A Comparative Survey: Reusing Small Pre-Trained Models for Efficient Large Model Training

Dhroov Pandey+
Department of Computer Science and Engineering
University of North Texas, Denton, TX, USA
dhroovpandey@my.unt.edu

Jonah Ghebremichael
Department of Computer Science
North Carolina State University, Raleigh, NC, USA
jghebre@ncsu.edu

Zongqing Qi
Department of Computer Science
Stevens Institute of Technology, Hoboken, NJ, USA
zqi10@stevens.edu

Tong Shu*
Department of Computer Science and Engineering
University of North Texas, Denton, TX, USA
tong.shu@unt.edu

***Abstract*—Training large language models is becoming increasingly complex due to the rapid expansion in their size, resulting in significant computational costs. To address this challenge, various model growth methodologies have been proposed to leverage smaller pre-trained models to incrementally build larger models and reduce computational requirements. These methods typically involve mapping parameters from small models to large ones using either static functions or learned mappings. Although these approaches have demonstrated effectiveness, there is a lack of comprehensive comparative evaluations in the literature. Additionally, combining different methodologies could potentially yield superior performance. This study provides a uniform evaluation of multiple state-of-the-art model growth techniques and their combinations, revealing that efficient combination techniques can reduce the training cost (in TFLOPs) of individual methods by up to 80%.**

I. Introduction

Large language models (LLMs) have become increasingly popular in the recent years and continue to move towards mass adoption as their capabilities expand and output improves [1]. This continued improvement is powered by increasing their trainable parameters up to billions and scaling up models [22]. Training LLMs from scratch, however, comes with a prohibitively high cost. The larger the model, the longer the training time, which comes with increased computational expenditure and resource consumption. To broaden adoption of LLMs and reduce their training cost, it is important to find more efficient ways to increase their size [20], [25], [27].

A recent development in increasing transformer training efficiency is the adoption of model growth methodologies which use smaller pre-trained models to initialize large models [2], [13], [16], [17], [24]. The large models can be grown by increasing the number of layers (making the model deeper) or by increasing the dimension of the model (making the model wider) through a transformation on the pre-trained model's parameters. The large models initialized via model growth are usually more advanced on their loss curves than randomly initialized models. Hence, they require less training to achieve similar performance, resulting in reduced computational costs. There has been extensive work that has introduced or examined various model growth techniques on transformer architecture [5], [21]. The model growth methodologies can be classified into static and learning-based initialization frameworks [14], [21]. The static methods perform a fixed operation on the pre-trained models parameters, such as duplicating each layer by stacking or interleaving model layers to increase the model depth, whereas the learning-based methods train a meta model that optimizes the initialization of the large model. The training cost of the meta model is usually very low compared to the that of the large model. These methodologies have showed very promising results in transformer model growth, and can be employed to lower compute cost. As a result, large transformers can become accessible in many environments where computational resources can be limited.

However, there is lack of a comprehensive evaluation of existing model growth methodologies with uniform benchmarks, hindering their selection and improvement. In this paper, we compare existing model growth methods and their combinations with the same set of datasets and pre-trained models, and discover more effective model growth methods based on their combinations. Our contribution can be summarized as follows:

- We provide a uniform comparison of existing static and learning-based model growth methods in different experimental settings for multiple use cases.
- We test various combinations of existing model growth methods with extensive experiments and identify the best combination approach, which can reduce training cost by up to 80%.
- We explore model growth methodologies beyond a single pre-trained models towards multiple small models that were pre-trained on disjoint datasets. As far as we know, this is the first work to leverage multiple pre-trained

This research work is done in the Smart High-performance and Ubiquitous Systems (SHU'S) lab at the University of North Texas (UNT) and sponsored by National Science Foundation under Grant No. OAC-2306184 with UNT.

+ Dhroov Pandey is an undergraduate student in Computer Science at UNT.

* Corresponding author: Tong Shu, Assistant Professor, Department of Computer Science and Engineering, University of North Texas, Denton, TX 76207, USA (ORCID: 0000-0001-8617-1772)

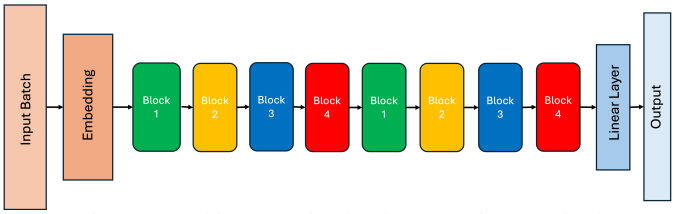


Fig. 1: Stacking: Static depth expansion method

small models to initialize LLMs for efficient training. We compare the model growth methods, typically used for growing models from a single pre-trained model and adapt them for multiple pre-trained models, based on a uniform criteria and benchmarks.

II. Model Growth Techniques

Attempts in model growth of neural networks can be traced back to the 1990s [6]. There are two popular techniques to enlarge the scale of a deep learning model: making a model deeper (depth expansion) and making a model wider, i.e., increasing the dimension of the model (width expansion). These techniques have evolved over time, with recent advancements focusing on maintaining model efficiency while increasing size, particularly in the context of large language models.

While model growth for neural networks has been discussed for decades, only recently have these techniques been applied to deep learning architectures. Chen et al. proposed a form of layer stacking for convolutional neural networks [3]. Gong et al. all observed that the attention patterns of many layer higher layers in trained BERT models were similar to the lower layers [7]. Also, they suggested that there might be some redundancy in the model's architecture, and particularly that knowledge from shallow layers could be transferred to deeper layers. Recently, many new and updated techniques for model growth have been presented to reduce computational costs. However, some contemporary research has posited that while model growth works well initially, it either is even or worse than training from scratch when large amounts of training data [8] are given.

A. Depth-based Static Methods

To make a model deeper, multiple existing methods stack the model blocks in different orders. The basic unit of operation is the transformer blocks.

Stacking [5]: A transformer model has several blocks corresponding to encoder and decoder layers. The stacking approach is a straightforward way of increasing the depth of a model by duplicating model layers and concatenating them to the small model as shown in Fig. 1. This is a very popular methodology for depth expansion used by [5], where it is referred to as *G-Stack*. Du et al. found that G-Stack, performed the best out of various model growth techniques that were evaluated, and demonstrated that G-Stack continue to have benefits even with an abundance of training data. Stacking has also been used for efficiently training BERT-based models by [7], where it is referred to as *StackBERT*.

Interleaving [5]: This method is similar to stacking, except instead of concatenating the duplicated layer after the model, they are interwoven into the structure as shown in Figure

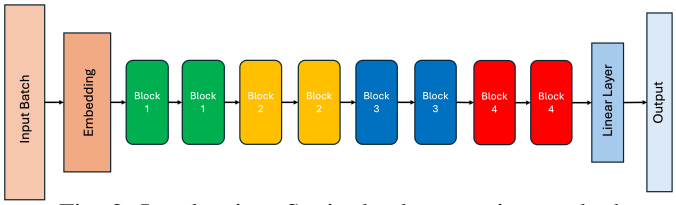
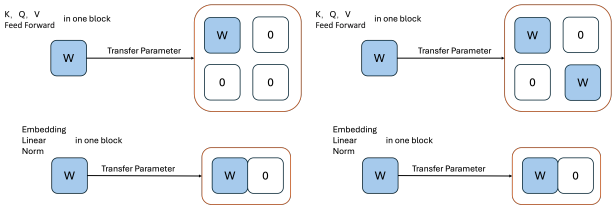
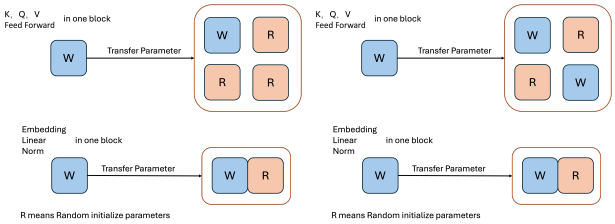


Fig. 2: Interleaving: Static depth expansion method



(a) *G-Zero* (b) *G-Zero v2*

Fig. 3: Static width expansion based on G-Zero



(a) *G-Random* (b) *G-Random v2*

Fig. 4: Static width expansion based on G-Random

Fig. 2. Intuitively, this approach seeks to mimic the effect of increasing the resolution of an image by increasing pixel density.

Identity Injection [5]: This approach refers to growing a model deeper by adding *identity* layers, which output their input, in an interleaved or stacked manner. This preserves the loss function of the smaller model and is able to initialize the large model's training at the intersection of its loss curve with the small model's loss curve.

B. Width-based Static Methods

Different from deeper expansion, the wider expansion changes the shape of the basic block, and extends all parameters matrix and structures to a larger size compared to the small model in one block.

G-Zero [5]: Width expansion technique increases the dimension of the transformer model. The key idea is to copy the parameters of the small model and fill the remaining space with zeros. Consider a $2k \times 2k$ block matrix in the larger model, where k is the block matrix size in the small model, there are two ways to implement this method: i) The block at the top left corner contains the copy of the small model parameters and the rest of the blocks are 0. ii) The matrix is initialized as block diagonal, where the diagonal blocks are copies of small model parameters. Fig. 3 visualizes the two ways of using G-Zero. In this work, we will refer to method (a) as *G-Zero* and method (b) as *G-Zero v2*.

G-Random [5]: Similar to G-Zero, G-Random expands the width of the model by copying the small model parameters. However, instead of filling the remaining space with zeros, G-

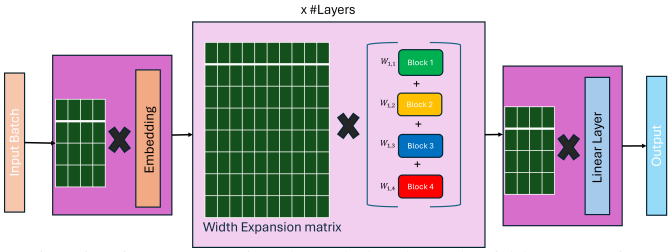


Fig. 5: LiGO: Learning-based depth and width expansion

Random fills it randomly. There are two ways to implement G-Random: i) Block $(0, 0)$ contains the copy of small model parameters, the rest are random. ii) Diagonal blocks contain copies of small model parameters, the rest are random. Fig. 4 shows the two ways of using G-Random. We will refer to method (a) as *G-Random* and method (b) as *G-Random v2*.

C. Learning-Based Methods

Dynamic model growth methodologies have employed an optimization-based approach to learn trainable parameters that act as transformation operators to map small model parameters to the large model's parameter space. The approach seeks to combine the width and depth expansion instead of performing them separately.

LiGO [21]: Wang et al. use depth expanders (i.e., linear combinations of all the model layers) to map l_1 layers in a small model to l_2 layers in a large model and use width expanders to map $d_1 \times d_1$ parameters in a small model to $d_2 \times d_2$ parameters in a large model as shown in Fig. 5. LiGO uses matrix factorization to perform depth and width expansion with $L_{l_2 \times l_1}$ and $R_{l_1 d_2^2 \times l_1 d_1^2}$ matrices, respectively. The matrix R is further factorized as a Kronecker product of $A_{d_2 \times d_1}$ and $B_{d_1 \times d_2}$ to reduce the number of trainable parameters.

MANGO [14]: Pan et al. transform their optimization parameters into a multi-linear operator via a tensor ring matrix product decomposition. This yields four smaller matrices that map the small model parameters onto the large model parameter space.

D. Multi-Model Methods

In machine learning, the practice of leveraging multiple models has a long history and has consistently enhanced model performance. Ensemble learning techniques, such as bagging, boosting, and stacking, utilize the strengths of individual models by combining their predictions [4]. These methods are well-documented for their effectiveness in reducing overfitting and improving predictive accuracy by averaging out the errors of individual models.

While widely used large language models (LLMs) are versatile and generalized, numerous smaller pre-trained models are specifically designed for narrower tasks. For example, customer service bots on retail websites are often trained on particular product catalogs, frequently asked questions, and company policies. This specialization enables them to deliver accurate and contextually relevant responses within their domain, without relying on the extensive knowledge base of a general-purpose LLM.

As LLMs become increasingly sophisticated, the demands for specialized models continue to rise. Companies will need to enhance the capabilities of these specialized models, pushing them toward greater generalization. This is often achieved by expanding the size of the model and its associated dataset. In this context, model growth can be leveraged to incorporate pre-existing models with distinct datasets. By applying the ensemble approach to Transformers, we can utilize the insights gained from each model to minimize the computational resources required for scaling up.

This approach introduces new challenges, as the growth of multiple models necessitates careful consideration of how to harmonize their loss curves. With a single model, there is only one loss curve—the initial smaller model—making it straightforward to identify the optimal point for expansion. In contrast, managing multiple models adds complexity, as the ideal expansion point for one model may negatively impact another. This results in a multidimensional optimization problem, requiring us to navigate the trade-offs between the performance of different models and identify a compromise that maximizes overall results.

III. Datasets

We used three textual datasets to train and evaluate our language models: a collection of Shakespeare's works, WikiText-2, and WikiText-103.

The tiny **Shakespeare** dataset comprises 40,000 lines from the works of William Shakespeare [9]. Using this dataset allows our models to capture the nuances of Early Modern English, understand complex literary devices, and handle diverse text structures. Given the small nature of this dataset, we decided to use character-based tokenization for simplicity.

The **WikiText-2** dataset is a modern English corpus extracted from Wikipedia articles [12]. It consists of nearly 3 million tokens and retains the original text structure, including punctuation and capitalization. WikiText-2 is curated to contain long, coherent pieces of text, providing a context-rich environment for language modeling tasks. This dataset is well-suited for evaluating a model's ability to process modern, well-structured prose and its capacity to generalize across the diverse topics and domains present in Wikipedia.

The **WikiText-103** dataset is an extended version of WikiText-2, containing over 118 million tokens [12]. Like WikiText-2, it is derived from Wikipedia articles, but offers a significantly larger and more diverse corpus. This dataset includes a wide range of topics, making it ideal for training large-scale language models. The size and diversity of WikiText-103 allow for a thorough evaluation of a model's scalability and performance in understanding and generating extensive textual content. For both WikiText-2 and WikiText-103, we opted to use GPT2Tokenizer, a pretrained tokenizer from HuggingFace.

TABLE I: Summary of the datasets

Dataset	Learning rate	Tokenizer	# Tokens
Shakespeare	0.001	Character-based	1115393
WikiText-2	0.0003	GPT2Tokenizer	2922460
WikiText-103	0.0003	GPT2Tokenizer	118450716

Table I summarizes the hyperparameter settings for training initialized large models on each dataset.

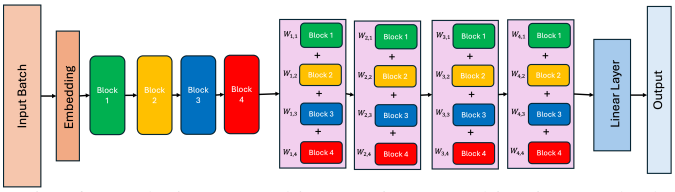


Fig. 6: StackLiGO: Stacking + LiGO combination method

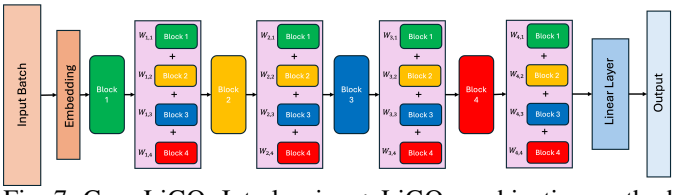


Fig. 7: CrossLiGO: Interleaving + LiGO combination method

IV. Benchmark Models

We benchmarked some of the model growth methodologies (in §II) in a uniform setting to fairly compare their performance on diverse use cases. In addition, we tested various models generated by combining those methodologies to evaluate the best possible permutations of width and depth expansion in conjunction with static and learning-based expansion.

A. Combination Methods for Model Growth

We combine the model growth techniques with the following methods to make models deeper, wider, or both.

StackLiGO is the model combined from the LiGO method with the Double stack. As shown in Fig. 6, the first section of the model is copied from the small model, and the second part of the model uses the LiGO combined block instead of the small model.

For **CrossLiGO**, shown in Fig. 7, we use techniques similar to interleaving. We stack the blocks one by one instead of the entire small model.

The **G-Zero + Stacking** is the double-stacking version of the wide model. By using the wide extension method, we obtained the wide block of the wide model. Then, we stack these blocks by following the Stacking strategy shown in Fig. 1.

Similar to G-Zero + Stacking, **G-Zero + Interleaving** is the wide model version of Interleaving. Using the wide blocks constructed by the G-zero method, we stack the wide blocks as the Interleaving shown in Fig. 2.

B. Growth from a Single Model

We use a decoder-only transformer architecture for our benchmarks. We summarize the model architectures in our use cases below.

1) Small Pre-Trained Model: Table II summarizes the model configurations for the small pre-trained model.

TABLE II: Pre-trained model configuration

Dataset	#Layers	Model dimension
Shakespeare	4	32
WikiText-2	4	128
WikiText-103	4	128

2) Deeper Large Model: Table III summarizes the configurations for the depth-expanded large model.

TABLE III: Depth expansion model configuration

Dataset	#Layers	Model dimension
Shakespeare	8	32
WikiText-2	8	128
WikiText-103	8	128

3) Deeper and Wider Large Model: Table IV summarizes the model configurations for large model expanded by depth and width.

TABLE IV: Depth and width expansion model configuration

Dataset	#Layers	Model dimension
Shakespeare	8	64
WikiText-2	8	256
WikiText-103	8	256

C. Combination of Multiple Models

1) Small Pre-Trained Models: Table V summarizes the model configurations for the small pre-trained models used for model growth.

TABLE V: Pre-trained models configuration

Dataset	#Layers	Model dimension
WikiText-2	4	128
WikiText-103	4	128

2) Deeper Large Model: Table VI summarizes the configurations for the depth-expanded large model.

TABLE VI: Depth expansion multi-model configuration

Dataset	#Layers	#Models	Model dimension
WikiText-2	8	2	128
WikiText-103	8	2	128
WikiText-103	12	3	128
WikiText-103	16	4	128

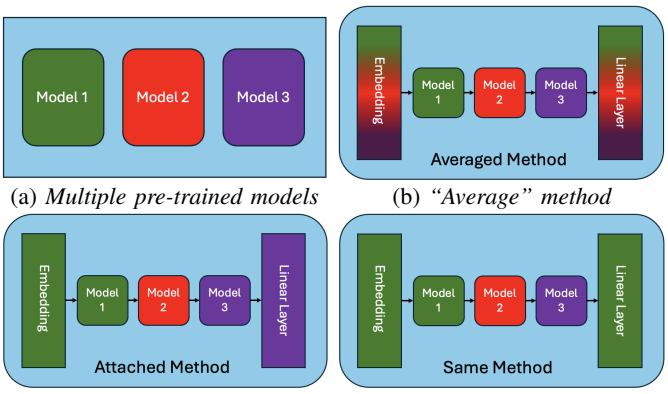
V. Evaluation

A. Experimental Setting

1) Single Pre-trained Model: We used a single small pre-trained model as the base to grow larger models using various techniques discussed above and compare them based on their computational efficiency and prediction loss. We evaluate two use cases: i) a larger model expanded by depth and width and ii) a larger model expanded by just depth. We omit the width-based expansion case since we empirically observed that width expansion does not save us much compute, similar to the observations in [5]. The learning-based growth methods are trained for 10 epochs to optimize initialization of large models. We use the same hyperparameters for each initialized large model being trained on a dataset given in Table I.

2) Multiple Pre-trained Models: We modify the techniques discussed above to initialize a large model based on the parameters of multiple small models. Like our experiments based on a single pre-trained model, we exclude width-based expansion due to its limited computational efficiency gains. For learning-based methods, the modifications are straightforward and intuitive: we use the parameters of all the pre-trained models as input for the learned initializer. For static methods, we incorporate randomization and normalization to select the parameters in the following two ways:

Stacking: We randomly choose the order in which the models are stacked, and used three different ways to select the embedding and the final dense layer (shown in Fig. 8): i) The "average" method uses the mean of the embedding and



(a) *Multiple pre-trained models* (b) *"Average" method*

(c) *"Attached" method* (d) *"Same" method*

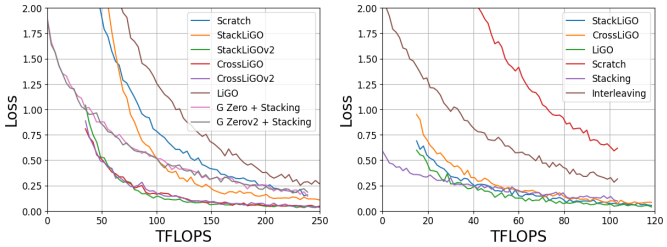
Fig. 8: Comparison of Multi-Model Approaches

final dense layer parameters for each model to initialize the large model (in Fig. 8b). ii) The "attached" method uses the embedding from the model stacked first and the final linear layer from the model stacked last (in Fig. 8c). iii) The "Same" method uses the embedding and the final linear layers from the same model chosen randomly in the stack (in Fig. 8d).

Interleaving: We choose a random order in which the model layers are interleaved. The order remains consistent for all the layers. The embedding and final dense layer are initialized in the same manner as the Stacked model.

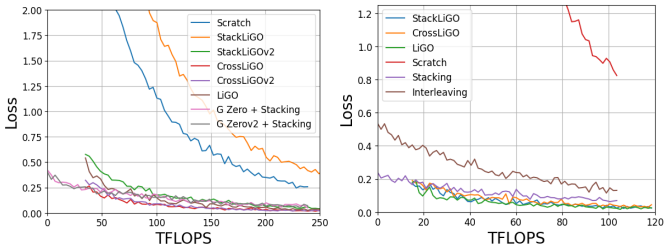
B. Growth from a Single Model

We evaluated model growth from a single pre-trained model by measuring the computational cost of training the grown models and the scratch model. Then, we plotted the validation loss of the model against the training cost (in TFLOPS) in Figs. 9, 10, and 11. We observe that combination of techniques can outperform the original methodologies in some use cases. StackLiGOv2 saves the most computational cost in Wiki-2 depth+width expansion, closely followed by CrossLiGO and CrossLiGOv2 as shown in Fig. 9a. However, for the depth-only expansion, we see in Fig. 9b that LiGO is most effective, although it's performance is very similar to CrossLiGO and StackLiGO. This can be due to the fact that in learning-based depth expansion, the number of trainable parameters is small for the initial optimization. Hence, LiGO can achieve greater optimization due to having more trainable parameters compared to StackLiGO and CrossLiGO, which have half of their layers set to static. Fig. 10 illustrates that for the largest dataset, Wiki-103, in our benchmark, CrossLiGO demonstrates the most efficient model growth in both depth and width expansions and LiGO continues to excel as the best initializer in depth-only expansions, consistent with the findings from the Wiki-2 evaluation. Fig. 11 shows that learning-based methods underperform on the Shakespeare dataset in both depth-plus-width and depth-only expansion scenarios. In the depth-plus-width expansion, the scratch model emerges as the top performer. However, for depth-only expansion, stacking and interleaving methods incur the lowest computational cost. Therefore, we can conclude that depth expansion is the most effective way to grow a model for smaller datasets and width expansion can be introduced as the dataset size becomes sufficiently large. Furthermore, we notice that static methods are optimal for expanding models trained on smaller datasets, while learning-based methodologies begin to dominate as data complexity increases.



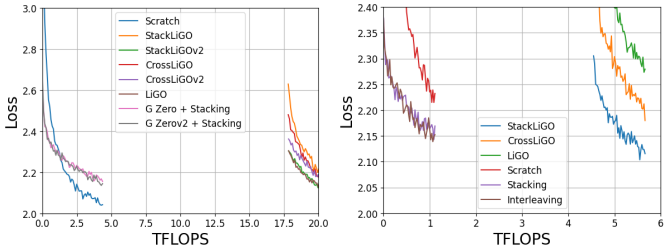
(a) *Width plus Depth expansion* (b) *Depth-only expansion*

Fig. 9: A single pre-trained model over Wiki-2



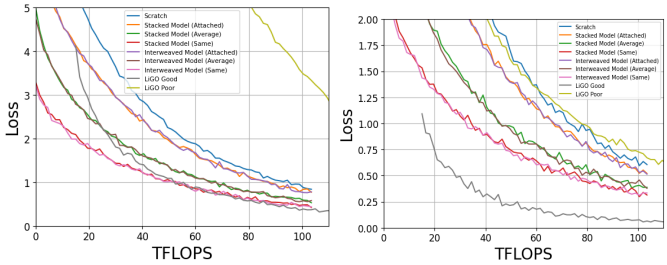
(a) *Width plus Depth expansion* (b) *Depth-only expansion*

Fig. 10: A single pre-trained model over Wiki-103



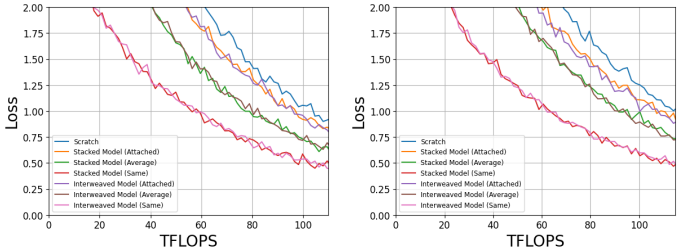
(a) *Width plus Depth expansion* (b) *Depth-only expansion*

Fig. 11: A single pre-trained model over Shakespeare



(a) *Wiki-2* (b) *Wiki-103*

Fig. 12: Two pre-trained models



(a) *Three pre-trained models* (b) *Four pre-trained models*

Fig. 13: Multiple pre-trained models over Wiki-103

C. Combination of Multiple Models

We found that all techniques can improve training from scratch. The techniques of Stacking and Interleaving both yielded advantages, yet the main difference in the results seems to relate to how closely the embedding and final linear layers fit onto the stacked models. The "attached" method which set the embedding layer to the first model and the final linear layer to the last model was conceptually appealing but yielded the worst results. In fact, this was nearly identical to the scratch approach. Averaging the combination of the parameters from each layer was an improvement, but the best performance came from the "same" method which simply duplicating both the embedding and final linear layer from any one model.

From Fig. 12, we can see that constructing the embedding and final linear layers with the "average" method tends to converge faster alongside the "same" method. However, Fig. 13 shows that when increasing the number of different small models, the performance of the "average" method in relation to the multi-model "same" method (in Fig. 8d) becomes worse. The results indicate that the more pre-trained small models are used, the less the averages of layers aligned with the blocks.

By reusing the boundary layers (i.e., embedding and final linear layers) from any one model, the "same" method produces the best results of any multi-model method. This indicates that harmonizing boundary layers provides significant initial benefits, resulting in lower loss. Even though multiple smaller models are incorporated into the intermediate layers, matching boundary layers from a single model outperforms averaging or attaching them from different models. This performance is likely to stem from the inherent alignment between boundary layers when they come from the same model.

Note that, as a learning-based method, LiGO performed very inconsistently. Fig. 12 displays two performance of the two LiGO initialized models: one is good, but the other is poor. If LiGO is able to get a good initialization, it outperforms all the static methods by a substantive margin, whereas if the initialization is poor, the large model is very computation-expensive. Therefore, due to its performance inconsistency, we consider the methodology unreliable for initializing a large model from multiple pre-trained models.

Our results indicate that each smaller model contains a specific knowledge base extracted from its subset of the data. Utilizing these small models to train a more sophisticated large model with model growth techniques can consume lower computation costs. However, it is important to choose the correct combination method for integrating these smaller models into a larger one. The "same" method, which reuses both embedding and final liner layers from a single model, appears to provide the best starting point for further training, likely due to the inherent alignment between these layers.

VI. Discussion

We omit the evaluation of combination techniques that utilize G-Random, because we empirically observed that the initialized models were often getting stuck in a local minima during training. The same thing happened to LiGO-based model in the case of multiple pre-trained models. For two pre-trained models, the initializations were inconsistent, so the model might perform extremely well, moderately, or very poor. With the number of pre-trained models increasing, LiGO degrades performance. Specifically, the large model initialized by LiGO would perform poorly or get stuck in a local minima more possibly. This might be because finding the optimal jump point (from small model parameter to large model parameter) across multiple loss curves is far more complex than for a single loss curve when there is only one pre-trained model.

While we discovered various conclusive results by benchmarking several model growth methodologies and their combinations, our work was limited to a decoder-based transformer model. We leave incorporating an encoder-based cross attention architecture to future work. Further improvements can be made to model growth evaluation by including model tuning and its associated costs. Lastly, while our evaluation utilized multiple static methods, we categorized learning-based techniques under one umbrella and only benchmarked the LiGO framework under that category to limit the combinatorial space of method exploration. An evaluation involving other learning-based initialization frameworks such as [14] can provide a more comprehensive comparison of model growth operators. Also, we will validate this technique in training broader deep learning applications [10], [11], [15], [18], [19], [23], [26].

For our evaluation of model growth from multiple pre-trained models, we had to generate the pre-trained models by partitioning each dataset and training small models on the disjoint training data. However, this approach suffers from the data uniformity issue, which causes the pre-trained models learning from completely disjoint datasets to have common knowledge, rather than become separate specialists. This can be improved by using different datasets under a common domain to generate pre-trained models and growing a large model from those models that learns from the union of the pre-trained models' datasets.

VII. Conclusion

In this work, we compared several existing model growth methodologies to generate a large model from a single pre-trained model and from multiple pre-trained models. We also evaluated combinations of existing methodologies for a comprehensive review regarding how these methods can interact with one another. We discovered that learning-based methodologies are more powerful than static ones to save compute in the case of a single pre-trained small model. However, the learning-based methods performed poorly when the number of pre-trained models increases. Also, we found that methodologies that use randomization to initialize the large model's parameters were less efficient, because the initial loss might be close to a local minima so that the model would either get stuck or the descent would be very slow. In addition, static methods that do not employ random parameters for the large model initialization have more consistent performance. Finally, we conclude that model growth methodologies are an effective tool to enhance Transformer training efficiency.

ACKNOWLEDGMENT

This research is sponsored by National Science Foundation under Grant No. OAC-2306184 with the University of North Texas.

REFERENCES

[1] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P. S. Yu, Q. Yang, and X. Xie. A survey on evaluation of large language models. ACM Transactions on Intelligent Systems and Technology (TIST), 15(3), Mar 2024.

[2] C. Chen, Y. Yin, L. Shang, X. Jiang, Y. Qin, F. Wang, Z. Wang, X. Chen, Z. Liu, and Q. Liu. bert2BERT: Towards reusable pretrained language models. In Annual Meeting of the Association for Computational Linguistics (ACL), volume 1, pages 2134–2148, Dublin, Ireland, May 2022.

[3] T. Chen, I. J. Goodfellow, and J. Shlens. Net2Net: Accelerating learning via knowledge transfer, 2016. https://arxiv.org/pdf/1511.05641.

[4] T. G. Dietterich. Ensemble methods in machine learning. In Proc. of the First Intl. Workshop on Multiple Classifier Systems (MCS), pages 1–15, Cagliari, Italy, Jun 2000.

[5] W. Du, T. Luo, Z. Qiu, Z. Huang, Y. Shen, R. Cheng, Y. Guo, and J. Fu. Stacking your transformers: A closer look at model growth for efficient LLM pre-training, 2024. https://arxiv.org/pdf/2405.15319.

[6] S. Fahlman. The recurrent cascade-correlation architecture. In Proc. of Conf. on Neural Information Processing Systems (NeurIPS), volume 3, pages 190–196, Denver, CO, USA, 1990.

[7] L. Gong, D. He, Z. Li, T. Qin, L. Wang, and T. Liu. Efficient training of BERT by progressively stacking. In Proc. of Intl. Conf. on Machine Learning (ICML), volume 97, pages 2337–2346, Long Beach, CA, USA, Jun 2019.

[8] J. Kaddour, O. Key, P. Nawrot, P. Minervini, and M. J. Kusner. No train no gain: Revisiting efficient training algorithms for transformer-based language models. In Proc. of Conf. on Neural Information Processing Systems (NeurIPS), volume 36, pages 25793–25818, New Orleans, LA, USA, Dec 2023.

[9] A. Karpathy. Tinyshakespear. https://raw.githubusercontent.com/karpathy/char-rnn/master/data/tinyshakespeare/input.txt.

[10] T. Kundu and T. Shu. HIOS: Hierarchical inter-operator scheduler for real-time inference of DAG-structured deep learning models on multiple GPUs. In Proc. of the 25th IEEE International Conference on Cluster Computing (Cluster), pages 95–106, Santa Fe, NM, USA, Nov 2023.

[11] Y. Li, J. Baik, M. M. Rahman, I. Anagnostopoulos, R. Li, and T. Shu. Pareto optimization of cnn models via hardware-aware neural architecture search for drainage crossing classification on resource-limited devices. In Proc. of Workshops of ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC-W), pages 1767–1775, Denver, CO, USA, Nov 2023.

[12] S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models. In Proc. of Intl. Conf. on Learning Representations (ICLR), Toulon, France, Apr 2017.

[13] Y. Pan, Y. Yuan, Y. Yin, J. Shi, Z. Xu, M. Zhang, L. Shang, X. Jiang, and Q. Liu. Preparing lessons for progressive training on language models. In Proc. of AAAI Conf. on Artificial Intelligence (AAAI), volume 38, pages 18860–18868, Vancouver, British Columbia, Canada, Feb 2024.

[14] Y. Pan, Y. Yuan, Y. Yin, Z. Xu, L. Shang, X. Jiang, and Q. Liu. Reusing pretrained models by multi-linear operators for efficient training. In Proc. of Conf. on Neural Information Processing Systems (NeurIPS), pages 3248–3262, New Orleans, LA, USA, Dec 2023.

[15] D. Pandey and T. Shu. AM-DGCNN: Leveraging graph attention networks and edge attributes for link classification in knowledge graphs. In Proc. of Workshops of IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC-W), Atlanta, GA, USA, Nov 2024.

[16] S. J. Reddi, S. Miryoosefi, S. Karp, S. Krishnan, S. Kale, S. Kim, and S. Kumar. Efficient training of language models using few-shot learning. In Proc. of Intl. Conf. on Machine Learning (ICML), pages 14553–14568, Honolulu, Hawaii, USA, Jul 2023.

[17] S. Shen, PeteWalsh, K. Keutzer, J. Dodge, M. Peters, and I. Beltagy. Staged training for transformer language models. In Proc. of Intl. Conf. on Machine Learning (ICML), volume 162, pages 19893–19908, Baltimore, Maryland, USA, Jul 2022.

[18] T. Shu, Y. Guo, J. Wozniak, X. Ding, I. Foster, and T. Kurc. Bootstrapping in-situ workflow auto-tuning via combining performance models of component applications. In Proc. of ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC), pages 1–15, St. Louis, MO, USA, Nov 2021.

[19] T. Shu, Y. Guo, J. Wozniak, X. Ding, I. Foster, and T. Kurc. POSTER: In-situ workflow auto-tuning through combining component models. In Proc. of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), pages 467–468, Seoul, South Korea, Feb-Mar 2021.

[20] D. So, W. Manke, H. Liu, Z. Dai, N. Shazeer, and Q. V. Le. Searching for efficient transformers for language modeling. In Proc. of Conf. on Neural Information Processing Systems (NeurIPS), volume 34, pages 6010–6022, Virtual, Dec 2021.

[21] P. Wang, R. Panda, L. T. Hennigen, P. Greengard, L. Karlinsky, R. Feris, D. D. Cox, Z. Wang, and Y. Kim. Learning to grow pretrained models for efficient transformer training. In Proc. of Intl. Conf. on Learning Representations (ICLR), Kigali, Rwanda, May 2023.

[22] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus. Emergent abilities of large language models. Transactions on Machine Learning Research, 2022.

[23] J. M. Wozniak, P. Davis, T. Shu, J. Ozik, N. Collier, I. Foster, T. Brettin, and R. Stevens. Scaling deep learning for cancer with advanced workflow storage integration. In Proc. of the 4th Workshop on Machine Learning in HPC Environments (MLHPC) in conjunction with ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pages 114–123, Dallas, TX, USA, Nov 2018.

[24] H. Wu, W. Wang, T. Malepathirana, D. Senanayake, D. Oetomo, and S. Halgamuge. When to grow? a fitting risk-aware policy for layer growing in deep neural networks. In Proc. of AAAI Conf. on Artificial Intelligence (AAAI), volume 38, pages 5994–6002, Vancouver, British Columbia, Canada, Feb 2024.

[25] M. Zhang and Y. He. Accelerating training of transformer-based language models with progressive layer dropping. In Proc. of Conf. on Neural Information Processing Systems (NeurIPS), volume 33, pages 14011–14023, Virtual, Dec 2020.

[26] Y. Zhang, D. Pandey, D. Wu, T. Kundu, R. Li, and T. Shu. Accuracy-constrained efficiency optimization and GPU profiling of CNN inference for detecting drainage crossing locations. In Proc. of Workshops of ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC-W), pages 1780–1788, Denver, CO, USA, Nov 2023.

[27] B. Zhuang, J. Liu, Z. Pan, H. He, Y. Weng, and C. Shen. A survey on efficient training of transformers. In Proc. of the 32th International Joint Conference on Artificial Intelligence (IJCAI), pages 6823–6831, Macao, S.A.R, Aug 2023.

ARTIFACT DESCRIPTION/EVALUATION APPENDIX

A. Summary of the Experiments Reported

1) Abstract: We provide this artifact description to make our results more reproducible. We measure two main results: i) The model loss through training, ii) The compute cost of training in FLOPS.

2) Artifacts: source code GitHub Link
git clone https://github.com/SHUs-Lab/AI4S24DP.git

B. Experimental Setup

1) Relevant Hardware Details:

- GPU: NVIDIA A40
- GPU Memory: 48 GB

2) Operating Systems and Versions:

- Operating System: Rocky Linux
- Version: 9.4 (Blue Onyx)

3) Compilers and Versions:

- Compiler: Python 3.12

4) Libraries and Versions:

- Pytorch: 2.3.0+cu121
- Pandas: 2.2.2
- Transformers: 4.41.0
- Calflops: 0.3.2
- Datasets: 2.20.0

5) Input Datasets and Versions:

- Shakespeare: Downloaded from https://raw.githubusercontent.com/karpathy/charrnn/master/data/tinyshakespeare/input.txt
- Wikitext-2: Downloaded from huggingface https://huggingface.co/datasets/Salesforce/wikitext
- Wikitext-103: Downloaded from huggingface https://huggingface.co/datasets/Salesforce/wikitext

6) Other Installation Software: CUDA 12.1

C. Evaluation Experiments

For all experiments, install necessary python packages listed in Appendix B4 'Libraries' using 'pip'.

1) Single Pre-trained Model: The following shows how to get the benchmarks for static and dynamic model growth techniques.

Dynamic growth

a) To benchmark the Shakespeare dataset, use the file 'LiGO_Shakespeare.ipynb' in the directory 'Dynamic-ModelGrowth'.

b) To benchmark the Wiki-2 and Wiki-103 datasets, use the file 'LiGO_Wiki.ipynb' in the directory 'Dynamic-ModelGrowth'. To choose the dataset, modify block 3 by toggle commenting lines 2 and 5.

Static Growth

a) To benchmark the Shakespeare dataset, run the file 'WideAndDeep_Shakespeare.ipynb' in directory 'Static-ModelGrowth'.

b) To benchmark the Wiki-2 and Wiki-103 datasets, run the file 'WideAndDeep_Wiki.ipynb' in directory 'Static-ModelGrowth'. To choose the dataset, modify block 3 by toggle commenting lines 2 and 5.

2) Multiple Pre-trained Models: Do the following steps for model growth evaluation from multiple pre-trained models.

a) Files are located in 'Multi-Model' folder.

b) Choose dataset using the corresponding load_dataset() function call in both 'LiGO-MultiModel.ipynb' and 'Stacked-MultiModel.ipynb'.

c) Run the scripts 'LiGO-MultiModel.ipynb' and 'Stacked-MultiModel.ipynb' to generate performance results.

3) Calculating FLOPS: Run the following code at the end of any of the scripts to calculate the FLOPs associated with a particular model dimension.

```
sample = None
for i in train_dataloader:
    sample = i['input_ids']
    break
calculate_flops(model=Model(model_dim)
    .to(device), kwargs={'inp':sample})
```