage from the $i-1$ stage in STEP, where $n_0 = 0$. Let N_i represent the total number of layers in the i-th stage model, namely, $N_i = \sum_{k=1}^{i} n_k$. Moreover, $E(P_{\text{layer}})$ denotes the number of parameters for the single layer, P_{layer}, added by PET. Then, we can estimate the maximum memory requirement for the stage i, that is, M_i, as follows:

$$P_i^{\text{STEP}} = \begin{cases} 16n_iP_{\text{layer}} & \text{if } i = 1 \\ 16n_iP_{\text{layer}} + 4N_{i-1}P_{\text{layer}} & \\ \quad +16N_{i-1}E(P_{\text{layer}}) & \text{otherwise,} \end{cases} \tag{2}$$

where the term $4N_{i-1}P_{\text{layer}}$ represents the number of frozen model parameters already trained in the 1 to $i-1$ stages, the term $16n_iP_{\text{layer}}$ indicates the number of newly added model parameters with optimization states added in Procedure 2 and the term $16N_{i-1}E(P_{\text{layer}})$ represents the number of PET parameters with optimization states added in Procedure 3.

Let L be the number of layers for the model that is finally obtained. Then, the solution of the following minimization problem can minimize the maximum memory requirement during the pre-training:

$$\underset{\{n_1,\ldots,n_K\}}{\text{minimize}} \left\{ \max_{i=1,\ldots,K} P_i^{\text{STEP}} \right\} \quad \text{s.t.} \quad L = N_K \tag{3}$$

Details of the discussion with specific examples are presented in Appendix B.

4 Experiments

This section demonstrates the effectiveness of the proposed method, STEP, through the pre-training experiments of LLMs. We investigate whether STEP can achieve a comparable validation perplexity to vanilla pre-training at the same computational cost. We also compared with ReLoRA (Lialin et al., 2024) as a conventional method of the parameter-efficient pre-training method in a fair condition.

Stage1 →Stage2	Hidden	Layers
227M →352M	1024	18 →28
409M →668M	1760	11 →18
755M →1.2B	2048	15 →24

Table 1: The configuration of models used in the experiments using STEP. The number of parameters and layers for each model at different stages are shown.

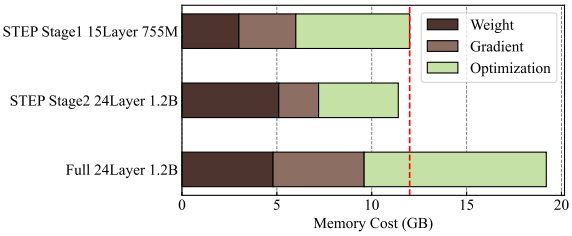

Figure 2: Memory consumption of pre-training 1.2B in Table 1. When using STEP, it is possible to increase the model size in Stage2 while keeping the memory usage consistent between both stages

4.1 Datasets and Model

We used C4 (Raffel et al., 2020) as the training data and 10M tokens exclusively extracted from C4 as the validation data. We used the identical training data for all experiments.

The model configuration follows an architecture based on LLaMA (Touvron et al., 2023). The detailed configurations are shown in Appendix C. To confirm the differences in behavior due to model size, we selected three model sizes, namely, 352M, 668M, and 1.2B.

4.2 Configuration of STEP

We evaluated the effectiveness of STEP when the Growth Layer operator is applied once during its pre-training. This means that we set $K = 2$ in Equation 3 for STEP. Given the number of layers L with the fixed dimension of hidden layers, we compute n_1 and n_2 that can minimize the maximum memory requirements by Equation 3. Table 1 shows the calculated numbers of layers and parameters when the target model sizes are one of {352M, 668M, 1.2B}. Figure 2 shows an example of memory requirements when the target model size is 1.2B for vanilla pre-training and each stage of the two-stage STEP.

Layers are added to the upper part of the Transformer layers. The discussion about the position where layers are added is provided in Appendix E.

	352M	668M	1.2B
Vanilla	17.96 (5.6G)	15.85 (10.7G)	14.61 (19.3G)
ReLoRA	21.75 (5.6G)	19.09 (10.7G)	17.81 (19.3G)
STEP	**18.14** (3.6G)	**16.15** (6.6G)	**14.84** (11.5G)

Table 2: Validation perplexities of vanilla pre-training (Vanilla), ReLoRA, and STEP. The numbers in parentheses indicate the maximum memory requirements for pre-training for each method in this experiment.

	352M	668M	1.2B
Stacking	19.03	16.89	15.52
Queueing	19.14	16.79	15.36
Interpolation-Copy	18.73	16.51	15.10
Interpolation-Mean	**18.38**	**16.23**	**14.92**

Table 3: Validation perplexities for different Growth Layer Operators

4.3 Results

Table 2 shows the validation es of vanilla pre-training, ReLoRA, and STEP. As shown in Table 2, STEP outperformed ReLoRA and achieved comparable validation to the vanilla pre-training while significantly reducing the maximum memory requirement from 5.6G to 3.6G (35.7% reduction), 10.7G to 6.6G (38.3% reduction), and 19.3G to 11.5G (40.4% reduction) for 352M, 668M, and 1.2B models, respectively. We also observed a desirable characteristic of increasing the model sizes, which led to a further reduction in maximum memory requirements, such as 35.7% to 40.4% for the 352M and 1.2B models, respectively. Based on these results, STEP has the potential to efficiently pra-train LLMs with reduced memory usage.

4.4 Ablation study

Type of Growth Layer Operators: We conducted an ablation study on Growth Layer Operators (Procedure 2) in STEP. We compared three Growth Layer operators: Stacking, Queueing, Interpolation-Copy, and Interpolation-Mean.

Stacking is proposed in Gong et al. (2019), which stacks additional layers. Queueing inserts new additional layers at the bottom. While the structure of layers resulting from Queueing is identical to that of Stacking, we need to consider this in STEP because PET is applied to the existing layers before Queueing. As in Gong et al. (2019), for both Stacking and Queueing, the weights of the additional layers are copied from the original layers. Interpolation-Copy and Interpolation-Mean are Interpolation operators that use copy and mean initialization in Section 3.2, respectively. To simplify the discussion regarding the location of layer addition, the number of layers to be added is set to be the same as the total number of the model before Growing layers; that is, the number of layers in the model is doubled compared to before the addition.

The results of this ablation study are shown in Table 3. The performance in all settings is Interpolation-Mean > Interpolation-Copy > Queueing $\approx$ Stacking. One possible reason Interpolation outperformed Stacking and Queueing is that it can add layers to preserve the overall mechanism better. Several existing studies (Meng et al., 2022; Chen et al., 2024, 2023) have reported analysis results indicating that Transformers have distinct roles for the lower, middle, and upper layers, and it is thought that Interpolation can maintain this structure, resulting in better performance compared to other operators. Moreover, Interpolation-Mean outperformed Interpolation-Copy. This result suggests that the mean initialization is superior to copy initialization. More detailed experimental explanations are described in the Appendix D.

5 Conclusion and Limitation

Pre-training LLM requires substantial memory, posing a challenge for research. We proposed a new training method called STEP, which enables LLM pre-training with limited memory requirements. Our experiments demonstrated that STEP achieved comparative performance to vanilla LLM pre-training while minimizing peak memory usage.

Several limitations of our study should be addressed in future research. First, while we conducted experiments with up to two stages in STEP, the effectiveness of using more than three stages remains unexplored. Second, although our methods reduced memory usage, we did not observe significant enhancements in training speed. Third, although validation is the standard metric for evaluating the performance of pre-training, it is still unknown whether the models pre-trained by the proposed method can improve the downstream tasks. To investigate the downstream task, we need to fine-tune all the pre-trained models. Finally, our experiments focused on relatively smaller model sizes compared to the recent LLMs with billions of parameters, such as those with 7B or more.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*.

Bo Chang, Lili Meng, Eldad Haber, Frederick Tung, and David Begert. 2018. Multi-level residual networks from dynamical systems view. In *International Conference on Learning Representations*.

Nuo Chen, Linjun Shou, Jian Pei, Ming Gong, Bowen Cao, Jianhui Chang, Jia Li, and Daxin Jiang. 2023. Alleviating over-smoothing for unsupervised sentence representation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3552–3566, Toronto, Canada. Association for Computational Linguistics.

Nuo Chen, Ning Wu, Shining Liang, Ming Gong, Linjun Shou, Dongmei Zhang, and Jia Li. 2024. Is bigger and deeper always better? probing llama across scales and layers.

Chengyu Dong, Liyuan Liu, Zichao Li, and Jingbo Shang. 2020. Towards adaptive residual network training: A neural-ODE perspective. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2616–2626. PMLR.

Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tieyan Liu. 2019. Efficient training of BERT by progressively stacking. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2337–2346. PMLR.

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

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

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*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5.

Changlin Li, Bohan Zhuang, Guangrun Wang, Xiaodan Liang, Xiaojun Chang, and Yi Yang. 2022. Automated progressive learning for efficient training of vision transformers. In *CVPR*.

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. 2024. ReloRA: High-rank training through low-rank updates. In *The Twelfth International Conference on Learning Representations*.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in gpt. In *Advances in Neural Information Processing Systems*, volume 35, pages 17359–17372. Curran Associates, Inc.

James O'Neill, Greg V. Steeg, and Aram Galstyan. 2021. Layer-wise neural network compression via layer fusion. In *Proceedings of The 13th Asian Conference on Machine Learning*, volume 157 of *Proceedings of Machine Learning Research*, pages 1381–1396. PMLR.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE.

Sheng Shen, Pete Walsh, Kurt Keutzer, Jesse Dodge, Matthew Peters, and Iz Beltagy. 2022. Staged training for transformer language models. In *International Conference on Machine Learning*, pages 19893–19908. PMLR.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Chengyue Wu, Yukang Gan, Yixiao Ge, Zeyu Lu, Jiahao Wang, Ye Feng, Ping Luo, and Ying Shan. 2024. Llama pro: Progressive llama with block expansion. *arXiv preprint arXiv:2401.02415*.

Yiqun Yao, Zheng Zhang, Jing Li, and Yequan Wang. 2024. Masked structural growth for 2x faster language model pre-training. In *The Twelfth International Conference on Learning Representations*.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. 2024. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*.

A About Zero Initialization

Shen et al. (2022); Wu et al. (2024) apply zero-initialization to some modules when applying Interpolation to preserve the value. However, as Yao et al. (2024) points out, the existing layers may receive gradients similar to the previous stage, leading to unnecessary constraints and potentially slowing down the model's convergence. Therefore, in this paper, we consistently refrained from using zero-initialization.

B STEP with LLaMA and LoRA

In STEP, we use ReLoRA (LoRA) for PET and LLaMA as the model. When not considering Grouped Query Attention (Ainslie et al., 2023) in LLaMA, the Self-Attention layer contains four matrices of size (d_{hidden}, d_{hidden}). Additionally, the FFN layer has three matrices of size ($\frac{8}{3}d_{\text{hidden}}$, d_{hidden}), and there are two vectors of size d_{hidden} for Layer Normalization. Therefore, P_{layer} is given by:

$$\begin{aligned} P_{\text{layer}} &= 4d_{\text{hidden}}^2 + 3(d_{\text{hidden}} \times \frac{8}{3}d_{\text{hidden}}) + 2d_{hidden} \\ &= 12d_{\text{hidden}}^2 + 2d_{\text{hidden}} \end{aligned} \tag{4}$$

Furthermore, since ReLoRA assigns two matrices of size (d, r) to a matrix of size (d, d), we have:

$$\begin{aligned} E(P_{\text{layer}}) &= 8(rd_{\text{hidden}}) + 3r(d_{\text{hidden}} + \frac{8}{3}d_{\text{hidden}}) \\ &= 19rd_{\text{hidden}} \end{aligned} \tag{5}$$

For example, if $d_{\text{hidden}} = 2048$ and $r = 128$, equation 2 becomes, in units of GB,

$$P_i^{\text{STEP}} = \begin{cases} 0.8n_i & \text{if } i = 1 \\ 0.2N_{i-1} + 0.8n_i + 0.079N_{i-1} & \text{otherwise} \end{cases} \tag{6}$$

C Details of training configurations

Configurations	Selected Value
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.95$)
Learning Rate	0.0003
LoRA rank	128
Learning Rate Schedule	cosine restarts (Lialin et al., 2024)
Restart warmup steps	500
Warmup Steps	1000
Training tokens (billions)	20B

Table 4: List of training configurations in our experiments

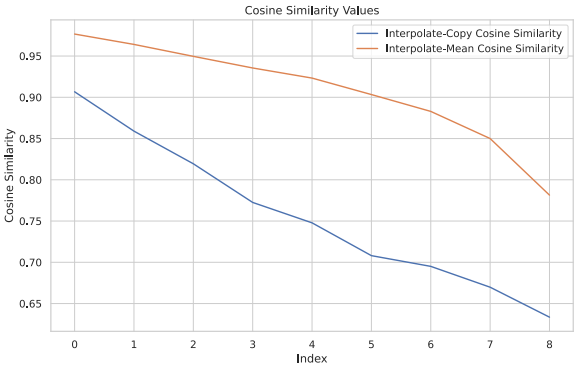

Figure 3: The image compares two methods for initializing added layers in Interpolation. The x-axis represents the index of the added layer, while the y-axis shows the cosine similarity between the output of the expanded model after adding the layer and the input to the original model before adding the layer. The blue line indicates the results when the added layer is initialized as a copy of the layer below, while the orange line shows the results when the added layer is initialized as the mean of the layers above and below. Initialization using the mean better preserves the connections between layers.

The training configurations used in the experiment are shown in Table 4. These settings are the same across all experiments.

D Interpolation-Copy and Interpolation-Mean

To verify whether the mean initialization in Interpolation actually possesses the desired properties, we compare the cosine similarity between the output of the added layer $\phi_{\text{new\_}i}$ after Interpolation and the input to the layer ϕ_{i+1} before Interpolation. In this case, if the output of $\phi_{\text{new\_}i}$ is similar to the input of ϕ_{i+1}, it can be considered that $\phi_{\text{new\_}i}$ appropriately processes the output from ϕ_i and outputs something that is easy for ϕ_{i+1} to process. We apply Interpolation to a 334M model with nine layers that have been trained on the C4 dataset (Raffel et al., 2020) and has a perplexity of 18.54 on the validation dataset. Interpolation expands the model to a 668M model with 18 layers. Subsequently, using the same validation dataset, we obtain the embeddings of the output from $\phi_{\text{new\_}i}$ in the expanded 668M model and the embeddings of the input to ϕ_{i+1} in the original 334M model before Interpolation. We then compare the cosine similarity between these two embeddings. The results are shown in Figure 3. As expected, using the average initialization yields a higher cosine similarity compared to copy initialization, suggesting that the

		668M
Vanilla		15.85 (10.7G)
bottom_index	0	16.58 (6.6G)
	1	16.40 (6.6G)
	2	16.25 (6.6G)
	3	**16.15** (6.6G)

Table 5: Validation perplexities when changing the location of the additions. As the bottom_index increases, it indicates that the additions are made closer to the top of the model.

connections between layers are better preserved.

E Effective position for adding new layers

This ablation study investigates the most effective position to add new layers when applying the Growth Layer operator using Interpolation-Mean in Procedure 2. In this ablation study, we perform layer additions on 409M →668M configurations in Table 1. We added the seven layers together starting from the bottom_index to the upper layer. In other words, when bottom_index is 3, the layers are added together at the top, and when it is 0, they are added at the bottom. The experimental results are shown in Table 5. As a general trend, the results indicate that adding layers to the upper part of the model leads to better performance improvements.