WHEN DOES PER-TOKEN HALTING HELP? ADAPTIVE TOKEN-WISE DAMPING FOR FIXED-POINT REASONING MODELS IN LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Fixed-Point Reasoning Models (FPRM) use a parameter-free halting rule based on the maximum token residual. A single slow-converging token can therefore keep an entire sequence active. We introduce Adaptive Token-Wise Damping (ATWD), which adapts damping independently at each token and periodically restores every token's initial step-size. This refresh guarantees bounded intervals between full-strength updates while iteration continues; it does not establish dynamical stability or reduce shared-block evaluation cost. Across three tasks, ATWD's benefits depend on the setting. Under causal attention, ATWD improves mean accuracy and observed cross-seed stability on the alternating group A_5, but on the harder symmetric group S_5 it fails to reach the baseline's late generalization transition within the matched training budget. Scalar-shaped replays show rapid residual convergence; separate prediction diagnostics suggest low-diversity collapse. On bidirectional Sudoku, ATWD exceeds a hard-freeze-training ablation on sequence accuracy in all three seeds tested (+25.5% relative in the ratio of means), but trails the scalar baseline in mean accuracy and cross-seed stability. Because this ablation changes both training-time decay and refresh, it does not isolate refresh's causal contribution. The hard-freeze-trained models also produce fewer duplicate-constraint violations among their incorrect grids, revealing a trade-off between exact solution accuracy and this measure of failure severity. These results identify task-dependent benefits and limits of token-wise damping and distinguish numerical convergence from solution correctness.

1 INTRODUCTION

Recurrent, weight-tied ("looped") Transformers (Vaswani et al., 2017) have emerged as a striking alternative to scaling parameter count or generating long chains of thought: iterating a single shared block toward convergence lets models with only a few million parameters match or exceed much larger language models on hard combinatorial reasoning tasks like Sudoku, mazes, and ARC-AGI (Chollet, 2019; Wang et al., 2025; Jolicoeur-Martineau, 2025). A central design question for such models is *how much computation to spend, and when to stop*. A common answer, following Adaptive Computation Time (Graves, 2016) and used by both HRM and TRM, is a learned halting head deciding whether to continue on the current example.

Fixed-Point Reasoning Models (FPRM) (Movahedi et al., 2026) replace this learned gate with a parameter-free alternative: the shared block is iterated as a damped fixed-point map, and the sequence halts once the iterate stops changing. Concretely, writing $\hat{y}_i[t] = f_\theta(y_i)[t]$ for the shared block's raw, undamped proposal at token t and iteration i, the residual is

$$r_i[t] \;=\; \frac{\|\hat{y}_i[t] - y_i[t]\|_\infty}{\|\hat{y}_i[t]\|_\infty + \epsilon}, \tag{1}$$

where $y_i[t] \in \mathbb{R}^d$ is the iterate entering iteration i, and the model halts the *entire* sequence once a single, sequence-level reduction of this quantity,

$$r_i \;=\; \max_{t \in [T]} r_i[t], \tag{2}$$

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

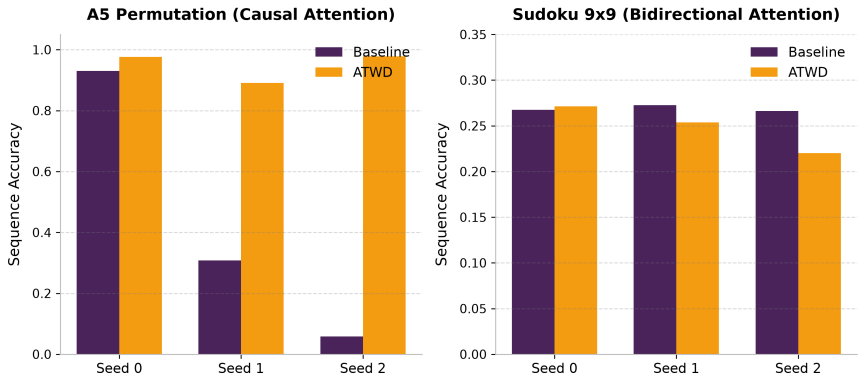


Figure 1: Per-token adaptive halting's cross-seed reliability effect inverts between tasks. **Left (A_5, causal):** ATWD improves accuracy and seed stability over baseline. **Right (Sudoku, bidirectional):** the pattern reverses – ATWD is markedly less stable, despite beating a hard-freeze ablation by a wide margin on every seed (Table 2).

drops below a threshold τ (or a companion damping step-size, applied identically to every token, falls below a floor). This removes the learned halting head, though the accompanying convergence theorem covers only scalar damping (Section 3.3).

The max reduction in Equation (2), however, means the sequence is only as fast as its single slowest token: on a 97-token Sudoku instance or a 128-step state-tracking sequence, one lagging token forces every other, already-converged token to keep iterating at full cost – the gap this paper addresses.

Two adjacent lines of work leave this gap open (Section 2): Jacobian- and spectral-radius regularization of looped models targets the shared block's *dynamics*, not *which tokens receive how much compute*, while token-level adaptive depth in AdaPonderLM (Song et al., 2026) and PonderLM-3 (Li et al., 2026) relies on *learned* per-token routers – the machinery FPRM's convergence-based halting was designed to avoid. Our contribution is a parameter-free, token-wise damping rule within this solver, not a learned exit policy.

We propose **Adaptive Token-Wise Damping (ATWD)**, a minimal extension of FPRM's solver: the per-sequence damping step-size becomes a per-token one, so a token with a persistently high residual receives smaller updates independently of the rest of the sequence, with no additional parameters or learned gate. Because excessive damping under *bidirectional* attention can cause premature commitment – a token whose updates go negligible may fail to respond when neighboring states change – ATWD adds an unconditional periodic refresh every R iterations that resets each token's damping state regardless of convergence history, bounding the interval between full-strength updates without changing how many tokens the shared block evaluates.

Our contributions are:

- **Adaptive Token-Wise Damping**, a parameter-free, per-token extension of FPRM's damped fixed-point solver, requiring a small, targeted change to an existing numerical routine rather than a new architecture.
- **A Bounded Staleness proposition**, a scheduling guarantee that every token receives a full-strength update at least once every R executed iterations – deliberately a scheduling argument, not an extension of FPRM's own dynamical-stability theorem to the per-token case (Section 3.3).
- **A systematic three-task empirical study** spanning causal attention (state tracking on A_5 and S_5, of increasing difficulty) and bidirectional attention (Sudoku, where premature commitment is a genuine risk), each with matched comparisons and diagnostic analyses, reporting results in whichever direction the data supports: improvement on A_5, a diagnostically characterized failure on S_5, and a mixed picture on Sudoku.

2 Related Work

Looped and recurrent reasoning models. Universal Transformers (Dehghani et al., 2019) and subsequent looped, weight-tied architectures established that iterating a single block can substitute for depth, typically with Adaptive Computation Time (ACT) (Graves, 2016) halting via a learned scalar gate. HRM (Wang et al., 2025) and TRM (Jolicoeur-Martineau, 2025) both instantiate this recipe for combinatorial reasoning (Sudoku, mazes, ARC-AGI), each with its own recurrent block and a learned halting head – HRM via a Q-learning-style objective, TRM via binary supervision of solution correctness. FPRM (Movahedi et al., 2026), our base architecture, dispenses with learned gating: the shared block is a damped fixed-point iteration, and halting reads convergence directly off the iterate (Equation (2)). Deep Equilibrium Models (Bai et al., 2019) instead locate a weight-tied transformation's equilibrium via a root-finding solver; FPRM uses damped iteration.

Token-level adaptive computation. Per-token early-exiting and depth-adaptive computation predate looped reasoning models and have been studied extensively for standard, non-recurrent Transformers. Most relevant here are two concurrent works bringing token-level adaptive depth to *recurrent* language-model pretraining: AdaPonderLM (Song et al., 2026) and PonderLM-3 (Li et al., 2026) both extend ACT-style pondering to operate per-token, using a learned MLP-based router to decide each token's exit iteration, with a KV-cache-reuse mechanism so early-exited tokens do not block the rest of the sequence. This targets our same high-level goal – decoupling per-token compute from the slowest token – but the mechanism differs entirely: both reintroduce a learned gate, the exact component FPRM's convergence-based halting eliminates, and both target next-token-prediction pretraining rather than task-specific training of a fixed-point reasoning model. ATWD instead extends FPRM's existing, parameter-free damping directly, adding no new parameters and no router.

Stability regularization for looped models. A separate line of work regularizes the *dynamics* of the shared block, not how compute is allocated across tokens. FPRM's own codebase includes an optional Hutchinson-estimator Jacobian-norm regularizer, left disabled in every experiment here to keep the baseline-vs-ATWD comparison controlled. Concurrently, STARS (Yang et al., 2026) constrains a looped language model's latent states toward stable fixed points via Jacobian spectral-radius regularization, targeting test-time collapse at high recurrence depth; Energy-Entropy Regularization (Lam, 2026) instead treats parameter updates as Hamiltonian flow to escape poor local minima when training looped Transformers from scratch. Neither addresses which tokens should keep receiving updates at inference time – the question ATWD answers.

3 Method

3.1 Background: Damped Fixed-Point Halting in FPRM

FPRM tracks, for each example, a single fixed-point iterate $y_i \in \mathbb{R}^{T\times d}$ across T token positions, updated by damping the shared block's raw proposal toward the current iterate:

$$y_{i+1}[t] \;=\; \eta_i[t]\,\hat{y}_i[t] \;+\; (1-\eta_i[t])\,y_i[t], \tag{3}$$

where $\hat{y}_i[t] = f_\theta(y_i)[t]$ is the shared block's proposal (Equation (1)) and $\eta_i[t] \in [0,1]$ is a damping step-size (zero is reserved for the hard-freeze ablation; we suppress the fixed input to f_θ in the notation). In the original FPRM solver, $\eta_i[t] = \eta_i$ is a single scalar shared by every token: it starts at η_0 and decays by a fixed rate $\delta \in (0,1)$ whenever the *sequence-level* residual (Equation (2)) stops improving for P consecutive iterations above the halting threshold τ – a patience-based rule suppressing oscillation (Movahedi et al., 2026, Theorem 3), not signaling convergence, which is read directly off the residual. Halting occurs once the sequence-level residual drops below τ or the shared step-size falls below 10^{-3}; the released evaluation path further reduces across the batch, so one unconverged example can keep its batch active (the solver's optional variational-dropout and additive-noise terms are disabled throughout; Section A.11).

3.2 Adaptive Token-Wise Damping

ATWD lets $\eta_i[t]$ vary independently per token, replacing the scalar solver state with a per-token one, and evaluating the same decay rule per token rather than on the sequence-level reduction. Concretely,

each token t maintains its own step-size $\eta_i[t]$, best residual since the last reset $b_i[t]$, and patience counter $p_i[t]$. At each iteration, after computing $r_i[t]$ (Equation (1)):

$$b_{i+1}[t] = \begin{cases} r_i[t] & r_i[t] < b_i[t] \\ b_i[t] & \text{otherwise} \end{cases} \qquad p'[t] = \begin{cases} P & r_i[t] < b_i[t] \\ p_i[t] - 1 & \text{otherwise.} \end{cases} \tag{4}$$

and a token *adapts* – its damping decays – only if it has stopped improving for P consecutive iterations *and* has not yet converged:

$$a_i[t] = \mathbb{1}\big[\, p'[t] \le 0 \ \wedge \ r_i[t] \ge \tau \,\big], \tag{5}$$

$$\eta_{i+1}[t] = \begin{cases} \delta \cdot \eta_i[t] & a_i[t] = 1 \\ \eta_i[t] & a_i[t] = 0 \end{cases} \qquad p_{i+1}[t] = \begin{cases} P & a_i[t] = 1 \\ p'[t] & a_i[t] = 0. \end{cases} \tag{6}$$

Equation equation 5 is the mechanism's central property, and one we verified before training (Section 4): a token with $r_i[t] < \tau$ does not adapt on that iteration and retains its *current* step-size, which may already have decayed. Equality at τ can trigger adaptation when patience is exhausted. Damping thus targets stagnation above threshold, not convergence. Setting $\delta = 0$ and disabling refresh gives a **hard-freeze** rule: after the first adaptation, the token receives no further updates until the solver state is reset. Our experimental ablation applies this rule during training; its evaluation decay remains 0.99 (Section 4.3).

3.3 Periodic Refresh and Bounded Staleness

Permanent per-token freezing can prevent a token from responding to subsequent changes in its context, particularly under *bidirectional* attention. ATWD guards against this with an unconditional periodic refresh, governed by a single global counter k_i per example (shared across all T tokens, incremented once per solver iteration): for a positive integer refresh interval R, whenever $k_i \bmod R = 0$, every token's step-size, patience, and best-residual tracker are reset to their initial values, *regardless of any token's individual convergence history*:

$$\begin{gathered} \eta_i[t] \leftarrow \eta_0, \quad p_i[t] \leftarrow P, \quad b_i[t] \leftarrow \infty \\ \forall\, t, \ \text{whenever } k_i \bmod R = 0. \end{gathered} \tag{7}$$

Section A.1 gives the solver pseudocode; the refresh yields a scheduling guarantee, stated and proved below.

Proposition 3.1 (Bounded Staleness under Periodic Refresh)**.** *Fix a positive integer refresh interval R, initialize $k_0 = 0$ and $\eta_0[t] = \eta_0$, and consider an uninterrupted solver run. At the start of every executed iteration i, for every token t, the number of consecutive solver iterations since t's step-size was last reset to its full initial value η_0 is at most $R - 1$.*

Proof. The refresh trigger $k_i \bmod R = 0$ in Equation equation 7 is a deterministic function of the shared, monotonically increasing counter k_i alone; it does not depend on any token's residual, patience, or convergence history. Every R consecutive iterations therefore contain exactly one refresh point, at which *every* token's step-size is unconditionally reset, regardless of what happened to it in between. For any iteration i, let $i_0 = R\lfloor i/R \rfloor \le i$ be the most recent refresh point; by construction $i - i_0 \le R - 1$, and no intervening per-token event can prevent, delay, or override that reset. □

We are deliberately explicit about what Proposition 3.1 does *not* claim. FPRM's own convergence guarantee for the damped map (Equation equation 3; Movahedi et al., 2026, Theorem 3) holds for a *fixed scalar* η: near a fixed point y^* with all Jacobian eigenvalues satisfying $\mathrm{Re}(\lambda) < 1$, the scalar damped map $g_\eta(y) = \eta f_\theta(y) + (1 - \eta)y$ converges locally to y^* for $\eta \in (0, \eta_{\max})$, a bound that need not equal our initial step-size η_0. A per-token damping matrix $D_\eta = \mathrm{diag}(\eta[1], \ldots, \eta[T]) \otimes I_d$ does not preserve this argument: self-attention couples tokens, so the block's Jacobian can have nonzero off-diagonal token blocks, and a diagonal matrix multiplying a non-diagonal Jacobian does not, in general, preserve its eigenvalue locations the way a scalar does. We therefore treat per-token damping as an empirically evaluated design choice, not an extension of Theorem 3 – Proposition 3.1 (a scheduling argument, not a dynamical-stability one) ensures a full-strength update within each R iterations, not a nonzero state change, convergence, correctness, or protection from halting before the next refresh. Sections 5.1 and 5.2 evaluate the resulting behavior empirically.

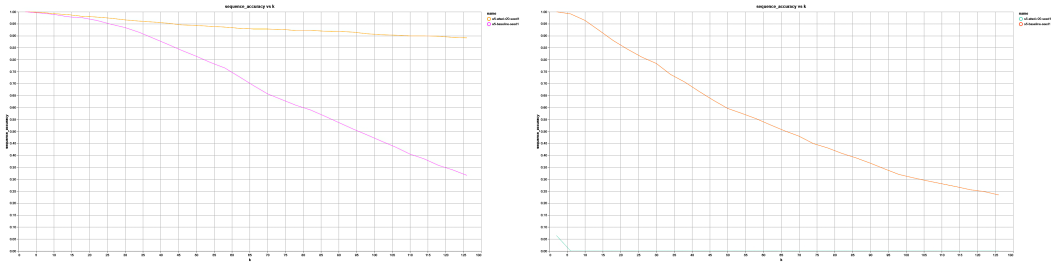


Figure 2: Sequence accuracy vs. evaluation prefix length k (seed 1; aggregate statistics in Table 1). **Left (A_5):** baseline degrades from near-perfect to about 0.32 by k=125, while ATWD stays close to 0.90. **Right (S_5):** baseline degrades more gradually to about 0.24; ATWD is near zero from the smallest k, consistent with its 0.000 ± 0.000 full-sequence accuracy.

4 Experimental Setup

We evaluate ATWD on three tasks: two causal-attention state-tracking tasks of increasing difficulty (A_5, S_5) and one bidirectional-attention constraint-satisfaction task (Sudoku) as a deliberate stress test. Per-run budgets are A_5: 15 epochs at batch 832 (≈11.1h, 3 seeds per arm); S_5: 30 epochs at 832 (≈22.0h over 2 sessions, 2 seeds); Sudoku: 5,500 epochs at 736 (≈10.5h, 3 seeds), all in two-GPU wall-clock time (full hyperparameters in Table 5, Appendix A.5).

4.1 Tasks and Datasets

State tracking (A_5, S_5). Each task is a group word-problem: given a sequence of elements from a finite group (A_5, order 60, or S_5, order 120), the label at position j is the running product of the first $j+1$ inputs, so predicting the sequence requires tracking cumulative group composition. We train on length-32 sequences and evaluate prefixes up to k=128 in a single causal solver run per batch; per-token accuracy averages the first k predictions, sequence accuracy requires all k correct, and no separate per-length run is needed. The 500,000-example training scale is a deliberate, disclosed reduction from an initially planned 2,000,000-example / 50-epoch schedule that our weekly compute budget could not support (generation commands in Section A.11).

Sudoku. Each puzzle is a 9×9 grid (81 tokens) plus 16 puzzle-embedding/workspace positions (97 tokens total), processed with full bidirectional attention; the task is to fill in the blanks subject to Sudoku's row/column/box constraints. We build a fixed set of 1,000 base puzzles from the `sapientinc/sudoku-extreme` benchmark with 1,000 symmetry-preserving augmentations each (digit relabeling, band/stack permutation, transpose) and truncate the evaluation split to a fixed 800 held-out puzzles to bound evaluation time under the solver's iteration budget (sizing rationale in Section A.11).

4.2 Architecture, Optimization, and Infrastructure

All experiments use the released FPRM architecture, with only the damping and refresh changes above, AdamW (Loshchilov & Hutter, 2019), `bfloat16` autocast, and `torch.compile` in PyTorch (Paszke et al., 2019); optimizer settings and checkpointing are in Section A.11. State tracking uses no positional encoding; Sudoku uses rotary embeddings (Su et al., 2021) and an EMA of the weights (decay 0.999, distinct from the iterate averaging of Polyak & Juditsky (1992)); every Sudoku evaluation number and downstream analysis (violation counts, convergence traces) uses these EMA weights, to avoid a train/analysis mismatch. All training ran on Kaggle's free-tier 2×T4, with schedules set from a real throughput benchmark, since GPU-generation differences (local Ampere vs. Kaggle's older Turing, without native `bfloat16` tensor cores) limit extrapolated estimates.

4.3 Ablations, Seeds, and Protocol

For state tracking we compare two arms: **Baseline**, FPRM's unmodified scalar (sequence-level) halting, and **ATWD** (Section 3.2) with refresh interval R=20. This value was fixed from a brief,

Table 1: Length generalization and cross-seed stability on causal permutation state tracking. S_5 used two seeds (Section 4.3). Accuracy aggregates length-128 sequences; variability is sample SD across seeds.

TASK	CONFIGURATION	PER-TOKEN ACC (MEAN ± STD)	SEQ ACC (MEAN ± STD)
A_5	BASELINE	0.784 ± 0.191	0.432 ± 0.449
A_5	ATWD ($R = 20$)	$\mathbf{0.977 \pm 0.019}$	$\mathbf{0.948 \pm 0.050}$
S_5	BASELINE	$\mathbf{0.605 \pm 0.182}$	$\mathbf{0.120 \pm 0.155}$
S_5	ATWD ($R = 20$)	0.020 ± 0.005	0.000 ± 0.000

single-seed calibration sweep over $R \in \{10, 20, 30, 50\}$ on A_5 at a short training budget (Appendix A.3); we did not retune it further, and reused the identical value for Sudoku without a separate sweep. For Sudoku we compare three arms to examine the premature-commitment hypothesis: **Baseline** (unmodified, sequence-level halting), **Hard-freeze** (ATWD's per-token mechanism with $\delta_{\text{train}} = 0$ and R=0, i.e. no periodic refresh; all three arms keep $\delta_{\text{eval}} = 0.99$, so it is a training intervention, not instantaneous evaluation-time freezing), and **ATWD** (R=20, identical to the state-tracking value).

Within every task, every arm shares identical architecture and solver hyperparameters except the ablation flags above; none was retuned separately by arm. We extended both S_5 arms from 15 to 30 epochs after observing limited baseline generalization at 15. We run A_5 and Sudoku over 3 seeds per arm; S_5 over 2, since both produced the same qualitative outcome with similar diagnostic signatures (Section 5.1), and a third seed had low incremental value against a $\sim$22-hour run's cost – an outcome-informed reduction limiting inference about cross-seed variability. Because the R calibration, the S_5 extension, and the omitted third seed all used intermediate results, the protocol was not fully prespecified. All means and SDs in Sections 5.1 and 5.2 are sample statistics ($n-1$ denominator) over seeds actually run, read as descriptive with only two or three per arm, not evidence of statistical significance.

5 RESULTS

5.1 STATE TRACKING: A_5 AND S_5

A_5: a clear, seed-robust win. ATWD improves both metrics at the aggregate level (Table 1, top): mean per-token accuracy rises from 0.784 ± 0.191 to 0.977 ± 0.019, and mean sequence accuracy from 0.432 ± 0.449 to 0.948 ± 0.050. The per-seed results show substantial baseline variability (0.930 / 0.308 / 0.058), while ATWD remains strong across the reported runs (0.976 / 0.891 / 0.978). Figure 2 (left) shows the same effect as a function of evaluation length for a representative seed.

S_5: a genuine, mechanistically-traceable failure to match baseline. S_5 tells a different story (Table 1, bottom). At the same 500,000-sample schedule – extended from 15 to 30 epochs once baseline had not yet reached its transition at the shorter budget (Appendix A.4) – baseline exhibits a seed-dependent, grokking-like transition (Power et al., 2022): sequence accuracy rises to 0.010 and 0.229 on the two seeds (mean 0.120 ± 0.155), each preceded by an extended plateau before a sharp, late jump. S_5's greater difficulty matches prior state-tracking literature: symmetric groups are a documented hard case (Grazzi et al., 2025), and S_5 (order 120) has twice A_5's states (order 60). ATWD does not reach this transition within the same budget: mean sequence accuracy stays at 0.000 ± 0.000 on both seeds, and per-token accuracy (0.020 ± 0.005) sits at essentially baseline's own *pre*-transition level – above chance ($1/120 \approx 0.0083$) but far below trained baseline (Figure 2, right).

Replayed trajectories from the correctness-tested solver (full figures and caveats in Appendix A.4) show why: ATWD's per-example residuals crash below 0.1 almost immediately while step-sizes stay pinned at 1.0 – no damping ever engages – whereas baseline's residuals only cross that threshold much later in the iteration budget. Separate prediction sampling found as few as 5 of 120 classes used per example, one occupying up to 86% of positions. This motivates a low-diversity failure hypothesis, not a fully isolated cause, since damping decisions (though detached from autograd) still shape the forward trajectory and gradients. Why ATWD produces this on S_5 but not A_5 remains open (Section 6).

Table 2: 9×9 Sudoku mean results (3 seeds per arm) on 800 held-out puzzles under the same 5,500-epoch schedule (full per-seed breakdown in Table 6, Appendix A.5). Mean and sample SD use unrounded W&B summaries.

ARM	SEQ. ACCURACY	PER-CELL ACC	LM LOSS
HARD FREEZE	0.198 ± 0.019	0.723 ± 0.009	0.798 ± 0.061
BASELINE	$\mathbf{0.269 \pm 0.003}$	$\mathbf{0.732 \pm 0.001}$	$\mathbf{0.583 \pm 0.005}$
ATWD ($R = 20$)	0.248 ± 0.026	0.725 ± 0.010	0.600 ± 0.021

Table 3: Mean row/column/box duplicate-constraint violations per test puzzle (lower is better; zero means no counted duplicates, not necessarily a valid solution). "Viol. if Failed" averages within each seed's failed subset, then across seeds.

ARCHITECTURE	OVERALL VIOL.	VIOL. IF FAILED	CLEANER?
HARD FREEZE	**24.85**	**30.97**	$\surd$ (+21.9% VS. BASE)
BASELINE	28.98	39.65	×
ATWD ($R = 20$)	29.33	39.04	×

5.2 SUDOKU: CENTRAL HYPOTHESIS — PERIODIC REFRESH VS. HARD-FREEZE

Table 2 shows that ATWD exceeds the hard-freeze-trained arm on sequence accuracy in *all three* seeds, not only in the mean – $+0.0813$ ($+42.8\%$ relative) at seed 0, $+0.0338$ ($+15.3\%$) at seed 1, $+0.0363$ ($+19.7\%$) at seed 2, a mean advantage of 0.0504 ($+25.5\%$ relative, computed as the ratio of the per-seed means before rounding). Pairwise margins use exact solved counts, rounded only for presentation. This three-seed advantage supports the combined intervention, not a separate causal effect of refresh.

Figure 3 characterizes evaluation-budget utilization, not the cause of the accuracy difference. Halting uses the maximum residual or step-size across the evaluation batch plus the iteration cap. Because all arms share $\delta_{\text{eval}} = 0.99$, Section A.9 rules out the step-size floor as an explanation, so the saturation order instead reflects the residual criterion, with later evaluations increasingly reaching the cap. This is compatible with, but not proof of, training-dependent differences in the learned dynamics – confirming a causal link would need matched-decay ablations and direct measurement of halting causes, and reaching the cap alone shows neither productive reasoning nor an efficiency benefit.

5.3 RELIABILITY INVERSION: ATWD VS. PLAIN BASELINE

Table 2 shows the scalar baseline ahead of ATWD on mean sequence accuracy (0.269 vs. 0.248, $+8.2\%$ relative) and on two of three seeds individually, losing only at seed 0 by a narrow margin (0.2675 vs. 0.2713). Its sample standard deviation is also smaller (0.003 vs. 0.019 for hard-freeze and 0.026 for ATWD, ratios of roughly 5.9–$7.9\times$ on the per-seed values), so both modified arms are more variable here, reversing the A_5 comparison's direction (Figure 1).

5.4 GRACEFUL FAILURE: CONSTRAINT VIOLATIONS

Constraint violations come from a separate EMA-checkpoint audit (Section A.12), whose solved counts differ slightly from the W&B evaluations in Table 2; conditional means (overall mean over one minus that run's exact accuracy) use only this audit's predictions. Hard-freeze has fewer violations both overall and among failures (21.9%/20.7% fewer than baseline/ATWD), on every seed individually. This is a descriptive trade-off, not causal evidence of commitment, since lower duplicate counts do not measure distance to validity and each arm fails on a different puzzle subset.

5.5 MECHANISTIC EVIDENCE

Replaying trained checkpoints on individual test examples gives token-level evidence: Figures 4 and 6 show the same example in two views: all 81 answer cells and 15 of 16 workspace positions

remain below threshold throughout the final window, while one workspace position crosses it intermittently – heterogeneous convergence, not a persistently active scratchpad. Across four puzzles under two checkpoints (seeds 0 and 2), six of eight puzzle–checkpoint pairs show near-total sustained convergence and two show partial convergence, not explained by difficulty (ratings 1, 3, 60, 1) but by which checkpoint solves the puzzle: puzzle 1 converges partially under seed 0 but cleanly under seed 2, and puzzle 3 shows the reverse (Figure 7).

These paired replays complement Section 5.3's aggregate finding at the puzzle level, without establishing that convergence variability causes accuracy variability. Table 7 summarizes three further associations (ATWD's earlier saturation; hard-freeze's step-count volatility and fewer failure violations) that motivate ablations, not a causal sequence. Replay methodology and sample-size caveats are in Appendix A.6.

6 Discussion

6.1 Why Does S_5 Fail, and Why Does Sudoku's Reliability Benefit Invert?

We can characterize both open questions from Section 5 more precisely than we can fully resolve them.

For S_5, neither the scalar-shaped replays nor the separate prediction inspections authenticate native ATWD replay settings or isolate the cause. S_5's larger group order is one plausible contributor among several (group structure, solver settings, training dynamics); a controlled intervention would be needed to attribute mode collapse to any one of them.

For the Sudoku reliability inversion, the more structurally grounded account concerns *coupling*: causal attention removes backward influence but not all cross-token dependence, whereas under bidirectional attention every token's trajectory can influence, and be influenced by, every other token's – so which tokens get damped, and when, is a tightly coupled, path-dependent process in which small early-training differences could plausibly be amplified far more than in the causal case. One specific, checkable hypothesis we did not confirm: periodic refresh – the very mechanism preventing permanent freezing – unconditionally resets every token's step-size every R iterations regardless of how close it is to a not-yet-settled solution, and could disrupt genuine partial progress. A spot-check of two non-converging examples showed spikes suggestively, but not conclusively, aligned with the refresh period; we report this as an open, checkable hypothesis rather than a demonstrated mechanism.

These hypotheses motivate studying attention coupling but do not establish it as the cause: the tasks also differ in data, objectives, budgets, and solver settings, and S_5 fails despite causal attention. The practical implication is to validate both accuracy and variability on the target task rather than extrapolate from the favorable A_5 result.

6.2 Numerical Convergence Is Not Correctness

A single caveat recurs across this paper's non-uniform findings: FPRM's halting rule (Equation (2)) reads convergence directly off the iterate's own numerical stability – it cannot distinguish a token stabilized on the correct answer from one stabilized on a wrong but self-consistent one. This is the limitation illustrated by S_5-ATWD (Section 5.1), where residuals fall quickly while few classes are predicted. A related pattern appears in Sudoku: for the initially inspected puzzle, seed 2 has all 81 grid cells below threshold throughout the final window, versus 80 for seed 0, yet its exact accuracy is lower (0.2200 vs. 0.2713) – a single-puzzle diagnostic, not a measured correlation over the test set, and the expanded four-puzzle analysis shows mixed checkpoint-dependent convergence (Section 5.5). Residual stabilization should therefore be distinguished from verified correctness, not treated as a certificate of a valid solution.

6.3 ATWD Is Not (Yet) a Compute Speedup

ATWD, as presented here, is a mechanistic and accuracy-oriented contribution, not a wall-clock efficiency one: per-token damping changes the *magnitude* of a token's update once its step-size has decayed, but does not skip computation for it, since the shared block f_θ still evaluates every token at every iteration to produce $\hat{y}_i$ (Equation (1)) regardless of step-size. An actual FLOPs reduction

would need an additional mechanism – e.g., excluding damped-below-floor tokens from the shared block's attention and MLP entirely, analogous to the KV-cache-reuse strategies AdaPonderLM and PonderLM-3 (Section 2) use for their learned gates – which we flag as future work, not an implicit promise of this design.

6.4 External Support and Its Limits

Related work motivates, rather than validates, our mechanism. A statistical-mechanics analysis of Sudoku identifies a genuine *freezing transition*, where an extensive fraction of cells become fixed across all solutions as constraint density increases (Prates & Lamb, 2018); a self-supervised Transformer-based CSP solver reports, independently, that deterministic per-variable updates converge faster initially but are more prone to premature stagnation than stochastic ones (Xu et al., 2025). Neither was designed with fixed-point reasoning models in mind, and backbone variables fixed across valid solutions differ from solver states frozen at incorrect values.

7 Conclusion

We introduced Adaptive Token-Wise Damping (ATWD), a minimal, parameter-free extension of FPRM's scalar damping to operate independently per token, paired with a periodic refresh giving a scheduling guarantee (Bounded Staleness, Proposition 3.1) rather than a dynamical-stability one. Across three tasks (Section 5), ATWD's benefits are real but task-dependent: it improves accuracy and cross-seed stability on A_5; fails to match baseline's S_5 generalization transition within budget; and on Sudoku beats a hard-freeze ablation on every seed while trailing the scalar baseline in mean accuracy and stability – an inversion of the A_5 pattern whose cause we do not establish (Section 6).

These outcomes make a task-dependent empirical case for token-wise damping, not a general advantage over scalar halting. Combining per-token damping with the Jacobian/spectral-radius regularization in Section 2, resolving the refresh-disruption hypothesis from Section 6, and realizing an actual compute saving from a damped token's step-size – not just a change in update magnitude – are the three directions we consider most worth pursuing next.

Limitations and Future Work

We summarize several boundaries of this study. All results use a single base architecture (FPRM), untested against other fixed-point or deep-equilibrium architectures, or FPRM's own maze and ARC-AGI benchmarks, for lack of compute budget. ATWD inherits its solver hyperparameters (τ, P, δ) unchanged from FPRM's own tuning (Table 5) rather than re-optimizing for the per-token setting, so its true best-case performance may differ from what is reported. The refresh interval $R{=}20$, calibrated on a single A_5 seed (Section A.3) and reused elsewhere without task-specific tuning to avoid chasing a preferred outcome, leaves open whether a task-specific R would change the Sudoku reliability inversion (Section 5.3). Matching epochs and batch size does not match realized computation, since arms execute different solver-step counts and wall-clock time; comparisons hold schedules fixed, not FLOPs. The Sudoku ablation also changes decay and refresh together (Section A.10's factorial extension would separate the two effects). Finally, small seed counts and replayed-puzzle samples limit statistical resolution – per-cell and sequence metrics share predictions rather than being independent replications, and the two replays per puzzle are not independent test-distribution draws – so larger seed sets, paired per-puzzle predictions, seed-vs-sampling uncertainty estimates, and failure-severity comparisons on puzzles all arms fail (not each arm's own subset) would strengthen the empirical case.

Two extensions follow: an adaptive or per-example refresh interval, motivated by the unconfirmed refresh-disruption hypothesis (Section 6), and a causal-vs-bidirectional comparison on tasks of intermediate coupling that controls for difficulty, budget, and solver settings.

Ethics Statement

This work involves no human subjects, crowdsourced annotation, or personally identifiable data. The state-tracking datasets (A_5, S_5) are generated synthetically from finite-group multiplication tables by

a fixed, deterministic algorithm; the Sudoku data derives from the publicly available Sudoku-Extreme benchmark, a puzzle dataset containing no personal, sensitive, or restrictively licensed content. We do not identify a specific foreseeable dual-use or societal-harm risk beyond those already broadly associated with more capable or more efficient general-purpose reasoning models, and we are not aware of any aspect of this work – a mechanistic study of a halting rule for looped Transformers on synthetic algorithmic tasks and a recreational puzzle – that raises concerns distinct from the existing literature it builds on. We have read the ICLR Code of Ethics and confirm that this submission adheres to it.

Reproducibility Statement

We report exact dataset-generation commands (Section A.11), the complete hyperparameter and compute budget for every arm and task (Table 5), and the full software and hardware environment for every reported run (Section A.11). Section A.2 documents the four correctness tests used to verify the per-token damping mechanism's equivalence, reduction, refresh, and heterogeneous-token behavior before any training run was launched. Section A.8 gives the exact outer-step/solver-iteration arithmetic needed to reproduce our reported step counts, and Section A.9 gives the numerical argument used to rule out an alternative explanation for the Sudoku halting-budget result in Section 5.2. Code implementing ATWD and every analysis script used to produce this paper's figures and tables is available at an anonymized repository (Section A.12; link withheld during double-blind review) and will be made public upon publication. Every mean and standard deviation reported in this paper is computed over exactly the number of seeds stated for that arm (Section 4.3); no seed's result was rounded, omitted, or excluded post hoc.

AI Use Statement

In this work, we used generative AI tools (Claude, Anthropic) for tasks in every category the ICLR AI Policy for Authors marks as requiring disclosure, including: developing the theoretical framing of Adaptive Token-Wise Damping and formulating Proposition 3.1, including identifying the key ingredient of its proof (that the refresh trigger is a function of a shared counter alone, independent of any token's residual or patience history) and drafting the proof itself; proposing and iteratively revising research hypotheses – for example, the S_5 mode-collapse hypothesis in Section 5.1, which itself superseded an earlier stepsize-collapse hypothesis once replayed stepsize trajectories contradicted it, and the refresh-disruption hypothesis in Section 6, which we explicitly did not confirm and report only as an open question; designing and giving feedback on the experimental protocol, including the calibration procedure in Section A.3 and the decision to hold solver hyperparameters fixed across arms (Section 4.3); generating the code underlying our synthetic state-tracking dataset generator and the Sudoku test-set truncation utility; implementing the Adaptive Token-Wise Damping modifications to the released FPRM solver and the accompanying correctness tests (Section A.2); and interpreting experimental results throughout, including the reliability-inversion and constraint-violation analyses in Sections 5.3 and 5.4. Of the policy's disclosure categories, two are inapplicable to this work and so were not used: assisting with translation, and transcribing recordings of research material. We also did not use generative AI tools to generate any reported metric, table value, or figure value directly: every number in this paper is read from a logged training run or computed by a deterministic analysis script.

Additionally, we used generative AI tools for tasks in the recommended-disclosure category, including drafting and editing most of the paper's prose; writing the plotting and analysis scripts used to produce every figure (the figures themselves plot real logged data, not AI-generated images); searching for and summarizing related work (Section 2); and suggesting the paper's structure, title, and keywords.

We have reviewed all AI-assisted work. Every reported number was cross-checked against its corresponding training log or raw output file before being written into a table or figure; as one example, an early version of our checkpoint-loading analysis script loaded the raw, non-EMA-averaged model weights while training itself logged EMA-evaluated metrics, a mismatch we caught and corrected before finalizing any Sudoku result (Section 4.2). A second, independent AI tool was used for a separate proofreading and numerical-consistency audit pass over the manuscript, and the proof of Proposition 3.1 was checked by hand against the refresh-trigger implementation in the

released code. We take responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

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

François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

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

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

Riccardo Grazzi, Julien Siems, Jörg Franke, Arber Zela, Frank Hutter, and Massimiliano Pontil. Unlocking state-tracking in linear rnns through negative eigenvalues. In *International Conference on Learning Representations*, 2025.

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

Wai-Lun Lam. Energy-entropy regularization: The true power of minimal looped transformers. *arXiv preprint arXiv:2601.09588*, 2026.

He Li, Feichen Song, Boyi Zeng, Shixiang Song, Zhiqin John Xu, Ziwei He, and Zhouhan Lin. Ponderlm-3: Adaptive token-wise pondering with differentiable masking. *arXiv preprint arXiv:2603.02023*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 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.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *International Conference on Learning Representations*, 2023.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 2019.

Boris T. Polyak and Anatoli B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

Marcelo O.R. Prates and Luis C. Lamb. Problem solving at the edge of chaos: Entropy, puzzles and the sudoku freezing transition. *arXiv preprint arXiv:1810.03742*, 2018.

Feichen Song, He Li, Ruoxuan Wang, Boyi Zeng, Shixiang Song, Chuang Wang, Kexin Xu, Ziwei He, and Zhouhan Lin. Adaponderlm: Gated pondering language models with token-wise adaptive depth. *arXiv preprint arXiv:2603.01914*, 2026.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical reasoning model. *arXiv preprint arXiv:2506.21734*, 2025.

Yudong W. Xu, Wenhao Li, Scott Sanner, and Elias B. Khalil. Self-supervised transformers as iterative solution improvers for constraint satisfaction. *arXiv preprint arXiv:2502.15794*, 2025.

Yijiong Yang, Xiaoyu Han, Wei Zhang, Zheng Wei, Yifan Shao, Yuxuan Guo, and Ming Li. Stabilizing recurrent dynamics for test-time scalable latent reasoning in looped language models. *arXiv preprint arXiv:2605.26733*, 2026.

A Additional Experimental Details

A.1 Solver Pseudocode

Algorithm 1 gives the full ATWD solver iteration corresponding to Equations equation 1–equation 7 in Section 3.

Algorithm 1 One ATWD solver iteration (per example)

Input: y_i, b_i, p_i, η_i, counter k_i
$\hat{y}_i \leftarrow f_\theta(y_i)$ {one shared-block forward pass}
for each token t **do**
  $r_i[t] \leftarrow \|\hat{y}_i[t] - y_i[t]\|_\infty / (\|\hat{y}_i[t]\|_\infty + \epsilon)$
  $y_{i+1}[t] \leftarrow \eta_i[t]\, \hat{y}_i[t] + (1 - \eta_i[t])\, y_i[t]$
  if $r_i[t] < b_i[t]$ **then**
    $b_{i+1}[t] \leftarrow r_i[t];\ p'[t] \leftarrow P$
  else
    $b_{i+1}[t] \leftarrow b_i[t];\ p'[t] \leftarrow p_i[t] - 1$
  end if
  if $p'[t] \le 0$ **and** $r_i[t] \ge \tau$ **then** {stuck, not converged}
    $\eta_{i+1}[t] \leftarrow \delta \cdot \eta_i[t];\ p_{i+1}[t] \leftarrow P$
  else
    $\eta_{i+1}[t] \leftarrow \eta_i[t];\ p_{i+1}[t] \leftarrow p'[t]$
  end if
end for
$k_{i+1} \leftarrow k_i + 1$
if $R > 0$ **and** $k_{i+1} \bmod R = 0$ **then** {unconditional refresh}
  $\eta_{i+1} \leftarrow \eta_0 \mathbf{1};\ p_{i+1} \leftarrow P\mathbf{1};\ b_{i+1} \leftarrow \infty\mathbf{1}$
end if

A.2 ATWD Correctness Tests

The supplied test source and recorded console output document four passing checks. Their assertion coverage is narrower than a proof of solver equivalence or exhaustive numerical correctness:

1. **Equivalence.** The scalar-path smoke test checks residual and step-size tensor shapes with token-wise damping disabled. It does not compare outputs with an independent copy of the original solver, so it does not establish full behavioral equivalence.
2. **Reduction.** With token-wise damping enabled, if every token receives identical proposals and identical initial states, the per-token and scalar solvers produce matching step-size trajectories within floating-point tolerance. Refresh is disabled in this test; residual trajectories are recorded but not explicitly compared.
3. **Periodic refresh.** Step-size resets to its initial value η_0 after each tested iteration whose completed-step counter is divisible by R. This check does not assert the absence of other resets or verify the patience and best-residual trackers.
4. **Heterogeneous tokens.** Given one token held static (residual near zero, effectively already converged) and a second perpetually reset to a fresh random target (residual persistently high), the already-converged token retains its full step-size while the non-converging token is damped once its patience is exhausted – checking the intended ordering of their final step-sizes under Equation equation 5.

A.3 Refresh Interval Calibration

The refresh interval $R{=}20$ used throughout Section 5 was selected via a brief, single-seed calibration sweep on A_5 at a short (2-epoch) training budget, not treated as a free parameter retuned per task or per result. Table 4 reports the outcome. Aggregate accuracy ranges from 0.2005 to 0.2148, with $R{=}20$ highest on both aggregate accuracy and the reported $k{=}58$ metric. A single short-budget seed

does not establish insensitivity to R or identify its best value after full training. We fixed this value for every subsequent A_5, S_5, and Sudoku ATWD run without further tuning.

Table 4: Single-seed, short-budget refresh-interval calibration sweep on A_5. `acc@k=58` is per-token accuracy at evaluation prefix length 58, a representative mid-range point.

R	eval/acc	acc@k=58	train/acc	train/loss
10	0.2097	0.4366	0.868	1.249
20	**0.2148**	**0.4538**	0.837	1.430
30	0.2116	0.4426	0.866	1.176
50	0.2005	0.4229	0.916	1.216

A.4 S_5 Training Schedule

Both S_5 arms were initially trained at the same 15-epoch schedule used for A_5; at that budget, baseline itself had not yet generalized (`eval/accuracy` ≈ 0.03, well below its later value). Because the state-tracking configuration uses a flat learning-rate schedule (no warmup, constant rate), extending training by resuming from the existing optimizer and dataset-position checkpoint introduces no learning-rate discontinuity, so we resumed both arms to 30 epochs rather than restart from scratch. Baseline subsequently underwent a sharp, late transition consistent with grokking (Power et al., 2022; Nanda et al., 2023) between epoch 15 and 30 (`eval/accuracy` $0.027 \rightarrow 0.477$ on seed 0; a similarly sharp transition on seed 1), the result reported in Table 1. ATWD showed no comparable transition over the same additional 15 epochs on either seed (`eval/accuracy` $0.0235 \rightarrow 0.0238$ on seed 0), despite training accuracy continuing to rise (Section 5.1). We made no further extension or arm-specific budget adjustment. The initial extension was informed by intermediate results, rather than prespecified before training.

Replay evidence for the mode-collapse hypothesis (Section 5.1). The seed-1 archives record 200 sequence-level residuals and scalar-shaped step-sizes for eight examples each (not token-resolved trajectories, and not authenticated as matching the original training-time replay settings). In the ATWD-trained checkpoint's replay, every residual stays below 0.1 from indices 5–10 onward (zero-based), with step-sizes fixed at 1.0 throughout – no damping event ever fires. Seven of eight baseline examples sustain the same threshold only from indices 122–131 onward; the eighth does not sustain it within the 200 recorded iterations, and baseline's final step-sizes span 0.859–0.941. All four final S_5 W&B summaries report `eval/steps`$= 160$ (the cap), so this rapid replayed convergence is not itself evidence of early native evaluation halting. Separate, guide-recorded prediction inspections (the archives retained no predictions to re-verify these counts) found 5–12 of 120 predicted classes used per example, with one class occupying 57–86% of positions despite varying targets.

A.5 Relocated Figures and Tables

The following items are relocated here for main-text space; each is discussed in full, and its finding stated in full, at the point in the main text noted in its caption.

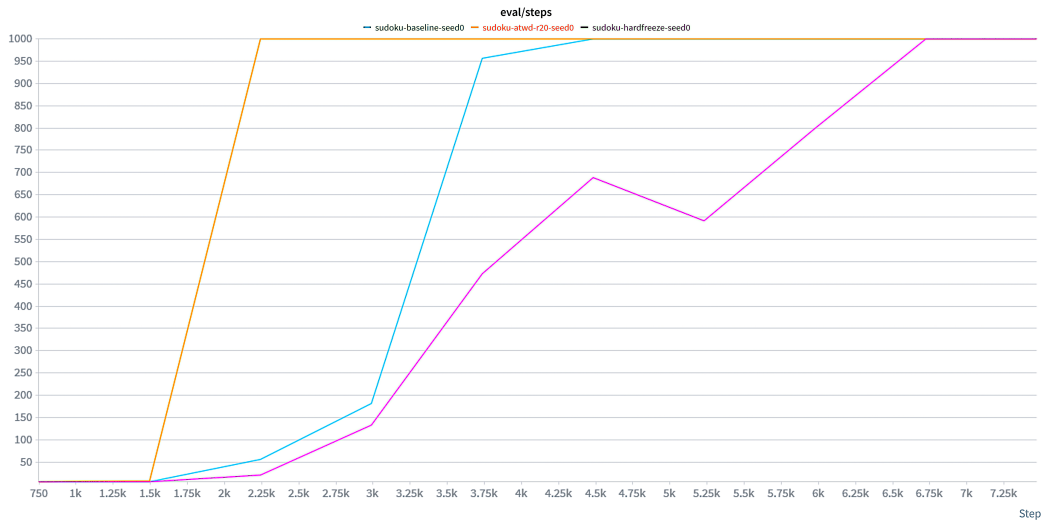


Figure 3: Fixed-point iterations used at halting (`eval/steps`), tracked across training, for all three arms at seed 0. ATWD saturates the 1,000-iteration cap almost immediately (≈2.2k training steps); baseline saturates later (≈4.5k); hard-freeze saturates slowest and least smoothly (≈6.75k). Lower is not better here – see discussion below.

Table 5: Full architecture and solver hyperparameters by task, including the per-run budgets summarized in Section 4; wall-clock is two-GPU elapsed time, excluding resumed-run idle time, set from a real throughput benchmark (Section 4.2). Values apply to baseline and ATWD; hard-freeze instead uses $\delta_{\text{train}} = 0$ and no refresh (Section 4.3). Training outer steps contain 4 or 6 solver iterations; evaluation outer steps contain one (Section A.8).

	A_5	S_5	Sudoku
Attention	causal	causal	bidirectional
Tokens T (train / eval)	32 / 128	32 / 128	97 / 97
Hidden size	512	512	512
H-/L-cycles	3 / 4	3 / 4	3 / 6
Grad-tracked steps	4	4	6
Train / eval max outer steps	128 / 160	128 / 160	12 / 1000
τ (fp_thresh)	0.1	0.1	0.1
δ (train / eval)	0.90 / 0.99	0.95 / 0.99	0.90 / 0.99
Patience P	5	10	5
Refresh R (ATWD only)	20	20	20
Batch size	832	832	736
Epochs	15	30	5,500
Learning rate	1e-4	1e-4	1e-3
Train examples	500,000	500,000	1,000×1,000 aug.
Seeds per arm	3	2	3
Wall-clock time/run	≈11.1h	≈22.0h (2 sessions)	≈10.5h

Table 6: Full per-seed breakdown of Table 2 (9×9 Sudoku, 800 held-out puzzles, 5,500-epoch schedule). Mean and sample SD use unrounded W&B summaries; individual entries are rounded for presentation.

ARM	SEQ. ACCURACY	PER-CELL ACC	LM LOSS
HARD FREEZE (S0)	0.1900	0.7255	0.839
HARD FREEZE (S1)	0.2200	0.7304	0.827
HARD FREEZE (S2)	0.1838	0.7137	0.729
BASELINE (S0)	0.2675	0.7329	0.582
BASELINE (S1)	**0.2725**	0.7324	**0.578**
BASELINE (S2)	0.2663	0.7302	0.588
ATWD (S0)	**0.2713**	**0.7334**	**0.582**
ATWD (S1)	0.2538	0.7265	0.595
ATWD (S2)	0.2200	0.7144	0.623
HF MEAN	0.198 ± 0.019	0.723 ± 0.009	0.798 ± 0.061
BASE MEAN	$\mathbf{0.269 \pm 0.003}$	$\mathbf{0.732 \pm 0.001}$	$\mathbf{0.583 \pm 0.005}$
ATWD MEAN	0.248 ± 0.026	0.725 ± 0.010	0.600 ± 0.021

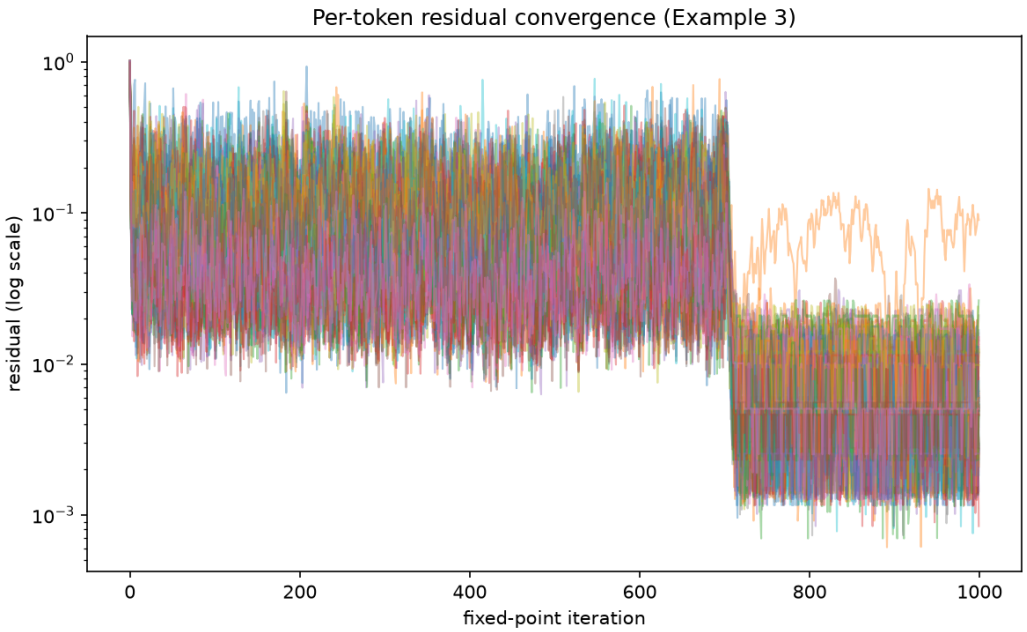


Figure 4: Per-token residual trajectory for a representative Sudoku example (ATWD, seed 0). Almost every one of the 97 positions spends roughly the first 700 iterations in a dense, noisy plateau before a sudden, lasting transition into a tight converged band. One position (visible oscillating above the band after the transition) does not fully settle.

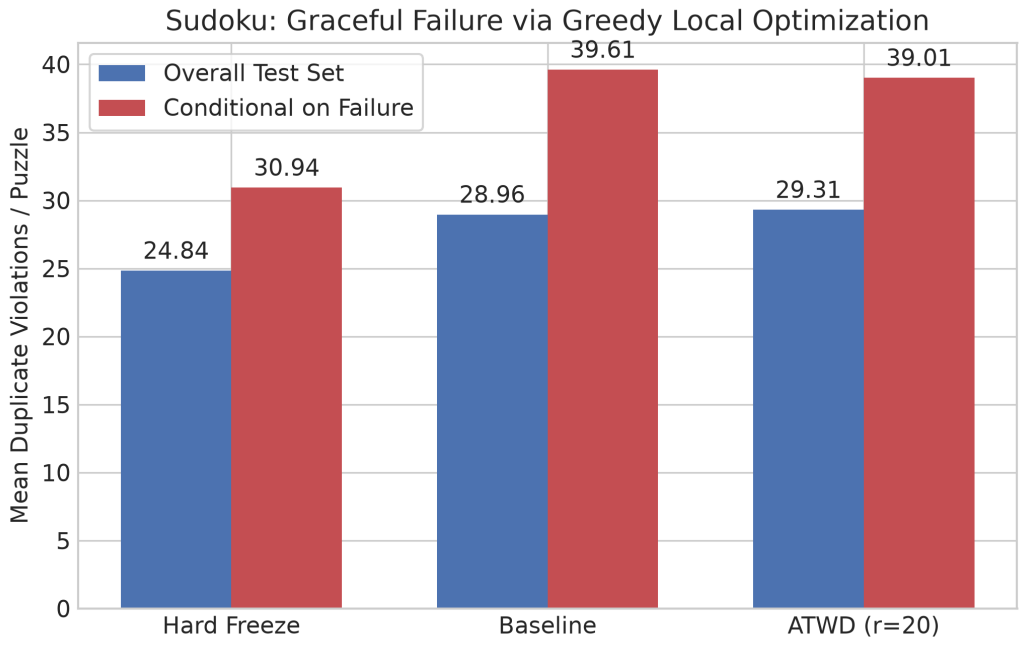


Figure 5: Mean constraint violations per puzzle, overall and restricted to puzzles each arm failed to solve exactly, averaged over three seeds per arm. Hard-freeze is worst on accuracy (Table 2) but produces the cleanest failures.

Table 7: Diagnostic profile of fixed-point optimization dynamics on Sudoku, consolidating findings established in Sections 5.2 and 5.5.

METRIC	HARD FREEZE	BASELINE	ATWD ($R = 20$)
HALTING-BUDGET SATURATION	SLOWEST (≈6.75K STEPS)	INTERMEDIATE (≈4.5K STEPS)	**FASTEST** (≈2.2K STEPS)
TRAIN/STEPS VOLATILITY	ELEVATED / VOLATILE	LOW & STABLE	LOW & STABLE
VIOLATIONS IF FAILED (TABLE 3)	**30.97**	39.65	39.04

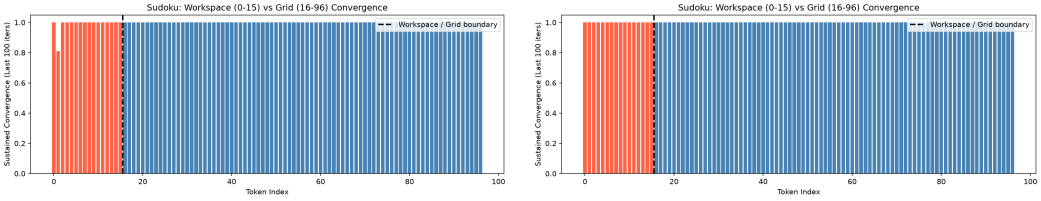


Figure 6: Fraction of the final 100 iterations each of the 97 sequence positions spends below τ, for two Sudoku examples under the seed-0 checkpoint. **Left (Example 3):** the same example as Figure 4; all 81 grid-cell positions (16–96) sustain convergence for the full window, and exactly one of the 16 puzzle-embedding "workspace" positions (0–15) does not (≈80%) – the same position still visibly oscillating in Figure 4. **Right (Example 2):** all 97 positions, including every workspace position, sustain convergence for the entire final 100 iterations. Six of eight puzzle–checkpoint pairs show near-total convergence like Example 2; the exceptions are puzzle 1 under seed 0 (Figure 7, right) and puzzle 3 under seed 2 (not shown).

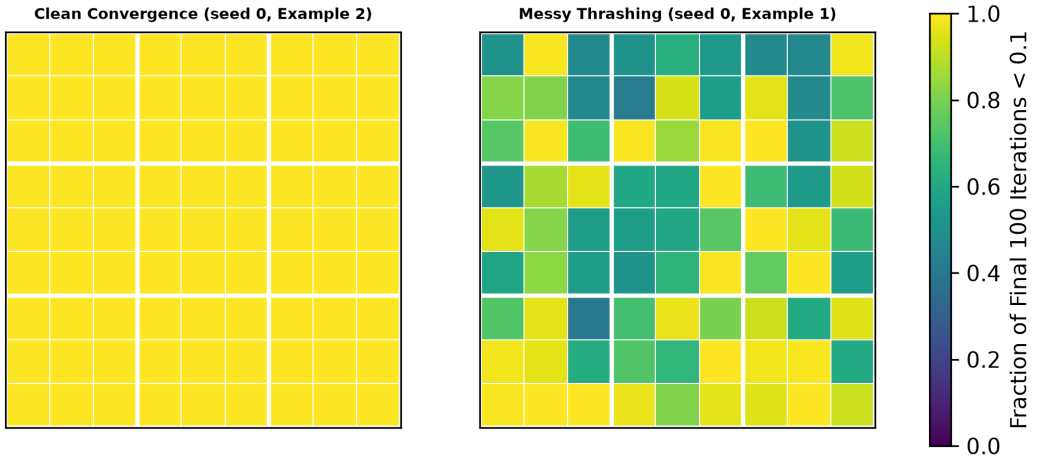


Figure 7: Per-cell sustained-convergence heatmaps under the seed-0 checkpoint. Left: puzzle 2, whose grid remains below threshold throughout the final 100 iterations. Right: puzzle 1, with partial convergence across the grid. Under seed 2, puzzle 1 converges cleanly while puzzle 2 remains clean; the reverse seed dependence occurs on puzzle 3 (not shown).

A.6 Additional Convergence Example

The second Sudoku example (seed 0) referenced throughout this appendix is now shown directly alongside the first in Figure 6 (right panel) rather than as a separate figure here.

The convergence figures in Section 5.5 derive from a fixed set of eight puzzle–checkpoint replays (the same four Sudoku puzzles under two checkpoints, not eight independent checkpoints or puzzles), inspected as given rather than selected from a larger pool of favorable examples; this small sample limits conclusions about how frequently these patterns occur across the test distribution.

A.7 Notation Summary

Table 8 summarizes the symbols used throughout Section 3.

Table 8: Notation used in the method description.

Symbol	Meaning
T	number of token positions
d	hidden dimension
$t \in [T]$	token index
i	solver iteration index
$y_i[t]$	fixed-point iterate at token t, iteration i
$\hat{y}_i[t]$	shared block's raw, undamped proposal
$r_i[t]$	per-token residual (Equation (1))
r_i	sequence-level residual, $\max_t r_i[t]$
τ	halting threshold on the residual
$\eta_i[t]$	damping step-size
η_0	initial step-size
δ	step-size decay rate
P	patience (iterations before adapting)
$b_i[t]$	best residual seen since the last reset
$p_i[t]$	patience counter
$a_i[t]$	adaptation indicator (Equation (5))
R	periodic refresh interval
k_i	global refresh counter (per example)
f_θ	shared block (weight-tied across iterations)

A.8 Outer Steps and Solver Iterations: A Worked Example

Table 5's "outer step" and "solver iteration" terminology is easy to conflate, so we spell out the arithmetic once here. During training, one outer step corresponds to n_{back} solver iterations executed with gradients tracked ($n_{\text{back}} = 4$ for state tracking, $n_{\text{back}} = 6$ for Sudoku); during evaluation, one outer step corresponds to exactly one solver iteration, since gradients are not tracked and the fixed-point loop runs directly. Consequently, a state-tracking run's train/eval outer-step caps of $128/160$ correspond to at most $128 \times 4 = 512$ gradient-tracked solver iterations during training and exactly 160 solver iterations during evaluation, while a Sudoku run's caps of $12/1000$ correspond to at most $12 \times 6 = 72$ gradient-tracked training iterations and up to $1{,}000$ evaluation iterations – the latter matching the iteration cap referenced throughout Section 5.2 as `eval/steps`. The asymmetry (few, expensive training iterations; many, cheap evaluation iterations) reflects that only evaluation needs to run the solver to convergence; training truncates at a fixed, gradient-tracked budget by design.

A.9 Numerical Robustness of the Evaluation-Decay Floor Argument

Section 5.2 argues that the hard-freeze step-size floor cannot explain the observed `eval/steps` saturation order under the documented evaluation settings. We state this argument formally here.

Lemma A.1 (bfloat16-robustness of the floor argument)**.** *Let $\delta_{\text{eval}} = 0.99$, patience $P = 5$, and evaluation iteration cap $N_{\max} = 1{,}000$. At most $\lfloor N_{\max}/P \rfloor = 200$ decay events can occur for any token or example, since Equation* (5) *requires P consecutive non-improving iterations per event. In exact real arithmetic, the largest possible cumulative decay is $\delta_{\text{eval}}^{200} = 0.99^{200} \approx 0.134$. Representing δ_{eval} in `bfloat16` rounds it to $\widehat{\delta} = 0.98828125$ (relative error at most 2^{-8} per multiplication); compounding this worst-case rounding error over 200 multiplications gives $\left(\widehat{\delta} \cdot (1 - 2^{-8})\right)^{200} \approx 0.0433$. Both bounds exceed the hard-freeze floor of 10^{-3} by more than an order of magnitude.*

Under the evaluation settings used throughout this paper, the step-size floor therefore cannot be the mechanism by which any evaluated arm halts early; the saturation order in Figure 3 reflects the residual criterion, on this quantitative basis rather than as an assumption.

A.10 Toward a Factorial Ablation of Decay and Refresh

Sections 4.3 and 6 note repeatedly that the Sudoku hard-freeze arm changes training-time decay and periodic refresh together, so it cannot by itself isolate refresh's individual contribution. Table 9 makes the missing comparison explicit: a fourth arm, matched to ATWD's decay rate but with refresh disabled, would let the difference between arms (2) and (3) isolate the effect of periodic refresh alone, holding decay fixed, while the difference between arms (1) and (2) isolates the effect of raising the decay rate alone, holding refresh fixed at zero.

Table 9: A factorial extension of the Sudoku ablation. Arm (4) is not run in this paper; it is the comparison needed to isolate refresh's individual contribution.

Arm	δ_{train}	R	Status
(1) Hard-freeze	0	0	run (Section 5.2)
(2) Matched-decay, no-refresh	0.90	0	*not run*
(3) ATWD	0.90	20	run (Section 5.2)

A.11 Software and Compute Environment

Table 10 records the exact software versions used for every reported run, as logged by Weights & Biases at run time.

Dataset-generation commands. `build_state_tracking_dataset.py --group {A5,S5} --k-train 32 --k-eval-max 128 --train-samples 500000;`

`build_sudoku_dataset.py --subsample-size 1000 --num-aug 1000.`

Sudoku dataset sizing. The 1,000-puzzle set comfortably clears the batch size of 736, since the data loader draws at most one example per base puzzle per pass; augmentations add training diversity at no extra optimization-step cost, since step count depends only on the base-puzzle count.

Optimization and checkpointing. AdamW with β_1=0.9, β_2=0.95; state tracking used weight decay 0.01, no warmup, and a constant learning rate, Sudoku weight decay 0.001 with a 2,000-step warmup. The solver's optional variational-dropout and additive-noise terms (both default 0) are disabled, hence inert, in every reported run. Training was launched with `torchrun --nproc-per-node=2`, checkpointing every `eval_interval` epochs (5 for state tracking, 550 for Sudoku) plus a weights-only checkpoint every 1,000 steps as protection against session timeouts.

Table 10: Software environment for all reported training runs.

Component	Version
Python	CPython 3.12.13
PyTorch	2.11.0+cu129
OS	Linux 6.12.90+ (glibc 2.35)
W&B client	0.26.0
Training GPUs	2×NVIDIA T4 (Kaggle free tier)
Local development GPU	NVIDIA RTX 3050 (smoke tests only)

A.12 Code and Data Availability

The implementation is available at an anonymized repository (link withheld for double-blind review) and will be made public upon publication. The research execution guide records training commands, schedule changes, and analysis corrections. Reproduction should use the run-specific configuration rather than the repository's default training budgets.