LOOP DROPOUT: REGULARIZING SHARED UPDATES IN LOOPED LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped language models separate computational depth from parameter count by repeatedly applying the same transformer block. Adapting these models requires a shared update that remains effective as hidden states evolve throughout the recurrent computation. Our empirical analysis reveals a pronounced late-loop bias in standard low-rank adaptation (LoRA): the shared update is more effective at later loop positions. This imbalance motivates training shared updates under varying combinations of their applications. Randomly omitting adapter applications alone, however, does not improve task performance; it reduces expected update strength during training while leaving inference unchanged. We introduce Loop Dropout, which couples stochastic masking of adapter applications with inverse-survival rescaling to preserve expected update strength and promote effective adaptation across loops. Extensive experiments demonstrate improved mathematical reasoning across model sizes, adapter ranks and training recipes, with benefits extending to general instruction tuning and code generation. Loop Dropout outperforms existing LoRA variants and adapter regularizers, while further analysis shows stronger early-loop adaptation. Every backbone loop remains active, and inference applies the adapter at all loops using standard LoRA without additional trainable parameters or inference computation.

1 INTRODUCTION

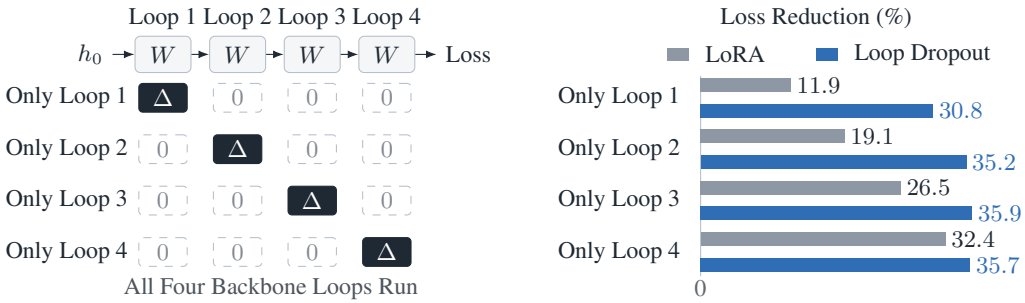
Looped language models separate computational depth from parameter count by repeatedly applying the same transformer block (Dehghani et al., 2019; Geiping et al., 2025; Zhu et al., 2025). This reuse allows compact models to perform multiple stages of computation and match substantially larger standard language models (Zhu et al., 2025). Adapting these models requires a shared update that remains effective as hidden states evolve throughout the recurrent computation.

Low-rank adaptation (LoRA) learns a compact update to frozen pretrained weights (Hu et al., 2022), with variants modifying its parameterization, optimization and regularization (Hayou et al., 2024; Kalajdzievski, 2023; Liu et al., 2024; Lin et al., 2024). In a looped model, the same update acts on different hidden states at different distances from the final prediction. Standard LoRA fine-tuning activates all applications together and optimizes their combined effect through the final loss. Step-resolved data attribution shows that training examples influence the loops unevenly (Kaissis et al., 2026), motivating an empirical analysis of how effectively the learned update works across the recurrence.

Our empirical analysis reveals a pronounced late-loop bias in shared adaptation. Figure 1a evaluates a trained adapter at each loop position in turn, with the adapter active only at that position and the backbone running all four loops. For Ouro-1.4B fine-tuned on GSM8K with standard LoRA, loss reduction falls from 32% when the update is applied at the fourth loop to 12% when applied at the first, relative to the same frozen model. The same late-loop bias persists across training configurations. This imbalance motivates learning an update that supports effective adaptation throughout the recurrent computation.

Dropout encourages features to remain useful across different combinations of other features (Srivastava et al., 2014). For a shared adapter, this principle suggests learning across varying combinations of its applications. However, we find that randomly omitting adapter applications alone does not

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.



(a) Single-Loop Activation

(b) Single-Loop Loss Reduction

Figure 1: **Loop Dropout reduces the late-loop bias of shared adaptation.** Ouro (Zhu et al., 2025) uses four loops through a shared transformer block by default. To probe adaptation across loops after GSM8K fine-tuning, we activate the adapter in one loop at a time while retaining all four backbone loops. LoRA's loss reduction falls from 32% at the fourth loop to 12% at the first; Loop Dropout maintains 31–36% across positions. All reductions are relative to the frozen model and use final-loop teacher-forced test loss.

improve task performance. Stochastic omission creates a training–inference mismatch: it reduces the expected training update, while inference applies the full update at every loop.

We therefore introduce Loop Dropout, which couples stochastic masking of adapter applications with inverse-survival rescaling to promote effective adaptation across loops. Each retained update is divided by its survival probability, preserving the expected update strength during training. This design trains the shared update under varying application patterns while keeping every backbone loop active. At inference, all applications are active and the adapter follows standard LoRA, with no additional trainable parameters or inference computation.

Figure 1b shows that Loop Dropout maintains loss reductions of 31–36% across all four loops, narrowing the disparity between early and late applications. When adapters trained with four loops are evaluated at eight loops without further fine-tuning, Loop Dropout outperforms LoRA by 4.75 percentage points on GSM8K.

Extensive experiments demonstrate that Loop Dropout delivers consistent gains on mathematical reasoning and improves general instruction tuning, with the clearest benefits on code generation. Loop Dropout outperforms existing LoRA variants and adapter regularizers such as CoTo (Zhuang et al., 2025b), highlighting the value of tailoring adaptation to the recurrent structure of looped language models.

Our contributions are as follows:

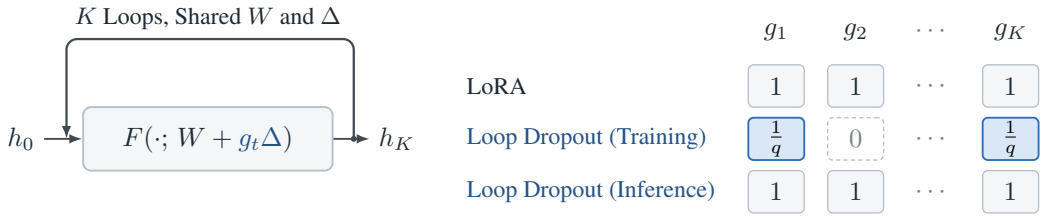
- We identify a pronounced late-loop bias in standard LoRA fine-tuning, exposing the challenge of learning shared updates that remain effective across loops.
- We introduce Loop Dropout, coupling independent masks over complete applications of the shared update with inverse-survival rescaling to promote effective adaptation across loops while retaining standard LoRA inference.
- We demonstrate improved mathematical adaptation and transfer across model sizes, adapter ranks and training recipes, supported by stronger early-loop adaptation, zero-shot generalization to deeper recurrence and comparisons with alternative regularizers.

2 Preliminaries

Looped computation. A looped language model embeds an input sequence into h_0 and applies a transformer block F with shared parameters W for K loops:

$$h_t = F(h_{t-1}; W), \qquad t = 1, \ldots, K, \qquad z = \mathrm{Head}(h_K). \tag{1}$$

We call each application of the shared transformer block a *loop*; the recurrence depth K is the number of loops. Increasing K adds computation without another copy of the block parameters. We use the



(a) Shared Recurrent Block (b) Gates on the Shared Update

Figure 2: **Loop Dropout regularizes shared adaptation across loops.** The shared block F reuses the same W and Δ over K loops. Following Eq. (3), training uses $g_t = b_t/q$, where $q = 1 - p$ and $b_t \sim \text{Bernoulli}(q)$ independently for each example and loop. The training row illustrates one sampled pattern: retained updates have gate $1/q$, and dropped updates have gate 0. LoRA and inference use $g_t = 1$ at every loop. A zero gate removes only Δ; every backbone loop remains active.

base Ouro models (Zhu et al., 2025), whose default recurrence depth is $K = 4$, and fix the depth for all examples rather than using their adaptive exit gates.

Shared low-rank updates. LoRA (Hu et al., 2022) adapts a frozen matrix $W_i \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ through $\Delta_i = \frac{\alpha}{r} B_i A_i$, where $A_i \in \mathbb{R}^{r \times d_{\text{in}}}$, $B_i \in \mathbb{R}^{d_{\text{out}} \times r}$, and r is the rank. We write Δ for the collection of these updates across the recurrent block. Standard fine-tuning then optimizes

$$\mathcal{L}_{\text{LoRA}}(\Delta) = \mathbb{E}_{(x,y)} \text{ CE}\big(\text{Head}(h_K), y\big), \qquad h_t = F(h_{t-1}; W + \Delta), \tag{2}$$

with loss on the target tokens y. Each Δ_i has one pair of trainable factors shared across all K loops, and its gradient includes contributions from every application. Training separate factors at each loop is an alternative, but multiplies the adapter parameter count by K at a fixed rank. We compare both equal-rank and equal-parameter versions of this alternative.

3 Loop Dropout

The late-loop bias identified in Section 1 motivates learning a shared update that remains effective throughout the recurrent computation. Loop Dropout couples loop-level masking with inverse-survival rescaling: masking trains the update under varying combinations of its applications, while rescaling preserves its expected strength at every loop. Figure 2 illustrates the gates used in training and at inference.

3.1 Loop-Level Masking

For each training example, Loop Dropout samples independent masks $b_t \sim \text{Bernoulli}(q)$, where $q = 1 - p \in (0, 1]$ is the survival probability, and computes

$$h_t = F(h_{t-1}; W + g_t\Delta), \qquad g_t = \frac{b_t}{q}, \qquad t = 1, \ldots, K. \tag{3}$$

One mask is shared by all adapted modules and token positions within a loop, as illustrated in Figure 2b. If $b_t = 0$, the loop uses only the pretrained weights; otherwise, it uses the shared update scaled by $1/q$. The loss is evaluated at the final loop as in Eq. (2), all backbone weights remain frozen, and Algorithm 1 summarizes the procedure.

Masking a complete application changes both the input states reaching later loops and the set of adapter applications that contribute to the final prediction. Each active application is thus trained with varying combinations of earlier and later applications. Sharing one mask across all adapted modules makes the complete adapter application the unit of regularization.

3.2 Inverse-Survival Rescaling

Masking alone reduces each loop's expected training update from Δ to $q\Delta$, while inference uses the full update Δ. The factor $1/q$ in Eq. (3) compensates for this reduction, matching each loop's

Algorithm 1 Training with Loop Dropout

Require: frozen block $F(\cdot; W)$, shared LoRA factors Δ, depth K, survival probability q

for each training example (x, y) in a mini-batch **do**
  $h_0 \leftarrow \mathrm{Embed}(x)$
  for $t = 1, \ldots, K$ **do**
    $b_t \sim \mathrm{Bernoulli}(q)$ independently; $g_t \leftarrow b_t/q$
    $h_t \leftarrow F(h_{t-1}; W + g_t\Delta)$ ▷ same gate for all modules and tokens
  end for
  $\ell_{x,y} \leftarrow \mathrm{CE}(\mathrm{Head}(h_K), y)$
end for
Update the LoRA factors using the mean mini-batch loss.
Inference: set $g_t = 1$ at every loop.

expected training update to its inference update. Writing $g_t = 1 + \varepsilon_t$ with $\varepsilon_t = (b_t - q)/q$, the effective weights of loop t decompose as

$$W + g_t\Delta = \underbrace{W + \Delta}_{\text{inference weights}} + \underbrace{\varepsilon_t\Delta}_{\text{zero-mean perturbation}}, \qquad \mathbb{E}[\varepsilon_t] = 0, \qquad \mathrm{Cov}(\varepsilon) = \frac{p}{q} I_K. \tag{4}$$

The perturbation acts along the learned update and leaves the pretrained weights unchanged. At $p = 1/2$, each loop applies either 2Δ or no update with equal probability, while its mean update remains Δ. This is the mean-preserving convention of inverted dropout (Srivastava et al., 2014), applied to complete adapter applications.

This property extends to the composed recurrent output at first order. Let $z_\eta(g)$ denote the final logits when loop t uses $W + \eta g_t\Delta$, where η is an auxiliary update scale for expansion about the frozen computation at $\eta = 0$.

Lemma 1 (First-order consistency of recurrent adaptation)**.** *Fix an input, update Δ, depth K and survival probability $q \in (0, 1]$. Suppose F is twice continuously differentiable in its state and weights, and* Head *is twice continuously differentiable, near the frozen trajectory. For independent $b_t \sim \mathrm{Bernoulli}(q)$, as $\eta \to 0$,*

$$\begin{aligned} \mathbb{E}_b\, z_\eta(b/q) &= z_\eta(\mathbf{1}) + O(\eta^2), \\ \mathbb{E}_b\, z_\eta(b) &= z_{q\eta}(\mathbf{1}) + O(\eta^2). \end{aligned} \tag{5}$$

Each first-order contribution includes an update's effect propagated through all subsequent backbone loops. Rescaling preserves the mean of their sum, while unscaled masking multiplies it by q. Appendix C.1 derives these contributions and bounds the second-order remainder. Thus rescaling aligns the leading adaptation effect during training with that at inference, complementing the varying application patterns produced by masking. The comparison in Table 5 shows higher accuracy when loop-level masking is coupled with this rescaling.

3.3 Training Across Application Patterns

The two components jointly train the shared update across a distribution of application patterns while preserving its expected strength. Let $\ell_\Delta(g)$ denote the loss for one example under a gate vector g, and let $\mathbf{1}_S$ indicate the subset S of loops with active updates. The objective is

$$\mathbb{E}_b\, \ell_\Delta(b/q) = \sum_{S \subseteq \{1,\ldots,K\}} q^{|S|} p^{K-|S|}\, \ell_\Delta(\mathbf{1}_S/q). \tag{6}$$

This weighted average jointly optimizes single-loop and multi-loop applications of the same update, exposing the shared parameters to both sparse and dense application patterns. Section 4.4 examines how the learned update's effectiveness changes across loop positions.

For a fixed learned update, centering the gates at $\mathbf{1}$ also gives a local regularization view of this objective. Expanding Eq. (6) around the inference gate yields

$$\mathbb{E}_b\, \ell_\Delta(b/q) = \ell_\Delta(\mathbf{1}) + \frac{p}{2q} \sum_{t=1}^{K} v_t^\top H\, v_t + R, \qquad v_t = \left.\frac{\partial z}{\partial g_t}\right|_{g=\mathbf{1}}, \tag{7}$$

where $H \succeq 0$ is the Hessian of the cross-entropy with respect to the logits z and R collects the higher-order terms together with the part of the gate Hessian that involves second derivatives of z. The first term is the standard LoRA objective; the nonnegative curvature term describes local regularization along each application's logit sensitivity, with strength $p/(2q)$ equal to half the gate variance (Wager et al., 2013; Bishop, 1995). Appendix C gives the derivation, the exact scaling relations and the gate covariance of the controls in Section 4.5.

Our default uses $p = 0.5$ and requires only a scalar gate on each adapter output. The method adds no trainable parameters or inference computation to standard LoRA. Implementation details and alternative mask distributions are given in Appendices B and G.1.

4 Experiments

We evaluate mathematical adaptation on Ouro-1.4B and Ouro-2.6B and compare with existing adaptation methods, then examine early-loop effectiveness, the roles of masking and rescaling, and robustness across adapter ranks and recurrence depths.

4.1 Experimental Setup

Models and tasks. We use the base Ouro models with four loops. Mathematical fine-tuning follows the MetaMathQA pipeline of LoRA-Pro (Yu et al., 2024; Wang et al., 2025): one epoch on 100k GSM-type examples, followed by zero-shot evaluation on GSM8K (Cobbe et al., 2021) and MATH-500 (Hendrycks et al., 2021b; Lightman et al., 2024) with the MetaMath scorer. Instruction tuning of Ouro-1.4B uses a 100k-example draw from Tülu 2 (Ivison et al., 2023) and evaluates code generation with EvalPlus (Chen et al., 2021; Austin et al., 2021; Liu et al., 2023), knowledge and reasoning with MMLU and BBH (Hendrycks et al., 2021a; Suzgun et al., 2023), truthfulness with TruthfulQA (Lin et al., 2022), and instruction following with IFEval (Zhou et al., 2023). A smaller recipe fine-tunes directly on GSM8K and supports the initial activation diagnostic and extended robustness studies. Appendices D and E specify all recipes and scoring rules.

Methods. Default adapters use rank 16 and $\alpha/r = 2$ on the seven projections of the recurrent block, giving 15.1M trainable parameters on Ouro-1.4B and 30.3M on Ouro-2.6B. Loop Dropout uses $p = 0.5$. We compare with LoRA (Hu et al., 2022), LoRA+ (Hayou et al., 2024), CoTo (Zhuang et al., 2025b) and LoRA Dropout (Lin et al., 2024), and isolate masking and sharing through the controls in Section 4.5. Appendix J.3 also compares rsLoRA (Kalajdzievski, 2023) under the direct GSM8K recipe (Table 24). Main mathematical comparisons tune five learning rates per method; perturbation controls fix a common rate.

Training and reporting. Comparisons match training examples, optimizer steps, sequence length and evaluation protocol. Main mathematical results report mean and sample SD over three training seeds. LoRA+ uses $\eta_B/\eta_A = 4$ and the shifted candidate grid in Appendix D.2.

4.2 Overall Performance

Mathematical reasoning and transfer. Table 1 shows that Loop Dropout improves both mathematical benchmarks at both model sizes. On Ouro-1.4B, GSM8K accuracy increases by 1.19 percentage points over LoRA, while MATH-500 rises from 38.87% to 47.53%, an 8.67-point gain. On Ouro-2.6B, the corresponding gains are 1.11 and 4.33 points. Both benchmarks improve in every training seed at each size. The larger gains on MATH-500 show stronger transfer from grade-school training problems to competition mathematics. This pattern suggests that regularizing the shared update helps transfer learned reasoning patterns, without external knowledge or additional supervision. We additionally evaluate Loop Dropout on LoopUS-Qwen3-4B (Park et al., 2026) and Huginn (Geiping et al., 2025), extending the comparison to other model families in Appendix F.1.

Instruction tuning. On Ouro-1.4B, Loop Dropout improves HumanEval+ by 2.24 points and TruthfulQA MC2 by 0.87 points over LoRA. Table 2 also shows higher means on MMLU and IFEval. The six-benchmark average is 61.51, compared with 60.98 for LoRA. Appendix F.3 reports all nine instruction-tuning metrics, whose average is also higher for Loop Dropout than for LoRA.

Table 1: **Mathematical reasoning performance.** Accuracy in %, mean ± SD over three training seeds per method. Bold marks the highest mean within each model and benchmark.

	Ouro-1.4B			Ouro-2.6B		
Benchmark	LoRA	LoRA+	Loop Dropout	LoRA	LoRA+	Loop Dropout
GSM8K	85.34 ± 0.64	85.54 ± 0.23	**86.53 ± 0.44**	87.62 ± 0.52	87.21 ± 0.38	**88.73 ± 0.31**
MATH-500	38.87 ± 1.67	38.13 ± 0.42	**47.53 ± 2.61**	42.87 ± 1.17	43.40 ± 0.53	**47.20 ± 0.53**

Table 2: **Instruction-tuning performance.** Tülu 2 recipe; accuracy in %, mean ± SD over three training seeds. HumanEval+ uses EvalPlus; IFEval reports prompt-level strict accuracy. Average uses the six-benchmark set in Appendix F.3, computed per seed. Bold marks the higher mean in each column.

Method	HumanEval+	MMLU	TruthfulQA MC2	IFEval	Average (6)
LoRA	67.68 ± 3.05	68.64 ± 0.40	47.32 ± 1.02	46.33 ± 1.26	60.98 ± 0.66
Loop Dropout	**69.92 ± 0.35**	**68.91 ± 0.13**	**48.19 ± 0.68**	**46.46 ± 1.02**	**61.51 ± 0.09**

Table 3: **Comparison with existing adaptation methods.** Ouro-1.4B, rank 16, mathematical recipe. Accuracy is mean ± SD over three training seeds; training time and peak memory are their means. Displayed costs use an H100 80GB and exclude learning-rate search and evaluation. All methods use 15.1M adapter parameters.

Method	GSM8K	MATH-500	Train (min)	Memory (GB)
LoRA	85.34 ± 0.64	38.87 ± 1.67	110.6	45.36
LoRA+	85.54 ± 0.23	38.13 ± 0.42	112.8	45.40
CoTo-on-Ouro	85.60 ± 0.76	37.80 ± 1.06	94.1	45.28
LoRA Dropout	85.77 ± 0.70	42.13 ± 0.31	444.9	45.37
Loop Dropout	**86.53 ± 0.44**	**47.53 ± 2.61**	128.9	45.36

4.3 Comparison with Existing Adaptation Methods

We compare Loop Dropout with alternative low-rank adaptation and regularization methods under the same mathematical training recipe and hyperparameter tuning budget. Table 3 places accuracy alongside the measured cost of training.

Comparison with adapter regularizers. CoTo-on-Ouro shares each physical layer's adapter switch across its recurrent uses and increases the active fraction during training; LoRA Dropout introduces sparsity within the low-rank update. Loop Dropout couples masks on complete loop applications with inverse-survival rescaling, tailoring regularization to the recurrent use of the shared update. On MATH-500, it exceeds CoTo-on-Ouro by 9.73 points and LoRA Dropout by 5.40 points, with positive differences in all three training seeds against both methods. Its GSM8K margins are 0.94 and 0.76 points, respectively. These comparisons show that regularizing the repeated applications of the update yields stronger mathematical transfer than the evaluated alternatives. Appendix I gives the baseline configurations and per-seed results.

Efficiency comparison. Loop Dropout improves mathematical accuracy at the parameter count and inference computation of standard LoRA. Relative to LoRA Dropout, it reduces measured training time by 71% while improving accuracy on both benchmarks. Appendix D.3 specifies the hardware and timing procedure.

4.4 Early-Loop Adaptation

We now examine whether the task gains are accompanied by stronger early-loop adaptation, addressing the late-loop bias identified in Section 1. We first activate a trained adapter at only one of four loops, retain all four backbone loops, and measure the reduction in final-loop teacher-forced loss

Table 4: **GSM8K generation with one adapter application.** Ouro-1.4B after MetaMath-GSM fine-tuning; accuracy in %, mean ± SD over three seeds. Only the indicated adapter application is enabled, with all four backbone loops retained.

Method	Only loop 1	Only loop 2	Only loop 3	Only loop 4
LoRA	0.03 ± 0.04	0.30 ± 0.46	2.63 ± 3.07	6.07 ± 4.42
Loop Dropout	1.31 ± 0.69	64.72 ± 13.56	86.23 ± 0.84	86.18 ± 0.78

relative to the same frozen model. This intervention isolates how much the learned update reduces prediction loss when applied alone at each position, without shortening the recurrent computation.

Reducing the late-loop bias. Figure 1b quantifies the imbalance after standard LoRA fine-tuning on GSM8K: Loop Dropout narrows the gap in loss reduction between the first and final applications from 20.5 to 4.9 percentage points. At the higher learning rate of the same recipe (Appendix K), LoRA's loss reduction falls from 37.0% at the fourth loop to 11.3% at the first, while Loop Dropout retains 33.8–38.4% across positions. The improvement is largest at the earliest loop, strengthening the applications that standard fine-tuning leaves least effective.

Generation from individual applications. After MetaMath-GSM fine-tuning, we evaluate GSM8K generation with the adapter active at one loop while retaining all four backbone loops. Table 4 compares all four activation positions. At the second loop, Loop Dropout reaches 64.72% accuracy, compared with 0.30% for LoRA; at the third and fourth loops, it reaches 86.23% and 86.18%, compared with 2.63% and 6.07%. These results show that the update trained with Loop Dropout can support generation with a single application at loops two through four.

With every adapter application enabled, the readout diagnostic in Appendix K.2 also shows lower teacher-forced loss at every MATH-500 readout, most strongly at the first loop. This connects stronger early-loop prediction to the normal adapted computation.

4.5 Ablation Studies

The ablations examine the contributions of loop-level masking and inverse-survival rescaling, and how their benefit depends on sharing the update.

Loop-level masking and rescaling. The controls separate expected update strength, mask granularity and variation across loops. The *unscaled* variant uses gates in $\{0, 1\}$, while the *module-wise* variant retains rescaling but samples masks independently for each adapted linear map. A *dose control* draws the same gates as Loop Dropout and replaces them by their mean across loops, preserving their sum for each example while removing differences between loops. It can disable all adapter applications together, but never selectively removes an individual application.

Loop Dropout achieves the highest accuracy on both benchmarks in Table 5. Coupling the loop masks with rescaling improves over unscaled masking, and masking complete applications gives higher

Table 5: **Ablating loop-level masking and inverse-survival rescaling.** Ouro-1.4B, rank 16, MetaMath-GSM recipe. Accuracy in %, mean ± SD over three seeds. All rules use learning rate 10^{-4}, and noise strength is $c = 1$. Per-seed results are in Tables 10 and 15.

Training rule	GSM8K	MATH-500
LoRA	85.32 ± 0.92	40.13 ± 1.62
Unscaled	84.71 ± 0.19	39.33 ± 1.70
Dose control	85.57 ± 0.88	38.60 ± 1.71
Module-wise	85.52 ± 0.62	41.13 ± 0.76
Low-rank weight noise	85.70 ± 0.54	39.13 ± 0.83
Parallel noise	85.62 ± 0.64	37.93 ± 1.33
Loop Dropout	**87.57 ± 0.55**	**44.00 ± 1.40**

Table 6: **Sharing × masking on Ouro-1.4B.** MetaMath-GSM recipe with independent development selection over five learning rates per method; accuracy in % for training seed 101. All adapters are active at inference.

			Without loop mask		With loop mask	
Adapters	Rank	Params	GSM8K	MATH-500	GSM8K	MATH-500
Shared	16	15.1M	84.69	37.00	87.04	48.40
Independent	4	15.1M	84.08	39.60	87.57	44.40
Independent	16	60.6M	84.53	39.40	86.43	43.80

MATH-500 accuracy than independent module-wise masks. These comparisons support training across combinations of complete applications while preserving expected update strength.

Matched perturbations. We compare random low-rank weight noise and scalar noise along the learned update, each matched to the expected perturbation energy of Loop Dropout. At unit strength, the scalar control also matches the mean and covariance of the Loop Dropout gate, but lacks its probability mass at zero and permits negative gates. Loop Dropout outperforms both controls on both benchmarks; the additional configurations in Table 15 retain this ordering. Matching the perturbation magnitude or the first two gate moments therefore does not reproduce the benefit of removing complete adapter applications.

Sharing and masking. Table 6 examines how the gains from masking depend on whether the update is shared across loops. Independent rank-four adapters match the total parameter count of the shared rank-16 adapter, while independent rank-sixteen adapters match its per-loop capacity. Each row compares masked and unmasked training under the same hyperparameter candidate budget. Masking improves both benchmarks for all three adapter settings. On MATH-500, the gain is 11.40 points with shared adapters, compared with 4.80 and 4.40 points with independent rank-four and rank-sixteen adapters. The shared row uses the independently tuned configurations in Table 3; Appendix D.2 specifies the training and evaluation settings for this comparison.

The larger-model comparison in Table 18 also distinguishes sharing from adapter capacity. On Ouro-2.6B, Loop Dropout exceeds independently tuned per-loop rank-four and rank-sixteen adapters by 3.60 and 5.80 points on MATH-500, respectively. Allocating separate updates to the loops therefore does not recover the transfer performance of the shared update trained with Loop Dropout.

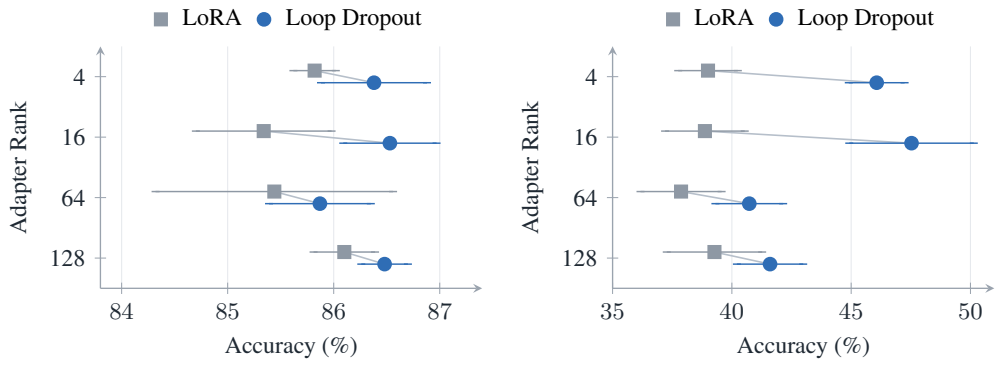
Additional training variants. Table 12 compares fixed-size masks, positional profiles, schedules and learned gates under the direct GSM8K recipe. No variant exceeds uniform Bernoulli masking at both learning rates, so we use uniform masks without an additional schedule or learned gates. Appendix G reports these variants in full.

4.6 Robustness Across Adapter Ranks and Recurrence Depths

We test whether the benefits of regularizing shared adaptation extend across adapter capacities and beyond the training recurrence depth. The rank study uses the MetaMath-GSM recipe, and the depth study uses the direct GSM8K recipe.

Gains across ranks. With the same tuning budget for each method and rank, Loop Dropout improves mean accuracy on both mathematical benchmarks at all four ranks in Figure 3; Table 20 gives the numerical results. MATH-500 gains are 7.07, 8.67, 2.87 and 2.33 points at ranks 4, 16, 64 and 128, respectively. At rank 16, the gain is positive in all three seeds and across all seven subject means. The decomposition in Appendix J.1 attributes 7.87 points of this 8.67-point gain to problems where both methods produce extractable answers. The transfer benefit therefore persists across adapter capacities after tuning each method independently.

Zero-shot generalization to deeper recurrence. We evaluate zero-shot generalization beyond the training depth by applying saved adapters at additional loops without further fine-tuning.



(a) GSM8K (b) MATH-500

Figure 3: **Mathematical accuracy across adapter ranks.** Ouro-1.4B after MetaMath-GSM fine-tuning. Markers and horizontal bars show mean ± SD over three seeds. Both methods use the same five-rate search budget at each rank, with $\alpha/r = 2$; Table 20 gives the values.

For adapters trained at four loops, Loop Dropout exceeds LoRA by 2.98 and 4.75 percentage points on GSM8K when evaluated at six and eight loops, respectively. At eight loops, Loop Dropout achieves 75.6% accuracy, compared with 70.8% for LoRA. Figure 4 shows the advantage growing from 1.90 points at the training depth of four loops to 4.75 points at eight loops. Adapters trained at six loops also give a 2.73-point gain over LoRA when evaluated at eight.

Loop Dropout achieves higher mean accuracy in all nine training–evaluation depth pairs, covering both matched and shifted recurrence depths. These results connect training across application patterns with stronger generalization beyond the depth used in fine-tuning. Appendix J.2 gives the absolute accuracies, uncertainty and evaluation-depth curves.

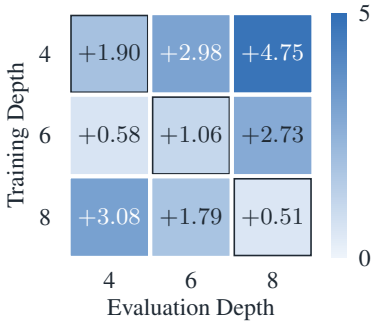


Figure 4: **Depth transfer on Ouro.** Loop Dropout gains over LoRA on GSM8K (percentage points). Outlined cells match training and evaluation depth.

5 Conclusion

Adapting looped language models requires a shared update that remains effective as hidden states evolve across the recurrence. We identify a pronounced late-loop bias in standard LoRA fine-tuning and introduce Loop Dropout to promote adaptation across loop positions. The method couples stochastic masking of complete adapter applications with inverse-survival rescaling, exposing the shared update to varying application patterns while preserving its expected strength. Every backbone loop remains active, and inference follows standard LoRA without additional trainable parameters or computation.

Experiments demonstrate improved mathematical adaptation and transfer across model sizes, adapter ranks and training recipes, with the largest gains on transfer to competition mathematics. Comparisons with LoRA variants, adapter regularizers and matched perturbations support the proposed training design. Further analysis shows stronger early-loop adaptation and an advantage over LoRA when adapters are evaluated beyond their training depth. These findings highlight effective adaptation across loops as a key consideration for fine-tuning looped language models, and establish Loop Dropout as a simple way to train reusable shared updates.

AI USE STATEMENT

We used ChatGPT-6 Astra and Claude Fable 5.1 to assist with writing and to identify potentially relevant work. All details were manually verified by the authors, who take full responsibility for the content of the paper.

ETHICS STATEMENT

This study uses publicly released models and existing benchmark datasets and does not recruit human participants or collect new user data. The adapted models inherit risks associated with their pretrained weights and fine-tuning data. Deployment requires application-specific reliability and safety evaluation beyond the benchmarks studied here.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. URL https://arxiv.org/abs/2108.07732.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, 2019. URL https://arxiv.org/abs/1909.01377.

Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to Ponder. In *ICML Workshop on Automated Machine Learning*, 2021. URL https://openreview.net/forum?id=1EuxRTe0WN.

Dan Biderman, Jacob Portes, Jose Javier Gonzalez Ortiz, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, Cody Blakeney, and John P. Cunningham. LoRA learns less and forgets less. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=aloEru2qCG.

Chris M. Bishop. Training with Noise is Equivalent to Tikhonov Regularization. *Neural Computation*, 7(1):108–116, 1995. doi: 10.1162/neco.1995.7.1.108.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Raj Dabre and Atsushi Fujita. Recurrent Stacking of Layers for Compact Neural Machine Translation Models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 6292–6299, 2019. doi: 10.1609/aaai.v33i01.33016292. URL https://ojs.aaai.org/index.php/AAAI/article/view/4590.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1807.03819.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient Finetuning of Quantized LLMs. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.14314.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-Adaptive Transformer. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1910.10073.

Angela Fan, Edouard Grave, and Armand Joulin. Reducing Transformer Depth on Demand with Structured Dropout. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SylO2yStDr.

Tianyu Fu, Yichen You, Zekai Chen, Guohao Dai, Huazhong Yang, and Yu Wang. Think-at-hard: Dynamic looped transformers for improved reasoning. *arXiv preprint arXiv:2511.08577*, 2025. URL https://arxiv.org/abs/2511.08577.

Yarin Gal and Zoubin Ghahramani. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, 2016. URL https://arxiv.org/abs/1512.05287.

Leo Gao et al. A framework for few-shot language model evaluation. https://github.com/EleutherAI/lm-evaluation-harness, 2023. Zenodo, version 0.4.

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. *arXiv preprint arXiv:2502.05171*, 2025. URL https://arxiv.org/abs/2502.05171.

Alex Graves. Adaptive Computation Time for Recurrent Neural Networks. *arXiv preprint arXiv:1603.08983*, 2016. URL https://arxiv.org/abs/1603.08983.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training Large Language Models to Reason in a Continuous Latent Space. In *Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Itxz7S4Ip3.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. LoRA+: Efficient low rank adaptation of large models. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2402.12354.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021a. URL https://arxiv.org/abs/2009.03300.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2021b. URL https://arxiv.org/abs/2103.03874.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-Efficient Transfer Learning for NLP. In *International Conference on Machine Learning*, 2019. URL https://proceedings.mlr.press/v97/houlsby19a.html.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Zhiqiang Hu, Lei Wang, Yihuai Lan, Wanyu Xu, Ee-Peng Lim, Lidong Bing, Xing Xu, Soujanya Poria, and Roy Ka-Wei Lee. LLM-Adapters: An adapter family for parameter-efficient fine-tuning of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://arxiv.org/abs/2304.01933.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, 2016. URL https://arxiv.org/abs/1603.09382.

Hamish Ivison, Yizhong Wang, Valentina Pyatkin, Nathan Lambert, Matthew Peters, Pradeep Dasigi, Joel Jang, David Wadden, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. Camels in a changing climate: Enhancing LM adaptation with Tulu 2. *arXiv preprint arXiv:2311.10702*, 2023. URL https://arxiv.org/abs/2311.10702.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. LoopFormer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. *arXiv preprint arXiv:2602.11451*, 2026. URL https://arxiv.org/abs/2602.11451.

Georgios Kaissis, David Mildenberger, Juan Felipe Gomez, Martin J. Menten, and Eleni Triantafillou. Step-resolved data attribution for looped transformers. *arXiv preprint arXiv:2602.10097*, 2026. URL https://arxiv.org/abs/2602.10097.

Damjan Kalajdzievski. A rank stabilization scaling factor for fine-tuning with LoRA. *arXiv preprint arXiv:2312.03732*, 2023. URL https://arxiv.org/abs/2312.03732.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large Language Models are Zero-Shot Reasoners. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2205.11916.

Dawid J. Kopiczko, Tijmen Blankevoort, and Yuki M. Asano. VeRA: Vector-based Random Matrix Adaptation. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NjNfLdxr3A.

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. URL https://arxiv.org/abs/1909.11942.

Cheolhyoung Lee, Kyunghyun Cho, and Wanmo Kang. Mixout: Effective regularization to finetune large-scale pretrained language models. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1909.11299.

Yu-Ang Lee, Ching-Yun Ko, Pin-Yu Chen, and Mi-Yen Yeh. Learning rate matters: Vanilla LoRA may suffice for LLM fine-tuning. *arXiv preprint arXiv:2602.04998*, 2026. URL https://arxiv.org/abs/2602.04998.

Brian Lester, Rami Al-Rfou, and Noah Constant. The Power of Scale for Parameter-Efficient Prompt Tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021. URL https://aclanthology.org/2021.emnlp-main.243/.

Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. GSM-Plus: A comprehensive benchmark for evaluating the robustness of LLMs as mathematical problem solvers. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://arxiv.org/abs/2402.19255.

Xiang Lisa Li and Percy Liang. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, 2021. URL https://aclanthology.org/2021.acl-long.353/.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2305.20050.

Stephanie Lin, Jacob Hilton, and Owain Evans. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022. URL https://aclanthology.org/2022.acl-long.229/.

Yang Lin, Xinyu Ma, Xu Chu, Yujie Jin, Zhibang Yang, Yasha Wang, and Hong Mei. LoRA dropout as a sparsity regularizer for overfitting control. *arXiv preprint arXiv:2404.09610*, 2024. URL https://arxiv.org/abs/2404.09610.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2205.05638.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.01210.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2402.09353.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. The Flan Collection: Designing Data and Methods for Effective Instruction Tuning. In *International Conference on Machine Learning*, 2023. URL https://proceedings.mlr.press/v202/longpre23a.html.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1711.05101.

Sean McLeish, Ang Li, John Kirchenbauer, Dayal Singh Kalra, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Jonas Geiping, Tom Goldstein, and Micah Goldblum. Teaching Pretrained Language Models to Think Deeper with Retrofitted Recurrence. *arXiv preprint arXiv:2511.07384*, 2025. URL https://arxiv.org/abs/2511.07384.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. PiSSA: Principal Singular Values and Singular Vectors Adaptation of Large Language Models. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2404.02948.

Mang Ning, Enver Sangineto, Angelo Porrello, Simone Calderara, and Rita Cucchiara. Input perturbation reduces exposure bias in diffusion models. In *International Conference on Machine Learning*, 2023. URL https://arxiv.org/abs/2301.11706.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2203.02155.

Taekhyun Park, Yongjae Lee, Dohee Kim, and Hyerim Bae. LoopUS: Recasting Pretrained LLMs into Looped Latent Refinement Models. *arXiv preprint arXiv:2605.11011*, 2026. URL https://arxiv.org/abs/2605.11011.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. AdapterDrop: On the efficiency of adapters in transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021. URL https://arxiv.org/abs/2010.11918.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2502.17416.

Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. Recurrent Dropout without Memory Loss. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, 2016. URL https://aclanthology.org/C16-1165/.

Vera Soboleva, Aibek Alanov, Andrey Kuznetsov, and Konstantin Sobolev. T-LoRA: Single image diffusion model customization without overfitting. *arXiv preprint arXiv:2507.05964*, 2025. URL https://arxiv.org/abs/2507.05964.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

Jan-Martin O. Steitz and Stefan Roth. Adapters strike back. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. URL https://arxiv.org/abs/2406.06820.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, 2023. URL https://aclanthology.org/2023.findings-acl.824/.

Guo Tang, Shixin Jiang, Heng Chang, Nuo Chen, Yuhan Li, Huiming Fan, Jia Li, Ming Liu, and Bing Qin. LoopRPT: Reinforcement Pre-Training for Looped Language Models. *arXiv preprint arXiv:2603.19714*, 2026. URL https://arxiv.org/abs/2603.19714.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpaca: A Strong, Replicable Instruction-Following Model. Stanford Center for Research on Foundation Models, 2023. URL https://crfm.stanford.edu/2023/03/13/alpaca.html.

Joshua Vendrow, Edward Vendrow, Sara Beery, and Aleksander Madry. Do large language model benchmarks test reliability? *arXiv preprint arXiv:2502.03461*, 2025. URL https://arxiv.org/abs/2502.03461.

Ivan Viakhirev, Kirill Borodin, Amirah Almutairi, Serguei Barannikov, Maxim Abramov, and Grach Mkrtchian. Think Shallow, Solve Deep: Controlling Recurrent Dynamics for Reliable Test-Time Depth. *arXiv preprint arXiv:2608.18222*, 2026. URL https://arxiv.org/abs/2608.18222.

Stefan Wager, Sida Wang, and Percy Liang. Dropout training as adaptive regularization. In *Advances in Neural Information Processing Systems*, 2013. URL https://arxiv.org/abs/1307.1493.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of Neural Networks using DropConnect. In *International Conference on Machine Learning*, pp. 1058–1066, 2013. URL https://proceedings.mlr.press/v28/wan13.html.

Shaowen Wang, Linxi Yu, and Jian Li. LoRA-GA: Low-rank adaptation with gradient approximation. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2407.05000.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-Instruct: Aligning Language Models with Self-Generated Instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023. URL https://aclanthology.org/2023.acl-long.754/.

Zhengbo Wang, Jian Liang, Ran He, Zilei Wang, and Tieniu Tan. LoRA-Pro: Are low-rank adapters properly optimized? In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2407.18242.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural*

Language Processing: System Demonstrations, 2020. URL https://aclanthology.org/2020.emnlp-demos.6/.

Kevin Xu and Issei Sato. On expressive power of looped transformers: Theoretical analysis and enhancement via timestep encoding. In *International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2410.01405.

Xiao-Wen Yang, Ziyu Han, Xi-Hua Zhang, Wen-Da Wei, Jie-Jing Shao, Lan-Zhe Guo, and Yu-Feng Li. Stabilizing Recurrent Dynamics for Test-Time Scalable Latent Reasoning in Looped Language Models. *arXiv preprint arXiv:2605.26733*, 2026. URL https://arxiv.org/abs/2605.26733.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. MetaMath: Bootstrap your own mathematical questions for large language models. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.12284.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. AdaLoRA: Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=lq62uWRJjiY.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023. URL https://arxiv.org/abs/2311.07911.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason Eshraghian. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025. URL https://arxiv.org/abs/2510.25741.

Shaobin Zhuang, Yiwei Guo, Yanbo Ding, Kunchang Li, Xinyuan Chen, Yaohui Wang, Fangyikang Wang, Ying Zhang, Chen Li, and Yali Wang. TimeStep Master: Asymmetrical mixture of timestep LoRA experts for versatile and efficient diffusion models in vision. In *International Conference on Machine Learning*, 2025a. URL https://arxiv.org/abs/2503.07416.

Zhan Zhuang, Xiequn Wang, Wei Li, Yulong Zhang, Qiushi Huang, Shuhao Chen, Xuehao Wang, Yanbin Wei, Yuhe Nie, Kede Ma, Yu Zhang, and Ying Wei. Come together, but not right now: A progressive strategy to boost low-rank adaptation. In *International Conference on Machine Learning*, 2025b. URL https://arxiv.org/abs/2506.05713.

APPENDIX

A RELATED WORK

A.1 PARAMETER SHARING AND RECURRENT COMPUTATION

Architectures and latent reasoning. Universal Transformers, recurrently stacked translation models and ALBERT reuse parameters across depth (Dehghani et al., 2019; Dabre & Fujita, 2019; Lan et al., 2020). Deep equilibrium models instead define representations through a fixed point of a shared transformation (Bai et al., 2019). Huginn and Ouro bring recurrent depth to pretrained language models (Geiping et al., 2025; Zhu et al., 2025), while theoretical studies characterize the expressive benefits of looping and loop-index encoding (Saunshi et al., 2025; Xu & Sato, 2025). Recurrence can also be introduced into existing pretrained models: McLeish et al. (2025) use a curriculum of recurrence depths, and LoopUS combines block decomposition with selective gates, random deep supervision and adaptive exiting (Park et al., 2026). Coconut provides a related form of latent computation by feeding a model's final hidden state back as the next input embedding (Hao et al., 2025). These approaches establish several ways to reuse computation; our study takes the pretrained recurrence as given and trains a shared downstream update within it.

Adaptive depth and recurrent training. Adaptive Computation Time and PonderNet learn how much computation to perform before producing a prediction (Graves, 2016; Banino et al., 2021). Depth-Adaptive Transformers make predictions at different layers, using untied layers rather than repeatedly applying one block (Elbayad et al., 2020). For looped models, LoopFormer aligns trajectories of different lengths through shortcut consistency, while Think-at-Hard combines selective iteration with depth-aware LoRA modules for hard-token refinement (Jeddi et al., 2026; Fu et al., 2025). These choices alter the computation budget, the depth-dependent transformation, or both. Loop Dropout keeps every backbone loop active and uses the same adapter at every loop during inference; its randomization concerns the adapter's training-time participation.

Recent work also changes the training of recurrent dynamics directly. STARS combines random loop sampling with Jacobian spectral-radius regularization (Yang et al., 2026). Viakhirev et al. (2026) relate depth extrapolation to finite-time dynamics, studying fixed-point training on small reasoners and a separate latent-anchoring LoRA objective for Huginn. LoopRPT applies reinforcement signals to latent steps using an exponential-moving-average teacher and noisy latent rollouts (Tang et al., 2026). Our training rule uses the ordinary target-token loss and randomizes complete applications of a shared low-rank update, without an auxiliary state loss or teacher. Our depth-generalization experiments evaluate the learned update at recurrence depths beyond those used in fine-tuning.

A.2 PARAMETER-EFFICIENT ADAPTATION

What is trained. Parameter-efficient fine-tuning can add bottleneck adapters (Houlsby et al., 2019), optimize continuous prefixes or input prompts (Li & Liang, 2021; Lester et al., 2021), or learn vectors that scale activations (Liu et al., 2022). LLM-Adapters evaluates several adapter families for language-model reasoning tasks (Hu et al., 2023). LoRA represents weight updates with trainable low-rank factors (Hu et al., 2022), allowing the learned update to be merged into the frozen weights. AdaLoRA distributes a limited rank budget according to the importance of weight updates, QLoRA trains adapters through a quantized frozen backbone, and VeRA learns scaling vectors over shared frozen random factors (Zhang et al., 2023; Dettmers et al., 2023; Kopiczko et al., 2024). These methods change the parameter or memory budget of adaptation. We hold the LoRA parameterization fixed and study how its update is trained across repeated applications.

How the factors are optimized. LoRA+ assigns different learning rates to the two factors, rsLoRA changes their rank-dependent scaling, and DoRA separates weight magnitude and direction (Hayou et al., 2024; Kalajdzievski, 2023; Liu et al., 2024). PiSSA initializes trainable factors from principal singular components of pretrained weights, whereas LoRA-GA uses gradient information to initialize the adapter (Meng et al., 2024; Wang et al., 2024). LoRA-Pro modifies the low-rank optimization to approximate full fine-tuning updates (Wang et al., 2025). LoRA and full fine-tuning can also differ in fitting and retention (Biderman et al., 2024), while learning-rate sensitivity can change comparisons among adapters (Lee et al., 2026). Our experiments compare learning rates, ranks, independent

per-loop factors and LoRA variants. Loop Dropout retains the LoRA parameterization and couples masking of shared adapter applications with inverse-survival rescaling.

A.3 Stochastic Regularization

The object and granularity of masking. Dropout masks activations and DropConnect masks individual weights (Srivastava et al., 2014; Wan et al., 2013). Stochastic depth removes residual branches during training, while LayerDrop applies structured layer removal to Transformers and supports extracting shallower networks for inference (Huang et al., 2016; Fan et al., 2020). Loop Dropout masks the adaptation branch at a loop while retaining the complete pretrained transformation. Thus the model's depth stays fixed even when the learned update is absent from some loops. The module-wise and dose controls test whether coupling the adapted modules through one loop-level mask matters beyond perturbing individual modules or changing the total update strength.

Mask dependence across repeated computation is also a central issue in recurrent dropout. Gal & Ghahramani (2016) reuse a dropout mask across sequence time steps, and Semeniuta et al. (2016) drop recurrent candidate updates in a way designed to preserve long-term memory. Our recurrence is over depth for the same token sequence. We sample independently across loops, share each mask across modules and token positions, and apply it only to the fine-tuning update. This makes the mask's scope and dependence structure different from dropout on the recurrent hidden state or on the full recurrent weights.

Regularizing pretrained adapters. Mixout regularizes adaptation toward pretrained parameters (Lee et al., 2020). AdapterDrop removes adapter layers, Adapters Strike Back applies stochastic depth to adapter branches, and LoRA Dropout introduces sparsity into low-rank updates (Rücklé et al., 2021; Steitz & Roth, 2024; Lin et al., 2024). CoTo progressively increases adapter activation probabilities and studies layer-wise contributions and optimization (Zhuang et al., 2025b). These methods regularize the spatial structure or training schedule of adapters. Loop Dropout trains different occurrences of the same shared update in different combinations, with inverse-survival scaling and all occurrences active at inference. Untying the per-loop adapters changes which parameters each mask exposes to training, which motivates our equal-rank and equal-parameter comparisons.

Noise and iterative state perturbations. Classical noise regularization and analyses of dropout connect small perturbations to local sensitivity penalties (Bishop, 1995; Wager et al., 2013). Our gate-space expansion has this interpretation locally, whereas the finite-mask mixture in Eq. (6) is exact. Diffusion models provide another setting in which parameters are reused over evolving states: TimeStep Master allocates timestep LoRA experts, and T-LoRA adapts the rank to the denoising timestep (Zhuang et al., 2025a; Soboleva et al., 2025). Input perturbation changes the states seen during diffusion training to address a training–sampling mismatch (Ning et al., 2023). In our unrolled computation, removing an earlier application changes the model-produced state received by later applications of the same update. The matched-noise and training-rescaling controls test specific alternatives directly.

A.4 Supervision and Evaluation

Instruction adaptation has been studied through human demonstrations and preference feedback (Ouyang et al., 2022), model-generated instructions (Wang et al., 2023), and diverse task mixtures (Longpre et al., 2023; Ivison et al., 2023). For mathematical adaptation, MetaMath constructs additional training questions from existing mathematical data (Yu et al., 2024). Our two settings use existing supervised recipes and vary the update training rule within each recipe. The evaluation keeps mathematical accuracy (Cobbe et al., 2021; Hendrycks et al., 2021b; Lightman et al., 2024) separate from code correctness, truthfulness and instruction following (Chen et al., 2021; Liu et al., 2023; Austin et al., 2021; Lin et al., 2022; Zhou et al., 2023). Appendix E specifies the prompts, parsers and scoring rules for each setting.

B Implementation Details

Adapter placement. The Ouro recurrent block contains decoder layers with four attention projections (query, key, value and output) and three feed-forward projections (gate, up and down). Unless a control specifies otherwise, all seven projections receive shared LoRA factors, while embeddings, normalization and the output head remain frozen. The default rank is 16 and $\alpha = 32$, giving 15,138,816 trainable parameters for Ouro-1.4B and twice that number for Ouro-2.6B. The rank sweep keeps $\alpha/r = 2$; rsLoRA instead uses $\alpha/\sqrt{r}$. We initialize A with Kaiming-uniform weights and B to zero, store adapter parameters in float32, and cast their forward computation to the activation dtype.

Mask sampling. At each micro-batch, the implementation draws a $K \times B$ mask for the B examples and exposes the current loop to every adapted module. Each module multiplies its adapter output by the corresponding gate, broadcasting over token positions. A dropped application therefore removes all LoRA contributions from one loop while retaining its frozen operations. The gate is one at every loop during evaluation. A unit-gate shared adapter can be merged into the pretrained matrices in the usual way.

Sharing, scale and granularity controls. The unscaled control uses $g = b$. Per-loop adapters use a separate pair of factors for each loop, either at rank 16 per loop or at rank 4 to match the shared rank-16 parameter count. The dose control draws the same mask as Loop Dropout and sets every gate of an example to $K^{-1}\sum_t g_t$. It preserves the sampled gate sum, but removes loop-to-loop variation; the all-zero mask disables every application simultaneously. Module-wise masking draws independent masks for each adapted linear map and loop. The standard input-dropout baseline applies probability 0.1 dropout to the adapter input; it is distinct from the sparsity method named LoRA Dropout in Lin et al. (2024).

Matched noise controls. Both noise controls draw an event $a_t \sim \mathrm{Bernoulli}(p)$ per example and loop, with event draws paired to the removal events of Loop Dropout. Parallel noise uses

$$g_t = 1 + a_t c \xi_t / \sqrt{q}, \qquad \xi_t \sim \mathcal{N}(0, 1), \tag{8}$$

where the scalar is shared across modules and tokens at that loop. At $c = 1$, its mean and covariance equal those of the Loop Dropout gate. Low-rank weight noise instead perturbs each adapted matrix by

$$\widetilde{\Delta}_i = \Delta_i + \frac{a_t c \ \mathrm{stopgrad}(\|\Delta_i\|_F)}{\sqrt{q\, d_{\mathrm{out}} d_{\mathrm{in}} r_n}} G_{\mathrm{out}} G_{\mathrm{in}}, \tag{9}$$

where the two independent standard-Gaussian factors have shapes $d_{\mathrm{out}} \times r_n$ and $r_n \times d_{\mathrm{in}}$, with $r_n = 16$. Its expected perturbation energy is $c^2(p/q)\|\Delta_i\|_F^2$, matching Loop Dropout at $c = 1$. This is a random low-rank perturbation, not independent dense Gaussian noise on every weight.

Other masks and random-number streams. The structured variants use nonuniform loop probabilities, exactly sized subsets, random prefixes or suffixes, training schedules, or sensitivity-dependent probabilities, with rescaling by each loop's retention probability. Initialization and data order are fixed before masks are sampled, and masking and noise use separate random streams, so paired same-rank comparisons start from identical adapters as specified in Appendix D.2.

C Update Scaling and a Local Curvature View

C.1 First-Order Consistency Through the Recurrence

Proof of Lemma 1. Fix the input, Δ, K and q as in the lemma, and write

$$h_t^\eta(g) = F\big(h_{t-1}^\eta(g); W + \eta g_t \Delta\big), \qquad z_\eta(g) = \mathrm{Head}\big(h_K^\eta(g)\big), \tag{10}$$

with h_0 independent of η and g. At $\eta = 0$, all gate vectors give the same frozen trajectory h_t^0 and logits z_0. Define the state Jacobian $J_t = D_h F(h_{t-1}^0; W)$, the directional weight derivative $d_t = D_W F(h_{t-1}^0; W)[\Delta]$, and the head Jacobian $P = D\mathrm{Head}(h_K^0)$. Differentiating the recurrence at $\eta = 0$ gives

$$\dot{h}_t(g) = J_t \dot{h}_{t-1}(g) + g_t d_t, \qquad \dot{h}_0(g) = 0. \tag{11}$$

Unrolling this identity yields

$$\left.\frac{\partial z_\eta(g)}{\partial \eta}\right|_{\eta=0} = \sum_{t=1}^{K} g_t u_t, \qquad u_t = P J_K \cdots J_{t+1} d_t, \tag{12}$$

where the product is the identity for $t = K$. The vector u_t is the final-logit response to inserting the shared update at loop t, including its propagation through the remaining backbone loops. The same Δ appears in every d_t, while the state and propagation Jacobians can vary with t.

The smoothness assumptions and finite depth make the composed output twice continuously differentiable near the frozen computation. Consequently,

$$z_\eta(g) = z_0 + \eta \sum_{t=1}^{K} g_t u_t + O(\eta^2). \tag{13}$$

For fixed $q > 0$, the mask support is finite, so the remainder is uniform over the gate vectors b/q, b and $\mathbf{1}$. Taking expectations and using $\mathbb{E}[b_t/q] = 1$ and $\mathbb{E}[b_t] = q$ gives

$$\begin{aligned} \mathbb{E}_b\, z_\eta(b/q) &= z_0 + \eta \sum_{t=1}^{K} u_t + O(\eta^2), \\ \mathbb{E}_b\, z_\eta(b) &= z_0 + q\eta \sum_{t=1}^{K} u_t + O(\eta^2). \end{aligned} \tag{14}$$

Comparing these expressions with the expansions of $z_\eta(\mathbf{1})$ and $z_{q\eta}(\mathbf{1})$ proves Eq. (5).

An explicit remainder bound follows by writing $Z(a)$ for the final logits when loop t uses $W + a_t\Delta$, so that $z_\eta(g) = Z(\eta g)$. Choose a sufficiently small closed ball U about zero within the region of twice continuous differentiability. There is a finite M such that $\|D^2 Z(a)[v, v]\|_2 \le M\|v\|_2^2$ for all $a \in U$ and vectors v. For η small enough that the masked coefficients and their means lie in U, Taylor expansion around each mean cancels the expected linear term and gives

$$\begin{aligned} \left\|\mathbb{E}_b\, z_\eta(b/q) - z_\eta(\mathbf{1})\right\|_2 &\le \frac{M\eta^2}{2}\mathbb{E}\|b/q - \mathbf{1}\|_2^2 = \frac{MKp}{2q}\eta^2, \\ \left\|\mathbb{E}_b\, z_\eta(b) - z_{q\eta}(\mathbf{1})\right\|_2 &\le \frac{M\eta^2}{2}\mathbb{E}\|b - q\mathbf{1}\|_2^2 = \frac{MKpq}{2}\eta^2. \end{aligned} \tag{15}$$

The constants can depend on the fixed input, update and depth; the expansion holds with q fixed. The bound controls the nonlinear remainder, while Eq. (12) identifies the first-order adaptation preserved by inverse-survival rescaling. □

C.2 Exact Relations

For a fixed example and adapter Δ, let $\ell_\Delta(g)$ denote the loss under gates $g \in \mathbb{R}^K$. Loop Dropout uses $g_t = b_t/q$ with independent $b_t \sim \text{Bernoulli}(q)$ and $q = 1 - p$, so that $\mathbb{E}[g] = \mathbf{1}$ and $\text{Cov}(g) = \frac{p}{q} I$ as in Eq. (4). Equation (6) gives the exact expectation of the loss over the finite set of masks. At $p = 0.5$ and $K = 4$, it averages uniformly over 16 patterns, from no active update to all four applications. The all-zero pattern contributes a loss independent of the adapter; the other patterns provide different training contexts for its active applications. The total gate sum $D = \sum_t g_t$ satisfies $qD \sim \text{Binomial}(K, q)$, with $\mathbb{E}[D] = K$ and $\text{Var}(D) = Kp/q$. Thus the expected number of applications, counted with their scale, equals the K applications used at inference. At $p = 1/2$ and $K = 4$, its possible values are $0, 2, 4, 6, 8$ with probabilities $(1, 4, 6, 4, 1)/16$. The mean-preserving property is defined in parameter space; the recurrence retains its nonlinear dependence on the gate vector.

The unscaled rule has mean $q\mathbf{1}$ and covariance pqI. For any adapter Δ and mask $b \in \{0, 1\}^K$,

$$\ell_{\Delta/q}(b) = \ell_\Delta(b/q), \tag{16}$$

so the two rules parameterize the same family of masked computations. Let Δ_N and Δ_U denote the unscaled and rescaled adapters. By Eq. (16), the corresponding parameters $\Delta_N = \Delta_U/q$ represent

the same distribution of masked forward computations, $\ell_{\Delta_N}(b) = \ell_{\Delta_U}(b/q)$ for every b. This identity describes corresponding parameterizations; optimization additionally depends on the chosen factor initialization and learning rates. At unit-gate inference, the two corresponding updates differ by $1/q$; evaluating Δ_N with gate q restores the same effective update as evaluating Δ_U with gate one. The constant can thus be applied during training or folded into the adapter at inference; Loop Dropout applies it during training so that inference uses the unit gate of standard LoRA without a calibration constant, whereas the unscaled control in Table 5 trains and evaluates without it. Unscaled training samples its own inference gate with probability q^K; rescaled training instead centers its gate distribution at the inference gate.

C.3 Second-Order Approximation

To interpret sensitivity near the default inference gate, write $g = \mathbf{1} + \varepsilon$, with $\mathbb{E}[\varepsilon] = 0$ and covariance C. A Taylor expansion gives

$$\mathbb{E}\,\ell_\Delta(\mathbf{1}+\varepsilon) = \ell_\Delta(\mathbf{1}) + \frac{1}{2}\operatorname{tr}\big(C\nabla_g^2\ell_\Delta(\mathbf{1})\big) + R, \tag{17}$$

where R collects the expected higher-order terms. Let $v_t = \partial z/\partial g_t|_{g=\mathbf{1}}$ be the local logit sensitivity to gate t, let $J = [v_1, \ldots, v_K]$, and let H be the cross-entropy Hessian with respect to the stacked target-token logits, with the same averaging convention as the loss. The gate Hessian decomposes as

$$\nabla_g^2\ell_\Delta(\mathbf{1}) = J^\top H J + \sum_i \frac{\partial \ell_\Delta}{\partial z_i}\,\nabla_g^2 z_i, \tag{18}$$

and substituting $C = \frac{p}{q}I$ into Eq. (17) gives Eq. (7), with R collecting the second summand and the higher-order terms. Keeping only the Gauss–Newton part $J^\top H J$ is the gate-space analogue of local noise-regularization analyses and the adaptive-regularization view of dropout (Bishop, 1995; Wager et al., 2013). At $p = 1/2$ the gate perturbations have unit magnitude, so Eq. (7) is a local description of the objective, while Eq. (6) gives its exact form.

C.4 What the Controls Match

Let $P_\parallel = K^{-1}\mathbf{1}\mathbf{1}^\top$ and $P_\perp = I - P_\parallel$. The dose control replaces each sampled gate by D/K and hence has covariance $\frac{p}{q}P_\parallel$. Retaining exactly $m = qK$ uniformly sampled loops with gate $1/q$ gives covariance $\frac{p}{q}\frac{K}{K-1}P_\perp$. Independent Bernoulli masks retain both components. These covariance relations describe which local sensitivity directions enter Eq. (17); the distributions also differ in their support and higher moments. The exact mixture incorporates all of these distributional properties.

At unit strength, the parallel-noise control has the same mean and covariance as Loop Dropout, but includes negative and unbounded gates and lacks a point mass at zero. The comparison in Table 5 tests this continuous perturbation against finite masking with the same first two gate moments.

Finally, the unscaled expansion is centered at $q\mathbf{1}$ and has coefficient $pq/2$. Under the corresponding parameterizations in Eq. (16), the derivative scaling compensates for the difference between this coefficient and $p/(2q)$. The expansion therefore describes the local objective in each parameterization, with its sensitivities evaluated at the stated adapter and gate.

D Experimental Details

D.1 Models, Data and Splits

Models. Ouro-1.4B and Ouro-2.6B have recurrent blocks of 24 and 48 decoder layers, respectively, with hidden size 2048, 16 attention heads and vocabulary size 49,152. We use the base checkpoints with four loops unless a depth experiment specifies otherwise. The backbone uses bfloat16 and scaled-dot-product attention, and adaptive exit gates are disabled. Mathematical prompts do not use a chat template; instruction tuning uses the Tülu 2 turn format. Batched generation uses left padding with an attention mask that covers the recurrent cache.

MetaMath-GSM-100k. Following the data-processing recipe of LoRA-Pro (Wang et al., 2025), we retain MetaMathQA examples whose type contains "GSM", remove examples of at least 512 tokens under the Ouro tokenizer, and take the first 100,000 in source order for training. A further 500 GSM-type MetaMathQA examples, disjoint from the training examples, form the development split used for hyperparameter selection. Inputs use the Alpaca instruction template without an input field; targets are the reference response followed by the end-of-sequence token. We use the Ouro tokenizer and the adapter configuration and learning rates specified below.

Direct GSM8K. We train on 6,973 of the 7,473 training problems and hold out the remaining 500. Targets contain the reference solution with calculator annotations removed, retain the final "`#### N`" line, and end with the end-of-sequence token. This smaller recipe supports the broader learning-rate, mask and depth studies.

Tülu 2 instruction tuning. A fixed 100k-example draw from the mixture yields 98,415 trainable examples after filtering. Only assistant tokens enter the instruction-tuning loss; the mathematical recipes likewise use target-only loss. Examples beyond each recipe's sequence-length cap are dropped rather than truncated. Table 7 summarizes the three recipes.

Table 7: **Fine-tuning recipes.** Effective batch = micro-batch × gradient accumulation. The adapter recipes use AdamW (Loshchilov & Hutter, 2019) ($\beta_1 = 0.9$, $\beta_2 = 0.999$), zero weight decay, gradient clipping at 1.0, a cosine schedule decaying to 10% of the peak learning rate, and the final checkpoint.

	MetaMath-GSM-100k	GSM8K	Tülu 2-100k
Training examples	100,000	6,973	98,415
Epochs / optimizer steps	1 / 3,125	2 / ≈1,744	1 / 769
Effective batch	$8 \times 4 = 32$	$8 \times 1 = 8$	$8 \times 16 = 128$
Maximum sequence length	1,024	512	2,048
Warm-up	3%	5%	3%
Learning rates	See below	3e-5 to 6e-4	1e-4
Seeds	101–103	0–4 / 101–103	101–103
Evaluation	MetaMath evaluator	lm-eval (0-shot)	Appendix F.3

D.2 Hyperparameters

Adapters and seeds. Default adapters have rank 16 and $\alpha = 32$; the independently tuned rank study uses $r \in \{4, 16, 64, 128\}$ with $\alpha/r = 2$. LoRA+ uses a fourfold learning-rate multiplier for B relative to A. Studies with seeds 101–103 set the seed before attaching adapters, so paired same-rank methods share initial adapter weights and data order; the GSM8K learning-rate, depth and mask studies with seeds 0–4 use independent initializations. Each study block is reported separately.

MetaMath learning rates. The main mathematical comparisons and the independently tuned sharing and rank studies give each method five candidates: $\{1.25, 2.5, 5, 10, 20\} \times 10^{-5}$, or half these rates for the LoRA+ A factor. Each method chooses its rate by greedy-generation accuracy on the 500-example development split of Appendix D.1 with training seed 101, resolving ties toward the smaller rate. For the three-seed comparisons, seeds 102 and 103 are then trained at that rate. For shared rank-16 adapters, LoRA and Loop Dropout choose 2×10^{-4} and 5×10^{-5} on Ouro-1.4B, and 2×10^{-4} and 10^{-4} on Ouro-2.6B. CoTo and LoRA Dropout choose 2×10^{-4} and 10^{-4}, respectively, and LoRA+ chooses an A-factor rate of 5×10^{-5} on Ouro-1.4B and 10^{-4} on Ouro-2.6B.

The controlled comparison in Table 5 uses 10^{-4} for every training rule. The two-rate studies use $\{10^{-4}, 2 \times 10^{-4}\}$ for shared adapters and the unscaled and dose controls, and $\{5 \times 10^{-5}, 10^{-4}\}$ for the LoRA+ A factor. The module-wise control uses 10^{-4} in these studies. Table 13 gives the fixed-10^{-4} rank study. The noise controls use strength $c = 1$ at 10^{-4}; Table 15 adds a second configuration of each control at 2×10^{-4}, with $c = 2$ for parallel noise.

Sharing and masking. Table 6 compares training seed 101 for every configuration. Both independent-adapter ranks choose LR 2×10^{-4} with and without masking; shared LoRA and Loop Dropout use 2×10^{-4} and 5×10^{-5}, respectively. All masked configurations use $p = 0.5$ and inverse-survival rescaling. The diagnostics in Figure 6 use shared rank-16 checkpoints at these independently chosen rates, with seeds 101–103.

Other recipes. The direct GSM8K study evaluates $\{3\times10^{-5}, 10^{-4}, 2\times10^{-4}, 3\times10^{-4}, 6\times10^{-4}\}$, with five seeds at 10^{-4} and 3×10^{-4} and three elsewhere. The Tülu 2 comparison uses LR 10^{-4} for LoRA and Loop Dropout. All recipes use the final checkpoint.

Other looped transformers. Table 9 uses rank-16 adapters with $\alpha = 32$ and the MetaMath-GSM-100k recipe. LoopUS-Qwen3-4B uses eight loops, LR 10^{-4} and training seed 501. We select $p = 0.3$ for Loop Dropout on the development split from $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. Huginn uses 32 loops, LR 10^{-4} and $p = 0.5$ for Loop Dropout, with accuracy averaged over training seeds 301–303. Both methods use the same settings within each model, with $p = 0$ for LoRA and all adapter applications enabled at inference.

D.3 Compute

Training uses a single H100, H200 or B200 GPU and bfloat16, with Transformers 5.3.0 for LoopUS and 4.57.6 for the other models. The cost comparison in Table 3 uses H100 80GB measurements with PyTorch 2.8.0, scaled-dot-product attention and gradient checkpointing disabled for every reported method. All five methods use the same 100k examples, 3,125 optimizer steps and rank-16 adapters. Training time and peak memory are means over three training seeds. For each seed, training time covers one selected-configuration training run and excludes development search, generation, scoring and queue time. Table 8 reports the individual timings. The Ouro-2.6B five-candidate comparison uses B200 GPUs with PyTorch 2.7.1; its timing is kept separate from the H100 comparison. Direct GSM8K training at depth four takes roughly 12–14 minutes for Ouro-1.4B and 25–32 minutes for Ouro-2.6B.

Table 8: **Training time across seeds on H100 80GB.** Times are in minutes for seeds 101 / 102 / 103, with their mean and sample SD. Peak memory is the mean over the same runs.

Method	Per-seed time	Mean ± SD	Memory (GB)
LoRA	111.93 / 106.74 / 113.28	110.65 ± 3.45	45.36
LoRA+	111.20 / 107.74 / 119.40	112.78 ± 5.99	45.40
CoTo-on-Ouro	94.95 / 93.32 / 94.06	94.11 ± 0.82	45.28
LoRA Dropout	442.39 / 438.34 / 453.98	444.91 ± 8.12	45.37
Loop Dropout	130.86 / 128.16 / 127.65	128.89 ± 1.72	45.36

E Evaluation and Statistical Protocols

E.1 Prompts, Decoding and Scoring

MetaMath evaluation. The main mathematical evaluation follows the MetaMath scripts at revision `fe667b1`, using the Alpaca instruction prompt (Taori et al., 2023) and the response prefix "`Let's think step by step.`" used in zero-shot chain-of-thought prompting (Kojima et al., 2022). Evaluation is zero-shot and greedy, with the reference stop strings and maximum new-token budgets of 512 for GSM8K and 2,048 for MATH-500. The scorer extracts the answer after "`The answer is:`", then applies numeric comparison for GSM8K or the reference MATH normalization and equivalence rules. MATH-500 is the 500-problem subset released by Lightman et al. (2024).

Generation uses Hugging Face Transformers (Wolf et al., 2020) with left-padded batches and an attention mask that covers the recurrent cache. Every comparison uses the same prompt, decoding settings and scorer; Appendix H.2 additionally quantifies gains on problems where both methods provide extractable answers.

Direct GSM8K fine-tuning. These experiments use the zero-shot `gsm8k` task in lm-eval-harness (Gao et al., 2023), greedy decoding and a 256-token generation cap. Strict exact match compares the number after "####" with the reference on all 1,319 test problems. GSM8K-Platinum (1,209 relabelled problems) and GSM-Plus mini (2,400 perturbed problems) use the same extraction convention. This protocol is distinct from the MetaMath recipe and its scores are not pooled with that recipe.

Instruction-tuning evaluation. Appendix F.3 specifies the reported task metrics, generation budgets and official code/IFEval scoring rules. These use the instruction-tuned checkpoints and are kept separate from the mathematical recipe and raw-prompt base-model evaluations.

E.2 Seeds and Uncertainty

Main configurations use three training seeds; two learning rates of the GSM8K study use five. Tables report the mean and sample standard deviation over training seeds, which summarize training-seed variation; paired same-rank methods share initial adapter weights and data order as specified in Appendix D.2.

F Complete Benchmark Results

F.1 Other Looped Transformers

We also evaluate Loop Dropout on LoopUS-Qwen3-4B (Park et al., 2026) and Huginn (Geiping et al., 2025) under the MetaMath-GSM-100k recipe. Table 9 shows higher accuracy on both benchmarks for both models; on LoopUS-Qwen3-4B, the gains are 1.59 percentage points on GSM8K and 6.20 points on MATH-500. Appendix D.2 specifies their training settings.

Table 9: **Mathematical evaluation on other looped transformers.** Accuracy in %. Bold marks the higher value within each model and benchmark.

	LoopUS-Qwen3-4B		Huginn	
Benchmark	LoRA	Loop Dropout	LoRA	Loop Dropout
GSM8K	83.93	**85.52**	59.89	**60.11**
MATH-500	34.60	**40.80**	13.00	**13.80**

F.2 Per-Seed Results

Table 10: **Per-seed results of the mask-control comparison** on Ouro-1.4B under the MetaMath protocol, at the learning rates shown; accuracy in %.

Method	LR	GSM8K seeds 101 / 102 / 103	MATH-500 seeds 101 / 102 / 103
LoRA	1e-4	84.6 / 85.0 / 86.4	39.2 / 42.0 / 39.2
Loop Dropout	1e-4	87.7 / 87.0 / 88.0	44.0 / 45.4 / 42.6
Unscaled	1e-4	84.5 / 84.7 / 84.9	40.6 / 40.0 / 37.4
Unscaled	2e-4	83.8 / 83.8 / 84.8	35.6 / 38.0 / 37.4
Dose control	1e-4	85.4 / 84.8 / 86.5	36.8 / 40.2 / 38.8
Dose control	2e-4	85.5 / 84.9 / 85.9	37.8 / 42.4 / 37.8
Module-wise	1e-4	85.0 / 85.4 / 86.2	40.6 / 42.0 / 40.8

F.3 Instruction Tuning on Tülu 2

The instruction-tuning study uses a 100k-example draw from the Tülu 2 mixture, yielding 98,415 trainable examples after filtering. Ouro-1.4B is trained for 769 optimizer steps at context length 2,048 and effective batch size 128, with assistant-only loss and rank-16 adapters ($\alpha = 32$). LoRA and Loop

Dropout use learning rate 10^{-4}. Each method is trained with seeds 101–103, and same-seed methods share adapter initialization and data order. Table 11 reports all nine metrics computed for these checkpoints; Table 2 shows four benchmarks and the six-benchmark average. The six-benchmark average uses HumanEval+, MBPP+, MMLU, BBH, TruthfulQA MC2 and strict IFEval.

Table 11: **All downstream metrics after Tülu 2 instruction tuning of Ouro-1.4B.** Rank 16; accuracy in %, mean ± SD over three matched seeds. HumanEval+ and MBPP+ add the expanded EvalPlus tests to the original ones; IFEval reports prompt-level accuracy. The last three rows average the four code metrics, the six-benchmark set, and all nine metrics per seed; bold marks the highest average in each summary row.

Metric	LoRA	Loop Dropout
MMLU (0-shot)	68.64 ± 0.40	68.91 ± 0.13
BBH (3-shot CoT)	71.20 ± 0.14	70.78 ± 0.11
TruthfulQA MC2	47.32 ± 1.02	48.19 ± 0.68
HumanEval	71.54 ± 2.14	72.56 ± 0.61
HumanEval+	67.68 ± 3.05	69.92 ± 0.35
MBPP	75.57 ± 0.40	76.72 ± 0.53
MBPP+	64.73 ± 0.55	64.81 ± 0.79
IFEval strict	46.33 ± 1.26	46.46 ± 1.02
IFEval loose	50.59 ± 1.44	50.40 ± 0.11
Code average (4 metrics)	69.88 ± 1.21	**71.00 ± 0.46**
Average (6 benchmarks)	60.98 ± 0.66	**61.51 ± 0.09**
Average (9 metrics)	62.62 ± 0.78	**63.20 ± 0.14**

Evaluation. MMLU (Hendrycks et al., 2021a) (14,042 questions, 0-shot) and BBH (Suzgun et al., 2023) (27 tasks, 6,511 questions, 3-shot chain of thought) use lm-eval-harness (Gao et al., 2023); TruthfulQA MC2 averages the probability mass assigned to correct answers over 817 questions. The code evaluations use the pinned official EvalPlus scorer (Liu et al., 2023) on all 164 HumanEval (Chen et al., 2021) and 378 MBPP (Austin et al., 2021) tasks in its evaluation sets, with one greedy generation per task capped at 1,024 tokens; HumanEval+ and MBPP+ require passing the additional tests as well as the original ones. IFEval uses all 541 prompts, a 2,048-token cap and the pinned official evaluator with scoring seed zero, reporting prompt-level strict and loose accuracy. Code and IFEval generation use Tülu user/assistant turns. All configurations use the same environment, prompt definitions and benchmark instances.

Reading the table. Compared with LoRA, Loop Dropout increases the nine-metric average from 62.62 to 63.20, the six-metric average in Table 2 from 60.98 to 61.51, and the average over four code metrics from 69.88 to 71.00. It improves HumanEval+, MBPP and TruthfulQA MC2 by 2.24, 1.15 and 0.87 points, respectively; the MBPP improvement holds in every seed. On the remaining six metrics, Loop Dropout lies within about one point of LoRA. The seed standard deviations summarize training-seed variation, as described in Appendix E.2.

G Extended Ablations

G.1 Dropout Variants and Structural Controls

Rescaling and dropout probability. The GSM8K comparison favors rescaled dropout over unscaled dropout at both tested learning rates. At 10^{-4}, the accuracies are 80.6% for LoRA, 77.0% for unscaled dropout with $p = 0.5$, and 81.7% with rescaling; Table 5 gives the three-seed MetaMath comparison. Appendix C relates the two training parameterizations.

A three-seed probability study obtains 81.1 ± 0.5, 81.7 ± 0.2 and 81.6 ± 1.1 at $p = 0.25, 0.5, 0.75$, respectively, compared with 80.7 ± 0.2 for LoRA and 79.5 ± 0.5 for standard adapter input dropout. At the higher learning rate the corresponding means are 80.0, 81.5 and 81.0, versus 78.1 for LoRA. The middle probability gives the highest mean at both learning rates and is our default. Independent per-loop adapters remain below Loop Dropout under this recipe, with equal-rank accuracies of 80.1 / 76.9 and equal-budget accuracies of 79.7 / 79.5 at the two rates.

Table 12: **Structured, scheduled and adaptive masks** on the GSM8K recipe (Ouro-1.4B, strict exact match, %; two seeds, mean ± SD). Masks control adapter applications while all four backbone loops run; all variants keep the $1/P(\text{keep})$ rescaling. Bottom: five-seed follow-up of the two closest variants. No variant exceeds the uniform rule at both learning rates.

Variant	Mask	LR 1e-4	LR 3e-4
LoRA	none	80.6 ± 0.3	78.1 ± 1.0
Loop Dropout	uniform $p = 0.5$	81.7 ± 0.2	81.3 ± 0.7
Early-heavy profile	$p = (.75, .75, .25, .25)$	82.1 ± 0.2	79.4 ± 2.6
Late-heavy profile	$p = (.25, .25, .75, .75)$	80.0 ± 0.5	77.9 ± 1.0
Ramp down	$p = (.8, .6, .4, .2)$	80.7 ± 0.8	79.3 ± 0.4
Ramp up	$p = (.2, .4, .6, .8)$	80.0 ± 0.8	79.0 ± 2.6
Exactly two loops	random pair, $\times 2$	81.6 ± 0.4	78.2 ± 1.1
Exactly one loop	random loop, $\times 4$	81.0 ± 0.5	77.3 ± 2.4
Random prefix	loops $1..T$	79.9 ± 1.0	78.9 ± 0.1
Random suffix	loops $T..4$	79.9 ± 1.0	78.0 ± 0.5
Schedule p: $0.75 \to 0$	over training	81.0 ± 0.1	79.4 ± 0.2
Schedule p: $0 \to 0.75$	over training	81.3 ± 0.4	81.2 ± 0.8
Adaptive, more where sensitive	$p_t \propto$ sensitivity	81.4 ± 0.4	79.7 ± 1.2
Adaptive, less where sensitive	$p_t \propto 1/$sensitivity	81.2 ± 0.1	81.5 ± 0.6
Learned per-loop gates	no dropout	79.4 ± 1.0	80.1 ± 0.1
Learned gates + Loop Dropout	uniform $p = 0.5$	80.7 ± 0.8	82.8 ± 0.0
Early-heavy profile (5 seeds)	$p = (.75, .75, .25, .25)$	81.8 ± 0.7	79.3 ± 1.4
Learned gates + Loop Dropout (5 seeds)	uniform $p = 0.5$	81.2 ± 0.6	82.0 ± 0.8

Reading the variants. Uniform masking matches the early-heavy profile at the lower rate and exceeds it at the higher rate in the five-seed follow-up. At the higher rate, uniform masking also achieves higher accuracy than exactly sized subsets and schedules that anneal the dropout probability to zero. These comparisons support the uniform rule as a default across the two learning rates. Learning per-loop gates alone also remains below uniform masking on generation accuracy. Learned gates add trainable parameters and fall below uniform masking at 10^{-4}, so we keep the parameter-free rule.

H Rank and Noise Studies

These suites use Ouro-1.4B at depth four with the MetaMath-GSM-100k recipe and the same test documents, prompts and scoring rules as the rank-16 reference.

H.1 Rank Dependence at a Common Learning Rate

Table 13: **Paired comparisons at different shared adapter ranks.** Three seeds, LR 10^{-4} and $\alpha/r = 2$. Accuracy is in %; differences are in percentage points.

Rank	Benchmark	LoRA	Loop Dropout	Difference
4	GSM8K	85.82 ± 0.20	86.38 ± 0.50	$+0.56$
4	MATH-500	39.00 ± 1.25	46.07 ± 1.17	$+7.07$
64	GSM8K	85.44 ± 1.12	87.01 ± 0.49	$+1.57$
64	MATH-500	37.87 ± 1.70	43.33 ± 1.85	$+5.47$
128	GSM8K	85.06 ± 0.62	86.48 ± 0.22	$+1.42$
128	MATH-500	34.80 ± 1.00	41.60 ± 1.39	$+6.80$

The MATH-500 difference is positive for each of the nine rank-by-seed pairs, and the mean GSM8K difference is positive at every rank. Table 5 reports the rank-16 comparison at the same learning rate.

Table 14: **Decomposing the net MATH-500 accuracy difference.** Missing-answer rates are percentages. Both net-difference components use all 500 problems as denominator and sum to the overall gain in percentage points. This decomposition does not change the official scorer.

	Missing answers (%)		Net gain (points)		
Rank	LoRA	Loop Dropout	Both extractable	Missing in either	Total
4	8.80	9.60	6.80	0.27	7.07
64	7.60	5.60	4.80	0.67	5.47
128	7.80	5.67	6.00	0.80	6.80

H.2 Answer Availability and the MATH-500 Gain

Table 14 shows that most of the gain occurs where both methods produce extractable answers. Across the three ranks, improvements on pairs with two extractable answers account for 6.80, 4.80 and 6.00 points of the total gains. The advantage therefore persists on problems for which both methods satisfy the answer-extraction requirement.

H.3 Per-Seed Measurements

Table 15 reports each training seed separately for the rank sweep and for the noise controls, including a second configuration of each control trained at 2×10^{-4}.

Table 15: **Per-seed results of the additional suites.** Seeds are ordered 101 / 102 / 103. Accuracy is in %; c is the noise strength and is inapplicable to LoRA and Loop Dropout.

Method	Rank	LR	c	GSM8K	MATH-500
LoRA	4	1e-4	–	85.90 / 85.60 / 85.97	38.0 / 38.6 / 40.4
Loop Dropout	4	1e-4	–	86.13 / 86.05 / 86.96	45.6 / 47.4 / 45.2
LoRA	64	1e-4	–	84.15 / 86.20 / 85.97	36.2 / 39.6 / 37.8
Loop Dropout	64	1e-4	–	87.57 / 86.66 / 86.81	44.4 / 41.2 / 44.4
LoRA	128	1e-4	–	84.91 / 84.53 / 85.75	35.8 / 33.8 / 34.8
Loop Dropout	128	1e-4	–	86.35 / 86.35 / 86.73	43.2 / 40.8 / 40.8
Low-rank weight noise	16	1e-4	1	85.22 / 85.60 / 86.28	38.2 / 39.8 / 39.4
Low-rank weight noise	16	2e-4	1	85.90 / 84.08 / 86.81	37.8 / 39.0 / 35.6
Parallel noise	16	1e-4	1	85.29 / 85.22 / 86.35	36.8 / 37.6 / 39.4
Parallel noise	16	2e-4	2	85.06 / 86.13 / 86.13	38.4 / 37.4 / 36.8

I Additional Baseline Comparisons

This section gives the baseline configurations and per-seed measurements supporting Sections 4.2, 4.3 and 4.5. Three-seed summaries use the same three seeds for every method in their block.

I.1 Baseline Configurations and Per-Seed Results

CoTo-on-Ouro. The CoTo adaptation follows the physical-layer schedule of the official implementation. All seven adapted projections within a physical layer share a switch, and that switch remains fixed across recurrent uses and the effective optimizer batch. The keep probability starts at 0.1 and increases to one over the first 75% of optimizer steps, with a nonempty set of active layers and no inverse-survival scaling. The final checkpoint is evaluated with all adapters active. Each run uses the same 100k training examples, 3,125 updates and rank-16 parameterization as the other mathematical comparisons. The chosen learning rate is 2×10^{-4} from the five-rate grid $\{1.25, 2.5, 5, 10, 20\} \times 10^{-5}$, with the choice fixed before test evaluation. Table 16 gives the per-seed results.

LoRA+. LoRA+ chooses an A-factor rate of 5×10^{-5} on Ouro-1.4B from the halved five-candidate grid, with the fourfold multiplier for B.

Table 16: **Per-seed CoTo-on-Ouro results.** Full GSM8K and MATH-500 test sets under the MetaMath protocol; accuracy in %. The summary uses the sample standard deviation.

Seed	GSM8K	MATH-500
101	84.84	37.40
102	85.60	39.00
103	86.35	37.00
Mean ± SD	85.60 ± 0.76	37.80 ± 1.06

LoRA Dropout. We evaluate LoRA Dropout (Lin et al., 2024) with three training seeds on Ouro-1.4B. The baseline uses dropout probability 0.5 and four training masks, with deterministic inference using the expected masked update and no test-time ensemble, since ensembling would multiply inference cost by the number of masks. Its learning rate is 10^{-4}, chosen from the same five-candidate development grid as the other regularizers. Table 17 gives the per-seed results of both methods.

Table 17: **LoRA+ and LoRA Dropout across three training seeds.** Ouro-1.4B, rank 16, MetaMath-GSM recipe; accuracy in %.

	LoRA+		LoRA Dropout	
Seed	GSM8K	MATH-500	GSM8K	MATH-500
101	85.29	37.80	85.44	42.20
102	85.60	38.60	85.29	41.80
103	85.75	38.00	86.58	42.40
Mean ± SD	85.54 ± 0.23	38.13 ± 0.42	85.77 ± 0.70	42.13 ± 0.31

I.2 Adapter Sharing on Ouro-2.6B

Table 18 compares shared and independent adapters with five learning-rate candidates per method, development selection on seed 101 and three training seeds at the chosen rate. Independent rank-four adapters match the 30.3M parameters of the shared rank-16 adapter; independent rank-sixteen adapters use 121.1M parameters. The shared rows are the same checkpoints as in Table 1; Table 19 lists the per-seed results and chosen learning rates.

Table 18: **Shared and independent adapters on Ouro-2.6B.** MetaMath-GSM recipe, mean ± SD over three training seeds, in %. All backbone loops and all trained adapter applications are active at inference.

Method	Params	GSM8K	MATH-500
Shared LoRA, rank 16	30.3M	87.62 ± 0.52	42.87 ± 1.17
LoRA+, rank 16	30.3M	87.21 ± 0.38	43.40 ± 0.53
Independent, rank 4	30.3M	85.95 ± 2.59	43.60 ± 0.40
Independent, rank 16	121.1M	85.87 ± 0.87	41.40 ± 1.71
Loop Dropout, rank 16	30.3M	**88.73 ± 0.31**	**47.20 ± 0.53**

Table 19: **Per-seed Ouro-2.6B results after independent tuning.** Seeds are ordered 101 / 102 / 103; accuracy in %.

Method	LR	GSM8K	MATH-500
Shared LoRA, rank 16	2×10^{-4}	88.02 / 87.04 / 87.79	42.00 / 44.20 / 42.40
LoRA+, rank 16	10^{-4}	87.04 / 86.95 / 87.64	43.80 / 42.80 / 43.60
Independent, rank 4	2×10^{-4}	86.88 / 87.95 / 83.02	43.20 / 44.00 / 43.60
Independent, rank 16	2×10^{-4}	85.90 / 84.99 / 86.73	39.80 / 41.20 / 43.20
Loop Dropout, rank 16	10^{-4}	89.01 / 88.78 / 88.40	47.80 / 47.00 / 46.80

J Additional Robustness Results

This section gives the full numerical results supporting Section 4.6. The rank study uses the MetaMath-GSM recipe; the depth and shifted-test studies use the direct GSM8K recipe.

J.1 Adapter Rank with Independent Tuning

Both methods use the same five-rate grid $\{1.25, 2.5, 5, 10, 20\} \times 10^{-5}$ at each rank, with $\alpha/r = 2$. Each rank and method fixes its configuration independently before test evaluation, then uses that configuration for seeds 101–103. Figure 3 and Table 20 show positive mean differences on both benchmarks at every rank. The rank-16 entries are the same three-seed results as in Table 3. The shared row in Table 6 uses the corresponding seed-101 checkpoints. Table 13 reports the rank study at 10^{-4}.

Table 20: **Independent tuning at each adapter rank.** Ouro-1.4B, three seeds per rank and method; full GSM8K and MATH-500 evaluation, mean $\pm$ SD in %. Both methods have five learning-rate candidates at each rank.

Rank	Method	GSM8K	MATH-500
4	LoRA	85.82 ± 0.20	39.00 ± 1.25
4	Loop Dropout	86.38 ± 0.50	46.07 ± 1.17
16	LoRA	85.34 ± 0.64	38.87 ± 1.67
16	Loop Dropout	86.53 ± 0.44	47.53 ± 2.61
64	LoRA	85.44 ± 1.12	37.87 ± 1.70
64	Loop Dropout	85.87 ± 0.48	40.73 ± 1.42
128	LoRA	86.10 ± 0.29	39.27 ± 2.00
128	Loop Dropout	86.48 ± 0.22	41.60 ± 1.39

Table 21: **Per-seed results after independent rank-wise tuning.** Seeds are ordered 101 / 102 / 103; accuracy in %. The same learning rate is used for all three seeds in a row.

Rank	Method	LR	GSM8K	MATH-500
4	LoRA	10^{-4}	85.90 / 85.60 / 85.97	38.00 / 38.60 / 40.40
4	Loop Dropout	10^{-4}	86.13 / 86.05 / 86.96	45.60 / 47.40 / 45.20
16	LoRA	2×10^{-4}	84.69 / 85.37 / 85.97	37.00 / 40.20 / 39.40
16	Loop Dropout	5×10^{-5}	87.04 / 86.35 / 86.20	48.40 / 49.60 / 44.60
64	LoRA	10^{-4}	84.15 / 86.20 / 85.97	36.20 / 39.60 / 37.80
64	Loop Dropout	2×10^{-4}	85.60 / 85.60 / 86.43	42.00 / 39.20 / 41.00
128	LoRA	2.5×10^{-5}	85.90 / 85.97 / 86.43	37.00 / 40.80 / 40.00
128	Loop Dropout	10^{-4}	86.35 / 86.35 / 86.73	43.20 / 40.80 / 40.80

Where the rank-16 transfer gain occurs. Table 22 partitions the same MATH-500 test set by subject and difficulty. Loop Dropout improves mean accuracy in all seven subjects and all five difficulty levels, with the largest level-wise difference at level 3. The improvement therefore spans the subject groups rather than coming from a single category.

Table 22: **MATH-500 breakdown for independently tuned rank-16 adapters.** Mean ± SD over three seeds, in %; differences in percentage points. Each partition covers all 500 problems, with n denoting the number of problems per group. Checkpoints are identical to those in Table 20.

Group	n	LoRA	Loop Dropout	Difference
Algebra	124	58.33 ± 3.05	72.04 ± 3.05	+13.71
Counting & Probability	38	29.82 ± 9.24	39.47 ± 2.63	+9.65
Geometry	41	32.52 ± 6.14	38.21 ± 3.73	+5.69
Intermediate Algebra	97	19.59 ± 1.03	25.43 ± 6.21	+5.84
Number Theory	62	40.32 ± 5.59	44.09 ± 4.06	+3.76
Prealgebra	82	54.88 ± 4.40	62.60 ± 1.86	+7.72
Precalculus	56	14.88 ± 3.72	25.60 ± 2.73	+10.71
Level 1	43	75.97 ± 1.34	78.29 ± 3.55	+2.33
Level 2	90	60.74 ± 1.28	67.04 ± 2.31	+6.30
Level 3	105	45.40 ± 0.55	62.86 ± 1.90	+17.46
Level 4	128	34.11 ± 1.63	42.71 ± 5.32	+8.59
Level 5	134	11.69 ± 4.56	17.16 ± 2.69	+5.47

Answer availability. We also decompose the rank-16 MATH-500 accuracy difference according to whether the official scorer extracts an answer for both methods. Of the 8.67-point total gain, 7.87 points come from problems with two extractable answers and 0.80 from the remaining problems, using all 500 questions as the denominator for both components. The gain on pairs with two extractable answers is positive in each seed: 9.80, 8.40 and 5.40 points for seeds 101–103. Most of the improvement thus reflects correctness on questions where both methods satisfy the answer-extraction requirement.

J.2 Training and Evaluation Depth

Each saved adapter is evaluated at $K \in \{4, 6, 8\}$ without further fine-tuning, testing zero-shot generalization when evaluation extends beyond the training depth. Table 23 reports the absolute accuracies underlying the training-by-evaluation comparison in Figure 4. Figure 5 shows the evaluation-depth curves for adapters trained at four loops.

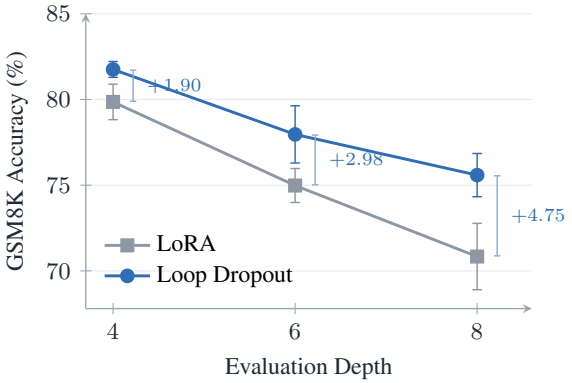


Figure 5: **Zero-shot generalization beyond the training depth.** Ouro-1.4B after direct GSM8K fine-tuning at four loops, evaluated without further training; accuracy is mean ± SD over three seeds. Annotations show the gain of Loop Dropout over LoRA in percentage points.

Table 23: **Accuracy across training and evaluation depths.** GSM8K recipe, Ouro-1.4B, LR 10^{-4} (A-factor rate for LoRA+), seeds 101–103; mean $\pm$ SD of strict exact match, %. Loop Dropout has higher mean accuracy than LoRA in all nine depth pairs. Comparisons within a training depth share the training budget; different training depths use different numbers of recurrent passes per optimizer step.

Method	Train K	Eval $K = 4$	Eval $K = 6$	Eval $K = 8$
LoRA	4	79.9 ± 1.0	75.0 ± 1.0	70.8 ± 1.9
LoRA	6	82.4 ± 1.1	82.3 ± 0.7	77.8 ± 0.7
LoRA	8	80.3 ± 1.4	81.8 ± 0.6	81.6 ± 0.9
Loop Dropout	4	81.8 ± 0.5	78.0 ± 1.7	75.6 ± 1.3
Loop Dropout	6	83.0 ± 0.3	83.3 ± 0.7	80.5 ± 0.8
Loop Dropout	8	83.4 ± 0.5	83.6 ± 0.5	82.1 ± 0.8
LoRA+	4	79.8 ± 0.5	72.8 ± 2.1	70.2 ± 2.2
rsLoRA	4	79.8 ± 0.9	75.0 ± 1.4	71.1 ± 2.4

J.3 Learning Rates and Shifted Mathematical Tests

Learning rate and training depth. Under direct GSM8K fine-tuning, Loop Dropout improves mean accuracy at every tested learning rate in Table 25. The matched-depth comparisons in Table 27 show higher mean accuracy for Loop Dropout across recurrence depths from two to eight loops. Tables 25 and 27 give the full curves.

Shifted mathematical test sets. Table 26 shows that saved GSM8K adapters retain positive margins on both the relabelled GSM8K-Platinum (Vendrow et al., 2025) and the perturbed GSM-Plus (Li et al., 2024) benchmark. The gains on GSM-Plus mini are 1.14 and 3.40 points at the two reported learning rates, extending the comparison to perturbed problem statements.

Table 24: **The gain persists across learning rates and model sizes under the GSM8K recipe.** Zero-shot strict exact match (%) after fine-tuning on the GSM8K training split, mean $\pm$ SD. The first and last blocks use three seeds; the middle block uses seeds 0–4. LR denotes the A-factor rate for LoRA+. Bold marks the highest mean within each model, seed set and learning rate.

Model	Method	LR	GSM8K
Ouro-1.4B	LoRA	1e-4	79.9 ± 1.0
Ouro-1.4B	LoRA+	1e-4	79.8 ± 0.5
Ouro-1.4B	rsLoRA	1e-4	79.8 ± 0.9
Ouro-1.4B	Loop Dropout	1e-4	$\mathbf{81.8 \pm 0.5}$
Ouro-1.4B	LoRA	1e-4	80.4 ± 0.6
Ouro-1.4B	Loop Dropout	1e-4	$\mathbf{81.9 \pm 0.4}$
Ouro-1.4B	LoRA	3e-4	78.6 ± 1.1
Ouro-1.4B	Loop Dropout	3e-4	$\mathbf{81.2 \pm 0.6}$
Ouro-2.6B	LoRA	1e-4	87.6 ± 0.8
Ouro-2.6B	Loop Dropout	1e-4	$\mathbf{89.0 \pm 0.3}$
Ouro-2.6B	LoRA	3e-4	85.9 ± 0.9
Ouro-2.6B	Loop Dropout	3e-4	$\mathbf{88.1 \pm 1.0}$

Table 25: **Learning-rate curve on the GSM8K recipe** (Ouro-1.4B, $K = 4$; lm-eval zero-shot strict exact match, %). Five seeds at 10^{-4} and 3×10^{-4}, three seeds elsewhere.

LR	LoRA	Loop Dropout	Δ
3e-5	80.4 ± 0.4	80.9 ± 0.3	+0.56
1e-4	80.4 ± 0.6	81.9 ± 0.4	+1.52
2e-4	79.8 ± 1.5	**82.2 ± 0.6**	+2.38
3e-4	78.6 ± 1.1	81.2 ± 0.6	+2.62
6e-4	72.4 ± 0.7	73.8 ± 0.8	+1.36

Table 26: **Transfer of the saved GSM8K-recipe adapters** (Ouro-1.4B, three seeds) to GSM8K-Platinum (1,209 relabelled problems) and GSM-Plus mini (2,400 perturbed problems); zero-shot strict exact match, %. "Input dropout" applies standard dropout of 0.1 to the LoRA input.

		GSM8K-Platinum		GSM-Plus mini	
Method	LR	Accuracy	Δ vs. LoRA	Accuracy	Δ vs. LoRA
LoRA	1e-4	82.4 ± 0.0	–	58.9 ± 0.7	–
Input dropout	1e-4	81.7 ± 0.4	−0.66	59.1 ± 0.4	+0.21
Loop Dropout	1e-4	**83.8 ± 0.4**	+1.38	**60.0 ± 0.2**	+1.14
LoRA	3e-4	80.3 ± 0.4	–	56.0 ± 1.0	–
Input dropout	3e-4	80.7 ± 1.7	+0.33	57.0 ± 1.1	+1.04
Loop Dropout	3e-4	**83.1 ± 0.7**	+2.73	**59.4 ± 0.4**	+3.40

Table 27: **Training and evaluating at the same recurrence depth** on the GSM8K recipe (Ouro-1.4B, LR 10^{-4}; strict exact match, %). Seeds 0–2 (five at $K = 4$) use independent initializations; seeds 101–103 share initialization across methods.

K	Seeds	LoRA	Loop Dropout	Δ
2	0–2	68.6 ± 0.8	70.7 ± 0.3	+2.17
4	0–4	80.4 ± 0.6	81.9 ± 0.4	+1.52
6	0–2	81.6 ± 0.6	83.2 ± 1.2	+1.59
4	101–103	79.9 ± 1.0	81.8 ± 0.5	+1.90
6	101–103	82.3 ± 0.7	83.3 ± 0.7	+1.06
8	101–103	81.6 ± 0.9	82.1 ± 0.8	+0.51

K Diagnosing and Strengthening Early-Loop Adaptation

K.1 Single-Loop Activation

For a trained shared adapter, we enable its update at one loop at a time while the backbone executes all four loops. The teacher-forced GSM8K diagnostics in this appendix each use 500 test problems. All active gates have unit strength, and the single-loop diagnostic scores the reference continuation at the final loop. The adapter was trained with the direct GSM8K recipe; Table 28 reports the two learning rates separately. We express the improvement as a percentage reduction in loss relative to the frozen model:

$$\rho_t = 100\,\frac{\mathrm{CE}(\mathbf{0}) - \mathrm{CE}(e_t)}{\mathrm{CE}(\mathbf{0})}. \tag{19}$$

Here e_t activates only loop t and $\mathrm{CE}(\mathbf{0}) = 0.8445$. All activation conditions use the same frozen-model loss as their reference, and higher values indicate stronger loss reduction.

Table 28: **Single-loop activation of a trained update.** Ouro-1.4B, direct GSM8K recipe. The four loss columns are final-loop teacher-forced cross entropy (lower is better). The last column separately reports standard all-on generation accuracy on all 1,319 test questions, in %.

		Final-loop loss				All-on
Method	LR	Only 1	Only 2	Only 3	Only 4	accuracy
LoRA	10^{-4}	0.7437	0.6834	0.6208	0.5709	80.59
Unscaled	10^{-4}	0.5309	0.5082	0.4984	0.5037	77.03
Loop Dropout	10^{-4}	0.5847	0.5475	0.5409	0.5432	81.73
LoRA	3×10^{-4}	0.7493	0.6835	0.6159	0.5317	77.86
Unscaled	3×10^{-4}	0.5152	0.4907	0.4824	0.4895	73.84
Loop Dropout	3×10^{-4}	0.5587	0.5302	0.5199	0.5198	82.11

At 10^{-4}, the single-loop loss reductions are 11.9/19.1/26.5/32.4% for LoRA and 30.8/35.2/35.9/35.7% for Loop Dropout. LoRA's early applications lag far behind its final application in loss reduction. Loop Dropout lowers the raw single-loop loss at every position relative to LoRA, with the largest improvement at the first loop. The second learning rate gives the same pattern: 11.3/19.1/27.1/37.0% for LoRA and 33.8/37.2/38.4/38.4% for Loop Dropout. Loop Dropout thus strengthens early applications and narrows the disparity between recurrent positions.

Masking and rescaling have distinct roles in this comparison, and their ordering follows the readout scale. All readouts use unit gates, the strength at which LoRA and the unscaled rule train each active application, whereas Loop Dropout trains each active application at strength $1/q = 2$; the single-loop readout therefore applies its update at half the training strength. At all-on inference, the four unit-gate applications equal the expected total update of Loop Dropout training, $\mathbb{E}[\sum_t g_t] = K = 4$, and twice that of unscaled training, $qK = 2$. Accordingly, unscaled masking gives the lowest single-loop losses at its training strength, while inverse-survival rescaling yields the stronger all-on generation result at both learning rates. The individual-loop diagnostic and the generation comparison therefore support learning reusable updates together with matching their training and inference scale.

K.2 Generation and Readouts after MetaMath Fine-Tuning

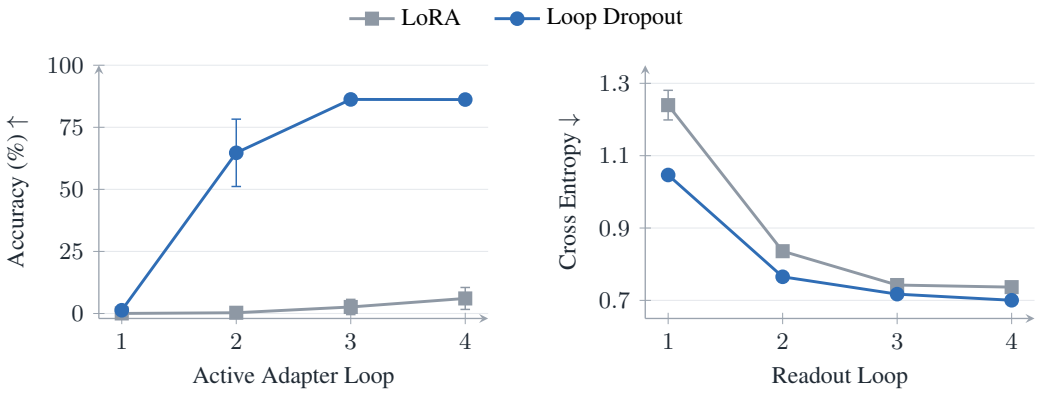
These diagnostics use the shared rank-16 MetaMath-GSM checkpoints, with LR 2×10^{-4} for LoRA and 5×10^{-5} for Loop Dropout, and training seeds 101–103. Table 4 activates one adapter application at unit strength and generates answers on the full GSM8K test set. The backbone executes all four loops for every activation position. All four positions are reported separately, with mean and sample standard deviation over the training seeds.

Table 29 keeps every adapter application enabled and scores the reference solutions at each loop's readout. The loss is token-weighted cross entropy; this diagnostic uses the full MATH-500 set. All readout positions use the same reference tokens.

The largest reduction occurs at the first readout, where cross entropy decreases from 1.240 to 1.047. Figure 6 plots the single-application generation results and the MATH-500 readout trajectory.

Table 29: **MATH-500 readouts along the fully adapted recurrence.** Ouro-1.4B after MetaMath-GSM fine-tuning. Token-weighted cross entropy, mean ± SD over three training seeds; lower is better. Every adapter application and all four backbone loops are enabled.

Readout	LoRA	Loop Dropout
Loop 1	1.240 ± 0.041	1.047 ± 0.010
Loop 2	0.836 ± 0.006	0.765 ± 0.005
Loop 3	0.742 ± 0.009	0.717 ± 0.002
Loop 4	0.737 ± 0.010	0.700 ± 0.005



(a) GSM8K Generation

(b) MATH-500 Readouts

Figure 6: **Generation and early readouts after MetaMath-GSM fine-tuning.** Ouro-1.4B, rank 16, four backbone loops. Left: GSM8K generation with only the indicated adapter application enabled. Right: MATH-500 teacher-forced cross entropy with all applications enabled, read out at each loop. Points and error bars show mean ± SD over three training seeds. Loop Dropout supports generation from individual applications at loops two through four and lowers MATH-500 loss at every readout, most strongly at the first loop.

L LIMITATIONS

Scope. Our experiments cover mathematical fine-tuning of Ouro at two model sizes and four adapter ranks, and instruction tuning of Ouro-1.4B at rank 16, under the learning rates and seeding protocols of Appendix D.2. Additional mathematical evaluations on LoopUS-Qwen3-4B and Huginn are reported in Appendix F.1.

Interpretation. The curvature analysis describes the objective locally; the finite-mask mixture gives its exact form. The controls compare masking rules as complete distributions, including their gate support and higher moments.