LOOP ATTENTION, NOT THE BLOCK

Anonymous authors
Paper under double-blind review

ABSTRACT

Recurrent-depth Transformers increase depth by repeating shared parameters, but most loop an entire block. That means recomputing both self-attention and feed-forward layers at every step, which inflates compute and often hurts language modeling. We move recurrence entirely inside self-attention. We introduce Gated Looped Attention (G-Loop), which keeps the feed-forward network outside the loop and repeatedly remixes the input values with the evolving state. Learned token- and channel-wise gates decide how much of each source to write at each step, trained end-to-end with backpropagation through time. As a result, a standard encoder Transformer handles both algorithmic reasoning and general language tasks without specialized routing and with modest parameter overhead. At 32 recurrent steps, G-Loop cuts inference FLOPs by up to 94.7% and memory by up to 73.9% compared to HRM and TRM-Att. It uses about 19% fewer parameters than TRM-Att to reach 85.9% on Maze-Hard and 72.5% on Sudoku-Extreme, beating its reported Maze score and closely approaching its Sudoku score. Under our cost protocol, the newer recurrent reasoners that score higher spend 57× to 4,900× as many inference FLOPs. Added to DeBERTa-v3-base, G-Loop improves the six-task GLUE development average by 1.25 points and SQuAD 2.0 F1 by 0.9 points over the baseline. Looping attention alone slashes compute, and gating the value pathway preserves the expressivity needed to reason.

1 INTRODUCTION

Many reasoning problems require large depth but not many parameters (Saunshi et al., 2025). Recurrent-depth Transformers exploit this by applying the same weights repeatedly, so that depth is set by the number of iterations rather than by the parameter count. The idea runs from Universal Transformers (Dehghani et al., 2019) and deep equilibrium models (Bai et al., 2019) through looped Transformers (Giannou et al., 2023; Yang et al., 2024) to recurrent-depth language models (Geiping et al., 2025; Zhu et al., 2025; Bae et al., 2025; Jeddi et al., 2026; Kapl et al., 2026). More recently, small recurrent models trained from scratch on about 1,000 examples have solved hard Sudoku and maze instances on which the reasoning LLMs evaluated alongside them score 0% (Wang et al., 2025; Jolicoeur-Martineau, 2025; Gao et al., 2025), and subsequent work has raised these scores further by improving signal propagation and by scaling test-time compute (Movahedi et al., 2026; Huang et al., 2026; Fein-Ashley & Rashidinejad, 2026).

Nearly all of these models loop an entire Transformer block, recomputing both self-attention and the feed-forward network (FFN) at every step. The FFN dominates the projection cost of that step: with the usual $4\times$ expansion it accounts for $8D^2$ of a block's $12D^2$ per-token projection cost at width D, and under backpropagation through time its activations are stored at every step. Looping also has a cost in language modeling: looped models reach worse perplexity than non-looped models of equal compute, even while reasoning better (Saunshi et al., 2025), and ALBERT's ablation suggests where such costs come from, since most of the accuracy lost by its fully shared encoder came from sharing the FFN, while sharing only the attention parameters cost little or nothing (Lan et al., 2020).

These results point to a narrower design. The FFN transforms each position on its own; exchanging information between positions, which iterative reasoning must repeat, is done by self-attention alone. We therefore confine recurrence to self-attention: within each layer, a damped attention update with one set of weights is applied for T steps, and the FFN runs once afterwards. This halves the per-token projection cost of a step, from $12D^2$ to $6D^2$ including the gate introduced below.

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.

Taking the FFN out of the loop changes what carries information between steps. Each attention step moves the state toward a mixture of value vectors, so the central design question becomes what those values are computed from. If they are read only from the evolving state, repeated averaging shrinks the differences between tokens, an effect analogous to the rank collapse that Dong et al. (2021) identified in pure-attention networks. We bound this contraction and prove that routing enters the bound only through how much the tokens' attention patterns overlap: the averaging that queries and keys control never enlarges the spread, and how much it can grow is bounded by the normalization and the value and output projections. The problem thus lies in the choice of values rather than in attention-only recurrence itself: a loop that re-reads its input escapes this contraction. Reading values only from the fixed layer input, which we call the *anchor*, does exactly that, but the values then never change, so the loop can recombine what it started with but cannot offer anything it derives as new content.

Gated Looped Attention (G-Loop) resolves this tension with a learned gate. At every step, a sigmoid gate computed from the anchor and the current state mixes the two, per token and per channel, and the mixture is projected to form the values. The gate acts only on the values: queries and keys are still computed from the evolving state, so the attention pattern changes from step to step. Each position thus decides how much of its original input to re-read and how much of what the loop has derived to pass on. Re-reading the input is familiar as *recall* in deep-thinking networks (Schwarzschild et al., 2021; Bansal et al., 2022) and as *input injection*, which adds it to the state (Bai et al., 2019; Yang et al., 2024; Jolicoeur-Martineau, 2025); G-Loop instead places it inside the value pathway, where attention routes it between positions like any other content. Using the same gate for input injection instead costs 5.1 to 7.4 points on Sudoku-Extreme at an identical parameter count.

We evaluate on Sudoku-Extreme and Maze-Hard (Wang et al., 2025). Their rules are fixed and each cell interacts with a small, fixed set of others, so what makes them hard is the depth of propagation (and, for Sudoku, of search) needed to resolve dependencies across the grid. We do not use ARC-AGI (Chollet, 2019), where the scores of current recurrent reasoners rest heavily on per-task identifier embeddings, augmentation, and test-time voting (ARC Prize Foundation, 2025; Roye-Azar et al., 2025), none of which is the recurrence itself. To show that attention-only recurrence is not specific to puzzles, we also replace the self-attention sublayers of the pretrained DeBERTa-v3-base encoder (He et al., 2023) with G-Loop and compare it with the unmodified encoder on GLUE (Wang et al., 2019) and SQuAD (Rajpurkar et al., 2018).

The results bear out this reasoning. With 5.7M parameters, G-Loop edges past the attention-based TRM (Jolicoeur-Martineau, 2025) on Maze-Hard and comes within 2.2 points of it on Sudoku-Extreme at an order of magnitude fewer inference FLOPs; of the more recent recurrent reasoners that surpass it in accuracy, those we measured do so at 57× to 4,900× as many inference FLOPs (§3.2). Controls with our own model isolate the design: looping the FFN as well costs up to 2.2× more compute for no gain in accuracy, and an untied encoder with 15× the parameters solves no puzzle. And the recurrence transfers: DeBERTa-v3-base with G-Loop in place of its attention, after the same continued pretraining and fine-tuning as DeBERTa-v3-base itself, is ahead of it on all eight language tasks, by 1.25 points on the six-task GLUE average.

Taken together, the evidence supports a simple recipe: loop only the attention, and let a learned gate decide how much of the input each step re-reads. We make three contributions:

- **Attention-only recurrence.** We loop self-attention alone and apply the FFN once per layer, halving the per-step projection cost at no loss in accuracy. One recurrence serves both regimes: a 5.7M-parameter model on the accuracy–compute frontier of Sudoku-Extreme and Maze-Hard, and gains on every GLUE and SQuAD task when dropped into DeBERTa-v3-base.

- **An analysis of token collapse.** We prove that a damped attention-only loop whose values come from the state alone shrinks the differences between tokens geometrically unless its projections amplify them, and that the averaging itself cannot enlarge them, because the attention pattern enters the bound only through how much its rows overlap. Trained state-only models show this collapse.

- **A gated value pathway.** G-Loop gates, per token, channel, and step, how much of the anchor enters the values. Ablations attribute the rise from 0% to 72.5% on Sudoku-Extreme to this gate rather than to added parameters or input injection alone.

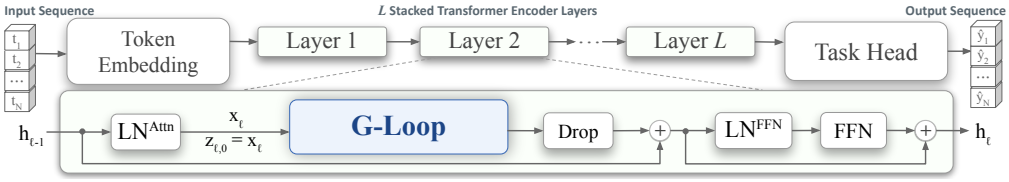


Figure 1: **G-Loop in an encoder layer.** G-Loop replaces the self-attention sublayer: from the anchor x_ℓ it iterates attention for up to N steps, after which the residual and the FFN are applied once.

2 Method

2.1 Attention-Only Recurrence

In a Transformer layer, self-attention is the only operation that moves information between tokens; the FFN transforms each token on its own. Extra depth pays off when a token's output depends on information that must travel through other tokens first, so we place recurrence on the exchange alone: G-Loop replaces the self-attention sublayer of a standard pre-norm encoder layer (Figure 1) and iterates a single attention step with shared weights, while the FFN still runs once per layer.

Layer ℓ normalizes its input into an *anchor* $x_\ell = \mathrm{LN}_\ell^{\mathrm{attn}}(h_{\ell-1})$, sets the state $z_{\ell,0} = x_\ell$, and updates it for $k = 0, 1, \ldots, N-1$, or until early exit stops the loop sooner (§2.3). Weights are shared across steps but not across layers. The final state z_ℓ replaces the attention output of a standard layer, and the rest of the layer is unchanged:

$$u_\ell = h_{\ell-1} + \mathrm{Drop}(z_\ell), \qquad h_\ell = u_\ell + \mathrm{FFN}_\ell\big(\mathrm{LN}_\ell^{\mathrm{ffn}}(u_\ell)\big). \tag{1}$$

With L=4 layers and N=32, a forward pass performs at most 128 attention steps but only four FFN evaluations. Because the FFN and the outer residual act only on the loop's output, neither can shape what happens inside it, and the layer input reaches later steps only if each step reads the anchor explicitly; what each step reads is therefore the central design question.

For grid puzzles, the n cells are flattened into a sequence (n=81 for Sudoku, n=900 for a 30×30 maze) with two-dimensional rotary position embeddings (Su et al., 2024; Heo et al., 2024), and a linear head predicts each cell. For language, we start from the pretrained 12-layer DeBERTa-v3-base encoder (He et al., 2023) and loop the attention sublayer of every layer but the last, which the enhanced mask decoder reuses; the language loop is G-Loop with a fixed damping and keys read from the anchor (Appendix G).

2.2 Why a Damped State-Only Loop Can Lose Token Differences

The most direct way to loop attention is to recompute everything from the current state, values included: $v_k = W^V \bar{z}_k$, where $\bar{z}_k$ is the normalized state. We call this the *state-only* loop. In our trained controls it fails and its tokens become increasingly alike; this section gives the mechanism and a condition under which the damped update produces it.

To see why, recall what one attention step does. Queries and keys produce an attention matrix A whose row i says how much token i reads from every other token (its *routing*); the values say what each token offers (its *content*); and token i receives $(Av)_i$, the average of the values weighted by its row of A. Averaging makes tokens alike: two tokens that read the sequence in similar ways receive similar averages, whatever the values are. We measure how different the tokens are by the *spread* $\mathrm{diam}(Z) = \max_{i,j} \|z_i - z_j\|_2$, which is zero exactly when all tokens are identical. How much one step of averaging shrinks it depends on how much the tokens' attention overlaps. For tokens i and j, the sum $\sum_r \min(A_{ir}, A_{jr})$ is the attention mass they place on the same positions. Let $\delta(A)$ be the smallest such overlap over all pairs. Dobrushin's classical bound (Dobrushin, 1956) then says that for any values Y,

$$\mathrm{diam}(AY) \le \big(1 - \delta(A)\big)\,\mathrm{diam}(Y),$$

that is, one round of averaging shrinks the spread by at least the overlap of the two least similar readers. The unmasked softmax attention used here gives every token positive weight, so $\delta(A) > 0$

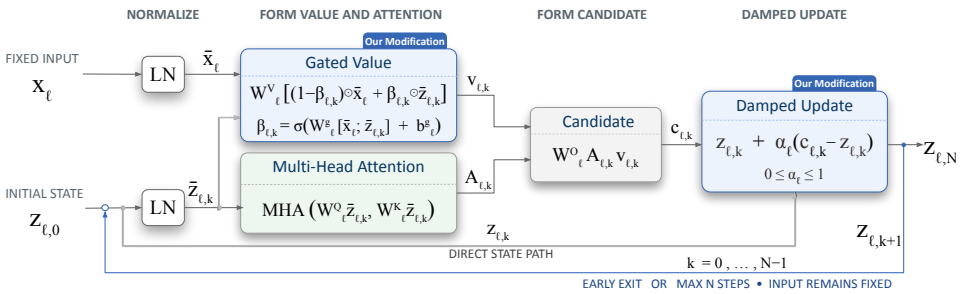


Figure 2: **One G-Loop step.** The fixed anchor x_ℓ and the current state $z_{\ell,k}$ are normalized. Queries and keys from the state produce the attention pattern $A_{\ell,k}$, which decides where information flows. The gated value blends anchor and state, per token and channel, and decides what flows. Attention applied to this value and projected by W_ℓ^O gives the candidate $c_{\ell,k}$, and the damped update moves the state a learned fraction α_ℓ of the way toward it. The step repeats for $k = 0, \ldots, N-1$ with the anchor held fixed, unless early exit ends the loop sooner. The two blocks marked *Our Modification* are the changes relative to a looped attention sublayer.

and every averaging step strictly shrinks the spread of what it averages, though by a factor that may be arbitrarily close to one.

A standard Transformer block adds the attention message to the old state through a unit-weight residual path; for an additive update $z_{k+1} = z_k + \alpha c_k$ the same argument yields a factor $1 + \alpha\mu_k$, with μ_k as in Eq. (2) below, and certifies no contraction. This is consistent with Dong et al. (2021), where skip connections counteract the rank collapse of pure attention stacks (related analyses study oversmoothing in deep attention stacks (Shi et al., 2022; Noci et al., 2022; Geshkovski et al., 2023; Wu et al., 2024)). Our damped update instead replaces an α-fraction of the old state by the attention candidate (§2.3), so the factor becomes $1 - \alpha + \alpha\mu_k$, below one whenever $\mu_k < 1$; and with state-only values every step draws its values from the current state alone, so any loss of spread is fed back into the next step's source, and whether it compounds depends on the amplification factors in μ_k. Proposition 2 (Appendix A) makes this exact:

$$\operatorname{diam}(z_{k+1}) \le \big(1 - \alpha(1 - \mu_k)\big) \operatorname{diam}(z_k), \qquad \mu_k = \Lambda_k \, \kappa_V \, \omega \, (1 - \delta_k), \tag{2}$$

where $\alpha \in (0, 1)$ is the damping, δ_k the smallest attention overlap over heads at step k, Λ_k the realized amplification of the spread by the normalization, and κ_V and ω bounds on its amplification by the value and output projections. If $\mu_k \le \bar{\mu} < 1$ at every step, then $\operatorname{diam}(z_K) \le (1 - \alpha(1 - \bar{\mu}))^K \operatorname{diam}(z_0)$; with G-Loop's per-group damping the same bound holds with $\alpha(1 - \mu_k)$ replaced by $\alpha_{\min} - \alpha_{\max}\mu_k$, so contraction then requires $\mu_k < \alpha_{\min}/\alpha_{\max}$ (Appendix A). Every factor in the condition is measurable on a trained trajectory, and the trained state-only model shows the failure mode analyzed here: its spread falls to 4.7% of its initial value in 32 steps (§3.4).

The form of μ_k also says where a fix can come from. In this bound, routing enters only through the overlap δ_k: queries and keys can loosen the factor by making the attention rows more distinct, but the averaging itself never makes its output more spread out than the values it receives. Routing only redistributes what the values carry, so a persistent source of token differences that attention is to pass between tokens must enter through the values.

A second source of content. The simplest such fix lets the values read the anchor as well. If they read a fixed blend $(1 - \gamma)\bar{x} + \gamma\bar{z}_k$ of the normalized anchor and the state, the anchor turns the homogeneous upper-bound recursion of Eq. (2) into an affine one, $\operatorname{diam}(z_{k+1}) \le a_k \operatorname{diam}(z_k) + b_k$ with b_k proportional to the anchor's spread, so the bound need not decay to zero even when $a_k < 1$ (Proposition 4). An upper bound cannot prove that differences survive, but the homogeneous contraction argument no longer applies. A fixed γ, however, imposes the same compromise on every token at every step. G-Loop lets each token choose.

2.3 Gated Looped Attention

Figure 2 shows one step. It reads the fixed anchor and the current state, both normalized, and produces the next state in four stages: the state alone decides routing, through its queries and keys;

the value, the content that attention moves, is a gated blend of anchor and state; attention applied to the value gives a candidate; and the damped update moves the state part of the way toward it. The gated value and the damped update are new; the other stages are standard components configured to stay well behaved over many steps. We drop the layer index ℓ below.

Normalize and route. Both inputs pass through the same normalization, and routing is computed from the state:

$$\bar{x} = \mathrm{LN}^{\mathrm{loop}}(x), \qquad \bar{z}_k = \mathrm{LN}^{\mathrm{loop}}(z_k), \qquad A_k = \mathrm{MHA}\big(W^Q \bar{z}_k,\ W^K \bar{z}_k\big).$$

Sharing the normalization puts anchor and state on the same scale, so the gate below compares like with like. A_k holds the attention matrices of H heads, each a softmax over the n tokens acting on its own $d_h = D/H$ channels of the value; it is recomputed at every step and uses no causal mask, so routing can follow the computation as it unfolds. The query, key, and output projections are spectrally normalized (Miyato et al., 2018), which bounds the attention logits (Appendix A) and holds ω in Eq. (2) near one.

Gated value. The value blends anchor and state, token by token and channel by channel, with a gate that reads both:

$$\beta_k = \sigma\big(W^g[\bar{x}; \bar{z}_k] + b^g\big) \in (0,1)^{n \times D}, \qquad v_k = W^V\big[(1-\beta_k) \odot \bar{x} + \beta_k \odot \bar{z}_k\big].$$

At $\beta \equiv 1$ this is the state-only loop of §2.2; at $\beta \equiv 0$ the values come from the anchor alone and stay fixed, so the loop can re-route but never update what it moves. The gate is a GRU-style update gate (Cho et al., 2014; Srivastava et al., 2015), as in the slot updates of Locatello et al. (2020), but acting on the values rather than on the state; it chooses between the two separately for every token and channel, and can change its choice from one step to the next: in a partly solved Sudoku, a cell that has just been determined can both change how much others attend to it, through its key, and offer its new conclusion as content, through a high β, while a cell that is still open keeps offering its given. Because the gate reads both operands, it can tell where the state has moved away from the anchor and hand over to it there. Two details keep the change small. The blend is formed before W^V, so anchor and state share one value projection and the gate adds only W^g and b^g, $2D^2{+}D$ parameters per layer; and queries and keys are untouched, so the gate changes what attention moves but never where. Unlike input injection and recall, which add the re-read input to each token's own state (Bai et al., 2019; Yang et al., 2024; Jolicoeur-Martineau, 2025; Bansal et al., 2022), or internal recall, which lets it condition the update (Labovich, 2026), the gated value makes the anchor content that attention routes, so a token can read other tokens' anchors, not only its own; §3.3 compares it with fixed, learned-scalar, per-channel, and linear blends, and with the same gate used for input injection.

Damped update. Attention applied to the value gives the candidate, which is what a standard attention sublayer would output. Instead of adding it to the state, we move the state part of the way toward it:

$$c_k = W^O A_k v_k, \qquad z_{k+1} = z_k + \alpha \odot (c_k - z_k), \qquad \alpha_h = \sigma(\eta_h).$$

This damped fixed-point iteration (Mann, 1953) is not required to converge. The step size is learned, with one coefficient α_h per group of d_h state channels, indexed by head, shared by all tokens and all steps; η_h is an unconstrained scalar initialized at zero, so every head starts at $\alpha_h = \frac{1}{2}$ and can learn to move faster or slower than the others. The new state is a convex combination of the old state and the candidate, so it stays on the scale of its inputs however many steps the loop runs, whereas an additive residual accumulates every update; this lets the loop run for a different number of steps at test time and makes early exit meaningful. The price is the one §2.2 describes: a damped update does not by itself preserve token differences, which is exactly what the gated value supplies. The two changes thus play different roles: the gate acts before mixing and sets what content attention moves; the damping acts after mixing and sets only how far the state moves. A small change of state is therefore no evidence of what the gate did, and all value-pathway comparisons share the same damping.

Early exit. The loop need not run all N steps: once a fraction $\tau{=}0.95$ of the tokens have relative change $r_{i,k} = \|z_{i,k+1} - z_{i,k}\| / \|z_{i,k}\| < \epsilon{=}10^{-3}$, the layer stops and passes its current state on as z_ℓ. N is therefore a maximum, and we report the realized number of steps with our results. Early exit is a stopping rule, not a learned halting mechanism (Graves, 2016) or a convergence guarantee.

Scope. Proposition 2 covers the state-only endpoint only. It does not bound G-Loop: a gate that differs across tokens adds to the bound a term that grows with the state's distance from the anchor,

which is exactly the freedom the gate exists to use. What the gated loop does is therefore measured rather than proved (§3.4, Appendix F.2). On trained models the gate is a graded blend, with fewer than 9% of its entries saturated and a mean that rises from 0.49 at the first step to 0.71 at the last; the token spread stays at 78% of its initial value after 32 steps, against 4.7% for the state-only loop; and the predictions decoded from intermediate states settle before the state does. The loop is thus a fixed-budget computation whose answer stabilizes, not a solver of a fixed-point equation, which is why we train it by backpropagation through the loop rather than by implicit differentiation.

Cost accounting. Per token, a full-block step spends $12D^2$ multiply–accumulates on projections: $4D^2$ for attention and $8D^2$ for a $4\times$ FFN. A G-Loop step spends $6D^2$, the same $4D^2$ plus $2D^2$ for the gate, so taking the FFN out of the loop halves the per-step projection cost and the gate gives back a quarter of the saving. Both designs also pay about $2nD$ per token and step to form and apply the attention pattern, which dilutes the saving on long sequences, and G-Loop pays the FFN's $8D^2$ once per layer. Per layer and token this is $N(6D^2 + 2nD) + 8D^2$ against $N(14D^2 + 2nD)$ for the same model with the FFN inside the loop: $2.15\times$ at n=81 and $1.65\times$ at n=900, against measured $2.15\times$ and $1.62\times$ (§3.4; Table 9 in Appendix D). Under backpropagation through time the FFN's activations are likewise stored once per layer rather than once per step, and early exit can only lower the cost further.

3 Experiments

Our experiments ask whether attention-only recurrence is competitive with full-block recurrent models, at what cost, and whether it carries over to language (§3.2); which design choices matter (§3.3); and whether the gains come from the design or from its budget (§3.4).

3.1 Setup

Tasks. Sudoku-Extreme and Maze-Hard (Wang et al., 2025) test deep iterative reasoning from very little data: 1,000 unusually hard 9×9 puzzles for training and about 423K for testing, with about 29% of the cells *given* and the rest *blank*, and 1,000 30×30 mazes each for training and testing whose shortest path is longer than 110 steps, which the model marks cell by cell (Appendix B). We report exact-solve accuracy: a puzzle counts only if every predicted cell is correct.

Baselines. We compare with HRM (Wang et al., 2025) and with TRM (Jolicoeur-Martineau, 2025) in its self-attention (TRM-Att) and MLP-mixing (TRM-MLP) variants, all of which loop full blocks, at their reported accuracies, and report the more recent FPRM (Movahedi et al., 2026), EqR (Huang et al., 2026), and Attractor Models (Fein-Ashley & Rashidinejad, 2026) at their published accuracies; EqR and the Attractor Model were also run under our cost protocol, FPRM was not.

Protocol. G-Loop uses one configuration for both tasks (D=320, L=4, H=10, at most N=32 steps per layer; 5.7M parameters) and is trained with the deep supervision of HRM and TRM (Appendix B). All accuracies are with early exit on, under which the L=4 loops run 123.6 ± 3.9 of the $L\times N = 128$ possible steps on Sudoku-Extreme and 118.0 ± 5.2 on Maze-Hard; for cost and diagnostics we turn early exit off and report the worst case, measuring inference FLOPs, latency, and peak memory per puzzle for every model under one protocol (Appendix D). Results are averaged over three seeds.

3.2 Main Results

Reasoning. Table 1 shows that large reasoning LLMs solve none of these puzzles, while small recurrent models solve most of them. With 1.3M fewer parameters than TRM-Att, G-Loop is 2.2 points behind it on Sudoku-Extreme and 0.6 ahead on Maze-Hard, within its seed standard deviation, at $10.6\times$ and $18.7\times$ fewer FLOPs. TRM-MLP's MLP over the sequence reaches 87.4% on Sudoku-Extreme but does not scale to the 900-cell mazes, where it solves none with 19M parameters and G-Loop solves 85.9% with 5.7M. The Attractor Model, FPRM, and EqR are more accurate, and buy their accuracy with inference budgets of a different kind. Measured under our protocol, the Attractor Model's 18.9 extra points on Sudoku-Extreme cost $57\times$ G-Loop's FLOPs and its 7.2 extra points on Maze-Hard cost $101\times$; EqR's 99.8% on Sudoku-Extreme, reached with 8,192 function evaluations per puzzle, costs $4{,}900\times$. FPRM, which we could not measure because no implementation was

Table 1: **Results on Sudoku-Extreme and Maze-Hard.** Exact-solve accuracy (%) and, for the models measured under our protocol, forward GFLOP per puzzle with early exit off. Baseline accuracies are as reported; G-Loop is the mean over three seeds. EqR: training budget / with test-time scaling; its GFLOP are at the test-time-scaling budget. †As reported; not reproduced by Movahedi et al. (2026). Bold: best per column.

		Exact-solve accuracy		GFLOP per puzzle	
Model	Params	Sudoku-Extreme	Maze-Hard	Sudoku	Maze
DeepSeek-R1 / Claude 3.7 / o3-mini-high	671B / – / –	0.0	0.0	–	–
Transformer (direct prediction)	27M	0.0	0.0	–	–
HRM	27M	55.0	74.5	219.9	2,996.7
TRM-Att	7M	74.7	85.3	457.2	5,350.3
TRM-MLP	5M / 19M	87.4	0.0	410.1	7,745.5
Attractor Model†	27.3M	91.4	**93.1**	2,471.6	28,984.0
FPRM	7M	94.2	87.0	–	–
EqR	5M	86.4 / **99.8**	–	209,979.9	–
G-Loop	5.7M	72.5	85.9	**43.0**	**286.1**

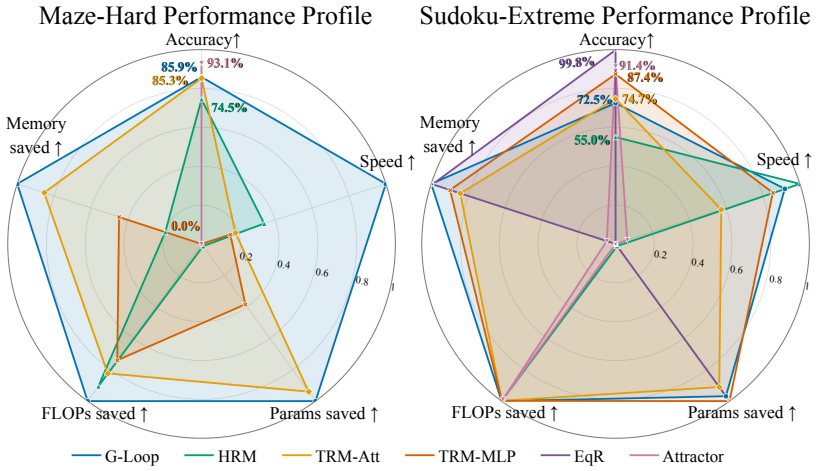


Figure 3: **Inference profile on Maze-Hard (left) and Sudoku-Extreme (right).** Higher is better: accuracy is exact-solve accuracy; on each other axis a recurrent model's score is the best recurrent model's value divided by its own, so the most efficient sits at 1; early exit off (values in Table 11).

public at the time of writing, iterates a full two-layer block up to 35,000 times per puzzle under fixed-point halting. Each G-Loop pass, by contrast, runs at most 128 attention steps, each about half the cost of a block step. On both tasks G-Loop therefore lies on the accuracy–compute frontier: no measured model is both cheaper and more accurate, and every model that is more accurate spends 9.5× to 4,900× more compute.

Cost. Figure 3 places this accuracy against every cost we measure. G-Loop needs the fewest FLOPs and the least peak memory of the recurrent models on both tasks (only EqR's peak memory on Sudoku-Extreme comes close, 2% above G-Loop's), up to 74% less memory than HRM, and it is the fastest recurrent model on Maze-Hard, 5.1× faster than TRM-Att and 64× faster than the Attractor Model. On Sudoku-Extreme its latency is 9% higher than HRM's, because its 128 steps run one after another and on an 81-cell grid each step is too cheap to hide that.

Language. To test whether G-Loop carries over to language, we replace the self-attention sublayers of the pretrained DeBERTa-v3-base encoder with G-Loop (§2.1, Appendix G) and compare it with DeBERTa-v3-base itself under identical conditions: both start from the same checkpoint, both receive 10k updates of masked language modeling on WikiText-103 (4,096 tokens per update), and both are fine-tuned on six GLUE tasks and SQuAD with one fixed recipe, five seeds, and the final checkpoint, with no hyperparameter search and no use of the development sets for selection. The

Table 2: **G-Loop in a pretrained language encoder.** Development-set results for DeBERTa-v3-base and for the same checkpoint with G-Loop in place of its attention sublayers, both given the same 10k updates of continued masked-language-model pretraining and the same fine-tuning recipe; mean over five fine-tuning seeds. CoLA: Matthews correlation; MRPC: accuracy / F1; other GLUE tasks: accuracy; all $\times 100$; Avg over the six tasks, MRPC by accuracy. SQuAD: F1 / EM. The first row is the published result for the checkpoint (He et al., 2023), whose authors report only MNLI and SQuAD 2.0 for the base model.

	GLUE							SQuAD	
Model	MNLI-m	QNLI	SST-2	CoLA	RTE	MRPC	Avg	v1.1	v2.0
DeBERTa-v3-base (published)	90.6	–	–	–	–	–	–	–	88.4 / 85.4
DeBERTa-v3-base	90.5	94.2	96.0	68.8	86.1	90.8 / 93.5	87.73	93.1 / 87.1	88.4 / 85.4
+ G-Loop	**91.2**	**94.8**	**96.6**	**71.0**	**88.5**	**91.8 / 94.2**	**88.98**	**93.8 / 87.9**	**89.3 / 86.4**

Table 3: **What the values read.** Sudoku-Extreme exact-solve accuracy (%, mean $\pm$ std over 3 seeds). Rows share routing, damping, and training and differ only in v_k; t is Welch's statistic against the row above; γ is a fixed scalar, γ_ℓ a learned scalar per layer, $\beta_\ell \in (0,1)^D$ a learned vector per layer, and β_k the G-Loop gate. Vanilla is a standard encoder with one attention step per layer and no loop.

Variant	Value v_k	Blend varies by	Exact	t
State-only	$W^V \bar{z}_k$	–	0.0 ± 0.0	–
Anchor-only	$W^V \bar{x}$	–	42.9 ± 1.8	41.3
Scalar-fixed	$W^V[(1-\gamma)\bar{x} + \gamma \bar{z}_k]$	nothing	48.8 ± 0.7	5.3
Scalar-learned	$W^V[(1-\gamma_\ell)\bar{x} + \gamma_\ell \bar{z}_k]$	layer	57.3 ± 1.1	11.3
Static-β	$W^V[(1-\beta_\ell) \odot \bar{x} + \beta_\ell \odot \bar{z}_k]$	layer, channel	60.4 ± 0.9	3.8
Linear-concat	$W^{\text{cat}}[\bar{x}; \bar{z}_k]$	layer (linear map)	65.1 ± 1.3	5.2
G-Loop	$W^V[(1-\beta_k) \odot \bar{x} + \beta_k \odot \bar{z}_k]$	token, channel, step	$\mathbf{72.5 \pm 0.8}$	8.4
Vanilla	one plain attention step	–	0.2 ± 0.1	–

loop runs at most 16 steps per layer with early exit; Appendix G lists where it departs from the puzzle sublayer. Table 2 reports the results. Our DeBERTa-v3-base run lands on the published numbers for the checkpoint (90.5 against 90.6 on MNLI-m, the same 88.4/85.4 on SQuAD 2.0), and the G-Loop encoder is ahead of it on every task, by 0.6 to 1.0 points on six of them and by 2.2 and 2.4 on CoLA and RTE, raising the six-task GLUE average by 1.25 points, from 87.73 to 88.98. Only the seed means were retained, so we do not test these differences for significance; and the comparison matches initialization, data, and recipe, not budget, since G-Loop adds its gate and a position table per layer and up to 16 attention steps at inference (Appendix G).

3.3 What the Loop Needs

What the values read. Table 3 turns the analysis of §2.2 into a controlled experiment, and three findings stand out. First, the collapse it describes is real: *State-only* solves no puzzles (0.0%). Second, the anchor is the single most important ingredient: *Anchor-only* already reaches 42.9%. Third, from there every extra degree of freedom in how anchor and state are blended helps, and what helps is adaptivity rather than capacity: *Linear-concat* is the most flexible and most parameter-heavy of the blends fixed across tokens and steps, yet G-Loop beats it by 7.4 points (72.5 against 65.1). That last step lets the blend change from token to token and step to step, which a partly solved puzzle calls for, since different cells resolve at different times and no fixed blend can follow them.

Where the anchor enters. Re-reading the input at every step is not new: input injection adds it to the state. Table 4 asks whether it matters that G-Loop feeds it into the values instead, where attention routes it between tokens. The *State* rows keep state-only values and add the anchor, projected ($W^O W^V \bar{x}$) or raw ($\bar{x}$), to the damped update, so it reaches each token directly without passing through attention (Appendix C); in every row w is a learned scalar per layer or a gate of G-Loop's form.

Table 4: **Where the anchor enters.** Sudoku-Extreme exact-solve accuracy (%). The anchor is re-read at every step, with weight w; the three gated variants have equal parameter counts.

Anchor enters	Anchor term	Routed by attention	w learned scalar	w gate
Values (G-Loop)	$(1-w)\odot\bar{x}$ in v_k	yes	57.3	**72.5**
State, projected	$w\odot W^O W^V\bar{x}$ in the update	no	55.4	67.4
State, raw	$w\odot\bar{x}$ in the update	no	54.0	65.1

Table 5: **Matched-budget controls.** Exact-solve accuracy (%). Each control matches G-Loop in parameters or compute but changes its design.

Control	Matched in	Differs from G-Loop	Params	Sudoku	Maze
G-Loop	–	–	5.7M	$\mathbf{72.5 \pm 0.8}$	85.9 ± 0.6
FFN in the loop	parameters	FFN applied at every step	5.7M	67.8 ± 0.9	86.8 ± 1.3
Wide-static-β	parameters	fixed blend, no gate	5.7M	62.8 ± 1.1	–
Untied encoder	parameters	no loop, untied layers	6.2M	0.0	0.0
Untied encoder	compute	no loop, untied layers	85.2M	0.0	0.0

The values win in both columns, and by more once the weight is adaptive: with a learned scalar, anchoring through the values (57.3%) is ahead of the projected and raw state variants (55.4% and 54.0%) by 1.9 and 3.3 points; with a gate, G-Loop (72.5%) is ahead of the same two variants (67.4% and 65.1%) by 5.1 and 7.4 points, at identical parameter count. We read this as a statement about routing: added to the state, the anchor can only restore a token's own input; inside the values it becomes content that attention routes, so each token can also read the inputs of the tokens it attends to, and the gate can choose which of them matter.

3.4 Matched Compute and Parameters

Table 5 asks whether the same budget, spent differently, could match G-Loop. It cannot. *FFN in the loop* has G-Loop's parameters but spends $2.15\times$ its compute on Sudoku and $1.62\times$ on Maze, as the count of §2.3 predicts, and buys nothing with it: it is 4.7 points worse on Sudoku (67.8% against 72.5%) and 0.9 points better on Maze (86.8% against 85.9%), a difference within noise. *Wide-static-β* is the static-β blend of Table 3, enlarged to G-Loop's parameter count, so the two differ only in whether the blend adapts per token and step; it reaches 62.8%, 9.7 points below G-Loop (t=12.3), so the gate's value lies in what it computes, not in its size. Most telling, spending the budget on depth instead of recurrence fails outright: an *untied encoder* with separate weights in every layer and no loop solves no puzzle, whether it matches G-Loop's parameters (6.2M) or its compute (85.2M). Measurements of the trained loops (Appendix F.2) confirm the mechanism: with state-only values the token spread collapses to 4.7% of its initial value after 32 steps, against 76% and 78% when the values read the anchor.

4 Conclusion

Recurrent depth does not require looping the whole block. Looping only self-attention halves the cost of each step, and the one thing that breaks, the loss of token differences in a damped attention-only loop, is repaired by gating what attention moves rather than where. With 5.7M parameters, G-Loop edges past the attention-based TRM on Maze-Hard and comes within 2.2 points of it on Sudoku-Extreme at an order of magnitude fewer FLOPs; under our cost protocol, the reasoners that score higher spend $57\times$ to $4{,}900\times$ as many. Every ablation points to the gated value, not to parameters, compute, or input injection, as the source of the gain, and G-Loop transfers to language: dropped into DeBERTa-v3-base, it fine-tunes ahead of the original on all eight tasks. Our evidence has limits (Appendix I): two grid benchmarks and one encoder, latency that grows with sequential steps on short inputs, and a theory for the state-only endpoint only. Decoder language models, conditioned on demonstrations rather than task identifiers, are the natural next test.

AI USE STATEMENT

In this work, generative AI tools were used for language editing, improving the clarity and organization of the manuscript, and assisting with the presentation of experimental results in tables and captions. Generative AI was not used to generate research ideas, design or run experiments, produce experimental measurements, determine benchmark outcomes, produce figures, or make decisions about the scientific conclusions of the work.

All mathematical formulations, model designs, experimental protocols, reported results, and scientific claims were written or verified by the authors. AI-assisted text was checked against the underlying implementation, experimental logs, and source literature, and was revised or discarded when it did not accurately reflect the work. Any AI-assisted code or analysis used during development was inspected and validated by the authors before use.

The authors take full responsibility for the final content of this work, including all text, claims, code, figures, and other artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section 2 specifies G-Loop completely: §2.3 and Figure 2 give one step, with the shared normalization, the gated value, the damped update, and the early-exit rule, and Eq. (1) gives how the loop sits in the encoder layer. The results of §2.2 are stated with their conditions along the trajectory of the loop, and Appendix A proves every claim, including the per-group damping case used by all trained models. Appendix B describes the benchmarks, the configuration (Table 6), the deep-supervision training procedure, and how early exit enters evaluation; Appendix C defines every control in Tables 3–5 exactly; Appendix D gives the analytic cost count, the measurement protocol for FLOPs, latency, and memory, and all measured values; Appendix E defines each diagnostic reported in Appendix F.2; and Appendix G does the same for the language experiments. All benchmarks are public and we use their published splits: Sudoku-Extreme and Maze-Hard as released with HRM, and the GLUE and SQuAD development sets. Every control shares all settings with G-Loop except the component named, and reasoning results report the mean and standard deviation over three seeds. We provide anonymized code, training configurations, and the scripts that generate every table as supplementary material.

REFERENCES

ARC Prize Foundation. The hidden drivers of HRM's performance on ARC-AGI. https://arcprize.org/blog/hrm-analysis, 2025. Published 15 August 2025.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron C. Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Advances in Neural Information Processing Systems*, 2025. arXiv:2507.10524.

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

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

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, 2014.

François Chollet. On the measure of intelligence, 2019. URL https://arxiv.org/abs/1911.01547.

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

Roland L. Dobrushin. Central limit theorem for nonstationary Markov chains. I. *Theory of Probability and Its Applications*, 1(1):65–80, 1956. doi: 10.1137/1101006.

Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, 2021.

Jacob Fein-Ashley and Paria Rashidinejad. Solve the loop: Attractor models for language and reasoning, 2026. URL https://arxiv.org/abs/2605.12466.

Zitian Gao, Lynx Chen, Yihao Xiao, He Xing, Ran Tao, Haoming Luo, Joey Zhou, and Bryan Dai. Universal reasoning model, 2025. URL https://arxiv.org/abs/2512.14693.

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

Borjan Geshkovski, Cyril Letrouit, Yury Polyanskiy, and Philippe Rigollet. The emergence of clusters in self-attention dynamics. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, 2023.

Alex Graves. Adaptive computation time for recurrent neural networks, 2016. URL https://arxiv.org/abs/1603.08983.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. DeBERTaV3: Improving DeBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. In *International Conference on Learning Representations*, 2023.

Byeongho Heo, Song Park, Dongyoon Han, and Sangdoo Yun. Rotary position embedding for vision transformer. In *Proceedings of the European Conference on Computer Vision (ECCV)*, volume 15068 of *Lecture Notes in Computer Science*, pp. 289–305, 2024. doi: 10.1007/978-3-031-72684-2_17.

Benhao Huang, Zhengyang Geng, and J. Zico Kolter. Equilibrium reasoners: Learning attractors enables scalable reasoning. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. arXiv:2605.21488.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *International Conference on Learning Representations*, 2026. arXiv:2602.11451.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks, 2025. URL https://arxiv.org/abs/2510.04871.

Ferdinand Kapl, Emmanouil Angelis, Kaitlin Maile, Johannes von Oswald, and Stefan Bauer. From growing to looping: A unified view of iterative computation in LLMs, 2026. URL https://arxiv.org/abs/2602.16490.

Hyunjik Kim, George Papamakarios, and Andriy Mnih. The Lipschitz constant of self-attention. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 5562–5571, 2021.

Asher Labovich. Stability and generalization in looped transformers, 2026. URL https://arxiv.org/abs/2604.15259.

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

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

W. Robert Mann. Mean value methods in iteration. *Proceedings of the American Mathematical Society*, 4(3):506–510, 1953. doi: 10.1090/S0002-9939-1953-0054846-3.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

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, 2026. URL https://arxiv.org/abs/2606.18206.

Lorenzo Noci, Sotiris Anagnostidis, Luca Biggio, Antonio Orvieto, Sidak Pal Singh, and Aurélien Lucchi. Signal propagation in transformers: Theoretical perspectives and the role of rank collapse. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2018.

Antonio Roye-Azar, Santiago Vargas-Naranjo, Dhruv Ghai, Nithin Balamurugan, and Rayan Amir. Tiny recursive models on ARC-AGI-1: Inductive biases, identity conditioning, and test-time compute, 2025. URL https://arxiv.org/abs/2512.11847.

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*, 2025.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? Generalizing from easy to hard problems with recurrent networks. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Han Shi, Jiahui Gao, Hang Xu, Xiaodan Liang, Zhenguo Li, Lingpeng Kong, Stephen M. S. Lee, and James T. Kwok. Revisiting over-smoothing in BERT from the perspective of graph. In *International Conference on Learning Representations*, 2022.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

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

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.

Xinyi Wu, Amir Ajorlou, Yifei Wang, Stefanie Jegelka, and Ali Jadbabaie. On the role of attention masks and LayerNorm in transformers. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

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

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, 2025. URL https://arxiv.org/abs/2510.25741.

Overview of the Appendix

Appendix A proves the bounds of §2.2. Appendices B–E give the setup, the exact definition of every control, how costs are counted and measured, and how the loop is diagnosed. Appendix F adds results on test-time depth and loop dynamics, Appendix G describes the language experiments, Appendix H explains why ARC-AGI is not used, and Appendix I states limitations.

A Contraction of Looped Attention

This appendix proves the two bounds used in §2.2: that a damped loop whose values are computed from the state alone shrinks the differences between tokens whenever a measurable per-step factor stays below one (Proposition 2), and that letting the values also read the anchor turns this homogeneous upper-bound recursion into an affine one that need not decay to zero (Proposition 4). Both bounds are stated along the actual trajectory of the loop, so every quantity in them can be measured on a trained model. Neither says anything about G-Loop's own dynamics.

Setting. A state is a matrix $Z \in \mathbb{R}^{n\times D}$ with one row z_i per token, and linear maps act on each row separately. We measure how different the tokens are by the *spread* $\operatorname{diam}(Z) = \max_{i,j} \|z_i - z_j\|_2$, which is zero exactly when all rows are equal. Head h has a value map $W_h^V : \mathbb{R}^D \to \mathbb{R}^{d_h}$, the block of W^V that produces its d_h value channels, and an attention matrix A_h whose entries are nonnegative and whose rows sum to one. Softmax attention has this property with or without rotary embeddings, provided dropout is off, as it is in all our measurements. Additive biases in the value and output projections cancel in differences between token rows and are omitted; the attention matrices are treated as realized, so no particular parameterization of queries and keys is assumed. For such a matrix A, let $\delta(A) = \min_{i,j} \sum_r \min(A_{ir}, A_{jr})$ be the smallest attention mass that two tokens place on the same positions; $1-\delta(A)$ is Dobrushin's contraction coefficient. For finite, unmasked softmax logits every entry of A is positive, so $\delta(A) > 0$; normalized inputs and spectrally normalized W^Q and W^K bound the logits and hence $\delta(A)$ from below, but that bound is exponentially small in the logit range and numerically uninformative, so every trajectory quantity below uses the realized overlap.

Lemma 1 (Dobrushin)**.** *For any row-stochastic A and any $Y \in \mathbb{R}^{n\times d}$, $\operatorname{diam}(AY) \le (1 - \delta(A))\operatorname{diam}(Y)$.*

Proof. Fix rows a_i, a_j and let $s = \sum_r \min(a_{ir}, a_{jr})$. The vector $a_i - a_j$ sums to zero, and its positive and negative parts each have total mass $1-s$. If $s = 1$ the rows are equal and there is nothing to prove. Otherwise $a_i - a_j = (1-s)(p-q)$ for probability vectors p and q, so $\|(AY)_i - (AY)_j\| = (1-s)\big\|\sum_r p_r y_r - \sum_s q_s y_s\big\| \le (1-s)\operatorname{diam}(Y)$, because $\sum_r p_r y_r - \sum_s q_s y_s = \sum_{r,s} p_r q_s (y_r - y_s)$ is a convex combination of differences of rows. Taking the maximum over i, j and using $s \ge \delta(A)$ gives the claim. □

The lemma is the precise form of "tokens that attend alike receive alike": averaging can never increase the spread of what it averages, and it shrinks it by at least the overlap of the two least similar rows.

Proposition 2 (Contraction of state-only values)**.** *Consider one step with state-only values, $v_k = W^V \bar{z}_k$ with $\bar{z}_k = \mathrm{LN}^{\mathrm{loop}}(z_k)$, candidate $c_k = W^O A_k v_k$, and update $z_{k+1} = (1-\alpha)z_k + \alpha c_k$ with $\alpha \in (0,1)$. Let $\Lambda_k = \operatorname{diam}(\bar{z}_k)/\operatorname{diam}(z_k)$ be the realized gain of the normalization (set to 1 when $\operatorname{diam}(z_k) = 0$), $\delta_k = \min_h \delta(A_{k,h})$, $\kappa_V = \big(\sum_h \|W_h^V\|_2^2\big)^{1/2}$, $\omega = \|W^O\|_2$, and $\mu_k = \Lambda_k \kappa_V \omega (1-\delta_k)$. Then*

$$\operatorname{diam}(z_{k+1}) \le \big(1 - \alpha(1-\mu_k)\big)\operatorname{diam}(z_k), \qquad \operatorname{diam}(z_K) \le \textstyle\prod_{k<K}\big(1-\alpha(1-\mu_k)\big)\operatorname{diam}(z_0).$$

Proof. If $\operatorname{diam}(z_k) = 0$, all rows are equal, every later state is token-uniform, and the claim holds. Otherwise, the values of head h, $W_h^V \bar{z}_{k,i}$, differ across tokens by at most $\|W_h^V\|_2 \operatorname{diam}(\bar{z}_k)$. By Lemma 1, the head's output Y_h, whose row i is $\sum_r A_{k,h,ir} W_h^V \bar{z}_{k,r}$, has $\operatorname{diam}(Y_h) \le (1 - \delta_k)\|W_h^V\|_2 \operatorname{diam}(\bar{z}_k)$. Concatenating heads, $\|y_i - y_j\|^2 = \sum_h \|y_{h,i} - y_{h,j}\|^2$, so $\operatorname{diam}(Y) \le (1 - \delta_k)\kappa_V \operatorname{diam}(\bar{z}_k) = (1 - \delta_k)\kappa_V \Lambda_k \operatorname{diam}(z_k)$, and $\operatorname{diam}(c_k) \le \omega \operatorname{diam}(Y) = \mu_k \operatorname{diam}(z_k)$. Finally, $\|z_{k+1,i} - z_{k+1,j}\| \le (1-\alpha)\|z_{k,i} - z_{k,j}\| + \alpha\|c_{k,i} - c_{k,j}\|$; taking the maximum over pairs gives the one-step bound, and induction gives the product. □

Corollary 3 (A routing-independent bound)**.** *Under the conditions of Proposition 2, for any row-stochastic attention matrices, and hence for any queries and keys that produce them,* $\operatorname{diam}(z_{k+1}) \le \big(1 - \alpha(1 - \Lambda_k \kappa_V \omega)\big) \operatorname{diam}(z_k)$.

Proof. $\delta_k \ge 0$, so $\mu_k \le \Lambda_k \kappa_V \omega$. □

What the two results say. Proposition 2 splits the per-step factor into the roles of the attention step: the normalization and the value projection may amplify differences, the averaging never does, and the output projection may amplify again. In the bound, routing enters only through the overlap δ_k, and dropping it gives the routing-independent candidate bound $\operatorname{diam}(c_k) \le \Lambda_k \kappa_V \omega \operatorname{diam}(z_k)$ from which Corollary 3 follows, so $\Lambda_k \kappa_V \omega < 1$ is a sufficient condition for contraction under any routing ($\Lambda_k \kappa_V \omega < \alpha_{\min}/\alpha_{\max}$ under the per-group damping below). This is the precise sense in which routing is non-expansive: for given values, queries and keys change which convex combination each token receives but cannot make the averaged output more spread out than the values. Three limits of the result matter. First, no Lipschitz constant is assumed: Λ_k is the realized effect of the normalization on the current state, and the bound compares tokens within one sequence at the realized attention matrices rather than two inputs of the map $Z \mapsto \mathrm{Attn}(Z)$, so no derivative of A_k enters. A global bound $\Lambda_k \le \Lambda$ would give an a priori version, but LayerNorm's local sensitivity can grow as a token's channel variance shrinks, and its global Lipschitz constant scales like the learned gain over $\sqrt{\epsilon}$, with ϵ the variance floor, far too loose to be informative (the Lipschitz constant of self-attention itself is analyzed by Kim et al. (2021)); the condition is therefore best checked with Λ_k and δ_k measured on the trajectory and with κ_V and ω from exact singular value decompositions, and κ_V can exceed one, since W^V is not spectrally normalized (Table 6). Second, the spread is a seminorm on differences between tokens: the bound constrains how alike the tokens become, not whether the trajectory converges. Third, the result covers only state-only values.

Channel-wise damping. G-Loop damps with one coefficient per group of channels, a vector $\alpha \in (0,1)^D$ applied to every row. Since $\|(1-\alpha) \odot u + \alpha \odot w\| \le (1 - \alpha_{\min})\|u\| + \alpha_{\max}\|w\|$, the same argument gives $\operatorname{diam}(z_{k+1}) \le \big(1 - \alpha_{\min} + \alpha_{\max}\mu_k\big) \operatorname{diam}(z_k)$, which reduces to Proposition 2 when all coefficients are equal; contraction then requires $\mu_k < \alpha_{\min}/\alpha_{\max}$. All loops trained on the puzzles, the state-only control included, use this damping, so this is the condition that applies to them.

Proposition 4 (Fixed anchored blend)**.** *If instead* $v_k = W^V\big[(1-\gamma)\bar{x} + \gamma \bar{z}_k\big]$ *with* $\gamma \in [0,1)$*, then* $\operatorname{diam}(z_{k+1}) \le a_k \operatorname{diam}(z_k) + b_k$*, with* $a_k = 1 - \alpha(1 - \gamma\mu_k)$ *and* $b_k = \alpha(1-\gamma)\kappa_V \omega(1-\delta_k) \operatorname{diam}(\bar{x})$*, so that* $a_k < 1$ *requires* $\gamma\mu_k < 1$*. If* $a_k \le a < 1$ *and* $b_k \le b$ *for all* k*, then* $\limsup_k \operatorname{diam}(z_k) \le b/(1-a)$.

Proof. The rows of the blend differ by at most $(1-\gamma)\operatorname{diam}(\bar{x}) + \gamma\Lambda_k \operatorname{diam}(z_k)$. As in the proof of Proposition 2, $\operatorname{diam}(c_k) \le \kappa_V \omega(1-\delta_k)\big[(1-\gamma)\operatorname{diam}(\bar{x}) + \gamma\Lambda_k \operatorname{diam}(z_k)\big]$, and the damped update gives the recursion. Unrolling $d_{k+1} \le a\, d_k + b$ yields $d_k \le a^k d_0 + b(1-a^k)/(1-a)$. □

With the per-group damping of our experiments, the same argument replaces a_k by $1 - \alpha_{\min} + \alpha_{\max}\gamma\mu_k$ and b_k by $\alpha_{\max}(1-\gamma)\kappa_V\omega(1-\delta_k)\operatorname{diam}(\bar{x})$, so the contraction condition becomes $\gamma\mu_k < \alpha_{\min}/\alpha_{\max}$. For constant factors the recursion yields $\limsup_k \operatorname{diam}(z_k) \le b/(1-a)$, that is, $(1-\gamma)\kappa_V\omega(1-\delta)\operatorname{diam}(\bar{x})/(1-\gamma\mu)$ for scalar damping and $\alpha_{\max}(1-\gamma)\kappa_V\omega(1-\delta)\operatorname{diam}(\bar{x})/(\alpha_{\min} - \alpha_{\max}\gamma\mu)$ under per-group damping: unlike the homogeneous state-only bound, this bound need not vanish. It gives no positive lower bound on the spread; it shows only that the homogeneous contraction argument no longer applies. Empirically, the anchored models keep most of their spread (Table 12).

B Experimental Setup

This appendix gives what is needed to reproduce the reasoning experiments: the tasks, the configuration, the training procedure, and how early exit enters evaluation.

Tasks. Both benchmarks follow Wang et al. (2025). A Sudoku-Extreme puzzle is a 9×9 grid flattened into n=81 tokens over the digits 0–9, where 0 marks a blank cell; about 29% of the cells are given. A Maze-Hard instance is a 30×30 grid flattened into n=900 tokens that mark walls, open cells, start, and goal, and the model labels the cells on the shortest path. A puzzle counts as solved only if every predicted cell is correct.

Configuration. Both tasks use the same model (Table 6); only the input length and the output head differ. The two parameter counts differ by 3,205 because of the task-specific embedding and head. The gate bias starts at zero and W^g at its default initialization.

Table 6: Configuration shared by G-Loop and all controls on the reasoning tasks.

Width D	320	Parameters (Sudoku / Maze)	5,757,491 / 5,754,286
Layers L	4	Spectral normalization	W^Q, W^K, W^O
Heads H, head width d_h	10, 32	Early exit τ, ϵ	0.95, 10^{-3}
Maximum steps N	32	Deep-supervision segments J	3 / 1
FFN expansion	$4\times$	Seeds	3

Training. A *segment* is one forward pass through all L layers, each running its loop for up to N steps. Within a segment we backpropagate through every step of every loop, with per-token cross-entropy on the cells to be predicted. Following the deep supervision of HRM and TRM (Wang et al., 2025; Jolicoeur-Martineau, 2025), each training sample runs for J segments; each segment starts from the detached state of the previous one and is supervised on its own output, so later segments train the model to refine a prediction it has already made.

Early exit and evaluation. All accuracies are measured with early exit on. The four loops then run on average 123.61 ± 3.85 of the 128 possible steps on Sudoku-Extreme and 117.99 ± 5.20 on Maze-Hard, so early exit shortens the computation only slightly: the loops mostly run to their maximum depth, consistent with the state still moving at the last step (Appendix F.2). Costs and diagnostics are measured with early exit off, so that every loop runs all N steps. Within a forward pass the loop applies one fixed operator: spectral-norm estimates are computed once and reused across steps, and one dropout mask is sampled per pass. All evaluations run with dropout disabled.

C Definitions of the Controls

Each control in §3 is designed to rule out one alternative explanation of G-Loop's results. Table 7 summarizes which, and the paragraphs below define each control exactly.

Table 7: What each control tests.

Control	Alternative explanation it tests	Result
Value-pathway ladder	any anchored blend would do	Table 3
Wide-static-β	the gate helps only through its parameters	Table 5
Input injection	re-reading the input matters, not where it enters	Table 4
FFN in the loop	the FFN must repeat for the loop to reason	Table 5
Untied encoders	depth or compute, not recurrence, does the work	Table 5

All variants use the layer of Eq. (1), the normalization, routing, and damped update of §2.3, the configuration of Table 6, and the same data, training, and seeds; each differs from G-Loop only in the component named below.

Value pathway. Table 8 defines the variants of Table 3. Parameters are counted relative to the state-only model; at D=320 and L=4, $D^2L = 409{,}600$.

Table 8: Value-pathway variants. γ is a fixed scalar, γ_ℓ a learned scalar per layer, and $\beta_\ell \in (0,1)^D$ a learned vector per layer shared by all tokens and steps.

Variant	Value v_k	Added parameters
State-only	$W^V \bar{z}_k$	0
Anchor-only	$W^V \bar{x}$	0
Scalar-fixed	$W^V[(1-\gamma)\bar{x} + \gamma \bar{z}_k]$	0
Scalar-learned	$W^V[(1-\gamma_\ell)\bar{x} + \gamma_\ell \bar{z}_k]$	$L = 4$
Static-β	$W^V[(1-\beta_\ell) \odot \bar{x} + \beta_\ell \odot \bar{z}_k]$	$DL = 1{,}280$
Linear-concat	$W^{\text{cat}}[\bar{x}; \bar{z}_k]$ with $W^{\text{cat}} \in \mathbb{R}^{D \times 2D}$, replacing W^V	$D^2L = 409{,}600$
G-Loop	$W^V[(1-\beta_k) \odot \bar{x} + \beta_k \odot \bar{z}_k]$	$(2D^2{+}D)L = 820{,}480$

Linear-concat combines anchor and state with an unconstrained linear map, in the style of the adapter with which Geiping et al. (2025) merge the input embedding into their recurrent state; unlike G-Loop, it has no multiplicative, input-dependent gate. *Wide-static-β* (Table 5) is static-β enlarged until it has exactly G-Loop's parameter count, so that any remaining gap cannot come from parameters. *Vanilla* applies one plain attention step per layer, with no loop, damping, or loop normalization; it differs from G-Loop in several ways at once and serves only as a reference.

Input injection. The *State* variants of Table 4 keep state-only values and add an injection term to the damped update,

$$z_{k+1} = z_k + \alpha \odot (c_k + w \odot a - z_k), \qquad a \in \{\bar{x},\ W^O W^V \bar{x}\},$$

where w is a learned scalar per layer or a gate computed exactly as G-Loop's β_k. The projected anchor reuses the layer's own W^V and W^O, so the two gated variants have exactly G-Loop's parameter count. Besides bypassing attention, these variants add the anchor on top of an ungated state contribution instead of trading the two off, so they are not a one-factor ablation of G-Loop. They answer a narrower question: whether adaptive input injection with the same gate and budget can match a gated value.

FFN in the loop. The layer's FFN moves inside the loop and is applied after every damped update, with its weights shared across steps; the value pathway is G-Loop's. Moving the FFN adds no parameters. The variant is not full-block recurrence, since the outer residual and normalization remain outside the loop.

Untied encoders. Two standard encoders with separate weights in every layer and no loop replace recurrence with depth. One has 6.2M parameters, matching G-Loop's parameter count; the other has 85.2M, matching its compute.

D Cost Measurement

The cost claims of the paper rest on two things: an analytic count that explains where the saving comes from, and measurements under one protocol. This appendix shows that the two agree.

Table 9: Per-token, per-step cost in multiply–accumulates (MACs). A linear map costs as many MACs as its weight matrix has entries.

	Weights	Full-block loop	G-Loop
Attention projections W^Q, W^K, W^V, W^O	$4 \times (D{\times}D)$	$4D^2$	$4D^2$
FFN $D \to 4D \to D$	$2 \times (D{\times}4D)$	$8D^2$	once per layer
Gate W^g	$D{\times}2D$	—	$2D^2$
Total per step		$12D^2$	$6D^2$

Counting. Table 9 counts multiply–accumulates (MACs) per token and step. Per layer and token, with n tokens and N steps, a G-Loop layer costs $N(6D^2+2nD)+8D^2$ MACs: $6D^2$ per step for the four attention projections and the gate, $2nD$ per step for forming the attention scores and applying them, and $8D^2$ for the FFN, applied once. The FFN-in-loop control keeps the gate and applies the FFN at every step, so it costs $N(14D^2 + 2nD)$. A forward pass multiplies this by n tokens and L layers, and FLOPs count two per MAC.

Prediction against measurement. Table 10 compares this count with the measured FLOPs. On Sudoku-Extreme the measurements cover all J=3 deep-supervision segments, so the prediction does too. The count matches the measurement to within about 2% for both models and both tasks, including the predicted ratio between them. The saving is therefore structural, not an artifact of implementation: it is exactly what removing the FFN from the loop should give.

Table 10: Forward GFLOP per puzzle at N=32: predicted from the MAC count (D=320, L=4) and measured.

	Sudoku-Extreme (n=81)		Maze-Hard (n=900)	
Model	Predicted	Measured	Predicted	Measured
G-Loop	43.04	43.04	280.2	286.1
FFN in the loop	92.4	92.6	463.0	463.0
Ratio	2.15×	2.15×	1.65×	1.62×

Measured costs. Table 11 lists the values behind Figure 3: forward FLOPs, latency, and peak memory per puzzle for all models under one protocol. G-Loop runs with early exit off, so its costs are worst-case. Vanilla, a standard encoder of our size with one attention step per layer, marks the cost of no recurrence at all.

Table 11: Inference cost per puzzle, with G-Loop at N=32 and early exit off, and the Attractor Model and EqR at the inference settings of their reported accuracies (EqR at its test-time-scaling budget). Accuracies of the baselines are as reported.

Model	Params (M)	Exact (%)	GFLOP	Latency (ms)	Peak memory (GiB)
Maze-Hard					
Vanilla	4.9	0.0	18.900	8.80	0.0610
G-Loop	5.7	85.9	286.068	157.43	0.0641
HRM	27.0	74.5	2,996.650	449.90	0.1570
TRM-Att	7.0	85.3	5,350.290	800.80	0.0810
TRM-MLP	19.0	0.0	7,745.530	930.10	0.1280
Attractor Model	27.3	93.1	28,984.041	10,109.69	0.1798
Sudoku-Extreme					
Vanilla	4.9	0.2	2.490	25.20	0.0290
G-Loop	5.7	72.5	43.040	460.15	0.0318
HRM	27.0	55.0	219.920	422.60	0.1220
TRM-Att	7.0	74.7	457.240	736.50	0.0460
TRM-MLP	5.0	87.4	410.120	495.10	0.0410
Attractor Model	27.3	91.4	2,471.610	6,556.90	0.1177
EqR	5.0	99.8	209,979.852	427,971.97	0.0324

Three patterns in these numbers are worth making explicit. First, memory: G-Loop's peak inference memory is within 10% of vanilla's on both tasks (0.0318 against 0.0290 GiB, and 0.0641 against 0.0610 GiB), because at inference the loop reuses one state buffer for all of its steps; recurrence adds depth at almost no memory cost. Second, latency does not follow FLOPs. G-Loop uses 5.1× fewer FLOPs than HRM on Sudoku-Extreme but has 9% higher latency, because its $L{\times}N = 128$ steps run one after another and on an 81-token grid each step is too small to keep the hardware busy. On the 900-token mazes, where each step is larger, G-Loop is also the fastest recurrent model, 2.9× faster than HRM, 5.1× faster than TRM-Att, and 64× faster than the Attractor Model. Latency on short inputs is a genuine cost of sequential recurrence: batching many puzzles recovers throughput,

but not the latency of a single one. Third, accuracy above G-Loop's is bought with compute. The Attractor Model's 18.9 extra points on Sudoku-Extreme and 7.2 on Maze-Hard cost $57\times$ and $101\times$ the FLOPs and $14\times$ and $64\times$ the latency; EqR's 27.3 extra points on Sudoku-Extreme cost $4{,}900\times$ the FLOPs and 428 seconds per puzzle against G-Loop's 0.46, although, as a 5M-parameter model that also reuses its buffers, it matches G-Loop's memory (0.0324 against 0.0318 GiB).

The more recent models. FPRM (Movahedi et al., 2026) could not be run under this protocol because no implementation was public at the time of writing, so we record its published evaluation budget instead: it iterates a two-layer block until a fixed-point residual falls below 0.1, with a cap of 35,000 iterations per puzzle at evaluation on both Sudoku-Extreme and Maze-Hard; its authors note that the step-size decay that gives its best accuracy also exhausts this budget. EqR (Huang et al., 2026) and the Attractor Model (Fein-Ashley & Rashidinejad, 2026) were run under the protocol, with their accuracies taken as reported. EqR reports its Sudoku-Extreme results at a training budget of $D{=}16$ outer iterations, each of which applies its two-layer block 21 times, that is, 42 layer applications per outer iteration and 672 in all; its 99.8% uses $D{=}64$ and $B{=}128$ restarts, 8,192 function evaluations per puzzle, and this is the budget whose cost Table 11 lists. The Attractor Model solves the loop with Anderson acceleration; Movahedi et al. (2026) could not reproduce its accuracies, which we therefore report as published, but its cost is measured like every other model's. For comparison, G-Loop runs $J{=}3$ segments of at most $L{\times}N = 128$ attention steps on Sudoku-Extreme, each step costing $6D^2 + 2nD$ MACs per token rather than a full block, and one segment on Maze-Hard.

E Diagnostics

These measurements test the mechanism behind the results rather than the results themselves: whether token differences collapse as §2.2 analyzes, and whether the gate behaves as §2.3 intends. All run on trained Sudoku-Extreme models with dropout disabled and early exit off, so that every loop runs all N steps.

Spread. We record the spread $\mathrm{diam}(z_{\ell,k})$, the largest distance between two token states, which is the quantity Proposition 2 bounds, and report it relative to its value at $k{=}0$.

Gate statistics. We record the distribution of β over tokens, channels, and steps; the fraction of entries within 0.05 of 0 or 1 (saturation); the mean of β at each step; and the correlation between a cell's mean β and whether it is blank in the input.

Relative state change and readout. The relative state change is $r_k = \|z_{k+1} - z_k\|_F / \|z_k\|_F$. To read out an intermediate step, we replace the loop's output by the intermediate state $z_{\ell,k}$ and apply the rest of the network unchanged.

Blank-cell accuracy. Exact-solve accuracy cannot distinguish a model that solves nothing from one that fills most cells correctly, so we also measure accuracy per cell. A fraction g of Sudoku cells is given, and a model gets those right by copying them. With cell accuracy c, accuracy on the blank cells is $b = (c - g)/(1 - g)$. A model that uses no information from other cells can do no better than guess among nine digits on a blank cell, so $b = 1/9 \approx 11.1\%$ is the floor for any cross-cell reasoning.

F Additional Results

F.1 Test-Time Depth

Because the damped update keeps the state on a fixed scale, the number of steps can be changed at test time. G-Loop is trained with $N{=}32$ steps per layer. Run with $N{=}16$ instead, it solves 56.3% of Sudoku-Extreme puzzles; with $N{=}64$, 71.8%, against 72.5% at the training depth. On Maze-Hard, accuracy is 85.9% at both $N{=}32$ and $N{=}64$. Running longer than in training thus costs at most 0.7 points, while halving the depth costs 16.2 points on Sudoku.

F.2 Loop Dynamics

Table 12: Loop diagnostics on Sudoku-Extreme (Appendix E). Spread after the last of N=32 steps, relative to its value at k=0; cell and blank-cell accuracy with $g = 0.291$ given cells. Dashes mark quantities not measured for that model.

Values read	Relative spread at k=32	Cell accuracy (%)	Blank-cell accuracy (%)
State-only (Table 3)	0.047	50.09	29.6
Anchor-only	0.761	–	–
G-Loop	0.784	87.94	83.0
Vanilla (one attention step)	–	74.32	63.8

Table 12 checks the analysis of §2.2 on the trained models, and three observations follow.

The collapse happens, and the anchor prevents it. With state-only values, the spread falls to 4.7% of its initial value after 32 steps. When the values also read the anchor, it stays at 76% (anchor-only) and 78% (G-Loop), consistent with Proposition 4.

Collapse destroys information rather than merely failing to add it. The state-only model is right on 29.6% of blank cells, above the 11.1% floor, so some information still crosses cells. But a single plain attention step (vanilla) is right on 63.8% of them: looping with state-only values leaves the model worse than not looping at all. G-Loop reaches 83.0%, and turns this into 72.5% exact solves.

The gate is a graded, input-dependent blend. Fewer than 9% of its entries lie within 0.05 of 0 or 1, so it neither switches the anchor on and off nor collapses to one of the endpoints of the ladder. Its mean rises from 0.49 at the first step to 0.71 at the last, so the values draw increasingly on the state. And it is higher on blank cells than on given ones (correlation 0.28): the cells that still have to be solved are the ones that rely most on what the loop has computed.

Finally, the loop does not reach a fixed point, and does not need to. With early exit off, the relative state change at the last step is still $r = 0.024$, yet the predictions decoded from intermediate states settle before the state does. G-Loop is best understood as a fixed-budget computation whose answer stabilizes, not as a solver of a fixed-point equation, which is also why we train it by backpropagation through the loop rather than by implicit differentiation.

G Language Experiments

The language experiments ask whether G-Loop, developed on grid puzzles, carries over to a language encoder. Rather than train an encoder from scratch at a budget far below what language understanding needs, we convert a pretrained one: both arms start from the public DeBERTa-v3-base checkpoint (He et al., 2023), one keeps its attention sublayers and the other replaces them with G-Loop, and both receive the same continued pretraining and fine-tuning. The published results of DeBERTa-v3-base are listed only to show that our recipe recovers them.

Architecture. Both arms are the 12-layer DeBERTa-v3-base encoder (D=768, 12 heads) with all weights taken from the checkpoint. In the G-Loop arm, the attention sublayer of layers 0–10 is replaced by the loop of §2.3, which reuses the layer's own W^Q, W^K, W^V, W^O, and LayerNorm; the last layer is left standard because the enhanced mask decoder reuses it with decoupled queries. The new parameters are the gate (W^g, b^g) and a relative-position table (512×768) in each looped layer, about 13.0M and 4.3M, roughly 20% of the 86M backbone, all with the framework's default initialization. The loop departs from the puzzle sublayer in four ways. (i) One step is

$$f(z, x) = \mathrm{LN}\big(x + W^O A_k v_k\big), \qquad z_{k+1} = (1-\alpha) z_k + \alpha f(z_k, x), \qquad z_0 = x,$$

with the anchor x the layer input held fixed, v_k the gated value of §2.3, and A_k computed from queries read from the state and keys read from the anchor; the residual adds the anchor and the sublayer's post-LayerNorm sits inside f, so there is no separate loop normalization. (ii) The damping is one fixed scalar, $\alpha = 0.8$, rather than a learned coefficient per group. (iii) Of the disentangled attention terms, content-to-content and content-to-position are recomputed at every step from the

state, while position-to-content is fixed by the anchor; relative-position buckets and embeddings do not depend on the state. (iv) Gradients do not pass through the loop: it is run without gradient tracking and a one-step phantom gradient is taken at the final state, in place of backpropagation through time. Spectral normalization is applied to W^Q, W^K, W^V, and W^O with one power iteration per forward pass, and not to the relative-position projections. With zero iterations, $\alpha = 1$, and spectral normalization off, the looped layer is exactly one standard DeBERTa attention application, which unit tests verify at the layer and backbone level. The loop runs at most $N{=}16$ steps per layer; early exit stops it once 95% of the tokens satisfy $\|z_{k+1} - z_k\| \leq 10^{-4}\|z_k\|$. The mean number of realized steps was not recorded.

Continued pretraining. Both arms use masked language modeling with a 15% masking rate (no replaced token detection) on the WikiText-103 training set, sequence length 128, 32 sequences (4,096 tokens) per update, and 10k updates, about 41M tokens in all; AdamW with peak learning rate 10^{-4}, 1k warmup updates, cosine decay to 10^{-6}, weight decay 0.01, and gradient clipping at 1.0; fp32 on 8×H100. Data order was shuffled without a fixed seed. Loss curves for both arms are logged per update and archived.

Fine-tuning and evaluation. CoLA, SST-2, MRPC, RTE, QNLI, MNLI (matched), SQuAD 1.1, and SQuAD 2.0 are run with one recipe for both arms: batch size 32, 100 epochs, up to 50k training examples per task, maximum sequence length 128 for GLUE and 384 for SQuAD, no hyperparameter search, five seeds, and the final epoch's checkpoint. SQuAD 2.0 uses a null-score threshold for answerability, and EM/F1 follow the official metric definitions in our own implementation. Evaluation is on the development sets, which are never used for training or for selection, and the same evaluation code serves both arms. Table 2 reports the mean over the five seeds; the per-seed values were not retained, so no standard deviations are given. QQP, STS-B, and MNLI-mismatched were not run.

Not done. The pretraining corpus was not checked for overlap with the development sets; both arms see the same corpus, and the checkpoint itself was pretrained on Wikipedia, so any overlap affects the two arms alike. The inference cost that the loop adds to the encoder was not measured, and the reported runs have not been reproduced from a clean checkout.

H Why We Do Not Evaluate on ARC-AGI

Each ARC-AGI task (Chollet, 2019) asks for a new transformation to be induced from a few demonstration pairs, so accuracy measures few-shot rule induction as much as iterative computation. The ARC results of current recurrent reasoners also depend on components outside the recurrence. For HRM, the ARC Prize analysis attributes much of the score to data augmentation and the outer refinement loop rather than to the hierarchical architecture (ARC Prize Foundation, 2025). For TRM, replacing the learned per-task identifier embedding with an uninformative one drops a released checkpoint to zero accuracy, and most of its accuracy is already reached at the first recursion step (Roye-Azar et al., 2025). An ARC comparison would therefore mostly measure these components, not the design of the recurrent operator. Conditioning a recurrent model on demonstrations instead of task identifiers is an important but separate problem.

I Limitations

Scope of the evidence. Our reasoning evaluation covers two grid benchmarks with fixed rules. Our language evaluation is a matched comparison between DeBERTa-v3-base and the same checkpoint with G-Loop in place of its attention, both after a small continued-pretraining budget (10k updates of 4,096 tokens); it controls initialization, data, and recipe but not parameters or inference compute, which the G-Loop arm spends more of, and only seed means were retained, so the differences are not tested for significance. The language loop uses a fixed damping and a one-step phantom gradient rather than backpropagation through the loop, so it is not the puzzle model unchanged; QQP and STS-B were not run, and the loop's inference cost in the encoder and its realized number of steps were not recorded. We do not test decoder language models, encoders trained from scratch, or tasks whose rules must be induced from examples (Appendix H).

Accuracy. G-Loop is not the most accurate recurrent reasoner on either benchmark (Table 1). Of the more recent models with higher accuracy, the Attractor Model and EqR were run under our cost protocol with their accuracies taken as reported, and FPRM could not be, as no implementation was public at the time of writing, so the comparison with FPRM rests on its published evaluation budget (Appendix D) rather than on FLOPs measured side by side. Running G-Loop for more steps than in training does not raise its accuracy (Appendix F.1), and we have not tried restarts or other forms of test-time scaling.

Cost. The loop runs its steps sequentially, so latency grows with the number of steps even when FLOPs are low, and on short inputs G-Loop is not the fastest model (Appendix D). Training by backpropagation through all steps stores the attention activations of every step, so training memory grows with N.

Theory and controls. Our analysis covers only the state-only endpoint of a damped loop; we prove nothing about G-Loop's own dynamics and rely on measurement. The gated input-injection comparison is not a one-factor ablation, and we have not compared against an additive residual inside the loop. The gate adds 14.3% parameters at our scale, which the capacity control addresses for the value-pathway comparison but not for every baseline.