TRAINING-FREE LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce **training-free looped transformers**, in which a lightweight inference-time wrapper loops a contiguous mid-stack block of layers of a frozen checkpoint without additional fine-tuning, continued training, or architectural changes. Unlike prior looped transformer methods that train with the looped structure end-to-end, we retrofit recurrence onto pretrained models at test time. We show that naive block reapplication usually degrades performance, highlighting the importance of the loop application strategy. Motivated by *viewing a pre-norm transformer block as a forward Euler step on an ODE*, we instead treat looping as a refinement of the same approximation, replacing one large update with smaller damped sub-steps. Across six dense, sparse mixture-of-experts (MoE), and multi-head latent attention (MLA) MoE model families, our training-free looping strategy yields consistent performance improvements over the original frozen checkpoints.

1 INTRODUCTION

Looped transformers (Giannou et al., 2023; Fan et al., 2025), Universal Transformers (Dehghani et al., 2019; Lan et al., 2020), and Deep Equilibrium models (Bai et al., 2019; 2020) all incorporate *recurrence* into the architecture (Xu & Sato, 2025a; Saunshi et al., 2025; Merrill & Sabharwal, 2025; Gong et al., 2025) and weights (Geiping et al., 2025a; Zhu et al., 2025; Prairie et al., 2026; Wu et al., 2025). Layers are tied across loop iterations and optimized accordingly, so the recurrence structure is inseparable from the trained parameters. As a consequence, looping cannot be applied to a model that was not trained with this methodology in mind, which is the case for almost all publicly released language-model checkpoints. This naturally gives rise to the core question of this paper:

> *Can we loop a frozen, off-the-shelf checkpoint directly at inference time, with **no fine-tuning**, **no continued training**, **no auxiliary parameters**, and **no architectural changes**?*

Concretely, modern open-weight LLMs, such as Qwen3 (Qwen Team, 2025), Llama-3.2 (Llama Team, 2024), Moonlight (Liu et al., 2025), and DeepSeek-V2-Lite (DeepSeek-AI, 2024), are the endpoints of a multi-stage pipeline that typically includes continued pretraining, supervised fine-tuning, and one or more rounds of preference-based post-training (Guo et al., 2025). We take whatever weights the model authors release and apply the loop wrapper at inference, with no further weight updates of any kind.

Several converging strands of evidence point towards a positive answer to this question. Men et al. (2025) show that whole mid-layer blocks of released transformer LMs can be deleted with minimal loss; Lad et al. (2025) report that mid-layers can be skipped, swapped, or repeated without catastrophic degradation (Li et al., 2025; Heakl et al., 2026); and Belrose et al. (2023) demonstrate that intermediate-layer logits already encode much of the final prediction (Takase & Kiyono, 2023; Reid et al., 2021). The prevailing interpretation of these results is one of *compressibility*—middle layers are redundant, so we can remove them—but this ignores a complementary perspective: the same redundancy that makes a mid-layer safe to delete makes it safe to *re-apply*. This insight admits a numerical analysis interpretation that we develop in Section 2.3: each pre-norm transformer layer is exactly one forward Euler step at $h{=}1$ on a per-block residual ODE, so *re-applying the block at inference is, geometrically, a finer integration of the ODE the network already approximates* (Li et al., 2022; Bai et al., 2019; 2020). In particular, K sub-steps of size $h{=}1/K$ better approximate the same $t{=}1$ endpoint that the unmodified network was trained to deliver.

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

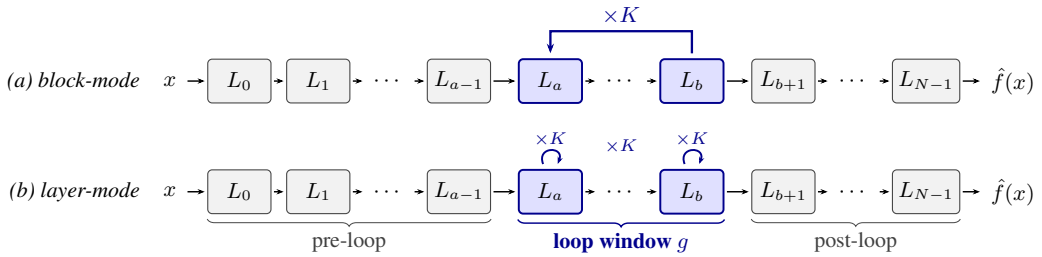


Figure 1: **Training-free looped transformer wrapper, two iteration modes.** A frozen checkpoint is augmented at inference by re-applying a contiguous mid-block $g = L_b \circ \cdots \circ L_a$ for K iterations before resuming the post-loop layers. No weights are changed and no new parameters are introduced; the cost is $(b - a + 1)(K - 1)$ extra forward passes through the loop window. **(a) Block-mode** iterates the whole window K times, $(L_b \circ \cdots \circ L_a)^K$. **(b) Layer-mode** iterates each window layer K times before passing on, $L_b^K \circ \cdots \circ L_a^K$, which is the safer default on Mixture-of-Experts backbones because it pins per-layer expert routing across iterations (Section 2.2). Section 2.3 interprets $g^{(K)}$ as sub-stepping the residual ODE the network already integrates with single-step forward Euler.

We show that this geometric motivation translates into measurable gains on modern checkpoints across *six model families*—Qwen3 (0.6B/1.7B/4B base & instruct and a 30B-A3B mixture-of-experts (MoE) model; Qwen Team, 2025), Qwen1.5-MoE-A2.7B, Llama-3.2 (1B/3B; Llama Team, 2024), Moonlight (Liu et al., 2025), and DeepSeek-V2-Lite (DeepSeek-AI, 2024)—and *45 (model, benchmark) cells* of training-free loop evaluation. Looped evaluation yields its strongest gains on knowledge-heavy multiple-choice (MC), reasoning, and coding benchmarks.

These improvements arise without any parameter updates, additional supervision, or benchmark-specific tuning, suggesting that repeated application of existing transformer blocks can expose latent inference-time computation. We further compare forward Euler integration with higher-order ODE solvers and fixed-point accelerators, including Anderson acceleration (Walker & Ni, 2011), heavy-ball momentum (Polyak, 1964), Aitken extrapolation (Aitken, 1927), and Runge–Kutta (RK) updates (Butcher, 2016).

Our main contributions are as follows:

1. **A training-free loop wrapper**, fully specified, with block and layer iteration modes and a family of loop-iteration strategies drawn from well-known numerical integration methods for the ODE that transformers implicitly approximate.
2. **Cross-architecture validation** on six model families and 45 (model, benchmark) cells under a single *out-of-the-box recipe* (K-stage Runge–Kutta with K=6 and β=0.5 at the middle four layers; block-mode for dense and layer-mode with a frozen router for MoE; FIRST KV cache, Appendix D) with *no per-cell hyperparameter tuning of any kind*.
3. **Layer-mode iteration for MoE**: $L_b^K \circ \cdots \circ L_a^K$ rather than $(L_b \circ \cdots \circ L_a)^K$ is necessary for Mixture-of-Experts checkpoints (Csordás et al., 2024; Bae et al., 2025a;b), where block-mode causes expert routing to thrash between iterations.

2 Training-free looped transformers

2.1 Loop wrapper

Let $f = L_{N-1} \circ \cdots \circ L_0$ denote a pretrained transformer with N decoder layers (Vaswani et al., 2017; Qwen Team, 2025; Llama Team, 2024; DeepSeek-AI, 2024; Liu et al., 2025), where L_i maps a residual stream of shape $\mathbb{R}^{T\times d}$ to itself. Choose a contiguous *loop window* indexed by $[a, b]$ with $0 \le a \le b \le N - 1$ and a loop count $K \ge 1$. The window induces an operator

$$g := L_b \circ L_{b-1} \circ \cdots \circ L_a, \qquad g : \mathbb{R}^{T\times d} \to \mathbb{R}^{T\times d}. \tag{1}$$

The looped wrapper splits the network into pre-loop layers $0, \ldots, a-1$, the looped middle, and post-loop layers $b+1, \ldots, N-1$:

$$\hat{f}(x) = \underbrace{(L_{N-1} \circ \cdots \circ L_{b+1})}_{\text{post-loop}} \circ g^{(K)} \circ \underbrace{(L_{a-1} \circ \cdots \circ L_0)}_{\text{pre-loop}}(x), \tag{2}$$

where $g^{(K)} : \mathbb{R}^{T \times d} \to \mathbb{R}^{T \times d}$ is a K-step iteration of g. When $a = 0$ or $b = N-1$, the corresponding boundary composition is the identity. The map $g^{(K)}$ depends on an *iteration mode* (Section 2.2) and a *loop strategy* (Section 2.3).

2.2 Block-mode versus layer-mode iteration

Given the window operator $g = L_b \circ \cdots \circ L_a$ in (1), there are two natural realizations of $g^{(K)}$,

$$\text{block-mode:} \quad g^{(K)}_{\text{blk}}(x) = (L_b \circ \cdots \circ L_a)^K(x), \tag{3}$$

$$\text{layer-mode:} \quad g^{(K)}_{\text{lyr}}(x) = L_b^K \circ L_{b-1}^K \circ \cdots \circ L_a^K(x). \tag{4}$$

In other words, block-mode iterates the entire window as one unit, whereas layer-mode iterates each layer before passing to the next (Figure 1). For dense models (Bae et al., 2025a;b), the two yield broadly similar quality. For mixture-of-experts (MoE) layers (DeepSeek-AI, 2024; Liu et al., 2025; Csordás et al., 2024), however, block-mode becomes unstable, since at iteration i the gating network of every MoE layer in the window sees a slightly perturbed hidden state and routes to a different subset of experts than at iteration $i-1$, and the accumulated routing-induced noise eventually overpowers the intended refinement. On the other hand, layer-mode iteration computes the gating decision once and applies the same expert mixture K times (Csordás et al., 2024), avoiding this failure mode and making it the correct default for MoE backbones.

2.3 Loop strategies

We first motivate several loop strategies with a structural observation. A standard pre-norm transformer layer L (Vaswani et al., 2017) implements

$$L(x) = x + \text{Attn}(\text{LN}_1(x)) + \text{MLP}(\text{LN}_2(x + \text{Attn}(\text{LN}_1(x)))). \tag{5}$$

The right-hand side is the layer's input plus a residual update. Generalizing this directly to the loop window operator g, which may be a single layer $g = L_i$ (the layer-mode case) or a contiguous composition $g = L_b \circ \cdots \circ L_a$ (the block-mode case), we define the *window residual field*

$$F_g(x) := g(x) - x. \tag{6}$$

By construction, $g(x) = x + F_g(x)$, which is exactly a forward Euler step with step size $h{=}1$ (Li et al., 2022)

$$x_1 = x_0 + h \cdot F_g(x_0) \tag{7}$$

on the autonomous ODE (Bai et al., 2019; 2020)

$$\dot{x} = F_g(x). \tag{8}$$

This holds regardless of whether the window contains one layer or many, and the two modes differ only in what F_g unfolds to. In layer-mode, F_g is just the single-layer residual field

$$F_g(x) = \text{Attn}_i(\text{LN}_1^i(x)) + \text{MLP}_i(\text{LN}_2^i(\cdot)).$$

In block-mode, telescoping the unrolled chain (proof in Appendix C) gives

$$F_g(x) = \sum_{i=a}^{b} F_{L_i}(y_i(x)), \qquad y_a(x) := x, \quad y_{i+1}(x) := y_i(x) + F_{L_i}(y_i(x)), \tag{9}$$

where $F_{L_i}(z) := L_i(z) - z$ is the layer residual field of layer i. In both modes, the post-loop layers $L_{b+1}, \ldots, L_{N-1}$ are trained to receive the approximation of $x(t{=}1)$ on F_g implicitly produced by one application of g and not the trajectory at any other time.

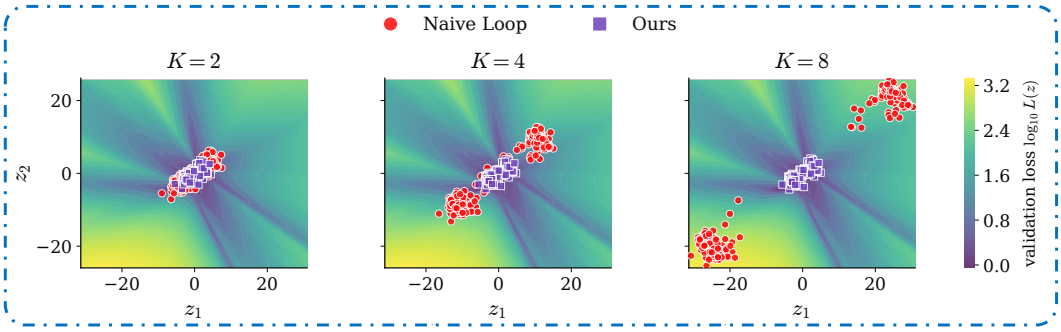


Figure 2: **Damped sub-stepping vs. naive looping.** A tiny MLP $\mathrm{pre}(\mathbb{R}^4 \to \mathbb{R}^2) \to$ block of 3 residual layers $(\mathbb{R}^2 \to \mathbb{R}^2) \to \mathrm{post}(\mathbb{R}^2 \to \mathbb{R}^2)$ is trained end-to-end on a 2-D regression target. Each panel fixes $K \in \{2, 4, 8\}$ and shows, over a 220×220 grid in the post-block hidden state $\mathbf{z}$, the median validation loss $L(\mathbf{z}) = \mathrm{med}_i \, \|\mathrm{post}(\mathbf{z}) - y_i\|^2$ (log scale, colored) together with the test-set scatters obtained by naive looping (red circles) and our damped sub-stepping (purple squares). The trained baseline induces a central low-loss valley (dark purple); our endpoints stay clustered in that valley at every K, while naive K-loop endpoints drift increasingly outward into high-loss regions (yellow) as K increases.

Naively looping g for K rounds, i.e., $x \leftarrow g(x)$ applied K times (Dehghani et al., 2019; Lan et al., 2020; Giannou et al., 2023; Fan et al., 2025; Saunshi et al., 2025; Xu & Sato, 2025a; Geiping et al., 2025a; Zhu et al., 2025; von Oswald et al., 2023; Ahn et al., 2023), is a K-step forward Euler integration of the window residual field with step size $h{=}1$, which approximates $x(t{=}K)$. But the post-loop layers are not trained to receive the trajectory at $t{=}K$, and empirically, naive K-fold looping degrades performance almost universally (Section 3). Figure 2 illustrates the effect of naive looping vs. sub-stepping on a tiny end-to-end trainable network with a 2-D bottleneck, where the input to the post-loop layers can be directly plotted. The damped sub-step endpoints stay clustered in the trained low-loss valley, while the naive K-loop endpoints drift into high-loss regions (Mahankali et al., 2024; Gatmiry et al., 2024; Chen et al., 2025). Numerically, when averaged over the test set, naive $K{=}2$ looping increases the MSE to 2.88 versus the baseline and sub-step values of 0.015 and 0.36, respectively ($\approx 8\times$ gap); at $K{=}8$ this degrades to an MSE of 335 versus the sub-step value of 1.04 ($\approx 320\times$ gap).

The principled goal of $g^{(K)}$ is therefore not to advance integration to $t{=}K$, but to *better approximate the same endpoint* $x(t{=}1)$ that the unmodified network already targets. To address this challenge, classical numerical analysis suggests sub-stepping the same total integration time $[0, 1]$ at finer resolution $h = 1/K$. Performing K Euler steps of size $h{=}1/K$ on (8) gives the damped update

$$x_{k+1} = \left(1 - \frac{1}{K}\right) x_k + \frac{1}{K} g(x_k), \tag{10}$$

which converges to the true $x(t{=}1)$ at order $O(1/K)$ and strictly improves upon the single-shot $h{=}1$ Euler step that the network implicitly implements at every layer. Because equations (6)–(8) are mode-agnostic, this principle applies in both layer-mode (sub-stepping the per-layer field $F_g = F_{L_i}$) and in block-mode (sub-stepping the composite field F_g in (9)).

More generally, we can consider numerical integration strategies beyond the forward Euler method to approximate $x(t{=}1)$. In particular, the damped Euler update (10) can be replaced by the s-stage explicit Runge–Kutta integrator

$$x_1 = x_0 + h \sum_{i=1}^{s} b_i k_i, \tag{11}$$

where the s stages are given by

$$k_1 = F_g(x_0), \qquad k_2 = F_g(x_0 + h(a_{21} k_1)), \qquad \ldots, \qquad k_s = F_g\left(x_0 + h \sum_{j=1}^{s-1} a_{sj} k_j\right), \tag{12}$$

Algorithm 1 **Block-mode** Runge–Kutta forward pass.

Input: x, block range $[a, b]$, number of stages s, Runge–Kutta coefficients $\{a_{ij}\}_{1\le j<i\le s}, \{b_i\}_{i=1}^{s}$

$x \leftarrow \text{PRELOOP}(x)$ ▷ *embedding, pre-block, etc.*
$y_0 \leftarrow x$ ▷ *initial state for the block*
for $i = 1, \dots, s$ **do** ▷ *iterate over the s Runge–Kutta stages*
  $y_i \leftarrow y_0 + \sum_{j<i} a_{ij} k_j$ ▷ *add weighted increments*
  $k_i \leftarrow (L_b \circ \cdots \circ L_a)(y_i) - y_i$ ▷ *residual of the full composed block*
end for
$x \leftarrow y_0 + \sum_{i=1}^{s} b_i k_i$ ▷ *s^{th} order Runge–Kutta update*
return $\text{POSTLOOP}(x)$ ▷ *layers after the loop block, output head*

Algorithm 2 **Layer-mode** Runge–Kutta forward pass.

Input: x, block range $[a, b]$, number of stages s, Runge–Kutta coefficients $\{a_{ij}\}_{1\le j<i\le s}, \{b_i\}_{i=1}^{s}$

$x \leftarrow \text{PRELOOP}(x)$ ▷ *embedding, pre-block, etc.*
for $\ell = a, \dots, b$ **do** ▷ *outer loop over each layer*
  $y_0 \leftarrow x$ ▷ *initial state for every layer*
  for $i = 1, \dots, s$ **do** ▷ *iterate over the s Runge–Kutta stages*
    $y_i \leftarrow y_0 + \sum_{j<i} a_{ij} k_j$ ▷ *add weighted increments*
    $k_i \leftarrow L_\ell(y_i) - y_i$ ▷ *residual of a single layer L_ℓ*
  end for
  $x \leftarrow y_0 + \sum_{i=1}^{s} b_i k_i$ ▷ *s^{th} order Runge–Kutta update*
end for
return $\text{POSTLOOP}(x)$ ▷ *layers after the loop block, output head*

with the coefficients $\{a_{ij}, b_i\}$ specified by a Butcher tableau. Algorithms 1 and 2 summarize this family of methods for block-mode and layer-mode implementations, respectively, where the step size is chosen to be h=1.

As a particularly effective example, we choose a specific Butcher tableau so that the RK output becomes an interpolation between the base output and the K-step damped Euler updates following (10). Concretely, we set

$$h = 1, \quad s = K, \quad \text{and} \quad a_{ij} = \frac{1}{K}\mathbb{I}(j < i), \tag{13}$$

so that the i^{th} stage evaluates F_g at the point

$$y_i = x_0 + \frac{1}{K}\sum_{j=1}^{i-1} k_j = F^{i-1}(x_0), \qquad k_i = F_g(y_i), \tag{14}$$

where $F(x) := x + \frac{1}{K}F_g(x)$ is the damped Euler update (10). We then choose $\beta \in [0, 1]$ and set the output weights as

$$b_1 = \beta + \frac{1-\beta}{K}, \qquad b_i = \frac{1-\beta}{K}, \quad i = 2, \dots, K, \tag{15}$$

which are nonnegative and sum to one. Under these coefficients, the RK output satisfies the identity

$$x_1 = x_0 + \sum_{i=1}^{K} b_i k_i = \beta g(x_0) + (1-\beta)F^K(x_0), \tag{16}$$

which leads to an efficient implementation (Algorithm 3 in Appendix C.3). The proof is provided in Appendix C.

Algorithm 3 can be interpreted as an RK method with a front-loaded quadrature rule controlled by β. In the extreme cases, $\beta = 0$ recovers the K-sub-step forward Euler endpoint, while $\beta = 1$ preserves

the original output $g(x_0)$. For intermediate values of β, the method effectively biases the trajectory toward the trained one-step endpoint by placing greater weight on the initial residual direction.

Beyond Runge–Kutta methods, there exists a wide range of numerical integration schemes that have been extensively studied in the ODE literature (Butcher, 2016). These methods are often designed to address specific properties of the underlying dynamics, and we list several prominent examples in Table 8 in Appendix F. These algorithms, which differ in order, accelerator, and step size choices, form the basis for the strategies that we evaluate.

3 Experiments

This section evaluates the proposed training-free loop wrapper across dense and MoE checkpoints, base and instruction-tuned variants, and both standard multi-head attention (MHA) and multi-head latent attention (MLA) backbones, using a fixed recipe with no per-cell hyperparameter tuning (Section 3.1), unless otherwise stated.

Main findings. Our method delivers the largest and most consistent gains on knowledge-intensive multiple-choice, reasoning, and coding tasks. Across architectures, most cells are positive or neutral, with failures concentrated in small distilled checkpoints on some knowledge-MC tasks (Section 3.5).

Table 1: **Window selection for the looping recipe. (a)** Window-size sweep on Qwen3-1.7B-Base under forward Euler (K=2): n=4 is the sweet spot with a sharp cliff at $n \geq 6$. **(b)** The n=4 choice generalizes across model scales and families. **(c)** Block-mode vs. layer-mode iteration on MoE backbones at the canonical loop window $[13, 16]$ (block K=3 / layer K=2): layer-mode yields +0.5 to +1.7 pp on hard MoE benchmarks. Numbers are Δpp vs. baseline. Bold marks the best window size in (a) and the better variant per row in (b) and (c); "—" indicates not run.

(a) Window-size sweep, Qwen3-1.7B-Base

n	Window	16-task	Δ
1	[14]	55.72	+0.18
2	[13,14]	55.60	+0.06
3	[13,14,15]	55.75	+0.21
4	[12–15]	**56.09**	**+0.55**
6	[11–16]	54.72	−0.82
12	[8–19]	54.91	−0.63
28 (all)	[0–27]	27.81	−27.73

(b) Window size across model scales

Model	n=3 Δ	n=4 Δ
Qwen3-0.6B-Base (Qwen Team, 2025)	**+0.57**	+0.22
Qwen3-1.7B-Base (Qwen Team, 2025)	+0.21	**+0.55**
Qwen3-4B-Base (Qwen Team, 2025)	+0.43	**+0.85**
Llama-3.2-3B-Instruct (Llama Team, 2024)	+0.23	**+0.45**

(c) Block-mode vs. layer-mode (MoE)

Benchmark	block (K=3)	layer (K=3)
Qwen1.5-MoE-A2.7B-Chat (Qwen Team, 2024)		
ARC-Challenge	+0.17	**+0.85**
CSQA	+0.16	**+0.33**
OBQA	**+1.00**	−1.40
MMLU	−1.18	**0.00**
SciQ	**+0.20**	—
GPQA-Main	−1.11	—
Moonlight-16B-A3B-Instruct (Liu et al., 2025)		
ARC-Challenge	−1.45	**+0.51**
CSQA	−0.66	**+0.49**
OBQA	−0.20	**+1.20**
MMLU	**+0.74**	—
SciQ	**-0.50**	−0.60
GPQA-Main	−2.00	**+0.90**

Robustness. MoE models require layer-mode iteration to avoid routing thrash (Table 1). Numerous ablation studies confirm robustness to loop strategy, loop count, and window placement (Section 3.6).

Comparisons and transfer. Compared with naive inference-time looped transformers, our method avoids collapse and performs best across tested cells (Section 3.7, Figure 4). A leakage-free transfer to Qwen3-30B-A3B-Instruct further supports cross-architecture generalization (Table 3).

3.1 Experimental setup

We evaluate six Hugging Face checkpoint families spanning dense and MoE backbones, base and instruction-tuned variants, and standard MHA vs. MLA: Qwen3 dense $\{0.6, 1.7, 4\}$B (base & instruct; Qwen Team, 2025), Qwen3-MoE 30B-A3B-Instruct (`qwen3_moe`; Qwen Team, 2025), Qwen1.5-MoE-A2.7B-Chat (`qwen2_moe`, distilled), Llama-3.2 $\{1, 3\}$B-Instruct (distilled from Llama-3.1-8B/70B; Llama Team, 2024), DeepSeek-V2-Lite-Chat (MLA + 64-expert MoE; DeepSeek-AI, 2024), and Moonlight-16B-A3B-Instruct (`deepseek_v3` family, MLA + MoE; Liu et al.,

Table 2: **Knowledge-heavy MC benchmarks with the loop wrapper applied out-of-the-box to each frozen checkpoint.** *No per-cell hyperparameter tuning*. Accuracy is in %, and Δ is the improvement in pp over the no-loop baseline at the same prompt, computed before rounding. Bold marks gains of at least 2 pp.

(a) Dense backbones

Model	Benchmark	Baseline	Loop (ours)	Δ (pp)
Qwen3-4B-Instruct	MMLU-Pro 5-shot	57.14	**59.79**	**+2.64**
	GPQA-Main 0-shot	33.71	**35.71**	**+2.01**
	CSQA 7-shot	78.87	79.93	+1.06
Llama-3.2-3B-Instruct	GPQA-Main 0-shot	29.91	31.03	+1.12
	MMLU-Pro 5-shot	31.64	32.36	+0.71
	MMLU 0-shot	59.66	60.39	+0.72
Llama-3.2-1B-Instruct	GPQA-Main 0-shot	27.90	29.69	+1.79

(b) MoE backbones

Model	Benchmark	Baseline	Loop (ours)	Δ (pp)
Qwen1.5-MoE-A2.7B-Chat	ARC-C 25-shot	48.29	**50.60**	**+2.30**
	CSQA 7-shot	79.61	81.33	+1.72
	OBQA 0-shot	31.60	33.20	+1.60
Moonlight-16B-A3B-Instruct	MMLU 0-shot	67.86	68.60	+0.74
	OBQA 0-shot	31.60	32.80	+1.20
DeepSeek-V2-Lite-Chat	ARC-C 25-shot	57.94	58.79	+0.85

2025). MC benchmarks include MMLU (Hendrycks et al., 2021), MMLU-Pro (Wang et al., 2024), GPQA-Main (Rein et al., 2024), ARC-Challenge (ARC-C) (Clark et al., 2018), CommonsenseQA (CSQA) (Talmor et al., 2019), and OpenBookQA (OBQA) (Mihaylov et al., 2018); differences are in percentage points (pp).

All multiple-choice evaluations use `lm-evaluation-harness` v0.4.11 (Gao et al., 2023) with `bfloat16` weights and SDPA attention. For every (model, benchmark) cell, we apply a single *out-of-the-box recipe*: K-stage Runge–Kutta (Algorithm 3) with K=6 and β=0.5 at the middle four layers, block-mode for dense backbones and layer-mode with a frozen router for MoE backbones, and the FIRST KV-cache strategy, with no per-cell hyperparameter search over position, K, or strategy. Per-cell variations are limited to the absolute layer indices, which follow mechanically from each architecture's layer count, and the decode mode, which follows from the evaluation type (Appendix D); a fully leakage-free transfer of the recipe *with no per-cell variation of any kind* to a held-out architecture (Qwen3-30B-A3B-Instruct) is reported in Table 3.

3.2 LAYER-MODE FOR MOE

On MoE backbones (DeepSeek-AI, 2024; Liu et al., 2025), default block-mode iteration causes *routing thrash*, in which each block iteration re-evaluates the gating function on slightly perturbed states, accumulating routing-induced rather than representation-induced changes. This can be fixed by switching to layer-mode iteration, as discussed in Section 2.2 and by Csordás et al. (2024) and Bae et al. (2025a). Table 1 compares the two modes on Qwen1.5-MoE and Moonlight, and layer-mode can be seen to flip most negative cells positive, yielding +0.5 to +1.7 pp on hard MoE benchmarks.

3.3 RESULTS ON KNOWLEDGE-HEAVY MC TASKS

Our main experimental results concern *knowledge-heavy multiple-choice tasks where the baseline is below ceiling, evaluated with a lenient extractor.* On this class of cells the loop acts as a *frozen-context knowledge refiner*, and the gains are largest and most robust. Table 2 and Figure 5 in Appendix J report the results for dense and MoE backbones.

The largest gains (+2.0 to +2.6 pp) appear on the hardest tasks (MMLU-Pro (Wang et al., 2024), GPQA-Main (Rein et al., 2024), ARC-Challenge (Clark et al., 2018)) for the strongest MHA backbones. MLA-based MoE models (DeepSeek-V2-Lite, Moonlight) move in the same direction but with 3–4× smaller magnitudes.

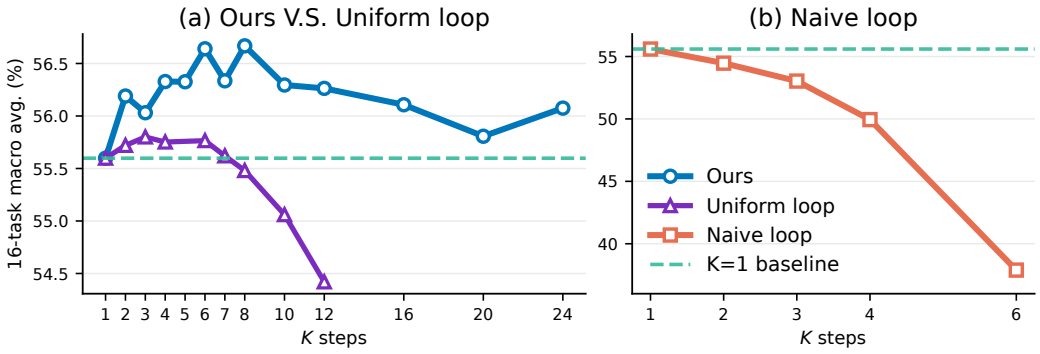


Figure 3: **Effect of loop count K on 16-task macro-average accuracy. (a)** Ours (Algorithm 3) is stable for K up to 24, while the uniform loop (Lys et al., 2026) (Table 8) fails to scale after $K \geq 6$. **(b)** The naive loop variant degrades monotonically, falling to 37.89% at $K=6$ (-17.71 pp).

Table 3: **Qwen3-30B-A3B-Instruct generalization check at $[22, 24]$.** The configuration is fixed in advance, with *no per-cell search or tuning*, i.e. the configuration was committed before any cell was scored, so each cell is a fully leakage-free transfer. Bold marks the better of two committed solver choices ($K=2$ stage Runge–Kutta in Algorithm 3 vs. Heun $K=1$, defined in Table 8). MMLU-flex is MMLU scored with flexible answer extraction.

	Accuracy / exact-match (↑)								PPL (↓)
Method	ARC-Easy	HellaSwag	SciQ	CSQA	TruthfulQA	MMLU-flex	GPQA-Main	SuperGPQA	LAMBADA
Baseline	79.04	77.74	94.80	78.71	34.15	66.67	37.28	31.00	4.12
K-stage RK	**79.38**	**77.93**	**95.00**	**79.85**	**34.64**	**67.46**	37.50	**31.70**	**4.11**
Δ vs. baseline	+0.34	+0.19	+0.20	+1.14	+0.49	+0.79	+0.22	+0.70	−0.01
Heun $K=1$	79.08	77.78	94.90	79.61	34.27	66.67	**37.72**	31.05	4.12
Δ vs. baseline	+0.04	+0.04	+0.10	+0.90	+0.12	0.00	+0.44	+0.05	0.00

3.4 Open-ended math and code reasoning

To test whether the gains extend beyond multiple-choice evaluation, we apply the recipe of Section 3.1 unchanged to Qwen3-30B-A3B (Qwen Team, 2025) on competition mathematics (AIME 2024, 2025, and 2026, and IMO-AnswerBench (Luong et al., 2025)) and on code generation (100 medium and hard Codeforces problems). These tasks require long free-form reasoning and exact final answers or executable programs, so the output space is not restricted to a fixed set of options. Table 4 shows that the looped model improves on all five tasks, by 4.5 to 5.3 pp and by 5.0 pp on average. On each AIME year we report avg@16 over its 30 problems, averaged over 10 base seeds, and the baseline and looped runs share the same seeds, so the comparison is paired. The model runs in thinking mode with Qwen's recommended sampling settings, AIME answers are graded with the AIME scorer of `lm-evaluation-harness`, and IMO-AnswerBench answers with AnswerAutoGrader, the official grader of IMO-Bench. Appendix H gives the full prompt, decoding, and grading protocol.

3.5 Cross-architecture summary

Across the six model families and a uniform knowledge-MC task suite, we score every cell under the fixed recipe of Section 3.1, resulting in 27 positive ($\Delta > +0.3$ pp, 60%) and 12 neutral ($|\Delta| \leq 0.3$ pp, 27%) cells. The remaining negatives concentrate in a single regime, namely *sub-3B distilled checkpoints on knowledge MC* (e.g., Llama-3.2-1B at multiple-choice MMLU (Hendrycks et al., 2021)). The wrapper produces a positive or neutral signal on *87% of cells* across dense, MoE, base, instruct, and distilled checkpoints.

Table 4: **Open-ended math and code reasoning on Qwen3-30B-A3B.** The loop recipe of Section 3.1 is applied unchanged, with no benchmark-specific tuning. IMO-AnsBench denotes IMO-AnswerBench. Numbers are accuracy (%), and Δ is the absolute gain in pp over the no-loop baseline.

Method	AIME 2024	AIME 2025	AIME 2026	IMO-AnsBench	Codeforces	Average
Qwen3-30B-A3B	80.4	70.9	65.0	34.0	44.0	58.9
+ looped (ours)	**84.9**	**75.6**	**70.2**	**39.3**	**49.1**	**63.8**
Δ vs. baseline	+4.5	+4.7	+5.2	+5.3	+5.1	+5.0

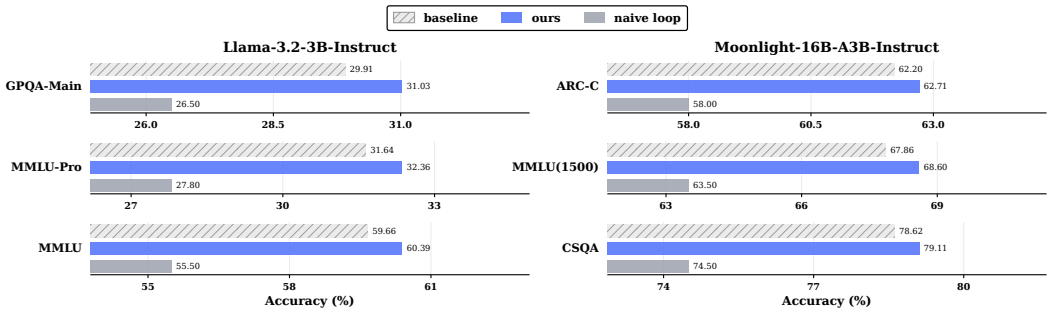


Figure 4: **Comparison with other looped transformer methods on Llama-3.2-3B-Instruct and Moonlight-16B-A3B-Instruct.** On each backbone we report three knowledge-MC benchmarks under three configurations: *baseline* (no loop, original checkpoint), *naive loop* with $K{=}4$, and *ours* (Algorithm 3). MMLU(1500) is a 1,500-question subset of MMLU.

3.6 Ablations

We systematically conducted numerous ablation studies, with full tables deferred to the appendix. These include the integration strategy and damping schedule (Appendix F), loop count K (Figure 3), window width and position (Table 1 and Appendix I), and ablations varying strategy, iteration count, and cache choice (Appendix E).

3.7 Comparison with other looped transformer methods

We further isolate the contribution of our method by comparing it against two natural alternatives on the same frozen checkpoints (Dehghani et al., 2019; Fan et al., 2025; Geiping et al., 2025a; Zhu et al., 2025; Bae et al., 2025a;b; Jeddi et al., 2026; Oncescu et al., 2026; Tang et al., 2026; Saunshi et al., 2025; Xu & Sato, 2025a; Wu et al., 2025). The naive looped transformer of Giannou et al. (2023) collapses on every cell, since the iterates leave the regime the post-loop layers were trained on (Section 2.3), and accuracy drops by several percentage points. In comparison, our method remains within the trained regime, producing the highest accuracy on nearly every cell. Figure 4 reports the results on Llama-3.2-3B-Instruct (GPQA-Main (Rein et al., 2024), MMLU-Pro (Wang et al., 2024), MMLU (Hendrycks et al., 2021)) and Moonlight-16B-A3B-Instruct (ARC-C (Clark et al., 2018), MMLU, CSQA (Talmor et al., 2019)).

4 Conclusion

Our training-free wrapper applies to frozen released checkpoints (Qwen3, Llama-3.2, DeepSeek-V2-Lite, Moonlight, Qwen1.5-MoE) by reinterpreting each pre-norm transformer block as a forward Euler step at $h{=}1$ and providing a better approximation via iterating a contiguous loop window K times. Block-mode and layer-mode iteration are both supported, with layer-mode required on MoE backbones to stabilize per-layer routing.

AI USE STATEMENT

We used generative AI assistants to convert the manuscript to the ICLR format, to draft and edit text, including appendix passages written from the authors' notes, to check references and cross-references, and to assist with plotting code. In our evaluation, IMO-AnswerBench responses are graded by AnswerAutoGrader, the official IMO-Bench grader, which queries Gemini 2.5 Pro (Appendix H). The authors reviewed all AI-assisted text and code, and we take full responsibility for the content of this paper, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work evaluates publicly released language models on public benchmarks. It involves no human subjects, no new data collection, and no personal data, and the method does not modify model weights.

REPRODUCIBILITY STATEMENT

Our method requires no training and wraps released Hugging Face checkpoints at inference time. The fixed recipe is specified in Section 3.1, the looped update in Algorithm 3 (Appendix C.3), and the decode-time KV-cache handling in Appendix D. The evaluation protocol and the vLLM implementation used for the open-ended reasoning results are described in Appendix H, and the software stack and compute footprint in Appendix G.

REFERENCES

Kwangjun Ahn, Xiang Cheng, Hadi Daneshmand, and Suvrit Sra. Transformers learn to implement preconditioned gradient descent for in-context learning. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023*, 2023.

Alexander C. Aitken. On Bernoulli's numerical solution of algebraic equations. *Proceedings of the Royal Society of Edinburgh*, 46:289–305, 1927.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *CoRR*, abs/2108.07732, 2021.

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

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron C. Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025*, 2025b.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*, pp. 688–699, 2019.

Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Multiscale deep equilibrium models. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*, 2020.

Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to ponder. *CoRR*, abs/2107.05407, 2021.

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022*, 2022.

Nora Belrose, Zach Furman, Logan Smith, Danny Halawi, Igor Ostrovsky, Lev McKinney, Stella Biderman, and Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens. *CoRR*, abs/2303.08112, 2023.

John C. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, 3rd edition, 2016.

Bo Chen, Xiaoyu Li, Yingyu Liang, Zhenmei Shi, and Zhao Song. Bypassing the exponential dependency: Looped transformers efficiently learn in-context by multi-step gradient descent. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2025*, Proceedings of Machine Learning Research, pp. 4447–4455, 2025.

Guanxu Chen, Dongrui Liu, and Jing Shao. Loop as a bridge: Can looped transformers truly link representation space and natural language outputs? *CoRR*, abs/2601.10242, 2026a.

Hung-Hsuan Chen. Thinking deeper, not longer: Depth-recurrent transformers for compositional generalization. *CoRR*, abs/2603.21676, 2026.

Lizhang Chen. Demystifying LION: a Hamiltonian perspective. Master's thesis, The University of Texas at Austin, 2025.

Lizhang Chen, Bo Liu, Kaizhao Liang, and Qiang Liu. Lion secretly solves a constrained optimization: As Lyapunov predicts. In *The Twelfth International Conference on Learning Representations, ICLR 2024*, 2024.

Lizhang Chen, Jonathan Li, Kaizhao Liang, Baiyu Su, Cong Xie, Chen Liang, Ni Lao, and Qiang Liu. Cautious weight decay. In *The Fourteenth International Conference on Learning Representations, ICLR 2026*, 2026b.

Lizhang Chen, Jonathan Li, and Qiang Liu. Muon optimizes under spectral norm constraints. *Trans. Mach. Learn. Res.*, 2026, 2026c.

Lizhang Chen, Jonathan Li, Qi Wang, Runlong Liao, Shuozhe Li, Chen Liang, Ni Lao, and Qiang Liu. ϕ-balancing for mixture-of-experts training. In *Proceedings of the 43rd International Conference on Machine Learning, ICML 2026*, 2026d.

Jeffrey Cheng and Benjamin Van Durme. Compressed chain of thought: Efficient reasoning through dense representations. *CoRR*, abs/2412.13171, 2024.

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. *CoRR*, abs/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. *CoRR*, abs/2110.14168, 2021.

Róbert Csordás, Kazuki Irie, Jürgen Schmidhuber, Christopher Potts, and Christopher D. Manning. MoEUT: Mixture-of-experts universal transformers. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024*, 2024.

Artur Back de Luca and Kimon Fountoulakis. Simulation of graph algorithms with looped transformers. In *Forty-first International Conference on Machine Learning, ICML 2024*, Proceedings of Machine Learning Research, pp. 2319–2363, 2024.

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *CoRR*, abs/2405.04434, 2024.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *7th International Conference on Learning Representations, ICLR 2019*, 2019.

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. In *The Thirteenth International Conference on Learning Representations, ICLR 2025*, 2025.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: A theoretical perspective. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023*, 2023.

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. A framework for few-shot language model evaluation, 2023. URL https://zenodo.org/records/10256836.

Yihang Gao, Chuanyang Zheng, Enze Xie, Han Shi, Tianyang Hu, Yu Li, Michael Ng, Zhenguo Li, and Zhaoqiang Liu. AlgoFormer: An efficient transformer framework with algorithmic structures. *Trans. Mach. Learn. Res.*, 2025, 2025.

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 *Forty-first International Conference on Machine Learning, ICML 2024*, Proceedings of Machine Learning Research, pp. 15130–15152, 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 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025*, 2025a.

Jonas Geiping, Xinyu Yang, and Guinan Su. Efficient parallel samplers for recurrent-depth models and their connection to diffusion language models. *CoRR*, abs/2510.14961, 2025b.

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, ICML 2023*, Proceedings of Machine Learning Research, pp. 11398–11442, 2023.

Zixuan Gong, Jiaye Teng, and Yong Liu. What makes looped transformers perform better than non-recursive ones (provably). *CoRR*, abs/2510.10089, 2025.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. In *The Twelfth International Conference on Learning Representations, ICLR 2024*, 2024.

Alex Graves. Adaptive computation time for recurrent neural networks. *CoRR*, abs/1603.08983, 2016.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi: 10.1038/s41586-025-09422-z.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Second Conference on Language Modeling, COLM 2025*, 2025.

Ahmed Heakl, Martin Gubri, Salman Khan, Sangdoo Yun, and Seong Joon Oh. Dr.LLM: Dynamic layer routing in LLMs. In *The Fourteenth International Conference on Learning Representations, ICLR 2026*, 2026.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021*, 2021.

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

Jonas Knupp, Jan Hendrik Metzen, Jeremias Bohn, Georg Groh, and Kristian Kersting. Depth-recurrent attention mixtures: Giving latent reasoning the attention it deserves. *CoRR*, abs/2601.21582, 2026.

Harsh Kohli, Srinivasan Parthasarathy, Huan Sun, and Yuekun Yao. Loop, think, & generalize: Implicit reasoning in recurrent-depth transformers. In *Third Conference on Language Modeling, COLM 2026*, 2026.

Yeskendir Koishekenov, Aldo Lipani, and Nicola Cancedda. Encode, think, decode: Scaling test-time reasoning with recursive latent thoughts. *CoRR*, abs/2510.07358, 2025.

Vedang Lad, Jin Hwa Lee, Wes Gurnee, and Max Tegmark. Remarkable robustness of LLMs: Stages of inference? In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025*, 2025.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020*, 2020.

Bei Li, Quan Du, Tao Zhou, Yi Jing, Shuhan Zhou, Xin Zeng, Tong Xiao, JingBo Zhu, Xuebo Liu, and Min Zhang. ODE transformer: An ordinary differential equation-inspired model for sequence generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022*, pp. 8335–8351, 2022. doi: 10.18653/v1/2022.acl-long.571.

Shuozhe Li, Vaishnav Tadiparthi, Kwonjoon Lee, Nakul Agarwal, Hossein Nourkhiz Mahjoub, Ehsan Moradi-Pari, Lizhang Chen, Amy Zhang, and Liu Leqi. Learning robust reasoning through guided adversarial self-play. *CoRR*, abs/2602.00173, 2026.

Ziyue Li, Yang Li, and Tianyi Zhou. Skip a layer or loop it? test-time depth adaptation of pretrained LLMs. *CoRR*, abs/2507.07996, 2025.

Jia Liang and Liangming Pan. Do Latent-CoT models think step-by-step? A mechanistic study on sequential reasoning tasks. *CoRR*, abs/2602.00449, 2026.

Kaizhao Liang, Bo Liu, Lizhang Chen, and Qiang Liu. Memory-efficient LLM training with online subspace descent. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024*, 2024.

Kaizhao Liang, Lizhang Chen, Bo Liu, and Qiang Liu. Cautious optimizers: Improving training with one line of code. In *The Fourteenth International Conference on Learning Representations, ICLR 2026*, 2026.

Runlong Liao, Jian Yu, Baiyu Su, Chi Zhang, Lizhang Chen, and Qiang Liu. Momentum guidance: Plug-and-play guidance for flow models. *arXiv preprint arXiv:2602.20360*, 2026.

Bo Liu, Lemeng Wu, Lizhang Chen, Kaizhao Liang, Jiaxu Zhu, Chen Liang, Raghuraman Krishnamoorthi, and Qiang Liu. Communication efficient distributed training with distributed Lion. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024*, 2024.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, Mengnan Dong, Zheng Zhang, Yongsheng Kang, Hao Zhang, Xinran Xu, Yutao Zhang, Yuxin Wu, Xinyu Zhou, and Zhilin Yang. Muon is scalable for LLM training. *CoRR*, abs/2502.16982, 2025.

Llama Team. The Llama 3 herd of models. *CoRR*, abs/2407.21783, 2024.

Wenquan Lu, Yuechuan Yang, Kyle Lee, Yanshu Li, and Enqi Liu. Latent chain-of-thought? decoding the depth-recurrent transformer. *CoRR*, abs/2507.02199, 2025.

Thang Luong, Dawsen Hwang, Hoang H. Nguyen, Golnaz Ghiasi, Yuri Chervonyi, Insuk Seo, Junsu Kim, Garrett Bingham, Jonathan Lee, Swaroop Mishra, Alex Zhai, Clara Huiyi Hu, Henryk Michalewski, Jimin Kim, Jeonghyun Ahn, Junhwi Bae, Xingyou Song, Trieu H. Trinh, Quoc V. Le, and Junehyuk Jung. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 35418–35442. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.1794. URL https://aclanthology.org/2025.emnlp-main.1794/.

Jonathan Lys, Vincent Gripon, Bastien Pasdeloup, Axel Marmoret, Lukas Mauch, Fabien Cardinaux, and Ghouthi Boukli Hacene. Inner loop inference for pretrained transformers: Unlocking latent capabilities without training. *CoRR*, abs/2602.14759, 2026.

Arvind V. Mahankali, Tatsunori Hashimoto, and Tengyu Ma. One step of gradient descent is provably the optimal in-context learner with one layer of linear self-attention. In *The Twelfth International Conference on Learning Representations, ICLR 2024*, 2024.

Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. ShortGPT: Layers in large language models are more redundant than you expect. In *Findings of the Association for Computational Linguistics, ACL 2025*, Findings of ACL, pp. 20192–20204, 2025.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. In *The Twelfth International Conference on Learning Representations, ICLR 2024*, 2024.

William Merrill and Ashish Sabharwal. A little depth goes a long way: The expressive power of log-depth transformers. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025*, 2025.

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

Amirkeivan Mohtashami, Matteo Pagliardini, and Martin Jaggi. CoTFormer: A chain of thought driven architecture with budget-adaptive computation cost at inference. In *The Thirteenth International Conference on Learning Representations, ICLR 2025*, 2025.

Son Nguyen, Lizhang Chen, Bo Liu, and Qiang Liu. Memory-efficient optimization with factorized Hamiltonian descent. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2025*, Proceedings of Machine Learning Research, pp. 2863–2871, 2025.

Son Nguyen, Bo Liu, Lizhang Chen, and Qiang Liu. Improving adaptive moment optimization via preconditioner diagonalization. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2026*, 2026.

Mohammadmahdi Nouriborji, Morteza Rohanian, and Omid Rohanian. Improving recursive transformers with mixture of LoRAs. *CoRR*, abs/2512.12880, 2025.

Costin-Andrei Oncescu, Depen Morwani, Samy Jelassi, Alexandru Meterez, Mujin Kwun, and Sham M. Kakade. The recurrent transformer: Greater effective depth and efficient decoding. *CoRR*, abs/2604.21215, 2026.

Francesco Pappone, Donato Crisostomi, and Emanuele Rodolà. Two-scale latent dynamics for recurrent-depth transformers. *CoRR*, abs/2509.23314, 2025.

Bowen Peng, Lizhang Chen, Baiyu Su, Jeffrey Quesnelle, Diederik P. Kingma, and Qiang Liu. DeMo: Decoupled momentum optimization. In *The Fourteenth International Conference on Learning Representations, ICLR 2026*, 2026.

Jacob Pfau, William Merrill, and Samuel R. Bowman. Let's think dot by dot: Hidden computation in transformer language models. In *First Conference on Language Modeling, COLM 2024*, 2024.

Boris T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

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

Qwen Team. Qwen1.5-MoE: Matching 7B model performance with 1/3 activated parameters. Technical report, Alibaba Group, 2024. URL https://qwen.ai/blog?id=qwen-moe.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025.

Machel Reid, Edison Marrese-Taylor, and Yutaka Matsuo. Subformer: Exploring weight sharing for parameter efficiency in generative transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, Findings of ACL, pp. 4081–4090, 2021.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level Google-proof Q&A benchmark. In *First Conference on Language Modeling*, 2024.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *The Thirteenth International Conference on Learning Representations, ICLR 2025*, 2025.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021*, pp. 6695–6706, 2021.

Baiyu Su, Lizhang Chen, and Qiang Liu. The curious case of AdamW, 2026. URL https://openreview.net/forum?id=kW5hSRG5wq.

Sho Takase and Shun Kiyono. Lessons on parameter sharing across layers in transformers. In *Proceedings of The Fourth Workshop on Simple and Efficient Natural Language Processing, SustaiNLP 2023*, pp. 78–90, 2023.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Volume 1 (Long and Short Papers)*, pp. 4149–4158, 2019.

Guo Tang, Shixin Jiang, Heng Chang, Nuo Chen, Yuhan Li, Huiming Fan, Jia Li, Ming Liu, and Bing Qin. LoopRPT: Reinforcement pre-training for looped language models. *CoRR*, abs/2603.19714, 2026.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pp. 5998–6008, 2017.

Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning, ICML 2023*, Proceedings of Machine Learning Research, pp. 35151–35174, 2023.

Homer F. Walker and Peng Ni. Anderson acceleration for fixed-point iterations. *SIAM J. Numer. Anal.*, 49(4):1715–1735, 2011.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023*, 2023.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022*, 2022.

Bohong Wu, Mengzhao Chen, Xiang Luo, Shen Yan, Qifan Yu, Fan Xia, Tianqi Zhang, Hongrui Zhan, Zheng Zhong, Xun Zhou, Siyuan Qiao, and Xingyan Bin. Parallel loop transformer for efficient test-time computation scaling. *CoRR*, abs/2510.24824, 2025.

Kevin Xu and Issei Sato. On expressive power of looped transformers: Theoretical analysis and enhancement via timestep encoding. In *Forty-second International Conference on Machine Learning, ICML 2025*, Proceedings of Machine Learning Research, 2025a.

Kevin Xu and Issei Sato. To CoT or to loop? A formal comparison between chain-of-thought and looped transformers. *CoRR*, abs/2505.19245, 2025b.

Liu Yang, Kangwook Lee, Robert D. Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *The Twelfth International Conference on Learning Representations, ICLR 2024*, 2024.

Chengting Yu, Xiaobo Shu, Yadao Wang, Yizhen Zhang, Haoyi Wu, You Wu, Rujiao Long, Ziheng Chen, Yuchi Xu, Wenbo Su, and Bo Zheng. SpiralFormer: Looped transformers can learn hierarchical dependencies via multi-resolution recursion. *CoRR*, abs/2602.11698, 2026a.

Qifan Yu, Zhenyu He, Sijie Li, Zhou Xun, Jun Zhang, Jingjing Xu, and Di He. Enhancing auto-regressive chain-of-thought through loop-aligned reasoning. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2026 - Volume 1: Long Papers*, pp. 2206–2222, 2026b.

Boyi Zeng, Shixiang Song, Siyuan Huang, Yixuan Wang, He Li, Ziwei He, Xinbing Wang, Zhiyu Li, and Zhouhan Lin. PonderLM: Pretraining language models to ponder in continuous space. In *The Fourteenth International Conference on Learning Representations, ICLR 2026*, 2026.

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

APPENDIX

A Related work **18**

B Notation and glossary **18**

C Proofs **18**

C.1 Proof of (9) 18

C.2 Proof of (16) 19

C.3 Efficient implementation 20

D Decode loop algorithm **20**

D.1 Decode-time looped forward pass 20

D.2 Snapshot/restore 21

D.3 `decode_mode` variants 22

E Ablations on Qwen3-1.7B-Base **22**

F Strategy ablation tables **22**

F.1 Anderson, heavy-ball, Aitken, α-schedules 23

F.2 Norm stabilization and polynomial blending 23

F.3 Layer-mode replicates the result 24

G Compute and reproducibility **25**

G.1 Software stack 25

G.2 Per-model resource footprint 25

H Evaluation protocol for open-ended reasoning **26**

H.1 Implementation in vLLM 27

I Robustness to window position across dense Qwen3 sizes **27**

J Per-benchmark accuracy on Qwen3 models **27**

A Related work

Looped and recurrent-depth transformers. A large body of work incorporates recurrence into the transformer at *training* time, from weight-tied stacks (Dehghani et al., 2019; Lan et al., 2020) and looped expressivity results (Giannou et al., 2023; Xu & Sato, 2025a; Merrill & Sabharwal, 2025; Gong et al., 2025; Yang et al., 2024; Geiping et al., 2025b) to length-generalization studies (Fan et al., 2025; de Luca & Fountoulakis, 2024; Schwarzschild et al., 2021; Bansal et al., 2022) and looped pretraining at scale (Geiping et al., 2025a; Zhu et al., 2025; Prairie et al., 2026; Tang et al., 2026; Wu et al., 2025). Adaptive-recursion variants (Gao et al., 2025; Csordás et al., 2024; Bae et al., 2025b;a; Nouriborji et al., 2025) and implicit-depth fixed points (Bai et al., 2019; 2020) likewise tie recurrence to training, as do recent elastic-depth latent reasoning models (Jeddi et al., 2026; Oncescu et al., 2026; Chen, 2026; Knupp et al., 2026; Yu et al., 2026a; Pappone et al., 2025; Koishekenov et al., 2025). Our wrapper instead targets *unmodified released checkpoints* (Qwen Team, 2025; Llama Team, 2024; DeepSeek-AI, 2024; Liu et al., 2025) with no training (Liang et al., 2024; Liu et al., 2024; Liang et al., 2026; Nguyen et al., 2025; Chen, 2025; Chen et al., 2026b; Nguyen et al., 2026), no auxiliary loss, and no architectural changes.

Inference-time compute and latent reasoning. A separate axis spends test-time compute on longer trajectories, e.g. chain-of-thought (Wei et al., 2022; Mohtashami et al., 2025), self-consistency (Wang et al., 2023), and reasoning-trained LMs (Guo et al., 2025), all leaving the per-token forward unchanged; theory characterizes their advantages (Feng et al., 2023; Merrill & Sabharwal, 2024; Xu & Sato, 2025b; Saunshi et al., 2025). Latent-reasoning lines instead push compute into a continuous space (Hao et al., 2025; Cheng & Van Durme, 2024; Pfau et al., 2024; Goyal et al., 2024; Zeng et al., 2026; Yu et al., 2026b; Geiping et al., 2025a; Li et al., 2026) or use adaptive halting to "ponder" (Graves, 2016; Banino et al., 2021; Kohli et al., 2026; Lu et al., 2025; Liang & Pan, 2026; Chen et al., 2026a), echoing the tuned-lens view of forward computation as iterative refinement (Belrose et al., 2023) and layer-redundancy findings (Men et al., 2025; Lad et al., 2025; Takase & Kiyono, 2023; Reid et al., 2021). Our intervention is orthogonal, as we use more compute *per token within a single forward pass* on a frozen checkpoint.

Numerical analysis methods. Treating a residual block as a forward Euler step motivates the family of integration strategies our wrapper exposes on the looped window, in line with the fixed-point picture (Bai et al., 2019; 2020). Within that family we benchmark Picard acceleration via Anderson (Walker & Ni, 2011), heavy-ball (Polyak, 1964; Chen et al., 2024; 2026c; Peng et al., 2026; Liao et al., 2026; Su et al., 2026), Aitken extrapolation, and higher-order Runge–Kutta solvers, and find none robustly beats damped Euler, suggesting the looped middle of a pretrained transformer (Giannou et al., 2023; Saunshi et al., 2025; Bae et al., 2025b) is not contractive in any useful sense (Bai et al., 2019; Belrose et al., 2023). Concurrent training-free directions manipulate depth via layer skipping (Men et al., 2025; Lad et al., 2025), but to our knowledge, no prior method combines mid-block looping and layer-mode iteration for MoE routing (Chen et al., 2026d) across modern checkpoints (Qwen Team, 2025; Llama Team, 2024; DeepSeek-AI, 2024; Liu et al., 2025).

B Notation and glossary

Table 5 consolidates the symbols used across the paper and the appendix.

C Proofs

C.1 Proof of (9)

We expand block-mode iteration as an explicit chain of per-layer forward Euler steps and verify the closed-form identity stated in Section 2.3.

Let $g = L_b \circ L_{b-1} \circ \cdots \circ L_a$ be the loop window operator. Each constituent layer satisfies the single-layer forward Euler identity $L_i(z) = z + F_i(z)$. Fix an input x. Set $y_a(x) := x$ and define the *intra-window residual stream* recursively by

$$y_{i+1}(x) := L_i(y_i(x)) := y_i(x) + F_i\big(y_i(x)\big), \qquad i = a, a+1, \ldots, b. \tag{17}$$

Table 5: Symbols, types, and where they are introduced.

Symbol	Type / range	Meaning
Network		
N	$\mathbb{N}_{>0}$	Number of decoder layers in the released checkpoint.
L_i	$\mathbb{R}^{T\times d}\to\mathbb{R}^{T\times d}$	Pre-norm transformer block i, $0 \le i \le N-1$.
f	composition	Unmodified network: $f = L_{N-1}\circ\cdots\circ L_0$.
T, d	$\mathbb{N}_{>0}$	Sequence length and hidden dimension.
Loop wrapper		
$[a, b]$	$0 \le a \le b \le N-1$	Loop-window layer indices (contiguous).
W	$b-a+1$	Loop window width (in layers).
K	$\mathbb{N}_{\ge 1}$	Loop iteration count.
g	$\mathbb{R}^{T\times d}\to\mathbb{R}^{T\times d}$	Block operator: $g = L_b\circ\cdots\circ L_a$ (Eq. 1).
$g^{(K)}$	$\mathbb{R}^{T\times d}\to\mathbb{R}^{T\times d}$	K-step iteration of g under the chosen strategy (Section 2.3).
$\hat{f}(x)$	$\mathbb{R}^{T\times d}$	Looped network output (2).
$g^{(K)}_{\text{blk}}, g^{(K)}_{\text{lyr}}$	maps	Block-mode and layer-mode realizations of $g^{(K)}$ (3)–(4).
ODE view		
F_g	$\mathbb{R}^{T\times d}\to\mathbb{R}^{T\times d}$	Window residual field $F_g(x) := g(x) - x$ (6).
F_{L_i}	$\mathbb{R}^{T\times d}\to\mathbb{R}^{T\times d}$	Single-layer residual field $L_i(x) - x$.
$y_i(x)$	$\mathbb{R}^{T\times d}$	Intra-window residual stream at layer i (9).
h	$\mathbb{R}_{>0}$	Forward-Euler step size; h=1 is the layer-built-in step.
α	$(0, 1]$	Damped-Euler coefficient α; $\alpha = h$.
$\kappa_1, \ldots, \kappa_4$	$\mathbb{R}^{T\times d}$	RK4 stage evaluations (Table 8).
m, β	$\mathbb{N}$, $\mathbb{R}$	Anderson memory depth and mixing parameter.
KV cache		
$\mathcal{C}_i$	cache slot	Per-layer KV slot for layer i.
ℓ_i	$\mathbb{N}$	Snapshot length of $\mathcal{C}_i$ before a loop iteration.
c	$\{\text{FIRST}, \text{LAST}, \text{NONE}\}$	Cache strategy (Algorithm 4).

By construction, $g(x) = y_{b+1}(x)$. Summing (17) from $i = a$ to $i = b$ gives

$$\sum_{i=a}^{b}(y_{i+1}(x) - y_i(x)) = \sum_{i=a}^{b} F_i\big(y_i(x)\big).$$

The LHS telescopes to $y_{b+1}(x) - y_a(x) = g(x) - x$. Therefore

$$g(x) - x = \sum_{i=a}^{b} F_i(y_i(x)) =: F_g(x), \tag{18}$$

which is exactly (9).

C.2 Proof of (16)

By (14), the standard K-substep forward Euler endpoint can be written as

$$F^{(K)}(x_0) = x_0 + \frac{1}{K}\sum_{i=1}^{K} k_i.$$

Algorithm 3 Explicit **block-mode** **layer-mode** Runge–Kutta with Butcher tableau (13)–(15). Blue-shaded steps apply to block-mode and red-shaded steps to layer-mode.

Input: x, range $[a, b]$, number of stages K, anchor weight $\beta \in [0, 1]$

$x \leftarrow \textsc{PreLoop}(x)$ ▷ *embedding, pre-block, etc.*
$\tilde{x} \leftarrow (L_b \circ \cdots \circ L_a)(x)$ ▷ *anchor (block-mode)*
for $\ell = a, \ldots, b$ **do** ▷ *outer loop over each layer*
  $\tilde{x} \leftarrow L_\ell(x)$ ▷ *anchor (layer-mode)*
  for $k = 1, \ldots, K-1$ **do** ▷ *iterate over the K Runge–Kutta stages*
    $y \leftarrow (L_b \circ \cdots \circ L_a)(x)$ $y \leftarrow L_\ell(x)$
    $x \leftarrow x + \frac{1}{K}(y - x)$ ▷ *Runge–Kutta update*
  end for
  $x \leftarrow \beta\tilde{x} + (1-\beta)x$ ▷ *anchor combination*
end for
return $\textsc{PostLoop}(x)$

Substituting into the RHS of (16), we have

$$\begin{aligned}
\beta g(x_0) + (1-\beta)F^{(k)}(x_0) &= \beta g(x_0) + (1-\beta)x_0 + \frac{1-\beta}{K}\sum_{i=1}^{K} k_i \\
&= \beta g(x_0) + (1-\beta)x_0 + \frac{1-\beta}{K}F_g(x_0) + \frac{1-\beta}{K}\sum_{i=2}^{K} k_i \\
&= x_0 + \left(\beta + \frac{1-\beta}{K}\right)k_1 + \frac{1-\beta}{K}\sum_{i=2}^{K} k_i,
\end{aligned}$$

where the second and third lines use $k_1 = F_g(x_0) = g(x_0) - x_0$. The conclusion follows upon expanding the LHS of (16) using the definitions in (15).

C.3 Efficient implementation

Equation (16) expresses the Runge–Kutta update with the Butcher tableau (13)–(15) as a blend of the anchor output and damped Euler steps, so the K stage increments never need to be stored. Algorithm 3 gives the resulting implementation for both block-mode and layer-mode.

D Decode loop algorithm

The looped forward pass we patch into a frozen model has two regimes that share most code but differ in how the loop body interacts with the KV cache. During *prefill* (`seq_len > 1`, no past KV) the loop body is run with `use_cache=False` and writes nothing to the cache; only a single *stash pass* after the loop writes the canonical KV (Zhu et al., 2025). During *decode* (`seq_len == 1`, an existing past KV) the loop body must attend to the past KV; otherwise, the new token is computed against a truncated context. However, it must also produce no net KV writes, since the stash pass will write the canonical entry afterwards. The mechanism is to snapshot per-loop-layer cache lengths before each loop iteration and crop back to those lengths immediately after.

D.1 Decode-time looped forward pass

Algorithm 4 states the full block-mode decode-time forward pass as a single procedure, including the pre-loop / loop-body / stash / post-loop phases, the per-iteration snapshot/restore protocol, and the cache-strategy branch. Algorithm 5 states the layer-mode counterpart, which differs only in how iterations and snapshot/restore are nested.

Algorithm 4 Decode-time block-mode looped forward pass with KV cache

Input: new-token hidden state $x \in \mathbb{R}^{1\times d}$, loop window $[a, b]$, loop count K, strategy update $\mathcal{S}$ (Euler/heavy-ball/Anderson/...), cache strategy $c \in \{\text{FIRST}, \text{LAST}, \text{NONE}\}$, per-layer KV cache $\mathcal{C} = (\mathcal{C}_0, \ldots, \mathcal{C}_{N-1})$

Output: Updated hidden state x'; $\mathcal{C}$ contains exactly one canonical KV entry per loop layer at the new decode position

1: **for** $i = 0, \ldots, a-1$ **do** ▷ *pre-loop: standard layers, 1 KV each*
2: $x \leftarrow L_i(x; \mathcal{C}_i, \texttt{use_cache} = \text{T})$
3: **end for**
4: $x_a \leftarrow x$ ▷ *save pre-loop input for* $c =$ FIRST
5: **for** $i = a, \ldots, b$ **do** ▷ *snapshot loop-layer cache lengths*
6: $\ell_i \leftarrow |\mathcal{C}_i|$
7: **end for**
8: **for** $k = 1, \ldots, K$ **do** ▷ K *loop iterations under strategy* $\mathcal{S}$
9: $y \leftarrow x$ ▷ *evaluation starts from* x
10: **for** $i = a, \ldots, b$ **do** ▷ *run body with* `use_cache` $=$ T
11: $y \leftarrow L_i(y; \mathcal{C}_i, \texttt{use_cache} = \text{T})$ ▷ *attn reads past KV at decode pos.*
12: **end for**
13: **for** $i = a, \ldots, b$ **do** ▷ *crop: discard iteration KV writes*
14: $\mathcal{C}_i \leftarrow \texttt{crop}(\mathcal{C}_i, \ell_i)$ ▷ *net cache effect of body* $= 0$
15: **end for**
16: $x \leftarrow \mathcal{S}(x, y, k)$ ▷ *e.g.* $x + \alpha(y - x)$ *for damped Euler*
17: **end for**
18: **if** $c \neq$ NONE **then** ▷ *stash phase: 1 KV write per loop layer*
19: $z \leftarrow \begin{cases} x, & c = \text{LAST}, \\ x_a, & c = \text{FIRST} \end{cases}$
20: **for** $i = a, \ldots, b$ **do**
21: $z \leftarrow L_i(z; \mathcal{C}_i, \texttt{use_cache} = \text{T})$ ▷ *canonical KV at decode pos.*
22: **end for**
23: **end if**
24: **for** $i = b+1, \ldots, N-1$ **do** ▷ *post-loop: standard layers, 1 KV each*
25: $x \leftarrow L_i(x; \mathcal{C}_i, \texttt{use_cache} = \text{T})$
26: **end for**
27: **return** $\text{LayerNorm}_{\text{out}}(x)$

Lines 4–5 of Algorithm 4 record the cache length each loop layer holds *before* the body runs. The body in lines 6–12 is then executed K times: each iteration runs the window with `use_cache=True` (so attention reads the genuine past KV at the new decode position, line 9), and the immediately-following crop in lines 10–11 truncates the cache back to ℓ_i, leaving zero net KV effect from the iteration. The strategy step on line 12 produces the next iterate x from the current input x and the body output y. Lines 13–16 are the stash phase: a single additional pass through the loop layers writes exactly one canonical KV entry per loop layer per decode position, using either the post-loop hidden state (c=LAST) or the pre-loop input (c=FIRST) as the input to that pass. Lines 17–19 finish with the standard post-loop layers and the final output norm. The total cache delta of the entire procedure is exactly $b - a + 1$ entries regardless of K, which is identical to the unmodified model.

D.2 Snapshot/restore

`transformers.DynamicCache.update()` unconditionally appends the new (k, v) to its per-layer tensor: there is no in-place overwrite mode. A naive K-iteration decode body would therefore write K extra KV entries per loop layer at the same logical decode position, then attend to all of them on the next decode step, corrupting the cache and producing a sequence of phantom prefix tokens. The snapshot/restore pattern is the cheapest fix that (a) lets each loop iteration attend to the genuine past KV and (b) leaves the cache exactly as it was pre-iteration. The single canonical KV is then

Algorithm 5 Decode-time *layer-mode* looped forward pass with KV cache (per-layer iteration; the safer default on MoE backbones, Section 2.2).

Input: Same as Algorithm 4.
Output: Same as Algorithm 4.

for $i = 0, \ldots, a-1$ **do** ▷ *pre-loop, standard*
  $x \leftarrow L_i(x;\, \mathcal{C}_i,\, \texttt{use_cache=T})$
end for
$x_a \leftarrow x$
for $i = a, \ldots, b$ **do** ▷ *outer: layers in order*
  $\ell_i \leftarrow |\mathcal{C}_i|$ ▷ *snapshot length for layer i*
  for $k = 1, \ldots, K$ **do** ▷ *inner: iterate this layer K times*
    $y \leftarrow L_i(x;\, \mathcal{C}_i,\, \texttt{use_cache=T})$ ▷ *MoE: routing pinned across k*
    $\mathcal{C}_i \leftarrow \texttt{crop}(\mathcal{C}_i,\, \ell_i)$ ▷ *discard iteration KV write*
    $x \leftarrow \mathcal{S}(x,\, y,\, k)$ ▷ *per-layer strategy update*
  end for
  $z \leftarrow x$ **if** c=LAST **else** x_a
  if $c \neq$ NONE **then**
    $z \leftarrow L_i(z;\, \mathcal{C}_i,\, \texttt{use_cache=T})$ ▷ *canonical KV at layer i*
  end if
end for
for $i = b+1, \ldots, N-1$ **do** ▷ *post-loop, standard*
  $x \leftarrow L_i(x;\, \mathcal{C}_i,\, \texttt{use_cache=T})$
end for
return $\text{LayerNorm}_{\text{out}}(x)$

written by the stash pass on the chosen `cache_strategy`. layer-mode looping uses an analogous per-layer snapshot/crop pair, applied K times to one layer before moving on.

D.3 `DECODE_MODE` VARIANTS

We expose three `decode_mode` settings. `bypass` skips the loop entirely on incremental decode and only loops during prefill (the default for loglikelihood-only evaluations). `full` loops every decode step. `first_n` loops only on the first n generated tokens (intended for chain-of-thought (CoT) prefix refinement; in practice never beats `full` when the loop helps and never recovers when it hurts). `full` is the default for the generation results reported in the main paper.

E ABLATIONS ON QWEN3-1.7B-BASE

We ablate strategy, K, window width, and cache strategy on Qwen3-1.7B-Base at the canonical window `[12-15]`. Table 6 reports the results.

Table 7 pulls representative rows from the 40+ configuration sweep in Appendix F, demonstrating that higher-order integrators do not help. The looped block is not a smooth ODE, and Anderson-style fixed-point acceleration (Walker & Ni, 2011; Bai et al., 2019; 2020) overshoots because the block is not contractive enough.

F STRATEGY ABLATION TABLES

This section reproduces, in full numerical form, the strategy comparisons that motivate a main result of the paper: *every classical fixed-point acceleration we tried fails to outperform K=2 damped Euler on the looped block.* All entries use Qwen3-1.7B-Base on the canonical loop window `[12-15]` with block-mode looping. The reference is K=2 damped Euler at 0.56113 macro on the 16-task aggregate.

Table 8 lists the iteration strategies compared in this section.

Table 6: Ablations of the loop wrapper on Qwen3-1.7B-Base `[12-15]`. Δpp on the 16-task macro vs. the no-loop baseline; the within-family reference cell is damped Euler at K=2, α=0.5. ↓↓ marks catastrophic divergence (perplexity blowup or strict-format regression). Bold marks the per-axis best. Panel (c) reports GSM8K (Cobbe et al., 2021), MMLU-Pro (Wang et al., 2024), and MBPP (Austin et al., 2021).

(a) Window width n (centered on `[12-15]`, K=2 damped Euler)

n	1	3	**4**	6	12	28 (full)
Δpp	+0.18	+0.21	**+0.55**	−0.82	−0.63	↓↓

(b) Iteration count K **& strategy** (n=4, mid-window)

	naive loop		*damped Euler*		*higher-order & accel.*
K	K=2	K=4	K=2, α=0.5	K=3, $\alpha\approx1/3$	Anderson, RK4, Heun, . . .
Δpp	+0.51	$-10.21_{(0.6B)}$	**+0.55**	$+1.05_{(4B)}$	see Table 7

(c) Cache strategy (n=4, K=2 damped Euler)

	16-task aggregate			*per-benchmark detail*		
strategy	`first`	`last`	`none`	GSM8K `none`	MMLU-Pro `none`	MBPP $\Delta_{\texttt{last-first}}$
Δpp	≥**0**	≥**0**	↓↓	−58.91	−8.86	+0.60

Table 7: Higher-order ODE solvers and fixed-point accelerators vs. damped Euler. Numbers are 16-task macro Δpp on Qwen3-1.7B-Base `[12-15]`; each column is referenced to its own damped Euler K=2, α=0.5 baseline. "—" = not tested in that mode. The full sweep (40+ configurations across both modes) is reported in Appendix F.

Family	Method (K=2 unless noted)	Block-mode Δ	Layer-mode Δ
Reference	damped Euler (α=0.5, h=1/2)	0.00	0.00
Higher-order Runge–Kutta	midpoint (RK2)	−0.85	−1.96
	Heun (RK2)	−0.92	−1.90
	RK4	−0.65	−2.34
Fixed-point acceleration	heavy-ball (α=0.5, β=0.3)	−0.12	−0.06
	Anderson (K=4, m=2, β=0.5)	−0.96	—
	Aitken Δ^2 (safeguarded)	−7.00	—
	Anderson, worst (K=8, m=3, β=1.0)	**-18.06**	**-19.56**

F.1 Anderson, heavy-ball, Aitken, α-schedules

24 configurations; *none* beat K=2 damped Euler. The closest is the $[0.6, 0.4]$ damped-Euler schedule at −0.03 pp (statistical tie). Anderson extrapolation degrades smoothly with K and catastrophically at K=8 (−18 pp), confirming the residual sequence $x_{k+1} - x_k$ in the loop is not approaching zero in a way Anderson can exploit.

F.2 Norm stabilization and polynomial blending

16 additional configurations; again none beat K=2 damped Euler. `norm_stab` (rescaling each iterate to a fixed L2 norm) is catastrophic at $K \geq 3$, confirming that the iterates are diverging *in direction* too.

Table 8: **Iteration strategies for numerical integration of the ODE (8).**

Strategy	Update Rule	Forward Passes
Naive iteration & forward Euler		
Naive Loop	$x_{k+1} = g(x_k)$	K
Forward Euler (Bai et al., 2019; 2020)	$x_{k+1} = x_k + \frac{1}{K} F_g(x_k)$	K
Higher-order Runge–Kutta (Butcher, 2016)		
Midpoint (RK2)	$x_{k+1} = x_k + hF_g(x_k + \frac{h}{2} F_g(x_k))$	$2K$
Heun (RK2)	$x_{k+1} = x_k + \frac{h}{2}(F_g(x_k) + F_g(x_k + hF_g(x_k)))$	$2K$
RK4	$x_{k+1} = x_k + \frac{h}{6}(\kappa_1 + 2\kappa_2 + 2\kappa_3 + \kappa_4)$	$4K$
Fixed-point accelerators		
Heavy-ball (α, β) (Polyak, 1964)	$x_{k+1} = x_k + \alpha F_g(x_k) + \beta(x_k - x_{k-1})$	K
Anderson (m, β) (Walker & Ni, 2011)	$x_{k+1} = (1-\beta)(x_k - (\Delta X_k)\gamma_k^\star) + \beta(g(x_k) - (\Delta F_k)\gamma_k^\star)$	K
Aitken Δ^2 (Aitken, 1927)	$x_{k+1} = x_k - \frac{(d_{1,k})^2}{d_{2,k}}$ (per-coordinate)	K
Uniform Loop (Lys et al., 2026)	$x_{k+1} = g\left(\frac{1}{k+1}\sum_{i=0}^{k} x_i\right)$	K

Notation. RK4: $\kappa_1 := F_g(x_k)$, $\kappa_2 := F_g(x_k + \frac{h}{2}\kappa_1)$, $\kappa_3 := F_g(x_k + \frac{h}{2}\kappa_2)$, $\kappa_4 := F_g(x_k + h\kappa_3)$.
Anderson: $\gamma_k^\star := \arg\min_\gamma \|f_k - (\Delta F_k)\gamma\|_2$ with $f_i := g(x_i) - x_i$ and ΔX_k, ΔF_k the matrices of the last m residual / state increments, respectively (Walker & Ni, 2011).
Aitken: $d_{1,k} := g(x_k) - x_k$, $d_{2,k} := g(g(x_k)) - 2g(x_k) + x_k$, applied per coordinate; the Steffensen safeguard clips $|x_{k+1} - x_k| \le |d_{1,k}|$ and requires K even.

F.3 Layer-mode replicates the result

In layer-mode the loop body is L_i^K applied per-layer rather than $g^K = (L_b \circ \cdots \circ L_a)^K$ applied to the block. Reference: layer-mode K=2 damped Euler at 56.347 on the same window. Table 9 lists every configuration.

Table 9: Layer-mode strategy sweep on Qwen3-1.7B-Base `[12-15]`. Δ vs layer-mode K=2 damped Euler.

Recipe	16-task	Δ vs layer K=2 damped Euler
K=2 heavy-ball α=0.5, β=0.3	56.292	−0.056
K=2 poly_blend w=[0.25, 0.5, 0.25]	56.181	−0.167
K=3 damped Euler α=0.3	55.278	−1.069
K=4 Euler	55.240	−1.107
K=2 damped Euler α=0.7	55.196	−1.151
K=6 Euler	54.998	−1.350
K=2 Heun	54.445	−1.902
K=2 midpoint	54.385	−1.962
K=1 RK4	54.242	−2.105
K=1 midpoint	54.178	−2.169
K=2 norm_stab α=0.5	54.098	−2.249
K=1 Heun	54.060	−2.287
K=2 RK4	54.005	−2.342
K=3 heavy-ball α=0.5, β=0.3	51.830	−4.517
K=3 Anderson m=2, β=1.0	44.035	−12.312
K=4 Anderson m=2, β=1.0	36.785	−19.562

Higher-order ODE solvers (Heun, midpoint, RK4) degrade more dramatically in layer-mode (−1.7 to −2.3 pp) than in block-mode (−0.7 to −1.1 pp): per-layer the map L_i is *not* better conditioned for higher-order integration than the composed g. Anderson in layer-mode is even more severe (−19.6 pp at K=4). The combined sweep covers 40+ acceleration configurations, and *none* robustly beats damped Euler. This is the empirical basis for the claim that *the looped transformer block is not a contractive map*, so fixed-point acceleration is the wrong tool.

G COMPUTE AND REPRODUCIBILITY

G.1 SOFTWARE STACK

All evaluations use `lm-evaluation-harness` v0.4.11 with deterministic seeds, default sampling settings per task, and `transformers` 4.46. Looped forwards are implemented as monkey patches on the model's `Model` class; patching preserves all original generation/eval code paths and is reversible. For the DeepSeek-V2/V3 family we additionally ship a small `DynamicCache` compatibility shim that re-exposes `seen_tokens`/`get_max_length`/`get_usable_length` to the bundled `modeling_deepseek` remote code.

The open-ended reasoning experiments of Section 3.4 instead run on vLLM 0.29 through a plugin, described in Appendix H.1.

G.2 PER-MODEL RESOURCE FOOTPRINT

Table 10 lists the approximate memory and wall-clock time per model.

Table 10: Approximate single-GPU memory and per-job wallclock for the representative jobs. Times are for the full 16-task suite at default `lm-evaluation-harness` batch sizes; chain-of-thought generation jobs (e.g. MMLU-Pro 5sh CoT) take 2–5$\times$ longer in `decode_mode=full`.

Model	VRAM (bf16)	Hardware used	Eval-suite wallclock
Qwen3-0.6B-Base / -Instruct	$\sim$2 GB	A100-40 / H100-80	$\sim$1 h
Qwen3-1.7B-Base	$\sim$4 GB	A100-40 / H100-80	$\sim$2 h
Qwen3-4B-Base / -Instruct	$\sim$8 GB	A100-40 / H100-80	$\sim$3 h
Llama-3.2-1B / $-$3B-Instruct	$\sim$3–7 GB	A100-40 / H100-80	$\sim$2 h
Qwen1.5-MoE-A2.7B-Chat	$\sim$30 GB	H100-80	$\sim$4 h
DeepSeek-V2-Lite-Chat (16B/2.4B)	$\sim$32 GB	H100-80	$\sim$5 h
Moonlight-16B-A3B-Instruct	$\sim$32 GB	H100-80	$\sim$8 h (`decode=full` 21 h)
Qwen3-30B-A3B-Instruct	$\sim$60 GB	H100-80	$\sim$10 h (19-task 0-shot)

H EVALUATION PROTOCOL FOR OPEN-ENDED REASONING

This section describes how the results in Table 4 are produced. The three AIME years share one protocol. IMO-AnswerBench uses the same generation setup and follows the official IMO-Bench grading protocol. Codeforces is described at the end of this section.

Data and sampling. We use the test splits of `math-ai/aime24`, `math-ai/aime25`, and `math-ai/aime26` on Hugging Face, each of which contains the 30 problems of one year's AIME I and AIME II. Following common practice for evaluating reasoning models (Guo et al., 2025; Qwen Team, 2025), we sample several responses per problem and report their mean accuracy. We draw 16 samples per problem, so each run on one year scores 480 samples. We repeat every run with 10 different base seeds and report avg@16 averaged over these runs, that is, over 4,800 samples per year. The random seed of sample j on problem i is $s_0 \cdot 10^6 + 1000\,i + j$, where s_0 is the base seed of the run. A baseline run and a looped run with the same s_0 therefore consume identical random streams, which makes their comparison paired.

Prompt and thinking mode. The user message is the problem statement followed by the instruction

```
Please reason step by step, and put your final answer
within \boxed{}.
```

We apply the model's own chat template with `add_generation_prompt=True`, with no system prompt and no few-shot examples. Qwen3 runs in thinking mode, which is the template default (`enable_thinking=True`), so it emits a `<think>...</think>` span before the final answer.

Decoding. We sample with temperature 0.6, top-p 0.95, and top-k 20, the settings Qwen recommends for thinking mode, with min-p 0 and no repetition penalty. The maximum generation length is 38,912 new tokens, which is Qwen's recommended setting for competition problems and the largest budget available without YaRN. Extending the context to 64K tokens with YaRN (factor 2) made the model's generations collapse, so no reported result uses YaRN.

Inference engine. Weights are in `bfloat16`, and generation runs on vLLM 0.29. Each run uses 4 engine replicas with tensor parallelism 2, prefix caching disabled, and full CUDA graphs during decoding.

Grading on AIME. We use, unchanged, the AIME scoring function `process_results` of `lm-evaluation-harness` v0.4.13. It takes the content of the last `\boxed{}` in the full output, including the thinking span, and falls back to the text between the first and last `$` when no box is present. The extracted string is compared with the reference answer after the string normalization of the `hendrycks_math` task (`is_equiv`). Samples truncated at the length limit are almost always scored as wrong. In a check of 1,440 AIME 2026 samples, only 2 truncated samples were scored as correct because a box appeared inside the thinking span, one in a baseline run and one in a looped run, so this effect is symmetric. Re-checking 2,880 samples with a lenient integer comparison found no correct answer that had been scored as wrong.

IMO-AnswerBench. We follow the official IMO-Bench protocol (`https://imobench.github.io`). We use IMO-AnswerBench (Luong et al., 2025), the short-answer part of IMO-Bench, in its current release (`answerbench_v2.csv`), which contains 400 Olympiad problems, 100 each in algebra, combinatorics, geometry, and number theory. Responses are generated with the same prompt, decoding settings, and inference engine as for AIME, with 10 base seeds. Many reference answers are sets, intervals, or symbolic expressions that string matching cannot compare reliably. We therefore grade with AnswerAutoGrader, the official grader of IMO-Bench, which prompts Gemini 2.5 Pro with the official grading prompt to extract the final answer and to judge whether it is mathematically equivalent to the reference answer.

Codeforces. We sample 100 problems from the medium and hard difficulty levels of Codeforces and evaluate them with the same 10 base seeds.

H.1 Implementation in vLLM

The open-ended reasoning runs need long sampled generations, so we implement the loop wrapper as a vLLM plugin rather than through the `transformers` patches of Appendix G. The plugin does not modify the vLLM source code.

Integration. The plugin is registered through vLLM's general-plugin entry point, so the engine and every worker process load it at startup. It registers a model class that subclasses vLLM's Qwen3-MoE implementation and is selected through `hf_overrides`, together with a loop configuration. The subclass replaces the forward pass of the inner model. After the weights are loaded, it detaches the router of every window layer from the fused MoE kernel, so that the kernel uses router logits supplied by the caller instead of recomputing them. This is how routing is frozen during iteration.

Forward pass. Layers before the window run unchanged, using vLLM's convention of carrying the hidden state and the residual stream as two separate tensors. At the window, the two are summed into the full residual state x, and the layers of the window are looped one at a time, which is layer-mode iteration (Algorithm 3). After the window, the full state is passed on as the hidden state with an empty residual, and the remaining layers and the final norm run unchanged.

Looping one layer. Each window layer is re-implemented on the full state: input norm, QKV projection, query and key norms, rotary embedding, attention through vLLM's paged-attention kernel, output projection, residual addition, post-attention norm, and the MoE block. The layer first runs one natural forward pass, which gives the anchor $g(x)$ and records its router logits, keys, and values. It then performs $K - 1$ damped Euler iterations with routing frozen to the natural router logits, and rescales the norm of the iterate by interpolation after each iteration. The output is $\beta\, g(x) + (1 - \beta)\, h$, where h is the final iterate, as in Equation (16). The blend is computed in `bfloat16`, since computing it in `float32` changes every generated sequence.

Key-value cache. Attention in every iteration writes the keys and values of the current tokens into their paged cache slots, overwriting the natural ones. After the iterations, the plugin writes the natural keys and values back into those slots. Later tokens therefore attend to the natural keys and values, which is the FIRST cache strategy of Appendix D, whereas the iterations within a prefill chunk attend to that chunk's iterated keys and values. All results in Table 4 use FIRST. Keeping the iterated keys and values instead lowered accuracy by 2.5 pp in our runs.

Execution. We disable `torch.compile` and capture each decode step as a full CUDA graph. Because K is fixed, the control flow of the loop is static and is captured directly. Operations that would synchronize the device are skipped while a graph is being captured. vLLM's fused add-RMSNorm kernel updates the hidden state and the residual in place, so every tensor that is reused in a later pass is cloned first.

Baseline. The baseline runs through the same plugin with $K = 1$, so it follows exactly the same full-state code path as the looped runs. We do not use stock vLLM as the baseline, because its outputs differ slightly in numerics from the full-state path.

I Robustness to window position across dense Qwen3 sizes

We sweep n=4 windows on the three Qwen3-Base sizes and count the windows with a positive 16-task aggregate (Table 11).

On Qwen3-4B-Base, 13 distinct windows spanning depth fractions 0.19–0.83 are simultaneously positive. Finding a useful window therefore does not require precise per-architecture tuning. This is consistent with the claim that the wrapper is robust on 87% of cells without per-cell hyperparameter search.

J Per-benchmark accuracy on Qwen3 models

Figure 5 shows per-benchmark accuracy with and without the loop wrapper for three Qwen3 variants.

Table 11: Number of n=4 windows with positive 16-task macro Δpp under canonical Euler K=2 block-mode looping. Many windows beat baseline on each size; the loss surface is broad.

Model	Layers	# positive n=4 wins
Qwen3-0.6B-Base	28	$\geq$ 6 of 13 tested
Qwen3-1.7B-Base	28	9 of 17 tested
Qwen3-4B-Base	36	13 of 16 tested

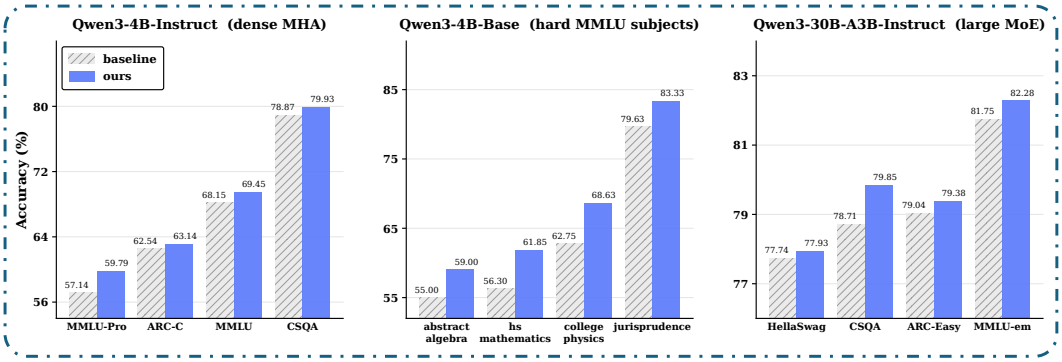


Figure 5: **Per-benchmark accuracy across three Qwen3 model variants under the training-free loop wrapper.** Each panel shows baseline (striped gray) vs. our wrapper (solid blue) on 4 knowledge-MC benchmarks. The per-panel y-axis is cropped to emphasize the baseline-to-loop gap. **Left:** Qwen3-4B-Instruct (dense MHA) on four mid-range general benchmarks. **Middle:** Qwen3-4B-Base on four hard MMLU 5-shot subjects, selected from MMLU's 57 subjects ("hs mathematics" is high-school mathematics, and MMLU-em is MMLU scored by exact match). **Right:** Qwen3-30B-A3B-Instruct (large MoE) on four mid-range benchmarks.