MIXTURE OF LATENT RECURSIONS FOR LOW-RANK ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Low-Rank Adaptation (LoRA) performs task-specific adaptation using a fixed low-rank architecture, whose expressiveness is typically increased by enlarging or adding learnable modules at a proportional parameter overhead. We introduce Mixture of Latent Recursions (MoLR), an adaptive recursion framework that enhances adaptation expressiveness while preserving parameter efficiency. MoLR recursively refines the down-projected latent representation of each input using a shared transformation. A lightweight threshold-based router independently determines whether each token should undergo further refinement, while dynamically adjusted thresholds encourage balanced utilization across recursion depths. Extensive experiments across diverse language and vision tasks demonstrate that MoLR consistently outperforms representative LoRA variants with negligible parameter overhead over LoRA. Further analyses show that MoLR learns input-dependent recursion-depth allocation and enables a controllable inference-time compute–performance trade-off without requiring retraining.

1 INTRODUCTION

As foundation models have grown to billions of parameters, full-parameter fine-tuning has become increasingly resource-intensive (Zhang et al., 2023c). To mitigate this challenge, Low-Rank Adaptation (LoRA) (Hu et al., 2022) has emerged as a prominent parameter-efficient fine-tuning mechanism owing to its simple formulation and broad applicability. Rather than directly updating the pre-trained model weights, LoRA keeps them frozen and parameterizes task-specific weight updates as the product of two trainable low-rank matrices. By optimizing these lightweight matrices only, LoRA can achieve performance comparable to, and in some cases exceeding, that of full-parameter fine-tuning while reducing the associated computational and memory costs.

Beyond enabling efficient fine-tuning, the low-rank factorization underlying LoRA and many of its subsequent variants (Kalajdzievski, 2023; Zhang et al., 2023b; Hayou et al., 2024; Meng et al., 2024; Wang et al., 2025) can be naturally interpreted from a latent-space perspective. Under this view, each input is first down-projected into a low-dimensional latent state and then directly mapped to a high-dimensional output update without any intervening processing. Since a single latent entry can affect multiple output dimensions through the learned latent-to-output mapping, transformations confined to this compact space can induce broad and coordinated changes in the overall adaptation. At the same time, the low dimensionality of the latent space offers a natural means of controlling the overhead of such transformations. Building upon these properties, recent work (Bałazy et al., 2024; Wu et al., 2024; Dong et al., 2025; Li et al., 2025b) has begun to explore introducing explicit latent-space computation to enhance the expressiveness of low-rank adaptation.

However, despite their demonstrated effectiveness, existing approaches that incorporate latent-space computation still struggle to decouple further gains in adaptation expressiveness from corresponding increases in the number of trainable parameters. In these methods, latent transformations are implemented through fixed feed-forward architectures whose learnable modules are separately parameterized. Consequently, realizing more complex latent transformations (e.g., by enlarging the latent dimensionality, stacking additional transformations, or introducing further learnable modules) typically requires expanding the architecture, resulting in parameter overhead that grows with the size or number of its parameterized components. This raises a central question: *how can we enrich latent-space computation without proportional parameter growth?*

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.

In this paper, we propose Mixture of Latent Recursions (MoLR), an adaptive recursion framework for low-rank adaptation. To answer this question, MoLR recursively refines the latent state of each input through a shared transformation, enabling richer latent-space computation without trainable-parameter growth proportional to the maximum recursion depth. To account for inherent semantic and contextual variation across inputs, we introduce a lightweight threshold-based router that independently determines at each step whether recursion should continue. This realizes input-dependent recursion depths while avoiding the future-token information leakage from the conventional sequence-wide expert-choice routing (Bae et al., 2025b). To mitigate routing decisions from collapsing to a narrow range of depths, we dynamically adjust the routing thresholds to encourage balanced depth allocation without incurring potentially interfering auxiliary-loss gradients.

We conduct extensive experiments across diverse language and vision tasks. MoLR achieves the highest performance among representative LoRA variants while requiring a parameter budget comparable to LoRA. Our analyses reveal token-wise recursion-depth patterns, suggesting that the learned routing is sensitive to semantic variation across tokens. We also observe test-time scaling within the trained recursion horizon, where larger inference-time depth caps improve performance without retraining. Finally, incorporating adaptive latent recursion into existing LoRA variants consistently improves their performance, demonstrating compatibility across distinct formulations and suggesting latent-space computation as a promising scaling axis for low-rank adaptation.

Our main contributions are summarized as follows:

- We propose MoLR, an adaptive recursion framework for low-rank adaptation.
- We introduce adaptive latent recursion, enabling input-dependent allocation of latent-space computation without parameter cost proportional to the maximum recursion depth.
- We demonstrate that MoLR surpasses representative LoRA variants on diverse tasks with token-wise depth allocation. We also showcase its test-time scaling behavior and the compatibility of adaptive latent recursion with distinct existing LoRA formulations.

2 Related Work

Low-Rank Adaptation. LoRA (Hu et al., 2022) enables parameter-efficient adaptation by freezing the pre-trained model parameters and learning low-rank weight updates. These updates can be merged into the base weights after fine-tuning, thereby introducing no additional inference-time computation. Recent work has progressed along three primary directions: (i) improving empirical performance and reducing the gap relative to full-parameter fine-tuning (Ye et al., 2026; Zhang et al., 2026b; Singhal et al., 2026); (ii) characterizing and improving its optimization dynamics, including conditioning, initialization, and the stability of feature learning (Castin et al., 2026; Zhang et al., 2026a; Wu et al., 2026); and (iii) extending its applicability to scalable multi-task and continual learning settings (Tian et al., 2026; He et al., 2026; Lu et al., 2026). These advances reinforce LoRA as a flexible and efficient framework for adapting large foundation models.

Latent-Space Computation in LoRA. To enhance the expressiveness of LoRA, recent work has introduced explicit learnable computation between its down- and up-projections, which can be broadly categorized into: (i) linear transformations (Bałazy et al., 2024; Wu et al., 2024); (ii) nonlinear transformations (Dong et al., 2025; Li et al., 2025a; Chen, 2026) and (iii) input-conditioned transformations that adapt the latent operator to individual inputs (Li et al., 2025b). These methods rely on fixed feed-forward architectures, such that realizing more complex latent transformations typically requires enlarging that architecture with a corresponding increase in the parameter count. This motivates us to explore how to enrich latent-space computation without proportional parameter growth.

Recurrence and Conditional Computation. Recurrence was introduced in early neural network models as a parameter-efficient mechanism for increasing computational depth through repeated application of shared transformations (Elman, 1990; Hochreiter & Schmidhuber, 1997). It was later extended to Transformer architectures, where recurrent or weight-tied blocks enable deeper computation, compact parameterizations, and iterative latent-space reasoning (Dehghani et al., 2019; Giannou et al., 2023; Bae et al., 2025a; Geiping et al., 2025). More recent work further incorporates conditional depth, allowing computation to vary across inputs or tokens according to their refinement needs (Schuster et al., 2022; Bae et al., 2025b; Jeddi et al., 2026).

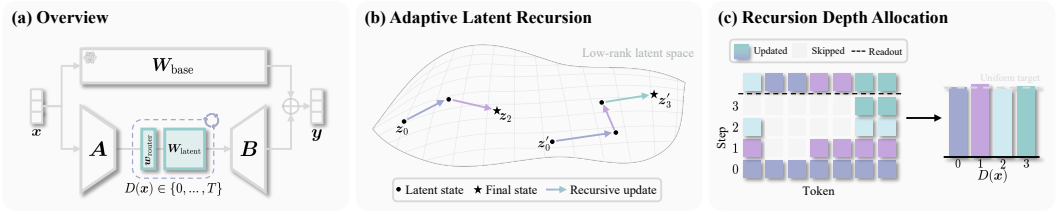


Figure 1: **MoLR**. (a) MoLR recursively refines each latent state to a router-determined depth $D(\boldsymbol{x})$. (b) By using the shared latent transformation, MoLR yields input-specific trajectories within the latent space without incurring trainable parameter growth as the maximum recursion depth T increases. (c) Once a token halts, subsequent recursive updates are skipped, and its current latent state is retained for readout. We encourage a uniform distribution over stopping depths to promote balanced computation allocation and supervision throughout the recursion horizon.

3 METHOD

In this section, we first present a latent-space interpretation of LoRA and then introduce MoLR, an adaptive recursion framework for low-rank adaptation, as shown in Fig. 1.

3.1 LOW-RANK ADAPTATION

LoRA (Hu et al., 2022) parameterizes updates to a frozen base weight $\boldsymbol{W}_{\text{base}} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ into learnable low-rank matrices. Given an input $\boldsymbol{x} \in \mathbb{R}^{d_{\text{in}}}$, the output $\boldsymbol{y}_{\text{LoRA}} \in \mathbb{R}^{d_{\text{out}}}$ is defined as:

$$\boldsymbol{y}_{\text{LoRA}} = \boldsymbol{W}_{\text{base}}\boldsymbol{x} + \frac{\alpha}{r}\boldsymbol{B}\boldsymbol{z}, \quad \boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} \in \mathbb{R}^r, \tag{1}$$

where $\boldsymbol{A} \in \mathbb{R}^{r \times d_{\text{in}}}$ and $\boldsymbol{B} \in \mathbb{R}^{d_{\text{out}} \times r}$ denote the down- and up-projection matrices, respectively, r is the rank that is chosen to satisfy $r \ll \min(d_{\text{in}}, d_{\text{out}})$, and α is a scaling coefficient. This formulation exposes a latent-space interpretation of LoRA: Each input $\boldsymbol{x}$ is first encoded by $\boldsymbol{A}$ into an r-dimensional latent state $\boldsymbol{z}$, which is directly mapped by $\boldsymbol{B}$ to an output residual without any intervening transformation. A natural way to increase the expressive capacity of LoRA is to enlarge r, at the cost of trainable-parameter overhead, $r(d_{\text{in}} + d_{\text{out}})$, that grows proportionally with r.

3.2 MIXTURE OF LATENT RECURSIONS

To decouple the complexity of latent-space computation from proportional growth in trainable parameters, we introduce MoLR, which recursively refines the latent state through a parameter-shared transformation across recursion steps. Given an input $\boldsymbol{x}$, the output $\boldsymbol{y}_{\text{MoLR}} \in \mathbb{R}^{d_{\text{out}}}$ is defined as:

$$\boldsymbol{y}_{\text{MoLR}} = \boldsymbol{W}_{\text{base}}\boldsymbol{x} + \frac{\alpha}{r}\boldsymbol{B}\boldsymbol{z}_T, \quad \boldsymbol{z}_0 = \boldsymbol{A}\boldsymbol{x}, \tag{2}$$

where T denotes the maximum recursion depth and $\boldsymbol{z}_T \in \mathbb{R}^r$ is the resulting latent state. For an input with zero executed recursion steps, $\boldsymbol{z}_T = \boldsymbol{z}_0 = \boldsymbol{A}\boldsymbol{x}$, so its output coincides with that of LoRA.

Adaptive Latent Recursion. To account for semantic and contextual variations across inputs, we introduce adaptive latent recursion with state-dependent routing. At each step $t \in \{0, \ldots, T-1\}$, MoLR first computes a continuation score $s_t \in (0, 1)$ for each active input with latent state $\boldsymbol{z}_t$:

$$s_t = \text{sigmoid}\left(\boldsymbol{w}_{\text{router}}^\top \boldsymbol{z}_t\right), \tag{3}$$

where $\boldsymbol{w}_{\text{router}} \in \mathbb{R}^r$ is the shared router weight across steps. The latent state is updated as follows:

$$\boldsymbol{z}_{t+1} = \begin{cases} \boldsymbol{z}_t + s_t \boldsymbol{W}_{\text{latent}}(\boldsymbol{z}_t + \phi(t)), & \text{if} \quad s_t > b_t, \\ \boldsymbol{z}_t & \text{otherwise}, \end{cases} \tag{4}$$

where $\boldsymbol{W}_{\text{latent}} \in \mathbb{R}^{r \times r}$ denotes the shared latent transformation across steps, $b_t \in \mathbb{R}$ is the routing threshold at step t, and $\phi : \{0, \ldots, T-1\} \to \mathbb{R}^r$ represents a fixed step-encoding function:

$$\phi(t)_{2i} = \sin\left(\frac{2\pi t(i+1)}{T}\right), \quad \phi(t)_{2i+1} = \cos\left(\frac{2\pi t(i+1)}{T}\right), \tag{5}$$

Table 1: **Structural comparison of LoRA variants.** For each variant, we report its equivalent latent-space operator, additional trainable parameters, and worst-case per-token multiply–accumulate operations (MACs) relative to LoRA. In the latent space, LoRA passes the latent state directly to up-projection; MoSLoRA applies a learnable linear transformation $\boldsymbol{W}_{\text{latent}} \in \mathbb{R}^{r\times r}$; TopLoRA applies an input-dependent diagonal transformation $\boldsymbol{\Sigma}(\boldsymbol{x}) \in \mathbb{R}^{r\times r}$; and MoLR performs adaptive recursive refinement with input-dependent depths.

Method	Input-Conditioned	Adaptive Depth	Latent Operator	ΔParams.	ΔMACs
LoRA	×	×	$\boldsymbol{I}_r$	–	–
MoSLoRA	×	×	$\boldsymbol{W}_{\text{latent}}$	r^2	r^2
TopLoRA	✓	×	$\boldsymbol{\Sigma}(\boldsymbol{x})$	rd_{in}	rd_{in}
MoLR	✓	✓	$\sum_{d=0}^{D(\boldsymbol{x})}\gamma_d(\boldsymbol{x})\boldsymbol{W}_{\text{latent}}^d$	r^2+r	$T(r^2+r)$

with $i \in \{0,\dots,r/2-1\}$ indexing coordinate pairs across r latent dimensions. Herein, an active input undergoes further refinement only when its state-dependent continuation score satisfies $s_t > b_t$. For inputs that continue, the step encoding $\phi(t)$ is added to provide explicit recursion-stage information before the shared transformation $\boldsymbol{W}_{\text{latent}}$ is applied. The resulting residual update is scaled by s_t, allowing the router to modulate the refinement and receive gradients through the executed update. Once an input fails the continuation criterion, its recursion terminates permanently, and its current latent state is preserved as the final representation $\boldsymbol{z}_T$.

For an input $\boldsymbol{x}$ that realizes a recursion depth $D(\boldsymbol{x}) \in \{0,\dots,T\}$, expanding the recursive update in Eq. (4) while omitting $\phi(t)$ for notational clarity yields:

$$\boldsymbol{z}_{D(\boldsymbol{x})} = \prod_{t=0}^{D(\boldsymbol{x})-1} \left(\boldsymbol{I}_r + s_t \boldsymbol{W}_{\text{latent}}\right) \boldsymbol{z}_0 = \sum_{d=0}^{D(\boldsymbol{x})} \gamma_d(\boldsymbol{x}) \boldsymbol{W}_{\text{latent}}^d \boldsymbol{z}_0, \tag{6}$$

where $\gamma_d(\boldsymbol{x})$ is determined by the continuation scores s_t across the executed steps, with $\gamma_0(\boldsymbol{x}) = 1$. The expansion shows that adaptive latent recursion induces an input-dependent polynomial in the shared latent transformation $\boldsymbol{W}_{\text{latent}}$. The realized depth $D(\boldsymbol{x})$ determines the maximum transformation order, while s_t governs the contribution of each order. As a result, MoLR yields a family of effective latent transformations without trainable parameter growth proportional to T. Detailed discussion on the exact expansion of Eq. (4) is provided in App. B.

Load Balancing. Unconstrained routing can concentrate training examples at a narrow range of stopping depths, leaving final adapter outputs at other depths with less direct supervision and potentially limiting the effectiveness of the shared latent transformation when recursion terminates at those depths. To address this issue, we specify a uniform target distribution over the realized recursion depths $D(\boldsymbol{x})$ that is defined as:

$$\mathbb{P}_{\text{target}}(D(\boldsymbol{x}) = d) = \frac{1}{T+1}, \quad \mathbb{E}_{\text{target}}[D(\boldsymbol{x})] = \frac{T}{2}, \quad \forall d \in \{0,\dots,T\}. \tag{7}$$

The target fraction of active inputs that should undergo another refinement at step t is then:

$$q_t = \mathbb{P}_{\text{target}}(D(\boldsymbol{x}) \geq t+1 \mid D(\boldsymbol{x}) \geq t) = \frac{\mathbb{P}_{\text{target}}(D(\boldsymbol{x}) \geq t+1)}{\mathbb{P}_{\text{target}}(D(\boldsymbol{x}) \geq t)} = \frac{T-t}{T-t+1}. \tag{8}$$

Following Wang et al. (2024a), we update the threshold at each recursion step using:

$$b_t \leftarrow b_t + \eta \,\text{sign}(\hat{q}_t - q_t)\,, \tag{9}$$

where $\eta \in \mathbb{R}_{>0}$ denotes the threshold learning rate, and $\hat{q}_t$ is the observed fraction of active inputs that continue at step t. We zero-initialize b_t so that every input initially traverses the full recursion horizon. This avoids premature suppression of recursion due to unreliable early routing decisions, facilitating the emergence of diverse depth allocation patterns in subsequent training.

Computational Overhead. Tab. 1 summarizes the equivalent latent operators, additional trainable parameters, and worst-case per-token multiply–accumulate operations (MACs) of different LoRA variants. It can be seen that the worst-case operational overhead of MoLR relative to LoRA is:

$$\frac{T(r^2+r)}{r(d_{\text{in}}+d_{\text{out}})} = \frac{T(r+1)}{d_{\text{in}}+d_{\text{out}}}. \tag{10}$$

Table 2: **Quantitative comparison of LoRA variants on NLU.** We report accuracy (↑) for all tasks except MRPC, QQP, and STSB, for which the combined score (↑) is used. #Param. denotes the number of trainable parameters. The **best** results are highlighted.

Model	Method	#Param.	CoLA	MNLI	MRPC	QNLI	QQP	SST-2	STSB	Avg.
RoBERTa-B	LoRA	1.33M	77.28	85.55	78.19	90.31	86.20	93.27	81.78	84.65
	DoRA	1.41M	77.75	85.27	76.56	90.27	86.20	93.42	81.74	84.46
	AdaLoRA	1.33M	77.95	85.44	75.96	90.94	86.60	93.46	85.39	85.11
	TopLoRA	1.99M	78.84	85.24	76.50	90.40	86.08	93.39	85.62	85.15
	HydraLoRA	1.58M	76.38	85.27	79.46	89.97	85.88	93.27	78.56	84.11
	MoSLoRA	1.33M	76.25	85.45	74.80	90.35	86.07	93.27	80.43	83.80
	AuroRA	1.34M	77.37	85.57	76.27	90.43	86.16	93.35	85.40	84.94
	MoLR	1.33M	**79.86**	**85.61**	**82.10**	**91.04**	**86.62**	**93.62**	**86.73**	**86.51**
DeBERTa-v3-L	LoRA	3.54M	86.45	91.80	74.92	95.25	89.74	95.60	86.06	88.55
	DoRA	3.76M	86.64	91.68	74.92	95.21	89.69	95.68	86.44	88.61
	AdaLoRA	3.56M	87.41	91.75	81.29	95.31	89.87	95.72	88.64	90.00
	TopLoRA	5.31M	87.25	91.77	82.24	95.23	89.46	95.87	89.78	90.23
	HydraLoRA	4.20M	85.94	91.62	75.30	95.08	89.60	95.91	85.22	88.38
	MoSLoRA	3.55M	86.19	91.57	77.18	95.13	89.64	96.10	85.54	88.77
	AuroRA	3.56M	86.29	91.61	77.87	95.17	89.45	95.83	85.99	88.89
	MoLR	3.55M	**87.50**	**91.83**	**84.96**	**95.40**	**90.31**	**96.29**	**90.85**	**91.02**

Under the standard low-rank regime, where $r \ll \min(d_{\text{in}}, d_{\text{out}})$ and the maximum recursion depth T is small, this ratio remains limited. Thus, MoLR realizes adaptive, input-conditioned higher-order latent transformations while preserving a computational footprint comparable to that of LoRA. Moreover, this expression represents a worst-case bound in which every token executes all T recursion steps. In practice, early stopping allows latent states to terminate at shallower recursion depths, thereby avoiding unnecessary transformations and reducing the realized computational cost.

4 Experiments

In this section, we evaluate MoLR across a broad range of architectures and tasks, covering natural language understanding (Sec. 4.1), natural language reasoning (Sec. 4.2), and image classification (Sec. 4.3). We then examine its key characteristics in Sec. 4.4 and App. A.4 and conclude with ablation studies in Sec. 4.5 that explore the design choices for individual components.

Implementation Details. We implement all models in PyTorch and train them on NVIDIA A100 GPUs. Unless otherwise specified, we apply adapters to all linear modules in the backbone, with $r = 8$ and $\alpha = 2r$ used across all baselines. For each baseline, we adopt the recommended hyperparameters from its paper or official codebase whenever available. For MoLR, we use $T = 2$ and $\eta = 1 \times 10^{-3}$ across all tasks. We repeat every experiment with three different random seeds and report the average performance, with the across-seed variability provided in App. A.4.

Baselines. We compare MoLR against representative LoRA variants spanning diverse methodological categories, including: (i) LoRA (Hu et al., 2022); (ii) DoRA (Liu et al., 2024), which decomposes the pre-trained weight into magnitude and direction, learning the magnitude separately while using LoRA to update the direction; (iii) AdaLoRA (Zhang et al., 2023c), which adaptively allocates rank budget among weight matrices; (iv) TopLoRA (Li et al., 2025b), which enables token-wise input-output projections; (v) HydraLoRA (Tian et al., 2024), which shares a down-projection across multiple up-projections and dynamically combines their outputs; (vi) MoSLoRA (Wu et al., 2024), which inserts a trainable mixer to fuse subspaces in LoRA; and (vii) AuroRA (Dong et al., 2025), which introduces adaptive nonlinearity between LoRA linear projectors.

4.1 Natural Language Understanding (NLU)

Settings. We fine-tune RoBERTa-B (Liu et al., 2019) and DeBERTa-v3-L (He et al., 2021) on seven tasks from the GLUE benchmark (Wang et al., 2019). Detailed settings are provided in App. A.1.

Table 3: **Quantitative comparison of LoRA variants on NLR.** We report accuracy (↑) for all other tasks and Pass@1 (↑) for HumanEval. #Param. denotes the number of trainable parameters. The **best** results are highlighted. HS = HellaSwag, HE = HumanEval, WG = WinoGrande.

Model	Method	#Param.	ARC-C	BoolQ	GSM8K	HS	HE	MMLU	WG	Avg.
LLaMA-3.2-3B	LoRA	12.16M	49.46	88.20	34.49	78.99	23.98	55.73	83.64	59.21
	DoRA	12.93M	50.11	88.16	35.03	79.02	24.19	55.67	83.80	59.42
	AdaLoRA	12.16M	46.19	81.91	23.08	77.96	25.20	55.79	78.98	55.59
	TopLoRA	18.12M	50.97	88.84	39.83	**79.42**	24.19	**56.25**	83.58	60.44
	HydraLoRA	14.51M	47.84	87.41	33.54	78.92	23.58	55.75	83.21	58.61
	MoSLoRA	12.17M	52.16	89.33	38.06	78.78	22.36	54.46	83.71	59.84
	AuroRA	12.18M	52.64	88.91	35.51	79.05	24.80	53.34	83.74	59.71
	MoLR	12.17M	**54.69**	**89.52**	**40.18**	79.11	**25.61**	55.48	**83.90**	**61.21**
Qwen-3-8B	LoRA	21.82M	59.61	90.12	81.43	79.51	59.35	74.02	82.90	75.28
	DoRA	23.22M	60.47	90.06	82.11	79.51	59.15	74.02	82.80	75.45
	AdaLoRA	21.82M	56.91	89.61	81.58	79.50	60.98	73.83	83.40	75.11
	TopLoRA	32.44M	62.88	90.72	81.35	**79.90**	55.49	**74.40**	83.53	75.47
	HydraLoRA	26.10M	56.40	89.67	81.81	79.25	61.18	74.01	82.37	74.96
	MoSLoRA	21.84M	57.39	89.78	81.60	79.36	**61.59**	73.99	82.90	75.23
	AuroRA	21.86M	60.24	90.85	81.53	79.19	55.90	73.61	82.85	74.88
	MoLR	21.84M	**64.53**	**91.57**	**82.21**	79.66	59.15	74.09	**83.66**	**76.41**

Results. Tab. 2 shows that MoLR achieves the highest average performance across both backbones and leads on nearly all language understanding tasks. Despite using a parameter budget comparable to LoRA, it outperforms higher-parameter variants such as TopLoRA and methods with explicit nonlinearity like AuroRA. These results highlight the effectiveness of adaptive latent recursion.

4.2 Natural Language Reasoning (NLR)

Settings. We fine-tune Llama-3.2-3B (Grattafiori et al., 2025) and Qwen3-8B (Yang et al., 2025) on tasks spanning commonsense reasoning, reading comprehension, general knowledge understanding, mathematical reasoning, and code generation. Detailed settings are provided in App. A.2.

Results. Tab. 3 shows that MoLR achieves the highest average performance on both backbones, obtaining the best results on five tasks with LLaMA-3.2-3B and four tasks with Qwen3-8B. The improvements are particularly pronounced on ARC-C, with consistent gains on BoolQ, GSM8K, and WinoGrande. These results suggest that MoLR generalizes beyond language understanding, delivering consistent improvements across diverse reasoning tasks, backbones, and model scales.

4.3 Image Classification (IC)

Settings. We fine-tune CLIP-ViT-B/16 (Radford et al., 2021) and SigLIP-ViT-L/16 (Zhai et al., 2023) on seven image classification benchmarks. Detailed settings are provided in App. A.3.

Results. As shown in Tab. 4, MoLR achieves the highest average accuracy on both backbones and the best performance on every dataset. It consistently outperforms all baselines, with the largest margin over the next-best method observed on DTD. Together with the language results shown in Tabs. 2 and 3, these findings indicate that adaptive latent recursion transfers effectively to vision tasks, highlighting its applicability across different architectures and modalities.

4.4 Analysis

Recursion Depth Allocation. Fig. 2a visualizes the token-wise recursion depth allocation for an MMLU example. Herein, deeper recursion frequently appears around task-relevant contexts such as "25 suitcases" and "Three are selected," whereas question scaffolding like "There are" and "What is" receives no recursion in most cases. Candidate answers and the final answer are also predominantly processed at maximum recursion depth. This non-uniform depth allocation is consistent with MoLR learning to adapt latent computation to semantic and contextual differences across tokens.

Table 4: **Quantitative comparison of LoRA variants on IC.** We report the top-1 test accuracy (↑) for all datasets. #Param. denotes the number of trainable parameters. The **best** results are highlighted. Abbreviations: ESAT = EuroSAT, RE45 = RESISC45.

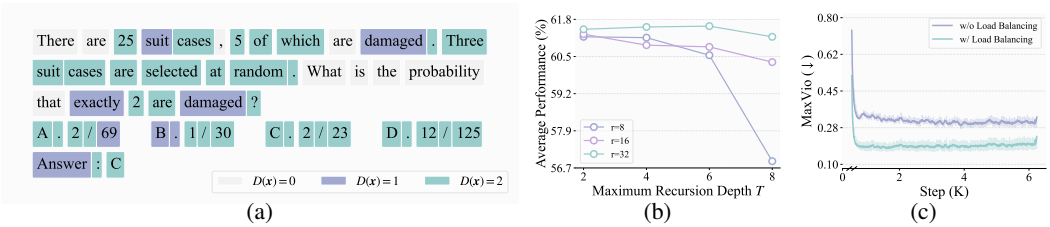
Model	Method	#Param.	Cars	DTD	ESAT	GTSRB	RE45	SUN397	SVHN	Avg.
CLIP-ViT-B	LoRA	1.33M	71.65	52.20	97.62	97.08	91.88	73.45	96.59	82.92
	DoRA	1.41M	71.85	52.38	97.67	97.15	92.06	73.56	96.62	83.04
	AdaLoRA	1.33M	70.54	46.24	97.59	97.30	92.13	74.32	96.58	82.10
	TopLoRA	1.99M	75.05	56.34	97.98	97.71	93.53	75.10	96.88	84.66
	HydraLoRA	1.58M	70.32	49.65	97.50	96.82	91.26	72.48	96.36	82.06
	MoSLoRA	1.33M	70.57	50.37	97.39	96.78	90.97	72.61	96.38	82.15
	AuroRA	1.34M	70.54	52.41	97.45	96.37	90.68	72.07	96.29	82.26
	MoLR	1.33M	**77.55**	**65.05**	**98.17**	**98.22**	**94.43**	**75.90**	**97.07**	**86.63**
SigLIP-ViT-L	LoRA	3.54M	94.91	66.13	97.94	98.23	93.41	78.42	97.13	89.45
	DoRA	3.76M	95.01	66.40	97.80	98.31	93.62	78.54	97.18	89.55
	AdaLoRA	3.54M	94.81	64.70	97.92	98.62	94.13	80.01	97.18	89.62
	TopLoRA	5.31M	95.33	68.12	98.25	98.75	94.91	80.33	97.35	90.44
	HydraLoRA	4.20M	94.72	65.13	97.52	97.78	92.64	77.58	96.95	88.90
	MoSLoRA	3.55M	94.78	65.39	97.59	98.08	93.03	77.99	97.00	89.12
	AuroRA	3.56M	94.87	67.27	97.57	97.37	92.18	77.55	96.73	89.08
	MoLR	3.55M	**95.77**	**73.30**	**98.31**	**98.89**	**95.39**	**80.91**	**97.45**	**91.43**

Maximum Recursion Depth. Fig. 2b examines the effect of the maximum recursion depth T across different ranks. At smaller ranks, performance is lower at larger T, whereas increasing r substantially attenuates this decline. This interaction suggests that longer recursion horizons may benefit from a more expressive latent space and higher-capacity shared latent transformation, which can better mitigate potential interference induced by reusing the same transformation across recursion steps. At $r = 32$, performance saturates and slightly declines at the largest T, indicating that an excessively large maximum depth is not inherently beneficial even with greater latent capacity. This observation is consistent with similar saturation or degradation reported in other recursive models (Prairie et al., 2026; Yang et al., 2026). Overall, these results suggest that longer recursion horizons are neither universally beneficial nor necessary once sufficient refinement has been achieved.

Recursion Depth Utilization. We evaluate how evenly MoLR utilizes the recursion depths using MaxVio (Wang et al., 2024a), defined as the maximum relative excess over the expected load under uniform depth utilization. We compare against a baseline that resamples the target depth distribution at each step rather than using the fixed distribution defined in Eq. (7). Fig. 2c shows that the threshold update in Eq. (9) consistently achieves lower MaxVio throughout training, indicating more balanced utilization across recursion depths. It also improves average NLR performance by 1.09 percentage points over the resampling baseline. These results validate the effectiveness of the load-balancing mechanism in promoting balanced optimization across stopping depths.

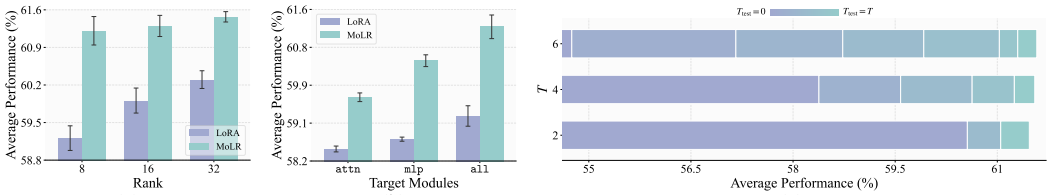
Robustness. Figs. 3a and 3b evaluate MoLR across varying ranks and target-module coverage. MoLR shows consistently strong performance across all evaluated configurations. This stability suggests that its effectiveness persists across the evaluated ranks and target-module sets, indicating its robustness across a range of adaptation budgets. Notably, MoLR with $r = 8$ outperforms LoRA with higher ranks despite using fewer trainable parameters. These results suggest that increasing computational expressivity within the latent space through adaptive latent recursion can be more effective and parameter-efficient than simply expanding the latent dimension. Detailed comparisons with additional representative LoRA variants are provided in App. A.4.

Test-Time Scaling. We investigate whether MoLR enables a controllable test-time compute-performance trade-off through adaptive latent recursion. For a model trained with maximum recursion depth T, we vary the inference-time maximum recursion depth $T_{\text{test}} \leq T$ while preserving routing and early stopping. Fig. 3c shows that the average performance consistently improves with increasing T_{test} across all recursion horizons, and the cumulative gain from $T_{\text{test}} = 0$ to $T_{\text{test}} = T$ is larger for checkpoints trained with larger T. Consistent with the test-time scaling behavior observed in Bae et al. (2025b), these results support that MoLR can trade additional inference computation for improved downstream performance without retraining.



(a) (b) (c)

Figure 2: **Analysis of adaptive latent recursion in MoLR on NLR.** (a) An example token-wise recursion depth allocation pattern on MMLU when $T = 2$. Each token is colored according to its most frequent recursion depth realized across all modules in layer 8 of Llama-3.2-3B. (b) Average performance when sweeping $T \in \{2, 4, 6, 8\}$ across ranks $r \in \{8, 16, 32\}$. (c) Comparison of MaxVio (↓) with and without load balancing averaged across all modules.



(a) (b) (c)

Figure 3: **Robustness and test-time scaling analysis of MoLR on NLR.** We evaluate how average performance changes with (a) rank r, (b) target modules, and (c) inference-time maximum recursion depth T_{test}. Detailed comparisons with additional LoRA variants for (a) and (b) are in App. A.4.

Computational Overhead. Tab. 5 shows that MoLR achieves the best overall performance while introducing controlled computational and memory overhead. MoLR increases per-step training time by about 13% and peak training memory by less than 2% over LoRA. At inference, although input-dependent routing prevents MoLR from being merged into the backbone, its per-token latency falls within the range of the other non-mergeable variants. These results show that MoLR improves downstream task performance with modest overhead compared with other non-mergeable variants.

Table 5: **Computational overhead of LoRA variants on NLR.** We report per-step training time, per-token inference latency, peak GPU memory, and average performance. Detailed profiling settings are provided in App. A.4.

Method	Training		Inference		Avg.
	Time (s)	Mem. (GB)	Lat. (ms)	Mem. (GB)	
LoRA	0.55	20.76	0.18	6.09	59.21
TopLoRA	0.59	21.27	0.44	6.15	60.44
HydraLoRA	0.84	29.67	0.60	6.14	58.61
AuroRA	0.61	21.15	1.15	6.13	59.71
MoLR	0.62	21.12	0.69	6.13	61.21

4.5 Ablation Studies

In this subsection, we conduct ablation studies to systematically examine the effects of key design choices in MoLR. Unless otherwise specified, all experiments use Llama-3.2-3B as the backbone and follow the experimental protocol described in Sec. 4.2.

Latent Recursion. Tab. 6a compares four latent recursion designs: (i) *no recursion*, which leaves the latent state unchanged; (ii) *fixed depth*, which applies the one-step gated linear transformation as in Eq. (4) to all latent states; (iii) *random depth*, which samples recursion depths from a uniform distribution with $T = 2$; and (iv) *input-dependent depth*, as defined in Eqs. (3) and (4). Among the variants designed for the same expected recursion depth, input-dependent depth achieves the highest average performance. Its advantage over uniformly sampled depth suggests that the gains are not explained by depth variation alone and is consistent with a benefit from input-dependent routing.

Routing. Tab. 6b compares different routing strategies: (i) *expert-choice routing*, which selects the highest-scoring tokens at each recursion step based on their current latent states $\boldsymbol{z}_t$; (ii) *token-choice routing from latent states*, which assigns each token a recursion depth in a single decision based on its initial latent state $\boldsymbol{z}_0$; (iii) *token-choice routing from inputs*, which instead predicts the recursion depth directly from $\boldsymbol{x}$; and (iv) *threshold-based routing*, as defined in Eq. (3). Threshold-based routing achieves the strongest overall performance. Unlike expert-choice routing, it makes token-wise decisions independently of other tokens, avoiding the information leakage induced by top-k

Table 6: **Ablation of MoLR on NLR.** We investigate different design choices for (a) latent recursion, (b) routing, (c) step conditioning, and evaluate sensitivity to (d) η. We report accuracy (↑) for all other tasks and Pass@1 (↑) for HumanEval. The **best** results and default designs are highlighted.

	Case	ARC-C	BoolQ	GSM8K	HS	HE	MMLU	WG	Avg.
(a)	No recursion	49.46	88.20	34.49	78.99	23.98	55.73	83.64	59.21
	Fixed	54.01	88.87	39.73	78.99	23.17	55.26	83.19	60.46
	Random	49.83	88.15	35.41	79.01	24.59	55.60	84.08	59.53
	Input-dependent	54.69	89.52	40.18	79.11	25.61	55.48	83.90	**61.21**
(b)	Expert-choice ($\boldsymbol{z}_t$)	53.70	89.57	39.80	78.95	24.59	55.46	83.87	60.85
	Token-choice ($\boldsymbol{z}_0$)	54.30	89.73	40.03	79.02	23.78	55.71	83.95	60.93
	Token-choice ($\boldsymbol{x}$)	54.21	89.46	40.23	79.15	23.78	55.74	84.08	60.95
	Threshold-based ($\boldsymbol{z}_t$)	54.69	89.52	40.18	79.11	25.61	55.48	83.90	**61.21**
(c)	No conditioning	54.49	89.44	40.89	79.11	24.19	55.39	83.77	61.04
	Fixed sinusoidal	54.69	89.52	40.18	79.11	25.61	55.48	83.90	**61.21**
	Learned additive	54.32	89.34	39.60	79.04	23.78	55.79	84.29	60.88
	Learned scaling	54.44	89.29	40.49	79.07	24.80	55.44	83.82	61.05
	Step-specific transform	54.35	89.37	41.37	79.05	25.81	55.73	84.06	61.39
(d)	$\eta = 1 \times 10^{-2}$	54.38	89.23	39.98	79.10	25.00	55.40	83.72	60.97
	$\eta = 1 \times 10^{-3}$	54.69	89.52	40.18	79.11	25.61	55.48	83.90	**61.21**
	$\eta = 1 \times 10^{-4}$	54.44	89.27	40.56	79.17	25.20	55.53	83.79	61.14

selection in causal models (Bae et al., 2025b). Unlike one-shot token-choice routing, it re-evaluates whether to continue recursion from the evolving latent state at each step, receiving task-gradient supervision at every executed update. Moreover, the shared router introduces only r trainable parameters, compared with $(T+1)r$ and $(T+1)d_{\text{in}}$ for token-choice routing based on $\boldsymbol{z}_0$ and $\boldsymbol{x}$, respectively. These results indicate that our threshold-based routing can effectively allocate latent computation adaptively while preserving causal compatibility and parameter efficiency.

Step Conditioning. Tab. 6c compares different step-conditioning mechanisms: (i) *no conditioning*, which applies the shared latent transformation directly to $\boldsymbol{z}_t$; (ii) *fixed sinusoidal encoding*, which adds $\phi(t)$ to $\boldsymbol{z}_t$; (iii) *learned additive encoding*, which adds a learned step embedding $\boldsymbol{e}_t \in \mathbb{R}^r$; and (iv) *learned scaling*, which applies learned feature-wise scaling. We additionally include *step-specific transform* as a higher-capacity reference, which augments the shared latent transformation $\boldsymbol{W}_{\text{latent}}$ with a zero-initialized, step-specific residual $\Delta\boldsymbol{W}_t \in \mathbb{R}^{r\times r}$. Among the parameter-shared conditioning options, fixed sinusoidal encoding achieves the strongest overall performance, demonstrating the benefit of explicitly conditioning the shared transformation on recursion progress. Its performance is also comparable to the step-specific transform variant while avoiding Tr^2 additional parameters per adapted module, yielding a more favorable performance-parameter trade-off.

Hyperparameter Sensitivity. Tab. 6d evaluates the sensitivity of MoLR to the threshold learning rate η. The results show no significant performance variation across the evaluated scales, with $\eta = 1 \times 10^{-3}$ achieving the best result. This indicates that MoLR is robust to the choice of η.

5 Conclusion

In this work, we introduce Mixture of Latent Recursions (MoLR), a low-rank adaptation framework that expands latent-space computation without trainable-parameter growth proportional to the maximum recursion depth. MoLR recursively refines the low-rank latent state of each token through a shared transformation, with lightweight threshold-based routing to enable adaptive allocation of latent-space computation across inputs. Extensive experiments across diverse language and vision tasks demonstrate that MoLR achieves the highest performance among competitive LoRA variants under a parameter budget comparable to LoRA. Further analyses reveal token-dependent depth-allocation patterns and a controllable trade-off between inference-time computation and performance within the trained recursion horizon. More broadly, these findings suggest that the low-rank latent space can serve as a reusable computational workspace, pointing to iterative latent refinement as a promising direction for scaling parameter-efficient adaptation.

AI USE STATEMENT

In this work, we used generative AI tools for assisting in the writing of proofs, designing or providing feedback on research methodology or experiments, and assisting with translation. We have not used generative AI tools for proposing or refining hypotheses, developing theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, implementing methods, interpreting results, or supporting qualitative or thematic data analysis, and the remaining tasks with required disclosure are not applicable to this work. Additionally, we used generative AI tools for creating or modifying scientific figures or images, summarizing or analyzing existing literature, brainstorming, and formatting references. We have reviewed all AI-assisted work. The AI-assisted proofs and mathematical derivations were verified by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

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

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-Recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Advances in Neural Information Processing Systems*, 2025b.

Klaudia Bałazy, Mohammadreza Banaei, Karl Aberer, and Jacek Tabor. LoRA-XS: Low-rank adaptation with extremely small number of parameters. *arXiv preprint arXiv:2405.17604*, 2024.

Valérie Castin, Kimia Nadjahi, Pierre Ablin, and Gabriel Peyré. Balanced LoRA: Removing parameter invariance to accelerate convergence. In *International Conference on Machine Learning*, 2026.

Sahil Chaudhary. Code alpaca: An instruction-following LLaMA model for code generation. GitHub repository, 2023.

Hung-Hsuan Chen. CeRA: Breaking the linear ceiling of low-rank adaptation with non-linearity retained at inference. In *Conference on Language Modeling*, 2026.

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

Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 105(10):1865–1883, 2017.

Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 2924–2936, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

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.

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

Ning Ding, Xingtai Lv, Qiaosen Wang, Yulin Chen, Bowen Zhou, Zhiyuan Liu, and Maosong Sun. SoRA: Sparse low-rank adaptation of pre-trained language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4133–4145, 2023.

Haonan Dong, Wenhao Zhu, Guojie Song, and Liang Wang. AuroRA: Breaking low-rank bottleneck of LoRA with nonlinear mapping. In *Advances in Neural Information Processing Systems*, 2025.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

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

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

Angeliki Giannou, Shashank Rajput, Jy-Yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pp. 11398–11442, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2025.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. LoRA+: Efficient low rank adaptation of large models. In *International Conference on Machine Learning*, 2024.

Lingfeng He, De Cheng, Huaijie Wang, Xi Yang, Nannan Wang, and Xinbo Gao. Task-driven subspace decomposition for knowledge sharing and isolation in LoRA-based continual learning. In *International Conference on Machine Learning*, 2026.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. DeBERTaV3: Improving DeBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*, 2021.

Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. EuroSAT: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019.

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*, 2021.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

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.

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

Damjan Kalajdzievski. A rank stabilization scaling factor for fine-tuning with LoRA. *arXiv preprint arXiv:2312.03732*, 2023.

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M. Asano. VeRA: Vector-based random matrix adaptation. In *International Conference on Learning Representations*, 2024.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *IEEE International Conference on Computer Vision Workshops*, pp. 554–561, 2013.

Jiacheng Li, Jianchao Tan, Zhidong Yang, Feiye Huo, Yerui Sun, Yuchen Xie, and Xunliang Cai. AFA-LoRA: Enabling non-linear adaptations in LoRA with activation function annealing. *arXiv preprint arXiv:2512.22455*, 2025a.

Shiwei Li, Xiandi Luo, Haozhao Wang, Xing Tang, Ziqiang Cui, Dugang Liu, Yuhua Li, Xiuqiang He, and Ruixuan Li. Beyond higher rank: Token-wise input-output projections for efficient low-rank adaptation. In *Advances in Neural Information Processing Systems*, 2025b.

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.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Haodong Lu, Chongyang Zhao, Minhui Xue, Lina Yao, Kristen Moore, and Dong Gong. Little by little: Continual learning via incremental mixture of rank-1 associative memory experts. In *International Conference on Machine Learning*, 2026.

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.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Hayden Prairie, Zachary Novack, Taylor Berg-Kirkpatrick, and Daniel Y. Fu. Parcae: Scaling laws for stable looped language models. *arXiv preprint arXiv:2604.12946*, 2026.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pp. 8748–8763, 2021.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling. In *Advances in Neural Information Processing Systems*, 2022.

Raghav Singhal, Kaustubh Ponkshe, Rohit Vartak, and Praneeth Vepakomma. ABBA-Adapters: Efficient and expressive fine-tuning of foundation models. In *International Conference on Learning Representations*, 2026.

Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012.

Chunlin Tian, Zhan Shi, Zhijiang Guo, Li Li, and Chengzhong Xu. HydraLoRA: An asymmetric LoRA architecture for efficient fine-tuning. In *Advances in Neural Information Processing Systems*, 2024.

Zichen Tian, Antoine Ledent, and Qianru Sun. Scalable multi-task low-rank model adaptation. In *International Conference on Learning Representations*, 2026.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*, 2019.

Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *arXiv preprint arXiv:2408.15664*, 2024a.

Shaowen Wang, Linxi Yu, and Jian Li. LoRA-GA: Low-rank adaptation with gradient approximation. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b.

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.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.

Taiqiang Wu, Jiahao Wang, Zhe Zhao, and Ngai Wong. Mixture-of-subspaces in low-rank adaptation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7880–7899, 2024.

Yize Wu, Ke Gao, Ling Li, and Yanjun Wu. Stable-LoRA: Stabilizing feature learning of low-rank adaptation. In *International Conference on Learning Representations*, 2026.

Jianxiong Xiao, James Hays, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. SUN database: Large-scale scene recognition from abbey to zoo. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3485–3492, 2010.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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. In *International Conference on Machine Learning*, 2026.

Jingqi Ye, Haonan He, Minglei Li, Fujun Han, Tao Chen, and Peng Ye. Gradient intrinsic dimensionality alignment: Narrowing the gap between low-rank adaptation and full fine-tuning. In *International Conference on Learning Representations*, 2026.

Mingqian Yu, Wenpeng Zhang, and Peilin Zhao. T-LoopFormer: Token-level elastic-depth looped transformers for latent reasoning with dynamic routing. *arXiv preprint arXiv:2609.15160*, 2026.

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

Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *IEEE/CVF International Conference on Computer Vision*, pp. 11975–11986, 2023.

Feiyu Zhang, Liangzhi Li, Junhao Chen, Zhouqiang Jiang, Bowen Wang, and Yiming Qian. IncreLoRA: Incremental parameter allocation method for parameter-efficient fine-tuning. *arXiv preprint arXiv:2308.12043*, 2023a.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. LoRA-FA: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303*, 2023b.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*, 2023c.

Qingyue Zhang, Chang Chu, Tianren Peng, Qi Li, Xiangyang Luo, Zhihao Jiang, and Shao-Lun Huang. LoRA-DA: Data-aware initialization for low-rank adaptation via asymptotic analysis. In *International Conference on Machine Learning*, 2026a.

Yilang Zhang, Xiaodong Yang, Yiwei Cai, and Georgios B. Giannakis. ScaLoRA: Optimally scaled low-rank adaptation for efficient high-rank fine-tuning. In *International Conference on Machine Learning*, 2026b.

A EXPERIMENTS

A.1 NATURAL LANGUAGE UNDERSTANDING

Datasets. Our evaluation is conducted on the GLUE benchmark (Wang et al., 2019), a widely adopted suite for measuring general natural language understanding across a broad range of sentence-level and sentence-pair classification tasks. We consider seven GLUE tasks, including CoLA, MNLI, MRPC, QNLI, QQP, SST-2, and STS-B.

Training and Evaluation. For training, we follow Wang et al. (2024b) and employ AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and zero weight decay, together with a cosine learning-rate schedule and a 3% warmup period. Each task is trained separately using a batch size of 32 and a maximum sequence length of 128. The peak learning rate for the adapters is set to 10^{-3} for AdaLoRA and 10^{-4} for all other methods to achieve meaningful comparisons. The task-specific classification head is fully fine-tuned at a learning rate of 10^{-4}. Performance is evaluated using accuracy for all tasks except MRPC, QQP, and STS-B. For MRPC and QQP, we report the mean of accuracy and F1 score, whereas for STS-B, we use the mean of Pearson and Spearman correlation coefficients.

A.2 NATURAL LANGUAGE REASONING

Datasets. We evaluate the models across a diverse set of benchmarks, including ARC-Challenge (Clark et al., 2018), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2020), MMLU (Hendrycks et al., 2021), and GSM8K (Cobbe et al., 2021). For code generation, models are trained on CodeAlpaca-20K (Chaudhary, 2023) and evaluated on HumanEval (Chen et al., 2021). These benchmarks cover a broad range of capabilities, including commonsense reasoning, reading comprehension, general knowledge understanding, mathematical reasoning, and code generation.

Training and Evaluation. We separately fine-tune on each benchmark with AdamW, using $\beta_1 = 0.9$, $\beta_2 = 0.999$, no weight decay, and a linear learning-rate schedule with 3% warmup. We use a batch size of 16 throughout. AdaLoRA is trained with a peak learning rate of 5×10^{-4}, whereas all other approaches use 5×10^{-5} for meaningful comparisons. The input length is capped at 128 tokens for ARC-Challenge, HellaSwag, and MMLU, 256 tokens for BoolQ and WinoGrande, and 512 tokens for GSM8K and CodeAlpaca-20K. Evaluation is performed with `lm-evaluation-harness` (Gao et al., 2024). We report length-normalized accuracy on ARC-Challenge and HellaSwag, exact-match accuracy on GSM8K, pass@1 on HumanEval, and standard accuracy on the remaining benchmarks.

A.3 IMAGE CLASSIFICATION

Datasets. We evaluate the models on a diverse set of image classification benchmarks, including Stanford Cars (Krause et al., 2013), DTD (Cimpoi et al., 2014), EuroSAT (Helber et al., 2019), GTSRB (Stallkamp et al., 2012), RESISC45 (Cheng et al., 2017), SUN397 (Xiao et al., 2010), and SVHN (Netzer et al., 2011). These datasets cover a broad range of visual recognition settings.

Training and Evaluation. We fine-tune on individual tasks using AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.999$, zero weight decay, and a cosine learning-rate schedule with 3% warmup. Adapters are applied only to the linear modules of the vision transformer. We use a batch size of 64 and set the peak learning rate to 10^{-3} for AdaLoRA and 10^{-4} for all other methods. Predictions are obtained using the prompt template "`a photo of a class`".

A.4 ANALYSIS

Across-Seed Variability. Tabs. 7, 9 and 10 report the standard deviations corresponding to the quantitative comparisons on NLU, NLR, and IC, respectively. The observed variation is consistently small relative to the performance improvements delivered by MoLR. This suggests that the gains of MoLR are stable across repeated runs and are unlikely to arise from experimental variability.

Robustness. Tabs. 11 and 12 extend the robustness analysis in Fig. 3 to representative LoRA variants. MoLR achieves the highest average performance across all evaluated ranks and target-module

Table 7: **Standard deviations for the quantitative comparison of LoRA variants on NLU.** We report the standard deviations corresponding to the results in Tab. 2.

Model	Method	CoLA	MNLI	MRPC	QNLI	QQP	SST-2	STSB	Avg.
RoBERTa-B	LoRA	0.58	0.12	0.39	0.22	0.05	0.40	0.48	0.08
	DoRA	0.25	0.81	0.16	0.20	0.09	0.29	0.55	0.16
	AdaLoRA	0.59	0.17	0.18	0.32	0.01	0.31	0.13	0.12
	TopLoRA	0.20	0.25	0.87	0.34	0.06	0.43	0.02	0.07
	HydraLoRA	0.72	0.06	0.85	0.34	0.09	0.13	0.12	0.13
	MoSLoRA	0.64	0.20	0.33	0.12	0.09	0.40	0.80	0.19
	AuroRA	0.25	0.12	0.28	0.14	0.14	0.50	0.53	0.12
	MoLR	0.33	0.12	0.47	0.33	0.12	0.63	0.53	0.05
DeBERTa-v3-L	LoRA	0.31	0.14	0.10	0.05	0.12	0.41	0.49	0.10
	DoRA	0.22	0.13	0.10	0.14	0.04	0.17	0.72	0.12
	AdaLoRA	0.06	0.09	0.84	0.05	0.10	0.17	0.49	0.15
	TopLoRA	0.33	0.15	0.97	0.45	0.06	0.35	0.58	0.14
	HydraLoRA	0.72	0.17	0.87	0.15	0.01	0.07	0.72	0.27
	MoSLoRA	0.44	0.22	0.72	0.05	0.05	0.23	0.68	0.17
	AuroRA	0.33	0.04	0.17	0.14	0.04	0.18	0.77	0.07
	MoLR	0.40	0.03	0.08	0.09	0.06	0.17	0.37	0.07

configurations, demonstrating that its gains persist across a broad range of adaptation budgets. Notably, MoLR at lower ranks outperforms competing variants at higher ranks, including methods that already incorporate explicit latent-space computation. This indicates that adaptively refining a compact latent representation can yield greater expressivity than simply increasing its dimensionality, and further suggests that scaling latent-space computation constitutes a complementary axis for increasing adaptation capacity beyond the conventional strategy of expanding the latent rank.

Computational Overhead. We evaluate computational efficiency on a single NVIDIA RTX A6000 using Llama-3.2-3B as the backbone and an adapter rank of $r = 8$, consistent with the setup in Sec. 4.2. Training efficiency is assessed with a batch size of 16 and a sequence length of 128 by recording the average step latency and peak CUDA memory across 50 synchronized optimization steps following 10 warm-up steps. For inference, we similarly measure per-token latency and peak CUDA memory over 50 synchronized next-token forward passes after 10 warm-up batches. Whenever possible, adapter parameters are merged into the backbone weights before inference.

Compatibility. We evaluate the generalizability of adaptive latent recursion by integrating it into three existing LoRA variants: VeRA (Kopiczko et al., 2024), LoRA-FA (Zhang et al., 2023b), and AuroRA (Dong et al., 2025). These methods span diverse latent-space constructions: VeRA employs frozen shared matrices, LoRA-FA freezes the down-projection, and AuroRA learns nonlinear latent transformations without frozen projections. Tab. 8 shows that adaptive latent recursion consistently improves all three variants across all tasks. These gains demonstrate that adaptive latent recursion generalizes across different LoRA formulations, enhancing adaptation expressiveness while preserving parameter efficiency.

Table 8: **Compatibility with existing LoRA variants.** We report average performance (↑) on diverse downstream tasks. #Param. denotes the number of trainable parameters. * shows that the variant is augmented with adaptive latent recursion. **Better** results are highlighted.

Method	NLU		NLR		CV	
	#Param.	Avg.	#Param.	Avg.	#Param.	Avg.
VeRA	0.10M	77.00	0.82M	52.66	0.10M	56.08
VeRA*	0.20M	**81.02**	0.99M	**54.70**	0.20M	**67.14**
LoRA-FA	0.66M	85.08	6.19M	59.05	0.66M	85.29
LoRA-FA*	0.67M	**85.37**	6.21M	**59.29**	0.67M	**85.46**
AuroRA	1.34M	84.94	12.18M	59.71	1.34M	82.26
AuroRA*	1.34M	**85.88**	12.18M	**60.18**	1.34M	**82.86**

B Discussion

In this section, we discuss MoLR from several perspectives, including its empirical behavior and broader implications, current limitations, and promising directions for future work.

Table 9: **Standard deviations for the quantitative comparison of LoRA variants on NLR.** We report the standard deviations corresponding to the results in Tab. 3.

Model	Method	ARC-C	BoolQ	GSM8K	HS	HE	MMLU	WG	Avg.
LLaMA-3.2-3B	LoRA	0.13	0.25	0.33	0.12	1.27	0.31	0.32	0.23
	DoRA	0.22	0.42	0.40	0.06	1.96	0.24	0.59	0.21
	AdaLoRA	0.20	0.22	0.31	0.05	0.35	0.22	0.37	0.02
	TopLoRA	0.40	0.23	0.31	0.05	0.35	0.15	0.71	0.12
	HydraLoRA	0.55	0.36	1.03	0.09	0.70	0.25	0.44	0.17
	MoSLoRA	0.24	0.22	0.40	0.13	0.93	0.23	0.09	0.14
	AuroRA	0.46	0.12	1.16	0.09	1.27	0.39	0.40	0.13
	MoLR	0.22	0.29	0.35	0.12	1.83	0.34	0.72	0.26
Qwen-3-8B	LoRA	1.69	0.22	0.50	0.18	2.14	0.15	0.39	0.10
	DoRA	1.83	0.18	0.33	0.01	2.66	0.13	0.55	0.32
	AdaLoRA	1.72	0.17	0.31	0.10	2.19	0.39	0.25	0.41
	TopLoRA	1.54	0.14	0.42	0.13	1.06	0.11	0.59	0.26
	HydraLoRA	1.74	0.18	0.62	0.06	1.96	0.14	0.09	0.09
	MoSLoRA	1.87	0.07	0.32	0.12	3.39	0.16	0.18	0.49
	AuroRA	1.43	0.10	0.53	0.05	0.35	0.17	0.18	0.10
	MoLR	0.33	0.05	0.17	0.09	1.17	0.23	0.27	0.43

Table 10: **Standard deviations for the quantitative comparison of LoRA variants on IC.** We report the standard deviations corresponding to the results in Tab. 4.

Model	Method	Cars	DTD	ESAT	GTSRB	RE45	SUN397	SVHN	Avg.
CLIP-ViT-B	LoRA	0.28	0.31	0.07	0.25	0.26	0.05	0.09	0.08
	DoRA	0.27	0.20	0.13	0.24	0.25	0.07	0.06	0.13
	AdaLoRA	0.15	0.16	0.09	0.03	0.15	0.15	0.07	0.01
	TopLoRA	0.38	0.52	0.10	0.10	0.08	0.17	0.09	0.15
	HydraLoRA	0.27	0.58	0.05	0.06	0.01	0.10	0.08	0.07
	MoSLoRA	0.35	0.23	0.10	0.12	0.23	0.12	0.05	0.04
	AuroRA	0.26	0.76	0.14	0.30	0.24	0.19	0.04	0.14
	MoLR	0.09	0.27	0.07	0.08	0.25	0.05	0.05	0.05
SigLIP-ViT-L	LoRA	0.04	0.13	0.22	0.07	0.03	0.13	0.02	0.01
	DoRA	0.05	0.03	0.10	0.11	0.06	0.05	0.06	0.03
	AdaLoRA	0.09	0.06	0.17	0.10	0.12	0.04	0.09	0.02
	TopLoRA	0.04	0.25	0.05	0.09	0.08	0.05	0.02	0.06
	HydraLoRA	0.06	0.03	0.08	0.04	0.15	0.05	0.03	0.01
	MoSLoRA	0.07	0.06	0.11	0.02	0.19	0.05	0.05	0.01
	AuroRA	0.08	0.19	0.06	0.03	0.12	0.10	0.05	0.03
	MoLR	0.06	0.18	0.10	0.08	0.12	0.14	0.06	0.08

Exact Expansion of Adaptive Latent Recursion For an input that executes $D(\boldsymbol{x})$ recursion steps, expanding Eq. (4) gives

$$\boldsymbol{z}_{D(\boldsymbol{x})} = \left[\prod_{t=0}^{D(\boldsymbol{x})-1} (\boldsymbol{I}_r + s_t \boldsymbol{W}_{\text{latent}})\right] \boldsymbol{z}_0 + \sum_{k=0}^{D(\boldsymbol{x})-1} \left[\prod_{j=k+1}^{D(\boldsymbol{x})-1} (\boldsymbol{I}_r + s_j \boldsymbol{W}_{\text{latent}})\right] s_k \boldsymbol{W}_{\text{latent}} \phi(k). \quad (11)$$

The exact expansion shows that a fixed step encoding provides a stage signal whose effect is shaped by subsequent shared updates and state-dependent routing. This may explain why learning separate step encodings is unnecessary in this setting, as the shared transformation and router can learn how to use a deterministic signal. Consistent with this interpretation, fixed sinusoidal conditioning has the highest average result among the parameter-shared conditioning variants as shown in Tab. 6.

Threshold-based Routing. As shown in Eq. (4), continuation is a discrete decision, and we do not propagate surrogate gradients through its binary indicator. For methodological simplicity, we separate task-driven router learning from load balancing. Specifically, the router receives task-loss gradients through the continuous score s_t that scales executed updates, and the threshold b_t is ad-

Table 11: **Quantitative comparison of LoRA variants under different rank capacities on NLR.** We sweep rank capacities and report accuracy (↑) for all other tasks and Pass@1 (↑) for HumanEval. The **best** results are highlighted. HS = HellaSwag, HE = HumanEval, WG = WinoGrande.

Rank Capacity	Method	ARC-C	BoolQ	GSM8K	HS	HE	MMLU	WG	Avg.
$r = 8$	LoRA	49.46	88.20	34.49	78.99	23.98	55.73	83.64	59.21
	TopLoRA	50.97	88.84	39.83	79.42	24.19	56.25	83.58	60.44
	MoSLoRA	52.16	89.33	38.06	78.78	22.36	54.46	83.71	59.84
	AuroRA	52.64	88.91	35.51	79.05	24.80	53.34	83.74	59.71
	MoLR	54.69	89.52	40.18	79.11	25.61	55.48	83.90	**61.21**
$r = 16$	LoRA	50.99	88.70	35.66	79.16	24.80	55.87	84.16	59.91
	TopLoRA	52.42	89.36	41.72	79.54	23.37	56.04	83.14	60.80
	MoSLoRA	53.10	89.50	39.47	78.97	25.00	55.38	83.53	60.71
	AuroRA	52.30	88.79	37.55	79.10	22.97	52.85	84.03	59.66
	MoLR	54.24	89.52	40.66	79.32	25.61	55.77	83.98	**61.30**
$r = 32$	LoRA	52.22	89.00	37.30	79.24	24.19	56.25	83.90	60.30
	TopLoRA	53.70	89.33	41.92	79.36	22.97	55.99	83.48	60.96
	MoSLoRA	53.24	89.37	38.99	79.23	24.59	55.85	84.53	60.83
	AuroRA	46.64	85.49	37.55	78.79	24.80	50.27	83.29	58.12
	MoLR	54.61	89.43	41.47	79.31	25.00	56.11	84.40	**61.47**

Table 12: **Quantitative comparison of LoRA-based methods under different target-module sets on NLR.** We vary the target-module set and report accuracy (↑) for all other tasks and Pass@1 (↑) for HumanEval. `attn` includes $\{\boldsymbol{W}_{\mathrm{Q}}, \boldsymbol{W}_{\mathrm{K}}, \boldsymbol{W}_{\mathrm{V}}, \boldsymbol{W}_{\mathrm{O}}\}$, `mlp` includes $\{\boldsymbol{W}_{\mathrm{Gate}}, \boldsymbol{W}_{\mathrm{Up}}, \boldsymbol{W}_{\mathrm{Down}}\}$, and `all` includes their union. The **best** results are highlighted. HS = HellaSwag, HE = HumanEval, WG = WinoGrande.

Target Module	Method	ARC-C	BoolQ	GSM8K	HS	HE	MMLU	WG	Avg.
`attn`	LoRA	47.52	87.24	32.80	78.51	25.81	55.08	82.32	58.47
	TopLoRA	49.63	88.43	36.11	79.09	23.58	55.22	82.58	59.24
	MoSLoRA	49.89	88.30	34.42	78.56	23.37	54.46	81.53	58.65
	AuroRA	50.17	88.12	33.53	78.65	25.61	53.32	82.45	58.84
	MoLR	51.93	88.99	36.74	78.68	23.98	54.82	82.24	**59.63**
`mlp`	LoRA	47.84	87.63	33.84	78.92	23.78	55.46	83.39	58.70
	TopLoRA	50.71	88.61	38.16	79.31	23.78	55.93	83.92	60.06
	MoSLoRA	51.25	88.47	35.73	78.73	24.39	55.26	83.74	59.65
	AuroRA	51.37	88.23	35.00	78.93	23.37	54.00	82.93	59.12
	MoLR	52.93	88.91	38.44	78.99	24.59	55.65	83.66	**60.45**
`all`	LoRA	49.46	88.20	34.49	78.99	23.98	55.73	83.64	59.21
	TopLoRA	50.97	88.84	39.83	79.42	24.19	56.25	83.58	60.44
	MoSLoRA	52.16	89.33	38.06	78.78	22.36	54.46	83.71	59.84
	AuroRA	52.64	88.91	35.51	79.05	24.80	53.34	83.74	59.71
	MoLR	54.69	89.52	40.18	79.11	25.61	55.48	83.90	**61.21**

justed from observed continuation rates by Eq. (9). The threshold determines whether an update executes but does not scale its magnitude once selected, so balancing adds no auxiliary gradient to the task objective. Whether surrogate gradients through the discrete continuation decision are necessary or beneficial, and whether they would blur the separation between task-driven router learning and load-driven threshold adjustment, remain open questions for future work.

Input-Dependent Computation Allocation. MoLR allocates latent-space computation according to the input, allowing different inputs to execute different numbers of recursion steps. This complements rank-allocation methods, which adjust the rank assigned to adapted weight matrices according to their importance (Zhang et al., 2023a;c; Ding et al., 2023). While these methods redistribute adaptation capacity across modules, increasing the total available rank still requires additional trainable rank components. In contrast, MoLR varies the computation applied to each input through a shared latent transformation, without trainable-parameter growth proportional to recursion depth.

Test-Time Scaling. For a fixed MoLR checkpoint trained with maximum recursion depth T, varying $T_{\text{test}} \leq T$ adjusts the inference-time adapter compute budget without requiring additional training. As shown in Fig. 3c, performance improves as T_{test} increases within each evaluated training horizon. This equips a frozen pre-trained backbone with a bounded test-time compute knob through a lightweight adapter, without requiring the recurrent backbone itself to be pre-trained from scratch (Geiping et al., 2025; Jeddi et al., 2026; Yu et al., 2026). Consequently, a single adapted checkpoint can operate under different compute budgets, using a smaller depth cap when latency is constrained and allowing additional latent refinement when more computation is available. Token-level halting further enables this budget to be allocated non-uniformly across inputs, potentially concentrating computation where additional refinement is most beneficial. More broadly, this mechanism may make adaptive inference-time computation accessible to existing language and vision models through parameter-efficient adaptation. It also provides a potentially complementary scaling axis to chain-of-thought prompting (Wei et al., 2022), where explicit multi-round reasoning can extend the sequence of steps and adaptive latent recursion can vary the computation applied within each step. We leave these open questions for future work.

Computational Overhead. Due to input-dependent routing and recursion, MoLR cannot be merged into the pre-trained weights. As shown in Tab. 5, it achieves the highest average performance with modest training overhead, while maintaining inference latency and memory comparable to other non-mergeable LoRA variants. Further efficiency improvements through algorithmic and hardware optimizations remain future work.