Growing Language Models (GLaM): Methods for Model Growth and Their Consequences

Eyas Ayesh
School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, GA 30332
eayesh3@gatech.edu

Shrenik Bhansali
School of Computer Science
Georgia Institute of Technology
Atlanta, GA 30332
sbhansali8@gatech.edu

Mohamed Ghanem
School of Computer Science
Georgia Institute of Technology
Atlanta, GA 30332
mghanem8@gatech.edu

Faris Qadan
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332
fqadan3@gatech.edu

Abstract

Training large language models (LLMs) is resource-intensive and often inaccessible to smaller research groups or organizations. One promising solution is model growth, a technique that incrementally expands a pre-trained model's architecture to achieve larger-scale capacity while reusing previously learned knowledge. Although model growth has been explored for encoder-based architectures, its application to decoder-only LLMs remains unexplored. In this work, we introduce novel growth operators and systematically evaluate their effects on training efficiency and downstream task performance in decoder-only models. Starting from a small pre-trained Pythia model, we apply both unidimensional (depth, attention heads, feed-forward expansion) and multidimensional (combined) growth operators. Our results show that unidimensional operators can significantly accelerate training while maintaining or improving performance. Additionally, although multidimensional operators come with a modest computational overhead, they yield more robust models that excel in zero-shot question-answering evaluations. These findings highlight the potential of model growth as a scalable and accessible strategy for improving LLMs, and suggest future directions for fine-tuning growth hyperparameters and evaluating multidimensional strategies at larger scales.

1 Introduction

Training large language models (LLMs) has become increasingly resource-intensive (Brown et al., 2020), posing challenges for research institutions and smaller companies aiming to develop or customize their own models. These resource demands also raise concerns about the environmental impact of training such models (Wu et al., 2022) (Hoffmann et al., 2022).

Model growth, a technique where larger models are initialized using the weights of smaller pre-trained models, offers a promising approach to reduce the computational and environmental costs of training. These techniques have shown promise (and success!) in speeding up BERT model training (Gong et al., 2019; Chen et al., 2021a), but their extension to decoder-only LLMs is remains significantly less explored.

A core component of model growth strategies is the use of growth operators, which define how a smaller model is expanded into a larger one while preserving or adapting its learned knowledge. In the context of LLMs, growth operators may involve adding layers, increasing embedding dimensions, or incorporating additional attention heads. These operators ensure that knowledge is effectively transferred from the original model to the expanded version, facilitating seamless scaling without compromising performance.

Our work advances this line of research by introducing novel growth operators and empirically evaluating their impact on model performance. Specifically, we assess how these operators influence training efficiency and downstream task performance in decoder-only models. By starting with smaller pre-trained decoder-only models and expanding them, our approach aims to balance reduced training time with maintained or enhanced model performance during growth.

The potential utility of model growth is clear; one such example of comes in the form of enhanc-

ing fine-tuning (Chu et al., 2024b) by addressing the limitations of fixed-capacity models in adapting to complex or resource-intensive tasks. Incrementally expanding a model's architecture through growth operators increases capacity while preserving pre-trained knowledge, enabling better task-specific adaptation. This approach is particularly valuable for multitask fine-tuning or low-resource settings, where selective expansions reduce computational overhead and mitigate overfitting. By integrating model growth into the fine-tuning (Chu et al., 2024b) process, models can dynamically scale to task requirements, improving performance while optimizing resource efficiency.

While promising results have been demonstrated in previous work (Du et al., 2024), advancing model growth requires further investigation into the mechanisms that enable effective knowledge transfer, as well as frameworks for optimizing growth strategies. By addressing these challenges, our work contributes to the development of scalable and efficient methods for training LLMs, enabling broader accessibility to advanced AI technologies.

2 Prior Related Work

The field of model growth and growth operators has its origins in techniques designed to improve the scalability and efficiency of deep learning models as they increase in size and complexity. Early works on dynamic model adjustment, such as structured pruning and architecture search, provided the foundation for more formalized approaches to model growth. Han et al. (Han et al., 2015) introduced structured pruning as a way to remove unimportant weights from neural networks, demonstrating that models could retain comparable performance with significantly reduced parameters. This established the principle that models could be dynamically adjusted without requiring retraining from scratch.

Expanding on this foundation, Liu et al. (2018) proposed progressive neural architecture search (PNAS), which incrementally grows models by adding layers or neurons during training. This approach demonstrated improved resource utilization, as models grew only to meet task-specific complexity, avoiding overparameterization. Similarly, Chen et al. (Chen et al., 2021a) showed that progressive expansion strategies in BERT-based models led to faster convergence rates and better downstream performance compared to training large models from scratch.

In the context of large-scale language models (LLMs) (Chowdhery et al., 2022; Zhang et al., 2022), growth operators have shown particular promise. Du et al. (2024) demonstrated that progressive growth could preserve knowledge from smaller pre-trained models while allowing for efficient scaling, achieving competitive performance on natural language understanding tasks with reduced computational costs. Growth operators, such as adding attention heads or extending embedding dimensions, have proven effective for aligning structural adaptations with the representations learned during pre-training, enabling efficient transfer of knowledge.

These findings are particularly relevant to fine-tuning, where pre-trained LLMs are adapted to task-specific domains or datasets. Howard and Ruder (Howard and Ruder, 2018) demonstrated the power of fine-tuning (Chu et al., 2024b) for achieving state-of-the-art performance in various NLP tasks with minimal additional training. Recent advancements have combined fine-tuning with structured growth techniques to enhance task-specific adaptations while retaining general knowledge. Chu et al. (Chu et al., 2024a) explored how growth operators can be integrated with instruction fine-tuning to improve downstream performance while maintaining computational efficiency. Techniques such as LoRA (Low-Rank Adaptation) (Hu et al., 2022) and adapter-based fine-tuning (Houlsby et al., 2019) have further highlighted the synergy between model growth and parameter-efficient fine-tuning, demonstrating significant improvements in resource-constrained scenarios.

The utility of model growth extends beyond computational efficiency to enhancing the scalability and flexibility of LLMs. By incrementally increasing capacity, growth operators allow models to adapt to more complex tasks while preserving performance on prior tasks. For instance, adapter-based approaches have shown that introducing small modular expansions can optimize performance in multitask learning without retraining the entire model (Pfeiffer et al., 2021). Additionally, progressive growth has been demonstrated to reduce overfitting in low-resource settings by controlling model capacity during adaptation, as shown in recent multitask studies (Chronopoulou et al., 2023).

While most prior work has focused on encoder-

based architectures, applying model growth to decoder-only LLMs introduces unique challenges, including handling sequential token dependencies and causal attention mechanisms. Touvron et al. (Touvron et al., 2023) highlighted the complexity of extending growth operators to autoregressive models and emphasized the need for novel techniques to overcome these architectural constraints. Our work builds on these foundations by developing and evaluating growth operators specifically designed for decoder-only LLMs, addressing the challenges of sequential processing and ensuring efficient adaptation to downstream tasks.

3 Model

Our work primarily utilized one baseline model from the Pythia suite: Pythia-14M (Biderman et al., 2023). This model is an autoregressive decoder-only architecture designed for large language modeling tasks, leveraging Transformer-based architectures for sequential text generation (Biderman et al., 2023). The model serves as a foundational baseline for exploring the impact of model growth techniques, where we modify its architecture using unidimensional and multidimensional growth operators.

3.1 Baseline Architecture

Pythia-14M is a lightweight Transformer-based model with roughly 14 million trainable parameters. Its architecture consists of $n = 6$ Transformer layers, each equipped with self-attention and feed-forward sublayers. The attention mechanism employs $h = 4$ attention heads, and the model's hidden dimension d is set to 512. The feed-forward network (ff) within each Transformer block is configured with an inner layer dimension $4d = 2048$, adhering to standard Transformer scaling rules. RoPE (Su et al., 2021) positional encodings are incorporated to capture sequence order, enabling the model to process sequential data effectively.

3.2 Model Growth Strategies

Through model growth, we modify this baseline architecture using two primary strategies: unidimensional growth and multidimensional growth, inspired by prior work on model stacking and progressive growth techniques (Du et al., 2024; Chen et al., 2021a). As explained below, we devise novel combinations of these growth operators, comparing them to (a) the lone pretrained model, and (b) the growing configurations proposed by the authors of LLM stacking (Du et al., 2024). We formalize both strategies as follows.

3.2.1 Unidimensional Growth.

Unidimensional growth involves altering a single architectural component while keeping all other dimensions fixed, allowing us to systematically evaluate the contribution of each modification. This includes adding depth by introducing additional Transformer layers (L), increasing the number of attention heads (h), expanding the hidden dimension (d). Each operation enhances specific aspects of the model's capacity, such as hierarchical representation learning, pattern recognition, or dependency resolution.

Depth Expansion Adding layers increases the depth of the Transformer architecture, enabling the model to learn deeper hierarchical features. Let $\mathcal{L}_{\text{new}} = \mathcal{L}_{\text{base}} + k$, where $\mathcal{L}_{\text{new}}$ represents the new number of layers, $\mathcal{L}_{\text{base}}$ is the original number of layers, and k is the number of layers added.

We will use two depth-wise growth operators from Du et al., $G^{\uparrow}_{\text{stack}}$ and $G^{\uparrow}_{\text{rand}}$.

$G^{\uparrow}_{\text{stack}}$ refers to initializing the new layers as copies of the previous, layers, whereas $G^{\uparrow}_{\text{rand}}$ refers to initializing the new layers as random values according to a Gaussian distribution, which can be formalized as the following:

$$W_{\text{new}} \sim \mathcal{N}(0, \frac{1}{d})$$

where W_{new} are the new weights, and d is the hidden dimension. Progressive stacking has been shown to shift the compute-performance curve by reducing the effective compute required to reach a given performance level (Chen et al., 2021b; Fang et al., 2022), described mathematically as:

$$C' = \alpha \cdot C^{\beta}$$

where C is the compute, C' is the effective compute with growth, and α, β are scaling coefficients (Du et al., 2024).

Attention Head Expansion Expanding the number of attention heads (h) increases the model's ability to capture finer-grained dependencies. Formally, each attention head computes a score where Q, K, V are the query, key, and value matrices, and $d_h = \frac{d}{h}$ is the dimension of each head. Increasing

h reduces d_h, potentially improving the granularity of attention computation. We will refer to this operator as $G_{\textbf{attn}}$.

Feed Forward Layer Expansion Expanding the hidden dimension (d) increases the representational capacity of each Transformer layer. For a feed-forward network (ff) with dimension $4d$, the computational complexity grows quadratically with d, necessitating careful initialization. Using Xavier initialization (Glorot and Bengio, 2010), the weights of the expanded ff are sampled as:

$$W_{\text{ff}} \sim \mathcal{U}\left(-\sqrt{\frac{6}{d_{\text{in}} + d_{\text{out}}}}, \sqrt{\frac{6}{d_{\text{in}} + d_{\text{out}}}}\right)$$

where d_{in} and d_{out} are the input and output dimensions, respectively. We will refer to this operator as $G_{\textbf{ff}}$.

Each unidimensional modification is evaluated independently, allowing us to observe its isolated impact on model performance, convergence, and training efficiency.

3.2.2 Multidimensional Growth.

Multidimensional growth involves simultaneously modifying multiple architectural components, allowing us to explore interactions and potential synergies between growth dimensions. For example, combining depth expansion ($\mathcal{L}$) with increased attention heads (h) can enhance both hierarchical representation learning and fine-grained dependency resolution.

Combined Growth Operators Let $g_1(\cdot)$ and $g_2(\cdot)$ represent two growth operators applied to components x_1 and x_2 of the model. The total growth operation can be expressed as:

$$G(x_1, x_2) = g_1(x_1) + g_2(x_2) + \epsilon$$

where ϵ accounts for potential interactions between the components. Empirical evidence from BERT stacking (Chen et al., 2021a) demonstrates that simultaneously increasing $\mathcal{L}$ and the FFN dimension $4d$ can yield synergistic improvements, provided the new parameters are initialized correctly to maintain gradient stability.

Initialization for Stability To ensure stability, new parameters are initialized to match the scale of pre-trained weights. Following the principles of progressive growth (Du et al., 2024), the initialization of added parameters W_{new} aligns with the distribution of existing weights, minimizing disruption to the pre-trained model and ensuring smooth integration of new components.

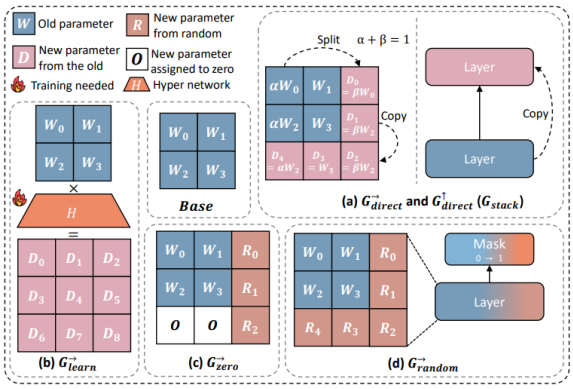

Figure 1: Simplified illustration of four growth operators (G_{direct}, G_{learn}, G_{zero}, and G_{random}) applied to different LLM layers (Du et al., 2024).

3.3 Implementation Details

Each growth operation is applied incrementally to preserve knowledge from the pre-trained baselines. New layers or attention heads are initialized using scaled Gaussian initialization, ensuring compatibility with the existing model parameters (Du et al., 2024). When modifying FFN dimensions, we maintain the same activation functions and dropout configurations as the baseline model, avoiding inconsistencies in forward and backward propagation. The token embeddings and positional encodings are extended when necessary to accommodate expanded model architectures, ensuring that pre-trained knowledge aligns with new capacity.

We note that the application of these operators is a trivial task; it serves as an intermediate step between two training stages for the initially small model. Upon application of each growth operator (or combination of growth operators), a secondary training stage is performed on the now-grown model, before commencing with evaluation.

4 Data

Throughout pre-training all models, we utilized the SlimPajama-627B dataset (Smith et al., 2024). The HuggingFace streaming framework (Wolf et al., 2020) was employed to efficiently handle the entire dataset, including its training, test, and validation splits, while ensuring compatibility with our shared storage infrastructure. The dataset was subsequently tokenized using GPT-NeoX (Black et al., 2021). Although the complete dataset was

tokenized, our experimentation was limited to 100 billion tokens, with 10 billion tokens used for initial pre-training and the remaining 90 billion tokens utilized during the growth stage of our pipeline. The same 100 billion tokens were consistently used for training all models to ensure comparability across experiments.

5 Evaluation

We employed two primary evaluation metrics: (a) **relative speedup** compared to the baseline scratch model (Pythia-14M), and (b) **convergence analysis**. These metrics provide insight into the efficiency and effectiveness of the training processes across different model growth strategies, as compared to the regularly trained Pythia-14M model.

To track and visualize these metrics effectively, we used the *Weights & Biases (wandb)* platform. This allowed detailed monitoring of loss curves, training dynamics, and overall model performance, enhancing transparency and reproducibility.

In addition to these training metrics, we evaluated all models on **zero-shot QA tasks** using benchmark suites provided by EleutherAI. These benchmarks test the models' ability to generalize and reason in contexts where they were not explicitly trained. Below, we briefly describe each task included in our evaluation:

- **ARC-c (Challenge)** (Clark et al., 2018): The AI2 Reasoning Challenge (ARC) contains a challenging subset of science questions requiring contextual understanding and logical deduction.
- **ARC-e (Easy)** (Clark et al., 2018): A subset of the ARC dataset consisting of simpler science questions that evaluate fundamental understanding and reasoning.
- **LogiQA** (Liu et al., 2020): A logical reasoning benchmark designed to test a model's ability to deduce correct answers from given premises, often resembling competitive exam tasks.
- **PIQA (Physical Interaction QA)** (Bisk et al., 2020): This benchmark tests commonsense physical reasoning, evaluating a model's understanding of how everyday objects interact and function.
- **SciQ** (Welbl et al., 2017): A dataset of multiple-choice science questions covering topics like biology, chemistry, and physics, sourced from school-level examinations to test scientific reasoning and factual recall.
- **Winogrande** (Sakaguchi et al., 2021): A commonsense reasoning benchmark inspired by the Winograd Schema Challenge, focusing on resolving pronoun references using contextual commonsense knowledge.

6 Experiments

6.1 Experimental Setup

Our experiments were designed to explore the impact of model growth operators on the Pythia-14M architecture, with evaluations focused on training efficiency and downstream task performance. The study involved two primary sets of experiments:

6.1.1 Recreation of Stacking Experiments

We first sought to recreate the experiments outlined in Du et al. (Du et al., 2024), adapting their methodology to our computational constraints and model choices. Instead of using larger-scale models and unconstrained resources as in the original paper, we applied stacking principles to the Pythia architectures. This adaptation allowed us to evaluate the validity of stacking strategies under significantly constrained conditions.

6.1.2 Combinatorial Growth Experiments

Next, we conducted a series distinct experiments, corresponding to all pairwise combinations of growth operators applied independently to each model. For the baseline Pythia-14M, we focus on the following multidimensional combinations of growth operators were tested:

- Adding layers (depth) and increasing attention heads.
- Adding layers and increasing feed-forward network (FFN) dimensions.
- Increasing attention heads and increasing FFN dimensions.
- depth-wise stacking combined with widthwise stacking, following the strategy forwarded by Du et al. (Du et al., 2024).

Most of these combinations were (to our knowledge) never explored in adjacent works. They applied to evaluate the interaction effects between

growth operators and their impact on model scalability and performance. Each model-operator pair was evaluated using the suite aforementioned metrics.

6.2 Resource Constraints and Challenges

6.2.1 Computational Limitations.

A significant challenge in our experiments stemmed from shared computational resource constraints on PACE-ICE. While the stacking experiments in prior work were performed using extensive multi-GPU setups with long training sessions, our resources were limited to either intermittent access to eight GPUs in two-hour blocks, or single-GPU sessions in sixteen-hour sessions on NVIDIA H100 GPUs. This posed unique challenges in maintaining experimental consistency and replicability, particularly for long-running training processes.

6.2.2 Ensuring Consistency.

To address these challenges, we implemented several strategies to ensure consistency across experiments:

- **Checkpointing and resuming:** All experiments were designed to support frequent checkpointing, enabling training to resume seamlessly after interruptions. This included thorough work to ensure result visualization remained consistent even when runs were cut short and later continued.
- **Standardized training duration:** Despite the variability in session lengths, we ensured that all models were trained for a consistent number of effective steps by carefully controlling learning rates and scheduling restarts.
- **Automated resource management:** We developed scripts to dynamically allocate GPU resources based on availability, and re-queue model trainings once a training session expires.

While these limitations restricted some of our initial plans, they underscored the practical applicability of our work, particularly in tightly time- and compute-constrained scenarios.

7 Results

In this section, we evaluate the performance of our multidimensional growth operators against that of both (a) our recreations of the unidimensional growth operators, and (b) a set of baseline models' performances. We measure the performance and effectiveness of these operators relatively, using the suite of evaluation tools described above.

7.1 Training Efficiency

We measured pre-training efficiency using the relative speedup as defined by the FLOPs formula:

$$\text{speedup} = \left(\frac{\text{FLOPs}_{\text{scratch}}}{\text{FLOPs}_{G}} - 1 \right) \times 100\%$$

Here, $\text{FLOPs}_{\text{scratch}}$ represents the total floating-point operations required by the scratch Pythia-14M model to achieve a given loss, and FLOPs_{G} denotes the operations required by the model grown using operator G.

As shown in Table 1, all multidimensional growth operators required more FLOPs than unidimensional ones, slowing down by $5\% - 15\%$. However, multidimensional growth still achieved a slight speedup ($2\% - 8\%$) compared to the scratch model, showing better computational efficiency overall.

Unidimensional growth operators demonstrated the highest speedups, with depth-wise growth yielding the most significant gains due to its ability to preserve learned knowledge. The reduced efficiency of multidimensional growth is likely due to overhead from additional growth dimensions like feed-forward scaling or attention head expansion, which did not significantly improve initialization efficiency.

7.2 Convergence Analysis

Rapid convergence is a highly desirable characteristic for LLM pre-training, especially in resource-constrained environments. Below, we analyze our models as they approach 90% of their final validation performance.

Figure 2 illustrates the loss curves of the various models following growth over the initial 20,000 training steps. Compared to the scratch baseline, it is clear that the depth-wise $G^{\uparrow}_{\text{stack}}$ operator achieves the fastest convergence.

Additionally, when compared to the scratch baseline, all models employing depth-wise growth demonstrate a more rapid reduction in loss, resulting in faster convergence.

	Unidirectional Growth		**Multidirectional Growth**			Baseline		
	G_{ff}	G_{attn}	$G^{\uparrow}_{\text{stack, ff}}$	$G^{\uparrow}_{\text{stack, attn}}$	$G^{\uparrow\rightarrow}_{\text{stack}}$	$G^{\uparrow}_{\text{stack}}$	$G^{\uparrow}_{\text{rand}}$	Scratch
Zero-Shot QA Task Performance Metrics								
ARC-c	20.34	19.75	20.98	21.40	19.28	21.62	22.18	18.43
ARC-e	22.33	25.20	26.12	24.11	25.55	24.54	26.47	24.62
Logiqa	20.12	21.75	22.10	22.90	22.58	22.58	19.35	21.20
PIQA	50.23	51.69	50.50	52.75	52.56	51.99	52.61	52.01
Sciq	19.45	18.90	20.30	19.95	20.10	20.20	20.20	19.10
Winogrande	48.12	49.65	48.80	50.75	50.59	50.30	49.80	49.09
Avg. accuracy	30.10	31.16	31.46	31.97	31.77	31.82	31.76	30.74
Efficiency Metrics								
Speedup ($\Delta\%$)	-4.8	-6.3	3.1	2.5	7.6	12.9	11.5	0.00

Table 1: Performance and efficiency metrics for zero-shot QA tasks across different model growth strategies. The table compares Unidirectional Growth (e.g., G_{ff}, G_{attn}), Multidirectional Growth (e.g., $G^{\uparrow}_{\text{stack, ff}}$, $G^{\uparrow}_{\text{stack, attn}}$, $G^{\uparrow\rightarrow}_{\text{stack}}$), and Baseline models (e.g., $G^{\uparrow}_{\text{stack}}$, $G^{\uparrow}_{\text{rand}}$, Scratch). Metrics include accuracy on six QA tasks (ARC-c, ARC-e, LogiQA, PIQA, SciQ, and Winogrande) and relative training speedup (% Δ in FLOPs) for each growth strategy.

7.3 Task Performance

We evaluated all models across a range of downstream tasks, and their performance is summarized in Table 1. Multidimensional growth operators consistently outperformed both the unidimensional growth operators and the baseline Pythia-14M model, albeit with increased training time. On average, multidimensional growth achieved higher accuracy (31.97% for $G^{\uparrow}_{\text{stack, attn}}$) compared to unidimensional operators (31.16% for G_{attn}) and the baseline (30.74%).

The improved performance of multidimensional growth operators can be attributed to the combined benefits of depth-wise expansion and feature scaling, which enhance the model's capacity to capture complex hierarchical and fine-grained representations. For instance, $G^{\uparrow}_{\text{stack, attn}}$ achieved the highest accuracy on reasoning-heavy tasks like LogiQA (22.90%) and commonsense tasks like Winogrande (50.75%), demonstrating the utility of these growth strategies in improving deductive reasoning and contextual understanding.

While depth-wise growth alone provides substantial improvements, combining it with feed-forward scaling or additional attention heads further boosts task-specific performance. This synergy is particularly evident in tasks like ARC-e (26.12% for $G^{\uparrow}_{\text{stack, ff}}$) and PIQA ($52.75\%$ for $G^{\uparrow}_{\text{stack, attn}}$).

However, the modest gains in task performance come at the cost of higher computational overhead and slower training convergence, as noted in the efficiency metrics. Therefore, while multidimensional growth operators produce more robust models, their application should be weighed against the computational resources and time available for training.

8 Analysis

8.1 Effectiveness of Unidimensional Growth

Our unidirectional growth results were consistent with the results of Du et al., and contradicted those of Gu et al. (Du et al., 2024; Gu et al., 2020). Depth expansion predictably outperformed the other unidirectional strategies, but our novel unidirectional growth operators (ff expansion and attention head growth) yielded more robust models than the baseline. While we knew that increased model depth facilitates better hierarchical representation learning, we also can see that attention head scaling allows for better fine-grained dependency resolution, as demonstrated by performance on Winogrande.

8.2 Effectiveness of Multidimensional Growth

The analysis from Section 8.1 can be extended to the multidimensional growth operators.

The majority of our multidimensional operators relied on using a depth-wise growth operator due to their empirical performance. Predictably, the depth-width combined G_{stack} and G_{zero} operators yielded faster convergence and faster training time

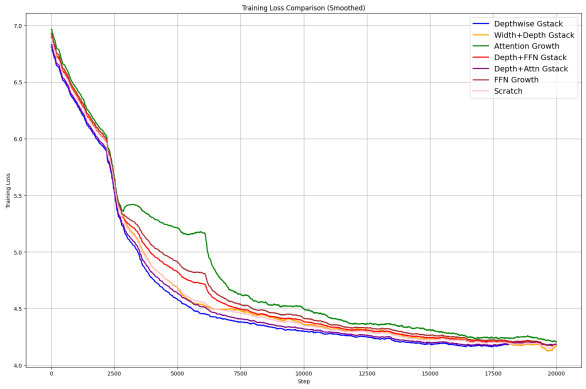

Figure 2: Training loss comparison across different model growth strategies. The blue line represents the *depth-wise* $G^{\uparrow}_{stack}$ operator, where layers are added as copies of the previous layers. The orange line corresponds to the *Width+Depth* $G^{\uparrow\rightarrow}_{stack}$ operator, expanding both layers and feed-forward network dimensions. The green line shows *Attention Growth* (G_{attn}), increasing the number of attention heads. The red line represents the *Depth+FFN* $G^{\uparrow}_{stack,\,ff}$ operator, while the purple line indicates the *Depth+Attn* $G^{\uparrow}_{stack,\,attn}$ operator. The brown line corresponds to *FFN Growth* (G_{ff}) alone, and the pink line shows the baseline *Scratch* model with no growth. Loss curves are smoothed to emphasize overall trends in convergence across training steps.

compared to the baseline. More importantly, these combinations produced more robust models.

The combinations of G_{stack} with our novel unidirectional operators created the most robust models, while still enabling a moderate speedup. However, when compared to the unidirectional depth-wise growth mechanisms, the added model robustness may not justify the increased training cost.

The combination of ff growth and attention head growth translated to a more robust model than the baseline, but at the cost of a slowdown. Additionally, when these operators were combined with G_{stack}, they yielded better performance improvements.

8.3 Future Directions

Our experiments were primitive due to the lack of ablation study. We were not able to conduct enough experiments to determine an optimal growth factor and were not able to test our strategies on numerous other models.

Due to these limitations, future work can directly be done on finding the optimal hyperparameters for multidimensional growth. Since our results demonstrate that multidimensional growth can translate to more robust models, tuning hyperparameters could result in more efficient training while retaining the performance increase.

Performing ablation studies over different models and different growth factors is necessary to determine if multidimensional growth is a viable approach to model growth. While our work introduces these novel operators, and performs the first batch of experiments, Du et al. demonstrates there is not a linear scaling law when it comes to model growth. Growing the models more or less in either dimension could translate to more effective model growth.

We additionally only combine two dimensions when growing the models – combining multiple operators could also lead to more effective growth. Based on our results, if we were to grow FFN dimensions, attention heads, and the number of layers, we could potentially produce a more robust model.

9 Conclusion

This study explored the impact of model growth strategies on enhancing the efficiency and effectiveness of training large language models (LLMs). By systematically examining unidimensional and multidimensional growth operators on the Pythia-14M architecture, we demonstrated the potential of these strategies to accelerate training, improve convergence rates, and enhance downstream task performance.

While multidimensional growth operators produced marginally more robust models, the slight performance increases came at the cost of training efficiency. Consequently, the utility of multidimensional growth strategies must be weighed against their computational overhead. Future work should focus on optimizing growth factors and exploring more combinations of growth dimensions to determine whether these strategies can consistently yield robust, efficient training pipelines. Through such refinement, multidimensional growth could evolve into a practical tool for scalable and adaptable LLM development.

10 Acknowledgments

We acknowledge the use of OpenAI's ChatGPT and Anthropic's Claude as tools to assist in refining and clarifying portions of this document.

References

Stella Biderman et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. *International Conference on Machine Learning*, pages 2397–2430.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. Piqa: Reasoning about physical commonsense in natural language. *arXiv preprint arXiv:1911.11641*.

Sid Black, Leo Gao, Phil Wang, Connor Leahy, Stella Biderman, et al. 2021. Gpt-neox: Large scale autoregressive language modeling in pytorch. https://github.com/EleutherAI/gpt-neox.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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.

Mia Xu Chen, Tomáš Kocisky, Edward Grefenstette, Colin Raffel, and Noam Shazeer. 2021a. Bert2bert: Towards reusable pretrained language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages x–y.

Mia Xu Chen, Mingxing Tan, and Quoc V Le. 2021b. The best of both worlds: Combining recent advances in neural architecture search. *ArXiv preprint arXiv:2106.07978*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *ArXiv preprint arXiv:2204.02311*.

Alexandra Chronopoulou, Shuo Fu, Lemao Liu, Ting Liu, and Graham Neubig. 2023. Adaptersoup: Weight averaging to improve adapter performance. *Transactions of the Association for Computational Linguistics*.

Eric Chu, Mia Xu Chen, et al. 2024a. Finetuning methods for large language models: A comprehensive survey. In *Proceedings of the 2024 Conference of the Association for Computational Linguistics*.

Shao-Heng Chu, Han Soo Kim, Nikhi Shanbhogue, and Dulani Wijayarathne. 2024b. Fine tuning methods for conversational ai systems. *Case Study*. Email: schu45@gatech.edu, hkim3190@gatech.edu, nshanbhogue6@gatech.edu, dwijayarathne3@gatech.edu.

Peter Clark, Mark Yatskar, and Kyle Richardson. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Wenyu Du et al. 2024. Stacking your transformers: A closer look at model growth for efficient llm pretraining. *NeurIPS*.

Lixiang Fang, Chenfei Wang, Yu Zhang, Pengcheng Xu, et al. 2022. Tess: Teacher-student low-rank adaptation for efficient fine-tuning of large language models. *ArXiv preprint arXiv:2205.12345*.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 249–256.

Yu Gong, Yu Li, Xi Chen, Xinyu Jiang, Bo Li, Yuexin Niu, Yuxuan Zhang, Cong Zhang, and Jingren Zhou. 2019. Efficient training of bert by progressively stacking layers. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, page 132–141.

Xiaodong Gu, Linting Wang, Ming Xue, Tao Qin, Dong Li, Weizhu Chen, and Tie-Yan Liu. 2020. Transformer growth: Efficient training of large transformers by progressively growing the model size. *ArXiv preprint arXiv:2010.XXXX*.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in Neural Information Processing Systems*, 28:1135–1143.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Ethan Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *Advances in Neural Information Processing Systems*.

Neil Houlsby, Alham F Giurgiu, Stanislaw Jastrzebski, Brianna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. *Association for Computational Linguistics*.

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 *Advances in Neural Information Processing Systems*.

Chenxi Liu, Barret Zoph, Mario Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34.

Zhengyuan Liu, Lei Zhang, Chao Tan, Songfang Chen, Fei Zhao, Shijin Huang, and Luo Si. 2020. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. *arXiv preprint arXiv:2007.08124*.

Jonas Pfeiffer, Uday Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. Adapterfusion: Non-destructive task composition for transformer models. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics*, pages 487–503.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740.

John Smith, Wei Li, Aditi Gupta, and etc. 2024. Slimpajama-627b: A deduplicated, multilingual, multi-document-format dataset for pretraining language models. *ArXiv preprint arXiv:2401.XXXX*.

Jianlin Su, Yu Lu, Shengfeng Pan, and Bo Wen. 2021. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*.

Hugo Touvron, Louis Martin, Kevin Stone, Pierre Albert, and etc. 2023. Llama 2: Open foundation and fine-tuned chat models. *ArXiv preprint arXiv:2307.09288*.

Johannes Welbl, Nelson F Liu, and Matt Gardner. 2017. Crowdsourcing multiple choice science questions. *arXiv preprint arXiv:1707.06209*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2020. Huggingface transformers. https://huggingface.co/.

Xiao Wu, Yicheng Zhang, and Danqi Chen. 2022. Sustainable and efficient training of large language models. *ArXiv preprint arXiv:2210.XXXX*.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Chris Dewan, Mona Diab, Emily Dinan, Sam Shleifer, et al. 2022. Opt: Open pre-trained transformers language models. *ArXiv preprint arXiv:2205.01068*.