ONE MODEL, MANY DEPTHS: A RECIPE FOR ELASTIC-DEPTH LATENT RECURRENT LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

A latent recurrent, or looped, language model applies a shared block of layers repeatedly, so one checkpoint can trade inference compute for accuracy by changing the loop count. At one loop, however, such models typically fall well short of a non-recurrent model of equal inference depth. This one-loop tax makes a separate non-recurrent model the better choice for latency-sensitive serving. We present Elastic Latent Recurrence (ELR), a recipe for training looped language models without this tax. Run at one loop, an ELR model costs the same to serve as a non-recurrent model of the same size and, when the two are trained on the same data, reaches similar downstream accuracy. Each additional loop of the same checkpoint then buys more accuracy for more inference compute. We show that two choices determine whether the one-loop accuracy survives mixed-depth training. The first is a gated update that sets a learned step size along each proposed state revision. The second is mixing loop counts within each optimizer update rather than across updates, which we implement by forcing a rotating subset of data-parallel ranks to run one loop. On a 4B dense hybrid model trained for 1T tokens and a hybrid mixture-of-experts model with 30B total and 3B active parameters trained for 300B tokens, each alongside a matched baseline, increasing inference depth from one to four loops improves the ten-task average by 7.25 and 4.75 points, respectively. Both models acquire elastic depth late in pretraining. Training at one loop for the first three quarters of pretraining and at mixed depth for the last quarter costs about 1.16$\times$ the baseline training FLOPs, and the resulting checkpoint improves at every added loop while giving the best one-loop quality of the schedules we tested. In our hybrid MoE experiments, reusing first-loop expert assignments on later loops removes repeated router evaluation and cuts mixed-depth training update time by 11.7% with no measurable change in downstream accuracy.

1 INTRODUCTION

Latent recurrent, or looped, language models apply a shared block of layers repeatedly, so a single checkpoint can spend more inference compute on a query by running more loops (Dehghani et al., 2019; Geiping et al., 2025; Zhu et al., 2025). When the loop count is varied during training, the same weights can be served at several depths, a property we call elastic depth (Geiping et al., 2025; Jeddi et al., 2026). For deployment this is an attractive property. The loop count becomes a per-request knob, much as a reasoning-effort setting is for chain-of-thought models, with the difference that no tokens are generated and, in principle, the lowest setting need cost no more than a standard model of the same size.

Whether this works in practice depends on where the quality-cost curve starts. Existing elastic-depth models are trained around large mean loop counts, and at one loop they fall far short of a non-recurrent model of equal depth. Huginn, trained with a mean recurrence of 32, trails its fixed-depth baseline on nearly every task, scoring 34.9 against 46.4 on ARC-Easy even after 800B tokens (Geiping et al., 2025). Parcae is below its Transformer baseline at one loop and above it at eight (Prairie et al., 2026). We call this gap the one-loop tax. A model that pays it must spend several loops recovering ground before additional compute buys anything, so a separately trained non-recurrent model remains the better choice for the latency-sensitive tier. A small loop range does

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.

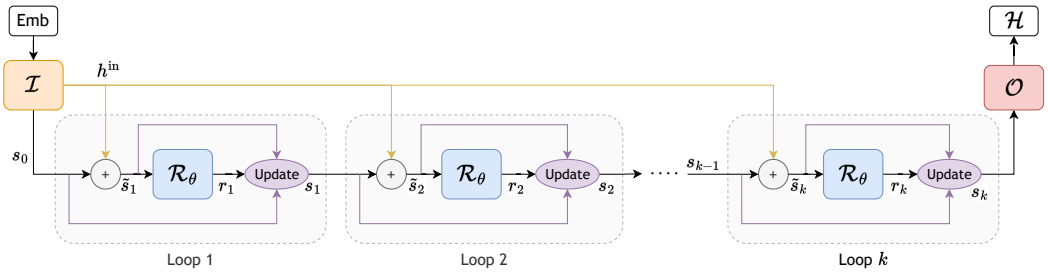


Figure 1: Elastic Latent Recurrence. The shared recurrent block $\mathcal{R}_\theta$ is applied k times between the input $\mathcal{I}$ and output $\mathcal{O}$ blocks. Each loop i re-injects h^{in} and updates the state with the adaptive-delta rule (Eq. 2), using a gate computed from $\tilde{s}_i$. At $k = 1$, every unique layer executes exactly once.

not remove the tax on its own; in our ablations, sampling uniformly over one to five loops still leaves the one-loop endpoint behind the baseline (Section 5.4). Avoiding the tax is a different question from whether looping is a compute-efficient way to scale a model (Prairie et al., 2026; Schwethelm et al., 2026b). We take the non-recurrent model as given and ask what it costs to make it elastic.

We present *Elastic Latent Recurrence* (ELR), a training recipe for looped Transformers that avoids the tax. ELR partitions a backbone into input, shared recurrent, and output blocks, so that at one loop every unique layer executes exactly once and the model has the same parameters and inference compute as a non-recurrent model of the same size. Two choices then decide whether the one-loop endpoint survives mixed-depth training. The first is the state update. Rather than replacing the state with the recurrent block's output, ELR takes a gated step along the proposed change, with a learned step size per token and channel. The second is where loop counts are mixed. Sampling one loop count per optimizer step and varying it across steps, as in Huginn's locked-step sampling, does not preserve the one-loop endpoint even when one loop is drawn a third of the time. Mixing loop counts within each step does. We implement this by forcing a rotating subset of data-parallel ranks to run one loop on every microbatch, which keeps step time at the sampled depth; sampling independently per rank is better still per step but raises step time by about a third. Although recurrent-depth models have been reported to diverge at learning rates that non-recurrent models tolerate (Prairie et al., 2026), every ELR model in this paper trains with its baseline's optimizer settings unchanged.

We train ELR alongside non-recurrent baselines on the same data at two scales, a 4B dense hybrid model trained for 1T tokens and a hybrid mixture-of-experts model with 30B total and 3B active parameters trained for 300B tokens, and use 0.5B models to ablate every design choice. At one loop, ELR is 0.5 points below its baseline on a ten-task average at 4B and 1.0 point above it at 30B-A3B. Each additional loop improves the average; at four loops the two models are 6.7 and 5.8 points above their baselines.

Mixed-depth training runs the recurrent block several times per example, but that cost need not be paid over the whole run. Both models train at one loop for the first three quarters of pretraining and at mixed depth only for the last quarter, for an estimated 16% more training FLOPs than the baseline against 62% for mixed-depth training throughout. For recurrent MoE models, *cross-loop route replay* evaluates each router once, on the first loop, and reuses its assignments on later loops while the selected experts still process each loop's updated representations.

Our contributions are as follows:

- **A recipe for elastic depth without a one-loop tax.** A gated state update and within-step mixing of loop counts preserve the one-loop endpoint; ablations show the common alternatives do not. At one loop, ELR reaches similar accuracy to a non-recurrent baseline trained on the same data, and every additional loop of the same checkpoint improves it.
- **Elastic depth acquired late in pretraining.** Mixed-depth training in the final quarter of the training run gives the full loop range for only 16% more training FLOPs than the baseline.
- **Cross-loop route replay for looped MoE models.** Reusing first-loop expert assignments on later loops cuts mixed-depth update time by 11.7% in our experiments with no measurable change in accuracy.

2 LATENT RECURRENT MODELS

Latent recurrent models, also called looped models, apply a shared block of layers repeatedly across depth to iteratively refine a hidden state. The idea descends from earlier work on depth-wise weight sharing and adaptive computation (Dehghani et al., 2019; Graves, 2016; Banino et al., 2021), and recent language models adopt it to decouple effective depth from unique parameter count (Geiping et al., 2025; Zhu et al., 2025; Bae et al., 2025b).

We describe a latent recurrent model as a composition of three blocks: an input block $\mathcal{I}$ with L_{in} layers, a recurrent block $\mathcal{R}_\theta$ with L_{rec} layers and parameters θ, and an output block $\mathcal{O}$ with L_{out} layers. Given a token sequence x, the input block produces a fixed representation $h^{\text{in}} = \mathcal{I}(\text{Emb}(x))$, from which a recurrent state is initialized and refined by k applications of the recurrent block:

$$s_0 = \text{Init}(h^{\text{in}}), \qquad s_i = \mathcal{G}_\theta\big(s_{i-1}, h^{\text{in}}\big), \quad i = 1, \dots, k$$

The transition map $\mathcal{G}_\theta$ wraps $\mathcal{R}_\theta$ and specifies how the state is updated at each iteration, with θ shared across all k applications. The output block and language-model head $\mathcal{H}$ then map the final state to logits $z = \mathcal{H}(\mathcal{O}(s_k))$. The choices of Init and $\mathcal{G}_\theta$ vary across architectures: Huginn (Geiping et al., 2025) initializes s_0 from a truncated Gaussian and projects the concatenation of s_{i-1} and h^{in}, whereas Ouro (Zhu et al., 2025) has no separate input or output blocks, initializing s_0 from the token embeddings and repeating the full stack without input re-injection.

The model has $D_{\text{unique}} = L_{\text{in}} + L_{\text{rec}} + L_{\text{out}}$ distinct layers, but a forward pass with loop count k traverses

$$D_{\text{eff}}(k) = L_{\text{in}} + k\, L_{\text{rec}} + L_{\text{out}} \tag{1}$$

layers, so each additional loop adds L_{rec} layers of inference compute and latency at a fixed parameter count. The loop count may be held constant or sampled from a distribution during training. We use the term elastic depth for a model trained over a range of loop counts and served at any k within that range, so that the loop count acts as an inference-time compute knob.

3 METHOD

This section introduces *Elastic Latent Recurrence* (ELR), a latent recurrent architecture and training recipe for serving a single checkpoint at any loop count within its trained range while avoiding the one-loop tax (Figure 1).

3.1 ELASTIC LATENT RECURRENCE

ELR uses the three-block architecture defined in Section 2, with approximately the middle half of the layers in $\mathcal{R}_\theta$ and the remaining layers split evenly between $\mathcal{I}$ and $\mathcal{O}$. At $k = 1$, every unique layer executes exactly once, with $D_{\text{eff}}(1) = D_{\text{unique}}$, matching a non-recurrent model of the same size in parameters and compute.

The recurrent state is initialized as $s_0 = h^{\text{in}}$. At each loop $i = 1, \dots, k$, $\mathcal{G}_\theta$ re-injects h^{in} into the current state, passes the result through $\mathcal{R}_\theta$, and applies a state update rule:

$$\begin{aligned} \tilde{s}_i &= s_{i-1} + h^{\text{in}} \\ r_i &= \mathcal{R}_\theta(\tilde{s}_i) \\ s_i &= \text{Update}(s_{i-1}, r_i, \tilde{s}_i) \end{aligned}$$

Adaptive-delta update. ELR uses an adaptive-delta rule that updates the state as

$$\begin{aligned} s_i &= \text{Update}(s_{i-1}, r_i, \tilde{s}_i) \\ &= s_{i-1} + \beta_i \odot (r_i - s_{i-1}) \end{aligned} \tag{2}$$

where the gate β_i sets a per-token, per-channel step size along the recurrent block's proposed revision $r_i - s_{i-1}$:

$$\beta_i = \sigma(W_\beta \tilde{s}_i + b_\beta) \odot \text{softplus}(\gamma)$$

Here $\text{softplus}(\gamma)$ is a learned per-channel base rate, scaled by a sigmoid gate for each token and channel. The parameters W_β, b_β, and γ are shared across loops and add $d^2 + 2d$ parameters for

hidden dimension d, independent of k. We initialize W_β and b_β to zero and γ to -2.25, giving an initial step size of approximately 0.05, so each loop starts as a small correction to the state. Section 5.1 compares adaptive delta with alternative state update rules.

3.2 Mixed-Depth Training with One-Loop Rank Overrides

To serve a single checkpoint at multiple loop counts, ELR is trained with mixed depths. For each microbatch, a loop count is sampled uniformly, $p(k) = 1/K$ for $k \in \{1, \ldots, K\}$, and the standard next-token loss is backpropagated through all k applications of the recurrent block. Uniform sampling improves accuracy at $k > 1$ but fails to match the non-recurrent baseline at $k = 1$ (Section 5.4). We therefore force a rotating fraction f of the data-parallel ranks to use $k = 1$ on every microbatch, while the remaining ranks use the sampled count. A rank then executes one loop with marginal probability p_1, and the model is trained at a mean loop count $\bar{k}$:

$$p_1 = \frac{1}{K} + \Big(1 - \frac{1}{K}\Big) f, \qquad \bar{k} = f + (1 - f)\,\frac{K+1}{2} \tag{3}$$

Unlike globally raising the sampling probability of $k = 1$, the override places one-loop and k-loop gradients in the same update whenever the sampled count exceeds one, and data-parallel reduction averages them before the optimizer step (Section 5.3). We choose f for each K so that p_1 remains approximately one third, and study sensitivity to this choice in Section 5.4.

3.3 Reducing the Cost of Recurrence

Delayed mixed-depth training. Mixed-depth training raises the compute of each step to about $D_{\text{eff}}(\bar{k})/D_{\text{eff}}(1)$ times that of a one-loop step (Eqs. 1 and 3). To limit this overhead, we train at $k = 1$ for most of the token budget and apply the schedule of Section 3.2 only in a final continuation phase. The two phases differ only in the distribution over k while the optimizer state and learning-rate schedule carry over unchanged. If the mixed-depth phase covers a fraction ρ of the training tokens, total training compute relative to the non-recurrent baseline is approximately

$$(1 - \rho) + \rho\,\frac{D_{\text{eff}}(\bar{k})}{D_{\text{eff}}(1)} \tag{4}$$

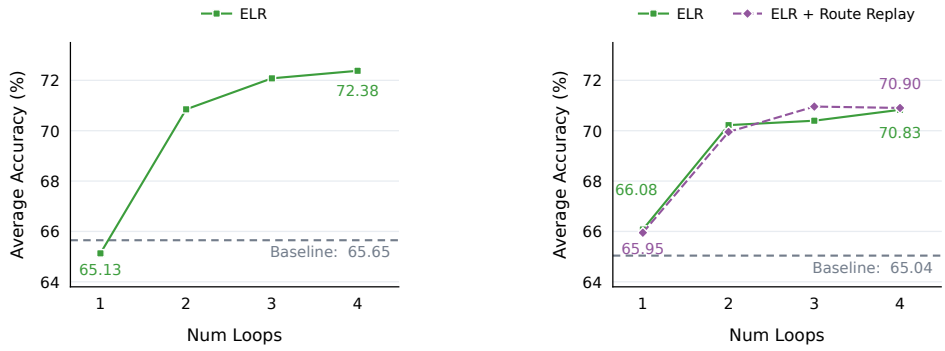
Section 5.5 studies how ρ trades one-loop quality against gains at additional loops.

Cross-loop route replay. In a recurrent MoE model, every loop re-executes the router of each MoE layer in the recurrent block. Cross-loop route replay instead computes each layer's expert assignments and routing weights on the first loop and reuses them on subsequent loops of the same forward pass. The selected experts still process each loop's updated representations, so replay removes repeated router evaluation while retaining expert dispatch, communication, computation, and output combination. During training, the load-balancing auxiliary loss is computed once on the first loop and scaled by the realized loop count k, matching the total weight it receives when accumulated over all k loops under dynamic routing. Replay assumes that first-loop routes remain useful at later loops. Section 4.3 tests this assumption by measuring how routing changes across loops and comparing dynamic routing with replay applied at inference only or during training, and reports the resulting training and inference cost.

4 Experiments

4.1 Experimental Setup

Models. We evaluate ELR on two hybrid models: GDP-Hybrid-4B, a dense model with approximately 4B parameters, and GDP-Hybrid-30B-A3B, a mixture-of-experts model with approximately 30B total and 3B active parameters. Both share a 52-layer layout of 23 Gated Delta Product (GDP) (Siems et al., 2025), 6 self-attention, and 23 feed-forward layers, which are dense MLPs in GDP-Hybrid-4B and MoE layers in GDP-Hybrid-30B-A3B. Each ELR model partitions these layers into 14 input, 24 recurrent, and 14 output layers and is compared with a non-recurrent baseline with the same backbone configuration. All ELR models follow the architecture of Section 3.1. For GDP-Hybrid-30B-A3B, we also train an ELR model with cross-loop route replay (Section 3.3). More architectural details are provided in Appendix A.1.



(a) GDP-Hybrid-4B

(b) GDP-Hybrid-30B-A3B

Figure 2: Average downstream benchmark accuracy across ten tasks versus inference loop count. Each curve represents a single model evaluated at multiple loop counts. Dashed horizontal lines mark the non-recurrent baselines. Per-task scores are in Tables 6 and 7.

Training. GDP-Hybrid-4B is trained for 1T tokens with AdamW, and GDP-Hybrid-30B-A3B for 300B tokens with Muon and AdamW. Both use a subset of the pretraining data used for Nemotron 3 Ultra (NVIDIA, 2026), with a sequence length of 8,192 and a global batch size of 768. Each non-recurrent baseline is trained on the same data and token budget as its ELR counterpart. All ELR models use delayed mixed-depth training (Section 3.3), training at one loop for the first 75% of tokens and then continuing with $k \in \{1, \ldots, 4\}$ and a 10% one-loop rank override, which gives $p_1 = 0.325$ and $\bar{k} = 2.35$ (Eq. 3). By Eq. 4, this schedule costs about $1.16\times$ the baseline's training compute. For GDP-Hybrid-30B-A3B, the ELR models with and without cross-loop route replay begin the mixed-depth phase from the same one-loop checkpoint. Full training settings are provided in Appendix A.

Evaluation. Each recurrent curve shows a single final checkpoint evaluated at every loop count in its trained range. We report validation loss and downstream accuracy on MMLU, MMLU-Pro, ARC-Challenge, GSM8K, WinoGrande, CommonsenseQA, RACE-high, HumanEval, MBPP, and MATH-500, and report their average. See Appendix B.1 for more details.

4.2 One-Loop Parity and Gains from Additional Loops

One-loop accuracy. At $k = 1$, ELR retains near-baseline accuracy at both scales (Figure 2). On GDP-Hybrid-4B, its ten-task average of 65.13 is 0.52 points below the non-recurrent baseline's 65.65. On GDP-Hybrid-30B-A3B, ELR and ELR+Route Replay reach 66.08 and 65.95, slightly above the baseline's 65.04. Per-task scores are reported in Tables 6 and 7. At one loop, individual tasks move in both directions; on GDP-Hybrid-4B, ELR trails the baseline by 5.3 points on GSM8K and leads by 5.2 on MATH-500.

Gains from additional loops. Each additional loop improves the ten-task average accuracy of ELR (Figure 2). From $k = 1$ to $k = 4$, the average rises by 7.25 points for GDP-Hybrid-4B and by 4.75 points for GDP-Hybrid-30B-A3B. The largest per-task gains are on GSM8K (+11.8 points) and MATH-500 (+11.3) for GDP-Hybrid-4B, and on MMLU-Pro (+7.5) and GSM8K (+6.5) for GDP-Hybrid-30B-A3B.

Diminishing returns. Most of the aggregate gain arrives in the first additional loop. For GDP-Hybrid-4B, two and three loops capture 79% and 96% of the gain from $k = 1$ to $k = 4$. For GDP-Hybrid-30B-A3B, they capture 87% and 91%. These curves show diminishing returns, which is consistent with later applications of $\mathcal{R}_\theta$ appearing in fewer training examples. With the one-loop override, the marginal loop-count probabilities during mixed-depth training are $p_1 = 0.325$ and $p_2 = p_3 = p_4 = 0.225$. An example reaches loop j only when $k \geq j$, so loops two, three, and four occur in 67.5%, 45%, and 22.5% of mixed-depth examples, giving a mean loop count of $\bar{k} = 2.35$ (Eq. 3).

Table 1: Cross-loop route replay on GDP-Hybrid-30B-A3B, relative to dynamically routed ELR (A). (B) applies replay to A's checkpoint at inference only while (C) is trained with replay. Validation loss and inference forward time are measured at $k = 4$ on 8K-token sequences. Training update time is averaged over the mixed-depth training phase and profiled at fixed $k = 4$.

Routing	Δ Val Loss at $k = 4$	Δ Training Update Time		Δ Inference Forward Time
		Mixed-depth	Fixed k=4	
(A) Dynamic routing		*reference*		
(B) Inference-only replay	$+0.0055$	–	–	-1.2%
(C) Replay training	$+0.0038$	-11.7%	-19.6%	-1.2%

Inference cost. Because a one-loop forward pass executes each unique layer exactly once, ELR at $k = 1$ matches the TTFT and per-token decode latency of its non-recurrent baseline. Each additional loop repeats only the recurrent block, reaching $2.33\times$ baseline TTFT and $2.06\times$ baseline decode latency at four loops for GDP-Hybrid-4B. Three loops therefore capture 96% of the accuracy gain from one to four loops at $1.87\times$ baseline TTFT. Quality–cost curves and measurement details are given in Appendix C.

4.3 Cross-Loop Route Replay

Routing in GDP-Hybrid-30B-A3B ELR changes little across loops in aggregate but substantially per token (Table 8). On 256 held-out 2K-token windows, per-expert selection counts at loops one and four have a cosine similarity of 0.978, averaged over the ten recurrent MoE layers, yet individual tokens retain only 53% of their first-loop experts at loop four. To isolate the effect of this drift on each layer's computation, we give each recurrent MoE layer its loop-four input from the dynamically routed model and compute the routed-expert output twice: once with the experts and weights the router selects for that input, and once with those recorded for the same token at loop one. Because both computations share the same input, their difference reflects only the routing decision. The two outputs have a mean cosine similarity of 0.767. A control that shifts first-loop routes across positions within each document, so that each token receives another token's routes, gives 0.227. First-loop routes therefore retain much, but not all, of the token-specific routing information.

The resulting quality cost is small, while training time falls substantially (Table 1). Inference-only replay (B) raises four-loop validation loss by 0.0055, and training with replay (C) narrows this to 0.0038 while leaving the ten-task average at $k = 4$ unchanged within noise (70.90 versus 70.83 for A). Replay reduces router evaluations from 53 to 23 per four-loop forward pass but retains expert dispatch and computation, so inference forward time falls by only 1.2% at 8K tokens. Training benefits more because replay also removes later-loop auxiliary-loss calculations and the backward work through repeated router evaluations. Update time falls by 11.7% over the mixed-depth training phase and by 19.6% at fixed $k = 4$, with expert and all-to-all call counts unchanged. More details are given in Appendix D.

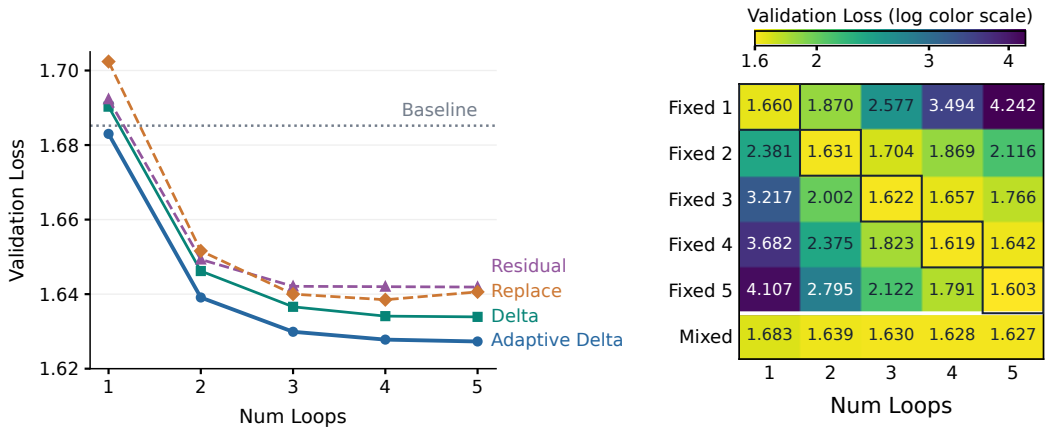
5 Ablations

We isolate the architectural and training choices of ELR on GPT-0.5B, a dense Transformer trained for 300B tokens on a subset of the pretraining data used for Nemotron 3 Ultra (NVIDIA, 2026), and report validation loss. Unless stated otherwise, each ELR variant trains with mixed-depth training from scratch with $K = 5$ and a 15% one-loop rank override, giving $p_1 = 0.32$ (Eq. 3).[1] Full results are given in Appendix E.

5.1 Adaptive Delta versus Alternative State Updates

Adaptive delta is the only state update rule that matches the baseline at $k = 1$, and it achieves the lowest loss at every loop count (Figure 3a). We compare it (Eq. 2) with two parameter-free

[1] GPT-0.5B omits input re-injection ($\tilde{s}_i = s_{i-1}$). The validation loss difference remains within noise so the conclusions of this section remain unchanged (see Appendix E.1).



(a) Comparison of State Update Rules

(b) Fixed versus Mixed Depth Training

Figure 3: Validation loss on GPT-0.5B. (a) Solid lines mark rules with a learned step size, dashed lines mark parameter-free rules, and the dotted line marks the non-recurrent baseline (1.685). (b) Fixed-depth vs mixed-depth training. Outlined diagonal cells mark matching training and evaluation depths. The Mixed row is the same adaptive-delta model in (a).

rules, replace ($s_i = r_i$) and residual ($s_i = s_{i-1} + r_i$), and with delta, which substitutes a learned, input-independent per-channel rate α, initialized to 0.05, for the input-dependent gate β_i. Delta ranks second at every depth but trails adaptive delta by 0.006–0.007, indicating that conditioning the step size on the input is better than a fixed per-channel rate. The parameter-free rules perform worst. Replace has the highest one-loop loss, 1.7024, and its loss rises from four to five loops, while residual stops improving after three loops.

5.2 Fixed-Depth versus Mixed-Depth Training

Mixed-depth training makes a single model usable across multiple loop counts during inference (Figure 3b). A model trained at one fixed loop count reaches the lowest loss at that count and degrades at every other k, with the gap growing with distance from its training depth in either direction. Because each fixed-depth model outperforms the mixed-depth model at its own depth, it serves as an empirical upper bound on the quality attainable at that depth. The mixed-depth model instead improves monotonically from one to five loops and trades this per-depth optimum for elasticity: a single checkpoint serves all five loop counts in place of five separately trained models, at some cost in loss at each depth.

The model trained only at $k = 1$ reaches a lower loss than the non-recurrent baseline (1.660 versus 1.685), although both execute each unique layer once. Without input re-injection, the two differ only in the gate, which at one loop is a learned skip around the recurrent block, so the gate appears to help even without recurrence.

5.3 Synchronized versus Independent-Rank Loop Sampling

The sampling scheme determines how many depths contribute gradients to each optimizer update. Under synchronized sampling, all data-parallel ranks run the same loop count in a step, so each update covers one depth, or at most two with one-loop overrides. Under independent-rank sampling, each rank draws its own loop count, so one update can cover every depth, but the step waits for the rank with the largest count. At $p_1 = 0.32$, the three schemes in Table 2 share the same marginal distribution over k, with $p_k = 0.17$ for $k \geq 2$, and differ only in how depths are grouped within updates. Loss decreases as more depths share an update. One-loop loss falls from 1.6871 under synchronized weighted sampling to 1.6830 with 15% overrides and 1.6774 under independent-rank sampling, and the same ordering holds at every $k \geq 2$. Independent-rank sampling gives the lowest

Table 2: Loop sampling schemes on GPT-0.5B (validation loss). Weighted sampling spreads the probability remaining after p_1 uniformly over $k \in \{2, \ldots, 5\}$, so rows with the same p_1 share the same marginal distribution over k. Depths/Update is the maximum number of distinct loop counts contributing gradients to one update. Step time is relative to synchronized uniform sampling.

p_1	Sampling	Depths/Update	Step Time	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
–	Non-recurrent baseline	–	–	1.6852	–	–	–	–
0.20	Synchronized, uniform	1	1.00×	1.7077	1.6473	1.6344	1.6318	1.6313
	Independent, uniform	≤ 5	1.36×	1.6937	**1.6330**	**1.6193**	**1.6157**	**1.6147**
0.32	Synchronized, weighted	1	1.00×	1.6871	1.6436	1.6343	1.6320	1.6316
	Synchronized, 15% override	≤ 2	1.00×	1.6830	1.6391	1.6299	1.6278	1.6273
	Independent, weighted	≤ 5	1.36×	**1.6774**	1.6334	1.6231	1.6200	1.6194

loss but increases step time by about 36%. One-loop overrides recover part of this gain at synchronized step time so we use them in our main experiments.

5.4 One-Loop Rank Overrides

Under synchronized sampling, one-loop rank overrides are needed for one-loop parity (Figure 4b). Without overrides ($f = 0$), uniform sampling reaches one-loop loss +0.0225 above the baseline. One-loop loss decreases monotonically as f increases, and $f = 0.15$ ($p_1 = 0.32$) is the smallest tested fraction that falls below the baseline. Larger fractions buy further one-loop gains at the cost of deeper loops, which motivates choosing f so that p_1 is approximately one third (Section 3.2). The benefit does not come from the higher one-loop marginal alone, since synchronized weighted sampling at the same p_1 without overrides remains above the baseline (Section 5.3).

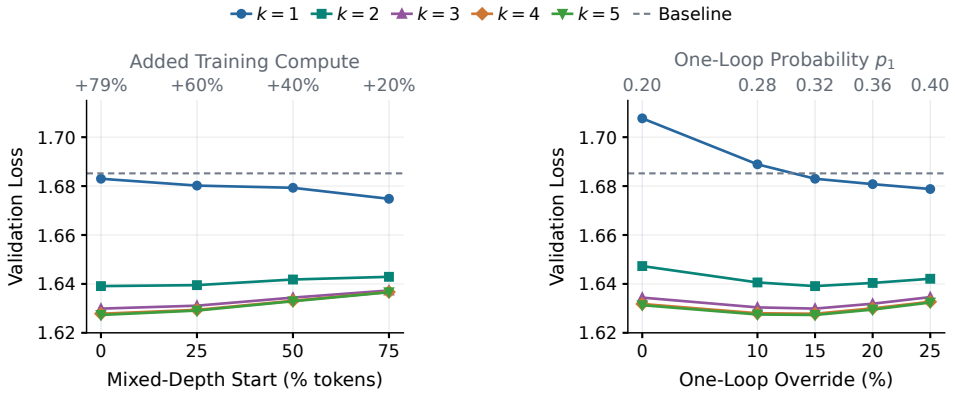
5.5 Delayed Mixed-Depth Training

Delaying mixed-depth training retains most of the gain from additional loops at a fraction of its training cost (Figure 4a). Mixed-depth training from scratch costs an estimated $1.79\times$ the baseline's training compute (Eq. 4). Starting it after 25%, 50%, or 75% of training tokens lowers this to $1.60\times$, $1.40\times$, and $1.20\times$, while retaining 97%, 90%, and 84% of the five-loop improvement over the baseline. At the 75% start, the estimated added training compute falls from about 79% to 20% for a 0.009 increase in five-loop loss, and one-loop loss stays below the baseline at every start point. We therefore use the 75% start for our main experiments (Section 4.1).

6 Related Work

Elastic recurrent depth. Huginn (Geiping et al., 2025) and Ouro (Zhu et al., 2025) train looped language models whose depth can vary at inference, Mixture-of-Recursions routes tokens to learned recursion depths (Bae et al., 2025b), and LoopFormer conditions a shared stack on normalized time and step size so that shortcut trajectories align with the full-depth trajectory (Jeddi et al., 2026). LoopFormer's dense comparisons match either unique parameters or block-level inference FLOPs but not both. During training, Huginn samples a single recurrence depth per step, shared by all data-parallel workers (Geiping et al., 2025), whereas the one-loop rank override mixes one-loop and deeper gradients within a single update. Closest to our goal, stochastic RINS drops recursion rounds during training and, with depth-specific linear adapters, matches or exceeds a training-compute-matched baseline at one recursion in 600M-parameter models (Alabdulmohsin & Zhai, 2025). That result is reported in perplexity, and without the adapters, stronger one-recursion protection reduces gains at higher depths. ELR reaches one-loop parity without depth-specific parameters, with downstream evaluation at 4B dense-hybrid and 30B-A3B MoE scale.

Regulating recurrent dynamics. Hyperloop Transformers apply matrix-valued hyper-connections after each loop (Zeitoun et al., 2026), Parcae constrains input-injection dynamics with stability conditions derived from a linearized system (Prairie et al., 2026), STARS regularizes the Jacobian spectral radius (Yang et al., 2026), and Training-Free Looped Transformers reinterpret



(a) Delayed Mixed-Depth Training

(b) One-Loop Override

Figure 4: Training choices on GPT-0.5B. Each x-axis point is a separately trained model, and each line shows validation loss at one inference loop count k across these models. The dashed line marks the non-recurrent baseline (1.685). (a) Point at which training switches from one loop to mixed depth ($f = 0.15$); 0% is mixed-depth training from scratch, and added compute is estimated from Eq. 4. (b) One-loop override fraction f, with mixed-depth training from scratch.

repeated frozen-model computation as damped substeps (Chen et al., 2026a). ELR's adaptive-delta update starts from a small step size but learns it end to end without explicit stability constraints, and is designed to match the non-recurrent baseline at one loop while improving with additional loops.

Acquiring recurrence late. Retrofitted Recurrence (McLeish et al., 2025) and Relaxed Recursive Transformers (Bae et al., 2025a) convert pretrained non-recurrent models into recurrent ones, using a recurrence curriculum and depth-specific low-rank adapters, respectively. Our delayed mixed-depth training instead continues a model that is recurrent by construction, changing only the loop-count distribution, with no architectural change or optimizer restart.

Recurrent MoE and serving. Sparse Universal Transformer (Tan et al., 2023) and Loop-MoE (Chen et al., 2026b) combine sparse expert routing with weight-shared depth. Read-ME decouples routing from a non-recurrent MoE backbone so that expert decisions can be precomputed for scheduling (Cai et al., 2024), whereas cross-loop route replay reuses a looped model's own first-loop routes on later loops. Continuous Depth Batching schedules tokens that exit at different loop counts (Schwethelm et al., 2026a), whereas we serve each request at a fixed loop count.

7 Conclusion

Elastic Latent Recurrence shows that a looped model can serve a single checkpoint across a range of inference depths without paying a one-loop tax. At one loop, an ELR model executes every unique layer exactly once and so matches the inference cost of a non-recurrent model of the same size. Trained on the same data, its average downstream benchmark accuracy is 0.52 points below the baseline at 4B and 1.04 points above it at 30B-A3B. Each additional loop of the same checkpoint then improves accuracy, by 7.25 and 4.75 points from one to four loops. Ablations show that two choices help retain the one-loop accuracy under mixed-depth training. The first is an adaptive-delta state update rule. The second is mixing loop counts within each optimizer update through rotating one-loop rank overrides. Elastic depth also need not be trained from scratch. Introducing mixed depths only in the final quarter of pretraining costs about $1.16\times$ the baseline's training compute, and cross-loop route replay reduces mixed-depth training step time in recurrent MoE models by 11.7% with no measurable change in downstream accuracy. Several questions remain open. Gains diminish after the second loop, which is consistent with later loops appearing in fewer training examples, but we have not tested whether reweighting the loop-count distribution recovers them. Our serving setting also fixes the loop count per request. Combining ELR with per-request or per-token adaptive depth, and testing it at larger scales and longer training budgets, are the next steps.

AI USE STATEMENT

In this work, we used generative AI tools to assist with implementing the method by writing and debugging parts of the training, evaluation, and profiling code. We also used them to write figure plotting code and to edit paper text for clarity and readability. The authors developed the ELR method, designed the experiments and ablations, and interpreted the results without generative AI assistance. Generating synthetic data, formulating mathematical claims or proofs, translation, and dataset cleaning are not applicable to this work. We reviewed all AI-assisted code and validated it before use and verified AI-edited text against the reported results. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies the training and inference of general-purpose language models. The models are pretrained base models without safety alignment, trained on a subset of the Nemotron 3 Ultra pretraining data (NVIDIA, 2026) and evaluated only on capability benchmarks. Like other such models, they can produce biased or harmful content and may reproduce sensitive information present in their training data, and ELR neither introduces nor mitigates these risks.

REPRODUCIBILITY STATEMENT

Section 3 describes the ELR architecture and training procedure, Section 4.1 summarizes the experimental setup, and Appendix B.1 specifies the downstream evaluation protocol. Appendix A gives the model configurations, optimizer and training settings for all models, including the non-recurrent baselines and ablation models. The experiments use a subset of the publicly released Nemotron 3 Ultra pretraining data (NVIDIA, 2026) and public evaluation benchmarks. Our implementation builds on the open-source Megatron-LM framework, and we plan to release our changes as a Megatron-LM fork together with the training configurations used in this paper.

REFERENCES

Ibrahim Alabdulmohsin and Xiaohua Zhai. Recursive inference scaling: A winning path to scalable inference in language and multimodal systems. In *Advances in Neural Information Processing Systems*, 2025.

Jacob Austin, Augustus Odena, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Sangmin Bae, Adam Fisch, et al. Relaxed recursive transformers: Effective parameter sharing with layer-wise LoRA. In *International Conference on Learning Representations*, 2025a.

Sangmin Bae, Yujin Kim, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Advances in Neural Information Processing Systems*, 2025b.

Andrea Banino, Jan Balaguer, et al. PonderNet: Learning to ponder. In *8th ICML Workshop on Automated Machine Learning*, 2021.

Ruisi Cai, Yeonju Ro, et al. Read-ME: Refactorizing LLMs as router-decoupled mixture of experts with system co-design. In *Advances in Neural Information Processing Systems*, 2024.

Lizhang Chen, Jonathan Li, et al. Training-free looped transformers. *arXiv preprint arXiv:2605.23872*, 2026a.

Mark Chen, Jerry Tworek, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Wenkai Chen, Tianshu Li, et al. LoopMoE: Unifying iterative computation with mixture-of-experts for language modeling. *arXiv preprint arXiv:2606.04438*, 2026b.

Peter Clark, Isaac Cowhey, et al. Think you have solved question answering? Try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Mostafa Dehghani, Stephan Gouws, et al. Universal Transformers. In *International Conference on Learning Representations*, 2019.

Jonas Geiping, Sean McLeish, et al. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, 2025.

Alex Graves. Adaptive Computation Time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Dan Hendrycks, Collin Burns, et al. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

Ahmadreza Jeddi, Marco Ciccone, et al. LoopFormer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *International Conference on Learning Representations*, 2026.

Guokun Lai, Qizhe Xie, et al. RACE: Large-scale ReAding comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017.

Hunter Lightman, Vineet Kosaraju, et al. Let's verify step by step. In *International Conference on Learning Representations*, 2024.

Sean McLeish, Ang Li, et al. Teaching pretrained language models to think deeper with retrofitted recurrence. *arXiv preprint arXiv:2511.07384*, 2025.

NVIDIA. Nemotron 3 Nano: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning. *arXiv preprint arXiv:2512.20848*, 2025.

NVIDIA. Nemotron 3 Ultra: Open, efficient mixture-of-experts hybrid mamba-transformer model for agentic reasoning. *arXiv preprint arXiv:2606.15007*, 2026.

Hayden Prairie, Zachary Novack, et al. Parcae: Scaling laws for stable looped language models. *arXiv preprint arXiv:2604.12946*, 2026.

Keisuke Sakaguchi, Ronan Le Bras, et al. WinoGrande: An adversarial Winograd schema challenge at scale. *Communications of the ACM*, 2021.

Kristian Schwethelm, Daniel Rückert, et al. Depth-adaptive inference of looped language models via continuous depth batching. *arXiv preprint arXiv:2608.09444*, 2026a.

Kristian Schwethelm, Daniel Rückert, et al. How much is one recurrence worth? iso-depth scaling laws for looped language models. *arXiv preprint arXiv:2604.21106*, 2026b.

Julien Siems, Timur Carstensen, et al. DeltaProduct: Improving state-tracking in linear RNNs via Householder products. In *Advances in Neural Information Processing Systems*, 2025.

Alon Talmor, Jonathan Herzig, et al. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

Shawn Tan, Yikang Shen, et al. Sparse Universal Transformer. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Yubo Wang, Xueguang Ma, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

Xiao-Wen Yang, Zi-Yu Han, et al. Stabilizing recurrent dynamics for test-time scalable latent reasoning in looped language models. In *International Conference on Machine Learning*, 2026.

Abbas Zeitoun, Lucas Torroba-Hennigen, et al. Hyperloop Transformers. *arXiv preprint arXiv:2604.21254*, 2026.

Rui-Jie Zhu, Zixuan Wang, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A Model Architecture and Training Details

All models are implemented in Megatron-LM. Each self-attention, feed-forward, GDP, and MoE layer counts as one layer. Every ELR model shares its backbone configuration with its non-recurrent baseline and adds only the adaptive-delta gate. The gate's $d^2 + 2d$ parameters amount to about 0.2% of the baseline parameters for GPT-0.5B and GDP-Hybrid-4B, and 0.02% of the total (0.2% of the active) parameters for GDP-Hybrid-30B-A3B.

A.1 Architectures

GPT-0.5B is a dense Transformer whose 30 layers alternate between self-attention and SwiGLU MLP layers, and its ELR variant partitions them 8/14/8. GDP-Hybrid-4B and GDP-Hybrid-30B-A3B follow the Nemotron 3 Nano hybrid layout (NVIDIA, 2025), with Mamba-2 layers replaced by Gated DeltaProduct (GDP) layers (Siems et al., 2025). Both use the same layer ordering, written below with G for GDP, A for attention, and F for a feed-forward layer, which is a dense MLP in GDP-Hybrid-4B and an MoE layer in GDP-Hybrid-30B-A3B. Bars mark the 14/24/14 ELR partition:

`GFGFGAFGFGFGAF | GFGFGAFGFGFGAFGFGFGAFGFG | FGFGAFGFGFGFGF`

All models use RMSNorm, linear layers without bias, untied embeddings, no dropout, BF16 training, and a 131,072-token vocabulary. Table 3 lists the remaining dimensions.

Table 3: Backbone configurations, shared by each ELR model and its non-recurrent baseline. Partial RoPE rotates 50% of each attention head's dimensions.

	GPT-0.5B	GDP-Hybrid-4B	GDP-Hybrid-30B-A3B
Layers (GDP / attention / FF)	0 / 15 / 15	23 / 6 / 23	23 / 6 / 23
ELR partition ($\mathcal{I}$ / $\mathcal{R}_\theta$ / $\mathcal{O}$)	8 / 14 / 8	14 / 24 / 14	14 / 24 / 14
Feed-forward type	Dense	Dense	MoE
Activation	SwiGLU	Squared ReLU	Squared ReLU
Hidden size d	1,024	3,072	2,688
FF intermediate size	3,968	12,288	1,856 per expert
Query heads / KV groups	8 / 4	24 / 8	32 / 2
Attention head dim.	128	128	128
Position encoding	RoPE (10^6)	None	Partial RoPE (10^5)
GDP heads / head dim. / state dim.	–	96 / 32 / 64	16 / 64 / 128
Routed experts / top-k	–	–	512 / 22
Expert latent dim. / shared expert size	–	–	672 / 3,712

MoE routing. GDP-Hybrid-30B-A3B computes sigmoid router scores in FP32. It balances expert load with a learned per-expert score bias and a sequence-level auxiliary loss with coefficient 10^{-4}.

A.2 Optimization

Table 4 lists the optimization settings, which each baseline shares with its ELR counterpart. All models train on a subset of the Nemotron 3 Ultra pretraining data (NVIDIA, 2026). GDP-Hybrid-30B-A3B applies Muon to two-dimensional weight matrices, with momentum 0.9, 16 Newton–Schulz iterations using Polar Express coefficients, and spectral update scaling with an extra factor of 0.2. AdamW updates all other parameters. The learning rate follows a warmup–stable–decay schedule with a $1 - \sqrt{\cdot}$ decay to the minimum learning rate over the final 15% of training tokens.

A.3 Mixed-Depth Training

A single majority loop count is drawn for each microbatch and shared by all data-parallel ranks. Of the N_{DP} ranks, $\mathrm{round}(fN_{\mathrm{DP}})$ contiguous ranks run one loop, and the starting index of this group shifts on every microbatch. Gradients are backpropagated through all executed loops without truncation.

Table 4: Optimization and ELR training settings. Each non-recurrent baseline uses the same data, token budget, and optimization settings as its ELR counterpart. p_1 and $\bar{k}$ are nominal values from Eq. 3.

	GPT-0.5B	GDP-Hybrid-4B	GDP-Hybrid-30B-A3B
Training tokens	300B	1T	300B
Optimizer	AdamW	AdamW	Muon + AdamW
AdamW (β_1, β_2) / weight decay	(0.9, 0.95) / 0.1	(0.9, 0.95) / 0.1	(0.9, 0.95) / 0.1
Peak / minimum learning rate	1.8×10^{-3} / 1.8×10^{-5}	1.2×10^{-3} / 1.2×10^{-5}	1.2×10^{-3} / 1.2×10^{-5}
Warmup tokens	8.4B	33.6B	8.4B
Sequence length / global batch	8,192 / 768	8,192 / 768	8,192 / 768
Gradient clipping	1.0	1.0	1.0
Initialization std	0.028	0.0162	0.0173
Maximum loop count K	5	4	4
One-loop override fraction f	0.15	0.10	0.10
p_1 / $\bar{k}$	0.32 / 2.70	0.325 / 2.35	0.325 / 2.35
Mixed-depth phase	All tokens	Final 25%	Final 25%

In delayed mixed-depth training, the continuation restores the model weights, optimizer state, and learning-rate schedule from the one-loop checkpoint. For GDP-Hybrid-30B-A3B, the dynamically routed and route-replay ELR models start from the same one-loop checkpoint. The fixed-depth GPT-0.5B models in Section 5.2 train without overrides.

B Downstream Benchmark Results

B.1 Evaluation Protocol

Evaluation uses task-specific decoding and scoring protocols. Every ELR checkpoint is evaluated at each loop count in its trained range, and the non-recurrent baseline uses the same protocol. Table 5 lists the benchmarks, few-shot counts, and metrics. GSM8K and MATH-500 answers are evaluated using mathematical equivalence via Math-Verify.

Table 5: Downstream evaluation benchmarks.

Benchmark	Domain	Shots	Metric
MMLU (Hendrycks et al., 2021)	Knowledge	5	Accuracy
MMLU-Pro (Wang et al., 2024)	Knowledge	5	Accuracy
ARC-Challenge (Clark et al., 2018)	Science QA	25	Accuracy
WinoGrande (Sakaguchi et al., 2021)	Commonsense	5	Accuracy
CommonsenseQA (Talmor et al., 2019)	Commonsense	7	Accuracy
RACE-high (Lai et al., 2017)	Reading comprehension	0	Accuracy
GSM8K (Cobbe et al., 2021)	Math	8 (CoT)	Answer accuracy
MATH-500 (Lightman et al., 2024)	Math	4	pass@1
HumanEval (Chen et al., 2021)	Code	0	pass@1
MBPP (sanitized) (Austin et al., 2021)	Code	3	pass@1

B.2 GDP-Hybrid-4B

Table 6 reports the per-benchmark scores underlying Figure 2a. All ELR columns come from a single checkpoint evaluated at $k = 1, \ldots, 4$.

Table 6: Downstream benchmark scores (%) for GDP-Hybrid-4B trained for 1T tokens.

Benchmark	Baseline	ELR			
		$k = 1$	$k = 2$	$k = 3$	$k = 4$
MMLU	62.1	62.5	66.7	67.2	67.6
MMLU-Pro	41.3	39.5	45.0	47.0	47.3
ARC-Challenge	79.5	79.8	83.8	84.6	85.0
WinoGrande	70.8	73.5	74.8	75.2	75.6
CommonsenseQA	71.4	68.8	75.5	77.2	77.8
RACE-high	76.9	74.4	80.9	81.2	82.4
GSM8K	76.3	71.0	81.0	82.3	82.8
MATH-500	64.1	69.3	75.5	79.6	80.6
HumanEval	57.3	56.1	66.5	64.6	64.0
MBPP	56.8	56.4	58.8	61.9	60.7
Avg.	**65.65**	**65.13**	**70.85**	**72.08**	**72.38**

B.3 GDP-HYBRID-30B-A3B

Table 7 reports the per-benchmark scores underlying Figure 2b for ELR with dynamic routing and with cross-loop route replay (Section 4.3). Both ELR models start the mixed-depth phase from the same one-loop checkpoint.

Table 7: Downstream benchmark scores (%) for GDP-Hybrid-30B-A3B trained for 300B tokens.

Benchmark	Baseline	ELR				ELR + Route Replay			
		$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 1$	$k = 2$	$k = 3$	$k = 4$
MMLU	62.04	65.55	68.48	69.32	69.28	65.23	68.51	69.15	69.17
MMLU-Pro	39.07	38.40	44.35	45.78	45.94	39.27	44.23	45.42	45.68
ARC-Challenge	77.39	79.78	83.70	84.73	84.73	79.35	84.90	85.32	85.32
WinoGrande	74.74	74.03	76.72	77.19	78.69	74.03	75.45	75.85	76.01
CommonsenseQA	69.53	72.56	77.31	78.13	77.72	71.66	75.59	77.31	77.64
RACE-high	75.12	77.89	83.54	83.64	83.06	79.14	82.87	84.78	84.50
GSM8K	79.68	80.44	86.13	86.35	86.96	79.91	85.06	86.05	86.81
MATH-500	60.78	63.34	65.84	65.90	65.95	66.53	70.47	68.51	68.54
HumanEval	49.39	45.73	50.00	47.56	50.61	43.29	49.39	51.83	50.00
MBPP	62.65	63.04	66.15	65.37	65.37	61.09	63.04	65.37	65.37
Avg.	**65.04**	**66.08**	**70.22**	**70.40**	**70.83**	**65.95**	**69.95**	**70.96**	**70.90**

C INFERENCE LATENCY MEASUREMENTS

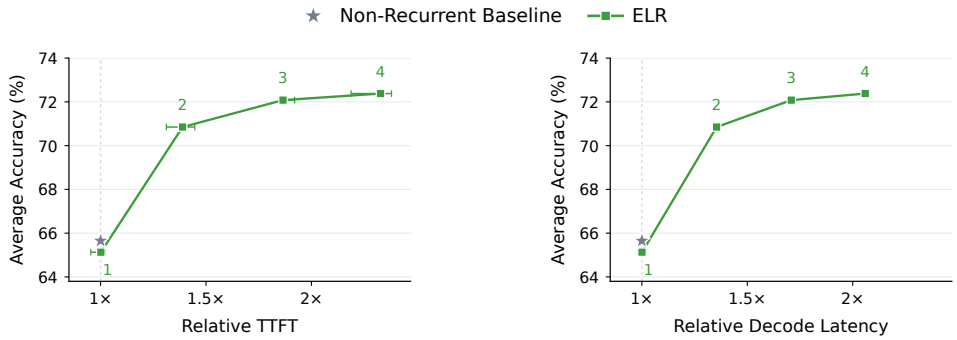
C.1 MEASUREMENT SETUP

We time the GDP-Hybrid-4B baseline and ELR checkpoints with vLLM 0.20.2 on one H100 80GB GPU, using BF16 weights, eager prefill, and full CUDA-graph decode. The main workload uses batch size one, 1,024 prompt tokens, and 128 greedily generated tokens with EOS ignored, so every request has a fixed output length. TTFT is measured from batch submission to the first output token, and per-token decode latency is the time between the first and last output tokens divided by 127. Inputs are pretokenized, and prefix caching, chunked prefill, speculative decoding, and `torch.compile` are disabled. We report median times over seven repetitions after two warm-up batches, normalized to the non-recurrent baseline under the same workload. The 95% confidence in-

tervals in Figure 5 are percentile-bootstrap intervals of the time ratio over 10,000 draws, resampling model and baseline repetitions independently. They reflect timing variability only.

C.2 Sensitivity to Workload

We repeat the measurements at prompt lengths of 128, 1,024, and 4,096 tokens and batch sizes of one and eight, using either CUDA graphs for decode with eager prefill or eager execution throughout. Across prompt lengths and batch sizes, one-loop ELR decode latency differs from the baseline by -0.26% to $+0.07\%$ with CUDA graphs and by $+0.41\%$ to $+2.16\%$ in eager mode. The corresponding TTFT differences are -1.54% to $+0.42\%$ with CUDA graphs and $+0.16\%$ to $+1.53\%$ in eager mode.



(a) Time to First Token

(b) Decode Latency

Figure 5: GDP-Hybrid-4B ten-task average accuracy versus (a) TTFT and (b) per-token decode latency, normalized to the non-recurrent baseline ($1\times$). Numbers indicate loop counts, and horizontal bars show 95% bootstrap intervals.

D Route Replay Analysis

D.1 Evaluation Protocol

We compare the dynamically routed checkpoint (A), replay enabled only during inference on A's checkpoint (B), and a separately trained replay checkpoint (C) from Section 4.3. Both checkpoints are trained for 300B tokens. We sample 256 validation documents with one 2,048-token window each. Results on 64 separate documents with 8,192-token windows follow the same pattern.

D.2 Routing and Output Similarity

Routing statistics compare loops one and four in A (Table 8). For each token, the retained fraction is $|S_1 \cap S_4|/22$, where S_1 and S_4 are its first- and fourth-loop sets of 22 selected experts. In contrast, frequency cosine compares aggregate expert-use counts across tokens and does not measure whether a given token keeps its experts. For the layer-output comparison, we hold each layer's fourth-loop input fixed on A's trajectory and compare the output under fresh routes with the output under first-loop routes, without propagating these outputs to later layers. As a control, first-loop routes and weights are circularly shifted within each document, which preserves aggregate usage but breaks token correspondence. Including the shared experts increases output similarity in both comparisons, but replay remains closer to fresh routing than the shifted-route control.

D.3 Timing Measurements

Mixed-depth training update time is averaged over all updates and the reduction from replay stays between 11.0% and 12.7% across training intervals. The fixed $k = 4$ comparison uses a single checkpoint with replay toggled, with identical code and hardware for both conditions. Inference forward time compares B with A in Megatron-LM on eight H100 GPUs with BF16 weights and

Table 8: Cross-loop routing and output comparisons, averaged over ten recurrent MoE layers. Routing statistics compare loops one and four in A, with a 95% document-bootstrap interval in brackets. Output cosines report first-loop replay / shifted-route control on the same fourth-loop inputs. Full-MoE outputs include shared experts and exclude the outer residual.

Metric	Value
Retained experts (%)	53.17 [52.37, 53.89]
Aggregate frequency cosine	0.978
Routed-output cosine	0.767 / 0.227
Full-MoE output cosine	0.918 / 0.718

expert parallelism of eight. We time full-context forward passes from GPU-resident inputs to logits, taking the maximum wall time across ranks after three warm-up passes. At four loops, replay reduces forward time by 1.21–2.96% across sequence lengths of 128 to 8,192 tokens and one or two documents per rank, with the smallest reduction at 8,192 tokens.

E Ablation Results

This appendix reports the validation losses underlying the GPT-0.5B ablations in Section 5.

E.1 Input Re-injection

Table 9 compares passing s_{i-1} to $\mathcal{R}_\theta$ directly with re-injecting the input-block output as $\tilde{s}_i = s_{i-1} + h^{\text{in}}$ (Section 3.1). All other settings follow the default GPT-0.5B ablation configuration. Repeating the fixed- versus mixed-depth, loop sampling, one-loop override, and delayed mixed-depth ablations with input re-injection gives the same ordering in each (Sections 5.2–5.5). We expect the state-update comparison (Section 5.1) to transfer as well.

Table 9: Input re-injection on GPT-0.5B (validation loss).

Recurrent-block input $\tilde{s}_i$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
Non-recurrent baseline	1.6852	–	–	–	–
s_{i-1}	**1.6830**	**1.6391**	**1.6299**	**1.6278**	**1.6273**
$s_{i-1} + h^{\text{in}}$	1.6858	1.6442	1.6352	1.6329	1.6325

E.2 State Update Rules

Table 10 lists the values plotted in Figure 3a (Section 5.1).

Table 10: State update rules on GPT-0.5B (validation loss).

State update rule	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
Non-recurrent baseline	1.6852	–	–	–	–
Adaptive delta	**1.6830**	**1.6391**	**1.6299**	**1.6278**	**1.6273**
Delta	1.6903	1.6462	1.6366	1.6341	1.6339
Residual	1.6923	1.6493	1.6421	1.6420	1.6419
Replace	1.7024	1.6516	1.6400	1.6385	1.6406

E.3 One-Loop Rank Overrides

Table 11 lists the values plotted in Figure 4b (Section 5.4).

Table 11: One-loop rank overrides on GPT-0.5B (validation loss) under synchronized uniform sampling with override fraction f. p_1 follows Eq. 3.

f	p_1	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
Non-recurrent baseline		1.6852	–	–	–	–
0	0.20	1.7077	1.6473	1.6344	1.6318	1.6313
0.10	0.28	1.6889	1.6406	1.6304	1.6280	1.6275
0.15	0.32	1.6830	**1.6391**	**1.6299**	**1.6278**	**1.6273**
0.20	0.36	1.6808	1.6404	1.6319	1.6300	1.6295
0.25	0.40	**1.6788**	1.6421	1.6346	1.6327	1.6324

E.4 Delayed Mixed-Depth Training

Table 12 lists the values plotted in Figure 4a (Section 5.5), with training compute estimated from Eq. 4.

Table 12: Delayed mixed-depth training on GPT-0.5B (validation loss). The start point is the percentage of training tokens completed at one loop before switching to mixed depth. Training compute is relative to the non-recurrent baseline.

Start point	Training compute	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
Non-recurrent baseline	1.00×	1.6852	–	–	–	–
0%	1.79×	1.6830	**1.6391**	**1.6299**	**1.6278**	**1.6273**
25%	1.60×	1.6802	1.6395	1.6311	1.6294	1.6291
50%	1.40×	1.6793	1.6418	1.6344	1.6331	1.6329
75%	1.20×	**1.6748**	1.6429	1.6373	1.6367	1.6366