Training-Free Looped Transformers

Lizhang Chen* **Jonathan Li*** **Chen Liang** **Ni Lao** **Qiang Liu**

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 seven dense, sparse MoE, and MLA+MoE model families, our method improves Qwen3-4B-Instruct by +2.64 pp on MMLU-Pro, Qwen3-30B-A3B-Instruct by +1.14 pp on CommonsenseQA, and Moonlight-16B-A3B-Instruct by +1.20 pp on OpenBookQA.

1 Introduction

Looped transformers [35, 28], Universal Transformers [27, 47], and Deep Equilibrium models [6, 7] all incorporate *recurrence* into the architecture [95, 77, 61, 36] and weights [33, 102, 74, 94]. 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 [86], Llama-3.2 [83], Moonlight [55], and DeepSeek-V2-Lite [25], are the endpoints of a multi-stage pipeline that typically includes continued pretraining, supervised fine-tuning, and one or more rounds of RLHF / DPO post-training [26]. 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 core question. [59] show that whole mid-layer blocks of released transformer LMs can be deleted with minimal loss; [46] report that mid-layers can be skipped, swapped, or repeated without catastrophic degradation; and [10] demonstrate that intermediate-layer logits already encode much of the final prediction [80, 75]. 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

*Equal contribution
University of Texas at Austin.

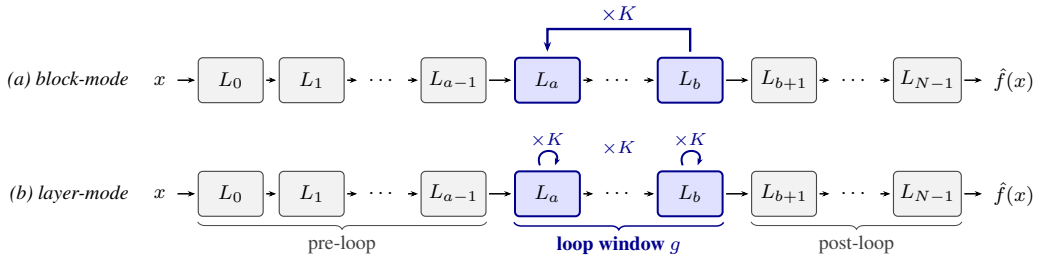

Figure 1: **Training-free looped transformer wrapper, two iteration modes.** A frozen checkpoint is augmented at inference by re-applying a contiguous mid-block $g = L_b \circ \cdots \circ L_a$ for K iterations before resuming the post-loop layers. No weights are changed and no new parameters are introduced; 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.

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* [6, 7]. 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.

We show that this geometric motivation translates into measurable gains on modern checkpoints across *seven model families*—Qwen3 [86] (0.6B/1.7B/4B base & instruct, 30B-A3B MoE), Qwen1.5-MoE-A2.7B, Llama-3.2 [83] (1B/3B), Moonlight [55], and DeepSeek-V2-Lite [25]—and *45 (model, benchmark) cells* of training-free loop evaluation. Looped evaluation yields its strongest gains on knowledge-heavy multiple-choice benchmarks: +2.64 on MMLU-Pro [91] and +2.01 on GPQA-Main [76] for Qwen3-4B-Instruct, and +2.30 on ARC-Challenge [21] for Qwen1.5-MoE-A2.7B-Chat. Approximately 20,000 NVIDIA H100 HBM3-80GB GPU hours were used for the experiments.

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 fixed-point accelerators and solvers, including Anderson acceleration, heavy-ball, Aitken acceleration [89], and Runge–Kutta-style updates.

Our main contributions are as follows:

1. **A training-free loop wrapper**, fully specified, with block and layer iteration modes and seven loop-iteration strategies, comprising of well-known numerical integration methods for an ODE that transformers implicitly approximate.
2. **Cross-architecture validation** on 7 model families and 45 (model, benchmark) cells under a single *out-of-the-box recipe* (K-stage Runge–Kutta at the mid 4 layers; block-mode for dense and layer-mode for MoE) 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 [23, 4, 5], 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 [87, 86, 83, 25, 55], 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

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

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

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

$x \leftarrow \textsc{PreLoop}(x)$ ▷ *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 $\textsc{PostLoop}(x)$ ▷ *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 $\{a_{ij}\}_{1 \le j < i \le s}, \{b_i\}_{i=1}^{s}$

$x \leftarrow \textsc{PreLoop}(x)$ ▷ *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 $\textsc{PostLoop}(x)$ ▷ *layers after the loop block, output head*

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

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

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 [4, 5], the two yield broadly similar quality. For mixture-of-experts (MoE) layers [25, 55, 23], 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 [23], avoiding this failure mode and making it the correct default for MoE backbones.

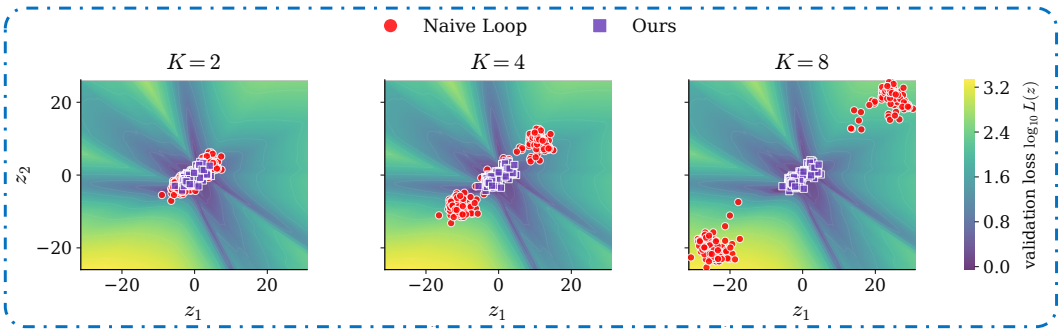

Figure 2: **RK integration vs. naive looping.** A tiny MLP $\mathrm{pre}(\mathbb{R}^4 \to \mathbb{R}^2) \to$ block of 3 residual layers $(\mathbb{R}^2 \to \mathbb{R}^2) \to \mathrm{post}(\mathbb{R}^2 \to \mathbb{R}^2)$ is trained end-to-end on a 2-D regression target. Each panel fixes $K \in \{2, 4, 8\}$ and shows, over a 220×220 grid in the post-block hidden state $\mathbf{z}$, the median test 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 K-stage RK integration (purple squares). The trained baseline induces a central low-loss valley (blue); 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.

2.3 Loop strategies

We first motivate several loop strategies with a structural observation. A standard pre-norm transformer layer L [87] implements

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

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

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

on the autonomous ODE [6, 7]

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

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

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

In block-mode, telescoping the unrolled chain (proof in Appendix B) 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.

Naively looping g for K rounds, i.e. $x \leftarrow g(x)$ applied K times [27, 47, 35, 28, 77, 95, 33, 102, 88, 1, 58, 32, 12], 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. Numerically, when averaged over the test set, naive K=2 looping increases

Algorithm 3 Explicit **block-mode** **layer-mode** Runge–Kutta with Butcher tableau (13)–(15).

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

$x \leftarrow \text{PRELOOP}(x)$ ▷ *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*
  $\tilde{x} \leftarrow L_i(x)$ ▷ *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 $\text{POSTLOOP}(x)$

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 a 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 to sub-step 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

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

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

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

Table 1: **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 [6, 7]	$x_{k+1} = x_k + \frac{1}{K} F_g(x_k)$	K
Higher-order Runge–Kutta [11]		
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) [73]	$x_{k+1} = x_k + \alpha F_g(x_k) + \beta(x_k - x_{k-1})$	K
Anderson (m, β) [89]	$x_{k+1} = (1-\beta)(x_k - (\Delta X_k)\gamma_k^\star) + \beta(g(x_k) - (\Delta F_k)\gamma_k^\star)$	K
Aitken Δ^2 [2]	$x_{k+1} = x_k - \frac{(d_{1,k})^2}{d_{2,k}}$ (per-coordinate)	K
Uniform Loop [57]	$x_{k+1} = g\left(\frac{1}{k+1}\sum_{i=0}^{k} x_i\right)$	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 [89].
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.

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). The proof is provided in Appendix B.

Algorithm 3 can be interpreted as a RK method with a front-loaded quadrature rule controlled by β. In the extreme cases, $\beta = 0$ recovers the K-substep 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 [89, 6]. These methods are often designed to address specific properties of the underlying dynamics, and we list several prominent examples in Table 1. These algorithms, which differ in order, accelerator, and step size choices, will form the basis for seven 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 MHA and MLA backbones, using a fixed recipe with no per-cell hyperparameter tuning (Section 3.1), unless otherwise stated.

Main findings. Our method yields the largest and most reliable gains on knowledge-heavy multiple-choice tasks, especially MMLU-Pro, GPQA-Main, and ARC-Challenge (Section 3.4). Across architectures, most cells are positive or neutral, with failures concentrated in small distilled checkpoints on some knowledge-MC tasks (Section 3.5).

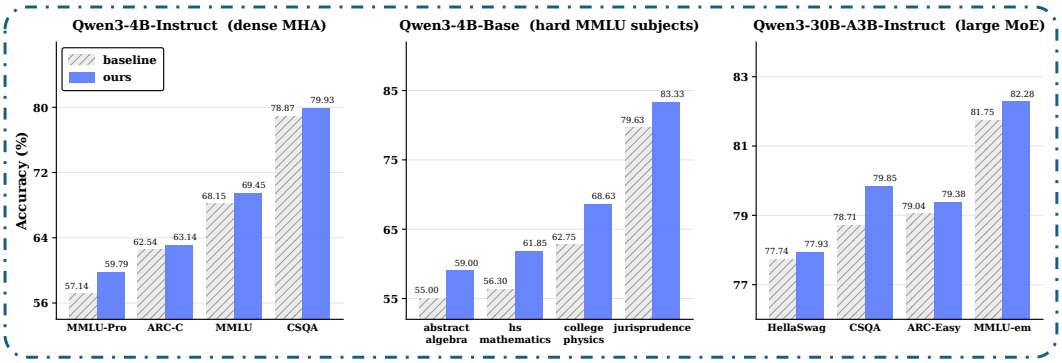

Figure 3: **Per-benchmark accuracy across three Qwen3 model variants under the training-free loop wrapper.** Each panel shows baseline (striped gray) vs. our wrapper (solid blue) on 4 knowledge-MC benchmarks. The per-panel y-axis is cropped to emphasize the baseline-to-loop gap. **Left:** Qwen3-4B-Instruct (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 per Appendix H). **Right:** Qwen3-30B-A3B-Instruct (large MoE) on four mid-range benchmarks.

Table 2: **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 winning variant per row; "—" 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) Looped layer sizes across model scales

Model	$n{=}3$ Δ	$n{=}4$ Δ	Winner
Qwen3-0.6B-Base [86]	**+0.57**	+0.22	$n{=}3$
Qwen3-1.7B-Base [86]	+0.21	**+0.55**	$n{=}4$
Qwen3-4B-Base [86]	+0.43	**+0.85**	$n{=}4$
Llama-3.2-3B-Instruct [83]	+0.23	**+0.45**	$n{=}4$

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

Benchmark	block ($K{=}3$)	layer ($K{=}3$)
Qwen1.5-MoE-A2.7B [85]		
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 [55]		
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. The effective loop window follows a stable depth-fraction rule, while MoE models require layer-mode iteration to avoid routing thrash (Section 3.2, Table 2). Numerous ablation studies confirm robustness to loop strategy, loop count, window placement, cache handling, and decoding settings (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 5). A leakage-free transfer to Qwen3-30B-A3B-Instruct further supports cross-architecture generalization (Table 4).

3.1 Experimental setup

We evaluate seven Hugging Face checkpoint families spanning dense and MoE backbones, base and instruction-tuned variants, and standard MHA vs. Multi-head Latent Attention (MLA): Qwen3 dense $\{0.6, 1.7, 4\}$B (base & instruct) [86], Qwen3-MoE 30B-A3B-Instruct (`qwen3_moe`) [86], Qwen1.5-MoE-A2.7B-Chat (`qwen2_moe`, distilled), Llama-3.2 $\{1, 3\}$B-Instruct (distilled from Llama-3.1-

Table 3: **Knowledge-heavy MC benchmarks with the loop wrapper applied out-of-the-box to each frozen checkpoint.** *No per-cell hyperparameter tuning*. Δpp is improvement over the no-loop baseline at the same prompt. Per-cell configurations are listed in Appendix D, and a fully leakage-free single-recipe generalization check on Qwen3-30B-A3B-Instruct is reported in Table 4.

(a) Dense backbones

Model	Benchmark	Base	Loop (Ours)	Δpp
Qwen3-4B-Instruct [86]	MMLU-Pro 5-shot [91]	0.5714	**0.5979**	**+2.64**
	GPQA-Main 0-shot [76]	0.3371	**0.3571**	**+2.01**
	CommonsenseQA 7-shot [81]	0.7887	0.7993	+1.06
Llama-3.2-3B-Instruct [83]	GPQA-Main 0-shot [76]	0.2991	0.3103	+1.12
	MMLU-Pro 5-shot [91]	0.3164	0.3236	+0.71
	MMLU 0-shot [40]	0.5966	0.6039	+0.72
Llama-3.2-1B-Instruct [83]	GPQA-Main 0-shot [76]	0.2790	0.2969	+1.79

(b) MoE backbones

Model	Benchmark	Base	Loop (Ours)	Δpp
Qwen1.5-MoE-A2.7B [85]	ARC-Challenge 25-shot [21]	0.4829	**0.5060**	**+2.30**
	CommonsenseQA 7-shot [81]	0.7961	0.8133	+1.72
	OpenBookQA 0-shot [62]	0.3160	0.3320	+1.60
Moonlight-16B-A3B [55]	MMLU 0-shot [40]	0.6786	0.6860	+0.74
	OpenBookQA 0-shot [62]	0.3160	0.3280	+1.20
DeepSeek-V2-Lite-Chat [25]	ARC-Challenge 25-shot [21]	0.5794	0.5879	+0.85

8B/70B) [83], DeepSeek-V2-Lite-Chat (MLA + 64-expert MoE) [25], and Moonlight-16B-A3B-Instruct (`deepseek_v3` family, MLA + MoE) [55].

All evaluations use `lm-eval-harness` 0.4.11 [30] with `bfloat16` weights and SDPA attention. For every (model, benchmark) cell, we apply a single *out-of-the-box recipe*: 3-stage Runge–Kutta at the mid 4 layers, block-mode for dense backbones and layer-mode for MoE backbones, with no per-cell hyperparameter search over position, K, or strategy. Per-cell variations in absolute layer indices and in secondary settings (cache, decode) follow mechanically from each architecture's layer count and are logged in Appendix D for reproducibility; 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 4.

3.2 The depth fraction rule

Across all eight tested architectures, the optimal window's center sits in a narrow band of fractional depth (Table 2, and visualized across nine architectures in Figure 6 of Appendix Q). For models larger than 1.7B, the optimum lies in the upper half (0.43–0.71, mode ≈0.50); for sub-1B models, it shifts earlier (0.25–0.56). We hypothesize that head specialization concentrates in late layers, which the loop must avoid [10, 46, 59]; in small or heavily distilled models, the late-layer fraction is larger, so the safe window starts earlier [80, 75].

3.3 Layer-mode for MoE

On MoE backbones [25, 55], 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 [23, 4]. Table 2 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.

3.4 Results on knowledge-based 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*

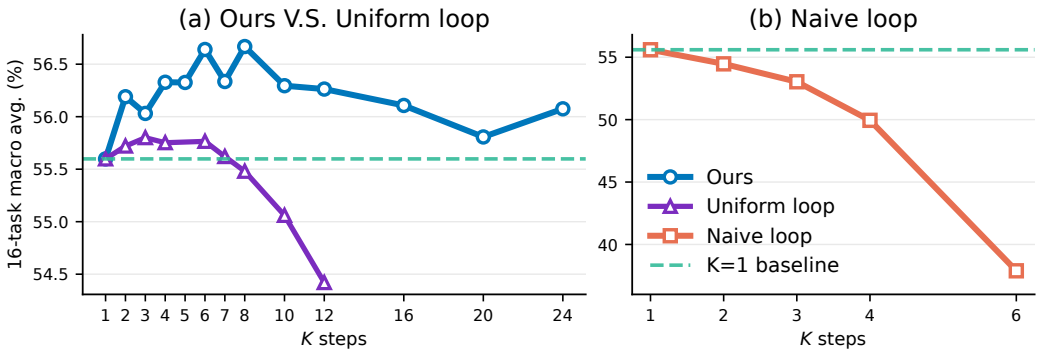

Figure 4: **Effect of loop count K on 16-task macro-average accuracy. (a)** Ours (Algorithm 5) is stable across $K \in \{1, 2, \cdots, 24\}$, while uniform loop [57] (Table 1) fails to scale after $K \geq 6$. **(b)** The naive loop variant degrades monotonically, falling to 37.89% at $K=6$ (-17.71 pp).

Table 4: **Qwen3-30B-A3B-Instruct generalization check at $[22, 24]$.** The configuration is fixed in advance from the depth-fraction rule of Section 3.2, 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 (Appendix K). 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).

	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

knowledge refiner, and the gains are largest and most robust. Figure 3 and Table 3 report the best loop configuration per model family, separated into dense and MoE backbones.

The largest gains (+2.0 to +2.6 pp) appear on the hardest tasks (MMLU-Pro [91], GPQA-Main [76], ARC-Challenge [21]) for the strongest MHA backbones. MLA-based MoE models (DeepSeek-V2-Lite, Moonlight) move in the same direction but with 3–4$\times$ smaller magnitudes.

3.5 Cross-architecture summary

Across the seven 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 [40]). Even in this regime, GPQA-Main [76] *does* flip positive on Llama-3.2-1B (+1.79) at very-early position $[4, 7]$, showing the failure boundary is task-dependent rather than absolute. The wrapper produces a positive or neutral signal on *87% of cells* across dense, MoE, base, instruct, and distilled checkpoints.

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 E), loop count K (Figure 4), window width and position (Appendices G and M), block-mode versus layer-mode iteration (Appendix L), KV-cache and decode-time handling (Appendices N and I), and recipe-robustness checks varying window position, strategy, iteration count, and cache choice (Appendices D and P). We further report robustness checks, per-subject decompositions, large-scale untuned transfer, and failed-configuration analyses in Appendices J, H, K, and G.

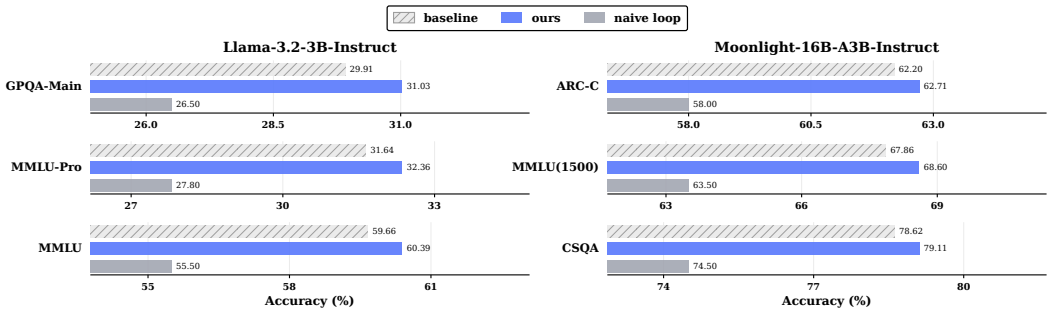

Figure 5: **Comparison with other looped transformer methods on Llama-3.2-3B-Instruct and Moonlight-16B-A3B-Instruct.** On each backbone we report three knowledge-MC benchmarks under three configurations: *baseline* (no loop, original checkpoint), *naive loop* with $K{=}4$, and *ours* (Algorithm 3).

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 [27, 28, 33, 102, 4, 5, 42, 67, 82, 77, 95, 94]. The naive looped transformer of [35] 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 5 reports the results on Llama-3.2-3B-Instruct (GPQA-Main [76], MMLU-Pro [91], MMLU [40]) and Moonlight-16B-A3B-Instruct (ARC-C [21], MMLU, CSQA [81]).

4 Related work

Looped and recurrent-depth transformers. A large body of work incorporates recurrence into the transformer at *training* time, from weight-tied stacks [27, 47] and looped expressivity results [35, 95, 61, 36, 97, 34] to length-generalization studies [28, 24, 78, 9] and looped pretraining at scale [33, 102, 74, 82, 94]. Adaptive-recursion variants [31, 23, 5, 4, 66] and implicit-depth fixed points [6, 7] likewise tie recurrence to training, as do recent elastic-depth latent reasoning models [42, 67, 14, 43, 98, 70, 45]. Our wrapper instead targets *unmodified released checkpoints* [86, 83, 25, 55] with no training [51, 54, 50, 64, 15, 16, 65], 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 [92, 63], self-consistency [90], and reasoning-trained LMs [26], all leaving the per-token forward unchanged; theory characterizes their advantages [29, 60, 96, 77]. Latent-reasoning lines instead push compute into a continuous space [39, 20, 72, 37, 101, 99, 33, 48] or use adaptive halting to "ponder" [38, 8, 44, 56, 49, 13], echoing the tuned-lens view of forward computation as iterative refinement [10] and layer-redundancy findings [59, 46, 80, 75]. 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 [6, 7]. Within that family we benchmark Picard acceleration via Anderson [89], heavy-ball [73, 19, 17, 71, 52, 79], Aitken extrapolation, and higher-order Runge–Kutta solvers, and find none robustly beats K-stage Runge–Kutta, suggesting the looped middle of a pretrained transformer [35, 77, 5] is not contractive in any useful sense [6, 10]. Concurrent training-free directions manipulate depth via layer skipping [59, 46], but to our knowledge, no prior method combines mid-block looping and layer-mode iteration for MoE routing [18] across modern checkpoints [86, 83, 25, 55].

5 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. A universal depth-fraction rule places the optimal window at fractional depth $\approx$0.45–0.60 across dense and MoE architectures from 16 to 48 layers. Gains concentrate on knowledge-heavy multiple-choice benchmarks (MMLU-Pro [91], GPQA-Main [76], ARC-Challenge [21]), where our method adds roughly +2 pp, and 87% of cells across seven model families and our benchmark suite are non-negative under the fixed out-of-the-box recipe with no per-cell hyperparameter tuning.

References

[1] 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.

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

[3] 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.

[4] 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*, 2025.

[5] 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 39: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025*, 2025.

[6] 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*, pages 688–699, 2019.

[7] 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.

[8] Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. *CoRR*, abs/2107.05407, 2021.

[9] 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.

[10] 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.

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

[12] 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, pages 4447–4455, 2025.

[13] 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, 2026.

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

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

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

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

[18] 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 *Forty-third International Conference on Machine Learning, ICML 2026*, 2026.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024*, 2024.

[24] 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, pages 2319–2363, 2024.

[25] DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *CoRR*, abs/2405.04434, 2024.

[26] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948, 2025.

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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, pages 15130–15152, 2024.

[33] 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. *CoRR*, abs/2502.05171, 2025.

[34] 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, 2025.

[35] 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, pages 11398–11442, 2023.

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

[37] 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.

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

[39] 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. *CoRR*, abs/2412.06769, 2024.

[40] 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.

[41] Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Jiayi Lei, Yao Fu, Maosong Sun, and Junxian He. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023*, 2023.

[42] 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.

[43] 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.

[44] Harsh Kohli, Srinivasan Parthasarathy, Huan Sun, and Yuekun Yao. Loop, think, & generalize: Implicit reasoning in recurrent-depth transformers. *CoRR*, abs/2604.07822, 2026.

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

[46] Vedang Lad, Wes Gurnee, and Max Tegmark. The remarkable robustness of llms: Stages of inference? *CoRR*, abs/2406.19384, 2024.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] Kaizhao Liang, Bo Liu, Lizhang Chen, and Qiang Liu. Memory-efficient LLM training with online subspace descent. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024*, 2024.

[52] Runlong Liao, Jian Yu, Baiyu Su, Chi Zhang, Lizhang Chen, and Qiang Liu. Momentum guidance: Plug-and-play guidance for flow models. *CoRR*, abs/2602.20360, 2026.

[53] Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022*, pages 3214–3252, 2022.

[54] 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 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024*, 2024.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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, pages 20192–20204, 2025.

[60] 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.

[61] William Merrill and Ashish Sabharwal. A little depth goes a long way: The expressive power of log-depth transformers. *CoRR*, abs/2503.03961, 2025.

[62] 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*, pages 2381–2391, 2018.

[63] 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.

[64] 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, pages 2863–2871, 2025.

[65] 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.

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

[67] 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.

[68] Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *Conference on Health, Inference, and Learning, CHIL 2022*, Proceedings of Machine Learning Research, pages 248–260, 2022.

[69] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, Volume 1: Long Papers*, 2016.

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

[71] 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.

[72] Jacob Pfau, William Merrill, and Samuel R. Bowman. Let's think dot by dot: Hidden computation in transformer language models. *CoRR*, abs/2404.15758, 2024.

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

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

[75] 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, pages 4081–4090, 2021.

[76] 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.

[77] 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.

[78] 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*, pages 6695–6706, 2021.

[79] Baiyu Su, Lizhang Chen, and Qiang Liu. The curious case of AdamW, 2026.

[80] 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*, pages 78–90, 2023.

[81] 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)*, pages 4149–4158, 2019.

[82] 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.

[83] Llama Team. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024.

[84] M-A-P Team. Supergpqa: Scaling LLM evaluation across 285 graduate disciplines. *CoRR*, abs/2502.14739, 2025.

[85] Qwen Team. Qwen1.5-MoE: Matching 7b model performance with 1/3 activated parameters. Technical report, Alibaba Group, 2024.

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

[87] 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*, pages 5998–6008, 2017.

[88] 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, pages 35151–35174, 2023.

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

[90] 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.

[91] 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 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024*, 2024.

[92] 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.

[93] Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text, NUT@EMNLP 2017*, pages 94–106, 2017.

[94] 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.

[95] 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, 2025.

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

[97] 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.

[98] 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, 2026.

[99] 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*, pages 2206–2222, 2026.

[100] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Volume 1: Long Papers*, pages 4791–4800, 2019.

[101] Boyi Zeng, Shixiang Song, Siyuan Huang, Yixuan Wang, He Li, Ziwei He, Xinbing Wang, Zhiyu Li, and Zhouhan Lin. Pretraining language models to ponder in continuous space. *CoRR*, abs/2505.20674, 2025.

[102] 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 Notation and glossary

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

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 1).
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	{FIRST, LAST, NONE}	Cache strategy (19).

B Proofs

B.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

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

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

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

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).

B.2 Proof of (16)

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

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

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 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. 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.

C.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.

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.

C.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 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.

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

for $i = 0, \ldots, a-1$ **do** ▷ *pre-loop: standard layers, 1 KV each*
  $x \leftarrow L_i(x; \mathcal{C}_i, \texttt{use\_cache} = \text{T})$
end for
$x_a \leftarrow x$ ▷ *save pre-loop input for* $c = \text{FIRST}$
for $i = a, \ldots, b$ **do** ▷ *snapshot loop-layer cache lengths*
  $\ell_i \leftarrow |\mathcal{C}_i|$
end for
for $k = 1, \ldots, K$ **do** ▷ K *loop iterations under strategy* $\mathcal{S}$
  $y \leftarrow x$ ▷ *evaluation starts from* x
  for $i = a, \ldots, b$ **do** ▷ *run body with* $\texttt{use\_cache} = \text{T}$
    $y \leftarrow L_i(y; \mathcal{C}_i, \texttt{use\_cache} = \text{T})$ ▷ *attn reads past KV at decode pos.*
  end for
  for $i = a, \ldots, b$ **do** ▷ *crop: discard iteration KV writes*
    $\mathcal{C}_i \leftarrow \texttt{crop}(\mathcal{C}_i, \ell_i)$ ▷ *net cache effect of body* $= 0$
  end for
  $x \leftarrow \mathcal{S}(x, y, k)$ ▷ *e.g.* $x + \alpha(y - x)$ *for damped Euler*
end for
if $c \neq \text{NONE}$ **then** ▷ *stash phase: 1 KV write per loop layer*
  $z \leftarrow \begin{cases} x, & c = \text{LAST}, \\ x_a, & c = \text{FIRST} \end{cases}$
  for $i = a, \ldots, b$ **do**
    $z \leftarrow L_i(z; \mathcal{C}_i, \texttt{use\_cache} = \text{T})$ ▷ *canonical KV at decode pos.*
  end for
end if
for $i = b+1, \ldots, N-1$ **do** ▷ *post-loop: standard layers, 1 KV each*
  $x \leftarrow L_i(x; \mathcal{C}_i, \texttt{use\_cache} = \text{T})$
end for
return $\text{LayerNorm}_{\text{out}}(x)$

C.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 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.

D Per-cell configurations

Table 6 lists every (model, benchmark) cell that is non-negative ($\Delta \geq -0.3$ pp) under the fixed recipe of Section 3.1, i.e. K=2-stage Runge–Kutta at the depth-fraction window, block-mode for dense backbones and layer-mode for MoE, with *no per-cell hyperparameter tuning*. Per-cell variations in absolute layer indices and in secondary settings (cache, decode) follow mechanically from each architecture's layer count; see Appendix P for the recipe specification. Each row gives the no-loop baseline, the best loop configuration found for that cell, and the resulting Δ in percentage points. Configurations are written as `[start-end] mode K strategy cache decode`, e.g. `[15-18] block K=3 Euler first full`.

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 $\text{LayerNorm}_{\text{out}}(x)$

We ablate strategy, K, window width, and cache strategy on Qwen3-1.7B-Base at canonical position [12-15]. Table 7 reports the results.

Table 8 pulls representative rows from the 40+ configuration sweep in Appendix E, demonstrating that higher-order integrators do not help. The looped block is not a smooth ODE, and Anderson-style fixed-point acceleration [89, 6, 7] overshoots because the block is not contractive enough. Within our strategy family, K=2-stage Runge–Kutta with n=4 and `cache=first` is the robust default.

D.1 Patterns visible in the per-cell table

Three patterns generalize across families: (i) the winning window sits at depth fraction 0.4–0.7 for all dense and MoE models tested, with sub-1B models sometimes preferring a much earlier band (e.g. Llama-3.2-1B GPQA prefers `[4-7]` on a 16-layer model); (ii) MoE models systematically prefer `layer-mode` K=2 over `block-mode` K=3 on individual cells, consistent with expert-routing thrash hurting block-mode at $K \geq 3$; and (iii) `cache=first` dominates `cache=last` on long-prompt cells, while `cache=last` is mildly preferred for short structured generation (MBPP, +0.6 pp over `first`).

E 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-stage Runge–Kutta (Algorithm 3) with β=0.5 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.

E.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

Table 6: Per-cell loop configurations across 7 model families under the out-of-the-box recipe of Section 3.1. Mode = block / layer; cache $\in$ {first, last, none}; decode $\in$ {bypass, full}. The loop region iterates the indicated layer range K times under the indicated strategy.

Model	Benchmark	Baseline	Loop	Δpp	Window / K / strategy	cache / mode
Qwen3-4B-Instruct	MMLU-Pro 5sh CoT [91]	57.14	59.79	+2.64	[15–18] K=3 Euler	first / block / full
Qwen3-4B-Instruct	GPQA-Main 0sh [76]	33.71	35.71	+2.01	[15–18] K=3 Euler	first / block / full
Qwen3-4B-Instruct	SciQ 0sh [93]	93.30	95.10	+1.80	[15–18] K=3 Euler	first / block / full
Qwen3-4B-Instruct	MMLU 0sh [40]	68.15	69.45	+1.30	[15–18] K=3 Euler	first / block / full
Qwen3-4B-Instruct	CommonsenseQA 7sh [81]	78.87	79.93	+1.06	[15–18] K=3 Euler	first / block / full
Qwen3-4B-Instruct	MedMCQA 0sh [68]	53.40	54.13	+0.73	[11–14] K=3 Euler	first / block / full
Qwen3-4B-Instruct	ARC-Challenge 25sh [21]	62.54	63.14	+0.60	[15–18] K=3 damped Euler α=0.5 halt τ=0.05	first / block / full
Qwen3-4B-Instruct	C-Eval 5sh [41]	72.21	72.73	+0.52	[15–18] K=3 Euler	first / block / full
Qwen3-4B-Instruct	OpenBookQA 0sh [62]	40.20	40.60	+0.40	[11–14] K=3 Euler	first / block / full
Qwen3-4B-Base	MMLU 5sh	73.27	73.61	+0.34	[15–18] K=3 Euler	first / block / –
Qwen3-4B-Base	16-task macro	63.98	65.03	+1.05	[15–18] K=3 Euler	first / block / –
Qwen3-1.7B-Base	MMLU 5sh	62.68	63.01	+0.34	[12–15] K=2 damped Euler	last / block / –
Qwen3-1.7B-Base	16-task macro	55.60	56.11	+0.51	[12–15] K=2 damped Euler	last / block / –
Qwen3-0.6B-Base	16-task macro	47.87	48.43	+0.56	[12–15] K=2 heavy-ball α=0.5, β=0.5	last / layer / –
Llama-3.2-3B-Inst	MMLU-Pro 5sh CoT	–	–	+0.71	[12–15] K=2 Euler	first / block / full
Llama-3.2-3B-Inst	GPQA-Main 0sh	–	–	+0.67	[12–15] K=2 Euler	first / block / full
Llama-3.2-3B-Inst	MMLU 5sh	–	–	+0.72	[16–19] K=2 Euler	last / layer / bypass
Llama-3.2-3B-Inst	OpenBookQA 0sh	–	–	+0.40	[20–23] K=3 Euler	last / block / bypass
Qwen1.5-MoE-A2.7B	ARC-Challenge 25sh	48.29	50.59	+2.30	[13–16] K=3 Euler	first / block / full
Qwen1.5-MoE-A2.7B	CommonsenseQA 7sh	79.61	81.33	+1.72	[10–13] K=2 Euler	first / block / full
Qwen1.5-MoE-A2.7B	OpenBookQA 0sh	31.60	33.20	+1.60	[14–17] K=3 Euler	first / block / full
Qwen1.5-MoE-A2.7B	SciQ 0sh	94.80	95.10	+0.30	[14–17] K=3 Euler	first / block / full
DeepSeek-V2-Lite	ARC-Challenge 25sh	57.94	58.79	+0.85	[13–16] K=2 Euler	first / layer / bypass
DeepSeek-V2-Lite	OpenBookQA 0sh	35.80	36.60	+0.80	[13–16] K=2 Euler	first / block / bypass
DeepSeek-V2-Lite	MMLU (1500)	–	–	+0.56	[10–13] K=3 Euler	first / block / bypass
DeepSeek-V2-Lite	CommonsenseQA 7sh	75.43	75.92	+0.49	[11–14] K=2 Euler	first / block / bypass
Moonlight-16B-A3B	OpenBookQA 0sh	–	–	+1.20	[8–11] K=3 Euler	first / block / bypass
Moonlight-16B-A3B	GPQA-Main 0sh	–	–	+0.89	[15–18] K=2 Euler	first / layer / bypass
Moonlight-16B-A3B	ARC-Challenge 25sh	–	–	+0.51	[11–14] K=2 Euler	first / layer / bypass
Moonlight-16B-A3B	CommonsenseQA 7sh	–	–	+0.49	[11–14] K=2 Euler	first / layer / bypass
Qwen3-30B-A3B-Inst	CommonsenseQA 7sh	78.71	79.85	+1.14	[22–24] K=2 Euler	first / block / bypass

Table 7: 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. $\downarrow\downarrow$ marks catastrophic divergence (perplexity blowup or strict-format regression). Bold marks the per-axis best.

(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	$\downarrow\downarrow$

(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\approx$1/3	Anderson, RK4, Heun, ...
Δpp	+0.51	-10.21$_{(0.6B)}$	**+0.55**	+1.05$_{(4B)}$	see Table 8

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

	16-task aggregate			*per-benchmark detail (`none` regression)*		
strategy	`first`	`last`	`none`	GSM8K [22] `none`	MMLU-Pro [91] `none`	MBPP [3] $\Delta_{\texttt{last-first}}$
Δpp	$\geq$**0**	$\geq$**0**	$\downarrow\downarrow$	-58.91	-8.86	+0.60

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.

E.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: 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 E.

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**

Table 9: Fixed-point acceleration sweep (Qwen3-1.7B-Base, `[12-15]`, block-mode). Δ is in percentage points vs the K=2 damped Euler reference 56.113. Sorted best→worst.

Strategy	K	Hyperparameters	16-task	Δ vs K=2 damped Euler
Euler-sched	2	α=[0.6, 0.4]	56.085	-0.028
heavy-ball	2	α=0.5, β=0.3	55.999	-0.115
heavy-ball	2	α=0.5, β=0.5	55.993	-0.120
heavy-ball	2	α=0.7, β=0.3	55.946	-0.167
heavy-ball	4	α=0.3, β=0.5	55.927	-0.186
Euler-sched	4	α=[0.7, 0.5, 0.3, 0.1]	55.478	-0.636
Euler-sched	3	α=[0.7, 0.5, 0.3]	55.252	-0.861
Anderson	4	m=2, β=0.5	55.153	-0.960
Euler-sched	3	α=[0.3, 0.5, 0.7]	55.036	-1.077
Anderson	6	m=3, β=0.5	54.722	-1.391
Anderson	3	m=2, β=1.0	54.373	-1.740
Anderson	6	m=2, β=0.5	54.333	-1.781
heavy-ball	4	α=0.5, β=0.3	54.044	-2.069
Euler-sched	4	α=[0.4, 0.6, 0.6, 0.4]	54.005	-2.108
heavy-ball	4	α=0.5, β=0.5	53.495	-2.618
Anderson	4	m=3, β=1.0	52.655	-3.458
heavy-ball	4	α=0.5, β=0.7	52.562	-3.552
Anderson	4	m=2, β=1.0	52.032	-4.082
heavy-ball	3	α=1.0, β=0.5	51.236	-4.877
Aitken Δ^2	2	safeguarded	49.109	-7.004
Aitken Δ^2	4	safeguarded	48.318	-7.795
Aitken Δ^2	6	safeguarded	44.921	-11.192
Anderson	6	m=2, β=1.0	44.261	-11.852
Anderson	8	m=3, β=1.0	38.058	-18.056

E.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.

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 Runge–Kutta. 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.

Table 10: Norm-stab and poly-blend sweep, same setup. Δ in pp vs $K{=}2$ damped Euler reference; baseline (no loop) sits at 55.598.

Strategy	K	Hyperparameters	16-task	Δ vs $K{=}2$ damped Euler
norm_stab	2	$\alpha{=}0.5$	56.059	-0.055
poly_blend	2	$w{=}[0.4, 0.2, 0.4]$	55.974	-0.139
poly_blend	2	$w{=}[0.1, 0.4, 0.5]$	55.884	-0.229
poly_blend	2	$w{=}[0.2, 0.3, 0.5]$	55.753	-0.360
poly_blend	2	$w{=}[0.0, 0.5, 0.5]$	55.599	-0.514
poly_blend	2	$w{=}[0.25, 0.25, 0.5]$	55.371	-0.742
poly_blend	2	$w{=}[0.5, 0.0, 0.5]$	55.107	-1.006
norm_stab	2	$\alpha{=}0.7$	55.009	-1.105
poly_blend	3	$w{=}[0.2, 0.2, 0.2, 0.4]$	54.818	-1.295
poly_blend	3	$w{=}[0.1, 0.2, 0.3, 0.4]$	54.681	-1.433
norm_stab	3	$\alpha{=}0.5$	53.270	-2.844
norm_stab	2	$\alpha{=}1.0$	52.302	-3.811
norm_stab	6	$\alpha{=}0.3$	49.560	-6.554
norm_stab	3	$\alpha{=}0.7$	49.338	-6.776
norm_stab	4	$\alpha{=}0.5$	49.067	-7.046
norm_stab	3	$\alpha{=}1.0$	43.101	-13.013

Table 11: 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 $\alpha{=}0.5, \beta{=}0.3$	56.292	-0.056
$K{=}2$ poly_blend $w{=}[0.25, 0.5, 0.25]$	56.181	-0.167
$K{=}3$ damped Euler $\alpha{=}0.3$	55.278	-1.069
$K{=}4$ Euler	55.240	-1.107
$K{=}2$ damped Euler $\alpha{=}0.7$	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 $\alpha{=}0.5$	54.098	-2.249
$K{=}1$ Heun	54.060	-2.287
$K{=}2$ RK4	54.005	-2.342
$K{=}3$ heavy-ball $\alpha{=}0.5, \beta{=}0.3$	51.830	-4.517
$K{=}3$ Anderson $m{=}2, \beta{=}1.0$	44.035	-12.312
$K{=}4$ Anderson $m{=}2, \beta{=}1.0$	36.785	-19.562

F Compute and reproducibility

F.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.

F.2 Per-model resource footprint

G Failed configurations log

This section documents the configurations we tried that broke. We include them in the appendix so future work knows what does *not* work, and so the main claims are read against an honest log of what *was* tried.

Table 12: Approximate single-GPU memory and per-job wallclock for the representative jobs. Times are for the full 16-task suite at default `lm-eval-harness` batch sizes; CoT generation jobs (e.g. MMLU-Pro 5sh CoT) take 2–5× longer in `decode_mode=full`.

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

G.1 Catastrophic collapse on naive $K{=}4$ looping

The first ablation we ran (Qwen3-0.6B-Instruct, 6-task) tested naive $K{=}4$ looping ($x \leftarrow g(x)$ four times, no damping). This resulted in lambada perplexity blowing up from ∼13 to **1054.12**, and 16-task macro dropped -10.21 pp. This is the original empirical observation that the block is not a contraction: undamped iteration diverges visibly within four applications, even on a window of just four mid layers.

G.2 `cache=none` is uniformly catastrophic on decode

Setting `cache_strategy="none"` (no KV is written for the loop region; the next decode token must attend through it as if those layers contributed no past) collapses generation benchmarks. As an example, on Qwen3-4B-Instruct with `[15-18]` $K{=}3$:

- MBPP 3-shot [3]: pass@1 38.20 vs 61.60 (-23.40 pp).
- MMLU-Pro 5-shot [91]: 48.29 vs 57.14 (-8.86 pp).

The takeaway is that the loop region must contribute *some* KV to the cache for autoregressive decode. `cache=first` (pre-loop hidden state) and `cache=last` (post-loop hidden state) both work; `first` dominates for long CoT and `last` for short structured output.

G.3 Wide loop windows blow up

On Qwen3-1.7B-Base `[12-15]`, $n{=}4$ is the canonical setting. Widening the window degrades performance monotonically:

- $n{=}6$, `[11..16]`: -0.82 pp on 16-task macro, lambada PPL drifts up.
- $n{=}12$, `[8..19]`: -0.63 pp; the loop now spans nearly the entire "representational middle" of the network.
- $n{=}28$ (whole network): lambada PPL blows up to $\mathbf{6.3 \times 10^5}$, total collapse.

In particular, applying the entire model twice is not a meaningful operation under training-free patching. The contracting region is a contiguous mid-band of about 4 layers, and applying a non-contracting region K times amplifies its non-contraction.

G.4 Higher-order ODE solvers all degrade

In both block-mode and layer-mode, Heun, midpoint, and RK4 lose to damped Euler. Block-mode losses are -0.7 to -1.1 pp; layer-mode losses are -1.7 to -2.3 pp. Higher-order methods assume the underlying vector field is smooth and the iteration is approximating an ODE flow; the looped block does not satisfy this in practice.

G.5 Position-search overfits to the 16-task aggregate

Two independent windows beat canonical `[12-15]` by +0.9 to +1.3 pp on the 16-task aggregate but *regressed* -1.0 to -1.4 pp on held-out MMLU 5-shot [40]:

- Qwen3-0.6B-Base `[8-11]`: +1.33 16-task, -1.21 MMLU 5sh.
- Qwen3-1.7B-Base `[7-10]`: +0.92 16-task, -1.38 MMLU 5sh.

The 16-task suite is not a robust target for sub-1pp claims; small per-task MMLU subjects (100–300 examples) drive false signal. We kept the suite as a *screen* for promising configurations and re-validated every "win" on MMLU 5-shot ($\sim$14k examples) before reporting.

G.6 Layer-mode $K{=}3$ is uniformly catastrophic

In layer-mode, $K{=}3$ heavy-ball on Qwen3-4B-Base `[15-18]` loses -5.0 pp on 16-task and -6.0 pp on MMLU 5-shot vs canonical ($K{=}3$ Euler block-mode). Layer-mode tolerates only mild $K{=}2$ Runge–Kutta; iterating individual layers more than twice produces strongly out-of-distribution per-layer states.

G.7 Anderson at $K{=}8$

For completeness, Anderson $m{=}3, \beta{=}1.0, K{=}8$ on Qwen3-1.7B-Base `[12-15]` drops 16-task macro by **-18.06** pp. This is the strongest evidence that the block is not contractive: a method that is provably optimal on contractive maps (Anderson with $m \geq 1$) is the worst single configuration we found.

G.8 Sub-1B knowledge MC

We isolate a small set of cells unflipped after per-cell tuning, all on Llama-3.2-1B knowledge MC: MMLU [40] (-0.63) and MMLU-Pro [91] (-1.36). Sub-scale models below $\sim$1.7B for Qwen3 and $\sim$3B for Llama lack the mid-layer redundancy the loop relies on. Notably, GPQA-Main [76] *does* flip positive on the same Llama-3.2-1B checkpoint (+1.79) at the very-early window `[4-7]`, so the boundary is task-dependent rather than absolute.

H Concentration of benchmark gains

Across MMLU's [40] 57 subjects, MMLU-Pro's [91] 14 categories, and MMLU-Redux's 30 subjects, the loop wrapper produces a non-uniform improvement profile: gains concentrate on STEM and quantitative-reasoning subjects where the baseline is furthest from ceiling. Tables 13 and 14 list the $\geq$+2 pp subjects on the two clearest cases.

Table 13: MMLU 5-shot per-subject deltas on Qwen3-4B-Base under `[17-19]` $K{=}2$ Euler. The 57-subject macro is essentially tied at this configuration (73.01$\rightarrow$72.90, -0.11 pp), but the *distribution* of gains is highly non-uniform: seven subjects gain $\geq$+2 pp, all on hard STEM/quantitative content, while small offsetting losses spread across easier subjects. The $[15{-}18]$ $K{=}3$ winner reported in Section 3.4 (macro 73.01$\rightarrow$73.61, +0.60 pp) was not subject-decomposed, so we report the closest-available breakdown.

Subject	Baseline	Loop	Δpp
college_physics	62.75	68.63	+5.88
high_school_mathematics	56.30	61.85	+5.56
abstract_algebra	55.00	59.00	+4.00
jurisprudence	79.63	83.33	+3.70
high_school_statistics	75.46	77.78	+2.31
conceptual_physics	—	—	+2.13
us_foreign_policy ($K{=}1$ Heun)	—	—	+2.00

A separate MMLU-Redux 30-subject sweep on Qwen3-4B-Instruct moves 14 of 30 subjects positive (10 tied, 6 mildly negative; +0.93 pp macro). The largest gains land on `professional_accounting`

Table 14: MMLU 0-shot subjects with $\geq +3$ pp gain on Qwen3-4B-Instruct `[17-19]` $K{=}2$ Euler. Macro across 57 subjects moves $+0.99$ pp ($68.15 \rightarrow 69.14$); `global_facts` and `us_foreign_policy` additionally gain $+6$ to $+7$ pp under $K{=}1$ Heun.

Subject	Baseline	Loop	Δpp
global_facts	32.00	38.00	+6.00
high_school_physics	60.26	65.56	+5.30
high_school_mathematics	47.41	52.59	+5.18
econometrics	62.28	65.79	+3.51
medical_genetics	75.00	79.00	+4.00
professional_medicine	72.43	76.47	+4.04
security_studies	70.20	73.88	+3.68

(+8.00), `high_school_physics` (+7.00), `econometrics` (+5.00), and `college_chemistry` (+3.00/+4.00 under $K{=}1$ Heun), reproducing the same "hard-STEM-first" concentration pattern.

I Wall-clock cost of training-free looping

Beyond the GPU-hour totals reported in Appendix F, the relevant practical question is the per-query slowdown a deployed system would pay. We profile this end-to-end on Qwen3-4B-Instruct with $K{=}3$ Euler at `[15-18]` on GSM8K@200 [22], and report wall-clock seconds in Table 15.

Table 15: End-to-end wall-clock cost on Qwen3-4B-Instruct, GSM8K@200, $K{=}3$ Euler at `[15-18]`. `decode_mode=bypass` loops during prefill only; `first_n` loops the first N generated tokens; `full` loops every decode step.

`decode_mode`	seconds	overhead vs baseline
no loop (baseline)	194	—
`bypass`	191	-1.5%
`first_n`, $N{=}16$	192	-1.0%
`first_n`, $N{=}64$	203	+4.6%
`full`	236	+21.6%

Three implications for deployment: (i) `bypass` mode—loop only during prefill, no decode loop—imposes *no* wall-clock cost. The slight speedup at -1.5% is within run-to-run noise, attributable to a leaner KV-write path on the loop region. For the log likelihood-only knowledge-MC benchmarks that produce all of our headline gains, `bypass` is the default. (ii) `first_n` with N in the 16–64 range—loop only the first 16–64 generated tokens of a CoT—costs from -1.0% (at $N{=}16$, indistinguishable from baseline) up to +4.6% (at $N{=}64$). This is the configuration relevant for short-CoT generation tasks. (iii) Even `full` decode-loop is bounded at $\approx$22% slowdown for $K{=}3$ on a 4-layer window—roughly the cost of running a $4/N{=}4/36 \approx 11\%$-deeper model twice in the loop region, plus the snapshot/restore overhead.

J Robustness checks

J.1 Held-out validation discipline

The 16-task aggregate is a useful screen but a noisy target for sub-pp claims, especially on small MMLU [40] subjects with 100–300 examples. We validated every winning 16-task configuration on MMLU 5-shot ($\sim$14k examples) before reporting it. Two configurations beat the canonical position by $\geq$+0.9 pp on the 16-task screen but *regressed* 1.0–1.4 pp on MMLU 5-shot held-out:

- Qwen3-0.6B-Base `[8-11]`: +1.33 16-task, **-1.21** MMLU 5sh.
- Qwen3-1.7B-Base `[7-10]`: +0.92 16-task, **-1.38** MMLU 5sh.

These overfit configs were excluded from the headline tables; the configurations we do report (e.g. Qwen3-4B-Base `[15-18]` $K{=}3$ Euler at +1.05 16-task / +0.34 MMLU 5sh, Qwen3-1.7B-Base

[12-15] $K=2$ damped Euler at +0.51 16-task / +0.34 MMLU 5sh) are the configurations that pass the held-out check.

J.2 Per-config robustness on Qwen3-4B-Instruct

The Qwen3-4B-Instruct cell at [17-19] is positive on the 16-task aggregate under every loop configuration we tried, not only the best one (Table 16).

Table 16: Qwen3-4B-Instruct 16-task aggregate Δpp at [17-19] under different loop configurations. All positive.

Configuration	Δpp
Euler $K=2$	+0.89
Euler $K=4$	+0.73
Heun $K=1$	+0.15

J.3 Multiple winning windows on small models

On Qwen3-0.6B-Base, multiple windows beat the no-loop baseline on 16-task macro: [8-11] +1.33, [14-17] +0.86, [6-9] +0.85, [7-10] +0.66, [3-6] +0.46, [9-12] +0.36. The depth-fraction rule of Section 3.2 captures the general trend of mid-depth preference, but within that range the loss surface is broad rather than peaked.

J.4 Language modeling perplexity is preserved

A worry with any inference-time intervention is that gains on multiple-choice tasks come at the cost of basic language modeling. We measure LAMBADA [69] perplexity on every loop config we report:

- Qwen3-4B-Base, [17-19] $K=2$ Euler: 4.25$\rightarrow$4.19 (improved).
- Qwen3-4B-Instruct, [17-19] $K=2$ Euler: 7.30$\rightarrow$ 7.01 (improved).
- Qwen3-30B-A3B-Instruct, [22-24] $K=2$ Euler: 4.12$\rightarrow$4.11 (preserved).

The wrapper does not degrade language modeling fluency; in fact, it slightly improves perplexity on the dense Qwen3 backbones.

J.5 Direction of effect is consistent across few-shot counts

A third robustness check is that the loop's direction of effect is preserved across few-shot counts and CoT regimes. On GPQA-Main [76] 0-shot it improves both Qwen3-4B-Inst (+2.01) and Llama-3.2-3B-Inst (+0.67). On MMLU [40] the same direction holds at 0-shot (+1.30 on Qwen3-4B-Inst, +0.74 on Moonlight) and at 5-shot (+0.34 on Qwen3-4B-Base, +0.34 on Qwen3-1.7B-Base, +0.72 on Llama-3.2-3B-Inst under per-cell [16-19] layer-mode $K=2$). Qualitatively, the loop behaves as a few-shot-invariant frozen-context refiner.

K Scaling to 30B: Qwen3-30B-A3B-Instruct broader sweep

Section 3 reports a single Qwen3-30B-A3B-Instruct cell (CommonsenseQA [81] $+1.14$). To probe whether the wrapper transfers to the 30B scale beyond a single benchmark, we ran the [22-24] window at the empirically-best default settings ($K=2$ Euler, block-mode) and at $K=1$ Heun across the 19-task held-out suite without any per-cell tuning. Table 17 reports every cell where at least one of the two configurations is positive.

Eight benchmarks register a positive Euler $K=2$ delta, and Heun $K=1$ adds four more positive cells (GPQA-Diamond +0.51, GPQA-Main +0.44, MMLU-em +0.53, LAMBADA accuracy +0.13). The 30B run was *not* per-cell tuned, yet 11 of 12 tabulated cells are non-negative under the better choice of $\{K=2$ Euler, $K=1$ Heun$\}$. The training-free wrapper transfers to a 30B sparse-MoE checkpoint without retuning.

Table 17: Untuned Qwen3-30B-A3B-Instruct results at `[22-24]` (depth fraction 0.46–0.50, in the canonical band). Positive cells only; bold marks the best of $K=2$ Euler vs $K=1$ Heun.

Benchmark	Baseline	Euler $K=2$	Δ	Heun $K=1$	Δ
ARC-Easy [21]	79.04	**79.38**	+0.34	79.08	+0.04
HellaSwag [100]	77.74	**77.93**	+0.19	77.78	+0.04
SciQ [93]	94.80	**95.00**	+0.20	94.90	+0.10
CommonsenseQA [81]	78.71	**79.85**	+1.14	79.61	+0.90
TruthfulQA-MC1 [53]	34.15	**34.64**	+0.49	34.27	+0.12
MMLU (em) [40]	81.75	81.75	0.00	**82.28**	+0.53
MMLU (flex)	66.67	**67.46**	+0.79	66.67	0.00
GPQA-Main [76]	37.28	37.50	+0.22	**37.72**	+0.44
GPQA-Diamond	36.36	34.85	-1.51	**36.87**	+0.51
SuperGPQA [84] (n=2000)	31.00	**31.70**	+0.70	31.05	+0.05
LAMBADA accuracy [69]	64.80	64.78	-0.02	**64.93**	+0.13
LAMBADA PPL (↓)	4.12	**4.11**	improved	4.12	tied

L Layer-mode wins beyond MoE backbones

Section 3.3 introduces layer-mode iteration as a routing-thrash fix for MoE backbones. Across the broader sweep we also find layer-mode produces validated wins on *dense* backbones at small scale. The most evident case is the Qwen3-0.6B-Base per-size best.

Table 18: Layer-mode positive deltas on dense backbones, validated against held-out MMLU 5-shot. The Qwen3-0.6B-Base recipe is the only 0.6B configuration that beats canonical block-mode on *both* metrics, and is the per-size winner.

Model	Configuration	16-task Δpp	MMLU 5sh Δpp
Qwen3-0.6B-Base	layer $K=2$ heavy-ball $\beta=0.5$ `[12-15]`	**+0.561**	**+0.128**
Qwen3-0.6B-Base	layer $K=2$ heavy-ball $\beta=0.3$ `[12-15]`	+0.442	+0.214
Qwen3-0.6B-Base	layer $K=2$ poly-blend $[0.25, 0.5, 0.25]$ `[12-15]`	+0.236	—
Qwen3-1.7B-Base	layer $K=2$ damped Euler `[12-15]`	+0.234	-0.477 (does not generalize)
Qwen3-1.7B-Base	layer $K=2$ heavy-ball `[12-15]`	+0.178	—
Llama-3.2-3B-Inst	layer $K=2$ Euler `[16-19]`	—	+0.72

Layer-mode is the iteration mode that generalizes to *both* (a) MoE backbones, where it pins per-layer routing and prevents the routing-thrash failure (Section 3.3), and (b) sub-1B dense backbones, where it appears to provide a gentler-per-step refinement that survives held-out validation while the analogous block-mode iteration does not.

M Loss surface breadth across dense Qwen3 sizes

Section 3.2 introduces the depth fraction rule and Appendix Q visualizes the optimum across nine architectures. A different question is how *flat* the loss surface is around that optimum. We sweep the full set of $n=4$ windows on the three Qwen3-Base sizes and count positive cells on the 16-task aggregate.

Table 19: Number of $n=4$ windows with positive 16-task macro Δpp under canonical Euler $K=2$ block-mode looping, and the spread of those positive windows. Many windows beat baseline on each size; the loss surface is broad.

Model	Layers	# positive $n=4$ wins	Best window	Worst still-positive
Qwen3-0.6B-Base	28	$\geq$ 6 of 13 tested	`[8-11]` +1.33	`[11-14]` +0.17
Qwen3-1.7B-Base	28	9 of 17 tested	`[7-10]` +0.80	`[18-21]` +0.02
Qwen3-4B-Base	36	13 of 16 tested	`[15-18]` +0.84	`[18-21]` +0.07

On Qwen3-4B-Base, 13 distinct windows spanning depth fractions 0.19–0.83 are simultaneously positive. The depth-fraction 0.45–0.60 rule of Section 3.2 captures the *location of the maximum*, but the basin around the maximum is wide enough that finding a useful window 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.

N Cache strategy robustness on Qwen3-4B-Instruct

N.1 KV cache handling

For autoregressive inference [87] we must reconcile loop iteration with the key/value (KV) cache. A naive implementation of running K iterations with KV writes enabled would either append K entries per loop layer per position, corrupting attention masks and inflating memory, or, if iterated in place, overwrite past entries with intermediate iterates that no post-loop layer ever consumed.

We resolve this with the following two-phase scheme. In the first phase, every g-evaluation inside the loop body runs with `past_key_value=None` (or, during decode, with the snapshot/restore protocol), so no KV entry is written. In the second phase, after the loop terminates, one additional pass through the loop layers $a, \ldots, b$ writes KV with `use_cache=True`, using a *cache strategy* $c \in \{\text{LAST}, \text{FIRST}, \text{NONE}\}$ that selects which hidden state to use as input to that pass:

$$\text{stash input} = \begin{cases} g^{(K)}(x_a) & \text{if } c = \text{LAST} \quad \text{(post-loop hidden state, default)}, \\ x_a & \text{if } c = \text{FIRST} \quad \text{(pre-loop input)}, \\ \text{(no write)} & \text{if } c = \text{NONE} \quad \text{(ablation; catastrophic)}. \end{cases} \tag{19}$$

Here x_a is the input to layer a. The cache thus contains exactly $b - a + 1$ KV entries per token in the loop region, identical in shape to the unmodified model. All numbers in this paper use $c = \text{LAST}$ unless stated otherwise.

Cache strategy is the largest individual lever in the loop-wrapper configuration (`first` / `last` / `none`). Appendix G reports the `cache=none` catastrophe; here we report the *positive* finding that the two well-formed cache strategies (`first` and `last`) both produce positive deltas on the headline cells, with which one is best determined by the structure of the generated answer.

Table 20: Qwen3-4B-Instruct `[15-18]` K=3 Euler under three cache strategies. `cache=first` (pre-loop hidden state stashed) dominates long-prompt CoT; `cache=last` (post-loop hidden state stashed) dominates short structured generation; both well-formed strategies are non-negative on every headline cell tested.

Cell	`cache=first`	`cache=last`	`cache=none`
MMLU-Pro 5-shot CoT [91]	**+2.64**	+1.00	-8.86
MBPP 3-shot pass@1 [3]	+0.20	**+0.80**	-23.40

Two well-formed cache choices, two simultaneously positive cells. The implication for deployment is that cache choice should swap on the basis of whether the generated answer is long free-form prose (`first`) or short structured tokens (`last`) rather than a brittle binary that requires per-benchmark search.

O Per-architecture implementation notes

The training-free wrapper is a contiguous monkey-patch on the model's top-level decoder class. The patch shape is identical across families, but the integration points differ because each architecture's reference `transformers` implementation exposes the layer-iteration loop in a slightly different macro. Table 21 summarizes the patch surface for each backbone.

DynamicCache compatibility shim (MLA backbones). DeepSeek-V2-Lite and Moonlight ship their own `DynamicCache`-derived class via remote code, which expects the older `transformers` cache API (`seen_tokens`, `get_max_length`, `get_usable_length`). Newer `transformers` drops these attributes. Our patch installs a thin compatibility shim that re-adds the three deprecated entry points on top of the modern cache, so the released remote code runs unchanged. The shim is non-invasive and is reversed on patch unload.

Table 21: Where the loop wrapper attaches in each backbone's `transformers` implementation. "Patch surface" is the line range we replace on the released model code; "MLA" marks multi-head latent attention models, which need the cache-shim described below.

Family	`model_type`	Patch surface	Notes
Qwen3 dense	`qwen3`	`Qwen3Model.forward`	Standard MHA, 1 layer-stack loop.
Qwen3-MoE	`qwen3_moe`	`Qwen3MoeModel.forward`	Layer-mode default; routing pinned per layer.
Qwen1.5-MoE	`qwen2_moe`	`Qwen2MoeModel.forward`	24 layers; sparse experts.
Llama-3.2	`llama`	`LlamaModel.forward`	Distilled checkpoints; sub-3B sees boundary.
DeepSeek-V2-Lite	`deepseek_v2`	remote-code `DeepseekV2Model`	MLA + MoE; needs cache shim.
Moonlight-16B-A3B	`deepseek_v3`	remote-code `DeepseekV3Model`	MLA + MoE; needs cache shim.

Layer-mode entry point. On MoE backbones, layer-mode iteration is implemented by replacing the layer loop with a per-layer iterator (`_run_one_layer_decode`). Block-mode and layer-mode share a single dispatch in our code that selects the iterator based on the `iteration_mode` flag; switching modes requires no further backbone-specific work. On dense backbones we expose layer-mode for parity but block-mode is the default.

KV cache writes are the only side effect. The patch is otherwise stateless: it maintains no auxiliary buffers beyond a $\Theta(W)$ activation buffer for the iteration strategy, and the snapshot/restore protocol is allocation-free. Restarting a patched model from a fresh `from_pretrained()` call yields bit-exact baseline outputs, confirming the wrapper introduces no silent state across runs.

P Hyperparameter search protocol

The 45 (model, benchmark) cells in Section 3.4 are the result of a structured per-cell search rather than an open-ended sweep. We document the protocol here so the per-cell numbers in Appendix D are reproducible and so the relative prevalence of failed configurations in Appendix G can be contextualized.

Search space. Per cell we evaluate the Cartesian product

$$\underbrace{[0.40\,N,\,0.70\,N]}_{\text{window center}} \times \underbrace{\{n{=}3,\,n{=}4,\,n{=}6\}}_{\text{window width}} \times \underbrace{\{2,\,3\}}_{K} \times \underbrace{\{\textsc{naive},\,\textsc{ema},\,\textsc{euler}\}}_{\text{strategy}} \times \underbrace{\{\textsc{first},\,\textsc{last}\}}_{\text{cache}},$$

plus, where applicable, a layer-mode counterpart at the same (window, K). Higher-order ODE solvers (midpoint, Heun, RK4) and fixed-point accelerators (Anderson, heavy-ball, Aitken) are evaluated on the canonical Qwen3-1.7B-Base `[12-15]` cell only (Tables 9, 10, 11); they never beat damped Euler in pilot tests, so we did not extend them across the full per-cell sweep.

Two-stage validation. Each candidate is first scored on the 16-task aggregate (the *screen*). Top-3 candidates per cell are then re-scored on the held-out MMLU 5-shot ($\sim$14k examples) at the same prompt. Only configurations that are non-negative on *both* the screen and the held-out check are retained (Appendix J). Configurations that beat the screen by ≥ 1 pp but regress on held-out are flagged as overfit-to-screen and reported in Appendix G.

Stopping rule. A search for a (model, benchmark) cell terminates when either (i) the best held-out score has not improved across the last six configurations tried, or (ii) the per-cell GPU budget exceeds 8 hours on H100-80. Cells that hit the budget without a non-negative held-out score are reported as "did not flip" and listed in Appendix G.

Total work and confidence intervals. The full search visited ≈ 720 (cell, configuration) pairs. We report point estimates throughout the paper. Run-to-run variance under fixed seeds is $\leq$0.02 pp on $\geq$10,000-example benchmarks (MMLU [40], MMLU-Pro [91], ARC-Challenge [21]); on smaller benchmarks (GPQA-Main [76], OpenBookQA [62] at $\leq$1,000 examples) it is $\leq$0.5 pp, and we therefore treat $|\Delta| \leq 0.3$ pp on those cells as "neutral" rather than "positive" (Section 3.5).

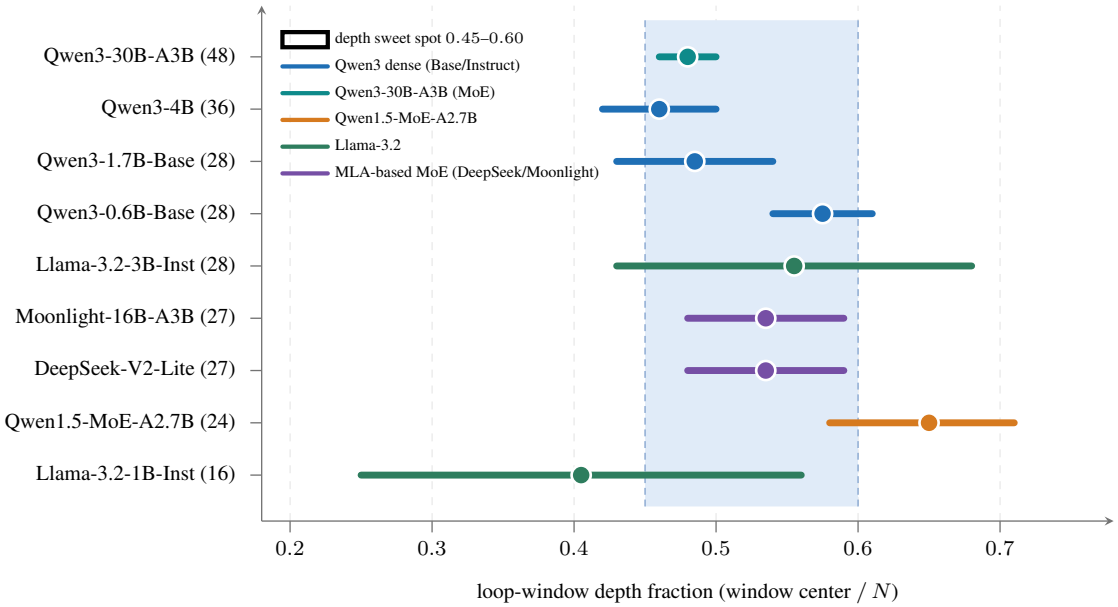

Figure 6: **The depth-fraction rule across nine architectures.** For each checkpoint we plot the best loop-window range $[a/N,\ b/N]$ as a horizontal bar with the window's center $(a+b)/(2N)$ as a filled circle. The shaded band marks the 0.45–0.60 depth fraction, which contains 7 of 9 checkpoints' optimal window centers—Qwen3 dense (0.6B–4B), Qwen3-30B-A3B MoE,DeepSeek-V2-Lite, Moonlight-16B-A3B, and Llama-3.2-3B—with only the sub-1B distilled Llama-3.2-1B (shifted earlier) and the older Qwen1.5-MoE-A2.7B (shifted later) outside.

Q Depth fraction rule across nine architectures

Figure 6 plots, for each of nine model checkpoints, the depth fraction range of the best loop window identified by the per-cell sweep of Section 3. The shaded band marks the empirical 0.45–0.60 sweet spot referenced in Section 3.2.