Progressive Residual Warmup for Language Model Pretraining

Tianhao Chen[1], **Xin Xu**[1], **Lu Yin**[2], **Hao Chen**[3], **Yang Wang**[4], **Shizhe Diao**[5], **Can Yang**[1,6,7,8,†]
[1]Department of Mathematics, HKUST
[2]School of Computer Science and Artificial Intelligence, SUAT
[3]Department of Computer Science and Engineering, HKUST
[4]Department of Mathematics, HKU [5]NVIDIA
[6]Big Data Bio-Intelligence Lab, HKUST
[7]State Key Laboratory of Nervous System Disorders, HKUST
[8]IAS Center for AI for Scientific Discoveries, HKUST

Abstract

Transformer architectures serve as the backbone for most modern Large Language Models, therefore their pretraining stability and convergence speed are of central concern. Motivated by the logical dependency of sequentially stacked layers, we propose **Pro**gressive **Res**idual Warmup (**ProRes**) for language model pretraining. ProRes implements an "early layer learns first" philosophy by multiplying each layer's residual with a scalar that gradually warms up from 0 to 1, with deeper layers taking longer warmup steps. In this way, deeper layers wait for early layers to settle into a more stable regime before contributing to learning. We demonstrate the effectiveness of ProRes through pretraining experiments across various model scales, as well as normalization and initialization schemes. Comprehensive analysis shows that ProRes not only stabilizes pretraining but also introduces a unique optimization trajectory, leading to faster convergence, stronger generalization and better downstream performance. Our code is available at https://github.com/dandingsky/ProRes.

1 Introduction

The Transformer [1] has become one of the most dominant backbones for Large Language Models (LLMs), underpinning recent advances across natural language processing and beyond. Among the many components of the Transformer architecture, residual connection [2] and normalization [3] play a key role in its optimization, enabling scaling LLMs up to trillion parameters [4].

Despite these advances, scaling Transformers poses unique optimization challenges. A large body of prior work has explored stabilizing Transformer training, including switching from Post-LN to Pre-LN [5], bounding model updates through depth- or width-aware initializations [6–12], and introducing additional flexibility into residual connections [13–16]. While effective, these methods are largely not *training-phase-aware*, in the sense that their mechanisms are typically applied at initialization, and then leave the optimizer to figure out how to adapt throughout training. In practice, Transformer training exhibits distinct phases [17]. Early training, such as the warmup stage, is often characterized by large and chaotic model updates, whereas the stable training phase involves relatively small and gradual modifications. Moreover, convergence dynamics are heterogeneous across depth, with shallow[1] layers tending to converge earlier than deeper layers [18]. These inherent dynamics

[†]Corresponding author.
[1]We use "shallow" or "early", and "deeper" or "later" interchangeably to describe layer order.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

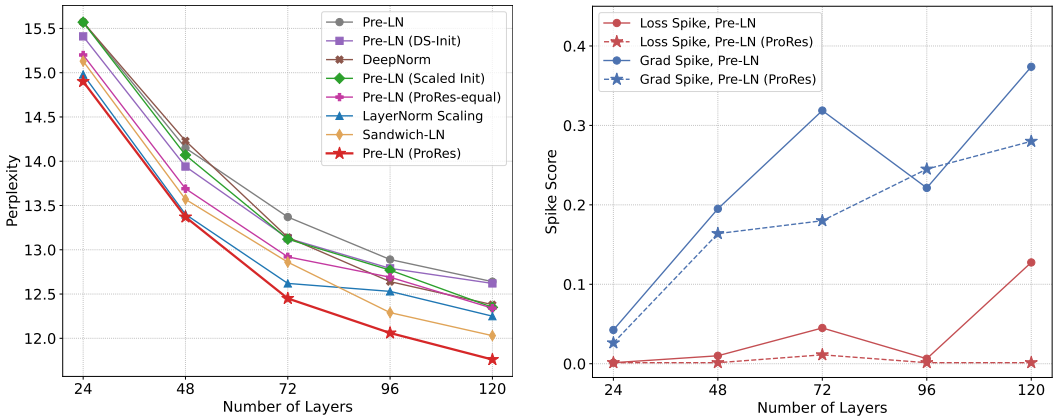


Figure 1: **Left**: Evaluation perplexity (↓) of different methods as model depth increases. **Right**: Spike score (↓) of loss and gradient norm as model depth increases. See Section 4.4 for details.

suggest that Transformer optimization may benefit from explicitly coordinating layerwise learning across different stages of training.

Motivated by these considerations, we revisit the role of residual connections in layerwise learning from a temporal perspective. Modern LLMs typically consist of sequentially stacked Transformer layers, where each layer updates its input representation[2] through residual branches consisting of attention and feed-forward sublayers. While residual connections enable effective optimization at scale, they also allow all layers to modify representations simultaneously from initialization. In the absence of explicit coordination, deeper layers may begin contributing before upstream representations have stabilized, potentially leading to inefficient updates or conflicting learning signals. This observation raises a natural question: can residual contributions be scheduled to respect the staged nature of Transformer training? In particular, can earlier layers be prioritized during early training, while allowing deeper layers to fully engage once upstream representations have matured?

In this work, we propose **Pro**gressive **Res**idual Warmup (**ProRes**), a simple and scalable approach for coordinating layerwise residual learning over the course of training. ProRes assigns each layer's residual with a predefined scaling factor. This scalar is initialized to 0, and gradually increases to 1 as training progresses. In particular, deeper layers take longer residual warmup than shallow layers, delaying their contribution to representations until shallow layers enter a more stable regime. As a result, shallow layers are prioritized in early training, while deeper layers progressively engage later on. This design controls model updates in the early stages, reduces unnecessary interference across layers, and preserves the expressive capacity of deep layers in later phases of optimization.

We evaluate ProRes through extensive language model pretraining experiments across a range of model scales, initialization methods, and normalization schemes. As shown in Figure 1, ProRes consistently improves depth scaling performance, enabling deeper models to achieve better perplexity without compromising training stability. We further analyze the optimization dynamics introduced by ProRes, shedding light on how coordinating learning order across depth affects learning behavior and representation evolution.

Our main contributions can be summarized as follows:

- We propose ProRes, a residual learning scheme that explicitly coordinates layerwise contributions while respecting the staged nature of Transformer convergence.
- Extensive pretraining experiments from 71M to 7B parameters show that ProRes improves performance across a wide range of model scales, initialization methods, and normalization schemes.

[2]We use "representation", "hidden state", and "activation" interchangeably. Unless otherwise mentioned, they refer to the token representation at certain Transformer layers.

- We analyze the learning dynamics induced by ProRes, providing insight into how learning order across layers influences training stability and efficiency.

2 Related Work

Transformer initialization and normalization. The objective of better initialization or normalization schemes is to stabilize training or accelerate convergence, by controlling the magnitude of activations, weights, gradients and model updates, etc. The original Post-LN [1] stabilizes activations by performing normalization after the residual connection. In more recent works, Pre-LN and its variants are adopted because of their stability and efficiency at scale [19, 5, 20, 21]. To enable effective depth scaling, several works proposed to scale layerwise weights according to the total number of layers, either uniformly [19, 22, 7, 10], or layer-specific [8, 23]. Yang et al. [11] proposed an initialization scheme that enables hyperparameter transfer among various scales. OLMo et al. [24] found that a simple normal initialization for all weights is more stable and transferable in their setting. A number of works investigated enhancing or replacing normalization with learnable gates or predefined scalings on the residual connection [14, 25–27, 12, 28]. Efforts have been made to combine the advantages of Pre-LN and Post-LN [15, 29, 30]. Several works explored enhancing representation capacity by expanding the width of residual stream [13, 16, 31].

Progressive growing and freezing layers. A similar line of research focuses on progressive training of Transformers. Gong et al. [32] proposed to train deeper BERT models by progressively stacking shallower ones, resulting in an overall shorter training time. Yang et al. [33] further proposed only training the newly added layers on top for improved efficiency. Similarly, Erdogan et al. [18] found that shallow layers converge earlier than deeper layers, and proposed a progressive freezing training method to reduce convergence wall time.

3 Method

3.1 Overview

We begin by describing the implementation of ProRes using Pre-LN Transformer as an example. The forward equation of a Pre-LN layer is:

$$x_{l+1} = x_l + \mathcal{F}(\mathrm{Norm}(x_l)), \tag{1}$$

where x_l is the input representation for layer l, $\mathcal{F}$ denotes the attention or FFN module, and Norm denotes normalization layer. ProRes modifies the residual connection with a scalar:

$$x_{l+1} = x_l + \alpha(l,t) \cdot \mathcal{F}(\mathrm{Norm}(x_l)). \tag{2}$$

Here $\alpha(l,t) \in \mathbb{R}$ is a predefined scalar that depends on the layer index l and the training step t. An example schedule for this variable is:

$$\alpha(l,t) = \min\left(\frac{t}{T \times l}, 1\right), \quad l = 1, \ldots, L, \tag{3}$$

where T is the warmup length for the first layer and L is the total number of layers. In other words, $\alpha(l,t)$ warms up from 0 to 1 linearly during training, and the warmup length increases linearly with layer index. Under this schedule, residual branches are activated sequentially from shallow to deep layers. We study multiple alternative schedules in Section 4.3.

Applying ProRes to other Transformer variants follows the same principle, as long as they consist of stacked Transformer layers with residual connection. All variants studied in this paper are summarized in Table 1.

3.2 Motivation of ProRes

ProRes is motivated by several principles.

Table 1: Forward equations of Transformer variants, with (✔) and without (✘) ProRes.

Method	ProRes	Forward Equation
Pre-LN	✘	$x_{l+1} = x_l + \mathcal{F}(\mathrm{Norm}(x_l))$
	✔	$x_{l+1} = x_l + \alpha(l,t) \cdot \mathcal{F}(\mathrm{Norm}(x_l))$
Post-LN	✘	$x_{l+1} = \mathrm{Norm}(x_l + \mathcal{F}(x_l))$
	✔	$x_{l+1} = \mathrm{Norm}(x_l + \alpha(l,t) \cdot \mathcal{F}(x_l))$
Sandwich-LN	✘	$x_{l+1} = x_l + \mathrm{Norm}(\mathcal{F}(\mathrm{Norm}(x_l)))$
	✔	$x_{l+1} = x_l + \alpha(l,t) \cdot \mathrm{Norm}(\mathcal{F}(\mathrm{Norm}(x_l)))$
DeepNorm	✘	$x_{l+1} = \mathrm{Norm}(\alpha \cdot x_l + \mathcal{F}_\beta(x_l))$
	✔	$x_{l+1} = \mathrm{Norm}(\alpha \cdot x_l + \alpha(l,t) \cdot \mathcal{F}_\beta(x_l))$
LayerNorm Scaling	✘	$x_{l+1} = x_l + \mathcal{F}(\mathrm{Norm}(x_l)/\sqrt{l})$
	✔	$x_{l+1} = x_l + \alpha(l,t) \cdot \mathcal{F}(\mathrm{Norm}(x_l)/\sqrt{l})$

Principle 1: Identity behavior at initialization. Deep residual networks benefit from behaving close to identity mapping at initialization, which helps control activation growth and ensures the network has well behaved gradients. Prior work has shown that normalization in residual architectures implicitly biases networks toward identity mappings at initialization [25]. ProRes makes this bias explicit and exact by initializing the residual scaling parameters as $\alpha(l,t) = 0$ when $t = 0$, making identity the exact starting point in the network's optimization trajectory.

Principle 2: Bounded model update with respect to depth and time. Stable optimization in deep Transformers requires controlling the overall magnitude of model updates so that training dynamics remain well behaved as model depth increases. Several prior works [6, 7, 10] established that preserving a bounded model update is critical for training deep Transformers. However, these bounds are primarily derived from model behavior at initialization, where updates are typically the most unstable, and may be unnecessarily conservative when applied uniformly throughout training. In practice, Transformer optimization progresses through distinct phases [17], such as warmup, stable training, and decay [34–37], during which the scale and smoothness of model updates differ substantially. Once the model enters a stable training regime, update constraints designed for initialization may limit learning capacity. Moreover, empirical studies show that shallow layers tend to converge earlier than deeper layers during training [18], which calls for layer-specific treatment. ProRes extends the bounded update principle from initialization to the entire training trajectory in a layerwise and temporal manner. By progressively activating residual branches, ProRes allows shallow layers to stabilize before deeper layers begin to contribute. This encourages representation updates to remain bounded across both depth and training time, stabilizing the warmup phase without sacrificing learning capacity in the stable training regime.

Principle 3: Respecting sequential learning and contribution order. In a Transformer with sequentially stacked layers, deeper layers' inputs depend on shallow layers, while shallow layers' gradients depend on deeper layers. Early in training, allowing deeper layers to introduce large residual modifications can inject noise into intermediate representations and skew gradient signals for shallow layers. ProRes enforces a sequential learning order by delaying the contribution of deeper residual branches, ensuring that deeper layers build upon stable representations rather than amplifying early-stage noise with randomly initialized deep residual branches.

4 Experiments

We evaluate ProRes across a range of pretraining settings to assess its effectiveness, scalability, and generality. First, we conduct pretraining experiments on models ranging from 130M to 1.3B parameters on 50B tokens, comparing and combining ProRes with several Transformer variants. We also scale up training to 7B parameters on Pre-LN in Appendix D. Second, we perform ablations on the residual warmup schedule $\alpha(l,t)$ to study the effect of learning and contribution order across layers. Third, we examine depth scaling behavior by training models with depths ranging from 12 to 120 layers. Finally, in Appendix E, we evaluate ProRes on an alternative pretraining corpus to verify its robustness across data distributions.

Table 2: **Left**: Perplexity (↓) on C4-en test set across model scales. Cells with ProRes are shaded by improvement magnitude. **Right**: Perplexity on WikiText and Lambada for 1.3B models. Final row shows mean perplexity reduction (baseline minus ProRes).

Method	ProRes	130M	350M	1.3B
Pre-LN	✘	14.67	12.36	10.32
	✔	14.30	11.74	9.86
Pre-LN (DS-Init)	✘	14.62	12.24	10.32
	✔	14.29	11.73	9.85
Pre-LN (Scaled Init)	✘	14.63	12.29	10.30
	✔	14.28	11.72	***9.84***
Sandwich-LN	✘	14.55	11.97	10.16
	✔	14.50	11.78	9.94
LayerNorm Scaling	✘	14.45	11.74	9.93
	✔	***14.22***	***11.62***	9.89
Post-LN	✘	14.84	12.74	11.62
	✔	14.72	11.92	10.53
DeepNorm	✘	14.57	12.38	10.32
	✔	14.45	11.97	10.09
Average ΔPPL	–	0.22	0.46	0.42

Method	ProRes	WikiText	Lambada
Pre-LN	✘	25.61	20.46
	✔	23.67	***15.05***
Pre-LN (DS-Init)	✘	25.77	19.83
	✔	23.98	15.57
Pre-LN (Scaled Init)	✘	25.63	20.93
	✔	***23.62***	15.42
Sandwich-LN	✘	24.90	19.77
	✔	24.12	16.09
LayerNorm Scaling	✘	24.28	20.79
	✔	24.19	18.79
Post-LN	✘	30.37	33.53
	✔	26.74	22.91
DeepNorm	✘	25.60	19.72
	✔	24.77	17.15
Average ΔPPL	–	1.58	4.86

4.1 Experiment Setup

Model architecture. All models are decoder-only Transformers with Llama-based Attention and MLP [38], which utilize RMSNorm [39], SwiGLU activation [40] and Rotary Position Embedding [41]. The initialization follows [24] where all layers' weights are initialized from truncated normal distribution $\mathcal{N}(0, 0.02^2)$. Depth-dependent initializations are scaled based on this default initialization, including DS-Init [8], Scaled Init [22], and DeepNorm [10].

Baselines and ProRes schedule. We evaluate ProRes on diverse Transformer variants listed in Table 1, including Pre-LN [5], Post-LN [1], Sandwich-LN [21], DeepNorm [10], and LayerNorm Scaling (LNS) [12]. For Pre-LN, we additionally experiment with two initialization methods, namely DS-Init (as implemented by Gururangan et al. [23]), and Scaled Init. For experiments with ProRes, we adopt the linear schedule in Equation (3) with $T = 1000$. We did not tune this schedule for the main experiment. The effect of schedule choice is studied separately in Section 4.3.

Pretraining data. We primarily train on the C4-en dataset [42, 43] tokenized with the Llama Tokenizer [38]. Unless otherwise stated, all experiments train on a randomly sampled subset with 50B tokens and sequences packed to length of 1024. Evaluation perplexities are calculated on a randomly sampled subset of the C4-en test set with 10M tokens. The set of experiments that train on ClimbMix [44] use the same preprocessing protocol, and evaluation perplexities on ClimbMix are calculated on a held out 10M tokens subset.

Optimizer and learning rates. All experiments use the AdamW optimizer [45, 46] with $\beta = (0.9, 0.95)$, weight decay 0.1, $\epsilon = 10^{-8}$, and gradient clipping 1.0. A global batch size of 512 with 100k training steps is adopted for all runs. We use the Warmup-Stable-Decay learning rate scheduler [34] with 2000 warmup steps and a linear decay to zero in the final 10% steps. Post-LN and DeepNorm use a longer warmup of 10% training steps for better convergence. Learning rates are tuned separately for each configuration with details provided in Appendix A.

Evaluation. Other than perplexity measured on test set of the pretraining corpus, we evaluate the pretrained models on general reasoning benchmarks using LM Evaluation Harness [47]. We report performance on PIQA [48], SIQA [49], HellaSwag [50], WinoGrande [51], ARC-Easy and ARC-Challenge [52], OpenBookQA [53], RACE [54], LAMBADA [55], and MMLU [56]. Perplexity is additionally reported on WikiText [57] and LAMBADA.

Table 3: Zero-shot accuracy (↑) on reasoning benchmarks for 1.3B models.

Method	ProRes	PIQA	SIQA	Hella	Wino	ARC-e	ARC-c	OBQA	RACE	Lambada	MMLU	Avg
Pre-LN	✘	72.85	41.86	52.45	55.41	53.54	26.45	33.40	31.29	39.30	27.69	43.42
	✔	73.34	41.04	56.39	57.54	54.84	28.24	34.20	33.21	42.71	28.44	**45.00**
Pre-LN (DS-Init)	✘	72.58	41.66	52.92	56.35	53.79	27.39	34.20	32.25	39.69	27.79	43.86
	✔	73.45	40.74	56.02	57.77	56.52	27.13	35.60	32.73	42.34	27.80	***45.01***
Pre-LN (Scaled Init)	✘	72.47	39.97	53.65	54.54	52.86	26.11	32.00	32.15	37.78	27.30	42.88
	✔	73.83	41.10	56.48	56.83	56.78	28.07	35.20	31.10	42.34	28.08	**44.98**
Sandwich-LN	✘	72.69	40.94	53.96	57.22	55.98	27.39	33.60	31.77	38.79	27.70	44.00
	✔	72.31	40.99	55.55	56.12	56.06	27.65	34.60	31.87	41.63	28.17	**44.49**
LayerNorm Scaling	✘	73.34	41.30	55.68	56.35	55.77	27.82	33.20	33.59	38.93	28.19	44.42
	✔	73.99	41.25	55.89	55.88	55.72	27.99	35.80	33.01	39.08	27.77	**44.64**
Post-LN	✘	69.97	39.71	47.00	53.43	50.42	24.74	31.40	30.14	33.24	26.24	40.63
	✔	72.74	40.94	51.81	54.62	53.58	27.56	34.40	31.77	37.82	27.56	**43.28**
DeepNorm	✘	72.96	39.76	52.48	54.30	54.08	26.79	34.80	32.15	40.27	27.89	43.55
	✔	73.23	40.43	54.71	53.99	55.89	26.37	34.80	32.73	42.31	27.98	**44.24**
Average ΔAcc	–	+0.86	+0.18	+2.67	+0.73	+1.85	+0.90	+1.71	+0.44	+2.89	+0.43	**+1.27**

4.2 Main Results

For the main experiment, we study the effectiveness of ProRes via pretraining models at scales of 130M, 350M, and 1.3B, and across various Transformer architectures.

Perplexity on the pretraining corpus. Table 2 shows perplexity on the test split of the pretraining corpus. Across all configurations, applying ProRes consistently leads to a notable decrease in perplexity, demonstrating its effectiveness as a general residual modification. Among the evaluated architectures, Post-LN benefits the most from ProRes. We attribute this to the normalization in Post-LN naturally biasing contribution towards deeper layers [5, 29], whereas ProRes corrects for this inherent tendency via sequentially activating shallow to deep residuals. These improvements are consistent across scales and more effective for larger models. In contrast, combining ProRes with LNS yields smaller gains at larger scales. We hypothesize that since LNS already down-weights deeper residuals by $1/\sqrt{l}$, the additional scaling induced by ProRes further suppresses their contribution, delaying effective learning in deeper layers within the fixed 100k training steps.

Performance on reasoning benchmarks. Table 3 reports zero-shot accuracy on several reasoning benchmarks. Overall, ProRes provides consistent improvements across architectures, yielding an average gain of 1.27% over the corresponding baselines. The largest gains are observed on HellaSwag, ARC-Easy, OBQA, and LAMBADA. On LAMBADA, ProRes increases accuracy by 2.89%, indicating enhanced language modeling of long range dependency. The top three average scores are achieved by Pre-LN models with ProRes, and the top five highest averages are all ProRes variants, highlighting the consistent benefit across architectures.

Generalization to out-of-distribution corpora. Table 2 reports perplexity on WikiText and LAMBADA. While ProRes reduces perplexity moderately on the pretraining corpus (ΔPPL ≈ 0.4), it yields substantially larger gains on these out-of-distribution datasets. Notably, the average perplexity reduction reaches 4.86 on LAMBADA, corresponding well with the observed accuracy improvement on this benchmark.

4.3 Warmup Schedule Ablation

We now study the influence of different ProRes schedules $\alpha(l, t)$ on training dynamics and model performance, with the goal of empirically validating the design motivations discussed in Section 3.2. We conduct ablation on a diverse set of schedules listed in Table 4, and across 3 representative Transformer variants: Pre-LN, Post-LN, and Sandwich-LN. For all ablation runs, we pretrain 350M models with 24 layers ($L = 24$) on 6B tokens over 60k training steps, which aligns with the Chinchilla Scaling Law [58]. Perplexities on test set are reported in Table 4. We summarize the main observations regarding layer activation order, warmup length, and schedule design below.

Table 4: **Left**: ProRes warmup schedules. Here $l \in \{1, \ldots, L\}$ denotes the layer index, t the training step, and T the warmup length for the first layer. "Warmup" denotes the number of training steps required for all layers to complete residual warmup. See Appendix F for visualization. **Right**: Ablation of ProRes warmup schedules. Perplexity (↓) on C4 for 350M models trained with 6B tokens. "Warmup" denotes the number of training steps required for all layers to complete residual warmup. Blue indicates improvement while red indicates degrading performance relative to no-ProRes baseline. Perplexities higher than 1000 are marked as *diverged*. **Additional baselines:** DeepNorm (15.71) and LNS (14.63), trained under the same configuration without ProRes.

Schedule	Warmup	$\alpha(l,t)$
linear	$T \cdot L$	$\min\left(\frac{t}{T \times l}, 1\right)$
linear-sqrt	$T \cdot L$	$\left(\min\left(\frac{t}{T \times l}, 1\right)\right)^{1/2}$
linear-square	$T \cdot L$	$\left(\min\left(\frac{t}{T \times l}, 1\right)\right)^{2}$
equal	T	$\min(t/T, 1)$
reverse	$T \cdot L$	$\min\left(\frac{t}{T \times (L-l+1)}, 1\right)$
stagewise-0	$T \cdot L$	$\text{clip}\left(\frac{t-T\times(l-1)}{T}, 0, 1\right)$
stagewise-L	$T \cdot L$	$\text{clip}\left(\frac{t-T\times(l-1)}{T}, 0, 1\right) \cdot (1-1/L)+1/L$
stagewise-$\sqrt{l}$	$T \cdot L$	$\text{clip}\left(\frac{t-T\times(l-1)}{T}, 0, 1\right) \cdot (1-1/\sqrt{l})+1/\sqrt{l}$
fix-L	–	$1/L$
fix-$\sqrt{L}$	–	$1/\sqrt{L}$
fix-$\sqrt{l}$	–	$1/\sqrt{l}$

Schedule	Warmup	Pre-LN	Post-LN	Sandwich-LN
–	–	15.21	16.83	14.76
linear	1k	14.60	15.98	14.63
linear	3k	14.49	15.65	14.56
linear	6k	14.44	15.33	14.51
linear	12k	***14.41***	15.15	14.45
linear	24k	14.43	15.12	14.46
linear	48k	14.57	15.18	***14.44***
linear-sqrt	24k	14.64	15.44	14.59
linear-square	24k	14.46	***14.84***	14.47
equal	1k	14.79	16.29	14.76
equal	12k	15.06	*diverged*	15.12
equal	24k	15.29	*diverged*	15.14
reverse	1k	14.98	*diverged*	14.96
reverse	12k	15.57	*diverged*	15.79
reverse	24k	15.65	*diverged*	16.78
stagewise-0	24k	19.51	*diverged*	18.31
stagewise-L	24k	15.35	15.09	14.81
stagewise-$\sqrt{l}$	24k	14.80	15.48	14.71
fix-L	–	15.77	20.18	15.47
fix-$\sqrt{L}$	–	15.48	17.04	14.99
fix-$\sqrt{l}$	–	14.98	16.21	14.82

Effect of layer contribution order. We observe that the order in which residual branches are activated across depth plays a significant role in optimization. Schedules that activate residuals progressively from shallow to deep layers ("linear") consistently outperform schedules that activate all layers simultaneously[3] ("equal") or prioritize deeper layers early in training ("reverse"). This trend holds across all three architectures, indicating that respecting the sequential dependency structure of stacked Transformer layers is beneficial for learning.

Warmup length and interaction with layer order. The "linear" schedule remains effective across a wide range of warmup lengths, provided the warmup is neither too short ($<$ 1k steps) nor so long that it occupies a substantial fraction of training ($>$ 48k steps). In contrast, the "equal" schedule is considerably more sensitive to warmup length and can lead to degraded performance or even divergence as warmup increases. We conjecture that the "equal" schedule primarily delays chaotic updates at initialization, but fails to respect the sequential dependency across layers; as a result, simultaneously scaling up all residuals may amplify representation and gradient noise. The "reverse" schedule, which activates deeper layers prior to shallow ones, becomes increasingly detrimental as warmup length increases. A longer warmup leads to a clearer separation of learning phases across layers, allowing deeper layers to dominate optimization early in training. Once this imbalance is established, it becomes difficult to correct by reactivating shallow layers later. This behavior mirrors a typical divergence pattern observed in Post-LN Transformers [5, 10], where the last few layers dominate model update, leaving shallow layers with vanishing gradients and limited signal to escape poor local minima.

Static vs. dynamic constraint on model update. In Principle 2 of Section 3.2, we hypothesized that techniques designed to bound model updates at initialization may be unnecessarily conservative once training enters a stable regime. We empirically verify this by observing that the "fix" schedules always perform worse than their "stagewise" counterparts (e.g. "fix-$\sqrt{l}$" and "stagewise-$\sqrt{l}$"), especially for Post-LN. The "fix" schedules constrain model update by scaling down the residual outputs according to model depth throughout the entire training process. "stagewise" variants apply an equivalent

[3]We note that BranchNorm [27] can be viewed as a special case of Post-LN with an "equal" ProRes schedule.

constraint at initialization, but progressively relax it by restoring residual magnitudes from shallow to deep layers during training. This dynamic treatment stabilizes the warmup phase comparably to "fix" schedules, without hurting learning potential in the stable training phase. Moreover, DeepNorm provides a more principled static approach for controlling model updates of Post-LN, yet still underperforms several ProRes schedules, further suggesting that static methods might fail to fully utilize the learning potential in the stable training phase.

Identifying the optimal ProRes schedule. The "linear" schedule is the most robust overall, achieving strong performance across all three architectures and a wide range of warmup lengths, and is the best-performing evaluated schedule for Pre-LN and Sandwich-LN. For Post-LN, "linear-square" and "stagewise-L" perform notably better than "linear", highlighting the importance of a sufficiently gentle introduction of residuals during early training, which corresponds to Principle 1 in Section 3.2. Overall, the optimal schedule is architecture-dependent, with "linear" serving as a strong default choice.

4.4 Depth Scaling Experiment

We investigate how ProRes facilitates residual learning as model depth increases, by comparing its depth scaling behavior against several baseline methods. Results are summarized in Figure 1.

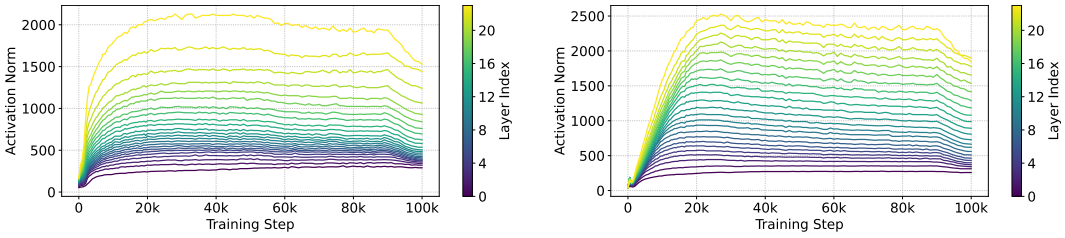
Setup. We start from a base configuration of a 71M-parameter Transformer with 12 layers and scale directly to 24, 48, 72, 96, and 120 layers, keeping all other dimensions fixed. For visual clarity in Figure 1, ProRes is applied only to Pre-LN, the best-performing combination according to Section 4.2, while studies across other normalization schemes are left to future work. Two ProRes schedules are evaluated: "linear" (*Pre-LN (ProRes)*) and "equal" (*Pre-LN (ProRes-equal)*). The same set of no-ProRes baselines from Section 4.1 is adopted, excluding Post-LN. All models are trained under the same setting as Section 4.1, with a learning rate of 2×10^{-3} for all depths except DeepNorm, which required additional search from $\{5 \times 10^{-4}, 1 \times 10^{-3}, 1.5 \times 10^{-3}, 2 \times 10^{-3}\}$ for optimal convergence. For *Pre-LN (ProRes)*, we use the "linear" schedule with $T = 1000$ at depths up to 72, and $T = 500$ for 96 and 120 layers to ensure the warmup length does not exceed the total training steps. No additional tuning is performed for the "linear" schedule. For *Pre-LN (ProRes-equal)*, we found that $T = 1000$ leads to degraded performance at 120 layers, and instead used $T = 12$k for all depths.

Depth scaling performance. Figure 1 shows that *Pre-LN (ProRes)* consistently delivers the best performance among all methods, with the lead becoming more pronounced as depth increases. LNS matches ProRes at depths up to 48 layers, but falls short beyond 72 layers. We hypothesize that the static scaling of LNS, while controlling activation growth at initialization, overly downscales deeper residuals as model depth increases, limiting their contribution and learning potential. This is also observed in the scaling trend for *Pre-LN (DS-Init)*, which initializes deeper layer weights to smaller values, leading to slower perplexity reduction beyond 72 layers. Sandwich-LN enforces a constant norm on residual outputs and provides better scaling potential than LNS at larger depths. Depth-dependent initializations improve the performance of Pre-LN mildly. DeepNorm surpasses Pre-LN from 72 layers onwards but requires careful tuning of learning rates for convergence.

Training stability. To understand how ProRes affects training stability, we measure loss and gradient spikes using the spike score defined by OLMo et al. [24], which quantifies the percentage of data points that are at least seven standard deviations away from a rolling average of the previous 1000 steps. We compute spike scores over the stable training phase (10%–90% of training progress). As shown in Figure 1, applying ProRes maintains near-zero loss spikes as depth increases, indicating that ProRes not only enhances performance but also stabilizes training.

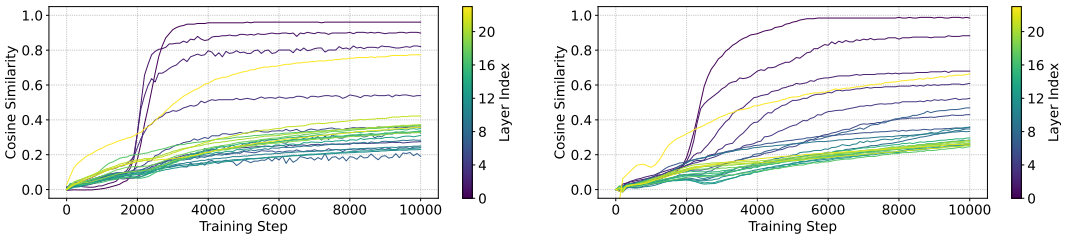
5 Analysis

In this section, we analyze the training dynamics that ProRes introduces. First, we show that applying ProRes naturally mitigates the exponential activation growth often observed in Pre-LN. Second, we study the evolution of layerwise representations during training, and show that ProRes enables smoother and more stable representation updates.



(a) Pre-LN without ProRes (b) Pre-LN with ProRes

Figure 2: Layerwise activation norm of Pre-LN, with and without ProRes.



(a) Pre-LN without ProRes (b) Pre-LN with ProRes

Figure 3: Cosine similarity of layerwise residuals between intermediate (first 10% steps) and final checkpoint.

5.1 Activation Growth in Pre-LN

Prior works have shown that Pre-LN Transformers often exhibit exponential activation growth across depth, rather than a desired linear growth [29, 12], which can limit the effective contribution of deeper layers [59, 60].

We demonstrate that applying ProRes naturally resolves this issue. Figure 2 shows the layerwise activation norms during training for 1.3B models in Section 4.2. In early training (0–20k steps), activation norms of vanilla Pre-LN quickly grows exponentially, while the ProRes variant facilitates a gentle and more linear growth. We attribute this behavior to differences in how residual updates are coordinated across depth. In vanilla Pre-LN, all layers update representations concurrently from initialization, even when upstream representations are still highly unstable. As a result, deeper layers may apply updates based on rapidly changing inputs, while shallow layers are simultaneously influenced by fluctuating gradient signals from deeper layers. By contrast, ProRes delays the contribution of deeper residual branches during early training, allowing shallow layers to first stabilize representations before deeper layers begin refining them. This staged interaction leads to more controlled activation growth across depth.

5.2 Evolution of Layerwise Representations

To further examine how ProRes affects training dynamics, we analyze the evolution of layerwise residual outputs over the course of training. Specifically, we measure the cosine similarity between residual outputs at intermediate checkpoints and those of the final fully trained model. Figure 3 shows that both vanilla Pre-LN and ProRes exhibit earlier convergence in shallow layers. However, the representation evolution under ProRes is notably smoother across layers. In contrast, vanilla Pre-LN shows frequent fluctuations in representation similarity throughout training, indicating less efficient evolution of residual outputs. These observations empirically support our hypothesis that explicitly coordinating residual contributions across depth reduces counterproductive updates and leads to more stable representation learning.

5.3 Comparison with Layerwise Learning-Rate Warmup

When $\alpha(l, t)$ is small, gradients to the corresponding residual-branch parameters are also attenuated. Therefore, one possible explanation is that ProRes acts as an implicit layerwise learning-rate schedule. To test this hypothesis, we compare three settings on 350M models: the standard Pre-LN baseline, ProRes with the linear schedule and $T = 1000$, and a layerwise learning-rate warmup control without forward residual scaling. In the control, parameters in layer l use $l \times 1000$ learning-rate warmup steps, matching the time at which the corresponding ProRes residual reaches full strength. Embedding and language-model-head parameters retain the default 2,000-step warmup, and all other training settings remain unchanged.

Table 5: Comparison with a matched layerwise learning-rate warmup. Perplexity (↓) on C4 for 350M Pre-LN models trained on 6B tokens.

Method	PPL (↓)	ΔPPL vs. Pre-LN
Pre-LN	15.21	–
Pre-LN (layerwise-LR warmup)	15.10	-0.11
Pre-LN (ProRes)	**14.43**	$\mathbf{-0.78}$

As shown in Table 5, the matched layerwise learning-rate warmup recovers 0.11 of the 0.78 perplexity reduction achieved by ProRes, or approximately 14%. This result indicates that layer-dependent learning-rate modulation only account for a small fraction of the improvement, while the remaining gap is likely due to ProRes's intervention on forward activations.

6 Conclusions

We introduced ProRes, a simple and scalable residual warmup scheme that explicitly coordinates layerwise learning over the course of Transformer training. By progressively activating residual contributions across depth, ProRes prioritizes shallow layers early while allowing deeper layers to engage once upstream representations stabilize. Extensive pretraining experiments show that ProRes consistently improves performance and depth scaling across model sizes, initialization methods, and normalization schemes. These results suggest that training-phase-aware residual scheduling is an effective and practical direction for improving Transformer optimization.

Acknowledgments and Disclosure of Funding

This work was partially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project Reference Number: AoE/E-601/24-N).

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[4] Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

[5] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International conference on machine learning*, pages 10524–10533. PMLR, 2020.

[6] Hongyi Zhang, Yann N Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. *arXiv preprint arXiv:1901.09321*, 2019.

[7] Xiao Shi Huang, Felipe Perez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In *International Conference on Machine Learning*, pages 4475–4483. PMLR, 2020.

[8] Biao Zhang, Ivan Titov, and Rico Sennrich. Improving deep transformer with depth-scaled initialization and merged attention. *arXiv preprint arXiv:1908.11365*, 2019.

[9] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the difficulty of training transformers. *arXiv preprint arXiv:2004.08249*, 2020.

[10] Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. Deepnet: Scaling transformers to 1,000 layers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(10):6761–6774, 2024.

[11] Greg Yang, Edward J Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer. *arXiv preprint arXiv:2203.03466*, 2022.

[12] Wenfang Sun, Xinyuan Song, Pengxiang Li, Lu Yin, Yefeng Zheng, and Shiwei Liu. The curse of depth in large language models. *arXiv preprint arXiv:2502.05795*, 2025.

[13] Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F Wong, and Lidia S Chao. Learning deep transformer models for machine translation. *arXiv preprint arXiv:1906.01787*, 2019.

[14] Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. In *Uncertainty in Artificial Intelligence*, pages 1352–1361. PMLR, 2021.

[15] Shufang Xie, Huishuai Zhang, Junliang Guo, Xu Tan, Jiang Bian, Hany Hassan Awadalla, Arul Menezes, Tao Qin, and Rui Yan. Residual: Transformer with dual residual connections. *arXiv preprint arXiv:2304.14802*, 2023.

[16] Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-connections. *arXiv preprint arXiv:2409.19606*, 2024.

[17] Melody Zixuan Li, Kumar Krishna Agrawal, Arna Ghosh, Komal Kumar Teru, Adam Santoro, Guillaume Lajoie, and Blake A Richards. Tracing the representation geometry of language models from pretraining to post-training. *arXiv preprint arXiv:2509.23024*, 2025.

[18] Goker Erdogan, Nikhil Parthasarathy, Catalin Ionescu, Drew A Hudson, Alexander Lerchner, Andrew Zisserman, Mehdi SM Sajjadi, and Joao Carreira. Layerlock: Non-collapsing representation learning with progressive freezing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19461–19470, 2025.

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

[20] Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4246–4253, 2020.

[21] Ming Ding, Zhuoyi Yang, Wenyi Hong, Wendi Zheng, Chang Zhou, Da Yin, Junyang Lin, Xu Zou, Zhou Shao, Hongxia Yang, et al. Cogview: Mastering text-to-image generation via transformers. *Advances in neural information processing systems*, 34:19822–19835, 2021.

[22] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[23] Suchin Gururangan, Mitchell Wortsman, Samir Yitzhak Gadre, Achal Dave, Maciej Kilian, Weijia Shi, Jean Mercat, Georgios Smyrnis, Gabriel Ilharco, Matt Jordan, Reinhard Heckel, Alex Dimakis, Ali Farhadi, Vaishaal Shankar, and Ludwig Schmidt. open_lm: a minimal but performative language modeling (lm) repository, 2023. URL `https://github.com/mlfoundations/open_lm/`. GitHub repository.

[24] Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, et al. 2 olmo 2 furious. *arXiv preprint arXiv:2501.00656*, 2024.

[25] Soham De and Sam Smith. Batch normalization biases residual blocks towards the identity function in deep networks. *Advances in Neural Information Processing Systems*, 33:19964–19975, 2020.

[26] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 32–42, 2021.

[27] Yijin Liu, Xianfeng Zeng, Fandong Meng, and Jie Zhou. Branchnorm: Robustly scaling extremely deep transformers. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11675–11687, 2024.

[28] Tianhao Chen, Xin Xu, Zijing Liu, Pengxiang Li, Xinyuan Song, Ajay Kumar Jaiswal, Fan Zhang, Jishan Hu, Yang Wang, Hao Chen, et al. Gpas: Accelerating convergence of llm pretraining via gradient-preserving activation scaling. *arXiv preprint arXiv:2506.22049*, 2025.

[29] Pengxiang Li, Lu Yin, and Shiwei Liu. Mix-ln: Unleashing the power of deeper layers by combining pre-ln and post-ln. *arXiv preprint arXiv:2412.13795*, 2024.

[30] Tianyu Li, Dongchen Han, Zixuan Cao, Haofeng Huang, Mengyu Zhou, Ming Chen, Erchao Zhao, Xiaoxi Jiang, Guanjun Jiang, and Gao Huang. Siamesenorm: Breaking the barrier to reconciling pre/post-norm. *arXiv preprint arXiv:2602.08064*, 2026.

[31] Zhenda Xie, Yixuan Wei, Huanqi Cao, Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Jiang Chang, Kuai Yu, et al. mhc: Manifold-constrained hyper-connections. *arXiv preprint arXiv:2512.24880*, 2025.

[32] Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tieyan Liu. Efficient training of bert by progressively stacking. In *International conference on machine learning*, pages 2337–2346. PMLR, 2019.

[33] Cheng Yang, Shengnan Wang, Chao Yang, Yuechuan Li, Ru He, and Jingqiao Zhang. Progressively stacking 2.0: A multi-stage layerwise training method for bert training speedup. *arXiv preprint arXiv:2011.13635*, 2020.

[34] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

[35] Atli Kosson, Bettina Messmer, and Martin Jaggi. Analyzing & reducing the need for learning rate warmup in gpt training. *Advances in Neural Information Processing Systems*, 37:2914–2942, 2024.

[36] Kaiyue Wen, Zhiyuan Li, Jason Wang, David Hall, Percy Liang, and Tengyu Ma. Understanding warmup-stable-decay learning rates: A river valley loss landscape perspective. *arXiv preprint arXiv:2410.05192*, 2024.

[37] Aleksandr Dremov, Alexander Hägele, Atli Kosson, and Martin Jaggi. Training dynamics of the cooldown stage in warmup-stable-decay learning rate scheduler. *arXiv preprint arXiv:2508.01483*, 2025.

[38] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[39] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

[40] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[41] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

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

[43] Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. *arXiv preprint arXiv:2104.08758*, 2021.

[44] Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan Su, Markus Kliegl, Zijia Chen, Peter Belcak, Yoshi Suhara, Hongxu Yin, et al. Climb: Clustering-based iterative data mixture bootstrapping for language model pre-training. *arXiv preprint arXiv:2504.13161*, 2025.

[45] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[46] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[47] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL `https://zenodo.org/records/12608602`.

[48] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

[49] Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.

[50] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

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

[52] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[53] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

[54] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

[55] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 1525–1534, 2016.

[56] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[57] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[58] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[59] Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A Roberts. The unreasonable ineffectiveness of the deeper layers. *arXiv preprint arXiv:2403.17887*, 2024.

[60] Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 20192–20204, 2025.

A Learning Rate Tuning

In this section, we detail the learning rate tuning for the main experiments in Section 4.1. We first note that the final performance of the WSD learning rate scheduler [34] is not quite sensitive to learning rate choices. A smaller learning rate tends to have a lower loss during the constant learning rate phase, but a larger learning rate would catch up after the decay stage, as long as it trains stably. A similar phenomenon is discussed in [24]. With this property in mind, our goal is to find a preferably larger learning rate such that training takes place smoothly, without notable loss spikes or gradient spikes. We only tune the learning rates for no-ProRes baselines, and directly apply their tuned learning rates to ProRes variants.

For the 1.3B models, we searched over learning rates $\{1 \times 10^{-3}, 6 \times 10^{-4}, 3 \times 10^{-4}\}$. We selected 6×10^{-4} for all baselines except Post-LN and DeepNorm, for which 3×10^{-4} yielded more stable training.

For the 350M models, we evaluated $\{5 \times 10^{-4}, 1 \times 10^{-3}, 1.5 \times 10^{-3}\}$ and adopted 1×10^{-3} for all models except Post-LN, which performed best with 5×10^{-4}.

For the 130M models, we considered $\{1 \times 10^{-3}, 1.5 \times 10^{-3}, 2 \times 10^{-3}\}$. The final choices were 1×10^{-3} for Post-LN, 1.5×10^{-3} for DeepNorm, and 2×10^{-3} for the remaining baselines.

On intermediate learning rates. We find that using an more fine-grained learning rate search grid can be unnecessary under the WSD LR scheduler. For example, for the 1.3B Pre-LN baseline, using learning rate 4.5×10^{-4} instead of 6×10^{-4} increases the final perplexity by only 0.02, which is negligible (and harmful to performance) compared to the magnitude of improvements from ProRes.

B Relation to Progressive Training

We clarify the distinction between ProRes and progressive training approaches such as stacking [32, 33] or freezing [18].

Conceptual differences. Progressive training methods (e.g., stacking) typically aim to improve training efficiency by gradually increasing model capacity over time, often in multi-epoch settings where data can be revisited. In contrast, ProRes operates at fixed model capacity and instead modulates residual contributions across depth to improve optimization and data efficiency. In this sense, ProRes is more closely related to initialization and normalization strategies that shape training dynamics, rather than methods that alter model capacity during training.

Training regime. A key practical difference lies in the training regime. Progressive stacking is commonly studied in multi-epoch settings, whereas modern large-scale language model pretraining is often conducted in a single-epoch regime. Under this constraint, reducing model capacity in early stages may limit the model's ability to fully utilize the available data, with effects that cannot be recovered later.

Empirical comparison. To better understand these differences, we implement progressive stacking [32] under the same pretraining setup as our main experiments (350M parameters, 50B tokens). Results are shown in Table 6.

Table 6: Comparison with progressive stacking under the same pretraining setup.

Method	PPL (↓)
Pre-LN	12.36
Pre-LN (ProRes)	11.74
Pre-LN (Stacking)	12.49
Post-LN	12.74
Post-LN (ProRes)	11.92
Post-LN (Stacking)	*diverge*

Stacking does not improve final perplexity in this setting and leads to instability for Post-LN. A plausible explanation is that changing model capacity during training introduces abrupt shifts in the residual pathways, which can disrupt optimization. In contrast, ProRes preserves model capacity throughout training and instead provides a smooth, coordinated adjustment of layerwise contributions.

Overall, these results suggest that ProRes and progressive stacking address different aspects of training: the former focuses on improving optimization dynamics at fixed capacity, while the latter targets efficiency via capacity scheduling.

C Additional Baselines on Residual Scaling

We report results for several residual scaling and initialization methods that are conceptually related to ProRes but were not included in the main text.

Experimental setup. All methods are evaluated under the same pretraining configuration as in our main experiments in Section 4.1 (350M parameters, 50B tokens).

Table 7: Comparison with additional residual scaling and initialization methods under the same pretraining setup (350M, 50B tokens).

Method	Perplexity (↓)
Pre-LN	12.36
Pre-LN (ProRes)	11.74
LayerScale [26]	12.27
ReZero [14]	*diverge*
SkipInit [25]	*diverge*
R-Fixup [6]	*diverge*
T-Fixup [7]	*diverge*

Results. Table 7 summarizes the results. LayerScale provides a modest improvement over the Pre-LN baseline, while ProRes yields a larger gain under the same setting. Other approaches (ReZero, SkipInit, R-Fixup, T-Fixup) aim to replace normalization with better initialization, and we found that they are unstable in BFloat16 training and do not produce valid runs, particularly at longer sequence lengths. In contrast, ProRes is designed to operate within standard normalized architectures and maintains stable training behavior in our experiments.

More on ReZero divergence. We made multiple attempts to converge the ReZero run, including LR and warmup search, tuning weight initialization, and using higher precision training. All of theses attempts failed to converge. To localize the failure, we reran ReZero with per-operation non-finite checks in an unfused attention implementation. We found that ReZero tend to produce much larger attention logits, which is likely due to the lack of normalization before the attention module. These growing logits ultimately cause NaN loss during training. Similar phenomenon is observed in SkipInit, R-Fixup, and T-Fixup, all of which replaced normalization with tuned initialization.

D Pretrain Experiments on 7B parameters

Table 8: Zero-shot accuracy (↑) on reasoning benchmarks for 7B models.

Method	PIQA	SIQA	Hella	Wino	ARC-e	ARC-c	OBQA	RACE	Lambada	MMLU	Avg
Pre-LN	76.01	**42.32**	63.34	58.09	62.37	29.86	36.80	34.55	47.45	29.43	48.02
Pre-LN (ProRes)	**76.82**	41.97	**65.23**	**59.83**	**63.43**	**30.72**	**39.20**	**35.60**	**50.09**	**30.40**	**49.33**

We conduct pretraining experiments on 7B parameter models following the settings in Section 4.1, using a learning rate of 3×10^{-4}. Vanilla Pre-LN and Pre-LN with ProRes are evaluated. For the

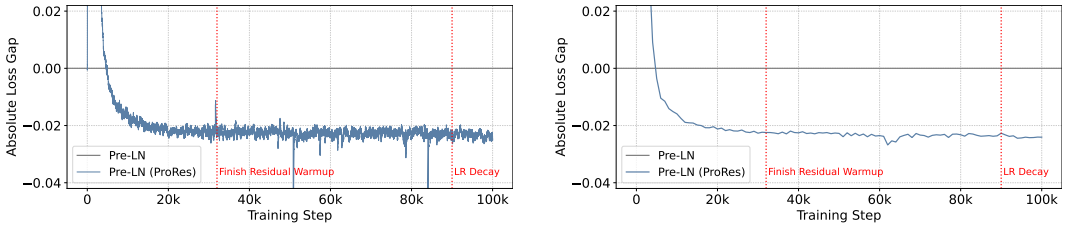


Figure 4: Loss gap across training steps of 7B models. **Left:** training loss gap (0.99 EMA smoothed) between Pre-LN and its ProRes variant. **Right:** evaluation loss gap calculated every 1000 steps.

Table 9: Perplexity (↓) on ClimbMix test split across model scales.

Method	ProRes	130M	350M	1.3B
Pre-LN	✘	11.11	9.43	8.38
	✔	10.82	9.13	8.09

ProRes variant, we adopt the linear schedule in Equation (3) with $T = 1000$, which corresponds to a residual warmup length of 32,000 steps given 32 layers in total.

Figure 4 shows the loss gaps during training. We note that ProRes has higher loss initially due to its restricted model updates when $\alpha(l, t)$ is small. As training proceeds, ProRes gradually unlocks learning potentials of each layer and achieves lower loss than the baseline. The loss gap keeps increasing steadily even after all residuals have finished warming up. Moreover, ProRes is able to maintain and even slightly increase its advantage after the learning rate decay stage.

Table 8 reports downstream evaluation at the final checkpoints. ProRes improves the average zero-shot accuracy from 48.02 to 49.33 and improves 9 of the 10 individual tasks. Because 50B training tokens are below the compute-optimal budget for a 7B model [58], these results should be interpreted as evidence from an undertrained setting rather than a production-scale evaluation.

E Pretrain Experiments on Alternative Corpus

To verify that ProRes generalizes well to pretraining corpus other than C4, we conduct a set of experiments on the ClimbMix dataset [44]. We adopt the same preprocessing pipeline in Section 4.1. Evaluation perplexities on held out test set are reported in Table 9.

F Visualization of ProRes Schedules

Figure 5 shows several ProRes schedule variants.

G Use of Compute Resources

All experiments were conducted on a single node of $8 \times$ H800 GPUs. Training under the settings in Section 4.1 required approximately 260h for 7B, 100h for 3B, 50h for 1.3B, 20h for 350M, and 7h for 130M models.

H License

We provide licenses of assets used in our paper, including model, dataset and code.

- C4 dataset [43]: This is the dataset we used in our pretraining experiments. It's released under the Open Data Commons License Attribution family License.
- ClimbMix dataset [44]: This is the dataset we used in Appendix E, released under CC-BY-NC-4.0.

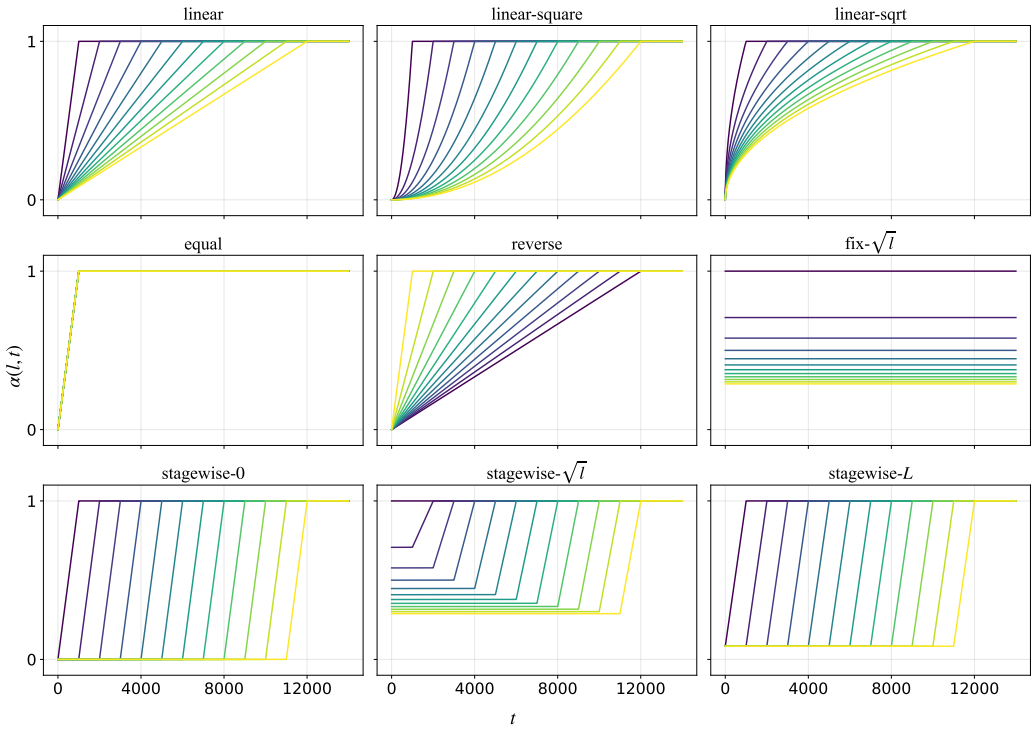


Figure 5: Visualization of various schedules in Table 4. We use $T = 1000$, $L = 12$ for visualizing $\alpha(l, t)$. Darker lines indicate shallow layers while brighter lines indicate deeper layers.

I Limitations and Future Work

Limitations. Our current scheduling strategy is based on simple, interpretable heuristics; while effective in our experiments, more adaptive variants that leverage training signals may further improve performance. In addition, we focus on a representative set of model scales, architectures, and training configurations; evaluating broader regimes (e.g., larger scales or alternative architectures such as MoE) would help further establish generality. Finally, although we provide initial insights into training dynamics, a more detailed characterization could deepen understanding of the mechanisms underlying the observed gains.

Future Work. An interesting direction is to develop self-adaptive warmup strategies that adjust layerwise contributions online during training. Another direction is to design metrics for online convergence estimation, which could enable more principled and automated schedule control.

J Impact Statement

This paper presents a method for improving the optimization and depth scaling of Transformer-based language models. The primary goal of this work is to advance the understanding and design of scalable training techniques in machine learning. While improved training stability and efficiency may facilitate the development of larger and more capable models, the broader societal impacts are consistent with those of existing large language models. We do not identify any new ethical concerns or societal risks that are unique to the techniques proposed in this work.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We proposed a method to improve training dynamics of Transformers, which is then extensively validated through our experiments.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussed limitations and future work in Appendix I.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: This paper does not cover theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide reproduction code in the supplementary materials.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Our reproduction code in supplementary materials provides data and code.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: All details are specified in the paper and submitted code.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Providing error bars for pretraining experiments is prohibitively expensive and out of our compute budget. However, we do provide a thorough set of experiments across various model scales and architectures that shows the significance and generality of our method.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide compute resources information in Appendix G.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We respectfully follow the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss potential impacts in Appendix J.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [No]

    Justification: Our models are trained with extensively cleaned public pretraining data (C4).

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We provide license information in Appendix H.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: Our paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: Our paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: Our paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: Our core method does not rely on LLM usage.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.