COMPUTATIONAL DEPTH AS AN ADAPTATION VARIABLE FOR PARAMETER EFFICIENT FINE TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Parameter efficient fine tuning adapts pretrained language models with a small number of trainable parameters. Most existing methods seek greater adaptation capacity by changing how these parameters modify a fixed pretrained computation graph. We study a complementary possibility in which the amount of pretrained computation itself becomes part of adaptation. We introduce recurrent PEFT by repeatedly executing a frozen pretrained middle stack while training lightweight adapters around the reused computation. Building on residual damped recurrence introduced in prior work, we consider both iteration specific and iteration shared adaptation. This construction increases effective computational depth without adding backbone parameters and enables direct comparison with conventional single pass PEFT under matched trainable parameter budgets. Across Qwen and Llama backbones from 0.6B to 8B parameters, recurrent adaptation remains broadly competitive on general reasoning and shows gains on GSM8K across all evaluated backbones. Minerva reveals a more pronounced backbone-dependent effect, with substantially larger gains on Qwen than on Llama models. We further find that these effects depend on the underlying PEFT parameterization and that the choice of recurrent execution remains consequential after fine-tuning.

1 INTRODUCTION

Parameter-efficient fine-tuning (PEFT) adapts large pretrained models while optimizing only a small fraction of their parameters. Existing methods have become increasingly sophisticated in how this limited adaptation capacity is parameterized, initialized, and distributed across the pretrained network. LoRA (Hu et al., 2022) represents weight updates through low-rank factors, while subsequent methods modify their decomposition, initialization, scaling, or allocation (Liu et al., 2024; Meng et al., 2024). Despite these advances, most PEFT methods retain the computational structure of the pretrained model. Each Transformer block is executed once, and adaptation determines how this fixed computation is modified. This leaves largely unexplored whether **computational depth itself can become an adaptation variable**.

The possibility is particularly compelling in light of recent progress on looped and recurrent-depth Transformers. Reusing the same computation across multiple logical depths decouples computational depth from parameter depth (Dehghani et al., 2019; Giannou et al., 2023; Yang et al., 2024). This recurrent structure provides a natural mechanism for iterative computation. Looped Transformers have been shown to learn iterative algorithms with substantially fewer distinct parameters (Yang et al., 2024), to implement multi-step computational procedures (Giannou et al., 2023; Gatmiry et al., 2024), and to benefit from additional recurrent depth on reasoning problems (Saunshi et al., 2025; Geiping et al., 2025). These findings suggest that computational depth can provide useful capacity independently of the number of distinct model parameters. They also motivate a different perspective on PEFT in which adaptation may control not only the trainable modification applied to a pretrained model, but also the amount of pretrained computation available to that modification.

While most looped models incorporate recurrent depth during pretraining, recent work has shown that this restriction is not strictly necessary. Specifically, Training-Free Looped Transformers (TFLT) (Chen et al., 2026) demonstrates that a conventionally pretrained dense Transformer can be converted into a recurrent model by repeatedly executing a middle stack, using residual damping to stabilize execution without additional training. This shows that computation learned under a

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.

single-pass dense structure can remain usable under recurrent execution. However, whether these properties persist when the transformed model is subsequently adapted, and whether parameter-efficient optimization can exploit the additional recurrent computation, remain largely unexplored.

We study this by treating computational depth as an adaptation variable. Under a fixed trainable-parameter budget, we compare conventional PEFT that spreads adaptation across the original single-pass network with recurrent PEFT that organizes adaptation around repeated execution of a frozen pretrained middle stack. The recurrent construction increases computational depth without duplicating the pretrained weights. We instantiate recurrent PEFT through Recurrent Layer Reuse (RLR), with iteration-specific adapters (RLR-I) and iteration-shared adapters (RLR-S), while adopting the residual-damped recurrence established by prior work (Chen et al., 2026). Across model scales and multiple PEFT parameterizations, we study whether recurrent computation remains useful after adaptation, how adaptation interacts with recurrent execution, and whether recurrent adaptation is particularly beneficial for multi-step reasoning.

Our contributions can be summarized as follows:

- We introduce **computational depth as an adaptation variable** for PEFT, decoupling trainable adaptation capacity from the amount of pretrained computation executed.
- We study **recurrent PEFT** under matched trainable-parameter budgets using both iteration-specific and shared adapters over reused pretrained layers.
- Across model scales and PEFT methods, recurrent adaptation remains competitive on general reasoning, improves GSM8K across backbones, and exhibits pronounced backbone and PEFT dependent effects on MATH.

2 Background

2.1 Parameter-Efficient Fine-Tuning

Parameter-efficient fine-tuning (PEFT) adapts a pretrained model while optimizing only a small subset of parameters. A representative approach is Low-Rank Adaptation (LoRA) (Hu et al., 2022), which freezes a pretrained linear transformation $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ and parameterizes its update as

$$Wx \mapsto Wx + \frac{\alpha}{r} BAx, \tag{1}$$

where $A \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B \in \mathbb{R}^{d_{\text{out}} \times r}$ are trainable. Most PEFT methods modify how such adaptation capacity is parameterized, initialized, or allocated while retaining the pretrained execution topology: each Transformer block is evaluated once. We study an additional design choice—how often selected pretrained computation is executed during adaptation.

2.2 Recurrent Layer Reuse

A standard Transformer applies a sequence of independently parameterized blocks,

$$h_{\ell+1} = T_\ell(h_\ell), \qquad \ell = 1, \ldots, L. \tag{2}$$

Consequently, parameter depth and computational depth typically grow together. Recurrent and parameter-shared Transformers decouple them by executing the same parameterized transformation at multiple logical depths (Dehghani et al., 2019; Yang et al., 2024; Geiping et al., 2025).

For a pretrained dense Transformer, however, repeated execution changes the topology under which the model was originally trained. TFLT shows that pretrained layers can be reused recurrently, raising the question of whether this additional depth can be exploited during adaptation.

2.3 Residual-Damped Recurrent Execution

Let M denote a pretrained stack that is to be evaluated K times. TFLT (Chen et al., 2026) showed that naively repeating pretrained computation can substantially perturb the original model and proposed residual-damped execution. Writing the stack-level residual as $G_M(h) = M(h) - h$, the

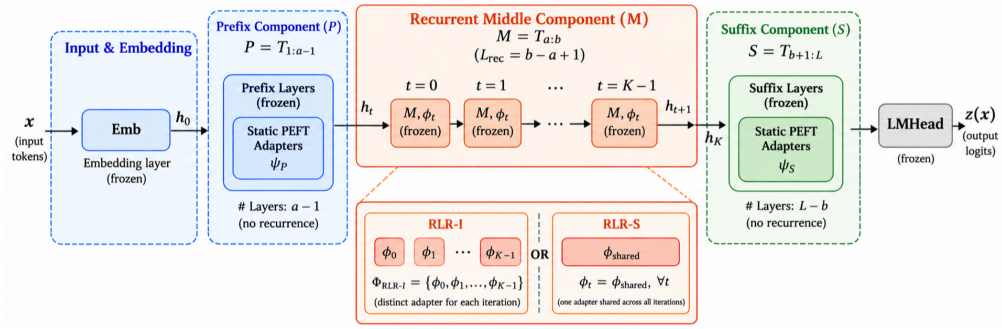

Figure 1: Overview of recurrent parameter-efficient adaptation. The pretrained Transformer is partitioned into a single-pass prefix P, a recurrent middle stack M, and a single-pass suffix S. All backbone parameters remain frozen. RLR-I uses iteration-specific adapters for repeated evaluations of M, whereas RLR-S shares the same adapters across recurrent iterations.

recurrent update is $h_{t+1} = h_t + \gamma G_M(h_t)$, where $\gamma = \frac{1}{K}$. This admits a forward-Euler interpretation in which increasing K refines the discretization without increasing the nominal update horizon. We adopt this stabilized recurrence formulation and study how recurrent reuse interacts with PEFT under a fixed trainable-parameter budget.

3 Method

3.1 Computational Depth as an Adaptation Variable

We study parameter-efficient adaptation when the execution topology of the pretrained model is allowed to vary. Conventional PEFT applies trainable updates to a fixed single-pass backbone. We instead retain all pretrained weights while allowing selected pretrained computation to be executed at additional logical depths. This separates two quantities that are typically coupled in PEFT, namely the number of trainable adaptation parameters and the amount of pretrained computation executed.

Let a pretrained decoder-only Transformer consist of L blocks $\{T_\ell\}_{\ell=1}^{L}$. For $T_{i:j} = T_j \circ \cdots \circ T_i$, we partition the network into three consecutive components

$$P = T_{1:a-1}, \qquad M = T_{a:b}, \qquad S = T_{b+1:L}, \qquad F = S \circ M \circ P. \tag{3}$$

Here, P and S denote the prefix and suffix, while M denotes the pretrained middle stack whose computation may be reused. The width of the middle block is $L_{\text{rec}} = b - a + 1$, with layer index set $\mathcal{I}_M = \{a, \ldots, b\}$. All pretrained parameters remain frozen. Trainable PEFT parameters are attached to selected linear transformations. For concreteness, a rank-r LoRA update to a frozen projection $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ is

$$Wx \mapsto Wx + \frac{\alpha}{r} BAx, \tag{4}$$

where $A \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B \in \mathbb{R}^{d_{\text{out}} \times r}$ are trainable. The construction is not specific to LoRA and can be instantiated with other PEFT parameterizations while leaving the pretrained mapping frozen.

3.2 Residual-Damped Recurrent Execution

To vary computational depth without introducing additional pretrained parameters, we repeatedly execute the middle stack M. Directly repeating a stack pretrained for single-pass execution can substantially perturb the original model. We therefore adopt the residual-damped recurrence introduced in TFLT (Chen et al., 2026).

Let $M_{\theta,\phi}(h)$ denote an adapted evaluation of the middle stack, where θ contains its frozen pretrained parameters and ϕ denotes trainable PEFT parameters. We define the stack-level residual

$$G_{M,\theta,\phi}(h) = M_{\theta,\phi}(h) - h. \tag{5}$$

Given $h_0 = P_{\theta,\psi_P}(\mathrm{Emb}(x))$, where ψ represents PEFT parameters for P and S, we execute the recurrent stack K times according to

$$\widetilde{h}_{t+1} = M_{\theta,\phi_t}(h_t), \tag{6}$$

$$h_{t+1} = h_t + \gamma\left(\widetilde{h}_{t+1} - h_t\right), \qquad t = 0, \ldots, K-1, \tag{7}$$

with $\gamma = \frac{1}{K}$. The resulting model output is $z(x) = \mathrm{LMHead}\left(S_{\theta,\psi_S}(h_K)\right)$.

Increasing K therefore increases the number of evaluations of pretrained computation while controlling the magnitude of each stack-level residual update. Our focus is not the damping rule itself, but how this recurrent computation behaves when its adaptation parameters are optimized through PEFT.

3.3 Iteration-Specific Recurrent Adaptation

In Recurrent Layer Reuse with Iteration-specific adapters (RLR-I), each recurrent evaluation has its own adapter parameters, while the pretrained parameters of the middle stack remain shared,

$$\Phi_{\mathrm{RLR-I}} = \{\phi_0, \ldots, \phi_{K-1}\}. \tag{8}$$

The prefix and suffix are executed once and use static adapters ψ_P and ψ_S.

Because each recurrent evaluation introduces a separate adapter set, we reduce the adapter rank accordingly so that the total number of trainable parameters matches the conventional PEFT baseline.

3.4 Shared Recurrent Adaptation

In Recurrent Layer Reuse with Shared adapters (RLR-S), a single adapter is shared across all recurrent evaluations,

$$\phi_t = \phi_{\mathrm{shared}}, \qquad t = 0, \ldots, K-1. \tag{9}$$

Because the same adapter is reused across recurrent evaluations, recurrence does not introduce additional trainable parameters. RLR-S therefore uses the same adapter rank as the conventional PEFT baseline.

3.5 Parameter-Matched Comparison

Our comparisons match trainable-parameter budgets rather than computation, since recurrent adaptation intentionally trades additional frozen-backbone computation for increased effective depth.

Parameter matching therefore holds adaptation capacity fixed while changing the execution topology. The recurrent configurations have effective computational depth

$$L_{\mathrm{eff}} = L + (K-1)L_{\mathrm{rec}}, \tag{10}$$

while the spread baseline retains $L_{\mathrm{eff}} = L$. This controlled construction is the basis for studying computational depth as an adaptation variable throughout our experiments.

4 Related Work

Parameter-efficient adaptation. PEFT methods adapt pretrained models while optimizing only a small parameter subset. LoRA (Hu et al., 2022) uses low-rank weight updates, while subsequent methods modify their decomposition, initialization, or allocation, including DoRA (Liu et al., 2024), PiSSA (Meng et al., 2024), and MiSS (Kang & Yin, 2026). These approaches primarily change how trainable capacity modifies a fixed pretrained execution graph. We instead study an additional design axis: allocating a fixed trainable-parameter budget when selected pretrained computation is executed recurrently.

Recurrent-depth Transformers. Parameter sharing across computational depth has been explored from the Universal Transformer (Dehghani et al., 2019) to recent looped and recurrent-depth Transformers (Yang et al., 2024; Geiping et al., 2025; Saunshi et al., 2025). These works demonstrate that computational depth need not require distinct parameters at every logical layer. Our setting instead begins with a conventionally pretrained dense Transformer and introduces recurrent reuse during PEFT, separating reuse of frozen pretrained computation from sharing of trainable adaptation parameters.

Retrofitting recurrence into pretrained Transformers. Relaxed Recursive Transformers (Chen et al., 2026) similarly combine recursive execution with depth-specific LoRA, but focus on compressing pretrained models through parameter sharing, whereas we keep the pretrained backbone frozen and study recurrent computation as an adaptation variable under matched PEFT budgets. Most closely, Training-Free Looped Transformers (Chen et al., 2026) reuses a contiguous middle stack of a pretrained Transformer and introduces residual-damped looping to stabilize repeated execution. We adopt this recurrence formulation to study recurrent adaptation under PEFT. Concurrent work also studies retrofitting recurrent depth through downstream training (Shapiro, 2026).

5 Experiments

5.1 Experimental Setup

Training setup. We evaluate base, non-instruction-tuned backbones across multiple model scales. Our primary experiments use Qwen3-0.6B (Yang et al., 2025) and Llama-3.2-1B (Llama Team, AI @ Meta, 2024), with additional scale-up evaluations on Qwen3-4B and Llama-3.1-8B. Models are instruction-tuned in two domains. For mathematical reasoning, we use MetaMathQA (Yu et al., 2024) with 395K examples. For general reasoning, we use Commonsense170K (Hu et al., 2023) with 170,420 examples. We use the complete training sets without subsampling. All models are trained for 3 epochs with AdamW, an initial learning rate of 2×10^{-5}, cosine learning-rate decay, and a warmup ratio of 0.03. The sequence length is 2,048 tokens and the effective batch size is 64 sequences. Additional training and implementation details are provided in Appendix B.

Evaluation. We evaluate the resulting models using `lm-evaluation-harness` (Biderman et al., 2024) after converting recurrent models to their equivalent unrolled form. Mathematical reasoning is evaluated with greedy chain-of-thought generation on GSM8K (Cobbe et al., 2021) using 5-shot prompting and Minerva (Lewkowycz et al., 2022) using 4-shot. General-domain adapters are evaluated on PIQA, ARC-Easy, ARC-Challenge, HellaSwag, Winogrande, CommonsenseQA, and OpenBookQA using 0-shot evaluation. Complete evaluation settings are provided in Appendix B.

Our experiments examine three aspects of recurrent adaptation. We first test whether increasing computational depth is useful under matched trainable-parameter budgets across model scales. We then evaluate how recurrent adaptation interacts with different PEFT parameterizations. Finally, we examine how the choice of recurrent execution affects the model both before and after fine-tuning.

5.2 Recurrent Adaptation under Matched Parameter Budgets

We first evaluate whether computational depth can serve as a useful adaptation variable when trainable capacity is held approximately fixed. Table 1 compares conventional LoRA with RLR-I and RLR-S under matched trainable-parameter budgets across different model families and scales.

On commonsense reasoning, recurrent adaptation remains broadly competitive with conventional LoRA. It improves the average score on Llama-3.2-1B and Qwen3-4B, remains comparable on Qwen3-0.6B, and shows a small degradation on Llama-3.1-8B. These results do not indicate a uniform advantage from recurrent execution. Rather, they show that allocating the same adaptation budget around additional computational depth can preserve competitive general-domain performance across model scales.

The pattern is more favorable on mathematical reasoning. On GSM8K, at least one recurrent configuration improves over conventional LoRA for every evaluated backbone, although the magnitude of the gain varies across models. Minerva reveals a sharper backbone-dependent pattern. On both

Table 1: Evaluation across model scales under matched trainable-parameter budgets. We compare conventional LoRA with RLR-I and RLR-S on seven commonsense benchmarks, GSM8K, and Minerva. GSM8K uses 5-shot chain-of-thought prompting and Minerva uses 4-shot prompting. Minerva is reported using exact match (EM) and MathVerify (MV). GSM8K is evaluated with both strict and flexible match, and we report their maximum as Best. Within each backbone, the best result is shown in **bold** and the second-best is underlined.

Method	Config	Commonsense Reasoning								GSM8K	Minerva	
		PIQA	ARC-E	ARC-C	HS	WG	CSQA	OBQA	Avg	Best	em	mv
Qwen3-0.6B	Dense	63.11	34.68	30.03	41.94	51.62	19.33	33.20	39.13	62.24	19.64	22.04
	RLR-I	63.28	35.48	28.75	42.07	51.78	21.21	32.00	**39.22**	**64.82**	**28.48**	**30.68**
	RLR-S	63.38	35.94	28.58	41.83	50.67	21.46	31.60	39.07	63.00	25.06	27.68
Llama-3.2-1B	Dense	69.70	35.52	30.80	56.19	54.54	19.25	35.60	43.09	14.48	3.72	**5.72**
	RLR-I	72.09	38.05	31.06	57.96	54.93	19.33	35.60	44.15	15.77	**4.18**	5.50
	RLR-S	70.24	37.21	31.57	58.61	54.38	23.18	35.40	**44.37**	**17.97**	3.62	5.54
Qwen3-4B	Dense	72.69	42.34	38.65	63.47	63.22	19.57	34.80	47.82	84.76	0.28	18.30
	RLR-I	72.96	45.24	39.08	62.87	64.96	19.66	35.00	48.54	**85.44**	0.92	**47.60**
	RLR-S	73.23	44.65	39.76	63.32	64.80	19.82	34.80	**48.63**	85.29	**1.68**	47.44
Llama-3.1-8B	Dense	77.64	45.33	40.70	72.97	65.27	20.72	36.00	**51.23**	71.19	18.36	**27.94**
	RLR-I	76.82	44.49	39.93	73.19	64.72	21.70	35.80	50.95	**73.46**	17.60	27.60
	RLR-S	76.82	44.57	40.53	72.60	64.09	20.97	36.80	50.91	71.19	**18.42**	27.70

Qwen models, recurrent adaptation substantially improves performance. The improvement is visible in both evaluation metrics for Qwen3-0.6B, while on Qwen3-4B the largest difference appears under MathVerify, where both recurrent variants substantially outperform the single-pass baseline. In contrast, the Llama backbones show little change on Minerva under recurrent adaptation. Together, these results suggest that the benefit of additional recurrent computation is not uniform across model families or tasks, but can be substantial for particular pretrained backbones.

We further examine the tradeoff between adapter capacity and recurrent depth in Appendix B.5. Under the same trainable-parameter budget, reallocating capacity toward greater recurrent depth substantially improves GSM8K while leaving the commonsense aggregate nearly unchanged. Since recurrent depth and adapter rank vary jointly in this comparison, we interpret this result as supporting evidence rather than an isolated causal effect of recurrent depth.

Neither RLR-I nor RLR-S consistently dominates across backbones. Their relative performance varies across both model families and evaluation domains. This suggests that the primary effect is associated with the recurrent organization of adaptation rather than a universal preference for shared or iteration-specific adapter parameters.

5.3 Interaction with PEFT Parameterization

In this section we examine whether recurrent adaptation is specific to the low-rank parameterization of standard LoRA. We apply the same recurrent construction to DoRA, PiSSA, and MiSS and compare each method with its single-pass counterpart under matched trainable-parameter budgets. Table 2 reports results on Llama-3.2-1B and Qwen3-0.6B.

Recurrent adaptation remains competitive across several PEFT families, but its effect depends strongly on both the underlying parameterization and the pretrained backbone. On Llama-3.2-1B, recurrent variants generally preserve or improve commonsense performance, while the mathematical results are more mixed. LoRA and DoRA show favorable GSM8K behavior, but their Minerva scores remain close to the single-pass baselines. PiSSA does not benefit consistently on either mathematical benchmark, while MiSS improves GSM8K but shows little change on Minerva.

A much stronger separation appears on Qwen3-0.6B. For LoRA and DoRA, recurrent adaptation substantially improves Minerva performance, with DoRA showing particularly large gains under both Minerva metrics. In contrast, PiSSA deteriorates markedly on Minerva despite remaining competitive on commonsense reasoning, while MiSS remains comparatively stable with modest improvements under MathVerify. These results indicate that recurrent adaptation is not uniformly beneficial across PEFT methods. Instead, its effect is jointly determined by the adapter parameter-

Table 2: Interaction between recurrent adaptation and PEFT parameterization. Each PEFT method is compared with its RLR-I and RLR-S counterparts under matched trainable-parameter budgets on Llama-3.2-1B and Qwen3-0.6B. GSM8K is evaluated with strict and flexible match, with Best denoting their maximum. Minerva is reported with exact match (EM) and MathVerify (MV). Within each PEFT method, the best result is shown in **bold** and the second-best is underlined.

Method	Config	Commonsense Reasoning								GSM8K	Minerva	
		PIQA	ARC-E	ARC-C	HS	WG	CSQA	OBQA	Avg	Best	em	mv
Llama-3.2-1B												
LoRA	Dense	69.70	35.52	30.80	56.19	54.54	19.25	35.60	43.09	14.48	3.72	**5.72**
	RLR-I	72.09	38.05	31.06	57.96	54.93	19.33	35.60	44.15	15.77	**4.18**	5.50
	RLR-S	70.24	37.21	31.57	58.61	54.38	23.18	35.40	**44.37**	**17.97**	3.62	5.54
DoRA	Dense	69.15	35.40	31.06	56.29	55.01	19.00	35.80	43.10	14.33	3.68	5.30
	RLR-I	71.71	37.50	31.57	58.02	55.49	18.84	35.00	**44.02**	14.25	**3.92**	**5.40**
	RLR-S	69.97	37.12	30.80	58.86	53.28	23.10	35.00	**44.02**	**18.04**	3.30	5.36
PiSSA	Dense	71.06	36.57	32.59	56.28	54.22	19.00	37.00	43.82	**24.03**	4.14	**6.72**
	RLR-I	70.57	37.37	32.25	57.29	54.78	17.94	36.20	43.77	18.73	**4.84**	6.28
	RLR-S	70.24	36.95	33.45	57.44	56.04	20.23	36.80	**44.45**	18.42	4.34	6.30
MiSS	Dense	67.85	35.69	29.35	50.00	50.43	19.33	36.00	41.24	20.70	**4.18**	6.22
	RLR-I	69.59	36.32	32.59	53.07	52.25	20.80	36.20	**42.97**	**22.67**	3.36	6.32
	RLR-S	70.73	37.67	30.12	54.18	51.46	19.25	34.60	42.57	22.06	3.38	**6.48**
Qwen3-0.6B												
LoRA	Dense	63.11	34.68	30.03	41.94	51.62	19.33	33.20	39.13	62.24	19.64	22.04
	RLR-I	63.28	35.48	28.75	42.07	51.78	21.21	32.00	**39.22**	**64.82**	**28.48**	**30.68**
	RLR-S	63.38	35.94	28.58	41.83	50.67	21.46	31.60	39.07	63.00	25.06	27.68
DoRA	Dense	63.33	34.22	29.69	41.98	51.62	20.64	33.00	39.21	61.26	13.78	16.18
	RLR-I	63.00	35.86	29.44	41.88	52.72	21.21	32.80	**39.56**	**64.82**	**28.74**	**30.98**
	RLR-S	62.84	35.77	29.61	41.84	50.67	21.38	31.60	39.10	63.23	25.64	28.32
PiSSA	Dense	63.06	36.53	29.35	45.09	54.14	19.66	33.40	40.18	**65.43**	**26.48**	**30.30**
	RLR-I	64.36	35.86	30.46	46.11	53.35	22.28	32.40	**40.69**	64.52	18.22	17.84
	RLR-S	64.69	36.15	30.29	46.21	51.85	23.10	32.00	40.61	65.35	17.42	17.86
MiSS	Dense	64.96	37.33	30.97	47.01	55.09	20.64	34.80	**41.54**	63.91	21.80	25.28
	RLR-I	65.61	36.28	30.63	45.37	51.62	19.82	34.20	40.50	63.91	21.60	26.24
	RLR-S	64.53	35.52	30.46	43.25	50.99	19.74	34.40	39.84	**64.75**	**22.48**	**27.32**

ization and the pretrained backbone, with Minerva revealing a particularly pronounced interaction on Qwen.

5.4 Understanding Recurrent Adaptation

The results above show that recurrent adaptation can remain competitive with conventional PEFT, while its effect on mathematical reasoning depends strongly on both the pretrained backbone and the underlying PEFT parameterization. We next examine the recurrent mechanism itself. We first study the model before fine-tuning to establish the behavior inherited from training-free looping, and then test whether the recurrence formulation continues to matter after parameter-efficient optimization. Finally, using RLR-I, we examine whether iteration-specific adapters applied to the same pretrained layers converge toward similar updates during fine-tuning.

5.4.1 Initialization under Recurrent Execution

TFLT (Chen et al., 2026) showed that repeatedly executing pretrained Transformer layers can substantially perturb the original model, and introduced residual-damped recurrence to control this effect without additional training. We examine the same starting condition under the architecture and recurrence horizons used for our PEFT experiments.

With all adapters zero-initialized, we compare residual-damped recurrence with $\gamma = 1/K$ against naive repeated execution with $\gamma = 1$. Table 3 reports held-out loss and bidirectional KL divergence relative to the original pretrained model.

Across both backbones, naive recurrence increasingly diverges from the pretrained model as the number of recurrent evaluations increases. Residual damping substantially limits this divergence across the evaluated recurrence horizons and avoids the rapid increase in held-out loss observed with naive repeated execution. These results reproduce the stabilization behavior of residual-damped looping in our setting and establish the starting condition used for recurrent adaptation.

Table 3: Effect of residual damping before fine-tuning. With all adapters initialized to zero, we compare residual-damped recurrence with $\gamma = 1/K$ against naive repeated execution with $\gamma = 1$ across different recurrence horizons. We report held-out loss and forward and reverse KL divergence relative to the original pretrained model.

Backbone	Schedule	K	γ	math loss	general loss	KL fwd	KL rev
qwen3-0.6b	stock base	1	–	3.808	5.900	–	–
	damped	2	0.5	3.874	6.024	0.026	0.030
	naive	2	1.0	4.483	6.604	0.133	0.164
	damped	4	0.25	3.951	6.128	0.046	0.057
	naive	4	1.0	5.360	8.477	0.820	1.290
	damped	8	0.125	3.997	6.188	0.058	0.075
	naive	8	1.0	6.953	9.965	3.840	5.675
	damped	16	0.0625	4.029	6.228	0.064	0.083
	naive	16	1.0	12.732	14.370	8.757	10.906
llama3.2-1b	stock base	1	–	6.703	10.264	–	–
	damped	2	0.5	6.158	9.069	0.056	0.059
	naive	2	1.0	8.830	12.872	0.154	0.166
	damped	4	0.25	5.637	8.257	0.094	0.103
	naive	4	1.0	9.682	14.435	0.736	0.913
	damped	8	0.125	5.413	7.924	0.115	0.127
	naive	8	1.0	10.379	16.517	2.349	2.769
	damped	16	0.0625	5.327	7.793	0.127	0.141
	naive	16	1.0	11.079	17.476	5.626	6.478

Table 4: Effect of recurrence formulation after parameter-efficient fine-tuning on Llama-3.2-1B. We compare additive recurrence, naive repeated execution, and residual-damped recurrence using shared and iteration-specific adapters under matched trainable-parameter budgets.

Mechanism	**Config**	**Commonsense Reasoning**								**GSM8K**
		PIQA	**ARC-E**	**ARC-C**	**HS**	**WG**	**CSQA**	**OBQA**	**Avg**	**Best**
Llama-3.2-1B										
Baseline	-	69.70	35.52	30.80	56.19	54.54	19.25	35.60	43.09	14.48
Additive	Per-Iter	68.88	38.47	28.41	49.43	58.17	19.66	33.20	42.32	13.27
	Shared	69.04	38.13	29.95	50.10	55.88	19.66	33.40	42.31	12.81
Naive	Per-Iter	69.97	38.43	31.83	53.62	56.04	20.80	34.60	<u>43.61</u>	14.48
	Shared	69.64	37.25	31.23	54.11	56.83	20.23	35.00	43.47	14.18
RLR	Per-Iter	72.09	38.05	31.06	57.96	54.93	19.33	35.60	<u>44.15</u>	<u>15.77</u>
	Shared	70.24	37.21	31.57	58.61	54.38	23.18	35.40	**44.37**	**17.97**

5.4.2 Recurrence after Parameter-Efficient Adaptation

Initialization stability alone does not establish that the recurrence formulation remains important after optimization, since PEFT could in principle compensate for differences in recurrent execution. We therefore compare three recurrence formulations after fine-tuning under the same training setup. These include an additive update that accumulates the full middle-stack output, naive repeated execution with $\gamma = 1$, and the residual-damped formulation with $\gamma = 1/K$. Each formulation is evaluated with both shared and iteration-specific adapters under matched trainable-parameter budgets.

Table 4 shows that the recurrence formulation continues to affect downstream performance after fine-tuning. The additive formulation under-performs the single-pass baseline under both adapter parameterizations. Naive repeated execution remains closer to the baseline but provides limited improvement. In contrast, residual-damped recurrence improves both the average commonsense score and GSM8K performance under both shared and iteration-specific adaptation.

Together with the initialization analysis, these results distinguish the role of residual damping before and after adaptation. Prior work establishes its utility for introducing recurrence into a pretrained dense model without training. Our results show that the choice of recurrent execution remains consequential after parameter-efficient optimization. We therefore use the residual-damped formulation throughout our main recurrent experiments.

Table 5: Similarity of iteration-specific adapters in RLR-I. We report the median cosine similarity between adapter parameters and the principal-subspace overlap of the input and output low-rank factors across adapters assigned to different recurrent iterations. Low alignment indicates that independently parameterized adapters do not collapse toward strongly similar low-rank updates during fine-tuning.

Configuration	n	**Median** $\cos(A_i, A_j)$	**Input Overlap** (A)	**Output Overlap** (B)
RLR-I ($r = 8$)	24	-0.0002	0.0056	0.090
RLR-I ($r = 4$)	9	-0.0006	0.0064	0.133
RLR-I ($r = 16$)	6	$+0.0007$	0.0122	0.034
RLR-I ($r = 2$)	3	$+0.0006$	0.0051	0.212
RLR-I ($r = 32$)	2	$+0.0010$	0.0115	0.139

5.4.3 Adapter Similarity Across Recurrent Iterations

RLR-I assigns an independent adapter to each recurrent evaluation of the same pretrained middle stack. This additional parameterization is only meaningful if the resulting adapters do not simply converge toward identical updates. We therefore examine the similarity of adapters learned at different recurrent iterations.

For each adapted projection, we measure the cosine similarity between flattened adapter factors and the principal-subspace overlap of the down-projection (A) and up-projection (B) factors across recurrent iterations. Table 5 reports the median similarities across the analyzed adapter checkpoints and rank configurations.

Iteration-specific adapters remain weakly aligned. Across the evaluated ranks, adapters from different recurrent iterations show near-zero parameter cosine similarity and low overlap between their input subspaces. Output subspaces exhibit somewhat greater overlap, but remain far from complete alignment. Thus, independently parameterized recurrent adapters do not collapse toward strongly aligned low-rank subspaces during fine-tuning.

Importantly, low subspace overlap alone does not imply that individual iterations perform semantically distinct computations, particularly because independently parameterized low-rank subspaces can exhibit low overlap even without specialization. We therefore interpret this analysis only as evidence against strong parameter redundancy across recurrent iterations, rather than as evidence that each iteration learns a distinct reasoning operation.

6 Limitations

Our study focuses on a controlled recurrent adaptation setting with a contiguous middle stack and a limited set of recurrence depths and window sizes. A broader search over where and how much pretrained computation to reuse may further improve performance. In addition, the pronounced backbone-dependent behavior observed on mathematical reasoning suggests that pretrained model characteristics influence the effectiveness of recurrent adaptation, which we leave for future investigation.

7 Conclusion

We studied parameter efficient fine tuning by treating computational depth as an adaptation variable. Rather than keeping the pretrained execution topology fixed, we reuse frozen pretrained layers recurrently while approximately matching the trainable parameter budget of conventional PEFT. Across Qwen and Llama backbones, recurrent adaptation remains broadly competitive on general reasoning while showing a more favorable pattern on mathematical reasoning. GSM8K improves across all evaluated backbones, while Minerva reveals a pronounced backbone dependent effect with substantially larger gains on Qwen than on Llama models. These results show that pretrained computation can provide an additional axis for parameter efficient adaptation, although its benefit depends on the backbone, PEFT parameterization, and recurrence formulation.

AI USE STATEMENT

Generative AI tools were used to assist with writing and language polishing, literature retrieval and discovery, research ideation, and drafting portions of the manuscript. All AI-assisted content was reviewed and revised by the authors, and literature suggestions and research claims were independently verified. The authors take responsibility for the final content of this work.

REFERENCES

Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Jeffrey Hsu, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, Charles Lovering, Niklas Muennighoff, Ellie Pavlick, Jason Phang, Aviya Skowron, Samson Tan, Xiangru Tang, Kevin A. Wang, Genta Indra Winata, François Yvon, and Andy Zou. Lessons from the trenches on reproducible evaluation of language models, 2024. URL https://arxiv.org/abs/2405.14782.

Lizhang Chen, Jonathan Li, Chen Liang, Ni Lao, and Qiang Liu. Training-free looped transformers, 2026. URL https://arxiv.org/abs/2605.23872.

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, 2021. URL https://arxiv.org/abs/2110.14168.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Khashayar Gatmiry, Nikunj Saunshi, Sashank J. Reddi, Stefanie Jegelka, and Sanjiv Kumar. Can looped transformers learn to implement multi-step gradient descent for in-context learning? In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 15130–15152. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/gatmiry24b.html.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, volume 38, pp. 46242–46293, 2025. doi: 10.52202/085713-1380. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/3b01972cf31e6fa0fe29e4b8b5c2a0a1-Abstract-Conference.html.

Angeliki Giannou, Shashank Rajput, Jy-Yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 11398–11442. PMLR, 2023. URL https://proceedings.mlr.press/v202/giannou23a.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 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*, pp. 5254–5276. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.319. URL https://aclanthology.org/2023.emnlp-main.319/.

Jiale Kang and Qingyu Yin. MiSS: Revisiting the trade-off in LoRA with an efficient shard-sharing structure. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=gohmWoUSoS.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.

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 *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32100–32121. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24bn.html.

Llama Team, AI @ Meta. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

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*, volume 37, pp. 121038–121072, 2024. doi: 10.52202/079017-3846. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/db36f4d603cc9e3a2a5e10b93e6428f2-Abstract-Conference.html.

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://openreview.net/forum?id=din0lGfZFd.

Mark Shapiro. Retrofitting recurrent depth into a pretrained language model: Installation, extrapolation, transfer, and retention at two parameter budgets, 2026. URL https://arxiv.org/abs/2608.11233.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Liu Yang, Kangwook Lee, Robert D. Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=HHbRxoDTxE.

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://proceedings.iclr.cc/paper_files/paper/2024/hash/c400474e8a36d0812fdee52739288b12-Abstract-Conference.html.

A Compute Accounting (Qwen3-0.6B Math Rows)

Table 6: Compute and efficiency accounting across architectural variants on Qwen3-0.6B (budget-matched at 10,092,544 trainable parameters).

K	Depth	TFLOPs/step	Eff. Batch	Wall (h)
1	28	238	32	9.1
2	42	358	32	8.2
2	32	272	32	7.8
4	40	340	32	9.6

Note: Recurrent execution increases compute according to the amount of reused computation. The wide-window $K{=}2$ configuration incurs approximately $1.5\times$ the FLOPs of the baseline (42 vs. 28 layer passes), while window-confined looping limits the effective depth to 32–40 layer passes.

B Implementation Details

This appendix lists the complete configuration needed to reproduce the main comparison: software environment, training hyperparameters, data pipeline, per-arm adapter geometry, and evaluation settings. All values are the defaults of the released training entry point; deviations are stated inline.

B.1 Software and hardware

Table 7: Environment.

Python	3.10.13
PyTorch	2.8.0 (CUDA 12.8)
Transformers	4.57.5 (looped training forward is version-pinned to it)
PEFT	0.20.0 (baseline arm only)
lm-evaluation-harness	0.4.12
DeepSpeed	0.18.9 (ZeRO stage 2; used for $\geq$4B-parameter arms)
Hardware	single node, NVIDIA H100 80 GB GPUs; bf16 throughout

B.2 Training hyperparameters

Training hyperparameters are listed in Table 8.

Table 8: Instruction-tuning configuration (identical across arms unless noted).

Optimizer	AdamW ($\beta_1{=}0.9$, $\beta_2{=}0.999$), bias/LayerNorm excluded from weight decay
Learning rate	2×10^{-5}
Schedule	cosine, warmup ratio 0.03
Weight decay	0.01
Gradient clipping	1.0 (harness default)
Epochs	3 (both domains; ablation screening cells use 1)
Sequence length	2,048 tokens (left-truncated, padded to fixed length)
Effective batch	64 sequences $\approx$ 131,072 tokens/step
Precision	bf16
Loss masking	chat template; user-prompt tokens masked (-100), loss on assistant tokens only
Thinking mode	disabled for Qwen3 (training and evaluation)
Adapter scaling	$\alpha/r = 2.0$ on every arm, no rsLoRA
Adapter dropout	0.05
Adapter init	A: Kaiming uniform; B: zeros (adapter output is zero at init)
Checkpoints	every 500 steps, retained for diagnostics; final adapter is evaluated

Table 9: Instruction-tuning corpora. Each domain is serialized once into a JSONL corpus with a fixed seed; no subsampling or deduplication.

	Math domain	General domain
Corpus	MetaMathQA	Commonsense170K
Size	395,000	170,420
Source fields	query / response	instruction / output
Formatting	chat template (user+assistant turns)	same
Corpus file	`math_full_seed42.jsonl`	`general_full_seed42.jsonl`
Steps (3 epochs, batch 64)	18,516	7,989

B.3 Datasets and preprocessing

Math-domain arms train only on the math corpus and general-domain arms only on the general corpus. Backbones are the *base* (non-instruct) model variants throughout; base tokenizers without a chat template receive the same fallback Llama-3 template used during training.

Correctness gates. Three gates must hold after any geometry or code change: (i) $K{=}1$ with zero adapters reproduces the stock model exactly; (ii) omitted γ equals explicit $\gamma{=}1/K$ in logits, and non-default γ survives a save→load→unroll round trip; (iii) the unrolled dense model matches the looped training forward. Gradient checkpointing is disabled for looped arms (backward recomputation re-executes window passes whose adapter routing depends on the pass index) and enabled for the single-pass baseline.

Evaluation form. Looped arms are evaluated as physically unrolled dense models: the window is duplicated K times with each pass's adapter merged into the weights and the damping folded into the residual scaling, after which standard generation and likelihood interfaces apply unchanged.

B.4 Evaluation configuration

Table 10: Benchmark configuration (lm-evaluation-harness 0.4.12, greedy decoding, chat template applied for CoT tasks). "EM" = exact match; flexible = regex-relaxed variant; MV = MathVerify.

Task	Shots	Metric	n	Generation
GSM8K-CoT	5	strict / flexible	1,319	CoT, $\leq$2,048 new tokens
Minerva	4	EM / MV	5,000	CoT, $\leq$512 new tokens
PIQA	0	acc (acc_norm preferred)	1,838	likelihood
ARC-Easy	0	acc (acc_norm)	2,376	likelihood
ARC-Challenge	0	acc (acc_norm)	1,172	likelihood
HellaSwag	0	acc (acc_norm)	10,042	likelihood
Winogrande	0	acc	1,267	likelihood
CommonsenseQA	0	acc	1,221	likelihood
OpenBookQA	0	acc (acc_norm)	500	likelihood

Math-domain adapters are evaluated on GSM8K and Minerva. GSM8K uses 5-shot chain-of-thought prompting with greedy decoding and is scored using strict and flexible exact match, with their maximum reported as Best. Minerva uses 4-shot prompting and is reported using exact match and MathVerify. General-domain adapters are evaluated on PIQA, ARC-Easy, ARC-Challenge, HellaSwag, Winogrande, CommonsenseQA, and OpenBookQA using 0-shot likelihood-based evaluation. All recurrent models are evaluated after conversion to their equivalent unrolled form.

B.5 Structural Ablations: Recurrent Depth and Window Width

We further examine how recurrent depth and window width affect performance under a fixed trainable-parameter budget. Starting from the RLR-I configuration, we vary the number of recurrent evaluations K, the corresponding adapter rank r, and the width of the recurrent window $w = L_{\mathrm{rec}}$.

Table 11: Structural ablations of recurrent depth and window width on Llama-3.2-1B under matched trainable-parameter budgets. Increasing recurrent depth jointly reduces adapter rank to preserve the parameter budget. The best result is shown in **bold** and the second-best is underlined for Avg and GSM8K Best.

Ablation Target	Configuration	Commonsense Reasoning								GSM8K
		PIQA	ARC-E	ARC-C	HS	WG	CSQA	OBQA	Avg	Best
Llama-3.2-1B										
RLR-I	$w = 4, K = 2, r = 8$	72.09	38.05	31.06	57.96	54.93	19.33	35.60	44.15	15.77
Loop Depth	$K = 8, r = 2$	70.51	39.10	31.66	55.70	56.67	19.49	34.80	43.99	21.76
Adapter Width	$w = 8$	72.91	39.98	31.74	55.89	55.72	19.16	35.00	44.34	17.36
	$w = 16$	61.43	33.29	26.88	46.32	50.43	21.87	33.60	39.12	12.51

Trading adapter rank for recurrent depth. Increasing the recurrence horizon while reducing adapter rank produces a different performance profile from the reference configuration. The deeper, lower-rank configuration substantially improves GSM8K while maintaining a similar average score on the commonsense suite. Because recurrent depth and adapter rank change jointly to preserve the parameter budget, this comparison does not isolate either factor independently. Instead, it demonstrates that the same trainable-parameter budget can be reallocated from per-step adapter capacity toward additional recurrent computation without necessarily degrading overall performance.

Effect of recurrent window width. Moderately increasing the recurrent window improves both reported aggregates relative to the narrower reference configuration, whereas extending recurrence across the full backbone substantially degrades performance. This suggests that the location and extent of recurrent reuse are important design choices: additional reused computation is not uniformly beneficial simply because the trainable-parameter budget is preserved. We leave a more systematic study of window placement and width to future work.