Stacking Your Transformers

A Closer Look at **Model Growth** for **Efficient LLM Pre-Training**

Wenyu Du[1] Tongxu Luo[1] Zihan Qiu Zeyu Huang Yikang Shen
Reynold Cheng Yike Guo Jie Fu[2]

The University of Hong Kong Hong Kong University of Science and Technology

Tsinghua University University of Edinburgh MIT-IBM Watson AI Lab

[1] Equal Contributions
[2] Corresponding Author

Table of Contents

What is Model Growth?

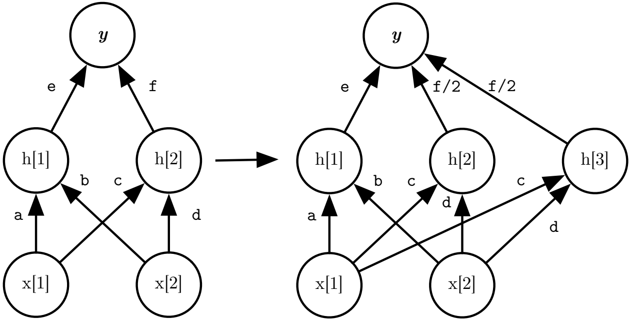
Aim

- Leverage trained smaller (base) models to accelerate the training of larger (target) models.
- Expect a faster speed given the same budget, compared with model trained from scratch.

Process

Train a small model → Grow → Continual training the large model

Example: Net2Net, ICLR 2016



$$h[1] = \mathrm{ReLU}(x[1] \cdot a + x[2] \cdot b)$$
$$h[2] = \mathrm{ReLU}(x[1] \cdot c + x[2] \cdot d)$$
$$y = \mathrm{ReLU}(h[1] \cdot e + h[2] \cdot f)$$

$$h[1] = \mathrm{ReLU}(x[1] \cdot a + x[2] \cdot b)$$
$$h[2] = \mathrm{ReLU}(x[1] \cdot c + x[2] \cdot d)$$
$$h[3] = \mathrm{ReLU}(x[1] \cdot c + x[2] \cdot d)$$
$$y = \mathrm{ReLU}(h[1] \cdot e + h[2] \cdot \frac{f}{2} + h[3] \cdot \frac{f}{2})$$

What is Model Growth?

Many Follow-ups

- **stackBert** ICML19, **bert2bert** ACL20, **stagedTrain** 22, **GradMax** ICLR22, **LiGO** ICLR23, **Lemon** ICLR24, **MSG** ICLR24, . . .

Impressive performance

- And they assert they can **speedup the training phase for about 30% to 60%**.

But ...

- These techniques are **underexplored** in pre-training LLM. Underexplored
- *Considering how expensive LLM pre-training is, if we could successfully* ***adopt model growth techniques to LLM pre-training****, which would be a great contribution to efficiency and resource-saving.* Expensive

Underexplored in Efficient LLM Pre-Training

- Model growth techniques are **underexplored** in pre-training LLM.

Figure: MSG ICLR24

Method	Wall time	GLUE Avg.	CoLA	SST-2	MRPC	STS-B
Full-B	26h, 10min	80.7(0.2)	52.2(0.8)	90.4(0.2)	**85.9(0.9)/90.1(0.5)**	**88.8(0.1)/88.4(0.1)**
N2N-sch1-B	**14h, 36min**	80.5(0.2)	52.4(1.5)	**91.1(0.4)**	84.3(0.9)/88.8(0.6)	88.3(0.1)/88.0(0.1)
MSG-sch1-B	**14h, 32min**	**81.0(0.2)**	**58.2(1.6)**	91.0(0.2)	85.0(0.5)/89.4(0.5)	88.1(0.1)/87.6(0.1)

Method	QQP	MNLI(m/mm)	QNLI	RTE	SQuADv1.1
Full-B	**90.6(0.1)/87.3(0.1)**	**82.5(0.3)/82.9(0.1)**	89.9(0.1)	**65.1(0.7)**	79.1(0.2)/86.9(0.2)
N2N-sch1-B	90.1(0.3)/87.0(0.1)	81.1(0.2)/82.1(0.1)	89.2(0.1)	66.3(0.5)	79.0(0.1)/86.7(0.0)
MSG-sch1-B	90.0(0.1)/87.0(0.1)	81.8(0.3)/82.4(0.2)	**89.9(0.1)**	63.1(1.6)	**79.6(0.5)/87.2(0.4)**

Table 3: Evaluation of Bert-base after fine-tuning on downstream tasks. For metrics, we use Matthews correlation for CoLA, Pearson/Spearman correlation for STS-B, accuracy/f1 for MRPC, QQP, and SQuAD, and accuracy for all the other tasks. The numbers are mean (standard deviation) computed across 3 runs.

Figure: LEMON ICLR24

Table 2: Downstream performance of BERT(12, 768) on the GLUE dataset: Large model expanded from BERT(6,384) achieves the best downstream performance. A potential reason for this may be its longer training duration (165k) compared to the BERT(6,512) (132k).

Dataset (Metric)	STS-B (Corr.)	MRPC (Acc.)	CoLA (Mcc.)	SST-2 (Acc.)	QNLI (Acc.)	MNLI (Acc.)	MNLI-mm (Acc.)	QQP (Acc.)
Train from scratch	0.744	83.33	0.19	88.88	87.80	80.28	81.17	89.62
LEMON (Ours), from BERT(6, 512)	0.848	83.82	0.36	90.14	88.76	80.92	81.57	89.91
LEMON (Ours), from BERT(6, 384)	**0.866**	**85.54**	**0.38**	**90.94**	**89.33**	**81.81**	**81.81**	**90.40**

Expensive in LLM Pre-Training

- The advance of LLM comes at the expensive cost of energey consumption[3].

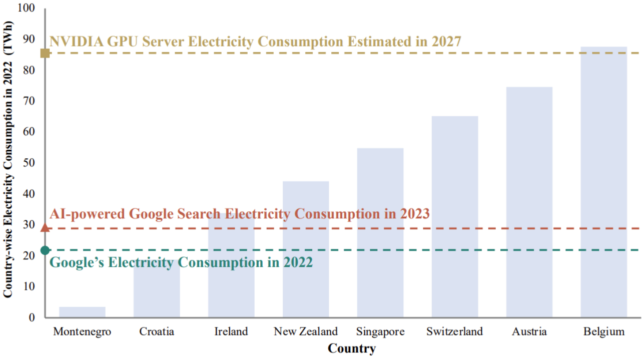


Figure 1: The electricity consumption comparison between countries and AI. Data source: [77].

[3] 1, *A Survey of Resource-efficient LLM and Multimodal Foundation Models*, 2024.

Investigate Model Growth for LLM Pre-Training

Therefore, in this work ...

Aim 🎯 in this work

We aim to investigate model growth for efficient LLM pre-training.

***In this presentation**

- This presentation is basically showing the **steps** involved in our investigation of this project.
- Particularly, we address **Three Obstacles** step by step.

Three Identified Obstacles and Three Corresponding Questions

- O1: Lack of comprehensive assessment
  $\Rightarrow$ Q1: Do Model Growth Methods Work in LLM Pre-Training?
- O2: The untested scalability
  $\Rightarrow$ Q2: Are These Methods scalable?
- O3: Lack of empirical guidelines
  $\Rightarrow$ Q3: How to use in practice?

```
Investigate O1(Lack of comprehensive assessment) Obstacle One
if Q1 is true:
    Investigate O2(The untested scalability) Obstacle Two
    if Q2 is true:
        Investigate O3(Lack of empirical guidelines) Obstacle Three
```

Obstacle One - Lack of Comprehensive Assessment

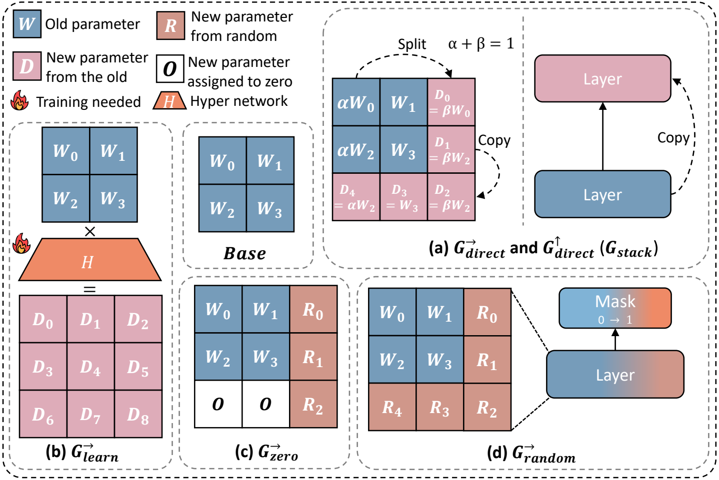
Aim

Examine whether model growth techniques actually work in LLM pre-training.

Process

1. Category model growth techniques into **four atomic growth operators**, G_{direct}, G_{learn}, G_{zero} and G_{random}.
2. Then we examine them into depthwise growth and widthwise growth, $G^{\uparrow}$ and $G^{\rightarrow}$.

Four Atomic Growth Operators: G



Tips

You may refer to animated `GIF` atomic growth operators.

Experiment Details

- Codebase: Tiny-llama codebase `https://github.com/jzhang38/TinyLlama`
- Dataset: Slimpajama-627B `https://huggingface.co/datasets/cerebras/SlimPajama-627B`

Process – Grow from 410M LLM to 1.1B LLM

Train a `LLM(6L;2048H)` for 10B tokens
↓
$\texttt{LLM(24L;2048H)} = G^{\uparrow}(\texttt{LLM(6L;2048H)})$
↓
Then train `LLM(24L;2048H)` for 100B tokens

Train a `LLM(24L;1024H)` for 10B tokens
↓
$\texttt{LLM(24L;2048H)} = G^{\rightarrow}(\texttt{LLM(24L;1024H)})$
↓
Then train `LLM(24L;2048H)` for 100B tokens

Experiment Results

	Depth				Width				Baseline
	$G^{\uparrow}_{direct}$	$G^{\uparrow}_{zero}$	$G^{\uparrow}_{random}$	$G^{\uparrow}_{learn}$	$G^{\rightarrow}_{direct}$	$G^{\rightarrow}_{zero}$	$G^{\rightarrow}_{random}$	$G^{\rightarrow}_{learn}$	*scratch*
Lambada (↑)	48.20	48.67	44.14	48.36	46.16	44.67	44.24	45.66	47.87
ARC-c (↑)	29.18	28.32	28.41	27.38	28.58	26.70	27.64	26.70	27.21
ARC-e (↑)	54.25	51.76	52.69	51.17	51.55	49.70	53.82	50.37	48.86
Logiqa (↑)	28.87	27.95	25.96	28.11	27.34	25.03	26.11	26.57	25.96
PIQA (↑)	71.98	71.81	70.78	71.16	69.47	69.74	70.13	69.91	69.64
Sciq (↑)	81.1	81.9	77.7	80.0	81.4	76.0	79.5	79.5	76.8
Winogrande (↑)	56.03	56.98	53.35	54.45	54.22	54.93	52.95	53.51	54.53
Avg. (↑)	52.80	52.48	50.43	51.52	51.25	49.54	50.63	50.32	50.12
Wikitext (↓)	16.73	17.35	17.85	16.93	18.03	18.76	18.29	18.44	17.98
Loss (↓)	2.151	2.161	2.258	2.156	2.209	2.249	2.227	2.233	2.204
Speed-up (↑)	49.1%	46.6%	-25.7%	48.6%	-0.7%	-17.9%	-13.8%	-15.4%	0.0%

Takeaways

- In general, $G^{\uparrow}$ is better than $G^{\rightarrow}$.
- **$G^{\uparrow}_{\mathbf{direct}}$ emerges as the clear winner.**
- **We denote $G^{\uparrow}_{\mathbf{direct}}$ as $G_{\mathbf{stack}}$.**

Back to Three Obstacles

Obstacle Two - The Untested Scalability

Aim

Is G_{stack} scalable (robust) in efficient LLM pre-training?

Process

1. Scale to training 3B and 7B LLMs.

Train a `LLM(8L;2560H)` for 10B tokens

↓

`LLM(32L;2560H)` $= G_{\text{stack}}($`LLM(8L;2560H)`$)$

↓

Then train `LLM(32L;2560H)` for 300B tokens

Train a `LLM(8L;4096H)` for 10B tokens

↓

`LLM(32L;4096H)` $= G_{\text{stack}}($`LLM(8L;4096H)`$)$

↓

Then train `LLM(32L;4096H)` for 300B tokens

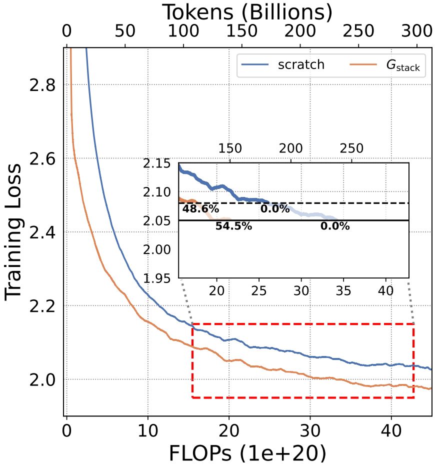


Figure: Training Loss on 3B

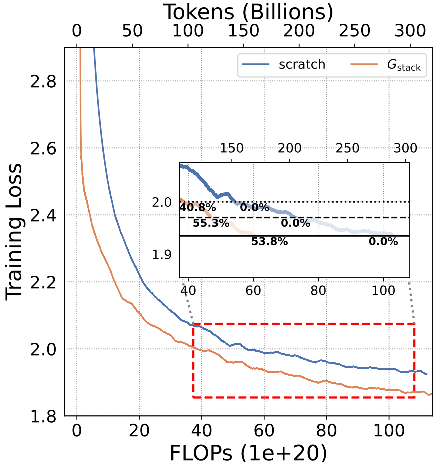


Figure: Training Loss on 7B

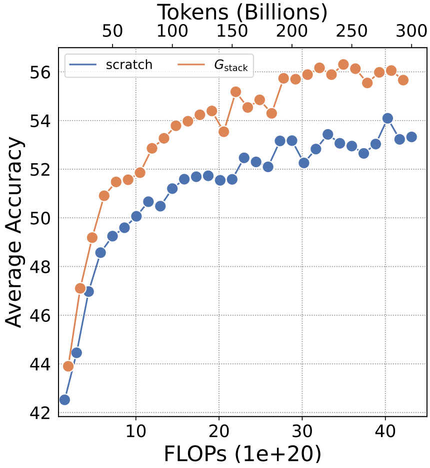


Figure: Average Accuracy on 3B

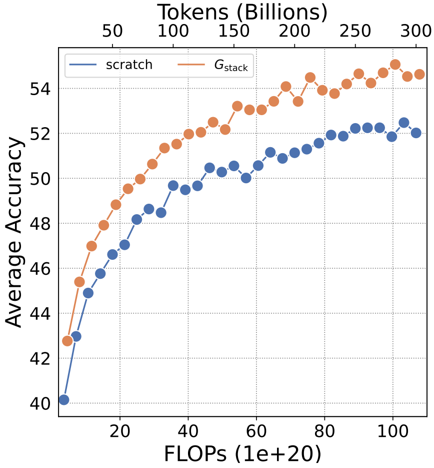


Figure: Average Accuracy on 7B

Obstacle Two - The Untested Scalability

Concern

Efficient strategies may initially learn faster but ultimately perform similarly or worse than vanilla training methods when given more training data.

Process

2. Scale to larger training tokens. We "overtrain" a 410M LLM for 750B tokens, which is almost **100 times larger** than Chinchilla scaling law recommended (8B).

Train a `LLM(6L;1024H)` for 10B tokens

↓

$\texttt{LLM(24L;1024H)} = G_{\text{stack}}(\texttt{LLM(6L;1024H)})$

↓

Then train `LLM(24L;1024H)` for 750B tokens

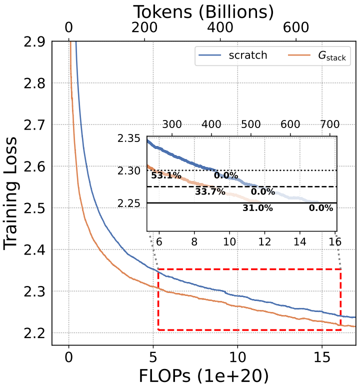


Figure: Training Loss on 410M with 750B tokens

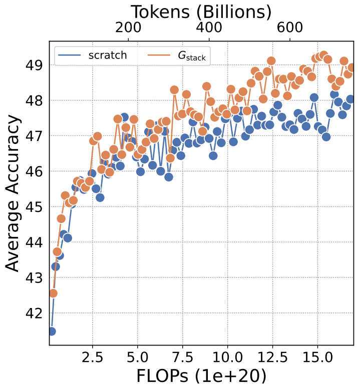


Figure: Average Accuracy on 410M with 750B tokens

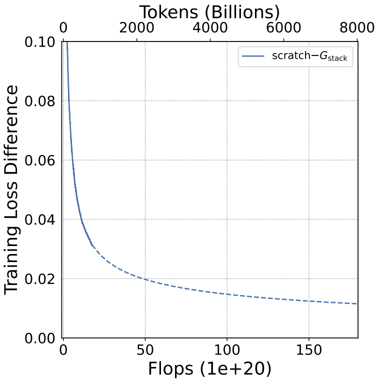
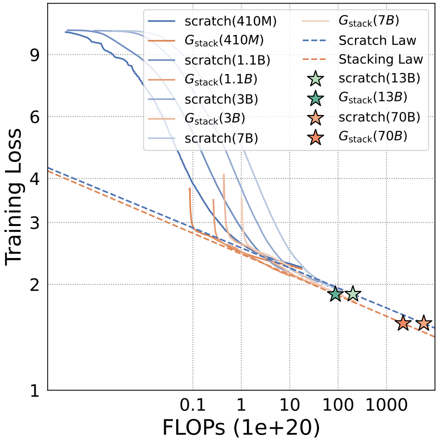


Figure: Loss Difference

Estimated Scaling Laws



- We plot our four models (410M, 1.1B, 3B, and 7B) on the same figure.
- Then uncover our "scaling law" using the G_{stack} operator: $L_C = aC^b$

Takeaways

- G_{stack} is **scalable** in both model scale and training tokens.
- G_{stack} scaling law exhibits **improved efficiency** compared to the scaling law estimated from baseline LLMs.

Back to Three Obstacles

Obstacle Three - Lack of Empirical Guidelines

Aim

How to use G_{stack} in practice?

Process

Determining Growth Timing (d) and Growth Factor (g).

- Growth timing d: the training token d for the small model.
- Growth factor g: the factor by which the model parameters increased after growth (roughly equivalent to the ratio of increased layers in G_{stack}).

$$log_{10}(d) = a\log_{10}(N) + \frac{b}{\log_{10}(C)} + c, \tag{9}$$

where C is the computing budget and N is the target parameter size.

Growth Timing *d*

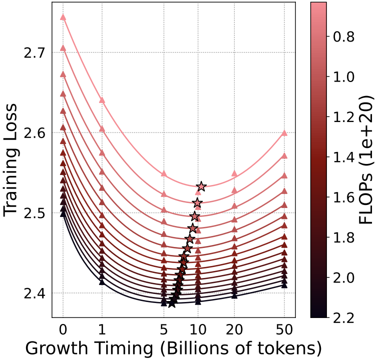


Figure: IsoFLOP on 410M

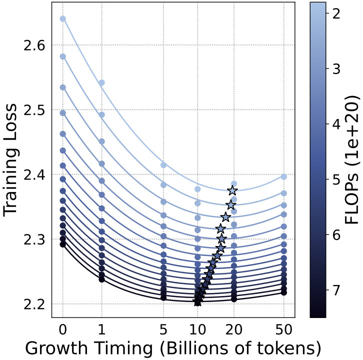


Figure: IsoFLOP on 1.1B

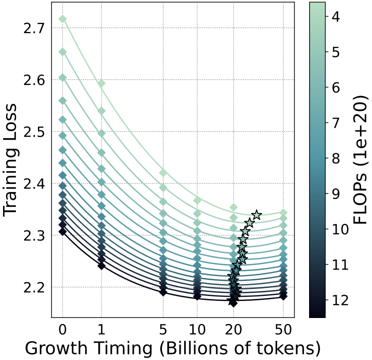
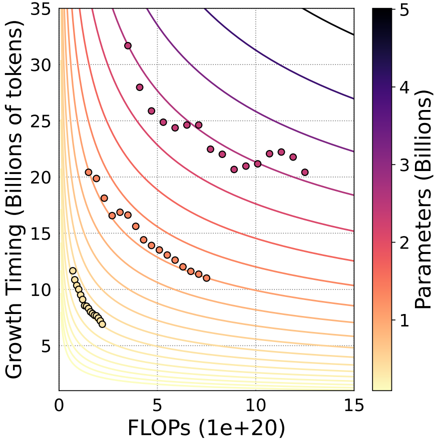


Figure: IsoFLOP on 3B

Predicting Growth Timing *d*



- We formalize a set of guidelines for effectively utilizing the G_{stack} operator. For growth timing d (tokens):
-
$$log_{10}(d) = 0.88\log_{10}(N) + \frac{163.27}{\log_{10}(C)} - 5.74 \quad (10)$$
- where C is the computing budget and N is the model parameters.

Growth Factor g

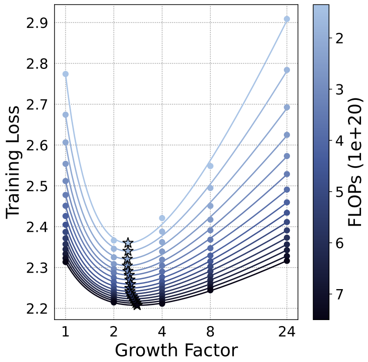


Figure: IsoFLOP on 410M

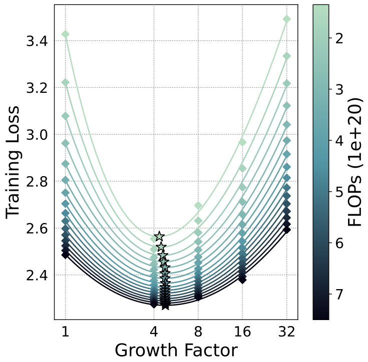


Figure: IsoFLOP on 1.1B

Takeaways

- For predicting growth timing d, please refer to Eq 10.
- For predicting growth factor g, due to computational constraints, we indicate that the optimal growth factor g lies between 2 and 4.

Takeaways of Three Obstacles

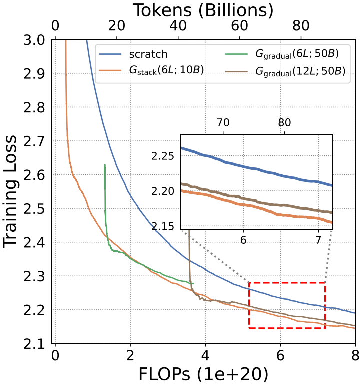
Back to Three Obstacles

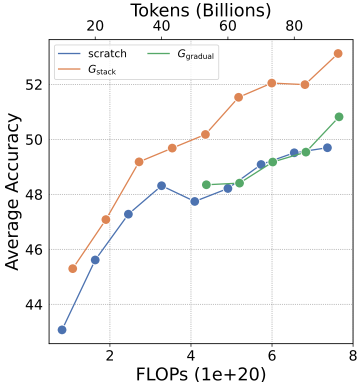
- Q1: Do Model Growth Methods Work in LLM Pre-Training?
  $\Rightarrow$ We summarize the existing model growth approaches into four operators and make a comprehensive evaluation, the depthwise growth G_{stack} beats all other methods.
- Q2: Are These Methods scalable?
  $\Rightarrow$ We scale up G_{stack} by extending the model size and training data scales. We find that G_{stack} operator has excellent scalability.
- Q3: How to use in practice?
  $\Rightarrow$ We systematically analyze the usage of the G_{stack} operator, focusing on growth timing and growth factor. We provide guidelines of equations for effectively utilizing the G_{stack} operator.

How to stack?

1. Gradual Stacking

- We compare our "one-hop" G_{stack} and gradual stacking approach (two-step: train-stack-train-stack).
- G_{stack} achieves a 2.4 higher average accuracy and 0.6 better Wikitext PPL than gradual stacking when pre-training large models for 100B tokens.

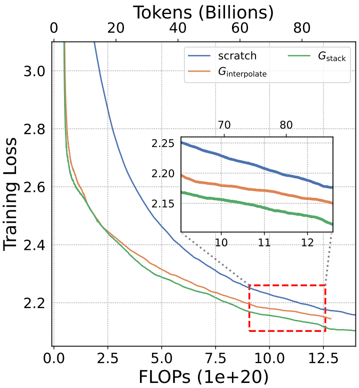


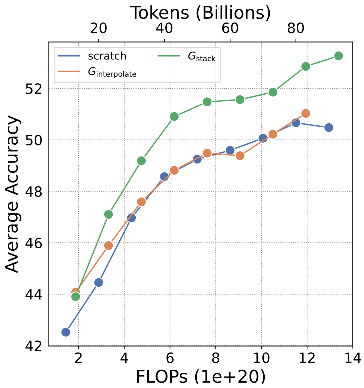


How to stack?

2. Interpolation

- G_{stack} involves taking the entire small model as a unit and directly stacking it, which can retain the connections between most layers.
- Interpolation involves replicating and interleaving each layer in the small model, which almost break the connections.





To measure the degree of adjacent inter-layer connections after stacking, we define the connection rate R_c:

$$R_c = \frac{Con_r}{Con_{all}} \quad (11)$$

where the Con_r is number of retained connections, the Con_{all} is number of all layers.

Example

For example, if we had a small model with three layers, denoted as $\{L_1, L_2, L_3\}$, and desired a model depth of 6, G_{stack} would result in $\{L_1, L_2, L_3, L_1, L_2, L_3\}$, where its $R_c = 80\%$. The interpolation approach would result in $\{L_1, L_1, L_2, L_2, L_3, L_3\}$, where its $R_c = 40\%$.

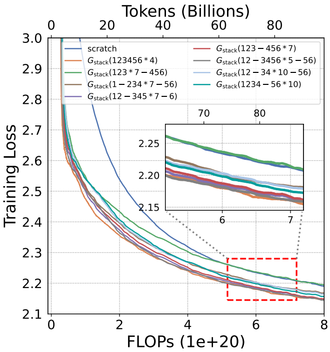
How to stack?

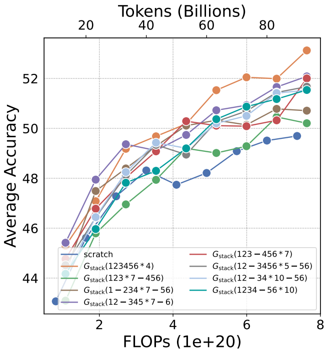
3. Partial Stacking

- We stack a small model with 6 layers ($\{L_1, L_2, \cdots, L_6\}$) to a 24 layers target model.
- Partial stacking has been explored in LLMs like LlamaPro[a], Solar[b]. But their goal is to stack an off-the-shelf LLM such as Llama2.

[a] 2, "Llama pro: Progressive llama with block expansion", 2024.

[b] 3, "Solar 10.7 b: Scaling large language models with simple yet effective depth up-scaling", 2023.





Eight partial stacking methods can be divided into three groups based on their loss.

- The first group, {123456*4, 12-3456*5-56, 12-345*7-6, 123-456*7}, achieves the best.
- The second group consisting of {1234-56*10, 12-34*10-56, 1-234*7-56}, performs just fine.
- The third group, {123*7-456}, performs poorly, even worse than the baseline.

Group	Method	Stacked parts	R_c
First	123456*4	all	87.0%
	12-3456*5-56	middle-back	78.3%
	12-345*7-6	middle-back	74.0%
	123-456*7	back	74.0%
Second	1234-56*10	back	60.7%
	12-34*10-56	middle	60.7%
	1-234*7-56	front-middle	74.0%
Third	123*7-456	front	74.0%

Takeaways

- we conclude that: all $>$ middle $\approx$ back $\gg$ front.
- Meanwhile, when the stacked parts are the same, the larger the R_c, the better the performance.

Conclusion

- This work empirically explores model growth approaches for efficient LLM pre-training.
- We first comprehensively evaluate model growth techniques into four atomic operators and explore depthwise growth G_{stack} beats all other methods and baselines in various evaluations.
- We next address concerns about the scalability of G_{stack} by extending the model and training data scales.
- Furthermore, we systematically analyze the usage of the G_{stack} operator, focusing on growth timing and growth factor.

Please visit homepage for the paper, codes and ckpts: https://llm-stacking.github.io/

Thanks!

References I

[1] Mengwei Xu et al. *A Survey of Resource-efficient LLM and Multimodal Foundation Models*. 2024. arXiv: 2401.08092 [cs.LG]. URL: https://arxiv.org/abs/2401.08092.

[2] Chengyue Wu et al. "Llama pro: Progressive llama with block expansion". In: *arXiv preprint arXiv:2401.02415* (2024).

[3] Dahyun Kim et al. "Solar 10.7 b: Scaling large language models with simple yet effective depth up-scaling". In: *arXiv preprint arXiv:2312.15166* (2023).