SCHEDULING RECURSIVE REASONING IN LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recurrent reasoning models have attracted growing attention for scaling test-time computation, typically by iteratively refining latent states with shared parameters. However, these models apply each learned update with a fixed unit scale, which can be conservative when updates make persistent progress and overly aggressive when they fluctuate, limiting the benefit of additional loops. To understand how the scale should vary along the trajectory, we first analyze the sensitivity of terminal loss to recurrent update scale. We show that its temporal average admits an exact decomposition into *persistent-progress* and *centered-fluctuation* contributions. Based on this, we introduce the **Trajectory Adaptive Progress–Fluctuation Scheduler (TAPS)**, which tracks their balance across recurrent updates and adapts the step size online. Theoretically, we establish sufficient conditions under which TAPS reduces expected terminal loss and reaches a target quality in fewer recurrent loops. Empirically, we show that TAPS improves terminal accuracy across structured reasoning tasks without retraining. By further incorporating the progress–fluctuation principle into training, TAPS yields additional accuracy gains with up to $1.56\times$ wall-clock speedup at matched baseline accuracy. The broad applicability of TAPS is supported by its effectiveness across diverse recurrent architectures and inference strategies. Together, these results establish update scale as a complementary control axis of recurrent inference alongside architecture and depth.

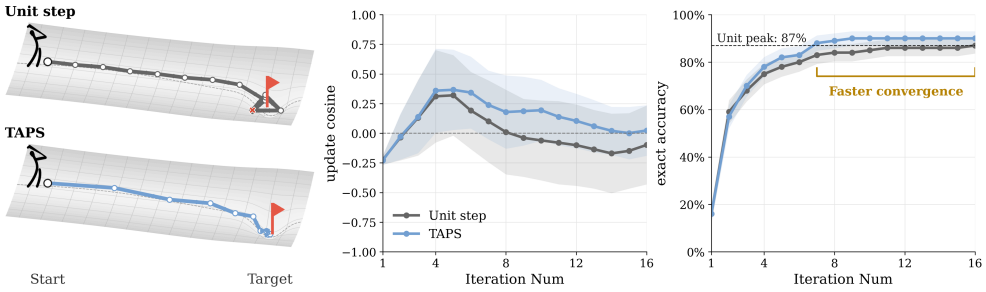


Figure 1: **Overview of TAPS.** (a) Adaptive step sizes reshape recurrent trajectories by modulating update magnitude at inference time. (b) TAPS maintains more consistent update directions than unit-step by balancing progress and fluctuation across iterations. (c) This improved trajectory accelerates recurrent convergence, reaching comparable performance with fewer iterations.

1 INTRODUCTION

Recurrent reasoning models scale test-time computation by repeatedly applying the same transformation to an evolving latent state (Dehghani et al., 2018; Yang et al., 2024; Zhu et al., 2025; Geiping et al., 2025). By reusing parameters across iterations, recurrence decouples effective reasoning depth from parameter count. This principle has been explored through layer recurrence (Gao et al., 2026; Nguyen & Lin, 2025), MoE recurrence (Chen et al., 2026b; Lee et al., 2026), and full-model recurrence (Jolicoeur-Martineau, 2025; Saunshi et al., 2025). Existing work has mainly focused on what computation is repeated and how many times it is applied, while the scale of each recurrent

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

update has largely remained unexplored. We study update scale as a complementary control axis alongside recurrent architecture and depth.

This perspective raises a natural question: how far should the state move at each recurrent step? Standard recurrent inference typically applies unit updates throughout the trajectory, although different stages may favor different scales. A unit step can be conservative when updates are persistent and aggressive when they fluctuate. Ideally, each scale would be chosen according to its effect on the terminal task loss, but this signal is unavailable at inference time. This leaves the evolving recurrent trajectory as a natural source of test-time information. We therefore ask whether this trajectory contains enough information to adapt the update scale and improve task performance.

In this paper, we introduce **Trajectory Adaptive Progress–Fluctuation Scheduler (TAPS)**, which adapts recurrent update scale online using only observed trajectory. We analyze the sensitivity of the terminal loss to recurrent update scale and show that its temporal average can be decomposed as $\bar{A}_k^w = P_k^w - S_k^w$, where P_k^w captures *persistent progress* along the recurrent trajectory and S_k^w captures *centered fluctuations*. This decomposition suggests a simple principle: take larger steps when updates make consistent progress and smaller steps when fluctuations dominate. TAPS operationalizes this principle by quantifying persistence and fluctuation across recurrent updates and adapting η_k according to their balance. Our design is inspired by the broader use of trajectory information for step-size adaptation in optimization (Barzilai & Borwein, 1988; Malitsky & Mishchenko, 2019; Vladarean et al., 2021).

The remaining issue is whether the trajectory signal is aligned with task performance and translates into practical inference gains. Our analysis gives sufficient conditions under which the persistence-fluctuation balance guides scale adjustment to reduce terminal loss. We further show how the resulting local gains can reduce the recurrent computation required to reach a target performance. Empirically, TAPS improves terminal accuracy on Sudoku and Maze without retraining (Table 1). Incorporating the same progress–fluctuation principle during training further strengthens these gains, improving accuracy and achieving up to $1.56\times$ wall-clock speedup at matched performance. **Beyond these primary settings, we demonstrate the broader applicability of TAPS across recurrent architectures and inference strategies.** Across inference strategies, TAPS combines with adaptive exit, hierarchical recurrence, fixed-point inference, and parallel recurrent computation (Figure 2 and Tables 2, 9, 10). Across architectures, it transfers to latent-state recurrence, recurrent language models, and intermediate-layer recurrence (Tables 1, 3 and 4).

Together, these results support a broader view of recurrent inference: architecture determines what computation is repeated, depth determines how long it is repeated, and update scale determines how strongly each step is applied. TAPS makes the third dimension adaptive to the evolving recurrent trajectory, providing a complementary mechanism for controlling recurrent computation.

2 Problem Setup

We consider data $(u, y) \sim P$, where $u \in \mathcal{U}$ is the model input and $y \in \mathcal{Y}$ is the corresponding target. A pretrained looped model with frozen parameters θ processes u by repeatedly refining a latent representation in $\mathcal{H} := \mathbb{R}^{n\times d}$. That is, the encoder $E_\theta : \mathcal{U} \to \mathcal{H}$ produces the initial state $\mathbf{X}_0 := E_\theta(u)$, and the same recurrent core $\Phi_\theta : \mathcal{H} \times \mathcal{U} \to \mathcal{H}$ is applied at each loop:

$$\mathbf{X}_{k+1} = \Phi_\theta(\mathbf{X}_k; u), \quad k = 0, 1, \dots.$$

The learned update at a latent state $\mathbf{X} \in \mathcal{H}$ is defined by $\mathbf{\Delta}_\theta(\mathbf{X}; u) := \Phi_\theta(\mathbf{X}; u) - \mathbf{X}$. Therefore, the standard loop can equivalently be written as

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \mathbf{\Delta}_\theta(\mathbf{X}_k; u),$$

which applies the learned update with unit scale. In Section B.2, a motivating example shows that such a constant update may not be satisfactory. In this paper, we expose such implicit unit-scale choice and study whether the update scale can be adaptively controlled across loops to improve inference efficiency without sacrificing task performance.

Specifically, for a fixed horizon $K \geq 1$, we replace the unit update by a vector $\boldsymbol{\eta} := (\eta_0, \dots, \eta_{K-1}) \in [\eta_{\min}, \eta_{\max}]^K$ with $0 < \eta_{\min} \leq 1 \leq \eta_{\max}$, which generates the scheduled trajectory

$$\mathbf{X}_0^{\boldsymbol{\eta}} := \mathbf{X}_0, \quad \mathbf{X}_{k+1}^{\boldsymbol{\eta}} := \mathbf{X}_k^{\boldsymbol{\eta}} + \eta_k \mathbf{\Delta}_\theta(\mathbf{X}_k^{\boldsymbol{\eta}}; u), \quad k = 0, \dots, K-1.$$

Here, η_k is the relaxation factor at the k-th loop: $\eta_k = 1$ recovers the standard loop, while $\eta_k > 1$ amplifies the learned update and $\eta_k < 1$ damps it. To measure the quality of a trajectory, let $R_\theta : \mathcal{H} \to \mathbb{R}^p$ denote the frozen readout that maps a latent state to the model output, and let $\ell : \mathbb{R}^p \times \mathcal{Y} \to \mathbb{R}_+$ denote the loss function. Then the task loss associated with latent state $\mathbf{X}$ is $L(\mathbf{X}; y) := \ell(R_\theta(\mathbf{X}), y)$. Since the target y is unavailable at inference time, we therefore consider an adaptive scheduling policy π that selects η_k using only the input u and information available up to loop k, without access to y or future states. Let $\mathbf{X}_K^\pi(u)$ denote the final latent state after K loops. The corresponding population risk is defined as

$$J_K(\pi) := \mathbb{E}_{(u,y)\sim P}\left[L\left(\mathbf{X}_K^\pi(u); y\right)\right].$$

For a target tolerance level $\epsilon > 0$, define

$$K_\varepsilon(\pi) := \inf\left\{K \geq 1 : J_K(\pi) \leq \varepsilon\right\},$$

which represents the number of loops required by policy π to attain the target performance. Let π_0 be the standard policy with $\eta_k \equiv 1$. Our goal is to develop an adaptive scheduling policy π such that $K_\varepsilon(\pi) < K_\varepsilon(\pi_0)$, thereby reaching the target performance with fewer loops than standard inference.

Notations. The Frobenius inner product of $\mathbf{A}, \mathbf{B} \in \mathcal{H}$ is denoted by $\langle \mathbf{A}, \mathbf{B}\rangle_F = \mathrm{tr}(\mathbf{A}^\top \mathbf{B})$. The norm of $\mathbf{A}$ is $\|\mathbf{A}\|_F = \sqrt{\langle \mathbf{A}, \mathbf{A}\rangle_F}$. For a trajectory $\{\mathbf{X}_k^{\boldsymbol{\eta}}\}$, write $\boldsymbol{\Delta}_k^{\boldsymbol{\eta}} := \boldsymbol{\Delta}_\theta(\mathbf{X}_k^{\boldsymbol{\eta}}; u)$. When the $\boldsymbol{\eta}$ is clear from context, we write $\mathbf{X}_k$ and $\boldsymbol{\Delta}_k$ for simplicity. For a scalar-valued function, $\nabla_{\mathbf{X}}$ denotes its gradient with respect to $\mathbf{X}$ under the Frobenius inner product.

3 The Impact of Scheduling: A decomposition of the Gradient

Before introducing TAPS, we first explore the impact of scheduling on model performance. We show that the temporally averaged gradient of the terminal task loss with respect to η can be decomposed exactly into two parts, namely the *persistent progress* and *centered fluctuations*.

Specifically, consider a fixed example (u, y), a horizon $K > 0$, and a schedule $\boldsymbol{\eta}$. Write $F(\mathbf{X}, \eta) := \mathbf{X} + \eta\boldsymbol{\Delta}_\theta(\mathbf{X}; u)$. We use $D_{\mathbf{X}}$ to denote differentiation with respect to $\mathbf{X}$. At loop j, we define the state Jacobian by

$$D_j := D_{\mathbf{X}} F(\mathbf{X}_j, \eta_j) = I + \eta_j D_{\mathbf{X}} \boldsymbol{\Delta}_\theta(\mathbf{X}_j; u), \tag{1}$$

where $D_j : \mathcal{H} \to \mathcal{H}$ describes the first-order propagation of a perturbation in $\mathbf{X}_j$ to $\mathbf{X}_{j+1}$. Let D_j^* denote the adjoint of D_j, characterized by $\langle D_j \mathbf{H}, \mathbf{Z}\rangle_F = \langle \mathbf{H}, D_j^* \mathbf{Z}\rangle_F$, $\mathbf{H}, \mathbf{Z} \in \mathcal{H}$. Set $\mathbf{G}_K := \nabla_{\mathbf{X}} L(\mathbf{X}_K; y)$ and propagate it backward via $\mathbf{G}_j := D_j^* \mathbf{G}_{j+1}$, $j = K-1, \ldots, 0$. The following proposition characterizes how each relaxation factor η_k affects the task loss at a fixed horizon K.

Proposition 1. *Suppose $\boldsymbol{\Delta}_\theta(\cdot; u)$ and $L(\cdot; y)$ are continuously differentiable along the realized trajectory. Then, for every $k < K$, we have*

$$\frac{\partial L(\mathbf{X}_K; y)}{\partial \eta_k} = \langle \mathbf{G}_{k+1}, \boldsymbol{\Delta}_k\rangle_F.$$

Let $A_k := -\langle \mathbf{G}_{k+1}, \boldsymbol{\Delta}_k\rangle_F$. By Proposition 1, to minimize the final loss, $A_k > 0$ favors increasing η_k while $A_k < 0$ favors decreasing it. However, A_k depends on the target y and the downstream gradient $\mathbf{G}_{k+1}$, and is therefore unavailable for designing η_k.

We next consider the aggregation of A_k over a temporal window, which connects this task-relevant quantity to temporal trajectory information and also provides insight into constructing an estimator of A_k. For any weights $w_{k,j} \geq 0$ with $\sum_{j=0}^k w_{k,j} = 1$, we define $\bar{A}_k^w := \sum_{j=0}^k w_{k,j} A_j$. Let $\bar{\mathbf{G}}_k^w := \sum_{j=0}^k w_{k,j} \mathbf{G}_{j+1}$ and $\bar{\boldsymbol{\Delta}}_k^w := \sum_{j=0}^k w_{k,j} \boldsymbol{\Delta}_j$. The persistent and fluctuation contributions are defined as

$$P_k^w := -\left\langle \bar{\mathbf{G}}_k^w, \bar{\boldsymbol{\Delta}}_k^w \right\rangle_F, \quad S_k^w := \sum_{j=0}^k w_{k,j} \left\langle \mathbf{G}_{j+1} - \bar{\mathbf{G}}_k^w, \boldsymbol{\Delta}_j - \bar{\boldsymbol{\Delta}}_k^w \right\rangle_F.$$

The following proposition provides an exact decomposition of $\bar{A}_k^w$.

Proposition 2. *For every normalized nonnegative weight w, it holds that $\bar{A}_k^w = P_k^w - S_k^w$.*

The term P_k^w captures the contribution of persistent temporal motion, while S_k^w captures that of centered fluctuations. Thus, a positive P_k^w increases $\bar{A}_k^w$, whereas a positive S_k^w decreases it. Their signs are not fixed in general; Section 4 identifies conditions under which these quantities can be well approximated. We note that $\bar{A}_k^w$ has an interpretation in terms of the terminal task loss; see Remark 1.

Remark 1 (Sensitivity Interpretation of $\bar{A}_k^w$). Consider the perturbed schedule $\boldsymbol{\eta}(t)$ with $\eta_j(t) = \eta_j + t w_{k,j}$ for $j \le k$ and $\eta_j(t) = \eta_j$ otherwise. By the chain rule, $-\frac{\mathrm{d}}{\mathrm{d}t} L(\mathbf{X}_K^{\boldsymbol{\eta}(t)}; y)|_{t=0} = \bar{A}_k^w$. Thus, $\bar{A}_k^w$ gives the first-order terminal-loss advantage of this weighted relaxation perturbation.

4 TAPS: Trajectory-Adaptive Progress–Fluctuation Scheduling

This section introduces TAPS. Section 4.1 first shows the construction of the pseudo estimates of P_k and S_k based on the temporal statistics of the learned latent updates, and then introduces the scheduling policy π. Section 4.2 establishes conditions under which TAPS identifies the task-improving direction and can lead to better performance.

4.1 Methodology

At loop k, TAPS summarizes updates $\{\boldsymbol{\Delta}_j\}_{j \le k}$ using exponential moving averages (EMA). Let $\beta \in [0, 1)$ control the temporal memory. Starting from $\mathbf{M}_{-1} := \mathbf{0} \in \mathbb{R}^{n \times d}$ and $v_{-1} = 0$, define

$$\mathbf{M}_k = \frac{\beta - \beta^{k+1}}{1 - \beta^{k+1}} \mathbf{M}_{k-1} + \frac{(1-\beta)}{1 - \beta^{k+1}} \boldsymbol{\Delta}_k, \quad v_k = \frac{\beta - \beta^{k+1}}{1 - \beta^{k+1}} v_{k-1} + \frac{1 - \beta}{nd(1 - \beta^{k+1})} \|\boldsymbol{\Delta}_k\|_F^2 .$$

The observable persistent and fluctuation energies are then defined by

$$\widehat{P}_k := \frac{1}{nd} \|\mathbf{M}_k\|_F^2 , \quad \widehat{S}_k := v_k - \widehat{P}_k,$$

To see their temporal explanation, define the normalized EMA weights by $w_{k,j} := (1-\beta)\beta^{k-j}/(1-\beta^{k+1})$ with the convention of $0^0 = 1$. By Lemma 6 in Appendix B, we have

$$\mathbf{M}_k = \sum_{j=0}^{k} w_{k,j} \boldsymbol{\Delta}_j, \quad \widehat{P}_k = \frac{1}{nd} \left\| \sum_{j=0}^{k} w_{k,j} \boldsymbol{\Delta}_j \right\|_F^2 , \quad \widehat{S}_k = \frac{1}{nd} \sum_{j=0}^{k} w_{k,j} \|\boldsymbol{\Delta}_j - \mathbf{M}_k\|_F^2 .$$

Thus, $\widehat{P}_k$ measures persistent motion in the learned updates, whereas $\widehat{S}_k$ represents their temporal fluctuation. TAPS then combines the two energies by

$$\widehat{B}_k := \frac{\widehat{P}_k - \gamma \widehat{S}_k}{\widehat{P}_k + \gamma \widehat{S}_k + \epsilon_{\text{num}}} \in [-1, 1], \tag{2}$$

where $\gamma > 0$ controls the relative weight of fluctuation and $\epsilon_{\text{num}} > 0$ prevents numerical instability when both energies are small.

Let $\rho > 0$ control the adaptation magnitude and let $K_{\text{warm}} \ge 0$ denote the number of warmup loops. During warmup, we set $\eta_k = 1$ while continuing to update the temporal statistics. Set

$$\eta_k = \text{clip}\Big(1 + \rho \widehat{B}_k, \eta_{\min}, \eta_{\max}\Big), \quad k \ge K_{\text{warm}}, \tag{3}$$

where $\text{clip}(x, l, u) := \min\{u, \max\{l, x\}\}$. Therefore, the persistent-dominated motion produces over-relaxation, while fluctuation-dominated motion produces damping.

Remark 2. We term the above instantiation as **TAPS (Adam)** since it adapts the first- and second-moment tracking used in Adam (Kingma & Ba, 2014) to quantify persistence and fluctuation across recurrent updates. Building on the persistence–fluctuation idea, we also provide other instantiations of TAPS such as **TAPS (GD)** and **TAPS (BB)** by varying how P_k and S_k are estimated and how their balance is mapped to the relaxation factor η_k; see Appendix D for details.

4.2 THEORETICAL PROPERTY

The proposed η_k relies on statistics $(\widehat{P}_k, \widehat{S}_k)$, whereas the task-improving direction is determined by the unobservable oracle A_k. We now study when $(\widehat{P}_k, \widehat{S}_k)$ provides a reliable proxy for this direction. Let π denote the proposed causal controller and write $\eta_k := \eta_k^{\pi}(u)$. For a fixed terminal horizon K, loop $k < K$, and $t \in [\eta_{\min}, \eta_{\max}]$, define $\boldsymbol{\eta}^{(k,t)} := (\eta_0, \ldots, \eta_{k-1}, t, 1, \ldots, 1)$. For each decision at loop k, all oracle quantities in the temporal window are evaluated on the same reference trajectory $\boldsymbol{\eta}^{(k,1)}$. In particular, for $j < k$, A_j denotes the coordinate sensitivity evaluated on this reference trajectory. Throughout this subsection, we set the EMA weights $w_{k,j} = (1-\beta)\beta^{k-j}/(1-\beta^{k+1})$.

We first assume that the observable energies are informative about their oracle contributions. Specifically, for each $k < K$, there exist constants $0 < a_- \le a_+$ and $0 \le b_- \le b_+$ with $b_+ > 0$, such that almost surely

$$a_-\widehat{P}_k \le \mathbb{E}[P_k^w \mid \widehat{P}_k, \widehat{S}_k] \le a_+\widehat{P}_k, \quad b_-\widehat{S}_k \le \mathbb{E}[S_k^w \mid \widehat{P}_k, \widehat{S}_k] \le b_+\widehat{S}_k. \tag{4}$$

These bounds only require conditional comparability between the observable energies and the oracle contributions, rather than point-wise agreement. Together with Proposition 2, they connect $(\widehat{P}_k, \widehat{S}_k)$ to the windowed oracle advantage $\bar{A}_k^w$. To relate $\bar{A}_k^w$ to A_k, we further assume that

$$\left|\mathbb{E}[A_k - \bar{A}_k^w \mid \widehat{P}_k, \widehat{S}_k]\right| \le \delta_k^{\text{drift}} \tag{5}$$

for some deterministic $\delta_k^{\text{drift}} \ge 0$. Further discussions on technical assumptions (4)–(5) are deferred to Appendix B.3.

Theorem 3 (Relaxation-direction). *Under equations (4) and (5), we have*

$$a_-\widehat{P}_k - b_+\widehat{S}_k - \delta_k^{\text{drift}} \le \mathbb{E}[A_k \mid \widehat{P}_k, \widehat{S}_k] \le a_+\widehat{P}_k - b_-\widehat{S}_k + \delta_k^{\text{drift}}. \tag{6}$$

If $b_-/a_+ \le \gamma \le b_+/a_-$, then

$$a_-\widehat{P}_k - b_+\widehat{S}_k > \delta_k^{\text{drift}} \implies \widehat{B}_k > 0, \quad \mathbb{E}[A_k \mid \widehat{P}_k, \widehat{S}_k] > 0,$$
$$b_-\widehat{S}_k - a_+\widehat{P}_k > \delta_k^{\text{drift}} \implies \widehat{B}_k < 0, \quad \mathbb{E}[A_k \mid \widehat{P}_k, \widehat{S}_k] < 0.$$

Thus, whenever $a_-\widehat{P}_k - b_+\widehat{S}_k > \delta_k^{\text{drift}}$ or $b_-\widehat{S}_k - a_+\widehat{P}_k > \delta_k^{\text{drift}}$, the sign of $\widehat{B}_k$ correctly indicates whether η_k should increase or decrease compared with $\eta_k = 1$.

We then examine whether the selected value of η_k also decreases the conditional expected terminal loss. For $k \ge K_{\text{warm}}$, define the terminal loss of $\boldsymbol{\eta}^{(k,t)}$ by $\phi_k(t) := L(\mathbf{X}_K^{\boldsymbol{\eta}^{(k,t)}}; y)$. The following theorem gives a finite one-factor gain bound for the selected relaxation factor.

Theorem 4 (One-factor gain bound). *Fix $k \ge K_{warm}$. Suppose that, for some finite constant $H_k \ge 0$, ϕ_k' is H_k-Lipschitz on the interval between 1 and η_k almost surely. Let*

$$g_k^{\text{lb}} := [\eta_k - 1]_+ \left(a_-\widehat{P}_k - b_+\widehat{S}_k - \delta_k^{\text{drift}}\right) + [1-\eta_k]_+ \left(b_-\widehat{S}_k - a_+\widehat{P}_k - \delta_k^{\text{drift}}\right) - \frac{H_k}{2}(\eta_k - 1)^2. \tag{7}$$

Under equations (4) and (5), we have $\mathbb{E}[\phi_k(\eta_k) - \phi_k(1) \mid \widehat{P}_k, \widehat{S}_k] \le -g_k^{\text{lb}}$ almost surely. Consequently, if $\eta_k > 1$ and $a_-\widehat{P}_k - b_+\widehat{S}_k - \delta_k^{\text{drift}} > H_k(\eta_k - 1)/2$, then $\mathbb{E}[\phi_k(\eta_k) - \phi_k(1) \mid \widehat{P}_k, \widehat{S}_k] < 0$. The same conclusion holds if $\eta_k < 1$ and $b_-\widehat{S}_k - a_+\widehat{P}_k - \delta_k^{\text{drift}} > H_k(1 - \eta_k)/2$.

We now study when the proposed schedule reaches a prescribed task quality with fewer loops. For $r = 0, \ldots, K$, let $\pi^{[r]}$ denote the hybrid policy that follows π given by (3) for the first r loops and uses unit relaxation thereafter. For $K > K_{\text{warm}}$, define the cumulative gain lower bound by $G_K^{\text{lb}} := \sum_{k=K_{\text{warm}}}^{K-1} \mathbb{E}[g_k^{\text{lb}}]$.

Theorem 5 (Cumulative gain and loop speedup). *Suppose the conditions of Theorem 4 hold for every $k = K_{\text{warm}}, \ldots, K-1$, each g_k^{lb} is integrable, and $L(\mathbf{X}_K^{\pi^{[r]}}; y)$ is integrable for $r = 0, \ldots, K$. Then*

$$J_K(\pi) \le J_K(\pi_0) - G_K^{\text{lb}}.$$

Consequently, for any $N > K$ with $J_N(\pi_0) < \infty$, if $G_K^{\text{lb}} \ge J_K(\pi_0) - J_N(\pi_0)$, then we have $J_K(\pi) \le J_N(\pi_0)$. For any $\epsilon > 0$, if $G_K^{\text{lb}} \ge J_K(\pi_0) - \varepsilon$, then $K_\varepsilon(\pi) \le K$. If, in addition, $K < K_\varepsilon(\pi_0)$, then $K_\varepsilon(\pi) < K_\varepsilon(\pi_0)$.

Therefore, whenever $G_K^{\text{lb}} \ge J_K(\pi_0) - \varepsilon$ for some $K < K_\varepsilon(\pi_0)$, the proposed policy reaches the target tolerance by loop K, while the unit-relaxation policy has not, yielding $K_\varepsilon(\pi) < K_\varepsilon(\pi_0)$.

Table 1: Accuracy (%) and inference speed of fixed and adaptive step-size schedules on Sudoku ($H = 32$) and Maze ($H = 16$), under inference-only control and training–inference co-design. N/R indicates that the unit-step target is not reached within the loop budget.

Inference method	**Sudoku** ($H = 32$)				**Maze** ($H = 16$)			
	Inference-only		*Co-designed*		*Inference-only*		*Co-designed*	
	Accuracy	*Speedup*	*Accuracy*	*Speedup*	*Accuracy*	*Speedup*	*Accuracy*	*Speedup*
Fixed schedules								
Standard ($\eta = 1.0$)	89.67	1.000×	90.60	1.194×	78.80	1.000×	78.80	1.249×
Constant ($\eta = 0.8$)	89.75	0.990×	91.25	1.275×	78.40	N/R	79.40	1.226×
Constant ($\eta = 1.2$)	85.43	N/R	85.28	N/R	77.00	N/R	77.40	N/R
Adaptive controllers (ours)								
TAPS (GD)	90.07	**1.042**×	91.38	**1.359**×	79.00	1.132×	79.80	1.508×
TAPS (P/S-Sign)	89.98	1.007×	91.23	1.302×	**79.10**	1.171×	**79.90**	**1.561**×
TAPS (Momentum)	90.10	1.040×	**91.39**	1.300×	78.90	**1.509**×	79.80	1.508×
TAPS (RMSProp)	90.10	1.037×	91.38	1.303×	79.00	1.131×	79.70	1.507×
TAPS (Adam)	90.09	1.026×	91.30	1.290×	**79.10**	1.117×	79.80	1.488×
TAPS (BB)	**90.19**	1.039×	91.27	1.263×	78.90	1.465×	79.60	1.465×

5 Experiments

5.1 Experiment Setting

Models. We evaluate our method across three forms of recurrent inference with different recurrence structures. **(a) Latent-state recurrence.** TRM (Jolicoeur-Martineau, 2025) performs iterative refinement directly in latent space. **(b) Language-model recurrence.** Ouro and Huginn-0125 (Zhu et al., 2025; Geiping et al., 2025) provide recurrent inference within language models. **(c) Intermediate-layer recurrence.** We induce recurrence in Qwen3-4B-Instruct by repeatedly applying selected intermediate layers (Chen et al., 2026a; Yang et al., 2025).

Inference Strategies. We evaluate step-size control across four ways of executing recurrent inference. **(a) Unit-step inference.** The standard baseline uses a fixed recurrent horizon with $\eta = 1$. **(b) Adaptive-exit inference.** Adaptive exit dynamically determines the recurrent horizon during inference (Movahedi et al., 2026). **(c) Fixed-point inference.** Fixed-point reasoning instead iterates toward an equilibrium state (Movahedi et al., 2026). **(d) Parallel recurrent inference.** PTRM performs recurrent inference over parallel paths (Sghaier et al., 2026).

Evaluation. We evaluate recurrent inference along two dimensions: Effectiveness and Efficiency.

- **Effectiveness**: Performance under full inference budget, measured by each benchmark's metric.
- **Efficiency**: Wall-clock speedup to reach the pretrained unit-step baseline's terminal performance.

Full benchmark descriptions and evaluation protocols are provided in Appendix C.1.

5.2 Step-Size Schedule at Inference and Training

Inference Only. We first ask whether TAPS can improve recurrent inference without retraining. We apply the inference-only controllers described in Appendix D to pretrained models, adapting recurrent step sizes online. Table 1 shows that all six variants improve terminal accuracy and reach the unit-step baseline's terminal performance in less wall-clock time on both tasks. Different variants lead on different metrics, but these gains hold across all six estimators. Fixed scales, by contrast, fail to improve accuracy and speed consistently across tasks.

Training-Inference Co-Design. We further shape the recurrent dynamics during training to better support adaptive inference. We penalize fluctuation only when it exceeds persistent progress:

$$\mathcal{L}_{\text{co}} = \mathcal{L}_{\text{task}} + \mathcal{L}_{\text{ACT}} + \mu \sum_{k} \left[\widehat{S}_k - \kappa \widehat{P}_k \right]_+ . \tag{8}$$

This preserves useful progress while suppressing fluctuation, yielding recurrent dynamics that are more compatible with adaptive step-size control. As shown in Table 1, co-design further improves

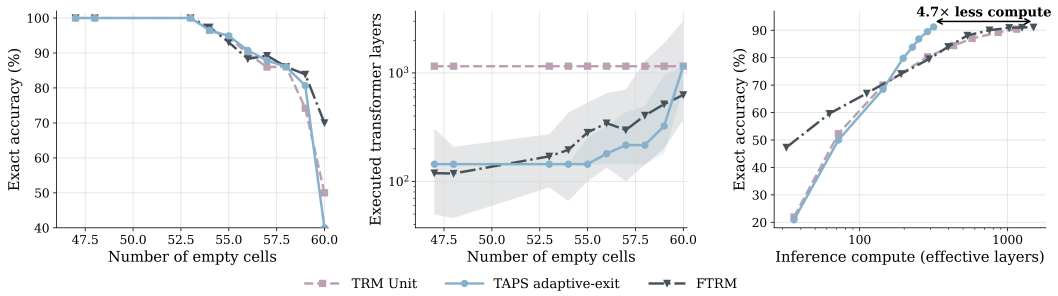


Figure 2: **Difficulty-adaptive inference on Sudoku-Extreme.** *Left:* Exact accuracy across puzzle difficulty. *Middle:* executed transformer layers, with median and 25th–75th percentiles. *Right:* test-time scaling curves showing Exact accuracy as the effective-layer budget increases.

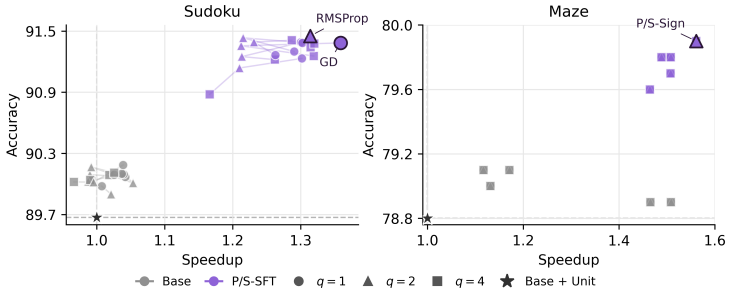


Figure 3: **Controller update frequency.** Accuracy–speedup trade-offs at $q \in \{1, 2, 4\}$ for inference-only (gray) and co-designed (purple) TAPS. Co-design improves the trade-off across refresh intervals.

terminal accuracy and inference efficiency by favoring persistent progress over fluctuation, enabling more aggressive step-size control: the best controller reaches 91.39% accuracy at $1.300\times$ speedup on Sudoku and 79.90% accuracy at $1.561\times$ speedup on Maze, compared with 89.67% and 78.80% under unit-step inference. We further analyze the resulting dynamics in Appendix C.2.

5.3 Comparison with Different Inference Strategies

Adaptive Exit. Different inputs can require different amounts of recurrent computation, making a fixed loop budget inefficient. FPRM (Movahedi et al., 2026) uses fixed-point convergence to stop each sample. We introduce a TAPS-based adaptive-exit rule that stops inference once recent applied updates become small and the prediction stabilizes; see Appendix C.3. We evaluate it on Sudoku-Extreme, using empty-cell count as an established difficulty proxy (Prates & Lamb, 2018). Figure 2 shows that TAPS allocates more compute to harder instances, with gains emerging mainly at high accuracy after sufficient trajectory history enables reliable exit. It reaches 91.2% Exact with 316.5 effective updates, using $4.7\times$ less compute than FPRM at comparable accuracy (91.1%).

Hierarchical Recurrent Inference. TRM and HRM organize recurrence into nested loops operating at different timescales (Jolicoeur-Martineau, 2025; Wang et al., 2025). This creates two choices for step-size control: where to rescale updates and how often to refresh the scale. TAPS operates at either level without altering the nested recurrence. Table 2 shows gains at both levels, with finer inner-loop control yielding higher terminal accuracy ($\Delta_{\mathrm{I-O}} > 0$ for all adaptive controllers, with a larger gap after co-design); the same holds on Maze (Appendix C.4). Figure 3 reveals a non-monotonic frequency trade-off: increasing q reduces controller evaluations but coarsens trajectory tracking, which can require more recurrent updates to reach the same accuracy.

Parallel recurrent inference. PTRM extends recurrent test-time computation from depth scaling to width scaling (Sghaier et al., 2026). It runs multiple latent rollouts in parallel and uses Gaussian noise to diversify them. We extend PTRM by applying TAPS independently within each rollout, without modifying its parallel sampling procedure. Table 10 shows that combining the two consis-

Table 2: Accuracy (%) and inference speed of inner- and outer-loop step-size scheduling on Sudoku ($H = 32$), before and after P/S co-design SFT. $\Delta_{\mathrm{I-O}}$ is inner minus outer terminal accuracy, in percentage points; N/R indicates that the unit-step target is not reached within 32 loops.

Inference method	Inference-only					P/S co-design SFT				
	Inner		*Outer*		$\Delta_{\mathrm{I-O}}$	*Inner*		*Outer*		$\Delta_{\mathrm{I-O}}$
	Accuracy	*Speedup*	*Accuracy*	*Speedup*		*Accuracy*	*Speedup*	*Accuracy*	*Speedup*	
Fixed schedules										
Standard ($\eta = 1.0$)	89.67	1.000×	89.67	1.000×	0.00	90.60	1.194×	90.60	1.194×	0.00
Constant ($\eta = 0.8$)	89.75	0.990×	89.57	N/R	+0.17	91.25	1.275×	90.89	**1.234**×	+0.36
Constant ($\eta = 1.2$)	85.43	N/R	89.49	N/R	−4.06	85.28	N/R	90.38	1.145×	−5.09
Adaptive controllers (ours)										
TAPS (GD)	90.07	**1.042**×	89.68	0.997×	+0.39	91.38	**1.359**×	**90.75**	**1.223**×	+0.64
TAPS (P/S-Sign)	89.98	1.007×	**89.78**	**0.997**×	+0.20	91.23	1.302×	90.63	1.177×	+0.60
TAPS (Momentum)	90.10	1.040×	89.78	0.996×	+0.32	**91.39**	1.300×	90.74	1.220×	+0.65
TAPS (RMSProp)	90.10	1.037×	89.74	0.991×	+0.36	91.38	1.303×	90.71	1.177×	**+0.67**
TAPS (Adam)	90.09	1.026×	89.68	0.993×	+0.41	91.30	1.290×	90.70	1.223×	+0.60
TAPS (BB)	**90.19**	1.039×	89.69	0.987×	**+0.49**	91.27	1.263×	90.68	1.173×	+0.58

Table 3: Accuracy (%) of fixed and adaptive step-size schedules under language-model recurrence, on Ouro-1.4B and Huginn-0125. *Avg.* is the mean over the four benchmarks.

Inference method	Ouro-1.4B					Huginn-0125				
	MMLU	*ARC-C*	*HellaSwag*	*GSM8K*	*Avg.*	*MMLU*	*ARC-C*	*HellaSwag*	*GSM8K*	*Avg.*
Fixed schedules										
Standard ($\eta = 1.0$)	68.23	52.73	71.65	51.78	61.10	31.45	37.80	**66.82**	19.41	38.87
Constant ($\eta = 0.8$)	59.75	50.17	68.02	33.89	52.96	30.97	37.54	66.54	19.79	38.71
Constant ($\eta = 1.2$)	33.67	40.36	46.82	10.54	32.85	27.05	35.15	60.76	8.34	32.83
Adaptive controllers (ours)										
TAPS (GD)	68.21	**52.99**	71.73	53.22	61.54	31.43	**37.97**	66.66	**20.32**	39.10
TAPS (P/S-Sign)	68.22	52.90	71.88	52.99	61.50	31.31	**37.97**	66.70	19.94	38.98
TAPS (Momentum)	68.27	52.56	71.85	56.03	62.18	31.50	**37.97**	**66.73**	19.86	39.02
TAPS (RMSProp)	**68.38**	52.82	71.69	53.98	61.72	31.34	37.80	66.66	**20.32**	39.03
TAPS (Adam)	68.12	52.90	**71.92**	55.57	62.13	31.41	37.71	66.62	20.24	39.00
TAPS (BB)	67.95	52.90	71.61	**58.98**	**62.86**	**31.70**	**37.97**	66.71	20.24	**39.15**

tently improves performance across parallel-sampling budgets. We further ablate the noise scale in Appendix C.5, showing that the gain remains robust across different levels of stochasticity.

Fixed-point inference. Fixed-point reasoning uses convergence of the latent dynamics as an adaptive halting criterion, stopping when $\|\mathbf{z}_{i+1} - \mathbf{z}_i\| \leq \epsilon$ (Movahedi et al., 2026). This criterion enables iterative refinement with additional computation until the latent dynamics stabilize. In contrast, while fixed-point inference favors diminishing state changes, our controller allows the update scale to increase or decrease according to local progress and fluctuation. Table 9 shows that TAPS reaches the higher-accuracy fixed-point targets with substantially fewer recurrent updates; the accuracy that fixed-point inference attains after 1,000 updates is matched within roughly 260.

5.4 Extensions to Other Model Architectures

Language-model recurrence. Recurrent language models improve parameter efficiency through repeated computation and have attracted increasing attention for latent reasoning (Geiping et al., 2025; Zhu et al., 2025). We test whether adaptive update-scale control transfers to this setting by applying TAPS to Ouro-1.4B and Huginn-0125 across four language benchmarks. As shown in Table 3, all six TAPS variants improve average accuracy over unit-step inference on both models, although the gains vary across tasks. In contrast, fixed rescaling is sensitive to the chosen scale and can substantially degrade performance, highlighting the value of adapting the scale along the recurrent trajectory. The same trend extends to the larger Ouro-2.6B model (Table 12).

Intermediate-layer recurrence. Training-free looped transformers provide a representative setting for intermediate-layer recurrence by repeatedly executing a block of hidden layers in an off-the-shelf LLM (Chen et al., 2026a). We apply TAPS to Qwen3-4B-Instruct (Yang et al., 2025) with all model

Table 4: Accuracy (%) and mean step-size multiplier $\bar{\eta}$ of intermediate-layer recurrence in Qwen3-4B-Instruct ($K = 3$). $\bar{\eta}$ is the relaxation factor averaged over recurrent forward passes; *Avg.* is the mean over the three benchmarks.

Inference method	CommonsenseQA		GPQA-Main		MMLU-Pro		Avg.	
	Accuracy	$\bar{\eta}$	*Accuracy*	$\bar{\eta}$	*Accuracy*	$\bar{\eta}$	*Accuracy*	$\bar{\eta}$
Baselines								
Qwen3-4B-Instruct (Base)	78.87	–	34.15	–	57.30	–	56.77	–
Loop ($\eta = 1.0$)	80.10	1.000	36.16	1.000	61.40	1.000	59.22	1.000
Adaptive controllers (ours)								
TAPS (GD)	80.26	1.112	**36.61**	1.125	61.00	1.133	59.29	1.123
TAPS (P/S-Sign)	**80.34**	1.099	36.38	1.120	60.10	1.133	58.94	1.117
TAPS (Momentum)	80.10	1.032	36.38	1.039	60.80	1.042	59.09	1.038
TAPS (RMSProp)	**80.34**	1.099	36.38	1.120	60.10	1.133	58.94	1.117
TAPS (Adam)	80.26	1.082	36.16	1.118	59.70	1.131	58.71	1.110
TAPS (BB)	**80.34**	1.052	36.16	1.087	**61.60**	1.057	**59.37**	1.065

weights frozen. As shown in Table 4, adaptive scaling can further improve the unit-step Loop, with TAPS (BB) achieving the best average accuracy. All controllers select $\bar{\eta} > 1$, indicating that these intermediate-layer loops generally favor mild over-relaxation.

6 Related Work

Designing Recurrent Inference. The recurrence $\mathbf{X}_{k+1} = \mathbf{X}_k + \eta_k \boldsymbol{\Delta}_\theta(\mathbf{X}_k; u)$ exposes three distinct design axes: what update to apply ($\boldsymbol{\Delta}_\theta$), how long to iterate ($K$), and how far to move at each step (η_k). Prior work has extensively explored the first two. Architecture-focused methods design the learned update $\boldsymbol{\Delta}_\theta$ (Liao & Poggio, 2016; Dehghani et al., 2018; Bai et al., 2019; Shomali et al., 2026; Koishekenov et al., 2025), while adaptive-depth methods vary K across inputs or tokens to allocate computation where needed (Graves, 2016; Banino et al., 2021; Bae et al., 2025; Chen et al., 2025). Convergence-based approaches also determine K online, using the evolving state to decide when to halt (Movahedi et al., 2026). Here, we turn to the third axis: how the update scale η_k should evolve along the recurrent trajectory.

Adaptive Scaling of Iterative Updates. The appropriate scale of a recurrent update can vary along the inference trajectory. Prior work adjusts this scale primarily to improve stability or convergence. Stability-oriented approaches use smaller fixed steps, either at inference time for frozen models or during training as a function of model depth and recurrent horizon (Chen et al., 2026a; Wang et al., 2026). Fixed-point methods use residual information to damp updates or construct accelerated directions, aiming to recover or accelerate convergence (Movahedi et al., 2026; Anderson, 1965; Walker & Ni, 2011). TAPS targets a different objective: task performance under a finite inference budget. It adapts η_k online according to the progress–stability tradeoff while preserving the direction of the learned update.

7 Conclusion

In this paper, we identify update scale as a new axis in recurrent inference, and we introduce TAPS, a trajectory-aware scheduler that adapts recurrent step sizes using local dynamics. Our theory characterizes how the balance between progress and instability guides beneficial step-size adjustments. Experiments demonstrate improved and generalizable recurrent inference across architectures and inference strategies without modifying model weights. Several directions remain open for future study. (i) Our theory relies on conditions linking trajectory statistics to task contributions, and extending these guarantees to broader recurrent dynamics remains an interesting direction. (ii) TAPS currently relies on local trajectory statistics as observable proxies, and incorporating richer signals may provide more effective recurrent computation control. (iii) Our training-inference co-design provides an initial step toward shaping recurrent dynamics for step-size scheduling, leaving deeper integration into training and broader architectures for future work.

AI USE STATEMENT

We used generative AI tools to aid and polish writing, to draft portions of the manuscript, and to assist with mathematical derivations and proofs. All AI-assisted material was reviewed by the authors. Mathematical arguments produced with AI assistance were independently verified for correctness, and AI-generated text was revised before inclusion in the manuscript. We take responsibility for the final content of this work, including all text, claims, proofs, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies adaptive recurrent update scaling to improve the effectiveness and efficiency of recurrent reasoning models. All experiments are conducted on publicly available, standard academic benchmarks and open-weight or publicly released models, and do not involve human subjects, personally identifiable information, or sensitive applications.

REPRODUCIBILITY STATEMENT

We provide detailed theoretical and experimental information to support reproducibility. Appendices A and B contain complete proofs of the main theoretical results and discuss the assumptions underlying our analysis. Appendix C documents the datasets, evaluation protocols, recurrent configurations, controller hyperparameters, random seeds, training recipe, and implementation details. Appendix D provides pseudocode for the unified inference procedure and all TAPS variants evaluated in the paper.

REFERENCES

Donald G Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12(4):547–560, 1965.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *Advances in Neural Information Processing Systems*, 38:96572–96617, 2025.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32, 2019.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021.

Jonathan Barzilai and Jonathan M Borwein. Two-point step size gradient methods. *IMA journal of numerical analysis*, 8(1):141–148, 1988.

Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, et al. Lessons from the trenches on reproducible evaluation of language models. *arXiv preprint arXiv:2405.14782*, 2024.

Lizhang Chen, Jonathan Li, Chen Liang, Ni Lao, and Qiang Liu. Training-free looped transformers. *arXiv preprint arXiv:2605.23872*, 2026a.

Wenkai Chen, Tianshu Li, Wenyong Huang, Yichun Yin, Lifeng Shang, and Chengwei Qin. Loopmoe: Unifying iterative computation with mixture-of-experts for language modeling. *arXiv preprint arXiv:2606.04438*, 2026b.

Yilong Chen, Junyuan Shang, Zhenyu Zhang, Yanxi Xie, Jiawei Sheng, Tingwen Liu, Shuohuan Wang, Yu Sun, Hua Wu, and Haifeng Wang. Inner thinking transformer: Leveraging dynamic depth scaling to foster adaptive internal thinking. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28241–28259, 2025.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Zitian Gao, Yilong Chen, Yihao Xiao, Xinyu Yang, Ran Tao, Joey Zhou, and Bryan Dai. Loop the loopies! *arXiv preprint arXiv:2607.16051*, 2026.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Advances in Neural Information Processing Systems*, 38: 41340–41391, 2025.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Ryan Lee, Jacob Biloki, Edward J Hu, and Jonathan May. Sparse layers are critical to scaling looped language models. *arXiv preprint arXiv:2605.09165*, 2026.

Qianli Liao and Tomaso Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv preprint arXiv:1604.03640*, 2016.

Yura Malitsky and Konstantin Mishchenko. Adaptive gradient descent without descent. *arXiv preprint arXiv:1910.09529*, 2019.

Sajad Movahedi, Vera Milovanović, Shlomo Libo Feigin, Alexander Theus, Thomas Hofmann, Valentina Boeva, T Konstantin Rusch, and Antonio Orvieto. Fixed-point reasoners: Stable and adaptive deep looped transformers. *arXiv preprint arXiv:2606.18206*, 2026.

Anthony Nguyen and Wenjun Lin. Intra-layer recurrence in transformers for language modeling. In *Canadian AI*, 2025.

Marcelo Prates and Luis Lamb. Problem solving at the edge of chaos: Entropy, puzzles and the sudoku freezing transition. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 686–693. IEEE, 2018.

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. *arXiv preprint arXiv:2311.12022*, 2023.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, volume 2025, pp. 14855–14881, 2025.

Amin Sghaier, Ali Parviz, and Alexia Jolicoeur-Martineau. Probabilistic tiny recursive model. *arXiv preprint arXiv:2605.19943*, 2026.

Behzad Shomali, Markus Frey, David Berghaus, Joachim Koehler, and Mehdi Ali. Loopmtp: A looped transformer guided by latent multi-token prediction. *arXiv preprint arXiv:2608.03624*, 2026.

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, Volume 1 (Long and Short Papers)*, pp. 4149–4158, 2019.

Maria-Luiza Vladarean, Yura Malitsky, and Volkan Cevher. A first-order primal-dual method with adaptivity to local smoothness. *Advances in Neural Information Processing Systems*, 34:6171–6182, 2021.

Homer F Walker and Peng Ni. Anderson acceleration for fixed-point iterations. *SIAM Journal on Numerical Analysis*, 49(4):1715–1735, 2011.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model, 2025. URL https://arxiv.org/abs/2506.21734.

Shaowen Wang, Bingrui Li, Ge Zhang, Wenhao Huang, Shen Yan, and Jian Li. On the residual scaling of looped transformers: Stability and transferability. *arXiv preprint arXiv:2606.18524*, 2026.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37: 95266–95290, 2024.

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

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International Conference on Learning Representations*, volume 2024, pp. 42195–42214, 2024.

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

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A TECHNICAL PROOFS

A.1 PROOF OF PROPOSITION 1

Consider any fixed $k < K$. Since $\mathbf{X}_k$ depends only on the preceding factor $(\eta_0, \ldots, \eta_{k-1})$, we have

$$\frac{\partial \mathbf{X}_{k+1}}{\partial \eta_k} = \boldsymbol{\Delta}_\theta(\mathbf{X}_k; u) = \boldsymbol{\Delta}_k. \tag{9}$$

For every $j = k+1, \ldots, K-1$, by the chain rule and equation (1), we have

$$\frac{\partial \mathbf{X}_{j+1}}{\partial \eta_k} = D_j \frac{\partial \mathbf{X}_j}{\partial \eta_k}.$$

Iterating from equation (9) yields

$$\frac{\partial \mathbf{X}_K}{\partial \eta_k} = D_{K-1} D_{K-2} \cdots D_{k+1} \boldsymbol{\Delta}_k,$$

where the product is the identity when $k = K-1$. Therefore, we have

$$\begin{aligned}
\frac{\partial L(\mathbf{X}_K; y)}{\partial \eta_k} &= \langle \nabla_{\mathbf{X}} L(\mathbf{X}_K; y), D_{K-1} \cdots D_{k+1} \boldsymbol{\Delta}_k \rangle_F \\
&= \left\langle D_{k+1}^* \cdots D_{K-1}^* \nabla_{\mathbf{X}} L(\mathbf{X}_K; y), \boldsymbol{\Delta}_k \right\rangle_F \\
&= \langle \mathbf{G}_{k+1}, \boldsymbol{\Delta}_k \rangle_F ,
\end{aligned}$$

where the final equality follows by repeatedly applying the adjoint recursion that $G_j = D_j^* G_{j+1}$.

A.2 PROOF OF PROPOSITION 2

By the definitions of A_j and $\bar{A}_k^w$, we have

$$\bar{A}_k^w = -\sum_{j=0}^{k} w_{k,j} \langle \mathbf{G}_{j+1}, \boldsymbol{\Delta}_j \rangle_F .$$

Expanding the centered cross term gives

$$\begin{aligned}
S_k^w &= \sum_{j=0}^{k} w_{k,j} \left\langle \mathbf{G}_{j+1} - \bar{\mathbf{G}}_k^w, \boldsymbol{\Delta}_j - \bar{\boldsymbol{\Delta}}_k^w \right\rangle_F \\
&= \sum_{j=0}^{k} w_{k,j} \langle \mathbf{G}_{j+1}, \boldsymbol{\Delta}_j \rangle_F - \left\langle \sum_{j=0}^{k} w_{k,j} \mathbf{G}_{j+1}, \bar{\boldsymbol{\Delta}}_k^w \right\rangle_F - \left\langle \bar{\mathbf{G}}_k^w, \sum_{j=0}^{k} w_{k,j} \boldsymbol{\Delta}_j \right\rangle_F \\
&\quad + \sum_{j=0}^{k} w_{k,j} \left\langle \bar{\mathbf{G}}_k^w, \bar{\boldsymbol{\Delta}}_k^w \right\rangle_F .
\end{aligned}$$

Since

$$\sum_{j=0}^{k} w_{k,j} = 1, \qquad \sum_{j=0}^{k} w_{k,j} \mathbf{G}_{j+1} = \bar{\mathbf{G}}_k^w, \qquad \sum_{j=0}^{k} w_{k,j} \boldsymbol{\Delta}_j = \bar{\boldsymbol{\Delta}}_k^w,$$

we obtain

$$S_k^w = \sum_{j=0}^{k} w_{k,j} \langle \mathbf{G}_{j+1}, \boldsymbol{\Delta}_j \rangle_F - \left\langle \bar{\mathbf{G}}_k^w, \bar{\boldsymbol{\Delta}}_k^w \right\rangle_F .$$

Therefore,

$$\bar{A}_k^w = -\left\langle \bar{\mathbf{G}}_k^w, \bar{\boldsymbol{\Delta}}_k^w \right\rangle_F - S_k^w = P_k^w - S_k^w,$$

which completes the proof.

A.3 Proof of Theorem 3

Proof. By Proposition 2, we have $\bar{A}_k^w = P_k^w - S_k^w$. Taking conditional expectations with respect to $(\widehat{P}_k, \widehat{S}_k)$ gives

$$\mathbb{E}[\bar{A}_k^w \mid \widehat{P}_k, \widehat{S}_k] = \mathbb{E}[P_k^w \mid \widehat{P}_k, \widehat{S}_k] - \mathbb{E}[S_k^w \mid \widehat{P}_k, \widehat{S}_k].$$

The lower bound in equation (4) for P_k^w and the upper bound for S_k^w therefore imply

$$\mathbb{E}[\bar{A}_k^w \mid \widehat{P}_k, \widehat{S}_k] \geq a_- \widehat{P}_k - b_+ \widehat{S}_k.$$

Similarly, using the upper bound for P_k^w and the lower bound for S_k^w gives

$$\mathbb{E}[\bar{A}_k^w \mid \widehat{P}_k, \widehat{S}_k] \leq a_+ \widehat{P}_k - b_- \widehat{S}_k.$$

Hence

$$a_- \widehat{P}_k - b_+ \widehat{S}_k \leq \mathbb{E}[\bar{A}_k^w \mid \widehat{P}_k, \widehat{S}_k] \leq a_+ \widehat{P}_k - b_- \widehat{S}_k.$$

Note that

$$\mathbb{E}[A_k \mid \widehat{P}_k, \widehat{S}_k] = \mathbb{E}[\bar{A}_k^w \mid \widehat{P}_k, \widehat{S}_k] + \mathbb{E}[A_k - \bar{A}_k^w \mid \widehat{P}_k, \widehat{S}_k].$$

By equation (5), we have

$$a_- \widehat{P}_k - b_+ \widehat{S}_k - \delta_k^{\text{drift}} \leq \mathbb{E}[A_k \mid \widehat{P}_k, \widehat{S}_k] \leq a_+ \widehat{P}_k - b_- \widehat{S}_k + \delta_k^{\text{drift}},$$

which proves equation (6). It remains to establish the two directional claims. By Lemma 6, $\widehat{P}_k, \widehat{S}_k \geq 0$. Since $\epsilon_{\text{num}} > 0$, the denominator of equation (2) is strictly positive. Thus the sign of $\widehat{B}_k$ is determined by $\widehat{P}_k - \gamma \widehat{S}_k$. Suppose $a_- \widehat{P}_k - b_+ \widehat{S}_k > \delta_k^{\text{drift}}$. Since $\delta_k^{\text{drift}} \geq 0$, we have $a_- \widehat{P}_k > b_+ \widehat{S}_k$, and hence $\widehat{P}_k > b_+ \widehat{S}_k / a_-$. The assumed upper bound $\gamma \leq b_+ / a_-$ then gives $\widehat{P}_k > \gamma \widehat{S}_k$, so the numerator of equation (2) is positive and therefore $\widehat{B}_k > 0$. Moreover, the lower bound in equation (6) gives

$$\mathbb{E}[A_k \mid \widehat{P}_k, \widehat{S}_k] \geq a_- \widehat{P}_k - b_+ \widehat{S}_k - \delta_k^{\text{drift}} > 0.$$

Now suppose $b_- \widehat{S}_k - a_+ \widehat{P}_k > \delta_k^{\text{drift}}$. By the same argument, we have $\widehat{P}_k < b_- \widehat{S}_k / a_+$. Since $\gamma \geq b_- / a_+$, it holds that $\widehat{P}_k < \gamma \widehat{S}_k$. Thus the numerator of equation (2) is negative and $\widehat{B}_k < 0$. Finally, the upper bound in equation (6) gives

$$\mathbb{E}[A_k \mid \widehat{P}_k, \widehat{S}_k] \leq a_+ \widehat{P}_k - b_- \widehat{S}_k + \delta_k^{\text{drift}} < 0.$$

This proves both directional claims. □

A.4 Proof of Theorem 4

Proof. At $t = 1$, ϕ_k and A_k are evaluated on the same reference schedule $\boldsymbol{\eta}^{(k,1)}$. Proposition 1 therefore gives $\phi_k'(1) = -A_k$. Since η_k is determined by $(\widehat{P}_k, \widehat{S}_k)$ through the controller, it can be taken outside conditional expectations.

By the first-order Taylor bound implied by the H_k Lipschitz continuity of ϕ_k', we have almost surely

$$\phi_k(\eta_k) - \phi_k(1) \leq (\eta_k - 1)\phi_k'(1) + \frac{H_k}{2}(\eta_k - 1)^2.$$

Taking conditional expectations gives

$$\mathbb{E}[\phi_k(\eta_k) - \phi_k(1) \mid \widehat{P}_k, \widehat{S}_k] \leq -(\eta_k - 1)\mathbb{E}[A_k \mid \widehat{P}_k, \widehat{S}_k] + \frac{H_k}{2}(\eta_k - 1)^2.$$

Using $\eta_k - 1 = [\eta_k - 1]_+ - [1 - \eta_k]_+$ and the two bounds in equation (6), we have

$$(\eta_k - 1)\mathbb{E}[A_k \mid \widehat{P}_k, \widehat{S}_k] \geq [\eta_k - 1]_+ \left(a_- \widehat{P}_k - b_+ \widehat{S}_k - \delta_k^{\text{drift}}\right) - [1 - \eta_k]_+ \left(a_+ \widehat{P}_k - b_- \widehat{S}_k + \delta_k^{\text{drift}}\right).$$

Substitution yields

$$\begin{aligned}
&\mathbb{E}[\phi_k(\eta_k) - \phi_k(1) \mid \widehat{P}_k, \widehat{S}_k] \\
&\leq -[\eta_k - 1]_+ \left(a_- \widehat{P}_k - b_+ \widehat{S}_k - \delta_k^{\text{drift}}\right) - [1 - \eta_k]_+ \left(b_- \widehat{S}_k - a_+ \widehat{P}_k - \delta_k^{\text{drift}}\right) + \frac{H_k}{2}(\eta_k - 1)^2 \\
&= -g_k^{\text{lb}},
\end{aligned}$$

which proves the claimed gain bound. If $\eta_k > 1$, the first margin condition in the theorem implies $g_k^{\mathrm{lb}} > 0$; if $\eta_k < 1$, the second margin condition implies $g_k^{\mathrm{lb}} > 0$. In either case, we have

$$\mathbb{E}[\phi_k(\eta_k) - \phi_k(1) \mid \widehat{P}_k, \widehat{S}_k] < 0,$$

which proves the two strict-improvement statements. □

A.5 Proof of Theorem 5

Proof. For $r = 0, \ldots, K$, define $Y_r := L(\mathbf{X}_K^{\pi^{[r]}}; y)$. By the definition of the hybrid policies, we have

$$Y_0 = L(\mathbf{X}_K^{\pi_0}; y), \quad Y_K = L(\mathbf{X}_K^{\pi}; y).$$

For each $k < K$, the policies $\pi^{[k]}$ and $\pi^{[k+1]}$ follow π for all loops before k. Since π is causal, these rollouts have the same state and EMA statistics up to loop k, and hence the same $\mathbf{\Delta}_k$, $(\widehat{P}_k, \widehat{S}_k)$, and controller factor η_k. Moreover, both policies use unit relaxation after loop k. Therefore, by the definition of ϕ_k, we have

$$\phi_k(1) = Y_k, \quad \phi_k(\eta_k) = Y_{k+1}.$$

It follows that

$$L(\mathbf{X}_K^{\pi}; y) - L(\mathbf{X}_K^{\pi_0}; y) = Y_K - Y_0 = \sum_{k=0}^{K-1} (Y_{k+1} - Y_k).$$

For $k < K_{\mathrm{warm}}$, the proposed policy uses $\eta_k = 1$, so $Y_{k+1} - Y_k = 0$. For every $k = K_{\mathrm{warm}}, \ldots, K-1$, Theorem 4 gives

$$\mathbb{E}[Y_{k+1} - Y_k \mid \widehat{P}_k, \widehat{S}_k] \le -g_k^{\mathrm{lb}}.$$

Taking expectations and applying the tower property yield

$$\mathbb{E}[Y_{k+1} - Y_k] \le -\mathbb{E}[g_k^{\mathrm{lb}}].$$

Hence,

$$J_K(\pi) - J_K(\pi_0) \le -\sum_{k=K_{\mathrm{warm}}}^{K-1} \mathbb{E}[g_k^{\mathrm{lb}}] = -G_K^{\mathrm{lb}},$$

which completes the proof of $J_K(\pi) \le J_K(\pi_0) - G_K^{\mathrm{lb}}$.

If $N > K$ and $G_K^{\mathrm{lb}} \ge J_K(\pi_0) - J_N(\pi_0)$, then

$$J_K(\pi) \le J_K(\pi_0) - G_K^{\mathrm{lb}} \le J_N(\pi_0),$$

which proves the matched-quality claim. Finally, if $G_K^{\mathrm{lb}} \ge J_K(\pi_0) - \varepsilon$, then

$$J_K(\pi) \le J_K(\pi_0) - G_K^{\mathrm{lb}} \le \varepsilon.$$

By the definition of $K_\varepsilon(\pi)$, $K_\varepsilon(\pi) \le K$. If, in addition, $K < K_\varepsilon(\pi_0)$, then

$$K_\varepsilon(\pi) \le K < K_\varepsilon(\pi_0),$$

and therefore $K_\varepsilon(\pi) < K_\varepsilon(\pi_0)$. □

B Additional Theoretical Results and Discussions

B.1 Exact EMA Mean–Variance Representation

Lemma 6 (Exact EMA Mean–Variance Representation)**.** *For the normalized weights* $w_{k,j} := (1-\beta)\beta^{k-j}/(1-\beta^{k+1})$ *defined in Section 4.1, we have*

$$\mathbf{M}_k = \sum_{j=0}^{k} w_{k,j} \mathbf{\Delta}_j, \quad v_k = \frac{1}{nd} \sum_{j=0}^{k} w_{k,j} \|\mathbf{\Delta}_j\|_F^2 .$$

Consequently, it holds that

$$\widehat{P}_k = \frac{1}{nd} \left\| \sum_{j=0}^{k} w_{k,j} \mathbf{\Delta}_j \right\|_F^2, \quad \widehat{S}_k = \frac{1}{nd} \sum_{j=0}^{k} w_{k,j} \|\mathbf{\Delta}_j - \mathbf{M}_k\|_F^2 .$$

Proof. Unrolling the recursion for $\mathbf{M}_k$ gives

$$\mathbf{M}_k = \sum_{j=0}^{k} \frac{(1-\beta)\beta^{k-j}}{1-\beta^{k+1}} \mathbf{\Delta}_j = \sum_{j=0}^{k} w_{k,j} \mathbf{\Delta}_j.$$

The same calculation for v_k yields

$$v_k = \frac{1}{nd} \sum_{j=0}^{k} w_{k,j} \left\| \mathbf{\Delta}_j \right\|_F^2.$$

The expression for $\widehat{P}_k$ follows directly from its definition. Finally, using $\sum_{j=0}^{k} w_{k,j} = 1$ and $\sum_{j=0}^{k} w_{k,j} \mathbf{\Delta}_j = \mathbf{M}_k$, we have

$$\frac{1}{nd} \sum_{j=0}^{k} w_{k,j} \left\| \mathbf{\Delta}_j - \mathbf{M}_k \right\|_F^2 = \frac{1}{nd} \sum_{j=0}^{k} w_{k,j} \left\| \mathbf{\Delta}_j \right\|_F^2 - \frac{1}{nd} \left\| \mathbf{M}_k \right\|_F^2 = v_k - \widehat{P}_k = \widehat{S}_k,$$

which completes the proof. □

B.2 Limits of constant relaxation: a toy example

We consider a simplified model to provide intuition that a nonconstant schedule may be more effective than any constant schedule. Let the task error at loop k be measured by $Q(\mathbf{e}_k) := \sum_{i=1}^{r} q_i e_{i,k}^2/2$, where $\mathbf{e}_k = (e_{1,k}, \ldots, e_{r,k})^\top$ represents $r \geq 2$ error components and $q_i > 0$ specifies the ith component contribution. Suppose each component evolves as $e_{i,k+1} = (1 - \eta_k \lambda_i) e_{i,k}$, where $\lambda_i > 0$ characterizes its response to the relaxation factor. After K loops, we have $e_{i,K} = e_{i,0} \prod_{k=0}^{K-1} (1 - \eta_k \lambda_i)$. Thus, choosing $\eta_k = 1/\lambda_i$ eliminates the ith error component. When the λ_i differ, different components favor different relaxation factors, suggesting an advantage for nonconstant schedules. The following proposition formalizes this separation.

Proposition 7. *Under the model above, suppose $q_i > 0$ and $e_{i,0} \neq 0$ for all i, the λ_i are pairwise distinct, $K \geq r$, and $\lambda_i^{-1} \in [\eta_{\min}, \eta_{\max}]$ for all i. Then $\min_{\boldsymbol{\eta} \in [\eta_{\min}, \eta_{\max}]^K} Q(\mathbf{e}_K^{\boldsymbol{\eta}}) = 0$. Moreover, for $i, j \in \{1, \ldots, r\}$ with $\lambda_i < \lambda_j$, every constant $\eta \in [\eta_{\min}, \eta_{\max}]$ satisfies*

$$Q(\mathbf{e}_K^{(\eta,\ldots,\eta)}) \geq \frac{\min\{q_i e_{i,0}^2, q_j e_{j,0}^2\}}{2} \left(\frac{\lambda_j - \lambda_i}{\lambda_j + \lambda_i} \right)^{2K} > 0.$$

Proof. By iterating the component dynamics, we have

$$e_{i,K} = e_{i,0} \prod_{k=0}^{K-1} (1 - \lambda_i \eta_k), \quad i = 1, \ldots, r.$$

Hence,

$$Q(\mathbf{e}_K^{\boldsymbol{\eta}}) = \frac{1}{2} \sum_{i=1}^{r} q_i e_{i,0}^2 \prod_{k=0}^{K-1} (1 - \lambda_i \eta_k)^2.$$

Choose $\eta_{i-1} = \lambda_i^{-1}$, $i = 1, \ldots, r$ and choose arbitrary admissible factors for the remaining $K - r$ loops. For each i, the corresponding product contains $1 - \lambda_i \eta_{i-1} = 0$, and hence $e_{i,K} = 0$. Therefore $Q(\mathbf{e}_K^{\boldsymbol{\eta}}) = 0$. The proof of the first claim is completed by the fact that $Q \geq 0$.

Now restrict to a constant factor η and fix any $i, j \in \{1, \ldots, r\}$ with $\lambda_i < \lambda_j$. Retaining only these two components gives

$$\begin{aligned} Q(\mathbf{e}_K^{(\eta,\ldots,\eta)}) &\geq \frac{1}{2} \left(q_i e_{i,0}^2 |1 - \lambda_i \eta|^{2K} + q_j e_{j,0}^2 |1 - \lambda_j \eta|^{2K} \right) \\ &\geq \frac{1}{2} \min\{q_i e_{i,0}^2, q_j e_{j,0}^2\} \left(|1 - \lambda_i \eta|^{2K} + |1 - \lambda_j \eta|^{2K} \right). \end{aligned}$$

By the triangle inequality, we have

$$\begin{aligned}\lambda_j|1-\lambda_i\eta| + \lambda_i|1-\lambda_j\eta| &\geq |\lambda_j(1-\lambda_i\eta) - \lambda_i(1-\lambda_j\eta)| \\ &= \lambda_j - \lambda_i.\end{aligned}$$

Therefore, we have

$$\max\{|1-\lambda_i\eta|,\, |1-\lambda_j\eta|\} \geq \frac{\lambda_j-\lambda_i}{\lambda_j+\lambda_i}.$$

At least one of the two terms in the preceding lower bound is therefore no smaller than $\left(\frac{\lambda_j-\lambda_i}{\lambda_j+\lambda_i}\right)^{2K}$. Consequently, we have

$$Q(\mathbf{e}_K^{(\eta,\ldots,\eta)}) \geq \frac{1}{2}\min\{q_i e_{i,0}^2, q_j e_{j,0}^2\}\left(\frac{\lambda_j-\lambda_i}{\lambda_j+\lambda_i}\right)^{2K},$$

which completes the proof. □

B.3 Discussion of assumptions

Recall that P_k^w and S_k^w depend on the task loss, whereas only information regarding the recurrent updates can be used to construct the surrogate estimates $\hat{P}_k$ and $\hat{S}_k$. We therefore derive properties of TAPS by leveraging two explicit conditions (4)–(5) in Section 4.2. In this section, we provide detailed discussions of these conditions. Section B.3.1 explains the role of each condition. Section B.3.2 further provides a toy but representative model to show that the required relations can hold exactly. All oracle quantities below use the reference schedules of Section 4.2.

B.3.1 How the assumptions connect the proxy to the decision

Let $\mathcal{Z}_k = (\widehat{P}_k, \widehat{S}_k)$. Combining the exact identity $\bar{A}_k^w = P_k^w - S_k^w$ from Proposition 2 with the comparison bounds in (4) gives

$$a_-\widehat{P}_k - b_+\widehat{S}_k \leq \mathbb{E}[\bar{A}_k^w \mid \mathcal{Z}_k] \leq a_+\widehat{P}_k - b_-\widehat{S}_k.$$

Hence a larger $\widehat{P}_k$ raises both bounds on $\mathbb{E}[\bar{A}_k^w \mid \mathcal{Z}_k]$. A larger $\widehat{S}_k$ lowers the lower bound and can only lower, or leave unchanged, the upper bound. The constants $a_\pm, b_\pm$ allow for scale differences caused by task curvature and downstream dynamics; unbiased or pointwise estimation is not required.

These bounds connect the observable energies to the EMA-weighted sensitivity $\bar{A}_k^w$, rather than directly to the current sensitivity $A_k = -\partial L(\mathbf{X}_K; y)/\partial\eta_k$. The temporal-mismatch condition (5) then supplies the second link by requiring $|\mathbb{E}[A_k - \bar{A}_k^w \mid \mathcal{Z}_k]| \leq \delta_k^{\text{drift}}$. It therefore expands the two endpoints above by $\pm\delta_k^{\text{drift}}$, giving the explicit lower and upper bounds on $\mathbb{E}[A_k \mid \mathcal{Z}_k]$ in (6). If the lower bound is positive, then increasing η_k above 1 is the loss-decreasing local direction in conditional expectation. If the upper bound is negative, decreasing η_k below 1 is the corresponding direction. When either strict sign condition holds and $b_-/a_+ \leq \gamma \leq b_+/a_-$, Theorem 3 shows that the sign of $\widehat{B}_k$ agrees with this direction, and the proposed strategy η_k makes the corresponding change. Together, the two conditions separate the approximation into two distinct requirements: proxy alignment with the task-dependent quantities and temporal alignment with the current decision.

B.3.2 A two-mode fixed-point example

We provide a two-mode fixed-point model in which conditions (4)–(5) hold exactly at the first controlled decision. Let $e_j = (x_j, z_j) \in \mathbb{R}^{1\times 2}$ denote the error from the desired fixed point after j updates, with initial error $e_0 = (x_0, z_0)$. This specializes $\mathbf{X}_j$ to $n = 1$ and $d = 2$; fixed input and target arguments are suppressed. For $0 < r < 1$, consider

$$L(x,z) = \frac{1}{2}\big((1-r)x^2 + (1+r)z^2\big), \qquad \Delta(x,z) = \big(-(1-r)x, -(1+r)z\big). \tag{10}$$

Write $H = \mathrm{diag}(1-r, 1+r)$ and $\Delta_j := \Delta(e_j) = -e_jH$. The recurrence $e_{j+1} = e_j(I_2 - \eta_j H)$ with I_2 the 2×2 identity is gradient descent on this quadratic. At unit relaxation, its two factors are r and $-r$: the first mode contracts without changing sign, while the second alternates. Increasing

the scale locally can therefore help one mode and hurt the other. These factors arise naturally from a quadratic with extreme curvatures $0 < \lambda_{\min} < \lambda_{\max}$: the balanced step $\alpha_* := 2/(\lambda_{\min} + \lambda_{\max})$ gives $\alpha_*\lambda_{\min} = 1 - r$ and $\alpha_*\lambda_{\max} = 1 + r$, where $r = (\lambda_{\max} - \lambda_{\min})/(\lambda_{\max} + \lambda_{\min})$. Thus unit relaxation here corresponds to the balanced step after absorbing α_* into the curvature.

Consider the EMA estimator of Section 4.1 with $\beta = r$ and $K_{\text{warm}} = 1$ such that $\eta_0 = 1$ and $e_1 = (rx_0, -rz_0)$. Before selecting η_1, the available updates Δ_0, Δ_1 have weights $w_{1,0} = r/(1+r)$ and $w_{1,1} = 1/(1+r)$, giving

$$M_1 = \frac{r\Delta_0 + \Delta_1}{1+r} = \left(-\frac{2r(1-r)}{1+r}x_0,\ 0\right).$$

Since the alternating factor is $-r = -\beta$, its contribution cancels from M_1 and hence from both $\widehat{P}_1$ and P_1^w. It remains in $\widehat{S}_1$, which also contains a contribution from the monotone mode.

Fix a horizon $K \geq 2$. Following Section 4.2, let $\eta^{(1,t)}$ be the K-step schedule with factor $t \in [\eta_{\min}, \eta_{\max}]$ at $j = 1$ and unit factors otherwise, and set $\phi_1(t) := L(e_K^{\eta^{(1,t)}})$, where e_K^{η} is the terminal error under schedule η. All oracle quantities below use the all-unit reference $\eta^{(1,1)}$, whereas η_1 denotes the controller's selected factor. The loss comparison therefore changes only the decision at $k = 1$, with unit relaxation thereafter.

Proposition 8 (Exact proxy–oracle proportionality and loss decrease)**.** *Under the setup above, every initial state satisfies*

$$P_1^w = a\widehat{P}_1, \quad S_1^w = b\widehat{S}_1, \quad A_1 = \bar{A}_1^w, \tag{11}$$

with $a = r^{2K-3}(1+r^2)$ and $b = 2r^{2K-2}$. For any initial-state distribution with finite second moment, these identities imply (4)–(5) at $k = 1$ with $a_- = a_+ = a$, $b_- = b_+ = b$, and $\delta_1^{\text{drift}} = 0$. Choose

$$\gamma = \frac{b}{a} = \frac{2r}{1+r^2}, \qquad 0 < \rho < \frac{2r}{1+r}, \qquad 0 < \eta_{\min} < 1 < \eta_{\max}. \tag{12}$$

For any $\epsilon_{\text{num}} > 0$, the controller (3) satisfies $\operatorname{sign}(\eta_1 - 1) = \operatorname{sign}(A_1)$ and

$$\phi_1(\eta_1) < \phi_1(1) \quad \textit{whenever} \quad (1-r)^2x_0^2 \neq (1+r)^2z_0^2.$$

Otherwise, $\eta_1 = 1$ and the loss is unchanged. For each fixed initial state, the gain bound g_1^{lb} in Theorem 4 equals the actual loss decrease when H_1 is chosen as the exact curvature of ϕ_1.

Proof. Let $U_0 = (1-r)x_0$ and $V_0 = (1+r)z_0$. The two-observation EMA gives

$$\widehat{P}_1 = \frac{2r^2}{(1+r)^2}U_0^2, \quad \widehat{S}_1 = \frac{r(1-r)^2}{2(1+r)^2}U_0^2 + \frac{r}{2}V_0^2. \tag{13}$$

Along the reference, $e_j = (r^j x_0, (-r)^j z_0)$ and $\Delta_j = (-r^j U_0, -(-r)^j V_0)$. The adjoint recursion gives $G_{j+1} = (r^{2K-j-1}U_0, (-r)^{2K-j-1}V_0)$, hence $A_j = r^{2K-1}(U_0^2 - V_0^2)$ for $j = 0, \ldots, K-1$. Using $w_{1,0}, w_{1,1}$ in the oracle definitions yields

$$P_1^w = \frac{2r^{2K-1}(1+r^2)}{(1+r)^2}U_0^2, \quad S_1^w = r^{2K-1}\left(\frac{(1-r)^2}{(1+r)^2}U_0^2 + V_0^2\right). \tag{14}$$

These expressions and $A_0 = A_1$ prove (11). In particular, $A_1 = a(\widehat{P}_1 - \gamma\widehat{S}_1)$. Under the single-factor schedule $\eta^{(1,t)}$, the terminal loss is

$$\phi_1(t) = \frac{r^{2K-2}}{2}\left((1-r)x_0^2(1-(1-r)t)^2 + (1+r)z_0^2(1-(1+r)t)^2\right).$$

Its curvature $H_1 := \phi_1''(t) = r^{2K-2}\big((1-r)U_0^2 + (1+r)V_0^2\big)$ is constant in t but depends on e_0; finite second moment alone need not give a uniform bound on H_1.

Set $\mathcal{N}_1 := \widehat{P}_1 + \gamma\widehat{S}_1 + \epsilon_{\text{num}} > 0$ and $\widetilde{s} := \rho A_1/(a\mathcal{N}_1)$. The actual change is $s := \operatorname{clip}(1 + \widetilde{s}, \eta_{\min}, \eta_{\max}) - 1 = \eta_1 - 1$. Because $1 \in (\eta_{\min}, \eta_{\max})$, clipping preserves the sign of nonzero $\widetilde{s}$ and can only reduce its magnitude. Moreover, $a\mathcal{N}_1 \geq r^{2K-1}(c_r U_0^2 + V_0^2)$, where $c_r := 1 + 2(1 -$

Table 5: Evaluation protocols for language-model and intermediate-layer recurrence experiments. All evaluations use `lm-evaluation-harness` (Biderman et al., 2024) with bfloat16 weights and greedy decoding for generative tasks. Ouro and Huginn use automatic vLLM batching, whereas intermediate-layer recurrence experiments use batch size 1.

Benchmark	Shots	Prompt format	Scoring
MMLU	0	standard	label log-likelihood
ARC-C / HellaSwag	0	standard	normalized label log-likelihood
GSM8K	0 (CoT)	`Let's think step by step`	extracted exact match
CommonsenseQA	7	standard	label log-likelihood
GPQA-Main	0	standard	label log-likelihood
MMLU-Pro (1k)	5 (CoT)	Qwen chat template	regex-extracted exact match

$r)^2/(1+r)^2 \geq 1$, so $H_1/(a\mathcal{N}_1) \leq (1+r)/r$. For $A_1 \neq 0$, we have $H_1 > 0$, and (12) ensures $0 < |s| \leq |\widetilde{s}| = \rho|A_1|/(a\mathcal{N}_1) < 2|A_1|/H_1$. The exact quadratic expansion gives

$$\phi_1(1) - \phi_1(\eta_1) = |s|\,|A_1| - \frac{H_1}{2}s^2 > 0.$$

For fixed e_0, the exact comparison constants and zero temporal mismatch make this precisely g_1^{lb}. If $A_1 = 0$, then $\widetilde{s} = s = 0$. □

By Proposition 8, the proposed persistence–fluctuation statistics can align exactly with their task-dependent oracle counterparts, with zero temporal mismatch at the controlled decision. We note that this simplified model captures a canonical local behavior of recurrent fixed-point dynamics: some modes contract monotonically, while others alternate in sign. By isolating one mode of each type, it represents the same persistence–fluctuation competition that motivates TAPS.

C Additional Experiments

C.1 Experimental Setup

Latent-state recurrence. We evaluate TRM (Jolicoeur-Martineau, 2025) on Sudoku-Extreme and Maze-Hard (Wang et al., 2025) with $L = 2$ recurrent layers, $H_{\mathrm{cycles}} = 3$, and $L_{\mathrm{cycles}} = 6$, giving a recurrent horizon of $H = 32$ on Sudoku and $H = 16$ on Maze. For Sudoku-Extreme, we evaluate five disjoint subsets of 10,000 test examples each (50,000 total; seeds 0–4), while Maze-Hard uses the full 1,000-example test set. We report exact accuracy: a prediction is correct only if the entire output grid matches the reference solution. All methods use the same evaluation examples within each task. For the difficulty-stratified analysis, we group Sudoku-Extreme puzzles by empty-cell count (Prates & Lamb, 2018).

Language-model recurrence. We evaluate Ouro-1.4B, Ouro-2.6B (Zhu et al., 2025), and Huginn-0125 (Geiping et al., 2025) on MMLU, ARC-Challenge, HellaSwag, and GSM8K (Hendrycks et al., 2020; Clark et al., 2018; Zellers et al., 2019; Cobbe et al., 2021), using the recurrent depth released with each checkpoint. Prompting and decoding settings follow Table 5; GSM8K is scored by exact match on the extracted final answer, and the remaining tasks by likelihood ranking over the candidate labels.

Intermediate-layer recurrence. We follow the target configuration of Chen et al. (2026a) and apply block recurrence to layers 15–18 of a frozen Qwen3-4B-Instruct (Yang et al., 2025), with $K = 3$ damped forward Euler stages applied during both prefill and decoding:

$$\mathbf{h}_{k+1} = \mathbf{h}_k + \frac{\eta_k}{3}\left(F(\mathbf{h}_k) - \mathbf{h}_k\right), \tag{15}$$

where F denotes the selected block. The Loop baseline fixes $\eta_k = 1$, i.e. an Euler step of $1/3$, and Base is the unmodified model. Benchmarks are CommonsenseQA (full validation split, 1,221 questions), GPQA-Main (448 questions), and a 1,000-question subset of MMLU-Pro drawn from the shuffled test split (Talmor et al., 2019; Rein et al., 2023; Wang et al., 2024); MMLU-Pro subject scores are weighted by evaluated sample count.

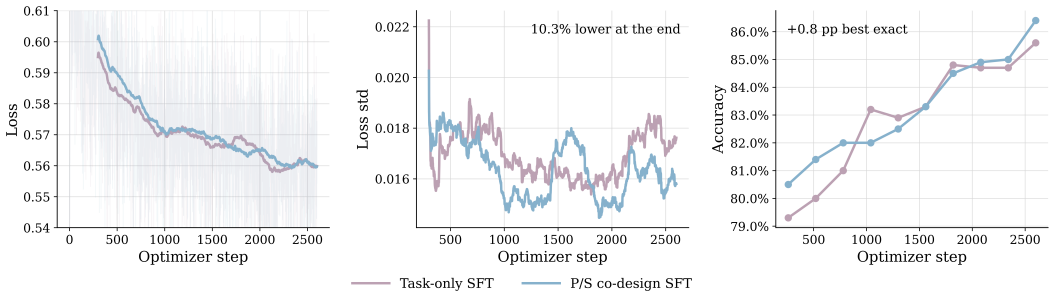


Figure 4: Training–inference co-design improves the stability and generalization of recurrent reasoning. *Left:* Training task loss, shown with raw observations and a 300-step moving average. *Middle:* Local standard deviation of the training loss over the same window; P/S co-design reduces late-stage variability by 10.3%. *Right:* Validation exact accuracy at matched checkpoints, where P/S co-design improves the best accuracy by 0.8 percentage points.

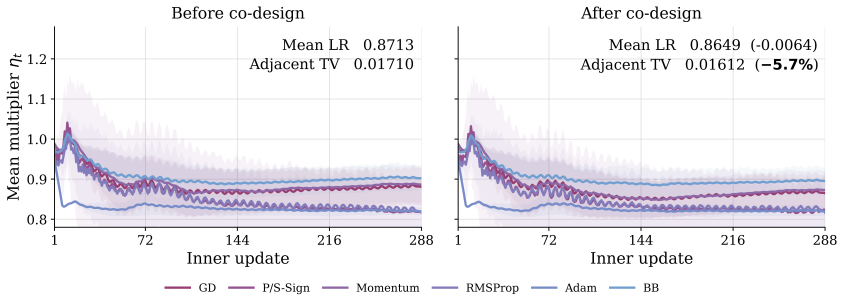


Figure 5: P/S co-design produces smoother inference-time step-size schedules. Each curve shows the mean multiplier of one TAPS controller over 100 Sudoku examples, smoothed with a nine-step centered moving average; shading denotes one standard deviation across examples. After co-design, the average multiplier changes only slightly, while the adjacent total variation decreases by 5.7%, indicating more temporally stable step-size adaptation.

Scheduler configuration. All six controllers share $\gamma = 1.5$, $\rho = 0.275$, $\eta_k \in [0.8, 1.2]$, and $\beta = 0.8$ where a moving average is used. A fresh controller is initialized at every forward call, so no history is carried across generated tokens; during prefill the controller aggregates over prompt positions and hidden dimensions to produce one multiplier per sequence. The first stage is a warmup step with $\eta_1 = 1$ whose statistics seed the two adaptive stages. We report $\bar{\eta} = \frac{1}{3}\sum_{k=1}^{3} \mathbb{E}_{\text{forward}}[\eta_k]$, so that $\bar{\eta} = 1$ under Loop. Hyperparameters were selected on CommonsenseQA and GPQA-Main and applied unchanged to MMLU-Pro.

C.2 Additional Experiments on Training Inference Co-design

Training setting. We conduct a paired SFT experiment initialized from the same pretrained Sudoku TRM checkpoint. Task-only SFT and P/S co-design use identical data order, optimization hyperparameters, architecture, and random seed, as summarized in Table 6. The task-only baseline retains unit-step recurrent inference. For co-design, TAPS is activated throughout training and the objective is augmented with

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{task}} + \mathcal{L}_{\text{ACT}} + \mu \sum_{k} \text{ReLU}(S_k - \kappa P_k)\,,$$

where μ controls the strength of the balance penalty and κ sets the tolerated fluctuation-to-progress ratio. We use $\mu = 10^{-3}$ and $\kappa = 4$ in all co-design experiments. Checkpoints are saved every 100 epochs and evaluated on the same 1,000 held-out examples.

Analysis. Co-design affects the recurrent dynamics in two complementary ways.

Table 6: Training recipe for task-only SFT and P/S co-design SFT on Sudoku-Extreme. The two runs share all listed settings except the inference controller and the balance penalty.

	Task-only SFT	P/S co-design SFT
Training hyperparameters		
Dataset	Sudoku-Extreme	
Initialization	TRM checkpoint at step 65,100	
Training budget	1,000 epochs / 2,600 optimizer steps	
Global batch size	384	
Learning rate	1×10^{-5}	
LR scheduler	100-step warmup + cosine decay	
Minimum LR ratio	0.1	
Weight decay	0.1	
EMA decay	0.999	
Random seed	0	
Recurrent configuration		
Recurrent layers	$L = 2$	
High-level cycles	$H_{\text{cycles}} = 3$	
Low-level cycles	$L_{\text{cycles}} = 6$	
Inference controller		
Training-time adaptation	Disabled	TAPS
Moment decay β	–	0.8
Adaptation radius ρ	–	0.2
Multiplier range	$\eta_k = 1$	$[0.8, 1.2]$
Controller warmup	–	2 updates
Fluctuation weight γ	–	1.0
Training objective		
Task objective	$\mathcal{L}_{\text{task}} + \mathcal{L}_{\text{ACT}}$	$\mathcal{L}_{\text{task}} + \mathcal{L}_{\text{ACT}}$
Balance ratio κ	–	4.0
Balance penalty	–	$10^{-3} \sum_k [S_k - 4P_k]_+$

Training dynamics. Co-design leaves the final training loss largely unchanged, but reduces its late-stage variability by 10.3%. At matched training loss, it lowers held-out terminal loss by 5.9% and improves exact accuracy by 0.8 percentage points. This suggests that the gain does not come from stronger fitting alone; rather, the additional objective favors recurrent trajectories with more stable dynamics and better held-out performance.

Inference controllability. Co-design also makes the resulting trajectories easier for TAPS to control. The mean multiplier changes only slightly, from 0.8713 to 0.8649, while its adjacent total variation decreases by 5.7%. Thus, the gain is not explained by uniformly smaller update scales. Instead, co-design yields smoother scale adaptation and reduces abrupt corrections across successive recurrent updates.

Overall, co-design aligns the learned recurrent dynamics more closely with test-time scale control. Rather than simply changing the average update magnitude, it produces trajectories on which TAPS can preserve productive updates, damp fluctuations when needed, and reach a better accuracy–efficiency trade-off.

C.3 Additional Experiments on Adaptive Exit

We use the magnitude of recent hidden-state updates to determine whether an example has converged. Let $\Delta_k = \Phi_\theta(X_k; u) - X_k$ denote the residual proposed at recurrent step k. We define the step-size-adjusted residual activity as

$$A_k^{\text{exit}} = \frac{\eta_k^2}{2nd} \left(\|\Delta_{k-1}\|_F^2 + \|\Delta_k\|_F^2 \right).$$

Table 7: Accuracy (%) and inference cost of adaptive-exit rules on 1,000 held-out Sudoku-Extreme examples. *Effective Updates* counts recurrent-layer applications per example, and *Compute Saved* is the reduction in mean updates relative to Base, where negative values indicate additional computation. Base and TAPS use a maximum of 32 outer loops; FPRM uses its official stopping rule.

Inference method	**Accuracy**	**Effective Updates**			**Compute Saved**
	Exact	*Mean*	*Median*	*Max*	
Baseline					
Base (unit step)	90.40	1152.0	1152	1152	–
FPRM	91.10	1497.0	284	25838	−29.9%
TAPS adaptive exit (ours)					
TAPS (GD)	90.10	317.6	180	1152	72.4%
TAPS (P/S-Sign)	89.10	319.0	**144**	1152	72.3%
TAPS (Momentum)	90.30	319.0	180	1152	72.3%
TAPS (RMSProp)	90.70	317.6	180	1152	72.4%
TAPS (Adam)	**91.20**	**316.5**	162	1152	**72.5%**
TAPS (BB)	90.50	317.9	**144**	1152	72.4%

Table 8: Accuracy (%) and inference speed of inner- and outer-loop step-size scheduling on Maze ($H = 16$), before and after P/S co-design SFT. $\Delta_{\mathrm{I-O}}$ is inner minus outer terminal accuracy, in percentage points; N/R indicates that the unit-step target is not reached within 16 loops.

Inference method	**Inference-only**					**P/S co-design SFT**				
	Inner		**Outer**		$\boldsymbol{\Delta}_{\mathrm{I-O}}$	**Inner**		**Outer**		$\boldsymbol{\Delta}_{\mathrm{I-O}}$
	Accuracy	*Speedup*	*Accuracy*	*Speedup*		*Accuracy*	*Speedup*	*Accuracy*	*Speedup*	
Fixed schedules										
Standard ($\eta = 1.0$)	78.80	1.000×	78.80	1.000×	0.00	78.80	1.249×	78.80	1.249×	0.00
Constant ($\eta = 0.8$)	78.40	N/R	78.80	1.245×	−0.40	79.40	1.226×	79.50	1.245×	−0.10
Constant ($\eta = 1.2$)	77.00	N/R	78.70	1.245×	−1.70	77.40	N/R	79.50	**1.660**×	−2.10
Adaptive controllers (ours)										
TAPS (GD)	79.00	1.132×	78.80	**0.989**×	+0.20	79.80	1.508×	79.00	1.647×	+0.80
TAPS (P/S-Sign)	**79.10**	1.171×	78.80	0.989×	+0.30	**79.90**	**1.561**×	**79.10**	1.647×	+0.80
TAPS (Momentum)	78.90	**1.509**×	78.70	0.989×	+0.20	79.80	1.508×	79.00	1.647×	+0.80
TAPS (RMSProp)	79.00	1.131×	**78.90**	0.989×	+0.10	79.70	1.507×	79.00	**1.647**×	+0.70
TAPS (Adam)	**79.10**	1.117×	**78.90**	0.988×	+0.20	79.80	1.488×	78.90	1.646×	**+0.90**
TAPS (BB)	78.90	1.465×	78.70	0.986×	+0.20	79.60	1.465×	78.90	1.642×	+0.70

Averaging two consecutive residuals reduces sensitivity to transient fluctuations, while the step-size factor adjusts the statistic to the scale of the recurrent update. We normalize this quantity by

$$R_k = \frac{A_k^{\mathrm{exit}}}{\max_{j \le k} A_j^{\mathrm{exit}} + \epsilon},$$

so that R_k measures the remaining update activity relative to its peak along the trajectory. We exit when the prediction remains unchanged and $R_k \le \tau$ for p consecutive loops after a minimum depth $k\mathrm{min}$. We use $k_{\min} = 4$, $p = 2$, and $\tau = 0.15$.

C.4 Additional Experiments on Update Frequency

The update frequency of TAPS introduces a natural accuracy–efficiency trade-off. Refreshing the controller less frequently reduces the overhead of computing η_k, but also coarsens its tracking of the recurrent trajectory. If this loss of adaptivity slows recurrent progress sufficiently, the saved controller cost can be offset, or even outweighed, by the additional recurrent computation required to reach the same task quality.

We study this trade-off from two complementary views. TRM maintains a high-level latent state z_H across outer cycles and a low-level latent state z_L within each cycle. For inner adaptation, Algorithm 2 is applied to every z_L update, providing fine-grained control of the recurrent trajectory. For outer

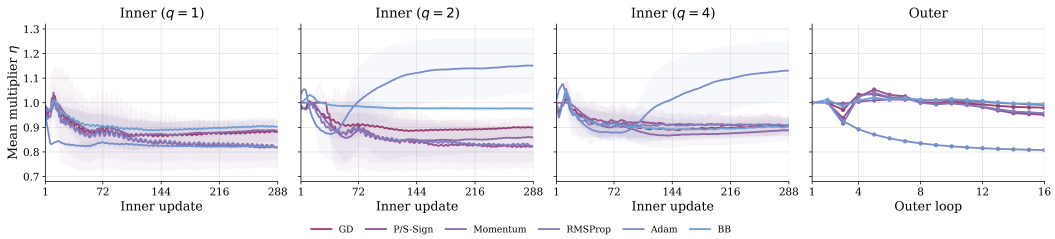


Figure 6: **Step-size trajectories across control granularities on Sudoku.** The first three panels show inner-loop control with $q \in \{1, 2, 4\}$, while the last shows outer-loop control; shading indicates variation across samples. Changing the refresh frequency alters the resulting schedule rather than merely reducing controller evaluations.

Table 9: Fixed-point inference efficiency on Sudoku.

Inference method	Fixed-point updates			
	50	100	500	1000
Fixed-point target				
Accuracy	65.20	71.40	84.20	87.40
Adaptive controllers (ours)				
TAPS (GD)	144.0	162.0	226.4	264.7
TAPS (P/S-Sign)	144.0	**161.4**	223.5	276.0
TAPS (Momentum)	144.0	162.1	225.5	264.3
TAPS (RMSProp)	144.0	162.5	229.7	257.3
TAPS (Adam)	144.0	162.0	233.7	**256.2**
TAPS (BB)	144.0	161.9	**220.5**	264.3

Table 10: Parallel sampling performance on Sudoku.

Inference method	Parallel-sampling budget			
	10	25	50	100
Fixed schedules				
Standard ($\eta = 1.0$)	97.8	98.3	98.9	98.9
Adaptive controllers (ours)				
TAPS (GD)	97.9	98.5	98.6	98.7
TAPS (P/S-Sign)	98.2	98.6	98.6	99.1
TAPS (Momentum)	98.2	**98.7**	98.6	98.9
TAPS (RMSProp)	98.2	98.6	98.9	99.1
TAPS (Adam)	**98.6**	**98.7**	**99.0**	99.0
TAPS (BB)	98.5	98.4	98.5	**99.2**

adaptation, the controller is refreshed only at outer-cycle boundaries: it measures the displacement of z_H and applies the resulting η_k to the carried z_L residual, while the z_H update itself remains a unit step. Outer adaptation can therefore be viewed as a lower-frequency version of the same update-scale control.

We further vary the controller refresh interval $q \in \{1, 2, 4\}$, where smaller q corresponds to more frequent adaptation and larger q reduces controller evaluations at the cost of coarser trajectory tracking. Table 8 compares fine-grained inner-loop control with lower-frequency outer-loop control on Maze, while Figure 3 reports the accuracy–efficiency trade-off across refresh intervals on both Sudoku and Maze.

Overall, reducing the update frequency lowers controller overhead but does not monotonically improve end-to-end efficiency. Coarser control can reduce accuracy and slow recurrent progress enough to offset the saved controller computation. As shown in Table 8, inner-loop scheduling consistently achieves higher Maze accuracy than outer-loop scheduling, with the gap increasing after co-design. Figure 3 further reveals a non-monotonic trade-off across q: less frequent updates save controller cost, but can move the model away from the best accuracy–efficiency operating point. Co-design improves this trade-off across refresh intervals, indicating that TAPS remains effective over a range of update frequencies rather than relying on a particular control granularity.

C.5 Additional Experiments on Parallel Sampling

Parallel-sampling performance. Table 10 shows that adaptive stepping is most useful when the parallel-sampling budget is limited. At $K = 10$, TAPS-Adam improves exact accuracy from 97.8% to 98.6%, while the gain narrows as the baseline approaches saturation at larger budgets. Nevertheless, the best TAPS variant improves over the unit-step baseline at every evaluated budget, reaching 99.2% with TAPS-BB at $K = 100$. These results indicate that update-scale adaptation can improve the quality of individual recurrent rollouts, reducing the amount of parallel sampling needed for high exact accuracy.

Table 11: Accuracy (%) of fixed and adaptive step-size schedules under Gaussian perturbations to the recurrent latent states, on 1,000 held-out Sudoku-Extreme examples ($H = 16$, 25 parallel rollouts). Noise with standard deviation σ is injected into either the low-level state z_L or the high-level state z_H, and each entry reports Best-Q accuracy over the 25 rollouts.

Inference method	Noise on z_L					Noise on z_H				
	$\sigma = 0.2$	$\sigma = 0.4$	$\sigma = 0.6$	$\sigma = 0.8$	$\sigma = 1.0$	$\sigma = 0.2$	$\sigma = 0.4$	$\sigma = 0.6$	$\sigma = 0.8$	$\sigma = 1.0$
Fixed schedule										
Standard ($\eta = 1.0$)	97.33	97.60	97.37	97.60	97.87	97.70	96.20	91.63	**87.27**	82.37
Adaptive controllers (ours)										
GD	97.70	97.50	97.53	98.07	97.70	98.03	96.17	91.17	86.50	81.10
P/S-Sign	97.70	97.70	97.90	97.80	97.73	97.60	96.17	90.63	85.33	80.77
Momentum	97.50	97.50	97.57	97.73	97.70	97.90	96.07	91.53	86.33	82.20
RMSProp	**97.93**	97.73	**98.13**	97.90	97.93	97.93	96.20	91.70	85.07	80.57
Adam	97.87	**98.07**	98.07	**98.20**	**98.07**	**98.27**	**96.33**	**91.77**	85.53	**82.43**
BB	97.83	97.73	97.83	97.83	98.00	97.90	96.30	91.57	85.47	80.93

Table 12: Accuracy (%) of fixed and adaptive step-size schedules under language-model recurrence, on Ouro-2.6B. *Avg.* is the mean over the four benchmarks.

Inference method	Ouro-2.6B				
	MMLU	*ARC-C*	*HellaSwag*	*GSM8K*	*Avg.*
Fixed schedules					
Standard ($\eta = 1.0$)	73.65	58.36	76.45	76.95	71.35
Constant ($\eta = 0.8$)	66.16	52.82	71.85	36.54	56.84
Constant ($\eta = 1.2$)	52.28	51.79	68.40	42.68	53.79
Adaptive controllers (ours)					
TAPS (GD)	73.69	58.36	76.69	77.41	71.54
TAPS (P/S-Sign)	73.66	58.62	**76.74**	**77.79**	**71.70**
TAPS (Momentum)	73.66	**58.87**	76.56	77.41	71.63
TAPS (RMSProp)	**73.71**	58.79	76.52	77.41	71.61
TAPS (Adam)	**73.71**	57.85	76.46	**77.79**	71.45
TAPS (BB)	73.69	58.62	76.63	76.88	71.46

Sensitivity to hidden-state perturbations. As shown in Table 11, our adaptive controllers remain robust when noise is applied to z_L, achieving comparable or improved Best-Q accuracy across all noise levels. In contrast, perturbing z_H causes accuracy to decrease as the noise level increases, with adaptive methods occasionally underperforming the fixed unit-step baseline. This suggests that adaptive step-size control relies on sufficiently reliable high-level hidden-state updates to estimate an appropriate step magnitude and direction. Strong perturbations to z_H corrupt this signal and consequently reduce the benefit of adaptation.

C.6 Additional Experiments on Recurrent Language Models

We further evaluate TAPS on Ouro-2.6B to examine whether the behavior observed on Ouro-1.4B extends to a larger recurrent language model.

As shown in Table 12, all six TAPS variants improve average accuracy over unit-step inference on Ouro-2.6B. The best configuration increases the four-task average from 71.35% to 71.70%, with gains on ARC-C, HellaSwag, and GSM8K while largely preserving MMLU performance. In contrast, fixed rescaling with either $\eta = 0.8$ or $\eta = 1.2$ substantially degrades performance. Together with the Ouro-1.4B results in Table 3, these results suggest that the benefit comes from adapting the update scale along the recurrent trajectory rather than from uniformly changing its magnitude.

Table 13: Latency and speedup of algebraic and operator-level acceleration for TAPS on Sudoku ($H = 16$). Latency is the cumulative CUDA time per example over 16 outer loops. *Alg.* reports the speedup from algebraic simplification relative to the original implementation, *Op.* the additional speedup from operator fusion relative to + *Alg.*, and *Overall* the combined speedup relative to the original implementation.

TAPS controller	CUDA latency (ms/example)			Speedup		
	Original	*+ Alg.*	*+ Alg. + Op.*	*Alg.*	*Op.*	*Overall*
Adaptive controllers (ours)						
TAPS (GD)	11.204	11.195	10.743	1.001×	1.042×	1.043×
TAPS (P/S-Sign)	11.219	11.029	10.768	**1.017**×	1.024×	1.042×
TAPS (Momentum)	11.236	11.207	10.745	1.003×	1.043×	1.046×
TAPS (RMSProp)	11.285	11.281	10.760	1.000×	**1.048**×	**1.049**×
TAPS (Adam)	11.312	11.332	10.872	0.998×	1.042×	1.040×
TAPS (BB)	11.453	11.481	11.091	0.998×	1.035×	1.033×
Mean	11.285	11.254	10.830	1.003×	1.039×	1.042×

C.7 Additional Experiments on Efficient TAPS Implementation

Algorithm-level acceleration. The original implementation evaluates the progress and stability scores separately:

$$\widehat{P}_k = \frac{1}{4nd}\|\Delta_{k-1} + \Delta_k\|_F^2, \qquad \widehat{S}_k = \frac{1}{4nd}\|\Delta_k - \Delta_{k-1}\|_F^2.$$

This requires separately constructing the sum and difference of two consecutive residuals and then reducing both tensors. We accelerate this computation by expanding the two quadratic forms and jointly computing their shared residual energies and inner product. The resulting implementation reuses these statistics to recover both $\widehat{P}_k$ and $\widehat{S}_k$, thereby avoiding redundant tensor materialization and repeated reductions. Because this is an exact algebraic transformation, it preserves both the controller output and the recurrent trajectory. As shown in Table 13, this optimization introduces no systematic slowdown and achieves an average speedup of $1.003\times$, with the largest improvement of $1.017\times$ obtained by TAPS (P/S-Sign).

Operator-level acceleration. After algebraic simplification, TAPS requires little additional arithmetic. A low FLOP count, however, does not guarantee low latency: eager PyTorch decomposes each controller update into several small GPU kernels, repeatedly reading the same residuals and materializing temporary tensors. Because the controller runs throughout recurrent inference, this overhead accumulates over hundreds of hidden-state updates. Figure 7 contrasts this fragmented execution with our fused design.

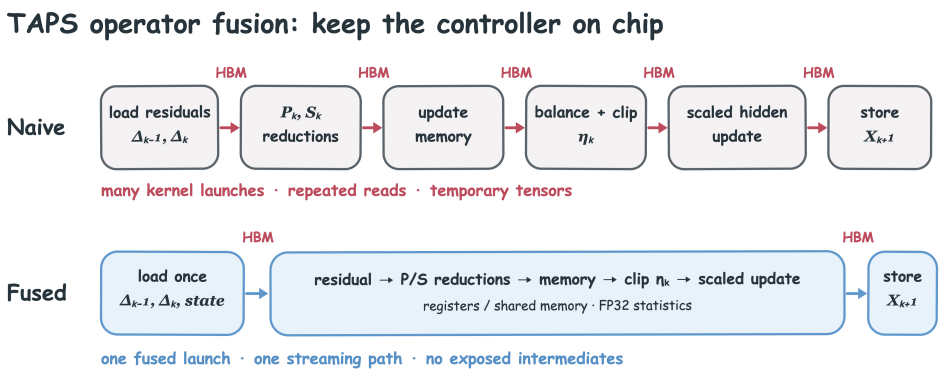


Figure 7: **Fusing the TAPS controller update.** The eager implementation executes each controller stage as a separate GPU kernel, whereas the fused operator keeps intermediate state on chip and completes the update through a single streaming path.

The fused Triton/CUDA operator reads each residual pair once, computes the progress and stability statistics, updates the controller memory, determines and clips η_k, and applies the scaled residual through a single execution path. Reductions use FP32 for numerical stability, while hidden-state updates retain the model's original computation dtype. This design reduces kernel launches, global-memory traffic, and intermediate tensor materialization without changing the controller equations. The ablation in Table 13 shows that fusion accounts for most of the practical runtime gain. Combined with the algebraic simplification, it yields consistent $1.033\times$–$1.049\times$ end-to-end speedups across TAPS variants, averaging $1.042\times$.

D Inference-Time Step-Size Schedulers

This section specifies the inference-time step-size schedulers used in our experiments. All schedulers operate on the same pretrained recurrent model and differ only in how they choose the step size at each recurrent update.

Fixed schedules. We first provide the algorithm of fixed schedules. We note that these baselines use no adaptive memory.

Algorithm 1 Standard and constant relaxation factors

Hyperparameter: η_{const} (Standard uses $\eta_{\text{const}} = 1$)

State: none

Update rule:

$$\eta_k \leftarrow \eta_{\text{const}}$$

return η_k

Unified scheduler interface. We introduce a unified version of TAPS. We represent a scheduler as a stateful rule

$$(\eta_k, q_{k+1}) = \mathcal{A}(\mathbf{\Delta}_k, q_k), \tag{16}$$

where q_k contains any history required by the scheduler. Fixed schedules have no memory, while adaptive schedulers use recent recurrent updates to determine η_k. Algorithm 2 summarizes the common inference procedure.

Algorithm 2 Unified inference-time relaxation scheduling

Input: u, initial state $\mathbf{X}_0$, tied recurrent core Φ_θ, horizon K

Scheduler: rule $\mathcal{A}$ with initial memory q_0 and warmup K_{warm}

for $k = 0, \dots, K-1$ **do**
  $\mathbf{\Delta}_k \leftarrow \mathbf{\Delta}_\theta(\mathbf{X}_k; u)$
  $(\eta_k, q_{k+1}) \leftarrow \mathcal{A}_k(\mathbf{\Delta}_k, q_k)$
  $\mathbf{X}_{k+1} \leftarrow \mathbf{X}_k + \eta_k \mathbf{\Delta}_k$

Output: $\mathbf{X}_K$

The following sections present implementations of TAPS with different rule $\mathcal{A}$.

D.1 TAPS with Adjacent-update Scores.

For $k \geq 1$, GD, P/S-Sign, Momentum, and RMSProp use

$$\widehat{P}_k = \frac{1}{4nd} \|\mathbf{\Delta}_{k-1} + \mathbf{\Delta}_k\|_F^2, \qquad \widehat{S}_k = \frac{1}{4nd} \|\mathbf{\Delta}_k - \mathbf{\Delta}_{k-1}\|_F^2.$$

$\widehat{P}_k$ measures the part that persists across two updates, whereas $\widehat{S}_k$ measures the part that alternates between them. During the two-step warmup, history-dependent scores are not evaluated: we set $\eta_k = 1$ and only store the states required by subsequent scheduler updates.

Algorithm 3 TAPS (GD)

Hyperparameters: $\gamma, \rho, \eta_{\min}, \eta_{\max}, \epsilon_{\text{num}}, K_{\text{warm}}$

State: previous update $\boldsymbol{\Delta}_{k-1}$

Update rule:

$$\widehat{P}_k \leftarrow \frac{1}{4nd} \|\boldsymbol{\Delta}_{k-1} + \boldsymbol{\Delta}_k\|_F^2$$
$$\widehat{S}_k \leftarrow \frac{1}{4nd} \|\boldsymbol{\Delta}_k - \boldsymbol{\Delta}_{k-1}\|_F^2$$
$$\widehat{B}_k \leftarrow \frac{\widehat{P}_k - \gamma \widehat{S}_k}{\widehat{P}_k + \gamma \widehat{S}_k + \epsilon_{\text{num}}}$$
$$\widetilde{\eta}_k \leftarrow \text{clip}(1 + \rho \widehat{B}_k, \eta_{\min}, \eta_{\max})$$
$$\eta_k \leftarrow 1 \text{ if } k < K_{\text{warm}}; \text{ otherwise } \eta_k \leftarrow \widetilde{\eta}_k$$

Store $\boldsymbol{\Delta}_k$ and **return** η_k

Algorithm 4 TAPS (P/S-Sign)

Hyperparameters: $\gamma, \rho, \eta_{\min}, \eta_{\max}, K_{\text{warm}}$

State: previous update $\boldsymbol{\Delta}_{k-1}$

Update rule:

$$\widehat{P}_k \leftarrow \frac{1}{4nd} \|\boldsymbol{\Delta}_{k-1} + \boldsymbol{\Delta}_k\|_F^2$$
$$\widehat{S}_k \leftarrow \frac{1}{4nd} \|\boldsymbol{\Delta}_k - \boldsymbol{\Delta}_{k-1}\|_F^2$$
$$\sigma_k \leftarrow \text{sign}(\widehat{P}_k - \gamma \widehat{S}_k)$$
$$\widetilde{\eta}_k \leftarrow \text{clip}(1 + \rho \sigma_k, \eta_{\min}, \eta_{\max})$$
$$\eta_k \leftarrow 1 \text{ if } k < K_{\text{warm}}; \text{ otherwise } \eta_k \leftarrow \widetilde{\eta}_k$$

Store $\boldsymbol{\Delta}_k$ and **return** η_k

Algorithm 5 TAPS (Momentum)

Hyperparameters: $\beta, \gamma, \rho, \eta_{\min}, \eta_{\max}, \epsilon_{\text{num}}, K_{\text{warm}}$

State: previous update $\boldsymbol{\Delta}_{k-1}$, scalar momentum μ_{k-1} with $\mu_0 = 0$

Update rule:

$$\widehat{P}_k \leftarrow \frac{1}{4nd} \|\boldsymbol{\Delta}_{k-1} + \boldsymbol{\Delta}_k\|_F^2$$
$$\widehat{S}_k \leftarrow \frac{1}{4nd} \|\boldsymbol{\Delta}_k - \boldsymbol{\Delta}_{k-1}\|_F^2$$
$$\widehat{B}_k \leftarrow \frac{\widehat{P}_k - \gamma \widehat{S}_k}{\widehat{P}_k + \gamma \widehat{S}_k + \epsilon_{\text{num}}}$$
$$\mu_k \leftarrow \beta \mu_{k-1} + (1 - \beta) \widehat{B}_k$$
$$\widetilde{\eta}_k \leftarrow \text{clip}(1 + \rho \mu_k, \eta_{\min}, \eta_{\max})$$
$$\eta_k \leftarrow 1 \text{ if } k < K_{\text{warm}}; \text{ otherwise } \eta_k \leftarrow \widetilde{\eta}_k$$

Store $(\boldsymbol{\Delta}_k, \mu_k)$ and **return** η_k

Algorithm 6 TAPS (RMSProp)

Hyperparameters: $\beta, \gamma, \rho, \eta_{\min}, \eta_{\max}, \epsilon_{\text{num}}, K_{\text{warm}}$

State: previous update $\mathbf{\Delta}_{k-1}$, scalar second moment r_{k-1} with $r_0 = 0$

Update rule:

$$\widehat{P}_k \leftarrow \frac{1}{4nd} \|\mathbf{\Delta}_{k-1} + \mathbf{\Delta}_k\|_F^2$$

$$\widehat{S}_k \leftarrow \frac{1}{4nd} \|\mathbf{\Delta}_k - \mathbf{\Delta}_{k-1}\|_F^2$$

$$\widehat{B}_k \leftarrow \frac{\widehat{P}_k - \gamma \widehat{S}_k}{\widehat{P}_k + \gamma \widehat{S}_k + \epsilon_{\text{num}}}$$

$$r_k \leftarrow \beta r_{k-1} + (1-\beta)\widehat{B}_k^2; \quad \widehat{r}_k \leftarrow \frac{r_k}{1-\beta^k}$$

$$\widetilde{\eta}_k \leftarrow \text{clip}\left(1 + \rho \frac{\widehat{B}_k}{\sqrt{\widehat{r}_k} + \epsilon_{\text{num}}}, \eta_{\min}, \eta_{\max}\right)$$

$$\eta_k \leftarrow 1 \text{ if } k < K_{\text{warm}}; \text{ otherwise } \eta_k \leftarrow \widetilde{\eta}_k$$

Store $(\mathbf{\Delta}_k, r_k)$ and **return** η_k

D.2 TAPS WITH MOMENT AND CURVATURE SCORES

The following schedulers define progress and stability from temporal moments or local directional curvature.

Algorithm 7 TAPS (Adam)

Hyperparameters: $\beta, \gamma, \rho, \eta_{\min}, \eta_{\max}, \epsilon_{\text{num}}, K_{\text{warm}}$

State: update mean $\mathbf{M}_{k-1}$, energy mean v_{k-1}, initialized by $\mathbf{M}_{-1} = \mathbf{0}$, $v_{-1} = 0$

Update rule:

$$\mathbf{M}_k \leftarrow \frac{\beta - \beta^{k+1}}{1-\beta^{k+1}} \mathbf{M}_{k-1} + \frac{1-\beta}{1-\beta^{k+1}} \mathbf{\Delta}_k$$

$$v_k \leftarrow \frac{\beta - \beta^{k+1}}{1-\beta^{k+1}} v_{k-1} + \frac{1-\beta}{nd(1-\beta^{k+1})} \|\mathbf{\Delta}_k\|_F^2$$

$$\widehat{P}_k \leftarrow \frac{1}{nd} \|\mathbf{M}_k\|_F^2; \quad \widehat{S}_k \leftarrow v_k - \widehat{P}_k$$

$$\widehat{B}_k \leftarrow \frac{\widehat{P}_k - \gamma \widehat{S}_k}{\widehat{P}_k + \gamma \widehat{S}_k + \epsilon_{\text{num}}}$$

$$\widetilde{\eta}_k \leftarrow \text{clip}(1 + \rho \widehat{B}_k, \eta_{\min}, \eta_{\max})$$

$$\eta_k \leftarrow 1 \text{ if } k < K_{\text{warm}}; \text{ otherwise } \eta_k \leftarrow \widetilde{\eta}_k$$

Store $(\mathbf{M}_k, v_k)$ and **return** η_k

Algorithm 8 TAPS (BB)

Hyperparameters: $\gamma, \rho, \eta_{\min}, \eta_{\max}, \epsilon_{\text{num}}, K_{\text{warm}}$

State: previous update $\mathbf{\Delta}_{k-1}$ and displacement $\mathbf{s}_{k-1} = \mathbf{X}_k - \mathbf{X}_{k-1}$

Update rule:

$$\mathbf{D}_{k-1} \leftarrow \mathbf{\Delta}_k - \mathbf{\Delta}_{k-1}$$

$$\kappa_k \leftarrow \frac{|\langle \mathbf{s}_{k-1}, \mathbf{D}_{k-1} \rangle_F|}{\|\mathbf{s}_{k-1}\|_F^2 + \epsilon_{\text{num}}}$$

$$\widehat{P}_k \leftarrow \frac{1}{nd}\|\mathbf{\Delta}_k\|_F^2; \quad \widehat{S}_k \leftarrow \kappa_k^2 \widehat{P}_k$$

$$\widehat{B}_k \leftarrow \frac{\widehat{P}_k - \gamma \widehat{S}_k}{\widehat{P}_k + \gamma \widehat{S}_k + \epsilon_{\text{num}}}$$

$$\widetilde{\eta}_k \leftarrow \text{clip}(1 + \rho \widehat{B}_k, \eta_{\min}, \eta_{\max})$$

$$\eta_k \leftarrow 1 \text{ if } k < K_{\text{warm}}; \text{ otherwise } \eta_k \leftarrow \widetilde{\eta}_k$$

$$\mathbf{s}_k \leftarrow \eta_k \mathbf{\Delta}_k$$

Store $(\mathbf{\Delta}_k, \mathbf{s}_k)$ and **return** η_k