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

Kazuki Yano[1] **Takumi Ito**[1,2] **Jun Suzuki**[1,3,4]
[1]Tohoku University [2]Langsmith Inc. [3]RIKEN [4] NII LLMC
yano.kazuki@dc.tohoku.ac.jp
{t-ito, jun.suzuki}@tohoku.ac.jp

Abstract

Pre-training large language models (LLMs) faces significant memory challenges due to the large size of model parameters. We introduce STaged parameter-Efficient Pre-training (STEP), which integrates parameter-efficient tuning techniques with model growth. We conduct experiments on pre-training LLMs of various sizes and demonstrate that STEP achieves up to a 53.9% reduction in maximum memory requirements compared to vanilla pre-training while maintaining equivalent performance. Furthermore, we show that the model by STEP performs comparably to vanilla pre-trained models on downstream tasks after instruction tuning.

1 Introduction

Large Language Models (LLMs) have become an indispensable foundational technology in artificial intelligence. Recent LLM development trends, based on scaling laws (Kaplan et al., 2020), involve pre-training Transformer models with a vast number of parameters on massive datasets (Brown et al., 2020). Consequently, the pre-training of LLMs requires substantial computational resources, typically involving thousands of GPUs (Touvron et al., 2023). This enormous computational demand presents a significant obstacle to LLM research.

To tackle this challenge, we consider methods for reducing the computational demand in LLM pre-training. While there are various approaches to reducing this, we introduce a pre-training method that maintains performance equivalent to vanilla pre-training while constraining the maximum GPU memory requirements to a predetermined threshold. Specifically, our approach combines model growth (Chen et al., 2022; Wang et al., 2024) through layer addition with parameter-efficient tuning techniques (Hu et al., 2022), which are commonly used in fine-tuning. For a detailed explanation of the proposed method, Figure 1 presents

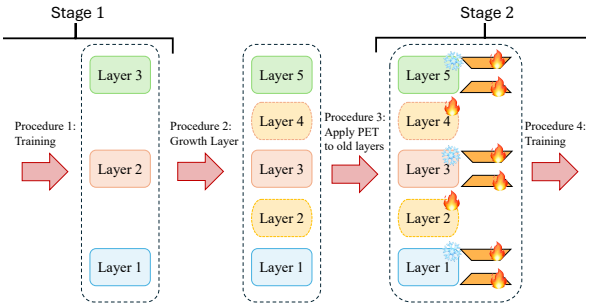

Figure 1: Overview of STEP (STaged parameter Efficient Pre-training). First, vanilla pre-training is performed on a small-scale model (Procedure 1). Subsequently, new layers are added to grow the pre-trained model (Procedure 2). The parameters of the pre-trained layers are then frozen, and Parameter-Efficient Training (PET) is applied for alternative training (Procedure 3), followed by retraining of the expanded model (Procedure 4). In Procedure 4, only the parameters added through layer expansion and the small-scale parameters introduced by PET are subject to training.

an overview of our procedure. Our approach formulates the maximum memory requirements for each stage of the sequential model growth as an integer programming problem, using model configurations as variables. We solve this optimization problem to determine the optimal model configurations for each stage, thereby controlling model growth settings to minimize peak memory usage prior to pre-training execution. This approach enables pre-training while maintaining memory requirements within a predetermined threshold. Hereafter, we refer to our method as STaged parameter Efficient Pre-training (STEP). We demonstrate that STEP achieves up to a 53.9% reduction in maximum memory requirements compared to vanilla pre-training while maintaining equivalent perplexity and performance on domain-specific tasks. Furthermore, we verify that STEP does not negatively affect the performance of downstream tasks by demonstrating that STEPed models perform on par

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers), pages 374–384

with the vanilla pre-trained model.

2 Related Work

Several memory-efficient training approaches have been actively developed in the literature of training LLMs (Rajbhandari et al., 2020; Korthikanti et al., 2023). One of the primary approaches involves reducing the number of trainable parameters. Notable examples include Parameter-Efficient Tuning (PET) methods such as Adapter (Houlsby et al., 2019) and LoRA (Hu et al., 2022). Meanwhile, to reduce FLOPs during pre-training, model growth techniques have been proposed (Chen et al., 2022; Pan et al., 2024), where training begins with a small-scale model and continues as the model parameters are gradually expanded. Our proposed method aims to achieve memory-efficient pre-training by appropriately combining PET and model growth techniques.

Paremeter-efficient Tuning. PET has primarily been developed for fine-tuning LLMs. For instance, LoRA is a technique that adds new adapters (low-rank matrices) while keeping the pre-trained LLM parameters frozen, and only trains these adapters. Since adapters typically contain few parameters, training can be accomplished with minimal memory requirements.

PET is now being applied to pre-training applications. Here, we describe two representative methods: ReLoRA (Lialin et al., 2024) and GaLore (Zhao et al., 2024). ReLoRA is a method for pre-training LLMs using LoRA. A distinctive feature of ReLoRA is that it begins with vanilla pre-training and transitions to LoRA during the training process. Consequently, from a peak memory requirement perspective, ReLoRA requires the same amount of memory as vanilla pre-training. GaLore is a method that leverages the low-rank structure of gradients to reduce optimizer states while maintaining performance equivalent to vanilla pre-training. Unlike ReLoRA, GaLore operates with low memory requirements throughout the entire training process. These methods can reduce memory usage compared to vanilla pre-training, but they slightly underperform.

Growing pre-trained model. Recent studies have shown that growing a smaller model and then continuing to train the larger model can achieve comparable performance with fewer FLOPs compared to training a large model from scratch (Shen et al., 2022; Chen et al., 2022; Pan et al., 2024). In these methods, the operation of increasing the model size is called the Growth Operator, expanding the dimensions of Transformer (Vaswani et al., 2017) layers and adding new layers. Since existing studies train the full parameters of the model, this approach does not reduce the maximum memory requirements.

3 STEP: STaged parameter Efficient 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 using the Growth Operator.

(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.

After finishing Procedure 4, we obtain the pre-trained model, or we can continue growing the layers by repeating Procedures 2 to 4, alternatively. Note that the first to fourth red right-arrows in Figure 1 corresponds to Procedures 1 to 4, respectively.

We select Interpolation used in Chang et al. (2018); Dong et al. (2020); Li et al. (2022) as the Growth Operator in Procedure 2, which adds new layers between existing layers.[1] Moreover, we select the low-rank adaptation method (Hu et al., 2022; Lialin et al., 2024) as PET parameters for performing Procedure 3.

3.2 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 model parameters, gradients, and optimizer state.[2] More-

[1] We discuss more detailed initialization of the new layers in Appendices A and B.

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

over, we assume that we use a typical Transformer model (Vaswani et al., 2017) and the Adam optimizer (Kingma and Ba, 2015) with mixed-precision training (Micikevicius et al., 2018). Specifically, model parameters and gradients are represented in 16-bit floating-point numbers, while optimizer states are represented in 32-bit floating-point numbers. 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

$$\begin{aligned} P_{\text{trn}} &= n(\underbrace{2P_{\text{layer}}}_{\text{model}}+\underbrace{2P_{\text{layer}}}_{\text{gradient}}+\underbrace{12P_{\text{layer}}}_{\text{optimizer}}) \\ &= 16nP_{\text{layer}}, \end{aligned} \tag{1}$$

where the Adam optimizer state consists of three parts: model, 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. Let N_i represent the total number of layers in the i-th stage model: $N_i = \sum_{k=1}^{i} n_k$, where $N_0 = 0$. Moreover, $E(P_{\text{layer}})$ denotes the number of parameters for a single layer, P_{layer}, added by PET.[3] Then, we estimate the maximum memory requirement for the stage i, that is, P_i^{STEP}, as follows:

$$\begin{aligned} P_i^{\text{STEP}} =& 16n_iP_{\text{layer}} + 2N_{i-1}P_{\text{layer}} \\ &+ 16N_{i-1}E(P_{\text{layer}}) \end{aligned} \tag{2}$$

where the $2N_{i-1}P_{\text{layer}}$ represents the number of frozen model parameters already trained in the 1 to $i-1$ stages, the $16n_iP_{\text{layer}}$ indicates the number of newly added model parameters with optimization states added in Procedure 2 and the $16N_{i-1}E(P_{\text{layer}})$ represents the number of PET parameters added in Procedure 3. Note that Eq. 2 is identical to Eq. 1 if $i = 1$ since $N_0 = 0$.

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:

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

This minimization problem is essentially an integer linear programming (ILP) problem since n_i for all i are non-negative integers. Thus, we can straightforwardly obtain the solution set $\{n_i\}_{i=1}^{K}$ by using a standard ILP solver or manual calculation if K is small, e.g., $K = 2$. Typically, K is small, at most $L-1$, and usually stays below $L/4$, ensuring the problem remains computationally tractable. As a result, the computational cost is negligible compared to LLM pre-training.[4]

[3] Appendix C discusses examples of P_{layer} and $E(P_{\text{layer}})$.

Model Size	Hidden	Layers
215M →368M	1600	7 →12
396M →680M	1536	14 →24
704M →1.2B	2048	14 →24
553M →956M →1.2B	2048	11 →19 →24

Table 1: The STEP configurations used in the experiments. The number of parameters and layers for each model at different stages are shown. The last row shows a three-stage growth process.

4 Experiments

We investigate whether STEP can perform equivalent to vanilla pre-training for LLMs at the same FLOPs.[5] We also compare ReLoRA (Lialin et al., 2024) and GaLore (Zhao et al., 2024) as parameter-efficient pre-training methods in a fair condition. Furthermore, to verify whether STEP would not negatively affect the performance of downstream tasks, we will perform instruction tuning on both the STEPed model and the vanilla pre-trained model and compare their performance.

4.1 Evaluation in pre-training

Datasets and model. We used FineWeb-Edu (Penedo et al., 2024) as the pre-training data. The model configuration follows LLaMA (Touvron et al., 2023). The detailed configurations are shown in Appendix F. We selected three different model sizes, namely, 368M, 680M, and 1.2B, to examine whether different model sizes lead to different trends.

Evaluation. We calculated the perplexities on two held-out validation sets: one from FineWeb-Edu (10M tokens) and the other from Wiki-Text (0.3M tokens) (Merity et al., 2017). Furthermore, we evaluated the accuracy of several typical downstream tasks for evaluating LLMs. [6]

Configuration of STEP. We focus on evaluating STEP when the Growth Layer Operator is applied once during its pre-training, that is, STEP-2stages

[4] More discussions of the complexity of ILP problems for STEP are in Appendix D.

[5] The detailed FLOPs computation is in Appendix E.

[6] Detailed evaluation settings and tasks are in Appendix G

	Perplexity ↓		Accuracy ↑						
	Validation	Wikitext	LAMBADA	ARC-e	ARC-c	Winogrande	PIQA	OBQA	HellaSwag
368M									
Vanilla (5.9G)	16.9	32.1	29.2	52.2	27.3	50.3	64.9	**32.4**	37.3
ReLoRA (5.9G)	17.4	33.1	28.8	51.9	27.8	50.5	65.1	31.2	36.5
GaLore (3.3G)	21.6	43.1	22.8	48.1	25.7	**51.2**	62.5	30.8	31.7
STEP-2stages (3.4G)	**16.7**	**31.5**	**31.5**	**52.3**	**28.4**	49.7	**65.5**	32.0	**37.8**
680M									
Vanilla (10.9G)	**14.6**	**26.0**	34.8	55.8	**30.2**	52.3	**69.7**	**36.2**	43.2
ReLoRA (10.9G)	15.1	27.3	34.0	54.1	29.0	52.1	67.3	33.8	42.1
GaLore (6.0G)	19.4	37.5	25.0	49.1	26.2	51.4	62.4	29.6	33.8
STEP-2stages (6.3G)	**14.6**	**26.0**	**35.4**	**56.0**	29.7	**55.3**	67.7	34.2	**43.7**
1.2B									
Vanilla (19.3G)	**12.9**	**22.1**	**39.9**	62.0	31.1	52.1	71.0	34.6	48.8
ReLoRA (19.3G)	13.5	23.6	37.0	60.3	31.1	51.9	70.1	34.6	46.6
GaLore (10.4G)	17.4	35.3	28.0	51.9	26.6	50.4	65.7	32.2	36.6
STEP-2stages (10.6G)	**12.9**	22.3	39.7	**62.4**	**34.3**	**54.8**	70.0	35.4	48.4
STEP-3stages (8.9G)	**12.9**	**22.1**	38.7	61.0	32.7	53.8	**71.2**	**35.6**	**48.9**

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

	Writing	Roleplay	Reasoning	Math	Coding	Extraction	STEM	Humanities	Average
Vanilla 1.2B	2.85	3.25	**2.60**	1.10	1.00	1.10	3.20	2.75	2.26
STEP-2stages 1.2B	**3.10**	**3.95**	1.95	1.00	1.05	1.10	**3.73**	2.60	**2.30**
STEP-3stages 1.2B	2.85	3.30	1.95	**1.35**	**1.10**	1.10	3.25	**3.20**	2.26

Table 3: Category-specific and average scores on MT-Bench to the answers generated by models instruction-tuned with vanilla pre-trained models (Vanilla) and STEPed models (STEP-2stages and STEP-3stages).

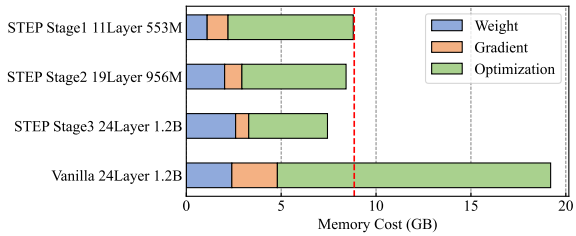

Figure 2: Memory consumption of pre-training 1.2B in Table 1. STEP allows for increasing the model size while keeping memory usage consistent at every stage.

($K = 2$). Additionally, we evaluate the STEP-3stages ($K = 3$) only for the 1.2B model.

Given the number of layers L with the fixed dimension of hidden layers, we compute $\{n_1, n_2\}$ for STEP-2stages, or $\{n_1, n_2, n_3\}$ for STEP-3stages, respectively, that can minimize the maximum memory requirements by Eq. 3. Table 1 shows the calculated numbers of layers when the target model sizes are one of {368M, 680M, 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 STEP-3stages.

The schedule for applying the Growth Layer Operator is set to occur when 75% of the total training steps for each stage have been completed.

Results. Table 2 shows the performance of vanilla pre-training, ReLoRA, GaLore, and STEP. STEP outperformed both ReLoRA and GaLore. Additionally, STEP achieved equivalent performance to the vanilla pre-training while significantly reducing the maximum memory requirement from 5.9G to 3.4G (42.3% reduction), 10.9G to 6.3G (42.2% reduction), and 19.3G to 8.9G (53.9% reduction) for 368M, 680M, and 1.2B models, respectively. Furthermore, the results of STEP-2stages and STEP-3stages at 1.2B parameters show that increasing the number of stages leads to further reduction in memory usage without compromising performance. These results suggest that STEP can efficiently pre-train LLMs with reduced memory usage.[7]

4.2 Evaluation in instruction tuning

Data and evaluation measure. For instruction tuning, we used the Alpaca dataset (Taori et al.,

[7]Appendix J discusses the mechanism behind STEP.

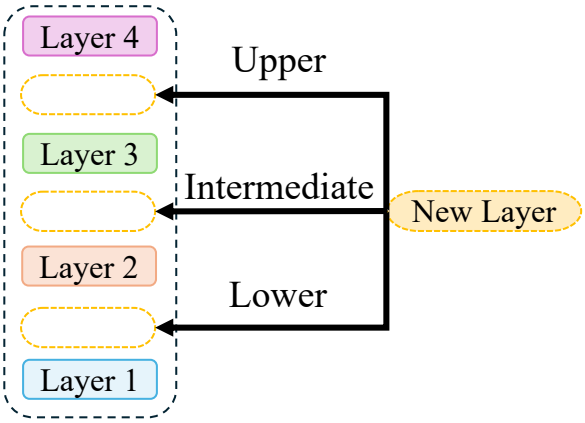

Figure 3: Illustration of different strategies for adding new layers in STEP. 'Upper' adds layers at the top, 'Intermediate' inserts layers in the middle, and 'Lower' adds layers at the bottom.

2023). Details of the training configurations are presented in Appendix H. We compare three 1.2B models one trained with vanilla pre-training, while the other two were trained using STEP (STEP-2stages, STEP-3stages). We evaluate these instruction-tuned models on MT-Bench (Zheng et al., 2024) by generating model responses to 80 multi-turn questions and assign a numerical rating out of 10 to each response by GPT-4 (Achiam et al., 2023).

Results. Table 3 shows the MT-bench scores of the vanilla pre-trained models (Vanilla) and STEPed models (STEP-2stages and STEP-3stages). We found that the scores of STEPed models were either equal to or slightly higher than those of the vanilla pre-trained model. These results indicate that STEP does not have a negative impact on downstream tasks.

5 Ablation Study

We examine the effective position for new layers and the effectiveness of PET, both key components of STEP. [8] We used the model settings with a target size of 680M from Table 1.

Effective position for adding new layers. We investigated the most effective position for performance improvement when using Interpolation-Mean in Procedure 2 of STEP. As shown in Figure 3, we conducted experiments for Upper, where new layers are added collectively at the top; Intermediate, where they are inserted in the middle; and Lower, where they are added at the bottom. Additionally, we conducted experiments for Random, where the position of additional layers is determined randomly.

	position	**680M**
Vanilla		14.56
STEP-2stages	Upper	**14.56**
	Intermediate	14.80
	Lower	15.06
	Random	14.82

Table 4: Validation perplexities for vanilla pre-trained models (Vanilla) and STEPed model (STEP-2stages) when changing the location of newly added layers.

		680M
Vanilla		14.56 (10.9G)
STEP-2stages	w/ PET	**14.56** (6.34G)
	w/o PET	14.66 (5.32G)

Table 5: Validation perplexities for vanilla pre-trained models (Vanilla) and STEPed model (STEP-2stages) w/ and w/o PET.

As shown in Table 4, we can see a trend that performance improves more when layers are added towards the upper part, and this is better than randomly deciding the location for layer addition.

The effect of PET parameters. This experiment verifies whether the PET introduced in STEP contributes to performance improvement. Specifically, we conducted an experiment skipping Procedure 3 in Section 3.1.

As shown in Table 5, PET contributes to performance improvement, and without it, the performance is inferior to the vanilla pre-trained model.

6 Conclusion

Pre-training LLM requires substantial memory, posing a challenge for LLM research. We proposed a novel training method called STEP, which enables LLM pre-training with reduced memory requirements. Our experiments demonstrated the effectiveness of STEP; specifically, STEP achieved equivalent performance to vanilla pre-training and downstream tasks after instruction tuning, while reducing peak memory usage by up to 53.9%. We hope our results encourage researchers who aim to engage in LLM pre-training research but have only limited computing resources.

[8] The ablation study on the initialization methods for new layers and the schedule of applying the Growth Operator is conducted in Appendix I.

Limitations

Several limitations of our study should be addressed in future research. First, our experiments have been limited to the FineWeb-Edu dataset and only LLaMA architecture. We need to see if the results can be replicated on other pre-training datasets and other architectures. Second, our experiments focused on relatively smaller model sizes compared to the recent LLMs with billions of parameters, such as those with 7B or more. Third, since STEP begins training with smaller models, it requires a larger amount of training tokens at the same FLOPs of vanilla pre-training. While we conducted experiments in situations where the training corpus is unconstrained, the effectiveness of STEP in data-constrained situations remains unexplored. Finally, this paper focuses its experiments on Transformers, as they are the most commonly used architecture for LLMs. However, the potential applicability to other architectures, such as State Space Models (Gu and Dao, 2024), has not been verified in this study.

Ethical Considerations

We exclusively used publicly available datasets for pre-training, fine-tuning, and evaluation. Moreover, we developed the language models entirely from scratch, avoiding the use of any publicly available models. Given that our proposal is a framework for pre-training language models, the risk of ethical concerns is minimal.

Acknowledgements

This work was supported by the "R&D Hub Aimed at Ensuring Transparency and Reliability of Generative AI Models" project of the Ministry of Education, Culture, Sports, Science and Technology, and JST Moonshot R&D Grant Number JPMJMS2011-35 (fundamental research).

In this research work, we used the "mdx: a platform for building data-empowered society".

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report.

Naman Agarwal, Pranjal Awasthi, Satyen Kale, and Eric Zhao. 2024. Stacking as accelerated gradient descent. *arXiv preprint arXiv:2403.04978*.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. 2023. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, Singapore. Association for Computational Linguistics.

Yonatan Bisk, Rowan Zellers, Ronan bras, Jianfeng Gao, and Choi Yejin. 2020. Piqa: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:7432–7439.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

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*.

Cheng Chen, Yichun Yin, Lifeng Shang, Xin Jiang, Yujia Qin, Fengyu Wang, Zhi Wang, Xiao Chen, Zhiyuan Liu, and Qun Liu. 2022. bert2BERT: Towards reusable pretrained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2134–2148, Dublin, Ireland. Association for Computational Linguistics.

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

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.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2024. A framework for few-shot language model evaluation.

Albert Gu and Tri Dao. 2024. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*.

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.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA.

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*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed precision training. In *International Conference on Learning Representations*.

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

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.

Yu Pan, Ye Yuan, Yichun Yin, Jiaxin Shi, Zenglin Xu, Ming Zhang, Lifeng Shang, Xin Jiang, and Qun Liu. 2024. Preparing lessons for progressive training on language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18860–18868.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, Berlin, Germany. Association for Computational Linguistics.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

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

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.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

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.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Yite Wang, Jiahao Su, Hanlin Lu, Cong Xie, Tianyi Liu, Jianbo Yuan, Haibin Lin, Ruoyu Sun, and Hongxia Yang. 2024. LEMON: Lossless model expansion. In *The Twelfth International Conference on Learning Representations*.

Chengyue Wu, Yukang Gan, Yixiao Ge, Zeyu Lu, Jiahao Wang, Ye Feng, Ying Shan, and Ping Luo. 2024. LLaMA pro: Progressive LLaMA with block expansion. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6518–6537, Bangkok, Thailand. Association for Computational Linguistics.

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*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy. Association for Computational Linguistics.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. 2024. Galore: Memory-efficient LLM training by gradient low-rank projection. In *Forty-first International Conference on Machine Learning*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2024. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36.

A The initialization of the new layer

When using Interpolation, most existing studies (Shen et al., 2022; Li et al., 2022; Wu et al., 2024) have adopted the method of copying weights from lower layers to initialize new layers, specifically $\phi_{2i}^{\text{new}} = \phi_{2i-1}^{\text{new}} = \phi_i$, which we call Interpolation-Copy. On the other hand, bert2BERT (Chen et al., 2022) proposed a method to expand the width by not only copying from lower layers but also mixing weights copied from both lower and upper layers, demonstrating improved performance compared to simple copying from lower layers. Inspired by this, we further extend Interpolation 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. Shen et al. (2022); Wu et al. (2024) apply zero-initialization, called function preserving initialization (FPI), to some modules when applying Interpolation to preserve the loss 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 convergence of the model. Therefore, we do not use FPI. The validity of these settings will be verified through experiments.

B Overfitting in smaller initial models

Although there might be concerns about overfitting in the STEP method due to initial training on smaller models, according to Kaplan's Scaling Law (Kaplan et al., 2020), overfitting can be mitigated with sufficient data. Given that pre-training of large language models typically involves vast amounts of data, this abundance of data in LLM pre-training scenarios theoretically minimizes overfitting risks.

C STEP with LLaMA and LoRA

In STEP, we use ReLoRA 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:

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

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

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

D Complexity of ILP Problems

The integer linear programming (ILP) used in STEP is not particularly complex. The upper bound on the number of growth stages is the final number of layers, L, e.g., $L = 24$. In practical applications, the number of growth stages, K, is typically small (e.g., $K = 2$ or $K = 3$, or at most around $L/4$). This results in a relatively small number of variables, which helps limit the problem's complexity. In our experiments using an integer programming solver, we obtained solutions within 2 or 3 seconds for cases where $K \approx 10$, though actual speed may vary depending on the performance of the hardware and the solver's implementation. Therefore, the computational cost is negligible compared to the LLM pre-training, which takes at least several hours, and is not a significant concern.

E FLOPs Computation

Let C be the FLOPs, N the number of non-embedding parameters, and T the total number of tokens used in training. Then, $C \approx 6NT$. The coefficient 6 represents the number of floating point operations required for one step, consisting of 2 floating point operations for the forward pass and 4 floating point operations for other calculations such as the backward pass. Therefore, if we denote the number of trainable parameters as $N_{\text{trainable}}$ and the number of frozen, untrainable parameters as $N_{\text{untrainable}}$, the FLOPs can be calculated as $C \approx (6N_{\text{trainable}} + 2N_{\text{untrainable}})T$.

F Details of pre-training configurations

We used GPT-2 vocabulary (Radford et al., 2019), although the architecture is based on LLaMA. The training configurations common to all model settings (368M, 680M, 1.2B) are shown in Table 6. The training configurations specific to each model setting are presented in Table 7. We adhered to the hyperparameter settings reported in the papers for ReLoRA (Lialin et al., 2024) and GaLore (Zhao et al., 2024). All experiments run on NVIDIA A100 GPUs.

Configurations	Selected Value
Common settings	
Optimizer	AdamW ($\beta_1 = 0.9, \beta_2 = 0.95$)
Weight decay	0.1
Learning rate schedule	cosine
Warmup steps	1000
Seq. len.	1024
ReLoRA settings	
LoRA rank	128
ReLoRA reset	5000
Restart warmup steps	500
GaLore settings	
GaLore rank	128
Update projection gap	200
Galore scale	0.25

Table 6: List of training configurations common to all model sizes in pre-training experiments in Section 4.1.

Re-initialization of learning rate scheduler. When adding layers in Procedure 2, we reset the optimizer state for old layers by applying PET to those. Moreover, in Procedure 4, to facilitate more efficient training of the new layers, the learning rate is rewarmed to the value used in Procedure 1.

G Evaluation of pre-trained models

Using the lm-evaluation-harness framework, we report the acc-norm score to follow Brown et al. (2020). For language modeling tasks, we evaluated perplexity on the Wiki-text dataset (Merity et al., 2017) and accuracy on the LAMBADA dataset (Paperno et al., 2016). We assessed zero-shot performance on various commonsense reasoning tasks, including WinoGrande (Sakaguchi et al., 2021), PIQA (Bisk et al., 2020), and HellaSwag (Zellers et al., 2019). Additionally, we measured zero-shot performance on question-answering tasks, specifically ARC (Clark et al., 2018) and OBQA (Mihaylov et al., 2018). We utilized the lm-evaluation-harness framework (Gao et al., 2024) and reported the acc-norm score to follow Brown et al. (2020).

H Details of instruction-tuning configurations

We show the training configurations used in the instruction tuning in Table 8. All three instruction-tuned models in Table 4.2 undergo full-parameter tuning.

I Extensive ablation study

Initialization of the new layer. As described in Section A, we investigate the impact of initialization. We conducted four experiments, with and without FPI, for both Interpolation-Copy and Interpolation-Mean. The results of this ablation study are shown in Table 9. As an overall trend, we can see that using FPI does not lead to significant performance improvements. We expected Interpolation-Mean to contribute more to performance improvement than Copy, and while this is true when FPI is not used, Interpolation-Mean with FPI showed the most significant performance degradation. FPI had little impact on performance and actually tended to degrade it, while Interpolation-Mean without FPI demonstrated the best performance results.

The schedule for applying the Growth Layer Operator. While in our experiments (Section 4.1), the Growth Layer Operator was applied at 75% of the training steps in each stage, this experiment examined the schedule timing in more detail. Specifically, we conducted four experiments, applying the Growth Layer Operator at 25%, 50%, 75%, and 100% completion of the training steps. The experimental results are shown in Table 10. As the results indicate, the best performance was achieved at 50% and 75% points, while applying the Growth Layer Operator at 25% and 100% points showed relatively poor results. One possible reason for this is that at the 25% point, the training of each layer has not yet progressed sufficiently, and applying PET to existing layers in this state may dramatically slow down the training of each layer. Additionally, applying the Growth Layer Operator at the 100% point may cause the model to escape from local optima due to learning rate rewarm and optimizer state resets, resulting in increased loss and requiring more training steps to converge to a better optimal solution.

J Discussion on the mechanisms behind STEP

In this section, in discussing why STEP works sufficiently well, we will focus our discussion on Model Growth and Parameter-Efficient Tuning, which constitute STEP.

Optimization dynamics of model growth. Recent research by Agarwal et al. (2024) has demonstrated that adding layers to the upper part of Transformer layers (a process known as "stacking") is particularly effective from an optimization perspective. Specifically, this work shows that stacking

	Learning rate	Learning rate schedule	Batch size	Training tokens	Training steps	FLOPS
368M						
Vanilla	5e-4	cosine	360K	7B	20K	1.63e+19
ReLoRA	5e-4	cosine restarts	360K	13B	40K	1.63e+19
GaLore	1e-2	cosine	360K	7B	20K	1.63e+19
STEP-2stages	5e-4	cosine	360K	11B	33K	1.63e+19
680M						
Vanilla	4e-4	cosine	688K	14B	20K	5.55e+19
ReLoRA	4e-4	cosine restarts	688K	23B	43K	5.55e+19
GaLore	1e-2	cosine	688K	14B	20K	5.55e+19
STEP-2stages	4e-4	cosine	688K	21B	33K	5.55e+19
1.2B						
Vanilla	3e-4	cosine	1179K	24B	20K	1.73e+20
ReLoRA	3e-4	cosine restarts	1179K	43B	43K	1.73e+20
GaLore	1e-2	cosine	1179K	24B	20K	1.73e+20
STEP-2stages	3e-4	cosine	1179K	39B	33K	1.73e+20
STEP-3stages	3e-4	cosine	1179K	53B	43K	1.73e+20

Table 7: Hyperparameters specific to each model setting and method in Table 2. Batch size is specified in tokens.

Configurations	**Selected Value**
Optimizer	AdamW ($\beta_1 = 0.9, \beta_2 = 0.95$)
Learning Rate	0.0001
Learning Rate Schedule	cosine
Warmup steps	100
epoch	2

Table 8: Training configurations in our instruction tuning in Section 4.2.

	Interpolation	**680M**
Vanilla		14.56
STEP-2stages	Copy w/ FPI	14.59
	Copy w/o FPI	14.60
	Mean w/ FPI	14.63
	Mean w/o FPI	**14.56**

Table 9: Validation perplexities for vanilla pre-trained models (Vanilla) and STEPed model (STEP-2stages) using different initialization of the new layer.

behaves more like accelerated gradient descent rather than simple gradient descent, enabling more efficient learning. This finding could potentially provide theoretical support for STEP's strategy of adding layers primarily to the upper portions of the model.[9] Furthermore, empirical observations reported in Chen et al. (2022) indicate that attention patterns learned by BERT models trained from scratch are commonly seen across layers. This insight helps explain why STEP can effectively learn basic attention patterns in its initial stages with a smaller model and then successfully transfer this knowledge to larger models as they grow.

[9]See Appendix I for this strategy.

	schedule timing	**680M**
Vanilla		14.56
STEP-2stages	100%	14.75
	75%	**14.56**
	50%	**14.56**
	25%	14.94

Table 10: Validation perplexities for vanilla pre-trained models (Vanilla) and STEPed model (STEP-2stages) at different schedule timings.

Local low-rank structure and parameter-efficient tuning. The effectiveness of Parameter-Efficient Tuning (PET) methods like LoRA (Hu et al., 2022) and ReLoRA (Lialin et al., 2024), which STEP utilizes, is grounded in the theory of local low-rank structure in neural networks. This theory posits that the updates to the weights of a neural network during training often lie in a low-dimensional subspace. By leveraging this property, PET methods can achieve comparable performance to full fine-tuning while updating only a small number of parameters. In the context of STEP, this background explains how we can maintain high performance while significantly reducing memory requirements. By applying PET to the layers trained in earlier stages, STEP can continue to update these layers efficiently without the need to store full-rank gradients and optimizer states.

Through these discussions, we can better understand why STEP is able to achieve comparable performance to traditional pre-training methods while significantly reducing memory requirements.