DELAYED DEPTH EXPANSION FOR LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped Transformers recently have gained attention for increasing computational depth by repeatedly applying weight-shared Transformers blocks. While improving parameter efficiency, it requires heavy recurrent computation for every token during training. We ask whether this recurrent depth must be paid for throughout the entire training pipeline. To this end, we introduce *Delayed Recurrent Depth Expansion (DDE)*, which first trains a shallower dense Transformer and introduces parameter-shared recurrent depth only later in training. Directly converting a trained dense model to repeated computation causes substantial functional disruption, so we use *step-relaxed recurrence* to provide a controlled dense-to-recurrent transition. Across 155M and 314M parameter models, DDE substantially reduces cumulative Transformer-block FLOPs at matched validation loss compared to training the corresponding recurrent architecture from scratch. Step relaxation notably reduces the instantaneous loss increase at conversion, with the advantage growing at larger recurrence depths, while remaining competitive with conventional looping when trained from scratch. We further show that the same conversion mechanism can be applied to pretrained language models across multiple model families and billion-parameter scales. These results demonstrate that the recurrent computation used at convergence need not be paid for throughout training, providing a simple route to more compute-efficient training of looped Transformers.

1 INTRODUCTION

Increasing computational depth is a fundamental way to enhance the capacity of Transformer models (Vaswani et al., 2017; Wang et al., 2022). While conventional Transformers increase depth by introducing additional parameterized layers, a recently emerging field, looped Transformers, provide an alternative, where a shared set of Transformer blocks can be applied repeatedly, increasing effective computational depth without proportionally increasing the number of unique parameters (Dehghani et al., 2019; Giannou et al., 2023; Geiping et al., 2025; Bae et al., 2025b). This separation between parameter count and computational depth makes recurrent architectures an appealing direction for improving the parameter efficiency of language models.

Looped Transformers make depth parameter-efficient, but not necessarily training-efficient. A model that applies a shared block stack K times still evaluates that stack K times for every training token. Existing recurrent-depth models typically instantiate their recurrent computation throughout training (Geiping et al., 2025; Bae et al., 2025b; Li et al., 2026). Related work reduces pretraining cost through progressive computation or model growth (Zhang & He, 2020; Chen et al., 2026b; Shen et al., 2022; Wang et al., 2023; Du et al., 2024; Wang et al., 2025), but these approaches either sparsify computation or expand the physical model. We instead ask whether full recurrent depth can be introduced only after a dense model has already been partially trained.

We investigate this question through **Delayed Recurrent Depth Expansion (DDE)**. Rather than training the final recurrent architecture from the first token, DDE first optimizes a shallower dense Transformer, allowing early training tokens to be processed with substantially less computation. At an intermediate point in training, we expand computational depth within a middle-loop design, where unshared prefix and suffix blocks flank a recurrent middle stack, by repeatedly evaluating the already-trained middle blocks with shared parameters and continuing optimization under the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

resulting recurrent architecture. Unlike progressive model growth, DDE keeps the number of unique Transformer parameters fixed and increases only recurrent computational depth.

Introducing recurrence into an already-trained dense network, however, creates a new challenge. A middle stack M optimized to be evaluated once can undergo a substantial functional change when it is abruptly replaced by repeated applications M^K. Indeed, we find that naive conversion causes a large instantaneous increase in validation loss, and that this disruption grows rapidly with recurrent depth. To enable controlled depth expansion, we adopt a *step-relaxed recurrence*, where each recurrent evaluation takes only a fractional residual step, $h_{k+1} = h_k + \frac{1}{K}\left(M(h_k) - h_k\right)$. This fractional residual update construction, previously used in relaxed looping and residual scaling (Chen et al., 2026a; Wang et al., 2026), divides the transformation across recurrent steps while keeping their nominal cumulative step size fixed. We use this property not as a new recurrence formulation, but as a mechanism for transitioning a trained dense model into deeper shared computation. Empirically, step relaxation substantially reduces the architectural shock caused by depth expansion.

DDE thereby decouples learning with recurrent depth from incurring its full computational overhead throughout training. We evaluate this strategy through the cumulative training compute required to reach a matched validation loss. This motivates a compute-centric view of recurrent training in which the relevant quantity is not simply the number of training tokens, but the cumulative computation required to reach a given model quality.

Our main contributions are:

- We introduce **Delayed Recurrent Depth Expansion (DDE)**, a training strategy that first learns with a shallow dense Transformer and introduces parameter-shared recurrent depth later in training. Unlike progressive layer growth, DDE increases computational depth while keeping the number of unique Transformer parameters fixed.
- We show that **step relaxation enables stable dense-to-recurrent conversion**. Across model scales, expansion points, and recurrence depths, relaxed conversion substantially reduces the instantaneous loss increase induced by naive recurrence, with the advantage becoming more pronounced as recurrent depth increases.
- We demonstrate that **recurrent depth need not be paid for throughout pretraining**. Across 155M and 314M parameter models, DDE reduces cumulative Transformer-block FLOPs at matched validation loss compared with training the corresponding recurrent architecture from scratch. We further demonstrate that the same conversion mechanism can be applied to pretrained language models across multiple model families and billion-parameter scales.

2 Related Work

Looped and recurrent-depth Transformers. Parameter sharing across depth has long been used to decouple computational depth from the number of unique parameters, from Universal Transformers (Dehghani et al., 2019) and ALBERT (Lan et al., 2020) to more recent looped and recurrent-depth Transformers (Giannou et al., 2023; Geiping et al., 2025; Bae et al., 2025b). Recent work further studies the optimization and stability of repeated parameter visits at increasing recurrent depth (Wang et al., 2026; Li et al., 2026). These works primarily ask how recurrent computation should be parameterized, stabilized, or allocated. We instead ask whether the target recurrent depth must be used throughout the entire training trajectory.

Dense-to-recurrent conversion. Several recent methods retrofit recurrent computation onto pretrained dense Transformers. Relaxed Recursive Transformers (Bae et al., 2025a) tie pretrained layers while introducing depth-specific LoRA modules to relax strict parameter sharing, whereas LoopCoder (Yang et al., 2026) converts a dense checkpoint into a recurrent block before further looped pretraining. Most closely related to our conversion mechanism, Training-Free Looped Transformers (TFLT) (Chen et al., 2026a) applies repeated computation to a contiguous middle stack of a frozen pretrained model and uses numerical sub-stepping, including damped Euler updates, to preserve the transformation expected by the original network without any additional training. Thus, neither dense-to-recurrent conversion nor step-relaxed recurrence itself is our contribution.

Progressive model growth. DDE is also related to progressive training methods that begin with cheaper models and increase capacity during optimization. Net2Net introduced function-preserving transformations for growing trained networks (Chen et al., 2016), while progressive stacking and Transformer growth progressively increase depth or other architectural dimensions during pretraining (Gong et al., 2019; Gu et al., 2021). The key distinction is that these methods grow the physical network toward a larger model, whereas DDE increases only the number of evaluations of an already-existing shared middle stack. Consequently, DDE expands computational depth while keeping the number of unique Transformer-block parameters fixed. Our work therefore connects progressive training with recurrent-depth models by asking whether expensive recurrent computation can itself be delayed until later in optimization.

3 BACKGROUND

Looped Transformers. Consider a decoder-only Transformer decomposed into an initial stack P, a middle stack M, and a final stack C. A standard dense model evaluates the middle stack once,

$$h_0 = P(\mathrm{Emb}(x)), \qquad h_1 = M(h_0), \qquad z = \mathrm{LMHead}(C(h_1)). \tag{1}$$

A looped Transformer instead reuses the same middle stack for K recurrent evaluations,

$$h_{k+1} = M(h_k), \qquad k = 0, \ldots, K-1. \tag{2}$$

If M contains L_M blocks, increasing K increases effective computational depth by repeatedly evaluating those blocks without introducing a distinct parameter set at every recurrence. The benefit in parameter efficiency therefore comes with additional per-token computation.

Step-relaxed recurrence. Rather than applying the full middle-stack transformation at every recurrence, step-relaxed looping (Chen et al., 2026a) expresses its residual transformation as $G_M(h) = M(h) - h$, and applies

$$h_{k+1} = h_k + \frac{1}{K} G_M(h_k) = h_k + \frac{1}{K}\left(M(h_k) - h_k\right). \tag{3}$$

The cumulative nominal step size therefore remains $\sum_{k=0}^{K-1} 1/K = 1$ as recurrent depth increases. We use this existing relaxed-looping construction as the basis for introducing recurrent depth into an already-trained dense model.

4 METHOD

Looped Transformers increase computational depth by repeatedly applying a shared stack of Transformer blocks. While parameter sharing avoids introducing a distinct parameter set at every effective depth, training a deeply looped model from the beginning requires paying the cost of all recurrent evaluations for every training token. We instead ask whether recurrent depth is necessary throughout the entire optimization process.

Our approach delays recurrent depth expansion until after an initial stage of dense training. As shown in Fig. 1, we first train a shallower dense Transformer. At an intermediate point in training, we convert the already-trained middle stack into a recurrent computation using a step-relaxed update, and continue optimization with the expanded computational depth. This allows early tokens to be processed by the cheaper dense architecture while reserving the more expensive recurrent computation for later stages of training.

4.1 FROM DENSE DEPTH TO RECURRENT DEPTH

Consider a decoder Transformer partitioned into three contiguous components, $F = C \circ M \circ P$, where P is an unshared *Pre* stack, M is a middle stack, and C is an unshared *Coda* stack. Let $M = B_{b+L_M-1} \circ \cdots \circ B_{b+1} \circ B_b$ contain L_M consecutive Transformer blocks. For an input sequence x, the dense model computes

$$h_0 = P(\mathrm{Emb}(x)), \qquad h_1 = M(h_0), \qquad z(x) = \mathrm{LMHead}(C(h_1)). \tag{4}$$

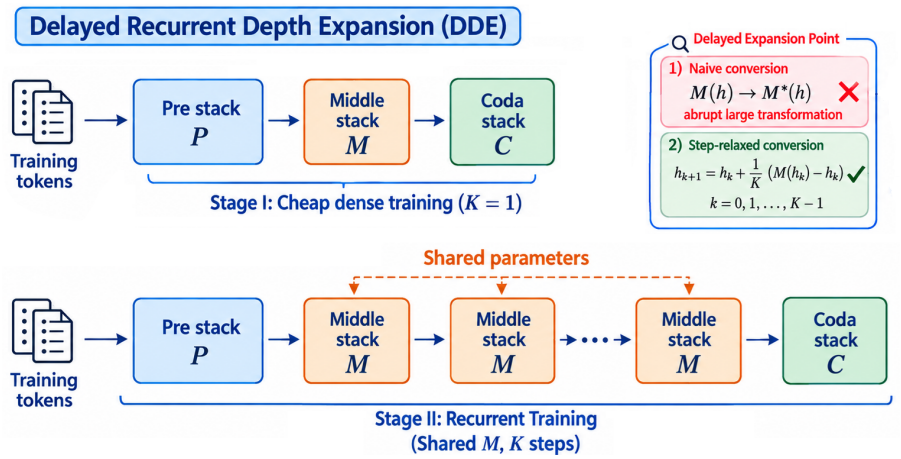


Figure 1: **Overview of Delayed Recurrent Depth Expansion (DDE).** DDE first trains the model densely and introduces parameter-shared recurrent depth only at a later expansion point. Naively repeating the trained middle stack causes an abrupt functional change, whereas step-relaxed recurrence distributes it across K updates, enabling a smoother dense-to-recurrent transition.

A conventional looped Transformer increases effective depth by repeatedly applying the entire middle stack, $h_{k+1} = M(h_k), k = 0, \ldots, K-1$. The same L_M middle blocks are therefore evaluated K times while sharing their parameters across all recurrences. The resulting model performs

$$L_P + KL_M + L_C \tag{5}$$

Transformer-block evaluations while retaining only $L_P + L_M + L_C$ unique blocks.

This parameter reuse makes recurrent depth attractive, but it also makes training more expensive: a model trained with recurrence count K from the first token incurs the KL_M middle-block evaluations throughout optimization. Our central strategy is therefore to postpone this computational expansion until later in training.

4.2 Step-Relaxed Depth Expansion

Directly replacing a single middle-stack evaluation by K full evaluations changes the computation from M to $M^K \equiv M \circ M \circ \cdots \circ M$. For a middle stack that has already been optimized as part of a dense model, this introduces an abrupt change in the transformation between the Pre and Coda.

To obtain a controlled transition, we adopt the Euler-style relaxed looping construction used in prior work such as TFLT (Chen et al., 2026a). Define the residual transformation of the middle stack as

$$G_M(h) = M(h) - h. \tag{6}$$

For a target recurrence count K, we replace the single dense middle-stack evaluation by

$$\begin{aligned} h_{k+1} &= h_k + \eta_K G_M(h_k) \\ &= (1-\eta_K)h_k + \eta_K M(h_k), \qquad \eta_K = \frac{1}{K}, \end{aligned} \tag{7}$$

for $k = 0, \ldots, K-1$.

When $K = 1$, equation 7 exactly recovers the original dense computation, $h_1 = M(h_0)$. For $K > 1$, each recurrence moves only partway toward the transformation proposed by M, while the same middle-stack parameters are reused at every step. The cumulative step coefficient remains fixed,

$$\sum_{k=0}^{K-1} \eta_K = K\eta_K = 1, \tag{8}$$

rather than scaling proportionally with K as in conventional recurrence.

This construction can be viewed as a forward-Euler discretization of

$$\frac{dh(t)}{dt} = G_M(h(t)), \qquad t \in [0, 1], \tag{9}$$

with step size $1/K$. We use this interpretation to motivate a controlled transition from the original middle-stack transformation to repeated shared computation.

4.3 Dense-First Recurrent Training

Let G denote the token position at which recurrent depth is introduced, and let T denote the total number of training tokens. Training proceeds in two stages.

Stage I: Dense training. For the first G tokens, we optimize the original dense model, $P \to M \to C$, corresponding to $K = 1$. During this stage, each token incurs only one evaluation of the middle stack. All model parameters are trained normally.

Stage II: Recurrent depth expansion. At token G, we retain all learned parameters and replace the single application of M by K parameter-shared relaxed evaluations,

$$P \to \underbrace{\Phi_{M,1/K} \to \cdots \to \Phi_{M,1/K}}_{K \text{ recurrent steps}} \to C \tag{10}$$

where $\Phi_{M,1/K}(h) = h + \frac{1}{K}\big(M(h) - h\big)$. No new Transformer blocks are introduced at conversion. Specifically, P, M, and C retain the parameters learned during Stage I, and every recurrent evaluation shares the same M. We then continue training all parameters under the expanded recurrent computation for the remaining $T - G$ tokens.

4.4 Compute Efficiency of Delayed Depth Expansion

The main computational advantage of our training strategy comes from delaying the expensive recurrent computation. Before depth expansion, each training token is processed by the original dense model. After expansion, each token is processed using the deeper recurrent computation. Thus, the final recurrent model has the same computational cost as its from-scratch counterpart. DDE saves training compute by using the cheaper dense model during the early stage of training.

We measure this advantage using *cumulative training FLOPs at matched validation loss*. Consider a dense-to-looped run that introduces recurrent depth after G training tokens and reaches a target validation loss $\mathcal{L}^\star$ after N_{expand} total tokens. Its cumulative training cost is

$$C_{\text{expand}}(\mathcal{L}^\star) = C_{\text{dense}}(0, G) + C_{\text{loop}}(G, N_{\text{expand}}), \tag{11}$$

where $C_{\text{dense}}(a, b)$ denotes the FLOPs required to train the dense model from token position a to b, and $C_{\text{loop}}(a, b)$ denotes the FLOPs required to train the target looped model over the same token interval.

For comparison, let N_{scratch} be the first token position at which the same looped architecture, trained recurrently from the beginning, reaches $\mathcal{L}^\star$. Its cumulative training cost is simply $C_{\text{scratch}}(\mathcal{L}^\star) = C_{\text{loop}}(0, N_{\text{scratch}})$. We report the relative FLOPs saving as

$$S_{\text{FLOPs}}(\mathcal{L}^\star) = 1 - \frac{C_{\text{expand}}(\mathcal{L}^\star)}{C_{\text{scratch}}(\mathcal{L}^\star)}. \tag{12}$$

5 Experiments

Our experiments are organized around five questions. First, we test whether step-relaxed recurrence remains competitive with conventional looping when trained from scratch (§ 5.2). Second, we ask whether recurrent depth can be introduced partway through training to reduce the cumulative compute required to reach a matched validation loss(§ 5.3). Third, we examine whether step relaxation limits the immediate functional disruption caused by dense-to-recurrent conversion, and how this effect changes with expansion point and recurrence depth(§ 5.4.1). Fourth, we study whether the reduced conversion shock leads to more stable and effective post-expansion training(§ 5.4.2). Finally, we test whether the same conversion mechanism transfers to pretrained language models across multiple model families and scales(§ 5.5).

Table 1: From-scratch comparison between conventional and step-relaxed looped Transformers at the 155M and 314M scales. Both models use the same unique parameters and recurrence count. Step relaxation matches or slightly improves conventional looping at both tested depths.

Model	Params	**Val PPL** ↓	PIQA ↑	ARC-E ↑	ARC-C ↑	HS ↑	WG ↑	CSQA ↑	OBQA ↑	SciQ ↑	**Avg.** ↑
Dense 6L	155M	19.37	0.6469	0.5707	0.2619	0.3477	0.5233	0.1900	0.3120	0.8130	0.4582
K4 Naive	155M	18.05	0.6496	0.5846	**0.2688**	**0.3740**	0.5036	0.1941	**0.3200**	0.8260	0.4651
K4 Ours	155M	**17.96**	**0.6523**	**0.5867**	**0.2688**	0.3713	**0.5343**	**0.1957**	0.3180	**0.8320**	**0.4699**
K8 Naive	155M	17.63	0.6529	**0.5909**	0.2688	0.3819	0.5272	0.1933	**0.3460**	**0.8400**	0.4751
K8 Ours	155M	**17.57**	**0.6594**	0.5838	**0.2773**	**0.3824**	**0.5335**	**0.1957**	0.3440	0.8360	**0.4765**
K4 Naive	314M	14.86	**0.6774**	0.6271	**0.3046**	0.4413	**0.5438**	**0.1949**	0.3320	0.8680	0.4986
K4 Ours	314M	**14.79**	0.6741	**0.6288**	0.3038	**0.4453**	0.5296	0.1876	**0.3500**	**0.8780**	**0.4996**
K8 Naive	314M	14.49	0.6768	0.6330	0.3046	0.4609	0.5572	0.1974	0.3460	**0.8650**	0.5051
K8 Ours	314M	**14.42**	**0.6817**	**0.6338**	**0.3089**	**0.4645**	**0.5699**	**0.2080**	**0.3560**	0.8450	**0.5085**

5.1 Experimental Setup

Models and Recurrence Rules. We study Llama-style decoder-only Transformers (Dubey et al., 2024) trained from scratch at two primary scales, 155M and 314M parameters, as well as across diverse open-source pretrained model families spanning multiple capacities (Qwen3-0.6B-Base (Yang et al., 2025), Llama-3.2-1B (Dubey et al., 2024), Llama-3.2-3B (Dubey et al., 2024), and OLMo-3-7B (Team OLMo et al., 2025)).

Dataset and Pretraining. Across all from-scratch experiments, we train on the 100BT sample of FineWeb (Penedo et al., 2024) using 1,024-token unpadded sequences and a batch size of 256 sequences (262,144 tokens). Detailed training configurations and hyperparameters are provided in Table 6.

Compute Accounting and Evaluation. Following standard language-model training compute accounting (Kaplan et al., 2020), we report cumulative training FLOPs. To isolate the computation affected by recurrent depth, we count active Transformer-block evaluations and exclude embedding and LM-head projections, which are identical across recurrence configurations. For compute-efficiency experiments, we use *iso-loss training compute*: the cumulative Transformer-block FLOPs required to reach a matched validation loss $L^\star$.

5.2 Relaxed Recurrence Matches Conventional Looped Transformers

Before examining delayed depth expansion, we first verify that step-relaxed recurrence remains competitive with conventional looping when both are trained from scratch. This comparison is important because DDE ultimately produces a step-relaxed recurrent architecture, since compute savings would be less meaningful if this target architecture were systematically weaker than conventional recurrence.

We therefore train conventional and step-relaxed looped Transformers from scratch under the same training configuration and parameter budget. The conventional model repeatedly applies the shared middle stack as $u_{k+1} = M(u_k)$, whereas the relaxed model uses $u_{k+1} = u_k + \frac{1}{K}\left(M(u_k) - u_k\right)$.

Table 1 summarizes language-modeling perplexity and zero-shot downstream performance. Across both parameter scales and recurrence depths, step relaxation achieves comparable or slightly better validation perplexity than conventional looping, while maintaining similar or slightly higher average downstream accuracy. These results indicate that the relaxed recurrence provides a competitive target architecture for the delayed-expansion experiments that follow.

5.3 Delayed Depth Expansion Reduces Iso-Loss Training FLOPs

Having established that step-relaxed recurrence provides a competitive target architecture, we next ask whether its full recurrent depth must be used throughout training. We compare delayed depth expansion against recurrent training from scratch using *iso-loss training compute*, defined as the cumulative Transformer-block FLOPs required to reach the same validation loss.

Table 2: Cumulative Transformer block FLOPs at matched validation loss $L^\star$. Delayed expansion trains 80B dense tokens followed by 20B recurrent tokens.

Config	$L^\star$	Scratch @ $L^\star$ (B tokens)	Scratch FLOPs (EFLOP)	Ours FLOPs (EFLOP)	Saved (EFLOP)	Saving (%)
155M, $K = 4$	2.9122	85.59	58.17	40.78	17.39	**29.9%**
155M, $K = 8$	2.8993	83.10	94.12	49.84	44.29	**47.1%**
314M, $K = 4$	2.7237	85.07	152.54	107.59	44.95	**29.5%**
314M, $K = 8$	2.7095	82.44	246.41	131.50	114.90	**46.6%**

Table 3: Validation-loss shock after recurrent depth expansion at 40B, 60B, and 80B tokens, measured on a fixed set of 512 validation windows. At each expansion point, naive and step-relaxed conversion are evaluated from the same dense checkpoint.

Scale	Expansion	Dense loss	Naive Δ	Relaxed Δ
155M	40B	3.153	+0.894	**+0.055**
	60B	3.102	+1.050	**+0.069**
	80B	3.036	+1.327	**+0.102**
314M	40B	2.988	+0.710	**+0.044**
	60B	2.930	+0.843	**+0.057**
	80B	2.849	+1.081	**+0.081**

Table 4: Instantaneous conversion shock at the 80B expansion point as a function of recurrence depth K, measured on a fixed set of 1,024 validation windows. Each entry reports the increase relative to the dense checkpoint, with the post-conversion validation loss shown in parentheses.

	155M		314M	
K	Ours	Naive	Ours	Naive
4	**+0.101** (3.104)	+1.278 (4.280)	**+0.083** (2.905)	+1.076 (3.898)
8	**+0.152** (3.155)	+4.028 (7.031)	**+0.136** (2.957)	+3.425 (6.246)
16	**+0.189** (3.192)	+7.383 (10.386)	**+0.184** (3.005)	+6.261 (9.083)
32	**+0.212** (3.215)	+9.299 (12.302)	**+0.220** (3.041)	+8.416 (11.238)

For each configuration, $L^\star$ is the final validation loss reached after 80B dense tokens followed by 20B recurrent tokens. We identify the earliest point at which the corresponding recurrent-from-scratch model reaches $L^\star$ and compare the cumulative FLOPs of the two procedures.

As shown in Table 2, delayed depth expansion reduces cumulative compute across all settings. The savings are around 30% at $K = 4$ and increase to around 47% at $K = 8$. These gains do not come from improved token efficiency. The recurrent-from-scratch models reach the matched losses after roughly 83–86B tokens, whereas delayed expansion uses the full 100B-token schedule. The advantage instead comes from processing the first 80B tokens with cheaper dense computation. As recurrent depth increases, this early-stage saving becomes larger because the cost of full recurrent training grows while the dense-stage cost remains unchanged.

5.4 Step Relaxation Stabilizes Dense-to-Recurrent Expansion

The compute savings above rely on introducing recurrent depth into a model that has already been optimized densely. This conversion is nontrivial. Naively replacing the learned middle-stack transformation M by its K-fold composition M^K can substantially perturb the model's function. We therefore examine step relaxation from two perspectives. First we investigate its effect on the immediate loss discontinuity at conversion, and second its effect on the subsequent optimization trajectory.

5.4.1 Step Relaxation Limits the Immediate Conversion Shock

We first isolate the architectural effect of conversion by evaluating each checkpoint immediately after recurrent depth expansion, before any additional optimization. This allows us to measure the functional perturbation caused solely by replacing the dense execution topology with recurrent computation.

Table 3 shows that naive conversion produces substantially larger validation-loss increases at every expansion point. Across the 155M and 314M models, the naive loss increase ranges from $+0.710$ to $+1.327$, while the relaxed conversion limits it to the $+0.044$ to $+0.102$ range. At the 80B checkpoint, for example, the increase drops from $+1.327$ to $+0.102$ for the 155M model and from $+1.081$ to $+0.081$ for the 314M model.

Table 4 further shows that the difference becomes more pronounced as recurrent depth increases. The loss increase under naive conversion grows rapidly with K, reaching $+9.299$ and $+8.416$ at

Table 5: Recovery after recurrent depth expansion at the 155M and 314M scales. All grafts start from the corresponding 80B dense checkpoint and continue training over the same 20B-token suffix. Step-relaxed rows are shaded.

Model	Step-0 Val Loss ↓	Shock ↓	Tokens to Recover ↓	Pass Dense Final ↓	Final Val Loss ↓	Final PPL ↓
155M Scale						
Dense 6L	3.0028	–	–	–	2.9636	19.37
Ours K4 scratch	2.9295	–	–	–	2.8881	17.96
Naive K4 (80B graft)	4.2804	+1.2776	89.49B	92.90B	2.9126	**18.40**
Ours K4 (80B graft)	**3.1037**	**+0.1009**	**89.24B**	**92.68B**	**2.9122**	**18.40**
314M Scale						
Dense 6L	2.8216	–	–	–	2.7750	16.04
Ours K4 scratch	2.7436	–	–	–	2.6939	14.79
Naive K4 (80B graft)	3.8978	+1.0762	90.09B	93.48B	2.7250	15.26
Ours K4 (80B graft)	**2.9045**	**+0.0829**	**89.81B**	**93.21B**	**2.7237**	**15.24**

$K = 32$ for the 155M and 314M models, respectively. In contrast, the step-relaxed conversion remains below $+0.22$ across all evaluated depths. Thus, scaling the recurrent depth magnifies the functional disruption of naive conversion, while the relaxed update keeps the converted model much closer to its pre-expansion operating point.

5.4.2 Stable Conversion Improves Post-Expansion Training

We next ask whether the smaller instantaneous perturbation translates into more favorable optimization after expansion. Starting from the same 80B-token dense checkpoints, we apply either naive or step-relaxed recurrence and continue training over the same 20B-token suffix.

As shown in Table 5, step relaxation substantially reduces the initial loss increase at both model scales. This smaller perturbation is followed by faster recovery: the relaxed conversions return to the pre-expansion validation loss approximately 0.25–0.28B tokens earlier and surpass the final dense-model loss sooner than their naive counterparts.

The advantage persists through continued training. At both model scales, the relaxed conversions reach lower final validation loss than naive grafting, indicating that the improved transition stability does not come at the expense of final optimization quality.

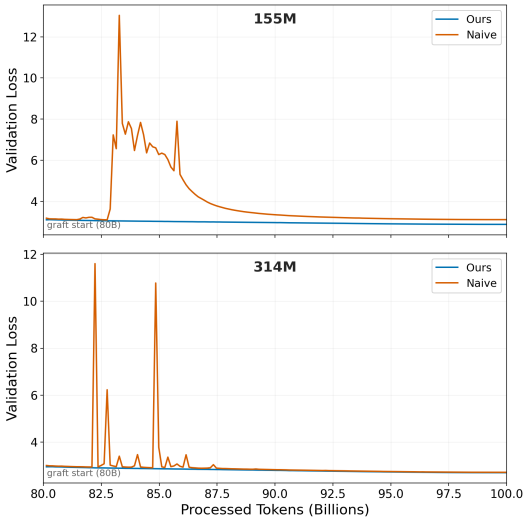


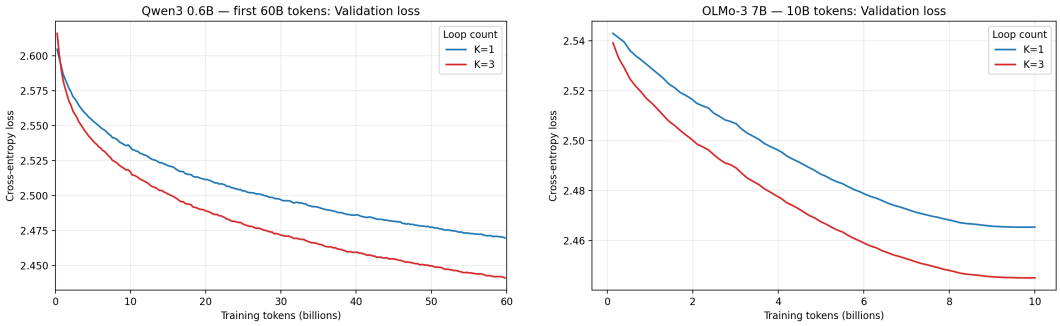
Figure 2: Training stability after recurrent depth expansion at $K = 16$. Naive recurrence exhibits pronounced loss fluctuations immediately after conversion, whereas step-relaxed DDE maintains a substantially smoother training trajectory.

The difference becomes especially visible at larger recurrent depth. Figure 2 shows continued training after conversion to $K = 16$. Naive recurrence exhibits pronounced loss fluctuations following conversion, whereas the step-relaxed model maintains a substantially smoother trajectory.

Together, these results show that step relaxation not only limits the immediate functional perturbation at conversion, but also provides a more stable starting point for subsequent recurrent training.

5.5 Retrofitting Pretrained Language Models

Finally, we test whether step-relaxed depth expansion transfers to existing pretrained language models. We retrofit Qwen3-0.6B, Llama-3.2-1B, Llama-3.2-3B, and OLMo-3-7B with $K = 3$ recurrent depth and continue full-parameter training on FineWeb. For each model, we compare against



(a) Qwen3-0.6B (60B) (b) Olmo3-1025-7B (10B)

Figure 3: Validation-loss trajectories during continued training of pretrained Qwen3-0.6B and OLMo-3-7B. Dense continuation ($K = 1$) is compared with step-relaxed recurrent conversion ($K = 3$). Additional pretrained-model trajectories are provided in the Appendix B.1

continued training of the original dense architecture ($K = 1$) under the same token budget and optimization schedule.

Figure 3 shows representative validation-loss trajectories for Qwen3-0.6B and OLMo-3-7B, with the remaining model curves provided in Appendix B.1. Across all evaluated models, Table 8 shows that the step-relaxed recurrent conversion reaches lower held-out FineWeb validation loss and perplexity than continued dense training. These results indicate that the conversion mechanism extends beyond the from-scratch setting to pretrained models across multiple architectures and parameter scales.

6 Limitations

Our study focuses on a fixed delayed-expansion schedule, where recurrent depth is introduced at a predetermined point and remains fixed thereafter. More adaptive strategies could determine when and how much recurrent depth to introduce based on the optimization state, potentially providing further efficiency gains. We also study DDE in decoder-only language models with recurrence applied to a contiguous middle stack. Although this setting captures the looped Transformer architectures considered in this work, extending delayed depth expansion to other architectures, recurrence placements, or modalities remains an interesting direction for future work. Finally, DDE reduces the compute required during training but does not reduce the per-token cost of the final recurrent architecture at inference time. Its benefit therefore lies in making the training trajectory more compute-efficient rather than making the resulting recurrent model cheaper to execute.

7 Conclusion

We introduced Delayed Recurrent Depth Expansion (DDE), a training strategy that postpones recurrent computation until later in optimization. Rather than training with the target recurrent depth from the beginning, DDE first optimizes a cheaper dense model and subsequently expands the middle stack into parameter-shared recurrent computation. Step-relaxed recurrence provides a controlled transition between these two regimes, substantially reducing the loss discontinuity introduced by dense-to-recurrent conversion and supporting stable continued training.

DDE reduces cumulative Transformer-block FLOPs at matched validation loss compared with recurrent training from scratch, with larger savings at greater recurrence depth. We further show that step-relaxed recurrence remains competitive with conventional looping when trained from scratch and that the same conversion mechanism extends to pretrained language models across multiple model families and scales. These results suggest that the recurrent depth required by the final model need not be used throughout the entire training trajectory, providing a simple approach for improving the training efficiency of looped Transformers without changing their final recurrent architecture.

AI USE STATEMENT

Generative AI tools, primarily ChatGPT, were used for language polishing, literature discovery, manuscript organization, and revising or drafting portions of the paper. All technical claims, citations, experimental results, and final text were reviewed and verified by the authors, who take responsibility for the final content of this work.

REFERENCES

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise lora. In *International Conference on Learning Representations*, 2025a.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *arXiv preprint arXiv:2507.10524*, 2025b.

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

Lizhang Chen, Jonathan Li, Chen Liang, Ni Lao, and Qiang Liu. Training-free looped transformers. *arXiv preprint arXiv:2605.23872*, 2026a.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations*, 2016.

Yao Chen, Yilong Chen, Yinqi Yang, Junyuan Shang, Zhenyu Zhang, Zefeng Zhang, Shuaiyi Nie, Shuohuan Wang, Yu Sun, Hua Wu, Haifeng Wang, and Tingwen Liu. Sparse growing transformer: Training-time sparse depth allocation via progressive attention looping. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 6168–6193, 2026b.

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.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019.

Wenyu Du, Tongxu Luo, Zihan Qiu, Zeyu Huang, Yikang Shen, Reynold Cheng, Yike Guo, and Jie Fu. Stacking your transformers: A closer look at model growth for efficient llm pre-training. In *Advances in Neural Information Processing Systems*, 2024.

Abhimanyu Dubey et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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, 2024.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, 2025.

Angeliki Giannou, Shashank Rajput, Jy-Yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, 2023.

Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tie-Yan Liu. Efficient training of bert by progressively stacking. In *International Conference on Machine Learning*, 2019.

Xiaotao Gu, Liyuan Liu, Hongkun Yu, Jing Li, Chen Chen, and Jiawei Han. On the transformer growth for progressive bert training. In *NAACL-HLT*, 2021.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020.

Shuzhen Li, Yifan Zhang, Jiacheng Guo, Quanquan Gu, and Mengdi Wang. Deeploop: Depth scaling for looped transformers. *arXiv preprint arXiv:2607.13491*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

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

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 2020.

Sheng Shen, Pete Walsh, Kurt Keutzer, Jesse Dodge, Matthew Peters, and Iz Beltagy. Staged training for transformer language models. In *International Conference on Machine Learning*, volume 162, pp. 19893–19908, 2022.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of NAACL-HLT*, pp. 4149–4158, 2019.

Team OLMo et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

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

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. Deepnet: Scaling transformers to 1,000 layers. *arXiv preprint arXiv:2203.00555*, 2022.

Jikai Wang, Juntao Li, Min Zhang, Zechang Li, Qingrong Xia, Xinyu Duan, Zhefeng Wang, and Baoxing Huai. When and how to grow? on efficient pre-training via model growth. In *Proceedings of the 16th Asian Conference on Machine Learning*, volume 260 of *Proceedings of Machine Learning Research*, pp. 95–110, 2025.

Peihao Wang, Rameswar Panda, Lucas Torroba Hennigen, Philip Greengard, Leonid Karlinsky, Rogerio Feris, David Daniel Cox, Zhangyang Wang, and Yoon Kim. Learning to grow pretrained models for efficient transformer training. In *International Conference on Learning Representations*, 2023.

Shaowen Wang, Bingrui Li, Ge Zhang, Wenhao Huang, Shen Yan, and Jian Li. On the residual scaling of looped transformers: Stability and transferability. *arXiv preprint arXiv:2606.18524*, 2026.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, 2017.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Jian Yang et al. Loopcoder: Scaling code intelligence via looped language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026.

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

Minjia Zhang and Yuxiong He. Accelerating training of transformer-based language models with progressive layer dropping. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

A Detailed Experimental Setup and Specifications

A.1 Dataset and Pretraining Hyperparameters

All from-scratch models are trained on the 100BT sample of FineWeb (Penedo et al., 2024). Text documents are concatenated and chunked into fixed-length sequences of 1,024 tokens without padding. Optimization is performed using AdamW (Loshchilov & Hutter, 2019) in `bf16` mixed precision. The batch size is 256 sequences per step (262,144 tokens). Optimization hyperparameters for both from-scratch and continued (delayed expansion) training are summarized in Table 6.

Table 6: Pretraining and continued training hyperparameters.

Hyperparameter	**From-Scratch Runs**	**Delayed Expansion Runs**
Dataset	FineWeb (100BT sample)	FineWeb (continuation suffix)
Context length	1,024 tokens	1,024 tokens
Batch size (tokens)	262,144 (256 windows)	262,144 (256 windows)
Precision	`bf16`	`bf16`
Optimizer	AdamW	AdamW
Betas (β_1, β_2)	(0.9, 0.95)	(0.9, 0.95)
Weight decay	0.1	0.1
Gradient clipping	1.0	1.0
Warmup schedule	Linear	Linear (1% of continuation window)
Learning rate schedule	Cosine decay	Cosine decay
Peak learning rate	6×10^{-4}	3×10^{-4}
Max training tokens	Up to 100B	20B (post-expansion suffix)

A.2 Architectural Configurations and Recurrence Rules

The architectures follow standard Llama-style decoder-only designs (Dubey et al., 2024) featuring pre-normalization with RMSNorm, SwiGLU activation functions, and Rotary Position Embeddings (RoPE). We examine models at two primary parameter counts: 155M and 314M unique parameters, alongside a 12-layer 463M dense reference model.

Block allocation. For recurrent models, the two central blocks constitute the recurrent middle stack M, flanked by two unshared initial blocks and two unshared final blocks. Applying K recurrent steps over M yields $4 + 2K$ total block evaluations per forward pass without altering parameter count. Structural specifications are detailed in Table 7.

Table 7: Model architecture specifications and effective block evaluations.

Model	**Parameters**	**Unique Blocks**	**Recurrence (K)**	**Effective Block Evaluations**
Dense 6L	155M	6	1	6
Looped $K = 4$	155M	6	4	12
Looped $K = 8$	155M	6	8	20
Dense 6L	314M	6	1	6
Looped $K = 4$	314M	6	4	12
Dense 12L	463M	12	1	12

Recurrence update rules. Given intermediate state u_k, we evaluate:

- **Conventional Loop:**

$$u_{k+1} = M(u_k) \tag{13}$$

- **Step-Relaxed Recurrence:**

$$u_{k+1} = u_k + \frac{1}{K}\left(M(u_k) - u_k\right) \tag{14}$$

A.3 Delayed Depth Expansion Protocol and FLOPs Accounting

Expansion schedule. Models start training as a 6-block dense baseline ($K = 1$). At expansion token step $G \in \{40\text{B}, 60\text{B}, 80\text{B}\}$, all weights are retained, and the middle stack recurrence is increased to K. In the primary setting ($G = 80\text{B}$), the model executes 80B tokens as a dense model followed by 20B tokens with recurrent depth.

Compute accounting. Following Kaplan et al. (2020), FLOPs are tracked cumulatively based on block evaluations:

- Prior to expansion ($< G$ tokens): Charged for 6 block evaluations per token.
- After expansion ($\geq G$ tokens): Charged for $4 + 2K$ block evaluations per token.
- Embedding and language model head projections are excluded, as they are identical across configurations.

Iso-loss training compute is evaluated by determining the earliest token step where the corresponding from-scratch looped model achieves the same final validation loss $L^\star$ as the delayed-expansion run, comparing the cumulative FLOPs expended by each method.

A.4 Evaluation Benchmarks

We evaluate zero-shot downstream performance using the `lm-evaluation-harness` (Gao et al., 2024) on eight standard benchmarks. PIQA evaluates physical commonsense reasoning (Bisk et al., 2020); ARC-Easy and ARC-Challenge evaluate grade-school science reasoning (Clark et al., 2018); HellaSwag measures commonsense sentence completion (Zellers et al., 2019); WinoGrande evaluates commonsense reasoning through adversarial Winograd-style problems (Sakaguchi et al., 2020); CommonsenseQA evaluates multiple-choice commonsense knowledge (Talmor et al., 2019); OpenBookQA evaluates science question answering requiring both provided facts and common knowledge (Mihaylov et al., 2018); and SciQ consists of multiple-choice science questions (Welbl et al., 2017). We report accuracy for PIQA, ARC-Easy, WinoGrande, CommonsenseQA, and SciQ, and normalized accuracy for ARC-Challenge, HellaSwag, and OpenBookQA, following the evaluation configuration used in our experiments.

B Additional Analysis

As illustrated by the graft loss gap in Figure 4, the difference between our relaxed graft and naive grafting remains strictly negative ($\Delta\text{loss} < 0$) across the entire continuation schedule, confirming that our trajectory consistently outperforms naive grafting at every optimization step.

Table 13 evaluates models trained at $K = 4$ under different test-time recurrence counts K'. Conventional recurrence is highly sensitive to extrapolating beyond its training depth, with validation perplexity increasing from 18.63 at $K' = 4$ to 33.51, 187.10, and over $8{,}000$ as K' increases to 8, 16, and 32. In contrast, the step-relaxed model remains stable over the same range, with perplexity changing gradually from 18.52 at $K' = 4$ to 21.51 even at $K' = 64$. Both models degrade when evaluated below their trained recurrence depth, but the relaxed model remains consistently better. These results show that step relaxation provides substantially greater robustness to changes in the recurrent discretization, particularly when increasing test-time depth beyond that seen during training.

B.1 Retrofit Loss Curves

Table 9 provides the full zero-shot evaluation for the $K = 4$ experiments, including the delayed-expansion grafts. At both model scales, step-relaxed recurrence trained from scratch achieves the best validation perplexity and the highest average downstream score among the $K = 4$ recurrent variants. The grafted models remain close to their from-scratch counterparts after only the 20B-token continuation stage, reaching an average score of 0.4652 versus 0.4699 at 155M and 0.4995 versus 0.4996 at 314M. These results indicate that delayed recurrent depth expansion preserves most

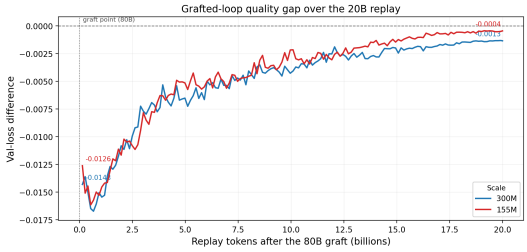
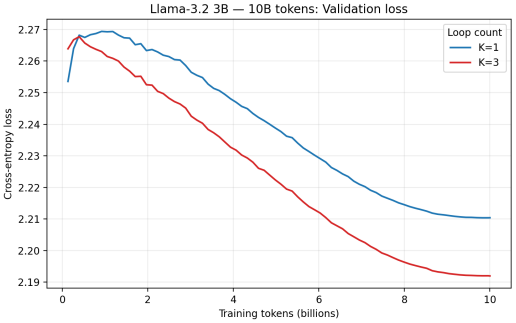


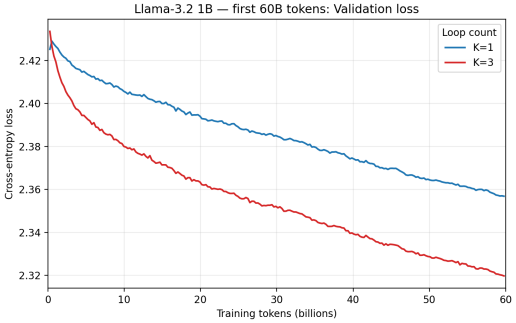
Figure 4: Validation-loss difference between step-relaxed and naive grafting throughout continued training. Negative values indicate lower validation loss for the step-relaxed graft.

Table 8: FineWeb validation performance after continued training of pretrained dense models and their $K = 3$ relaxed recurrent conversions.

Family	Model	Val loss ↓	PPL ↓
Llama-3.2-1B	Dense $K = 1$	2.3941	10.96
	Relaxed $K = 3$	**2.3902**	**10.92**
Llama-3.2-3B	Dense $K = 1$	2.2102	9.12
	Relaxed $K = 3$	**2.1920**	**8.95**
OLMo-3-7B	Dense $K = 1$	2.4653	11.77
	Relaxed $K = 3$	**2.4450**	**11.53**



(a) Llama-3.2-3B (10B)



(b) Llama-3.2-1B (60B)

of the downstream capability of the corresponding step-relaxed recurrent model while benefiting from substantially cheaper early-stage training.

Table 11 further examines optimization stability for recurrent models trained from scratch. At $K = 4$, naive and step-relaxed recurrence exhibit similar training-loss variation. As recurrent depth increases, however, the difference becomes more pronounced. At $K = 8$, step relaxation maintains nearly the same total variation as at $K = 4$, while the naive model becomes less smooth. At $K = 16$, both methods become more variable, but step relaxation substantially reduces the fluctuation, lowering TV from 49.63×10^{-3} to 30.95×10^{-3} while also achieving lower training loss. These results suggest that step relaxation becomes increasingly beneficial for optimization stability as recurrent depth grows.

Table 12 reports zero-shot downstream performance before and after continued training of the pretrained models. For the Llama models, the step-relaxed recurrent conversion recovers well during continued training and remains competitive with, or improves upon, continued dense training. The average score increases from 0.600 to 0.603 for Llama-3.2-1B and from 0.662 to 0.677 for Llama-3.2-3B. OLMo-3-7B shows a different behavior, with the recurrent model retaining a lower average zero-shot score despite its improved FineWeb validation performance. This indicates that gains on the continuation distribution do not necessarily translate uniformly to downstream task retention and may depend on the pretrained model and continuation data.

Table 9: Validation perplexity and zero-shot performance after 100B training tokens. *Naive* uses the conventional recurrence $u_{k+1} = M(u_k)$, while *Ours* uses the step-relaxed update $u_{k+1} = u_k + (M(u_k) - u_k)/K$. "Graft" denotes dense training to 80B tokens followed by recurrent depth expansion and 20B continued training. Metrics are zero-shot accuracy, except ARC-C, HellaSwag (HS), and OpenBookQA (OBQA), which use normalized accuracy. Avg is the unweighted mean over the eight LM-Eval tasks. ARC-E = ARC-Easy, ARC-C = ARC-Challenge, WG = WinoGrande, and CSQA = CommonsenseQA.

Model	Params	Val PPL ↓	PIQA	ARC-E	ARC-C	HS	WG	CSQA	OBQA	SciQ	Avg ↑
155M scale											
Dense 6L	155M	19.37	0.6469	0.5707	0.2619	0.3477	0.5233	0.1900	0.3120	0.8130	0.4582
$K = 4$ Naive	155M	18.05	0.6496	0.5846	0.2688	**0.3740**	0.5036	0.1941	**0.3200**	0.8260	0.4651
$K = 4$ Ours	155M	**17.96**	**0.6523**	**0.5867**	0.2688	0.3713	**0.5343**	**0.1957**	0.3180	**0.8320**	**0.4699**
$K = 4$ Ours (graft)	155M	18.40	0.6529	0.5829	0.2730	0.3627	0.5225	0.1966	0.3060	0.8250	0.4652
314M scale											
Dense 6L	314M	16.04	0.6730	0.6187	0.2730	0.4026	0.4996	0.2088	0.3380	0.8350	0.4811
$K = 4$ Naive	314M	14.86	**0.6774**	0.6271	**0.3046**	0.4413	**0.5438**	**0.1949**	0.3320	0.8680	0.4986
$K = 4$ Ours	314M	**14.79**	0.6741	**0.6288**	0.3038	**0.4453**	0.5296	0.1876	**0.3500**	**0.8780**	**0.4996**
$K = 4$ Ours (graft)	314M	15.24	0.6763	0.6334	0.3038	0.4304	0.5335	0.2129	0.3460	0.8600	0.4995

Table 10: Summary of evaluation metrics and downstream zero-shot tasks.

Evaluation Type	**Target / Suite**	**Metrics / Tasks**
Language Modeling	Held-out FineWeb split	Cross-entropy loss, Perplexity
Zero-Shot Evaluation	Gao et al. (2024)	PIQA, ARC-Easy, ARC-Challenge, HellaSwag, WinoGrande, CommonsenseQA, OpenBookQA, SciQ

Table 11: Training-loss smoothness for 155M recurrent models trained from scratch. TV is the mean absolute successive difference between training losses logged every 50 steps.

K	Naive loss ↓	Naive TV (10^{-3}) ↓	Relaxed loss ↓	Relaxed TV (10^{-3}) ↓
4	3.060	5.44	**3.051**	5.42
8	3.045	6.65	**3.033**	**5.42**
16	3.221	49.63	**3.110**	**30.95**

Table 12: Zero-shot accuracy for pretrained models converted to $K = 3$ recurrent depth and continued on FineWeb. *Pre* denotes zero-shot evaluations before continuation. Best continued results per family are bolded; Ours rows are shaded.

Family	Kind	PIQA ↑	ARC-E ↑	ARC-C ↑	HSwag ↑	WinoG ↑	CSQA ↑	OBQA ↑	Avg ↑
Llama-3.2-1B									
	Dense (K1) (pre)	0.754	0.663	0.370	0.642	0.602	0.517	0.376	0.608
	Ours (K3) (pre)	0.722	0.643	0.352	0.589	0.572	0.408	0.376	0.571
	Dense (K1) (10B)	**0.762**	0.689	**0.378**	0.645	0.610	**0.407**	0.380	0.600
	Ours (K3) (10B)	0.756	**0.694**	0.377	**0.656**	**0.631**	0.386	**0.394**	**0.603**
Llama-3.2-3B									
	Dense (K1) (pre)	0.762	0.745	0.463	0.741	0.691	0.664	0.406	0.679
	Ours (K3) (pre)	0.753	0.715	0.408	0.668	0.667	0.569	0.416	0.642
	Dense (K1) (10B)	**0.775**	0.783	0.470	0.728	0.679	0.484	0.424	0.662
	Ours (K3) (10B)	0.764	**0.792**	**0.488**	**0.741**	**0.701**	**0.529**	**0.446**	**0.677**
OLMo-3-7B									
	Dense (K1) (pre)	0.780	0.807	0.537	0.742	0.698	0.588	0.422	0.693
	Ours (K3) (pre)	0.743	0.768	0.519	0.726	0.651	0.544	0.416	0.664
	Dense (K1) (10B)	**0.777**	**0.816**	**0.531**	**0.752**	**0.696**	**0.639**	**0.446**	**0.703**
	Ours (K3) (10B)	0.749	0.777	0.488	0.721	0.659	0.547	0.418	0.664

Table 13: Test-time loop extrapolation and discretization scaling for models trained at $K = 4$. The naive recurrent model undergoes catastrophic divergence when evaluated beyond its training depth ($K' > 4$), whereas step relaxation maintains stable validation loss and perplexity across all recurrence counts.

	Naive $K = 4$			**Ours** $K = 4$ **(Step-Relaxed)**			
K'	Val Loss ↓	PPL ↓	$d_{\text{vs } K'=64}$	Step h	Val Loss ↓	PPL ↓	$d_{\text{vs } K'=64}$
64	11.4288	91,933.54	–	0.016	**3.0685**	**21.51**	–
32	8.9977	8,084.77	0.7801	0.031	**3.0499**	**21.11**	0.0321
16	5.2317	187.10	0.9672	0.062	**3.0168**	**20.43**	0.0732
8	3.5118	33.51	0.9924	0.125	**2.9692**	**19.48**	0.1483
4†	2.9248	18.63	0.9986	0.250	**2.9189**	**18.52**	0.3156
2	3.4635	31.93	0.9997	0.500	**3.3661**	**28.97**	0.6062
1	3.9571	52.30	1.0000	1.000	**3.8836**	**48.60**	0.7679

†Trained depth ($K = 4$).