RETHINKING AT FIXED POINTS IN LOOPED LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

A looped language model whose recurrent states converge to fixed points can replace its recurrent trajectory with the terminal state, in both training and inference. Near a fixed point, the endpoint can stand in for the whole trajectory and unlock a series of abilities: truncated backpropagation in training, terminal key-value cache reuse that shrinks the cache $5\times$ for a loop-5 recurrent core at almost no performance loss, distillation for up to $1.66\times$ faster prefill, and $1.5\times$ faster post-training updates that compute gradients from saved rollout states instead of replaying the trajectory. To better shape fixed points, we improve two existing techniques: depth sampling, whose depth prior we propose to learn from training signals with constraints; and input injection, which we force orthogonal update so that every recurrence receives the input intact. At 100M, 400M, and 1.6B parameters, the learned depth prior lowers perplexity under terminal cache reuse by 1.1–1.9% relative to Huginn's fixed prior; at 1.6B, it matches the full-cache downstream accuracy of fixed-depth training. Orthogonal injection lowers perplexity by up to 1.6% relative to leading injection schemes.

1 INTRODUCTION

Looped language models repeatedly apply a shared block of layers (Dehghani et al., 2019; Geiping et al., 2025), but whether they should learn fixed points remains contested. One camp forgoes fixed points and keeps the loop count small and fixed (Gao et al., 2026; Wang et al., 2026; Zhu et al., 2025). The other camp shapes fixed points (Bai et al., 2021; Yang et al., 2026; Huang et al., 2026) and loops much deeper. But in typical implementations, every loop adds cost: activations to store during training, keys and values (KV) to cache during decoding, compute to prefill each prompt, and a replay to score each rollout during reinforcement learning. In this work, we show that fixed points also make deep looped models cheaper to pretrain, serve, and post-train.

The fixed-point equation $h^\star = f(h^\star, x)$ mentions only the endpoint, never the path to it, so near a fixed point the endpoint can stand in for the full trajectory (Section 3). Reusing terminal KV caches across loops tests this view: it barely changes the zero-shot GSM8K accuracy of Huginn (Geiping et al., 2025) but collapses that of Ouro (Zhu et al., 2025) at four loops (Schwethelm et al., 2026). Huginn's states approach fixed points (Blayney et al., 2026); Ouro's do not.

Similarly, each cost above has a matching shortcut. In pretraining, backpropagating through only the last few loops approximates the equilibrium gradient (Liao et al., 2018; Geng et al., 2021) without the warm-up that implicit gradients require. In decoding, *terminal KV sharing* keeps only each prefix token's terminal KV (Figure 2b), with little performance loss. In prefilling, a distilled student can learn to predict the endpoint directly and replace the loops (Figure 2c). In reinforcement learning, scoring rollouts from saved fixed-point states accelerates the replay. Each shortcut removes a depth-dependent cost, so its savings grow with every additional loop.

Realizing these benefits requires training that shapes the fixed points. Two components do the shaping, and current choices for each fall short (Section 4). The first is the *depth prior*, the distribution of training depths. Because tokens converge at different depths (Figure 1), the sampled depth decides whether training sees the converged prefix context that terminal KV sharing supplies (Figure 2a). Training at a single depth predicts best at that depth but breaks KV sharing; Huginn's fixed broad prior (Geiping et al., 2025) supports sharing but predicts worse at the target depth. We learn the prior

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.

from prediction feedback and add an entropy term that keeps it broad enough for sharing. The second component is input injection (Bai et al., 2019a; Geiping et al., 2025; Prairie et al., 2026), which adds the input to the state at every recurrence. In Huginn and Parcae, the carried-over state already contains an input component, which can amplify or cancel the injected input. Our orthogonal input injection (OrthoInj) removes this component from the state, so every recurrence receives the input at same strength.

- Terminal KV sharing (Section 3) cuts the cache $3\times$ at five recurrences almost losslessly, distilled prefill is up to $1.66\times$ faster on 8K prompts, and rollout-state reuse speeds RL updates by $1.5\times$.
- From 100M to 1.6B parameters, our learned prior (Section 4.1) gives 1.1–1.9% lower shared-KV perplexity than Huginn's fixed prior at equal mean depth, for at most 1.6% more training FLOPs.
- We propose Orthogonal Injection (Section 4.2), which also drops prelude normalization and gives 0.7–1.6% lower perplexity than the strongest baseline, Parcae's injection, at 100M and 400M.

2 Fixed Points in Looped Language Models

Recurrent states in looped language models tend to approach fixed points as recurrence depth grows (Geiping et al., 2025; Blayney et al., 2026).[1] For an input $x = (x_1, \ldots, x_N)$, the state after r recurrent updates is $z^r = (\mathbf{H}^r, C^r)$: hidden states $\mathbf{H}^r = (h_1^r, \ldots, h_N^r)$ and the *KV banks* C^r written on update r, one per physical attention layer. The shared update map gives $z^{r+1} = F_\theta(z^r; x)$, and $z^\star$ is a *fixed point* if $F_\theta(z^\star; x) = z^\star$. We use Huginn (Geiping et al., 2025) for experiments. A prelude of non-shared blocks embeds the input as $e = P_\theta(x)$. A recurrent core of shared blocks starts from a random state z^0 and iterates $z^r = F_\theta(z^{r-1}; x)$ for $r = 1, \ldots, R$, injecting e at every recurrence (Section 4.2); we write $z_\theta^R(x)$ for the final state. A coda of non-shared blocks maps $z_\theta^R(x)$ to next-token logits.

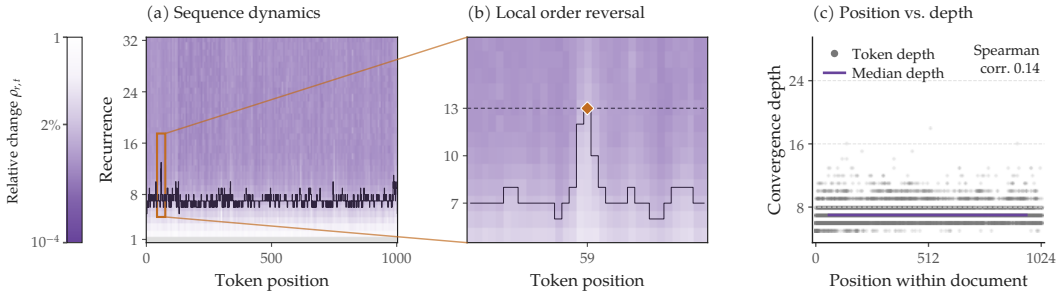


Figure 1: **Heterogeneous token convergence in Huginn.** (a) Relative hidden-state changes $\rho_{r,t}$, with convergence threshold $\tau = 2\%$. (b) Later tokens can approach approximate fixed points before earlier ones. (c) Position versus convergence depth; the purple line shows medians over 128-position intervals. Details in Appendix C.4.

We measure convergence by the tokenwise relative change $\rho_{r,t} = \left\|h_t^r - h_t^{r-1}\right\|_2 / (\left\|h_t^{r-1}\right\|_2 + \epsilon)$, with $\epsilon = 10^{-8}$. Token t is *converged from depth* r if $\rho_{r',t} < \tau$ for all $r \le r' \le R_{\text{obs}}$, the observation horizon; its *convergence depth* is the smallest such r.

Convergence in Huginn is *heterogeneous*: tokens in a sequence converge at different depths, not necessarily in causal order (Figure 1b), and convergence depth correlates only weakly with position (Spearman correlation 0.14; Figure 1c). A prefix $x_{<t}$ is a *converged prefix* from depth r if every token in it is converged from r; Section 3.2 relates such prefixes to KV sharing.

3 What Fixed-Point Shaping Enables

Fixed-point shaping trains a looped model with supervision near the fixed points of its recurrence (Bai et al., 2019a; Huang et al., 2026). At a fixed point, the endpoint can stand in for the

[1] "Fixed points" here include approximately stationary states at finite depths.

trajectory. Conventional looped language models pay for every recurrence in both training and inference; training with fixed-point shaping decouples both costs from depth.

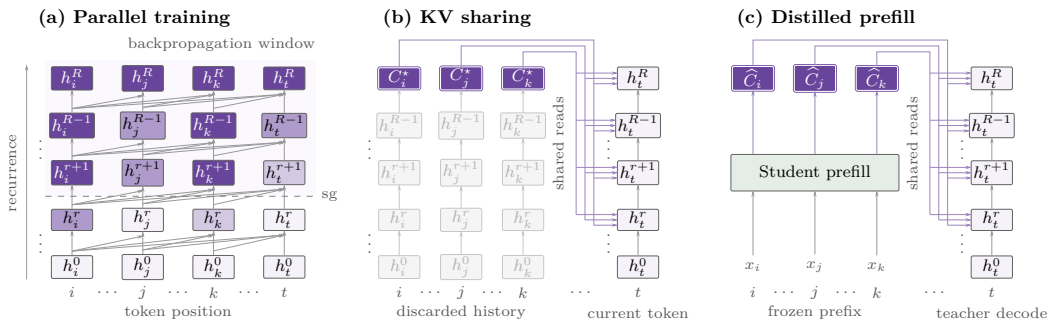


Figure 2: **The endpoint stands in for the recurrent trajectory.** Recurrence runs upward; ellipses mark omitted positions and depths. (a) In training, each update reads prefix states from the previous depth ; darker states are closer to convergence (as in Figure 1). Prefix tokens converge at different depths (Section 2), so the sampled depth R decides whether reads inside the backpropagation window see converged context; gradients stop at sg. (b) Decoding reads only the terminal prefix KV banks $C^{\star}_{<t}$ and discards the others, with nearly unchanged downstream accuracy.(c) During prefill, a distilled student replaces the recurrence and supplies approximate terminal banks $\widehat{C}_{<t}$.

3.1 Training: Endpoint Gradients

Full backpropagation through time (BPTT) differentiates through all R recurrences, so its cost grows with depth. When near a fixed point, the gradient can instead be computed from the endpoint.

Pretraining: truncated backpropagation. At a fixed point $z^{\star}$, let $J_{\star} = \partial_z F_\theta(z^{\star}; x)$ and $B_{\star} = \partial_\theta F_\theta(z^{\star}; x)$. If F_θ is continuously differentiable and the spectral radius of $J_{\star}$ is below one, implicit differentiation gives the *equilibrium sensitivity* (Bai et al., 2019a) $D_{\star} \equiv \mathrm{d}z^{\star}/\mathrm{d}\theta$, and truncating its Neumann series gives D_b (Liao et al., 2018; Geng et al., 2021)(derivation in Appendix A.2):

$$D_{\star} = (I - J_{\star})^{-1} B_{\star} = \sum_{k=0}^{\infty} J_{\star}^{k} B_{\star}, \quad D_b = \sum_{k=0}^{b-1} J_{\star}^{k} B_{\star}, \quad \tilde{D}_b = \sum_{k=0}^{b-1} \Big(\prod_{j=1}^{k} J_{R-j} \Big) B_{R-1-k}. \tag{1}$$

$D_{\star}$ depends only on $z^{\star}$, and it equals the $R \to \infty$ limit of full BPTT when that trajectory converges. Its truncated Neumann estimate D_b has a bias $D_{\star} - D_b = J_{\star}^{b} D_{\star}$ that decays geometrically in b. Both, however, are valid only once a stable fixed point exists. Applied from initialization, such endpoint gradients destabilize training (Appendix C.3), so implicit models first warm up with unrolled backpropagation (Bai et al., 2019a; 2020; Nguyen & Mauch, 2023).

Truncated BPTT (TBPTT) instead differentiates through only the last b recurrences (Williams & Peng, 1990; Aicher et al., 2019), so its memory and compute scale with b rather than R. Its gradient $\tilde{D}_b$ evaluates the Jacobians $J_r = \partial_z F_\theta(z^r; x)$ and $B_r = \partial_\theta F_\theta(z^r; x)$ at visited states z^r rather than at $z^{\star}$. This frees TBPTT from needing a fixed point: from any state, $\tilde{D}_b$ is the exact gradient of the loss through the last b recurrences with the earlier state held fixed, so every update descends a well-defined objective. As the visited states converge to $z^{\star}$, $\tilde{D}_b$ approaches D_b regardless of R. TBPTT is therefore as stable as full BPTT early in training and recovers the fixed-point gradient once fixed points form, using a single estimator with no warm-up phase or switch[2].

Post-training: reusing rollout states for RL. RL usually runs the recurrent forward pass twice: once without gradients to sample rollouts, and again with gradients to score them. Once pretraining has formed fixed points, equation (1) gives a local gradient from the endpoint alone, so the second pass need not replay the trajectory. We therefore keep the detached rollout states, score the sampled tokens from them, and backpropagate through a single differentiable recurrence at the endpoint with

[2]We additionally find that TBPTT **alone**, can already shapes fixed points without depth sampling; With sampled training depths, TBPTT performs on par with full BPTT. Details in Appendix C.3.

a truncated Neumann series. With Dr. GRPO (Liu et al., 2025) on GSM8K, reuse speeds up scoring and backpropgation by $1.5\times$ and stays within 1.6 points of full BPTT on pass@1. Although rollout generation still dominates RL training time, the update saving should grow with depth: recomputation and full BPTT replay every recurrence in the update, whereas reuse applies the recurrent block once. Appendix C.7 gives the protocol and full results.

3.2 Inference: Endpoint Caching and Distillation

Decoding: cache only terminal KV banks. The KV cache of conventional looped models grows linearly with recurrence depth. *Terminal KV sharing* removes this growth: each prefix token keeps only its terminal KV, and every recurrence of later tokens attends to it (Figure 2b).[3] This requires a model that predicts well from terminal context.

Training does not match this setting exactly. It updates all tokens of a sequence in parallel, so recurrence r of the current token reads prefix banks from depth r (Figure 2a); sharing instead supplies terminal banks from the first recurrence. This gap is small when the prefix banks change little across depth and the current token's update contracts, so an error introduced by early context decays over later recurrences. Sampled training depths push the model toward this regime: deep draws let supervised recurrences read nearly settled prefix banks, and supervising many depths penalizes states that drift as recurrence continues (Figure 3).

Lemma 1 formalizes the limiting case, in which the prefix banks converge. For the current token t, let h^r denote its state at recurrence r, $C^r_{<t}$ the prefix-dependent input to its update (including the prefix's per-layer KV banks), and $f_\theta(h; C)$ the hidden-state component of F_θ, as a function of state h and context C. By causality, $\{C^r_{<t}\}$ does not depend on the current token's states, so the prefix enters the current token's iteration as an external input sequence.

Lemma 1 (Joint iteration and frozen-prefix decoding, proof in Appendix A.1)**.** *Fix the parameters and the input sequence, and suppose $C^r_{<t} \to C^\star_{<t}$. Let $\mathcal{H} \subseteq \mathbb{R}^d$ be nonempty and closed, and let $\mathcal{C} = \{C^r_{<t}\}_{r\geq 0} \cup \{C^\star_{<t}\}$. Suppose that for every $C \in \mathcal{C}$, $f_\theta(\cdot; C)$ maps $\mathcal{H}$ into itself and is κ-contractive with $\kappa < 1$, and that for every $h \in \mathcal{H}$, $f_\theta(h; \cdot)$ is β-Lipschitz on $\mathcal{C}$. Then, from any initial states in $\mathcal{H}$, the joint and frozen-prefix iterations*

$$h^{r+1} = f_\theta(h^r; C^r_{<t}), \qquad \widetilde{h}^{r+1} = f_\theta(\widetilde{h}^r; C^\star_{<t}) \tag{2}$$

both converge to the unique fixed point $h^\star = f_\theta(h^\star; C^\star_{<t})$ in $\mathcal{H}$. Consequently, any readout continuous in state and context yields the same limiting next-token distribution under both iterations.

In the limit, therefore, sequence-parallel supervision near fixed points transfers to decoding with terminal KV sharing. Table 2 measures the sharing performance in trained models.

Prefill: distill the recurrent endpoint. Because decoding reads only terminal KV banks, prefill can be distilled into a student that predicts only the recurrent endpoint; without terminal sharing, the student would have to reproduce the KV banks of every recurrence (Horton et al., 2024). The student is a lightweight network S_ψ that replaces the iterated recurrent core during prefill; the prelude and coda are reused from the teacher. From the prelude output $e = P_\theta(x)$, the student predicts $\widehat{z} = S_\psi(e) \approx z^R_\theta(x)$; prefill then applies one teacher recurrence and the coda to $\widehat{z}$ and retains each physical attention layer's KV bank (Figure 2c). Because fixed-point shaping makes the endpoint an approximate fixed point, one recurrence from a nearby $\widehat{z}$ yields banks close to the teacher's terminal banks. Decoding is unchanged: the teacher runs at its original depth with terminal KV sharing.

We train the student against depth-R targets computed online from the frozen teacher, using normalized hidden-state regression (Romero et al., 2015) plus a next-token KL term:

$$\mathcal{L}_{\text{distill}}(\psi) = \mathbb{E}_x \left[\frac{\left\| \widehat{z} - z^R_\theta(x) \right\|_F^2}{N d\, \sigma_{\text{T}}^2} + \frac{1}{N} \sum_{t=1}^{N} \mathrm{KL}\big(p_t \,\|\, \widehat{p}_t\big) \right], \tag{3}$$

where N is the prompt length, d is the hidden width, and σ^2_{T} is a fixed teacher-state energy calibrated on training data, so the first term is a per-coordinate squared error on the teacher's scale. The second term compares the teacher's next-token distribution p_t at depth R with the distribution $\widehat{p}_t$ obtained by

[3] Decoding reads only terminal banks, so prefill and decoding can run at separate recurrence budgets (Appendix C.5); prefill gains saturate at moderate depth while decoding gains persist deeper.

applying the frozen coda and output head to $\widehat{z}$. We initialize the student from the teacher's recurrent core, so before training it approximately outputs the teacher's state after one recurrence; equation (3) then moves this output toward the depth-R endpoint (Appendix C.6.1).

Section 5.4 evaluates distilled prefill. With a suitable loss, the student may also be able to serve as a standalone language model or a drafter for speculative decoding; we leave these uses, and possible connections between recurrent state refinement and flow-based language modeling, to future work.

4 Improving Fixed-Point Shaping

The benefits in Section 3 depend on how training shapes fixed points. There are two common techniques do this shaping: depth sampling sets where supervision lands, and input injection sets what a fixed point retains about the input. We improve both, with a learned depth prior (Section 4.1) and orthogonal input injection (Section 4.2).

4.1 Learning a Depth Prior for Fixed-Point Training

The sampled training depth sets how converged the states are when supervision arrives, and thus how well the model tolerates terminal KV sharing (Section 3.2). Huginn (Geiping et al., 2025) samples the training depth from a shifted Poisson–log-normal (PLN) distribution:

$$\xi \sim \mathcal{N}(\mu, \sigma_\xi^2), \qquad K \mid \xi \sim \mathrm{Poisson}\big(\exp(\xi)\big), \qquad R = K + 1, \tag{4}$$

where $\sigma_\xi = \frac{1}{2}$ and $\mu = \log(\bar{R}-1) - \sigma_\xi^2/2$, so that $\mathbb{E}[R]$ equals the target mean $\bar{R} > 1$. Under a fixed training budget, fixed-depth training and PLN sampling trade off (Figure 3). Fixed-depth training concentrates supervision at one depth and predicts better there. PLN sampling spreads supervision across depths and yields the fixed-point behavior that supports KV sharing and test-time scaling.[4]

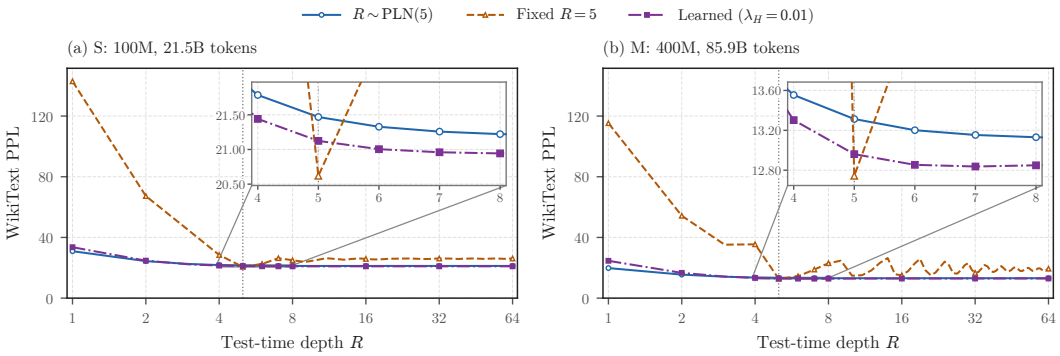


Figure 3: **The learned prior narrows the trade-off between fixed-depth training and PLN sampling.** Native WikiText-103 test PPL across test-time depths at matched training budgets. Fixed-depth training is best at training-matched depth $R = 5$ but degrades sharply at others. PLN sampling performs well across all depths but falls behind at $R = 5$. The learned prior lies between them, narrowing the gap at $R = 5$ while staying stable across depths.

We therefore *learn the depth prior*. The learned prior keeps supervision spread across depths but adapts its allocation from training feedback, balancing prediction quality against fixed-point shaping at the same mean depth.

Learned prior. We replace the PLN prior with a categorical prior $p_\phi = \mathrm{softmax}(\phi)$ over depths $\mathcal{R} = \{1, \ldots, R_{\max}\}$, with learnable logits ϕ. At each optimizer step n, every microbatch draws a depth $r \sim p_{\phi_n}$ and receives supervision at that depth. We learn p_ϕ from the resulting prediction feedback by minimizing

$$\mathcal{J}_{\mathrm{prior}}(\phi) = \underbrace{-\widehat{\mathbb{E}}_n\big[\mathrm{stopgrad}(A_n)\log p_\phi(r)\big]}_{\text{prediction quality}} \underbrace{-\lambda_H H(p_\phi)}_{\text{depth diversity}} + \underbrace{\lambda_m\big(\mathbb{E}_{p_\phi}[R] - \bar{R}\big)^2}_{\text{budget control}}, \tag{5}$$

[4] Appendix C.1 gives the depth distributions and comparison details (Figure 7).

where $\widehat{\mathbb{E}}_n$ averages over microbatches, weighted by their unmasked positions. The advantage A_n scores the sampled depth. We map the mean next-token CE ℓ to a reward $u = \omega(\mathrm{stopgrad}(\ell))$ with a decreasing function ω; we use inverse perplexity, $\omega(\ell) = \exp(-\ell)$. The advantage is then $A_n = s_{n-1}(u - \hat{u}_{n-1})$: an exponential moving average (EMA) baseline $\hat{u}_{n-1}$ centers the reward, and $s_{n-1} = \sigma_{\ell,n-1}/\sigma_{u,n-1}$ rescales it to the scale of CE, following PopArt (van Hasselt et al., 2016; Hessel et al., 2019). A positive A_n thus raises the probability of the sampled depth.

Entropy regularization. Prediction feedback under a budget constraint favors depths that already predict well. Left alone, it concentrates supervision on a narrow range of depths and can weaken fixed-point shaping. The entropy term $H(p_\phi) = -\sum_{r\in\mathcal{R}} p_\phi(r) \log p_\phi(r)$ counters this tendency. Increasing λ_H spreads the depth allocation. Since depths start at 1, the wider allocation extends mainly above $\bar{R}$, toward $R_{\max}$. This trades finite-budget predictive performance for supervision at later depths, where settled prefix context supports KV sharing (Appendix C.1). The coefficient λ_H thus gives a direct knob for balancing performance and memory efficiency.

Budget control. The third term penalizes deviations of the expected depth from the target $\bar{R}$, keeping the allocation near the prescribed forward recurrence budget. Appendix B.3 gives one update step (Algorithm 1) and implementation details.

4.2 Orthogonal Input Injection

Input injection feeds the same embedding q_t into every recurrence (Bai et al., 2019a; Geiping et al., 2025; Prairie et al., 2026), keeping the fixed points input conditioned. A common additive form is

$$\zeta_t^r = M h_t^r + q_t, \qquad \mathbf{H}^{r+1} = \Gamma_\theta(\zeta_1^r, \ldots, \zeta_N^r), \tag{6}$$

where h_t^r is the recurrent state at token t and recurrence r, M controls state carryover, and Γ_θ is the recurrent transformer core. Parcae (Prairie et al., 2026) uses a diagonal carryover $M = \Lambda = \mathrm{diag}(\exp(-\Delta \odot \alpha))$ with learned positive vectors Δ and α, so Λ has entries in $(0, 1)$ and $\|\Lambda\|_2 < 1$. It injects the prelude output $e_t = P_\theta(x)_t$ as $q_t = \Delta \odot W e_t$.[5]

Orthogonal injection. The carryover Λh_t^r can have a component along q_t that adds to or cancels the injected signal, so how much of the input a fixed point retains depends on the state. OrthoInj removes this component:

$$\zeta_t^r = Q_{q_t} \Lambda h_t^r + q_t, \qquad Q_{q_t} = I - \frac{q_t q_t^\top}{\|q_t\|_2^2 + \epsilon}. \tag{7}$$

For $q_t \neq 0$ and $\epsilon = 0$, the component of ζ_t^r along q_t is exactly q_t at every recurrence, including at a fixed point, while the orthogonal part evolves freely; the stabilizer $\epsilon > 0$ makes this approximate. Since $\|Q_{q_t}\|_2 \leq 1$, the projected carryover stays contractive: $\|Q_{q_t}\Lambda\|_2 \leq \|\Lambda\|_2 < 1$. Appendix B.4 collects all injection recipes.

5 Experiments

We first evaluate the two training changes from Section 4: the learned depth prior and OrthoInj. We then test distilled prefill (Section 3.2); Section 3.1 summarizes the RL results.

5.1 Setup

Models and training. We build looped models from Llama Transformer blocks (Touvron et al., 2023) at Small (S), Medium (M), and Large (L) scales (Table 1). Following Huginn (Geiping et al., 2025), each model has one prelude block, a two-block recurrent core, and one coda block. D4 is the untied baseline with the same physical depth and parameter count.

Table 1: **Training scales.**

Scale	Parameters	Tokens (B)
S	100M	21.5
M	400M	85.9
L	1.6B	343.6

[5] Parcae applies RMSNorm to e_t before injection; OrthoInj drops it. Ablations in Appendix C.2.

We train on a weighted multilingual mixture that includes TxT360 (Tang et al., 2024), tokenized with the Jais64k tokenizer (Sengupta et al., 2023). Optimization uses AdamW (Loshchilov & Hutter, 2019) with a warmup–stable–decay schedule (Hu et al., 2024): 5% linear warmup, then 10% cosine decay to one-tenth of the peak learning rate. We sweep the peak learning rate at each scale and tune the main configurations separately (Appendix B.1).

Evaluation. We report perplexity (PPL) on MIXVAL, a held-out set drawn from the TxT360 sources, and on WikiText-103 (Merity et al., 2016). Downstream accuracy covers LAMBADA (Paperno et al., 2016), HellaSwag (HS) (Zellers et al., 2019), PIQA (Bisk et al., 2020), ARC-Easy/Challenge (Clark et al., 2018), OpenBookQA (OBQA) (Mihaylov et al., 2018), and SciQ (Welbl et al., 2017). Generation uses GSM8K (GSM) (Cobbe et al., 2021), DROP (Dua et al., 2019), and MBPP+ (Austin et al., 2021; Liu et al., 2023). Unless noted, we evaluate at $R = 5$ with terminal KV sharing (Section 3.2): prefill caches only the terminal KV of each prompt toke, and choice tasks and generation then decode one token at a time. For PPL, we prefill the first half of each 8,192-token sequence and score the second half in one parallel pass, whose tokens read the prefix through terminal banks and one another as in training. Appendix D gives the full protocols.

5.2 LEARNING THE DEPTH PRIOR

A good depth prior should give low loss at the target depth and lose little when every iteration shares the terminal iteration's KV cache. We compare four priors with mean depth 5: fixed-depth training (Fixed-5), Huginn's Poisson log-normal prior (PLN-5), and our learned prior with and without the entropy term. Both learned priors start from PLN-5 and cap depth at $R_{\max} = 64$.

Table 2: **Fixed-depth training breaks KV sharing; the learned prior keeps it.** All models are evaluated at $R = 5$ with terminal KV sharing, which keeps four of the twelve KV banks; Sampled-depth rows train with the capped PLN prior at mean five. FLOPs are relative to Fixed R=5 at the same scale. Val. and Wiki prefill 4,096 tokens and score the next 4,096 (Appendix D.3). Generation reports GSM8K accuracy, DROP F1, and MBPP+ pass@1.

Training		PPL ↓		Downstream (%) ↑								Generation (%) ↑		
Prior	FLOPs	Val.	Wiki	LAMB.	HS	PIQA	ARC-E	ARC-C	OBQA	SciQ	AVG	GSM	DROP	MBPP+
S scale: (100M, 21.5B tokens)														
Fixed R=5	1.00	6.82	20.59	18.24	31.03	62.24	46.13	**25.09**	15.20	78.20	39.45	1.36	8.33	0.00
Fixed PLN-5	0.99	6.79	20.19	26.41	32.42	61.70	**47.56**	23.98	16.20	81.00	41.32	1.52	13.29	0.75
Learned, λ_H=0	0.99	6.68	**19.85**	26.97	32.23	62.57	45.66	24.83	16.80	**81.70**	41.54	1.44	12.54	**0.88**
Learned, λ_H=0.01	0.99	**6.67**	19.91	**27.36**	**32.51**	**63.00**	45.50	**25.09**	**17.20**	80.90	**41.65**	**1.59**	**14.08**	0.62
M scale: (400M, 85.9B tokens)														
Fixed R=5	1.00	4.68	12.79	34.99	41.83	67.03	56.65	29.61	**19.40**	**88.70**	48.32	3.34	14.98	3.25
Fixed PLN-5	0.98	4.67	12.55	**39.69**	44.64	67.68	56.61	**29.78**	16.40	88.60	49.06	13.72	17.75	10.38
Learned, λ_H=0	1.00	**4.57**	**12.24**	39.22	45.63	**68.88**	**58.75**	28.07	**19.40**	88.10	**49.72**	14.33	**20.29**	**13.12**
Learned, λ_H=0.01	1.00	4.58	12.28	38.58	**46.17**	67.41	57.58	28.75	17.00	88.00	49.07	**15.62**	17.11	10.38
L scale: (1.6B, 343.6B tokens)														
Fixed R=5	1.00	3.63	8.92	48.85	58.19	72.31	66.29	36.01	**24.40**	92.30	56.90	21.23	27.98	12.75
Fixed PLN-5	0.98	3.58	8.76	**55.17**	60.80	73.56	69.74	38.74	23.80	93.30	59.30	47.61	34.11	27.00
Learned, λ_H=0	1.00	3.56	8.75	53.02	59.79	72.63	68.35	38.82	24.00	92.50	58.44	36.47	31.01	19.38
Learned, λ_H=0.01	1.00	**3.54**	**8.68**	**55.17**	**61.57**	**73.72**	**71.09**	**40.44**	**24.40**	**93.60**	**60.00**	**47.92**	**34.56**	**27.25**

Fixed-depth training breaks KV sharing. Terminal KV sharing gives fixed-depth training the highest PPL at every scale and cuts its GSM8K accuracy by over half at M and L (Table 2).[6]

The learned prior keeps sharing intact and outperforms PLN-5. With terminal KV sharing, it lowers validation PPL relative to Fixed PLN-5 by 1.1–1.9%, for at most 1.6% more training FLOPs. It has the highest downstream average at S and L, ties Fixed PLN-5 at M, and leads all three generation tasks at L. The entropy term is essential at L: without it, GSM8K drops from 47.92 to 36.47. Appendix C.1 sweeps λ_H (Figure 5), gives training curves and further ablations.

[6] with full KV cache, Fixed R=5 has the lowest PPL at every scale, consistent with Figure 3. Full-KV results are in Appendix C.1.

5.3 Input Injection

We compare OrthoInj with DEQ-QKV (Bai et al., 2019a), Huginn-Linear (Geiping et al., 2025), and Parcae-Decay (Prairie et al., 2026), all trained with PLN-5 (Details in Appendix B.4).

Table 3: **OrthoInj gives better performance.** All models train with PLN-5 and are evaluated at $R = 5$ with terminal KV sharing, as in Table 2. Recipes are in Appendix B.4.

Model	PPL ↓		Downstream (%) ↑								Generation (%) ↑		
Injection	Val.	Wiki	LAMB.	HS	PIQA	ARC-E	ARC-C	OBQA	SciQ	AVG	GSM	DROP	MBPP+
S scale: (100M, 21.5B tokens)													
DEQ-QKV	7.40	23.05	25.17	30.56	60.99	43.31	23.81	15.40	76.10	39.33	**2.05**	9.42	0.12
Huginn-Linear	6.99	20.78	25.69	31.51	**62.35**	43.18	**24.66**	**17.20**	78.60	40.46	1.90	12.21	0.25
Parcae-Decay	6.78	**19.90**	26.28	32.14	61.86	**46.89**	23.81	16.40	81.60	41.28	1.90	12.08	0.25
OrthoInj	**6.67**	20.02	**27.54**	**32.69**	61.97	46.46	**24.66**	16.40	**81.80**	**41.65**	1.90	**13.66**	**1.62**
M scale: (400M, 85.9B tokens)													
Huginn-Linear	4.66	12.55	36.56	44.15	**68.12**	56.69	**29.44**	17.00	88.70	48.67	16.68	16.71	8.00
Parcae-Decay	4.62	12.45	37.78	44.63	**68.12**	57.58	27.90	**19.40**	87.80	49.03	15.01	20.17	8.25
OrthoInj	**4.59**	**12.32**	**38.37**	**46.16**	67.57	**57.95**	28.50	17.40	**89.10**	**49.29**	**17.51**	**21.04**	**10.25**

OrthoInj outperforms existing input injection schemes. With terminal KV sharing, it improves validation PPL over the strongest baseline, Parcae-Decay, by 1.6% at S and 0.7% at M, and has the highest downstream average at both scales (Table 3); at M it also leads all three generation tasks. Without KV reuse, it has the lowest validation and WikiText PPL at both scales. Appendix C.2 reports these results and ablates prelude normalization and projection.

5.4 Distilled Prefill

We distill the S and M learned-prior models of Section 5.2 into two-block students initialized from the teacher's recurrent core (Section 3.2). Every prefill path is followed by the same teacher decoding at $R = 5$, on the tasks of Table 2. We compare against full teacher prefill ($R = 5$), a single teacher recurrence ($R = 1$), and the dense D4 baseline, which prefills and decodes on its own.

Table 4: **Teacher-initialized distilled prefill keeps most of the teacher's quality at lower prefill latency.** All rows except D4 prefill the prompt with the listed path and then decode with the same learned-prior teacher at $R = 5$ and four terminal KV banks; D4 is the dense baseline and decodes itself. Both distilled students are trained with Eq. (3) (hidden and KL terms) and differ only in initialization. Val. and Wiki PPL: the path prefills the first $4{,}096$ tokens and the decoder scores the rest. Latency: mean $\pm$ s.d. of six 8K prefills at batch size 8.

Prefill	PPL ↓		Downstream (%) ↑								Generation (%) ↑			Latency ↓
Path	Val.	Wiki	LAMB.	HS	PIQA	ARC-E	ARC-C	OBQA	SciQ	AVG	GSM	DROP	MBPP+	8K (ms)
S scale: (100M, 21.5B tokens)														
D4	7.94	25.19	23.95	30.26	59.96	44.57	23.21	16.60	77.60	39.45	1.97	11.39	0.12	100.7±0.5
Teacher (R=5)	6.67	19.91	27.36	32.51	63.00	45.50	25.09	17.20	80.90	41.65	1.59	14.08	0.62	195.0±1.4
One recurrence (R=1)	6.85	21.02	19.29	31.13	61.21	41.62	23.98	15.40	73.70	38.05	1.52	10.86	0.00	110.7±0.1
Distilled, random init	6.72	20.32	20.88	31.47	62.68	45.12	24.57	16.00	79.00	39.96	1.52	12.44	0.00	125.3±0.7
Distilled, teacher init	6.70	20.16	23.56	31.93	63.17	43.86	25.17	17.00	79.70	40.63	1.74	14.41	0.62	125.9±1.0
M scale: (400M, 85.9B tokens)														
D4	5.55	15.73	31.19	37.43	63.71	50.72	27.65	15.80	82.10	44.08	2.35	13.12	2.50	229.8±1.2
Teacher (R=5)	4.58	12.28	38.58	46.17	67.41	57.58	28.75	17.00	88.00	49.07	15.62	17.11	10.38	495.8±3.5
One recurrence (R=1)	4.73	13.12	13.76	41.60	64.53	49.45	27.90	15.40	75.70	41.19	2.27	8.56	0.12	245.9±2.6
Distilled, random init	4.63	12.57	29.52	44.06	67.19	53.28	27.56	14.20	83.80	45.66	7.05	15.36	2.50	295.3±2.4
Distilled, teacher init	4.60	12.44	31.86	44.63	67.74	55.68	27.90	15.20	84.40	46.77	8.95	16.20	6.12	299.3±1.6

Distilled prefill is 1.5–$1.7\times$ faster than teacher prefill and keeps most of its quality. On 8K prompts, it stays within 0.03 of the teacher's validation PPL and trails its downstream average by 1.0 (S) and 2.3 (M) points (Table 4); at M it keeps more than half of the teacher's GSM8K and MBPP+ accuracy. The cheaper paths lose far more. A single teacher recurrence cuts latency only 12–18% further but loses about $3.5\times$ as much downstream accuracy. D4 is the fastest path, yet at M it is worse than distilled prefill on all but one quality metric.

Teacher initialization drives the student's quality, and the KL term raises its generation scores. Under the same hidden and KL objective, teacher initialization adds 0.7 (S) and 1.1 (M) downstream points and raises M MBPP+ from 2.50 to 6.12; adding the KL term raises a randomly initialized student's M MBPP+ from 0.25 to 2.50. Appendix C.6.1 ablates the loss terms and other uses of the student, and Appendix C.6.2 distills a deeper $R = 25$ teacher.

6 Related Work

Looped models and fixed points. Looped language models apply a shared block repeatedly (Dehghani et al., 2019) to scale test-time compute (Anil et al., 2022; Yang et al., 2026) and latent reasoning (Geiping et al., 2025; Zhu et al., 2025). As depth grows, their states approach fixed points (Geiping et al., 2025; Blayney et al., 2026), which DEQs solve for directly (Bai et al., 2019a). Prior work uses these fixed points to change how many recurrences run: it stabilizes them (Bai et al., 2021; Yang et al., 2026; Li et al., 2026), trains supervised states as attractors (Huang et al., 2026), or halts at convergence (Movahedi et al., 2026). We further identify and verify the efficiency gains that fixed points bring to training, inference, and post-training.

Looped model efficiency. Near a fixed point, further recurrences barely change the state. We show that this property explains two shortcuts looped models already take and enables two new ones. Truncated backpropagation (Geiping et al., 2025; Wang et al., 2025) thus approximates the equilibrium gradient without the warm-up that implicit gradients need (Liao et al., 2018; Geng et al., 2021; Wu Fung et al., 2021). Of the schemes that share KV across recurrences (Geiping et al., 2025; Zhu et al., 2025; Bae et al., 2025b), terminal sharing is likewise nearly lossless when training supervises near fixed points. Terminal sharing also opens a shortcut for prefill. While looped-model distillation keeps students recurrent (Appendix E.5), and prompt-KV predictors (Horton et al., 2024; Qiao et al., 2024) must predict each recurrence's KV; distilled prefill instead uses a non-recurrent student to predict only the endpoint, from which one teacher recurrence builds every KV bank. Furthermore, fixed-point reuse carries endpoints across computations: it warm-starts solvers on related inputs (Appendix E.4), and we bring it to RL, scoring rollouts from saved states instead of replaying the recurrence (Zhu et al., 2025).

Shaping fixed points. Two choices shape the fixed point: which depths training supervises, and how the input enters each recurrence. Randomized training depths promote path-independent equilibria (Anil et al., 2022), but existing depth distributions are fixed (Geiping et al., 2025), marginalized over at inference (Antorán et al., 2020), or learned as exits (Graves, 2016; Banino et al., 2021). Closest to ours, Ouro (Zhu et al., 2025) and RL-Halting (Kuo et al., 2026) learn entropy-regularized depth distributions from the recurrent state, but they unroll every training step to the maximum depth and do not target fixed-point shaping (Appendix E.1). Injecting the input into every recurrence makes the fixed point depend on it (Bai et al., 2019a; Bansal et al., 2022; Geiping et al., 2025). Parcae (Prairie et al., 2026) improves injection by forcing contractiveness to stablize training; We further observe that, the carried-over state can have a component along the injected input; OrthoInj projects it out so every recurrence receives the input intact (Appendix E.2).

7 Discussion and Conclusion

Near a fixed point, the endpoint of a looped language model can replace its recurrent trajectory in training, inference, and post-training. Whether training produces such fixed points depends on the depth prior and on input injection. A learned depth prior lowers perplexity over a fixed PLN prior at the same mean depth while keeping terminal KV sharing intact. OrthoInj projects the carried-over state off the injected input and drops the prelude normalization, giving lower perplexity than the injection schemes of Huginn and Parcae. On these models, terminal KV sharing, distilled prefill, and rollout-state reuse each cut memory or time at a small cost in quality.

Limitations. We train models of up to 1.6B parameters with one seed per configuration, and our ablations do not exhaust the design space. Larger scales and other model configurations remain untested, and the feasibility of these techniques in real-world applications needs further study.

AI USE STATEMENT

In this work, we used generative AI tools to implement methods, formulate mathematical claims, assist in writing proofs, and provide feedback on experiments. We did not use them to develop theoretical models or conceptual frameworks, provide critical ingredients for proofs, propose hypotheses, design experiments, or interpret results; the remaining required categories are not applicable. Additionally, we used generative AI tools to draft and edit the paper, make figures, identify relevant literature, and check and format references. The authors reviewed all AI-assisted work, checked every reported number against the raw results, and take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

Appendix A gives the assumptions and proofs of our theoretical results. Appendix B specifies the model configurations, training setup, learned depth prior (Algorithm 1), and input-injection recipes. Appendix C gives the settings of each experiment, and Appendix D defines the datasets and evaluation protocols.

REFERENCES

Christopher Aicher, Nicholas J. Foti, and Emily B. Fox. Adaptively truncating backpropagation through time to control gradient bias. In *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pp. 799–808, 2019. URL https://proceedings.mlr.press/v115/aicher20a.html.

Cem Anil, Ashwini Pokle, Kaiqu Liang, Johannes Treutlein, Yuhuai Wu, Shaojie Bai, J. Zico Kolter, and Roger B. Grosse. Path independent equilibrium models can better exploit test-time computation. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/331c41353b053683e17f7c88a797701d-Abstract-Conference.html.

Javier Antorán, James Urquhart Allingham, and José Miguel Hernández-Lobato. Depth uncertainty in neural networks, 2020. URL https://arxiv.org/abs/2006.08437.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise LoRA. In *International Conference on Learning Representations*, 2025a. URL https://arxiv.org/abs/2410.20672.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-Recursions: Learning dynamic recursive depths for adaptive token-level computation, 2025b. URL https://arxiv.org/abs/2507.10524.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, volume 32, 2019a. URL https://proceedings.neurips.cc/paper/2019/hash/01386bd6d8e091c2ab4c7c7de644d37b-Abstract.html.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Trellis networks for sequence modeling. In *International Conference on Learning Representations*, 2019b. URL https://arxiv.org/abs/1810.06682.

Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Multiscale deep equilibrium models. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://arxiv.org/abs/2006.08656.

Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Stabilizing equilibrium models by Jacobian regularization. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 554–565. PMLR, 2021. URL https://proceedings.mlr.press/v139/bai21b.html.

Shaojie Bai, Zhengyang Geng, Yash Savani, and J. Zico Kolter. Deep equilibrium optical flow estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 620–630, 2022a. URL https://arxiv.org/abs/2204.08442.

Shaojie Bai, Vladlen Koltun, and J. Zico Kolter. Neural deep equilibrium solvers. In *International Conference on Learning Representations*, 2022b. URL https://openreview.net/forum?id=B0oHOwT5ENL.

Xingjian Bai and Luke Melas-Kyriazi. Fixed point diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9430–9440, 2024. URL https://arxiv.org/abs/2401.08741.

Andrea Banino, Jan Balaguer, and Charles Blundell. PonderNet: Learning to ponder, 2021. URL https://arxiv.org/abs/2107.05407.

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. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/7f70331dbe58ad59d83941dfa7d975aa-Abstract-Conference.html.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/6239.

Hugh Blayney, Álvaro Arroyo, Johan Obando-Ceron, Pablo Samuel Castro, Aaron Courville, Michael M. Bronstein, and Xiaowen Dong. A mechanistic analysis of looped reasoning language models, 2026. URL https://arxiv.org/abs/2604.11791.

Jang Hyun Cho and Bharath Hariharan. On the efficacy of knowledge distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4794–4802, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? Try ARC, the AI2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

Victor Conchello Vendrell, Arnau Padrés Masdemont, Niccolò Grillo, Jordi Ros-Giralt, Arash Behboodi, and Fabio Valerio Massoli. Memory-efficient looped transformer: Decoupling compute from memory in looped language models, 2026. URL https://arxiv.org/abs/2605.07721.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1807.03819.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of NAACL-HLT*, pp. 2368–2378, 2019. URL https://aclanthology.org/N19-1246/.

Can Ufuk Ertenli, Emre Akbas, and Ramazan Gokberk Cinbis. Streaming multiscale deep equilibrium models. In *European Conference on Computer Vision*, 2022. doi: 10.1007/978-3-031-20083-0_12.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 2023. URL https://zenodo.org/records/10256836.

Zitian Gao, Yilong Chen, Yihao Xiao, Xinyu Yang, Ran Tao, Joey Zhou, and Bryan Dai. Loop the loopies!, 2026. URL https://arxiv.org/abs/2607.16051.

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, 2025. URL https://arxiv.org/abs/2502.05171.

Zhengyang Geng, Xin-Yu Zhang, Shaojie Bai, Yisen Wang, and Zhouchen Lin. On training implicit models. In *Advances in Neural Information Processing Systems*, 2021. URL https://arxiv.org/abs/2111.05177.

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

Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with PopArt. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3796–3803, 2019. doi: 10.1609/aaai.v33i01.33013796. URL https://ojs.aaai.org/index.php/AAAI/article/view/4266.

Maxwell Horton, Qingqing Cao, Chenfan Sun, Yanzi Jin, Sachin Mehta, Mohammad Rastegari, and Moin Nabi. KV prediction for improved time to first token, 2024. URL https://arxiv.org/abs/2410.08391.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, Xinrong Zhang, Zheng Leng Thai, Kaihuo Zhang, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, Chao Jia, Guoyang Zeng, Dahai Li, Zhiyuan Liu, and Maosong Sun. MiniCPM: Unveiling the potential of small language models with scalable training strategies. In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=3X2L2TFr0f.

Benhao Huang, Zhengyang Geng, and Zico Kolter. Equilibrium reasoners: Learning attractors enables scalable reasoning. *arXiv preprint arXiv:2605.21488*, 2026.

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. URL https://arxiv.org/abs/2602.11451.

Hsun-Yu Kuo, El Mahdi Chayti, Patrik Reizinger, Wieland Brendel, and Martin Jaggi. Stabilizing extrapolation in looped transformers via learned stochastic stopping, 2026. URL https://arxiv.org/abs/2606.29983.

Aakash Lahoti, Kevin Y. Li, Berlin Chen, Caitlin Wang, Aviv Bick, J. Zico Kolter, Tri Dao, and Albert Gu. Mamba-3: Improved sequence modeling using state space principles. *arXiv preprint arXiv:2603.15569*, 2026. URL https://arxiv.org/abs/2603.15569.

Shuzhen Li, Yifan Zhang, Jiacheng Guo, Quanquan Gu, and Mengdi Wang. DeepLoop: Depth scaling for looped transformers, 2026. URL https://arxiv.org/abs/2607.13491.

Renjie Liao, Yuwen Xiong, Ethan Fetaya, Lisa Zhang, KiJung Yoon, Xaq Pitkow, Raquel Urtasun, and Richard Zemel. Reviving and improving recurrent back-propagation. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3082–3091, 2018. URL https://proceedings.mlr.press/v80/liao18c.html.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, 2024.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation, 2023. URL https://arxiv.org/abs/2305.01210.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-like training: A critical perspective, 2025. URL https://arxiv.org/abs/2503.20783.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016. URL https://arxiv.org/abs/1609.07843.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? A new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, 2018. URL https://aclanthology.org/D18-1260/.

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5191–5198, 2020.

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.

Bac Nguyen and Lukas Mauch. Efficient training of deep equilibrium models, 2023. URL https://arxiv.org/abs/2304.11663. ICML 2022 Workshop.

Giyeong Oh, Woohyun Cho, Siyeol Kim, Suhwan Choi, and Youngjae Yu. Revisiting residual connections: Orthogonal updates for stable and efficient deep networks, 2025. URL https://arxiv.org/abs/2505.11881.

James O'Neill and Fergal Reid. Looped latent attention: Cross-loop KV compression for looped transformers, 2026. URL https://arxiv.org/abs/2607.15456.

J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*, volume 30 of *Classics in Applied Mathematics*. SIAM, 2000. doi: 10.1137/1.9780898719468. URL https://epubs.siam.org/doi/book/10.1137/1.9780898719468. Unabridged republication of the 1970 Academic Press edition.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 1525–1534, 2016. URL https://aclanthology.org/P16-1144/.

Andrei Cristian Popescu, Haitz Sáez de Ocáriz Borde, and Pietro Liò. Adaptive depth in looped transformers: Diagnosing learned halting gates and trajectory readouts, 2026. URL https://arxiv.org/abs/2607.20519.

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

Aurick Qiao, Zhewei Yao, Samyam Rajbhandari, and Yuxiong He. SwiftKV: Fast prefill-optimized inference with knowledge-preserving model transformation, 2024. URL https://arxiv.org/abs/2410.03960.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. FitNets: Hints for thin deep nets. In *International Conference on Learning Representations*, 2015. URL https://arxiv.org/abs/1412.6550.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial Winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/6399.

Sandy Research. Parcae: Official implementation, 2026. URL https://github.com/sandyresearch/parcae. Initialization: parcae_lm/utils/init.py and parcae_lm/models/parcae/init.py. Accessed September 15, 2026.

Kristian Schwethelm, Daniel Rückert, and Georgios Kaissis. Depth-adaptive inference of looped language models via continuous depth batching, 2026. URL https://arxiv.org/abs/2608.09444.

Neha Sengupta, Sunil Kumar Sahu, Bokang Jia, Satheesh Katipomu, Haonan Li, Fajri Koto, William Marshall, Gurpreet Gosal, Cynthia Liu, Zhiming Chen, Osama Mohammed Afzal, Samta Kamboj, Onkar Pandit, Rahul Pal, Lalit Pradhan, Zain Muhammad Mujahid, Massa Baali, Xudong Han, Sondos Mahmoud Bsharat, Alham Fikri Aji, Zhiqiang Shen, Zhengzhong Liu, Natalia Vassilieva, Joel Hestness, Andy Hock, Andrew Feldman, Jonathan Lee, Andrew Jackson, Hector Xuguang Ren, Preslav Nakov, Timothy Baldwin, and Eric Xing. Jais and Jais-chat: Arabic-centric foundation and instruction-tuned open generative large language models. *arXiv preprint arXiv:2308.16149*, 2023. URL https://arxiv.org/abs/2308.16149.

Liping Tang, Nikhil Ranjan, Omkar Pangarkar, Xuezhi Liang, Zhen Wang, Li An, Bhaskar Rao, Linghao Jin, Huijuan Wang, Zhoujun Cheng, Suqi Sun, Cun Mu, Victor Miller, Xuezhe Ma, Yue Peng, Zhengzhong Liu, and Eric P. Xing. TxT360: A top-quality LLM pre-training dataset requires the perfect blend. Hugging Face dataset, 2024. URL https://huggingface.co/datasets/LLM360/TxT360.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. URL https://arxiv.org/abs/2302.13971.

Hado P. van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems*, volume 29, 2016. URL https://proceedings.neurips.cc/paper/2016/hash/5227b6aaf294f5f027273aebf16015f2-Abstract.html.

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. URL https://arxiv.org/abs/2506.21734.

Shaowen Wang, Ge Zhang, Kairong Luo, Yuhao Wu, Shaofan Liu, Jiaheng Liu, Wenhao Huang, Shen Yan, and Jian Li. SMELT: Scaling laws for compute-matched MoE looped transformers, 2026. URL https://arxiv.org/abs/2609.01343.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, 2017. URL https://aclanthology.org/W17-4413/.

Ronald J. Williams and Jing Peng. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 2(4):490–501, 1990. doi: 10.1162/neco.1990.2.4.490. URL https://doi.org/10.1162/neco.1990.2.4.490.

Samy Wu Fung, Howard Heaton, Qiuwei Li, Daniel McKenzie, Stanley Osher, and Wotao Yin. JFB: Jacobian-free backpropagation for implicit networks, 2021. URL https://arxiv.org/abs/2103.12803.

Liu Yang, Kangwook Lee, Robert D. Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2311.12424.

Xiao-Wen Yang, Ziyu Han, Xi-Hua Zhang, Wen-Da Wei, Jie-Jing Shao, Lan-Zhe Guo, and Yu-Feng Li. Stabilizing recurrent dynamics for test-time scalable latent reasoning in looped language models. *arXiv preprint arXiv:2605.26733*, 2026. URL https://arxiv.org/abs/2605.26733.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019. URL https://aclanthology.org/P19-1472/.

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

APPENDIX CONTENTS

Appendix A gives the theoretical analysis, including the proof of Lemma 1. Appendix B specifies the methods and training configurations, including why depth sampling exposes the model to converged context. Appendix C collects supplementary experiments, including rollout-state reuse in RL, and Appendix D defines the shared datasets and evaluation protocols. Appendix E relates the learned depth prior, OrthoInj, terminal KV sharing, rollout-state reuse, and distilled prefill to prior work.

A Theoretical Analysis and Proofs **17**
A.1 Joint Iteration and Frozen-Prefix Decoding . . . 17
A.2 Truncated Backpropagation and Implicit Gradients . . . 18

B Additional Method Details **20**
B.1 Models and Training Contract . . . 20
B.2 Why Sample Training Depths? . . . 20
B.3 Learned Depth Prior . . . 20
B.4 Architecture Variants and Initialization . . . 22

C Additional Experimental Results **23**
C.1 Depth-Prior Learning and Test-Time Scaling . . . 23
C.2 Input Injection . . . 26
C.3 Truncation, Depth Sampling, and Fixed Points . . . 27
C.4 Recurrent-State and Cache Dynamics . . . 28
C.5 Independent Prefill and Decode Budgets . . . 30
C.6 Distilled Prefill . . . 31
C.7 Reusing States during Post-Training . . . 33

D Datasets and Evaluation Protocols **35**
D.1 Mixture-Matched Validation . . . 35
D.2 WikiText-103 and Zero-Shot Tasks . . . 35
D.3 Terminal-KV Evaluation . . . 36
D.4 Cached-Continuation Perplexity . . . 37
D.5 Generation Benchmarks . . . 37

E Extended Related Work **37**
E.1 Learned Depth Distributions and Exits . . . 37
E.2 Input Injection and Orthogonal Updates . . . 38
E.3 KV Sharing Across Recurrent Depth . . . 38
E.4 Fixed-Point Reuse . . . 38
E.5 Distillation for Looped Models . . . 39

A Theoretical Analysis and Proofs

We prove Lemma 1 and extend it to token-by-token decoding and finite depth (Appendix A.1), and relate truncated backpropagation to implicit gradients (Appendix A.2).

A.1 Joint Iteration and Frozen-Prefix Decoding

Fix parameters, tokens, injection, positions, and masks. As in Section 3.2, h includes all current-token recurrent variables and C all prefix inputs, including per-layer KV and additional recurrent state. If C contains inputs other than KV, sharing only the KV banks falls under Lemma 1 only when those other inputs also coincide. By the triangle inequality, the assumptions of Lemma 1 give, for all $h, \widetilde{h} \in \mathcal{H}$ and $C, \widetilde{C} \in \mathcal{C}$, the joint Lipschitz bound

$$\left\| f_\theta(h; C) - f_\theta(\widetilde{h}; \widetilde{C}) \right\| \le \kappa \|h - \widetilde{h}\| + \beta \|C - \widetilde{C}\|. \tag{8}$$

Proof of Lemma 1. By the contraction mapping theorem (Ortega & Rheinboldt, 2000), $f_\theta(\cdot; C^\star_{<t})$ has a unique fixed point $h^\star$ in the complete space $\mathcal{H}$, and $\widetilde{h}^r \to h^\star$. Let $\delta_r = \|h^r - h^\star\|$. Invariance keeps $h^r \in \mathcal{H}$, so Equation (8) and $h^\star = f_\theta(h^\star; C^\star_{<t})$ give

$$\delta_{r+1} \le \kappa\, \delta_r + \beta \|C^r_{<t} - C^\star_{<t}\|. \tag{9}$$

With $\eta_{\max} = \sup_r \|C^r_{<t} - C^\star_{<t}\| < \infty$, induction gives $\delta_r \le \max\{\delta_0, \beta\eta_{\max}/(1-\kappa)\}$, so $\bar{\delta} = \limsup_r \delta_r$ is finite, and taking $\limsup$ in Equation (9) gives $\bar{\delta} \le \kappa\, \bar{\delta}$. Hence $\bar{\delta} = 0$ and $h^r \to h^\star$. Both iterations thus reach $h^\star$ and $C^\star_{<t}$, so any readout continuous in both has the same limit. □

Extension to one-by-one decoding. For N tokens and nonempty closed sets $\mathcal{H}_t \subseteq \mathbb{R}^{d_t}$, write

$$C^r_{<t} = \Phi_{\theta,t}(h^r_{<t}), \qquad h^{r+1}_t = f_{\theta,t}(h^r_t; C^r_{<t}), \tag{10}$$

where $h^r_{<t} = (h^r_1, \dots, h^r_{t-1})$ and $C^r_{<1} = C_{<1}$. Each $\Phi_{\theta,t}$ is continuous and depth-shared, and for every context $C \in \Phi_{\theta,t}(\prod_{t'<t} \mathcal{H}_{t'})$, $f_{\theta,t}(\cdot; C)$ maps $\mathcal{H}_t$ into itself; on these states and contexts, $f_{\theta,t}$ satisfies Equation (8) with constants $\kappa_t < 1$ and $\beta_t < \infty$. From initial states in these sets, joint iteration and sequential decoding, which solves and freezes each token in turn, converge to the same limiting states; with common continuous readouts, they also give the same predictions.

Proof. Induct on t. At $t = 1$ both executions use the fixed context $C_{<1}$ and converge to the fixed point $h^\star_1$ of $f_{\theta,1}(\cdot; C_{<1})$. If $h^r_{t'} \to h^\star_{t'}$ for all $t' < t$, continuity gives $C^r_{<t} \to \Phi_{\theta,t}(h^\star_{<t}) =: C^\star_{<t}$, the context that sequential decoding freezes at position t. Invariance and closedness keep $C^r_{<t}$ and $C^\star_{<t}$ in $\Phi_{\theta,t}(\prod_{t'<t} \mathcal{H}_{t'})$, so Lemma 1 gives $h^r_t \to h^\star_t$. Continuity of the readouts completes the proof, whatever the order in which joint positions converge. □

Finite depth and approximate terminal context. Let $\eta = \|\bar{C}_{<t} - C^\star_{<t}\|$ and $\bar{h}^{r+1} = f_\theta(\bar{h}^r; \bar{C}_{<t})$, and assume that the invariance and Lipschitz conditions also hold for $\bar{C}_{<t}$. Equation (9) with $h^r, C^r_{<t}$ replaced by $\bar{h}^r, \bar{C}_{<t}$ unrolls to

$$\|\bar{h}^R - h^\star\| \le \kappa^R \|\bar{h}^0 - h^\star\| + \frac{\beta(1-\kappa^R)}{1-\kappa}\, \eta. \tag{11}$$

The two terms capture unfinished token refinement and imperfect context, respectively; the second bounds the fixed-point displacement by $\beta\eta/(1-\kappa)$. A readout or loss with state and context Lipschitz constants L_h and L_C differs by at most L_h times the right-hand side of Equation (11) plus $L_C\eta$. Here η includes errors in earlier decoded tokens' banks.

Empirical corroboration. We compare joint iteration with current-token iteration after solving and freezing the prefix, using the matched Fixed $R = 5$ and independently sampled PLN-5 Small Huginn checkpoints of Table 2. On prefixes passing the convergence checks, the two paths reach nearly identical endpoints within each model (Figure 4), consistent with Lemma 1. Freezing a depth-five prefix instead leaves a gap five to six orders of magnitude larger: the median relative endpoint gap is 0.27 for Fixed $R = 5$ and 0.033 for PLN-5, against about 4×10^{-7} with a converged prefix.

This finite-prefix control shows that the lemma needs the converged context. Qualified prefixes need about a hundred or more recurrences to converge, far beyond the depth five used for sharing, so practical sharing rests on the finite-depth bound in Equation (11) and on the measured cached perplexity (Table 2), not on exact convergence.

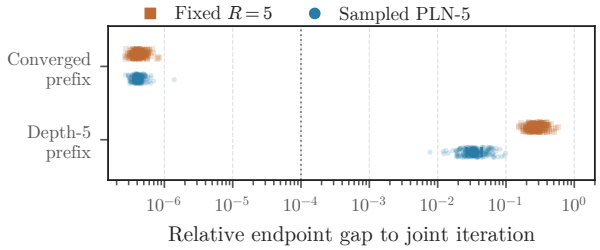


Figure 4: **Freezing a converged prefix reproduces joint iteration (Lemma 1).** Relative gap between the current-token endpoint of joint iteration and of iteration with the prefix frozen after convergence or after five recurrences. Dots are prefixes that pass the convergence checks (206 of 304 for Fixed $R = 5$, 191 of 304 for PLN-5); markers and bars give medians and interquartile ranges. The dotted line marks the 10^{-4} agreement tolerance.

We use 304 spans, 16 from each validation source, with 1,534 prefix tokens and one current token. Prefix-only and joint solves start independently from the model's evaluation-time state initialization. The frozen-prefix path updates only the current token, retaining each physical layer's historical KV and recomputing its own KV. All solves use FP32, PyTorch's reference (math) attention kernel, and at most 1,024 recurrences. A solve qualifies when the maximum per-token relative state change and per-layer K/V change remain at most 10^{-5} for eight steps, followed by 16 confirmation steps with state drift at most 5×10^{-5} and a final one-step check. This criterion rules out cycles.

All three solves (prefix-only, joint, and frozen-prefix current-token) qualify on 206 Fixed $R = 5$ and 191 PLN-5 spans; the other 98 and 113 spans fail these checks and are dropped. Qualified prefix solves need median depths of 152.5 (Fixed $R = 5$) and 114 (PLN-5), with maxima of 890 and 961. Every qualified pair agrees within relative state error 10^{-4} and prediction KL 10^{-7}; the observed maximum state gaps are 8.24×10^{-7} and 1.39×10^{-6}. The same checks pass from zero current-token initialization. The depth-five control uses the same qualified spans and requires its own current-token solve to pass. The two models' qualifying sets need not coincide.

A.2 Truncated Backpropagation and Implicit Gradients

For fixed input x, complete recurrent state $z = (\mathbf{H}, C) \in \mathbb{R}^{d_z}$ of Section 2, and parameters $\theta \in \mathbb{R}^{d_\theta}$, we connect Equation (1) to backpropagation through the final recurrent steps.

Finite-depth and equilibrium objectives. For a training-depth distribution p, the sampled-depth and equilibrium losses optimize different predictions:

$$\mathcal{L}_{\text{depth}} = \mathbb{E}_{R\sim p}[\ell_\theta(z^R, y)], \qquad \mathcal{L}_{\text{eq}} = \mathbb{E}[\ell_\theta(z^\star, y)]. \tag{12}$$

Full BPTT differentiates the finite trajectory; implicit differentiation differentiates a local equilibrium branch. Equal states need not have equal derivatives. For $F_{\theta,a}(z; x) = az + (1 - a)e$ with the prelude output $e = P_\theta(x)$, $z^0 = 0$, and $0 \le a < 1$, the equilibrium $z^\star = e$ is independent of a, whereas $z^R = (1 - a^R)e$ depends on it. Where $e = 0$, both states are zero, but $\mathrm{d}z^R/\mathrm{d}\theta = (1 - a^R)\,\mathrm{d}e/\mathrm{d}\theta$ differs from $\mathrm{d}z^\star/\mathrm{d}\theta = \mathrm{d}e/\mathrm{d}\theta$. TBPTT truncates the trajectory gradient; a convergent Neumann series approximates equilibrium sensitivity. Before convergence, the Neumann estimate is only a surrogate, so comparisons must state the objective and backward budget.

Equilibrium sensitivity and loss gradient. At $z^\star = F_\theta(z^\star; x)$ (Bai et al., 2019a), define $J_\star = \partial_z F_\theta(z^\star; x) \in \mathbb{R}^{d_z \times d_z}$ and $B_\star = \partial_\theta F_\theta(z^\star; x) \in \mathbb{R}^{d_z \times d_\theta}$, holding the state fixed in $B_\star$. Continuous differentiability near the equilibrium and invertibility of $I - J_\star$ give a locally differen-

tiable equilibrium branch by the implicit function theorem:

$$\underbrace{\frac{\mathrm{d}z^\star}{\mathrm{d}\theta}}_{D_\star} = \underbrace{\partial_z F_\theta}_{J_\star} \frac{\mathrm{d}z^\star}{\mathrm{d}\theta} + \underbrace{\partial_\theta F_\theta}_{B_\star}, \qquad (I - J_\star)D_\star = B_\star. \tag{13}$$

Thus $D_\star = (I - J_\star)^{-1}B_\star$. For a single example, $\mathcal{L}_{\mathrm{eq}}(\theta) = \ell_\theta(z^\star, y)$; define column gradients $g = \nabla_z \ell_\theta$ and $c = \nabla_\theta \ell_\theta$ at the equilibrium, holding the state fixed in c. The loss gradient is $G_{\mathrm{imp}} = c + D_\star^\top g$. Its adjoint form,

$$(I - J_\star^\top)v = g, \qquad G_{\mathrm{imp}} = c + B_\star^\top v, \tag{14}$$

uses the vector–Jacobian products (VJPs) $J_\star^\top v$ and $B_\star^\top v$ through one evaluation of F_θ, with v fixed, avoiding an explicit Jacobian or stored forward trajectory (Bai et al., 2019a).

Neumann truncation and a detached suffix. The stronger condition that $J_\star$ has spectral radius below one ensures $J_\star^b \to 0$, so the identity $(I - J_\star)\sum_{k=0}^{b-1} J_\star^k = I - J_\star^b$ yields

$$D_\star = \sum_{k=0}^{\infty} J_\star^k B_\star, \qquad D_b = \sum_{k=0}^{b-1} J_\star^k B_\star, \qquad D_\star - D_b = J_\star^b D_\star. \tag{15}$$

Equivalently, the iteration $v_0 = 0$, $v_{k+1} = g + J_\star^\top v_k$ gives $G_b = c + B_\star^\top v_b$ (Liao et al., 2018; Geng et al., 2021). Invertibility suffices for Equation (14); the spectral condition ensures convergence.

Starting from $\bar{z}^0 = \mathrm{stopgrad}(z^\star)$, each update $\bar{z}^{k+1} = F_\theta(\bar{z}^k; x)$ remains at the equilibrium and uses the same $J_\star, B_\star$. Its sensitivity after k updates is D_k, since $D_0 = 0$ and $D_{k+1} = J_\star D_k + B_\star$; hence the loss gradient is $c + D_b^\top g = G_b$. A Euclidean contraction bound controls the omitted tail:

Lemma 2 (Truncated backpropagation at equilibrium; adapted from Liao et al. (2018))**.** *Suppose F_θ is continuously differentiable near the current equilibrium, ℓ_θ is differentiable there, and $\|J_\star\|_2 \le \kappa < 1$. Start from $\mathrm{stopgrad}(z^\star)$ and differentiate b applications of the same deterministic recurrence, retaining all direct parameter paths. Then*

$$G_b = c + B_\star^\top \sum_{k=0}^{b-1} (J_\star^\top)^k g, \qquad \|G_{\mathrm{imp}} - G_b\|_2 \le \|B_\star\|_2 \, \|g\|_2 \, \frac{\kappa^b}{1-\kappa}. \tag{16}$$

Proof of Lemma 2. The sensitivity recursion above establishes the expression for G_b. Subtracting it from the infinite series and using $\|J_\star\|_2 \le \kappa < 1$ gives

$$\|G_{\mathrm{imp}} - G_b\|_2 = \left\| B_\star^\top \sum_{k=b}^{\infty} (J_\star^\top)^k g \right\|_2 \le \|B_\star\|_2 \, \|g\|_2 \sum_{k=b}^{\infty} \kappa^k = \|B_\star\|_2 \, \|g\|_2 \, \frac{\kappa^b}{1-\kappa}.$$

□

This specializes the TBPTT/Neumann connection (Liao et al., 2018; Geng et al., 2021). If only the spectral radius of $J_\star$ is below one, then for any γ between the spectral radius and one, $\left\|J_\star^k\right\|_2 = O(\gamma^k)$, and the same argument gives $\|G_{\mathrm{imp}} - G_b\|_2 = O(\|B_\star\|_2 \, \|g\|_2 \, \gamma^b)$. Direct paths remain essential: $B_\star$ includes input/prelude injection, and c includes the output head. Detaching prefix KV removes further dependencies and therefore changes the estimator.

Finite-trajectory BPTT and TBPTT. Along $z^{r+1} = F_\theta(z^r; x)$, let $J_r = \partial_z F_\theta(z^r; x)$ and $B_r = \partial_\theta F_\theta(z^r; x)$. The recursion $\partial z^{r+1}/\partial\theta = J_r \, \partial z^r/\partial\theta + B_r$ from a detached window start z^{R-b} gives $\partial z^R/\partial\theta = \tilde{D}_b$, the TBPTT sensitivity of Equation (1) with $i = R - 1 - k$:

$$\tilde{D}_b = \sum_{i=R-b}^{R-1} \left(J_{R-1} J_{R-2} \cdots J_{i+1}\right) B_i. \tag{17}$$

The empty product is I. Full BPTT includes all terms $i = 0, \ldots, R-1$, which give $\tilde{D}_R$, and $(J_{R-1} \cdots J_0)\,\mathrm{d}z^0/\mathrm{d}\theta$ for parameter-dependent initialization. Multiplying the transposed sensitivity by the endpoint loss gradient and adding the loss's direct parameter gradient gives the loss gradient.

Equation (17) reduces to D_b when all suffix Jacobians equal $J_\star, B_\star$ and approaches the endpoint Neumann estimate as they converge. Because early derivatives come from transient states and truncation adds its own error, neither estimator is uniformly more accurate. Controlling the error at approximate states additionally requires locally Lipschitz state and parameter Jacobians, a uniform state-Jacobian bound below one, regular loss derivatives, and small state errors throughout the window. A small endpoint residual alone is not enough.

B Additional Method Details

We specify the shared model and training configuration (Appendix B.1), explain why sampled training depths expose the model to converged context (Appendix B.2), and detail the learned depth prior of Section 4.1 (Appendix B.3), and the input-injection recipes of Section 4.2 (Appendix B.4).

B.1 Models and Training Contract

The model layout follows Section 5.1, and Table 5 gives the full configurations. All models use separate input and output embeddings. They use the Jais64k tokenizer (Sengupta et al., 2023) with a 64,256-token vocabulary and bfloat16 arithmetic. The weighted pretraining mixture covers web text, QA, code, mathematics, reasoning, planning, multilingual text, and synthetic data. Its QA component includes TxT360 (Tang et al., 2024). Architectural variants and the Dense-12 baseline have different parameter counts, so each comparison states its scale and configuration.

Each microbatch draws one training depth (Section 4.1), so a sampled-depth run draws one depth per gradient-accumulation step. Compute depends on the realized depths and backward truncation.

Learning-rate selection. Each scale has one learning-rate range, and every run tunes its peak rate within it: $[0.5, 4] \times 10^{-3}$ at S, $[0.25, 4] \times 10^{-3}$ at M, and $[0.125, 2] \times 10^{-3}$ at L. Screens keep the final WSD horizon but stop after 512–1,024 updates at S, 2,048–12,288 at M, and 8,192–32,768 at L, and select by native validation CE at $R = 5$ on 304 frozen 1,536-token sequences, distinct from MixVal. Every selected rate is interior to its run's grid. A variant that changes only λ_H or the depth-draw correlation reuses the rate of its screened configuration.

B.2 Why Sample Training Depths?

Let $p(R)$ be the training-depth distribution and ℓ_t the next-token CE at position t. Sampling one sequence-wide horizon optimizes $\mathcal{L}_{\text{depth}}$ of Equation (12) with $\ell_\theta(z^R, y) = N^{-1} \sum_t \ell_t$ (Geiping et al., 2025). Fix parameters, input, initialization, and a trajectory whose prefix KV is converged from depth r_c. Backpropagating through the final $1 \le b(R) \le R$ steps gives the expected number of backpropagated updates that read the converged prefix KV, which we call its exposure,

$$w_{r_c} = \sum_R p(R) \min\{b(R), (R - r_c)_+\}, \qquad (\cdot)_+ = \max(\cdot, 0). \tag{18}$$

The window $R - b(R) + 1, \dots, R$ intersects updates after r_c in $\min\{b(R), (R - r_c)_+\}$ steps. Equation (18) gives $w_{r_c} = 0$ for fixed $R = 5$ when $r_c \ge 5$, but positive PLN-5 exposure whenever $p(R > r_c) > 0$; sufficiently deep fixed training can also provide exposure. With our rule $b(R) = \min(R, 10)$ (Appendix B.3) and the depths drawn by the Small PLN-5 run of Table 2, PLN-5 gives $w_5 = 1.10$ ($p(R > 5) = 35.4\%$) and $w_{10} = 0.15$, so exposure concentrates on prefixes that converge within about ten recurrences. The count ignores attention weights and gradient magnitudes, and Figure 1 does not give per-bank convergence depths or training-time read traces.

Random-depth loss alone does not imply KV stationarity: $z^r = (z_a, (-1)^r z_b)$ has constant loss if the head reads only z_a, while any KV that reads the second component alternates. Supervising late converged states also does not ensure that early, transient states respond well to terminal KV. Exposure is thus necessary but not sufficient for sharing, so Table 2 measures sharing directly.

B.3 Learned Depth Prior

Every learned prior uses Equation (5). Table 10's four reward arms (CE, log CE, exp CE, and inverse perplexity, $-\exp(-\text{CE})$) use $\lambda_H = 0.01$, and Table 2 reports inverse perplexity at $\lambda_H \in \{0, 0.01\}$.

Table 5: **Model configurations at three scales.** Blocks count prelude, recurrent-core, and coda blocks of the looped models; D4 and D12 stack 4 and 12 blocks. Width and FFN give hidden and feedforward dimensions, and Q and KV give query and key–value head counts. Non-emb. and Total give dense (D4) parameter counts without and with embeddings; the looped rows add Huginn's width×width input adapter. The main text rounds the scales to 100M, 400M, and 1.6B. Parameter and token counts are in billions; the peak learning rate is tuned per run (Appendix B.1).

	S	M	L
A: Architecture and attention geometry			
Blocks (prelude/core/coda)		1 / 2 / 1	
Width	1,536	3,072	6,144
FFN	4,352	8,704	17,408
Q heads	24	48	96
KV heads	6	12	24
Head dim.	64	64	64
B: Parameters (B)			
Non-emb.	0.104	0.415	1.661
Total	0.301	0.810	2.451
Looped non-emb.	0.106	0.425	1.699
Looped total	0.304	0.819	2.488
C: Batch and token budget			
Sequence length		8,192	
Global batch (sequences)		512	
Tokens per update		4,194,304	
Updates	5,120	20,480	81,920
Tokens (B)	21.47	85.90	343.60
D: Optimizer			
Optimizer		AdamW	
(β_1, β_2)		$(0.90,\ 0.98)$	
ϵ		10^{-8}	
Weight decay		0.10	
Gradient clipping		1.00	
LR schedule		WSD	
Linear warmup		first 5% of updates	
Cosine decay		last 10%, to $0.1\times$ peak	

Algorithm 1: Depth-prior learning

Require: θ_n, ϕ_n, EMA statistics at $n-1$; $\nu, \lambda_H, \lambda_m, \bar{R}, b_{\max}$

$p_{\phi_n} \leftarrow \mathrm{softmax}(\phi_n)$

for microbatch i **do**

  $r_i \sim p_{\phi_n}$; $\ell_i \leftarrow \mathrm{CE}(\theta_n, r_i)$; $u_i \leftarrow \exp(-\ell_i)$ ▷ ℓ_i detached

  accumulate CE gradients through the final $\min(r_i, b_{\max})$ recurrences

  $A_i \leftarrow s_{n-1}(u_i - \hat{u}_{n-1})$

end for

$\mathcal{J}_n \leftarrow -\widehat{\mathbb{E}}_n[\mathrm{stopgrad}(A_i)\log p_{\phi_n}(r_i)] - \lambda_H H(p_{\phi_n}) + \lambda_m(\mathbb{E}_{p_{\phi_n}}[R] - \bar{R})^2$

$\theta_{n+1} \leftarrow \mathrm{ModelStep}(\theta_n)$; $\phi_{n+1} \leftarrow \mathrm{Adam}(\phi_n, \nabla_{\phi_n}\mathcal{J}_n)$

$\mathrm{EMA}_n[\chi] \leftarrow \nu\,\mathrm{EMA}_{n-1}[\chi] + (1-\nu)\widehat{\mathbb{E}}_n[\chi],\quad \chi \in \{\ell, u, \ell^2, u^2\}$

$\sigma_{\chi,n} \leftarrow \max\left\{\sqrt{\mathrm{EMA}_n[\chi^2] - \mathrm{EMA}_n[\chi]^2}, \epsilon\right\},\quad \chi \in \{\ell, u\}$

$\hat{u}_n \leftarrow \mathrm{EMA}_n[u];\qquad s_n \leftarrow \sigma_{\ell,n}/\sigma_{u,n}$

Sampling and controller updates. The fixed PLN-5 baseline caps Equation (4) at $R \le 64$ and recalibrates μ so that $\mathbb{E}[\min(R, 64)] = 5$. The learned prior initializes from this capped PLN: $\phi_0 = \log p_{\mathrm{PLN}}$ uses 64 logits; tail mass accumulates at depth 64 and all masses are positive. Sampling and update order follow Algorithm 1.

The backpropagation cap is $b_{\max} = 10$; earlier recurrent states are detached, but gradients still reach the prelude through the input injected at each backpropagated recurrence, and the output head. Equation (5) uses $\bar{R} = 5$ and $\lambda_m = 1$; global token weighting is specified below. The prior's separate Adam optimizer uses LR 10^{-3}, betas $(0.90, 0.999)$, and zero weight decay. During a prior warmup of 32 updates, the controller collects its baseline without changing the logits. The fixed arm freezes the logits in the same sampler.

Loss aggregation and baseline. For microbatch i, detached loss sums and unmasked target counts are reduced across ranks, and dividing by the global count N_i gives ℓ_i. The within-step average of a per-microbatch statistic χ_i is

$$\widehat{\mathbb{E}}_n[\chi] = \frac{\sum_i N_i \chi_i}{\sum_i N_i}. \tag{19}$$

For raw-CE control, with ℓ the globally reduced mean CE at sampled depth r, $\hat{\ell}_n$ the pre-step baseline, and $\bar{R}$ the target mean, the controller minimizes

$$\mathcal{J}_{\mathrm{CE}}(\phi) = \widehat{\mathbb{E}}_n\left[\left(\mathrm{stopgrad}(\ell) - \hat{\ell}_n\right)\log p_\phi(r)\right] - \lambda_H H(p_\phi) + \lambda_m\left(\mathbb{E}_{p_\phi}[R] - \bar{R}\right)^2. \tag{20}$$

Both regularizers are evaluated analytically over active depths and differentiated only through the logits. The objective and baseline share this weighting. The first weighted CE initializes the baseline, and its EMA also runs during the prior warmup:

$$\hat{\ell}_{n+1} = \nu\hat{\ell}_n + (1-\nu)\widehat{\mathbb{E}}_n[\mathrm{stopgrad}(\ell)]\,, \tag{21}$$

where $\nu \in [0, 1)$ is the EMA decay, set to 0.95.

Alternative prior signals. Three PLN-initialized arms replace raw CE by the rewards $\omega(\ell) = -\log(\max(\ell, 10^{-12}))$, $-\exp(\ell)$, or $\exp(-\ell)$ in Equation (5), computed after global token-weighted reduction. Their advantage is $A_i = s_{n-1}(u_i - \hat{u}_{n-1})$ with $u_i = \omega(\ell_i)$, where s is the ratio of previous CE and reward standard deviations, each floored at 10^{-12}; it starts at one. These arms retain the model CE objective and mean penalty, adding no forward passes.

Reward form and entropy. Table 10 names each arm by its cost $-\omega(\ell)$. Centering, scaling, and optimization order follow Algorithm 1. The entropy in Section 4.1 has logit derivative $\partial H/\partial\phi(r) = -p_\phi(r)[\log p_\phi(r) + H(p_\phi)]$, requiring no model forward pass. It does not constrain tail mass at a particular depth; Appendix C.1 evaluates its coefficient.

Uniform controls. $U_{1:64}$ initializes all logits equally (mean 32.50) and retains full support. $U_{1:10}$ restricts uniform initialization and normalization to depths 1–10 (mean 5.50), keeping depths 11–64 inactive. Both target a mean depth of five and use raw-CE feedback with the PLN arm's model and prior optimizers. The restricted control changes both initialization and support. Within the training budget, the mean penalty does not pull $U_{1:64}$ down to five: its final prior mean is 6.15 (Table 10).

Compute accounting. Backward-window visits add backward and recomputation work. Model-FLOP totals count each logical update once and exclude recomputation, controller work, and repeated work after restarts. Learning rates, schedules, and batch layouts follow Appendix B.1.

B.4 Architecture Variants and Initialization

Table 6 defines the recipes of Section 4.2; OrthoInj's anchor is the vector it projects against.

Input source and retention. DEQ-QKV injects the prelude output, which stays fixed across recurrences, into Q/K/V through separate bias-free, Gaussian-initialized projections in the two physical recurrent layers, shared across visits. It replaces the injection at the core entrance and the diagonal retention of earlier states, and keeps the prelude and the Gaussian initialization of the recurrent state.

Huginn-Linear uses a learned concatenation projection (the appended 1 gives its bias) followed by a parameter-free RMSNorm after the full core:

$$\zeta_t^r = W_{\mathrm{lin}}[h_t^r; e_t; 1], \qquad \mathbf{H}^{r+1} = \mathrm{RMSNorm}_{\mathrm{exit}}[\Gamma_\theta(\zeta_1^r, \ldots, \zeta_N^r)]\,. \tag{22}$$

Table 6: **Input-injection recipes.** $e = P_\theta(x)$ is the prelude output, $\tilde{e}$ is the injected input, and RMSNorm uses learned gains. DEQ-QKV injects at internal sites (superscript (j) indexes a recurrent layer); the others inject at the core entrance. $\Lambda = \mathrm{diag}(\exp(-\Delta \odot \alpha))$ with $\Delta = \mathrm{softplus}(\Delta_{\mathrm{raw}})$ and $\alpha = \exp(\alpha_{\mathrm{raw}})$, and OrthoInj applies Q_q of Equation (7) with $q = \Delta \odot W\tilde{e}$ and $\epsilon = 10^{-6}$.

Injection recipe	Injected input $\tilde{e}$	Input to recurrent core
DEQ-QKV	e	$W^{(j)}_{\mathrm{QKV}} h^{r,(j)} + W^{(j)}_e \tilde{e}$
Huginn-Linear (Geiping et al., 2025)	e	$W_{\mathrm{lin}}[h^r; \tilde{e}; 1]$
Parcae-Decay (Prairie et al., 2026)	$\mathrm{RMSNorm}(e)$	$\Lambda h^r + \Delta \odot W\tilde{e}$
Parcae w/o norm	e	$\Lambda h^r + \Delta \odot W\tilde{e}$
OrthoInj (ours)	e	$Q_q \Lambda h^r + q$

The diagonal injection recipes use

$$\Delta = \mathrm{softplus}(\Delta_{\mathrm{raw}}), \qquad \alpha = \exp(\alpha_{\mathrm{raw}}), \qquad \Lambda = \mathrm{diag}(\exp(-\Delta \odot \alpha)),$$
$$q_t = \Delta \odot W\tilde{e}_t, \qquad \zeta^r_t = \Lambda h^r_t + q_t. \tag{23}$$

Here $\Delta_{\mathrm{raw}}, \alpha_{\mathrm{raw}} \in \mathbb{R}^d$ are shared across visits, $\odot$ denotes channelwise multiplication, $W \in \mathbb{R}^{d\times d}$ is a full, bias-free input projection, and q_t is the injected vector of Section 4.2; the core update $\mathbf{H}^{r+1} = \Gamma_\theta(\zeta^r_1, \ldots, \zeta^r_N)$ follows Equation (6). With the prelude output $e = P_\theta(x)$ of Section 2, Parcae-Decay uses $\tilde{e} = \mathrm{RMSNorm}(e)$ with learned gains, $W_{\mathrm{init}} = I$, $\alpha_{\mathrm{init}} = 1$, and $\Delta_{\mathrm{init}} = \frac{1}{2}\log 5$ (implemented by inverse softplus). This gives retention $1/\sqrt{5}$, matching the intended linear-injection scale (Sandy Research, 2026); prelude normalization addresses large-output instability (Prairie et al., 2026, Appendix J). Parcae w/o norm uses $\tilde{e} = e$ and $\Delta_{\mathrm{init}} = 1$, retaining $W_{\mathrm{init}} = I$ and $\alpha_{\mathrm{init}} = 1$; it changes both normalization and initialization. OrthoInj uses $\tilde{e} = e$ without prelude RMSNorm and the Parcae-Decay initialization ($W_{\mathrm{init}} = I$, $\alpha_{\mathrm{init}} = 1$, $\Delta_{\mathrm{init}} = \frac{1}{2}\log 5$). It projects the carryover against the injected vector q as in Section 4.2, with FP32 projection inner products and $\epsilon = 10^{-6}$, and retains all block RMSNorms. An earlier OrthoInj variant, used only for the M learning-rate screen (Appendix B.1), uses $\Delta_{\mathrm{init}} = 1$ and the raw input e as its anchor.

Linear injection and recurrent normalization. The original Huginn normalizes before attention and FFN, after residual additions, and at the core exit (Geiping et al., 2025). We use PreNorm blocks; Huginn-Linear adds the parameter-free core-exit RMSNorm in Equation (22).

Without exit normalization, S PLN-25 training reached a nonfinite loss at all five tested learning rates. Replaying a fixed checkpoint traced the amplification to the injection interface; adding the exit RMSNorm kept the same inputs finite through recurrence 64. This replay only diagnoses forward stability; every evaluated Huginn-Linear model is trained from scratch with the exit norm.

C Additional Experimental Results

We report supplementary results and their settings; Appendix D gives the shared protocols.

C.1 Depth-Prior Learning and Test-Time Scaling

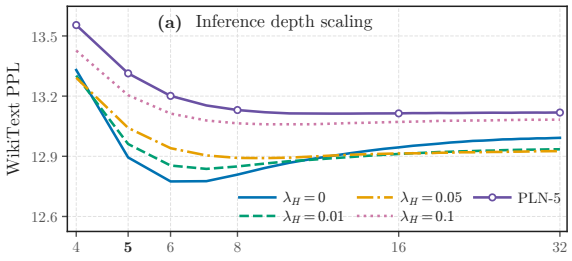
Training and evaluation. Every depth-prior run uses Parcae w/o norm injection (Table 6). Every sampled-depth row of Table 2 draws one depth per microbatch, independently across microbatches and shared across ranks. Runs use one training seed and the setup of Appendix B.1.

Native MixVal and CacheEval follow Appendix D.1 and Appendix D.4 at depth five. Sharing penalties are within-checkpoint ratios $100(\mathrm{PPL}_{\mathrm{terminal}}/\mathrm{PPL}_{\mathrm{FullKV}} - 1)$, not differences from native MixVal. Model-FLOP accounting follows Appendix B.3. For Fixed PLN-5, the trainer logs a constant nominal cost per update, so Tables 2 and 10 instead apply the learned priors' per-depth cost to the depths each run drew. Every column of Table 2 except FLOPs follows the terminal-KV protocol of Appendix D.3, and Table 7 evaluates the same checkpoints without KV reuse. The task AVG excludes the generation scores (GSM8K, DROP, and MBPP+).

Table 7: **Table 2 without KV reuse.** Same checkpoints as Table 2, evaluated at $R = 5$. Val. and Wiki are native PPL over full sequences, choice tasks are scored without a cache, and generation keeps all twelve visit-specific KV banks. D12 is the untied dense baseline with twelve distinct blocks (0.31B, 1.25B and 4.98B non-embedding parameters at S, M and L, three times the looped models) and the same twelve-bank KV cache; it is scored with the same native protocol and not bolded.

Training	PPL ↓		Downstream (%) ↑								Generation (%) ↑		
Prior	Val.	Wiki	LAMB.	HS	PIQA	ARC-E	ARC-C	OBQA	SciQ	AVG	GSM8K	DROP	MBPP+
S scale: (100M, 21.5B tokens)													
Fixed R=5	**6.50**	**20.62**	**27.79**	**32.57**	62.19	47.56	25.77	15.20	78.80	41.41	2.05	**15.28**	0.00
Fixed PLN-5	6.69	21.47	26.04	32.14	62.13	**47.94**	24.23	15.60	81.20	41.33	**2.27**	12.39	**0.62**
Learned, λ_H=0	6.60	21.14	26.94	32.09	62.19	45.92	24.66	**18.00**	**81.60**	41.63	1.90	12.72	0.25
Learned, λ_H=0.01	6.60	21.12	27.05	32.11	**62.84**	45.45	**26.02**	17.40	81.30	**41.74**	2.12	13.32	**0.62**
D12	5.80	18.20	31.67	34.78	64.80	50.93	26.02	15.80	82.30	43.76	1.59	12.50	2.75
M scale: (400M, 85.9B tokens)													
Fixed R=5	**4.51**	**12.74**	**40.19**	**46.04**	67.41	57.24	**30.55**	**19.00**	**90.20**	**50.09**	**15.54**	**21.85**	13.12
Fixed PLN-5	4.66	13.31	39.22	44.01	67.36	55.51	29.61	17.20	88.60	48.79	12.89	17.15	9.12
Learned, λ_H=0	4.55	12.89	39.08	45.30	**67.68**	**57.32**	28.75	18.20	88.40	49.25	14.40	19.35	**13.62**
Learned, λ_H=0.01	4.57	12.96	38.48	45.30	67.63	56.86	28.58	16.60	87.40	48.69	14.10	17.72	11.75
D12	4.18	11.67	42.11	49.25	70.13	60.14	30.80	19.20	90.50	51.73	15.92	21.14	16.00
L scale: (1.6B, 343.6B tokens)													
Fixed R=5	**3.51**	**8.99**	54.55	61.68	73.39	69.53	**40.27**	**25.80**	**94.10**	59.90	**50.64**	**36.10**	27.12
Fixed PLN-5	3.58	9.23	54.65	60.29	73.56	69.02	37.71	24.40	93.00	58.95	44.12	33.89	24.50
Learned, λ_H=0	3.53	9.05	**55.42**	**61.85**	72.91	70.08	**40.27**	23.00	93.10	59.52	47.23	35.20	27.38
Learned, λ_H=0.01	3.54	9.11	55.27	61.71	**73.72**	**71.55**	40.02	25.40	93.40	**60.15**	50.27	35.01	**28.88**
D12	3.34	8.36	57.73	64.59	74.37	72.31	42.32	25.00	94.50	61.55	49.58	39.18	30.88

Matched M entropy-coefficient ablation. Figure 5 varies $\lambda_H \in \{0, 0.01, 0.05, 0.1\}$ at M scale. As λ_H rises, the mean depth stays near five while $p_\phi(5)$ falls from 67.4% to 14.0% (Table 9). WikiText PPL at $R = 5$ rises from 12.89 to 13.20, and at $R = 32$, $\lambda_H = 0.01$ and 0.05 tie for the lowest PPL (12.93; panel (a)). Panel (b) isolates the cost of KV sharing by scoring each checkpoint with per-visit and shared KV. Sharing raises the continuation PPL on WikiText (after a 1,024-token prefill, Appendix D.2) of fixed-depth training from 12.19 to 16.71, whereas PLN-5 and all learned priors retain or slightly improve it. On WikiText and LAMBADA, higher entropy does not monotonically improve sharing. Without KV reuse (Table 7), $\lambda_H = 0$ gives slightly lower validation PPL than $\lambda_H = 0.01$ at M and L (4.55 vs. 4.57 and 3.53 vs. 3.54) and ties it at S, but with terminal KV sharing it falls behind at L (Table 2): GSM8K 36.47 vs. 47.92.



(b) KV cache sharing (PPL ↓)

Training	WikiText		MixVal		LAMBADA	
Prior	Per-visit	Shared	Per-visit	Shared	Per-visit	Shared
M scale: (400M, 85.9B tokens)						
Fixed $R = 5$	12.19	16.71	4.40	5.76	19.82	20.68
PLN-5	12.75	12.60	4.57	4.56	21.89	21.82
$\lambda_H = 0$	12.34	12.34	4.45	4.49	21.36	21.33
$\lambda_H = 0.01$	12.42	12.37	4.46	4.48	22.92	22.93
$\lambda_H = 0.05$	12.48	12.37	4.50	4.49	22.06	22.05
$\lambda_H = 0.1$	12.64	12.53	4.54	4.53	22.27	22.25

Figure 5: **Entropy trades PPL at the training depth for PPL at larger depths; learned priors keep KV sharing.** M scale. (a) WikiText-103 PPL across test-time depths; the bold tick marks the training mean $R = 5$. (b) PPL with per-visit KV (12 banks) and shared KV (4 banks). Panel (b) scores continuations or last words, so its values differ from panel (a).

These M runs differ only in λ_H: they share model, data, seed, learning-rate schedule, and the controller of Appendix B.3 (prior LR 0.001, $\lambda_m = 1$).

Paired readouts follow Appendix D.2 and Appendix D.4; paired LAMBADA last-word PPL averages over the 5,153 words and includes the prefill prediction. Native seven-task averages, with WinoGrande instead of SciQ (Appendix D.2), stay within 43.7–44.0 across λ_H.

Paired task accuracy with and without KV sharing. Table 8 reports paired FullKV and terminal task scoring for the learned priors at $R_{\text{pre}} = R_{\text{dec}} = 5$. Sharing does not lower the average of any learned prior (AVG Δ from $+0.25$ to $+0.70$ points), whereas Fixed $R = 5$ loses 0.61 points (FullKV and shared AVG 44.73 and 44.12); PLN-5 gives 44.30 and 44.44. Both layouts score identical requests in BF16 without FA3, and each answer option starts from a fresh cache.

Accuracy intervals use $2{,}000$ paired bootstrap draws per task, keeping task weights equal.

Table 8: **Terminal sharing does not lower the task accuracy of any learned prior.** M entropy sweep at $R_{\text{pre}} = R_{\text{dec}} = 5$. Per-task columns give Shared minus FullKV accuracy in points; the interval is a paired, task-stratified 95% bootstrap interval.

Prior	Shared − FullKV (points)							AVG (%)			
λ_H	LAMB.	HS	PIQA	ARC-E	ARC-C	WG	OBQA	FullKV	Shared	Δ	95% interval
M scale: (400M, 85.9B tokens)											
0	+0.06	+0.36	+1.64	+0.76	0.00	+1.90	+0.20	43.97	44.67	+0.70	[+0.31, +1.11]
0.01	+0.02	+0.75	−0.33	+0.93	−0.51	+0.47	+0.40	43.68	43.93	+0.25	[−0.14, +0.62]
0.05	+0.02	+0.69	+0.65	+0.84	+0.09	+0.87	0.00	43.77	44.22	+0.45	[+0.10, +0.80]
0.1	+0.04	+0.50	+0.82	+0.71	+0.26	+0.32	+0.60	43.96	44.42	+0.46	[+0.13, +0.80]

Table 9 records final priors and realized means. The CacheEval sharing penalty decreases monotonically with entropy, from $+0.96\%$ to -0.22%, and absolute shared MixVal PPL is lowest at $\lambda_H = 0.01$ (4.48, within 0.01 of $\lambda_H = 0$ and 0.05; Figure 5b). Intervals use $2{,}000$ paired sequence-CE bootstrap draws within each source, transformed by $100(\exp(\Delta\text{CE}) - 1)$; they quantify evaluation uncertainty, not training-seed variability or between-coefficient differences.

Table 9: **Higher entropy spreads the learned prior and lowers the sharing penalty.** Final priors of the M entropy sweep. $\widehat{\mathbb{E}}[R]$ is the mean realized draw and $H(p_\phi)$ is in nats; probabilities and penalties are percentages, with paired 95% bootstrap intervals.

Prior	Final depth distribution						Sharing penalty (%)	
λ_H	$\mathbb{E}_{p_\phi}[R]$	$\widehat{\mathbb{E}}[R]$	$H(p_\phi)$	$p_\phi(5)$	$p_\phi(R > 10)$	$p_\phi(R > 16)$	Penalty	95% interval
M scale: (400M, 85.9B tokens)								
0	5.007	5.014	1.067	67.40	0.32	0.010	+0.955	[+0.786, +1.136]
0.01	5.008	5.033	1.592	37.79	0.79	0.042	+0.340	[+0.189, +0.503]
0.05	5.009	5.040	2.091	19.32	2.69	0.156	−0.030	[−0.159, +0.101]
0.1	5.016	5.036	2.293	13.98	5.02	0.488	−0.224	[−0.342, −0.108]

Training trajectories and depth distributions. Figure 6 plots the training loss of the S and M checkpoints in Table 2. Figures 3 and 7 use the S/M checkpoints of Table 2: fixed PLN-5 with independent draws and learned priors with $\lambda_H = 0.01$ (LR 0.002 at S, 0.0015 at M). The learned M checkpoint is the $\lambda_H = 0.01$ curve of Figure 5, whose PLN-5 reference is also the Table 2 run.

Reward and initialization ablations. Table 10 differs from Table 2 in two ways: its S control is an earlier Fixed PLN run with independent draws, and Val. is native PPL on the LR-selection set (Appendix B.1) rather than terminal-KV PPL on MIXVAL. Four-task scores use summed answer-token likelihood on full splits (Appendix D.2).

Cached PPL uses four terminal banks and the common CacheEval protocol at $R_{\text{pre}} = R_{\text{dec}} = 5$. At M, the fixed PLN arm uses antithetic draws and the learned priors use independent draws; the arms otherwise share budget, model, data, seed, and optimizer.

With model CE training unchanged, inverse perplexity gives the best cached PPL and task mean among the S learned-PLN arms, while raw CE and inverse perplexity tie on native and WikiText PPL (within 0.02). At M, raw CE leads task accuracy and inverse perplexity leads WikiText and cached PPL, each costing about 1% more FLOPs than fixed PLN. Log and exponential CE have higher S perplexity than either. $U_{1:10}$ leads S task accuracy but has higher PPL; it also changes both initialization and support. $U_{1:64}$ increases compute and worsens validation and cached PPL. These controls match tokens, not compute.

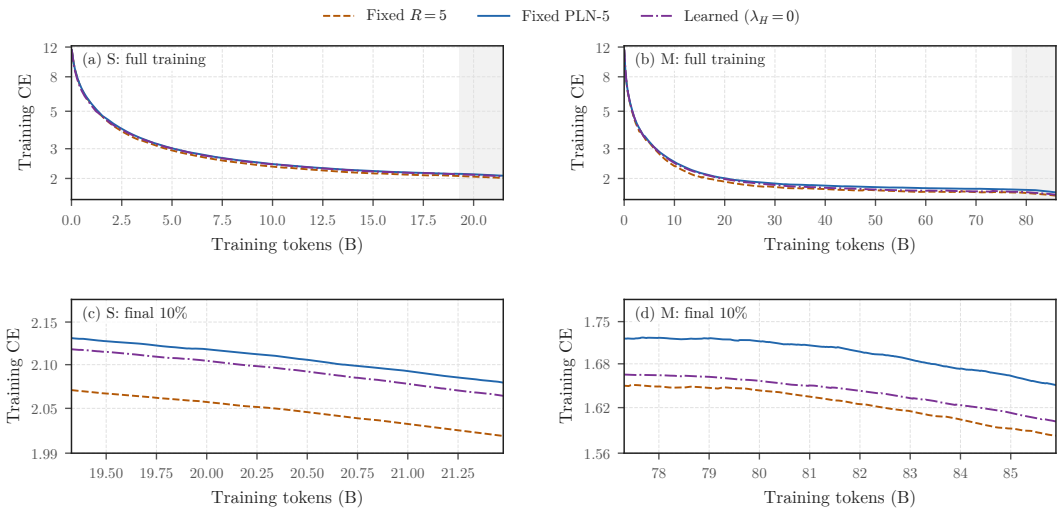


Figure 6: **Training loss for the S/M checkpoints in Table 2.** Fixed PLN-5 draws depths independently. S and M complete 21.5B and 85.9B tokens, respectively; learned priors use $\lambda_H = 0$. Curves show debiased exponential moving averages of optimizer-step CE (decay 0.999), with linear token axes and logarithmic CE axes; the lower panels expand the shaded final 10%. CE is measured at $R = 5$ for fixed-depth training and averaged over sampled depths for the priors. Learning rates and batch layouts follow Appendix B.1.

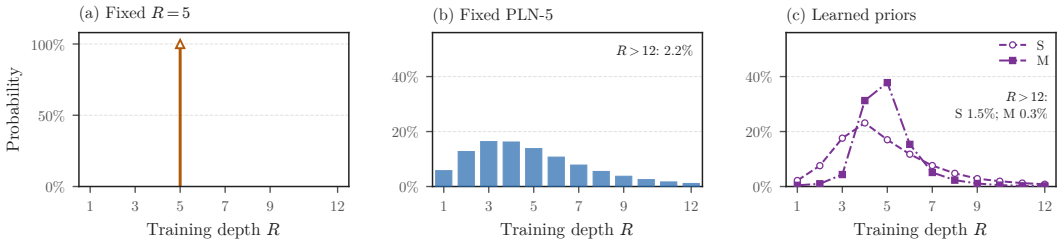


Figure 7: **Depth distributions for the scaling comparison.** Fixed $R = 5$, fixed PLN-5, and learned S/M priors ($\lambda_H = 0.01$) of Figure 3. Depths 1–12 are shown without renormalization; annotations give the remaining mass through depth 64. Panels (b,c) share a probability scale.

C.2 Input Injection

Models, optimization, batch layouts, and token budgets follow Appendix B.1, and Appendix B.1 gives the learning-rate grids. Equal token budgets need not mean equal compute.

Injection site and KV sharing. DEQ-QKV injects the input into queries, keys, and values. It breaks terminal KV sharing: decoding CacheEval one token at a time (Appendix D.4), its PPL is 16.52, against 6.43–6.69 for the other S schemes and 7.13 on MixVal without KV reuse (Table 11). At S, generation is near the floor for every scheme (GSM8K at most 2.05%, MBPP+ at most 1.62%); at M, OrthoInj leads all three generation tasks (Table 3).

Initialization and evaluation. Native validation CE at S selects the standard OrthoInj initialization of Appendix B.4. M OrthoInj uses the same initialization and one complete 20,480-update checkpoint for every score. Table 3 follows the terminal-KV protocol of Appendix D.3, and Table 11 its no-reuse protocol; generation is excluded from AVG. We keep weight decay at 0.1: in a single-seed S Parcae-Decay control (LR 0.0015, 5,120 updates), zero decay worsens validation and cached PPL from 6.63 and 6.45 to 6.76 and 6.58.

Table 10: **Reward and initialization ablations for the learned prior.** Rows name the reward signal. Learned rows use $\lambda_H = 0.01$ and start from capped PLN-5 ($U_{1:k}$: uniform over depths $1, \ldots, k$); the M Fixed PLN row uses antithetic draws. FLOPs are in 10^{19} and $\mathbb{E}_{p_\phi}[R]$ is the final prior mean. Val. is PPL on the LR-selection set; AVG averages four raw-likelihood tasks.

Training			PPL ↓			Downstream (%) ↑				
Prior	$\mathbb{E}_{p_\phi}[R]$	FLOPs	Val.	Wiki	Cached	HS	ARC-C	ARC-E	PIQA	AVG
S scale: (100M, 21.5B tokens)										
Fixed PLN	5.00	7.31	6.86	21.34	6.52	29.60	22.15	47.06	61.15	39.99
CE	5.00	7.33	**6.79**	**21.17**	6.47	29.46	21.80	46.89	61.81	39.99
$\log(\mathrm{CE})$	5.01	7.34	6.85	21.60	6.51	**29.69**	21.20	46.81	61.75	39.86
$\exp(\mathrm{CE})$	5.01	7.33	6.85	21.59	6.50	29.67	21.03	46.51	61.86	39.77
$-\exp(-\mathrm{CE})$	5.00	7.33	6.80	21.19	**6.46**	29.50	21.72	47.44	**62.35**	40.25
CE ($U_{1:10}$)	5.01	7.37	6.87	21.58	6.52	29.67	**22.40**	**47.65**	61.92	**40.41**
CE ($U_{1:64}$)	6.15	11.53	6.94	21.40	6.57	29.42	21.55	46.98	62.24	40.04
M scale: (400M, 85.9B tokens)										
Fixed PLN	5.00	91.01	4.72	13.16	4.49	36.15	25.49	58.86	68.66	47.29
CE	5.01	91.99	4.64	12.85	4.47	**36.98**	**28.24**	58.22	**69.64**	**48.27**
$-\exp(-\mathrm{CE})$	5.01	91.95	**4.63**	**12.77**	**4.46**	36.76	26.87	58.14	69.37	47.78
CE ($U_{1:10}$)	5.01	92.31	4.64	12.91	4.48	36.89	27.73	**59.28**	68.72	48.15

Table 11: **Table 3 without KV reuse.** Same checkpoints as Table 3, evaluated at $R = 5$. Val. and Wiki are native PPL over full sequences, choice tasks are scored without a cache, and generation keeps all twelve visit-specific KV banks.

Model	PPL ↓		Downstream (%) ↑								Generation (%) ↑		
Injection	Val.	Wiki	LAMB.	HS	PIQA	ARC-E	ARC-C	OBQA	SciQ	AVG	GSM8K	DROP	MBPP+
S scale: (100M, 21.5B tokens)													
DEQ-QKV	7.13	23.83	25.29	30.65	60.99	44.74	24.32	14.20	78.60	39.83	1.36	**14.61**	0.50
Huginn-Linear	6.89	22.23	25.91	31.40	**62.19**	43.35	24.06	**17.00**	79.10	40.43	1.67	11.84	0.50
Parcae-Decay	6.71	21.24	26.02	32.20	61.75	**47.26**	24.40	16.40	81.40	41.35	**2.35**	10.94	0.62
OrthoInj	**6.60**	**21.16**	**27.30**	**32.38**	61.92	45.45	**25.34**	16.60	**82.10**	**41.59**	1.36	13.34	**1.00**
M scale: (400M, 85.9B tokens)													
Huginn-Linear	4.65	13.33	36.33	43.90	67.85	56.94	**29.18**	17.40	**89.10**	48.67	**15.09**	16.82	8.38
Parcae-Decay	4.62	13.20	37.28	44.37	**68.06**	57.62	27.82	**19.20**	88.10	**48.92**	12.59	19.44	9.00
OrthoInj	**4.59**	**13.02**	**37.90**	**45.12**	67.63	**57.79**	27.99	17.60	88.20	48.89	14.25	**20.59**	**10.12**

From Parcae-Decay to OrthoInj. OrthoInj differs from Parcae-Decay by two switches: it projects the carryover against the injected vector q, and it drops prelude RMSNorm. Table 12 crosses the two switches at S. Each switch lowers validation PPL on its own: the projection from 6.71 to 6.67, and dropping prelude RMSNorm from 6.71 to 6.63. Their gains add up: the projection lowers PPL by a similar amount with and without prelude RMSNorm (0.04 and 0.03), and OrthoInj, which has both switches, reaches 6.60. OrthoInj also has the highest downstream average; the other three cells lie within 0.03 points of each other.

All four cells train with PLN-5 and independent depth draws, seed 1, common data, optimizer, and batch settings, $\Delta_{\text{init}} = \frac{1}{2}\log 5$, and 5,120 updates; transformer-block RMSNorms are retained throughout. Parcae-Decay and OrthoInj are the S models of Table 3. Each cell uses the LR selected by its own staged screen. The cell without either switch was screened with independent draws over $\{1, 1.5, 2, 3\} \times 10^{-3}$ and selects 2×10^{-3}; the other three screens used antithetic draws. Setting $\Delta_{\text{init}} = 1$ instead changes selection PPL by at most 0.03 (6.87 vs. 6.84 with prelude RMSNorm and the projection; 6.77 vs. 6.78 for OrthoInj).

Figure 8 shows the late training loss of the four cells. Over the final 256 updates, dropping prelude RMSNorm lowers mean training CE by 0.010 with or without the projection, and the projection lowers it by a further 0.002–0.003. These single-seed comparisons do not estimate seed variance.

C.3 Truncation, Depth Sampling, and Fixed Points

Cost. Training memory grows linearly with the window: peak allocation is 48, 86 and 124 GB for $b = 5, 10, 15$ at local batch 8. Full BPTT under PLN-5 must also hold the deepest draw; a depth-30

Table 12: **Dropping prelude RMSNorm and projecting the carryover each lower perplexity, and their gains add up.** S models trained with PLN-5 and evaluated at $R = 5$ without KV reuse. Norm is prelude RMSNorm and Proj. the carryover projection; Sel. is PPL on the LR-selection set; AVG is the seven-task mean of Table 11.

Training				PPL ↓			Down. (%) ↑
Recipe	Norm	Proj.	LR	Sel.	Val.	Wiki	AVG
S scale: (100M, 21.5B tokens)							
Parcae-Decay	✓		0.0015	6.86	6.71	21.24	41.35
+ projection	✓	✓	0.0015	6.84	6.67	21.26	41.37
− prelude norm			0.002	6.79	6.63	**21.14**	41.34
OrthoInj (−norm +proj.)		✓	0.002	**6.78**	**6.60**	21.16	**41.59**

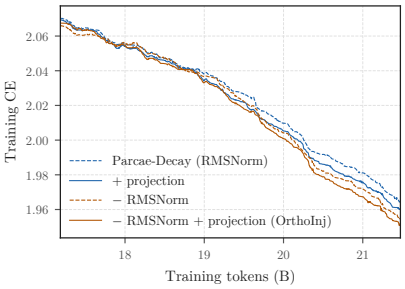


Figure 8: **Both switches lower late-training loss.** Training CE of the four cells of Table 12 over the final 20% of training, as a 512-step rolling median. All cells see the same sampled depths.

draw at local batch 8 exceeds device memory, and per-layer recomputation brings it to 28 GiB at about 1.3× the step time of $b = 10$. At a fixed depth $R = 5$, $b = 2$ trains 1.32× faster than full BPTT. In a Small PLN-25 model at $R = 16$, a window $b = 10$ lowers saved activations from 3.95 to 2.63 GiB, which stay constant for fixed b once $R \geq b$, while the core gradient keeps a median cosine similarity of 0.978 with full BPTT.

Quality with depth sampling. Table 13 varies only the backward window under PLN-5. Every window yields a flat depth curve and lossless terminal-KV sharing, and the target-depth perplexity stays within the spread between windows.

Truncation as a fixed-point shaper. Table 14 fixes the training depth, so depth sampling cannot shape fixed points. Full BPTT then degrades beyond the training depth and loses 21% perplexity under terminal-KV sharing. Backpropagating through only the last two recurrences removes both, with a random or a zero initial state, at a cost of about 0.07–0.10 CE at the training depth. The stop-gradient removes credit for iteration-specific computation in the detached prefix: the window is trained only to map whatever state the model reaches to a good prediction, which favors states that further recurrences leave unchanged. Without input injection, the prefix state is the only carrier of the input, so this pressure erases it: truncation then neither fits the training depth nor yields a fixed point. A two-term endpoint Neumann estimate, which uses only the Jacobian at the endpoint, collapses from initialization at every LR, because no fixed point exists early in training.

Relation to prior uses of truncation. Earlier looped models that extrapolate stably also truncate backpropagation, but in each case together with another source of depth variation. Progressive training detaches a prefix of random length, so the window lands at a random depth, and mixes in a full-depth loss (Bansal et al., 2022); its motivation, discouraging iteration-specific behavior, matches the mechanism above. The looped transformers of Yang et al. (2024) detach all but the last b loops, but grow the loop count by a curriculum and supervise every loop in the window; they find that larger windows help reach a fixed point. Huginn backpropagates through the last eight recurrences of a sampled depth (Geiping et al., 2025). Both Bansal et al. (2022) and Yang et al. (2024) also find input injection necessary. Table 14 removes these sources of depth variation: the depth is fixed, the window is the last two recurrences, and full BPTT is the control.

C.4 Recurrent-State and Cache Dynamics

This section gives the protocol behind the tokenwise convergence of Section 2 and measures how far hidden states and KV caches settle at larger depth.

Tokenwise dynamics of trained checkpoints. Figure 1 uses the Small Fixed PLN-5 checkpoint of Table 2 at step 5,120. We take eight frozen samples, one from each of the first eight validation sources in alphabetical order; each sample is the first 2,048 tokens of the source's first packed

Table 13: **With depth sampling, TBPTT matches full BPTT.** S models trained with PLN-5. IID rows share one LR; antithetic (Anti.) rows use a separately screened LR per window. Terminal KV gives the change in cached PPL from full to terminal KV at $R_{\text{pre}} = R_{\text{dec}} = 5$.

Training			WikiText PPL ↓		Terminal KV
Sampler	Window b	LR	R=5	R=64	Δ (%)
S scale: (100M, 21.5B tokens)					
IID	10	0.0015	21.47	21.21	−0.16
IID	full (64)	0.0015	21.15	20.94	−0.09
Anti.	5	0.0020	20.77	20.53	−0.15
Anti.	10	0.0019	20.93	20.72	−0.10
Anti.	15	0.0030	20.86	20.67	−0.09

Table 14: **At a fixed training depth, truncation shapes fixed points when the input is injected.** S models trained at fixed $R = 5$ with diagonal (Diag.) or no injection, each with its own screened LR. CE is native validation CE at $R = 5$. Without injection, the loop starts from the prelude output, with embeddings and prelude frozen. The two-term endpoint Neumann estimate collapsed at all five LRs within 1,024 steps.

Training			Val.	WikiText PPL ↓		Terminal KV
Init.	Injection	Backward	CE ↓	R=5	R=64	Δ (%)
S scale: (100M, 21.5B tokens)						
Random	Diag.	Full (b=5)	1.894	20.62	26.09	+21.9
Random	Diag.	TBPTT b=2	1.961	22.29	22.71	+2.9
Zero	Diag.	Full (b=5)	1.886	20.76	26.06	+20.7
Zero	Diag.	TBPTT b=2	1.984	23.01	23.26	+1.3
Random	Diag.	Neumann, b=2	collapsed at every LR			
Prelude	None	Full (b=5)	2.345	39.85	9349	+232
Prelude	None	TBPTT b=2	4.971	1046	37860	+1890

sequence, with document boundaries and causal masks preserved. The unsmoothed heatmap in panel (a) shows the first document, which ends at position 1,003; panel (c) pools all eight samples at document-local positions below 1,024.

State and convergence. Each continuous 64-update trajectory uses native normal initialization, executes the prelude and coda once, and generates no tokens. The relative-change diagnostic in Section 2 uses states after both core blocks and exit normalization, before the coda, with nonzero reference states and FP32 measurements on BF16 states. Under this criterion, Figure 1 sets $R_{\text{obs}} = 32$ and $\tau = 2\%$, requiring a sub-threshold suffix of at least four updates; Table 15 sets $R_{\text{obs}} = 64$. Tokens that do not meet the criterion have no observed convergence depth and leave display gaps. The logarithmic color scale is clipped to $[10^{-4}, 1]$, and the gray initialization row is unclipped.

Positional association. The zoomed panel (b) shows the latest-converging token. Panel (c) includes all qualifying positions, with medians in nonoverlapping 128-position bins: 13,707 positions qualify and two do not. For the correlation in Section 2, ties use average ranks; per-sequence values range from -0.10 to 0.47. No independent-token significance test is applied.

Table 15: **Hidden states and KV caches settle with depth.** S models trained with PLN-5 and PLN-25. **A:** a token is converged if every later update through recurrence 64 stays below the threshold for at least four updates; onset is the median over converged tokens. **B:** relative pre-RoPE KV change between recurrences $r - 5$ and r for PLN-25, with sequence-bootstrap intervals. Values are percentages except onset, which counts recurrences.

A: Adjacent hidden-state changes and converged-token coverage

Mean depth	Median $\rho_{64,t}$	Converged at 1% (%)	Converged at 2% (%)	Onset at 2%
5	0.33	99.74	99.93	7
25	0.23	97.39	99.09	11

B: KV changes over five recurrences; PLN-25

Recurrence r	Reference $r - 5$	Median KV change	95% interval	KV norm / norm at $r = 5$
25	20	1.23	[1.22, 1.24]	97.48
30	25	0.85	[0.84, 0.86]	97.48
160	155	0.59	[0.58, 0.60]	97.48

Huginn convergence measurements.

Adjacent hidden-state changes. Table 15A applies the initialization, precision and convergence protocol in Appendix C.4 through update 64 over all 16,384 positions; onset medians are computed over qualifying tokens only. The mean-5 and mean-25 models use LRs of 1.5×10^{-3} and 2×10^{-3}, respectively, and also differ in training recipe, so the two rows are not a controlled comparison of depth. Within 64 updates, more than 97% of tokens meet the 1% criterion in both models.

Five-recurrence KV changes. Panel B stacks the pre-RoPE keys and values of the banks C^r from both physical core layers into one matrix every fifth recurrence. On each of 1,216 MIXVAL sequences (Appendix D.1), we compute

$$\rho_r^C = \frac{\left\|C^r - C^{r-5}\right\|_F}{\sqrt{\frac{1}{2}(\|C^r\|_F^2 + \|C^{r-5}\|_F^2)}}. \tag{24}$$

The table reports sequence medians with pointwise 95% intervals from 10,000 sequence-bootstrap resamples; norm ratios use each sequence's $\left\|C^5\right\|_F$ as the reference. The five-recurrence KV change falls from 1.23% at $r = 25$ to 0.59% at $r = 160$, below the 2% single-update threshold of Section 2 even though it spans five recurrences, while the KV norm stays at 97.48% of its $r = 5$ value, so the remaining change occurs at nearly constant norm. Each continuous 160-recurrence trajectory uses two core-block applications per recurrence without token generation. These measurements characterize state and cache changes; they do not establish global contraction, a causal training-prior effect, or the quality of residual-based stopping.

C.5 INDEPENDENT PREFILL AND DECODE BUDGETS

Terminal KV sharing lets prefill depth R_{pre} and decode depth R_{dec} vary independently: every decoding visit to a physical attention layer reads the same terminal bank, so no decoding visit needs a matching prefill visit. R_{pre} controls only how refined this bank is, while R_{dec} controls how many recurrences each new token receives. Under the convergence conditions of Lemma 1, deeper prefill moves the bank toward its fixed point, so decoding should improve with R_{pre} and then saturate. Figure 9 confirms this for Small checkpoints trained at mean recurrence 5 and 25 once the decode depth is at least 5 (at least 2 for the mean-5 model): beyond $R_{\text{pre}} = 5$, cached log-perplexity changes by at most 0.01 at every decode depth. At the shallowest decode depths, a matched shallow prefill does better; for the mean-25 model at $R_{\text{dec}} = 2$, $R_{\text{pre}} = 1$ gives 2.89 against 3.13 for $R_{\text{pre}} \geq 5$. Decode depth matters more, especially for the mean-25 model, where raising R_{dec} from 1 to 10 lowers log-perplexity from 3.47 to 1.82 before it plateaus.



Figure 9: **Prefill gains saturate by $R_{\text{pre}} = 5$, and decode depth matters more.** Cells give the cached log-perplexity of Small checkpoints trained at mean recurrence 5 (left) and 25 (right), on CacheEval with terminal KV sharing, 1,024 prefix tokens, and continuation positions 2–512.

Both grids use checkpoints with Parcae w/o norm injection, reuse the terminal history, and keep the current token's KV local to each visit, with $2 + 2R_{\text{pre}}$ block applications per prefill token and $2 + 2R_{\text{dec}}$ per incremental input. Figure 9 evaluates $R_{\text{pre}}, R_{\text{dec}} \in \{1, 2, 5, 10, 15, 20, 25, 32\}$ on CacheEval. KV allocation stays constant across each grid's 64 settings.

C.6 Distilled Prefill

We first evaluate teacher-initialized students on the full battery of Section 5.4 (Appendix C.6.1) and then distill a deeper $R = 25$ teacher (Appendix C.6.2).

C.6.1 Teacher-Initialized Students under the Full Battery

Evaluation. Table 4 and Tables 16 and 17 use the data, requests, and scorers of Table 2. The prefill path processes the prompt, and the teacher at $R = 5$ processes everything after it while reading the four terminal KV banks the path produced. For Val. and Wiki, the path prefills the first $4{,}096$ tokens of each $8{,}192$-token sequence ($1{,}216$ and 38 sequences), and the teacher scores the remaining $4{,}096$. For the seven choice tasks, the path prefills every context token but the last, and the teacher scores the last context token and the candidate continuation. For GSM8K, DROP, and MBPP+, the path prefills the prompt and the teacher decodes with the settings of Appendix D.5. D4 and the standalone student decode themselves with their own KV. The teacher row therefore uses terminal KV throughout and equals the $\lambda_H = 0.01$ rows of Table 2; with the full cache, its downstream averages are 41.74 on Small and 48.69 on Medium (Table 7). Latencies are paired means of six warmed repetitions at batch eight, timed in the teacher's process; D4 is timed the same way in a separate job.

Teacher and student. The Small and Medium teachers in Section 5.4 are the learned-prior models of Table 2 ($\lambda_H = 0.01$) with Parcae input injection and no prelude RMSNorm. They were trained on 21.47B and 85.90B tokens with learning rates 2×10^{-3} and 1.5×10^{-3}, respectively. Targets are pre-coda states after five recurrent visits. The two-layer dense students match their teachers' widths (51.9M parameters on Small and 207.6M on Medium). Neither student shares weights between its two layers. Equation (3) uses energy normalization calibrated on 64 training rows.

Student training. At each scale, held-out hidden-state MSE on 128 prompts selects among learning rates 3×10^{-4}, 10^{-3}, 3×10^{-3}, and 6×10^{-3}. All run 128 updates, the top two continue to 512, and the selected 10^{-3} reaches $2{,}560$ updates on Small and $10{,}240$ on Medium. The cosine schedule uses eight warmup updates and reaches 10^{-4} at update $2{,}048$, retaining that rate thereafter. All stages use GBS 256 at sequence length $8{,}192$, so each final student sees one quarter of its teacher's training tokens (5.369B on Small and 21.475B on Medium); training excludes held-out rows.

Teacher initialization. Teacher-initialized students add a $d \times d$ input map to the two-block student (2.36M parameters on Small, 9.44M on Medium). It is set to $\mathrm{diag}(\Delta)W$, so it outputs the injected input q of Eq. (6), and the two blocks copy the recurrent core. Because the teacher's initial state is small (standard deviation $d^{-1/2}$ per coordinate), the untrained student approximately reproduces the teacher's state after one recurrence. They train for the same $2{,}560$ and $10{,}240$ updates at GBS 256 as the randomly initialized students, directly at learning rate 10^{-3}, the rate selected for those students. A control with the same input map, objective, and rate but random initialization (random, +KL) isolates the initialization. The KL loss is the second term of Eq. (3) and has weight one.

Results. Teacher initialization improves the prefill path (Table 16). With the hidden loss alone, it raises the Medium downstream average from 45.45 to 46.88, GSM8K from 5.53 to 8.42, and MBPP+ from 0.25 to 5.38. On Small, where generation is near zero for every model, it raises DROP from 11.51 to 14.25. With the KL term in both, it raises the downstream average from 45.66 to 46.77 on Medium and from 39.96 to 40.63 on Small, and Medium MBPP+ from 2.50 to 6.12. Adding the KL term, as in the main recipe, keeps the downstream average within 0.1 point and raises all three generation scores on both scales; on Medium, GSM8K rises from 8.42 to 8.95, DROP from 15.21 to 16.20, and MBPP+ from 5.38 to 6.12. Feeding the student's own KV to the teacher (KV direct) costs 2–7 downstream points relative to +1 rec. Running the student without the teacher (standalone) depends most on the KL loss: it raises the Medium average from 40.64 to 43.75 and lowers Val. PPL from 7.08 to 5.88. Even then, neither use beats D4, which has the same physical depth.

C.6.2 A Deeper Teacher at $R = 25$

Distilling a deeper endpoint should save more prefill time. We trained Small and Medium learned-prior teachers ($\lambda_H = 0.01$) whose PLN prior has mean 25, on the same 21.47B and 85.90B tokens

Table 16: **Teacher initialization helps the prefill path, and the KL loss raises its generation scores.** Same battery as Table 4, $R = 5$. *+1 rec.* applies one teacher recurrence to the student state (our path; the teacher-init +KL row is the main recipe), *KV direct* feeds the student's own KV to the teacher, and *Standalone* decodes with the student alone. +KL adds the second term of Eq. (3).

Prefill path			PPL ↓		Downstream (%) ↑		Generation (%) ↑			Latency ↓
Use	Init	Loss	Val.	Wiki	LAMB.	AVG	GSM	DROP	MBPP+	8K (ms)
S scale: (100M, 21.5B tokens)										
Teacher			6.67	19.91	27.36	41.65	1.59	14.08	0.62	195.0
D4			7.94	25.19	23.95	39.45	1.97	11.39	0.12	100.7
+1 rec.	random	hidden	6.73	20.42	24.24	40.44	1.90	11.51	0.12	125.4
	random	+KL	6.72	20.32	20.88	39.96	1.52	12.44	0.00	125.3
	teacher	hidden	6.70	20.24	24.63	40.73	0.83	14.25	0.38	126.3
	teacher	+KL	6.70	20.16	23.56	40.63	1.74	14.41	0.62	125.9
KV direct	teacher	hidden	6.88	21.36	18.86	38.68	1.29	10.44	0.00	105.1
	teacher	+KL	6.83	21.08	18.96	38.71	1.14	11.28	0.00	104.9
Standalone	teacher	hidden	9.06	30.64	19.41	38.00	2.27	9.41	0.00	100.1
	teacher	+KL	8.09	26.73	21.35	38.22	2.12	9.92	0.00	99.9
M scale: (400M, 85.9B tokens)										
Teacher			4.58	12.28	38.58	49.07	15.62	17.11	10.38	495.8
D4			5.55	15.73	31.19	44.08	2.35	13.12	2.50	229.8
+1 rec.	random	hidden	4.64	12.69	29.15	45.45	5.53	13.20	0.25	295.5
	random	+KL	4.63	12.57	29.52	45.66	7.05	15.36	2.50	295.3
	teacher	hidden	4.61	12.51	33.22	46.88	8.42	15.21	5.38	299.1
	teacher	+KL	4.60	12.44	31.86	46.77	8.95	16.20	6.12	299.3
KV direct	teacher	hidden	4.80	13.66	9.99	39.65	2.12	7.21	0.00	235.6
	teacher	+KL	5.07	14.27	15.85	40.81	1.29	9.39	0.00	235.2
Standalone	teacher	hidden	7.08	22.14	22.67	40.64	1.97	11.22	0.38	226.6
	teacher	+KL	5.88	17.56	29.40	43.75	2.58	12.47	1.00	226.0

at learning rates 2×10^{-3} and 1.5×10^{-3}, and evaluate them at $R = 25$. Their students follow the $R = 5$ recipe and budget. Table 17 places them next to the $R = 5$ results.

Table 17: **Distilling a deeper endpoint saves more prefill time but loses more quality.** Mean-25 learned-prior teachers at $R = 25$ next to the $R = 5$ teachers of Table 4, same battery. All students use the hidden loss alone at both depths, unlike the main recipe; standalone students decode themselves. The M $R = 25$ teacher-initialized rows reuse the timing of the randomly initialized student.

Prefill		PPL ↓		Downstream (%) ↑		Generation (%) ↑			Latency ↓
Teacher R	Path	Val.	Wiki	LAMB.	AVG	GSM	DROP	MBPP+	8K (ms)
S scale: (100M, 21.5B tokens)									
5	Teacher	6.67	19.91	27.36	41.65	1.59	14.08	0.62	195.0
	One recurrence	6.85	21.02	19.29	38.05	1.52	10.86	0.00	110.7
	Distilled, random init	6.73	20.42	24.24	40.44	1.90	11.51	0.12	125.4
	Distilled, teacher init	6.70	20.24	24.63	40.73	0.83	14.25	0.38	126.3
	Standalone	9.06	30.64	19.41	38.00	2.27	9.41	0.00	100.1
25	Teacher	6.35	18.59	28.82	41.66	2.73	13.99	1.00	648.4
	One recurrence	6.93	22.45	10.44	33.84	1.52	3.44	0.00	132.7
	Distilled, random init	6.48	19.46	20.20	38.64	1.67	11.80	0.00	156.1
	Distilled, teacher init	6.44	19.20	22.65	39.25	1.52	11.33	0.12	151.0
	Standalone	11.58	37.43	18.82	36.19	2.73	7.52	0.00	99.2
M scale: (400M, 85.9B tokens)									
5	Teacher	4.58	12.28	38.58	49.07	15.62	17.11	10.38	495.8
	One recurrence	4.73	13.12	13.76	41.19	2.27	8.56	0.12	245.9
	Distilled, random init	4.64	12.69	29.15	45.45	5.53	13.20	0.25	295.5
	Distilled, teacher init	4.61	12.51	33.22	46.88	8.42	15.21	5.38	299.1
	Standalone	7.08	22.14	22.67	40.64	1.97	11.22	0.38	226.6
25	Teacher	4.56	12.17	41.61	49.45	21.38	20.11	11.25	1847.7
	One recurrence	10.99	43.04	0.62	32.95	0.00	3.82	0.00	289.9
	Distilled, random init	4.70	12.92	19.87	42.09	1.74	10.30	0.00	342.3
	Distilled, teacher init	4.63	12.55	30.04	45.54	4.32	14.99	1.38	342.3
	Standalone	9.36	27.84	18.40	39.43	1.82	6.39	0.00	227.1

The savings grow with depth: at 8K, distilled prefill is $4.3\times$ faster than the $R = 25$ teacher on Small and $5.4\times$ on Medium, compared with $1.54\times$ and $1.66\times$ at $R = 5$. The quality cost grows faster. The teacher-initialized student trails its $R = 25$ teacher by 2.4 and 3.9 downstream points on Small and Medium, compared with 0.9 and 2.2 at $R = 5$, and it loses most of the deeper Medium teacher's generation gain (GSM8K 21.38 vs. 4.32). Teacher initialization matters more at this depth: on Medium it raises the downstream average from 42.09 to 45.54. A single recurrence no longer suffices: its downstream average is 33.84 on Small and 32.95 on Medium.

As a result, the $R = 25$ student is both slower and less accurate than the $R = 5$ student at each scale (151.0 vs. 126.3 ms and 39.25 vs. 40.73 on Small; 342.3 vs. 299.1 ms and 45.54 vs. 46.88 on Medium). Its lower Small PPL comes from the stronger $R = 25$ decoder, which scores all 4,096 tokens itself; relative to its own teacher, the $R = 25$ student's PPL gap is 1.7–3.4$\times$ the $R = 5$ gap on both scales. It is faster than the $R = 5$ teacher but falls well below it. Hidden-state error does not explain the gap. The depth-25 endpoint lies farther from the prelude output on both scales (copying the input leaves a relative error of 0.945 vs. 0.856 on Small and 0.997 vs. 0.942 on Medium). The Small students fit it less well (held-out relative MSE 0.070–0.100 vs. 0.058), but the Medium students fit it better (0.011–0.017 vs. 0.036) and still lose more quality. The deeper teacher's decoder appears more sensitive to small handoff errors. A stronger teacher need not give a better handoff, consistent with evidence that a larger teacher–student gap can weaken distillation (Cho & Hariharan, 2019; Mirzadeh et al., 2020). We therefore report $R = 5$ in the main text.

C.7 Reusing States during Post-Training

This section specifies the rollout-state reuse estimator of Section 3.1, gives the training protocol, and reports speed and accuracy with Dr. GRPO on GSM8K. Saved states are used only before the optimizer step changes the parameters, and missing, stale, or token-mismatched states are rejected.

Forward state and gradient estimator. For a saved endpoint z^R, token sequence x, and input-conditioned recurrent map $F_\theta(z; x)$ of Section 2, reuse constructs

$$\widetilde{z} = \mathrm{stopgrad}(z^R) + \big[F_\theta(\mathrm{stopgrad}(z^R); x) - \mathrm{stopgrad}\big(F_\theta(\mathrm{stopgrad}(z^R); x)\big)\big]. \tag{25}$$

Its forward value is z^R, and gradients flow through one application of F_θ. With $J_R = \partial_z F_\theta(z^R; x)$ as in Appendix A.2 and readout gradient g, Neumann-4 applies four VJPs from $v_0 = g$:

$$v_{k+1} = g + J_R^\top v_k, \qquad k = 0, \ldots, 3, \qquad v_4 = \sum_{j=0}^{4} (J_R^\top)^j g. \tag{26}$$

The resulting sum has five terms, including the identity; Neumann-3 stops after three products (four terms). The number in each Neumann label thus counts vector–Jacobian products. The parameter gradient combines $(\partial_\theta F_\theta)^\top v_4$ with direct readout derivatives; differentiable attention over saved states preserves cross-token derivatives. Recompute + Neumann-4 reconstructs z^R without gradients before applying this estimator; full BPTT differentiates all six recurrent visits. This endpoint Neumann estimate is approximate; Equation (25) implies no convergence guarantee. Reuse and recomputation differ only in the no-gradient forward pass that recomputation adds. At $R = 6$, the four state products of Neumann-4 are close to the six recurrent visits of full BPTT, so this experiment does not test how the saving scales with depth.

Model, data, and reward. The shared checkpoint in Section 3.1 is the Large learned-prior model of Table 2 ($\lambda_H = 0.01$, 343.6B pretraining tokens). Its input injection is Parcae w/o norm, the injection used by all Table 2 rows. Rollout and evaluation use visit-specific KV. Prompts use the eight-shot chain-of-thought format of Appendix D.5; generation stops at the next question marker, and the binary reward requires the extracted numeric answer to match the reference. Training draws questions from the 7,473-question GSM8K training split in a fixed shuffled order, excluding the eight demonstrations and two probe questions. Evaluation uses a fixed random sample of 500 test questions. We did not verify pretraining decontamination; all arms share the checkpoint, so contamination would shift absolute accuracy but not the comparison between arms.

Policy updates and sampling budget. Each arm samples 16 responses per training question once (temperature 0.8, 512-token cap), forming 200 two-question batches. Dr. GRPO centers each question's rewards within its group of 16 without standard-deviation scaling and divides the summed

token loss by the batch size times the completion cap. Updates use a clipping range of 0.2, no reference KL, the recorded rollout log-probabilities as the importance-ratio denominator, and the rollout temperature during replay. Zero-advantage batches skip the optimizer step, and the others receive one on-policy update, so arms with equal sampling budgets can differ in update and token counts.

Training uses replay microbatch two and gradient accumulation 16, so each update uses 32 responses. AdamW uses LR 10^{-6}, moments $(0.9, 0.95)$, $\epsilon = 10^{-8}$, zero weight decay, gradient-norm clipping at one, and a four-update linear warmup. Evaluation uses the same seed for every endpoint. It draws eight samples per question with the rollout sampler; pass@1 averages the eight outcomes, and pass@8 counts a question as solved if any sample is correct.

Recurrence and task choice. Two measurements favor $R = 6$ for the starting checkpoint. With 64 samples per question at temperature 0.8 on 500 GSM8K training questions, $R = 6$ reaches pass@1/8/64 of $38.1/73.8/91.4\%$, against $36.2/72.0/89.8\%$ at $R = 5$. With greedy decoding on the test split (Table 18), $R = 6$ also gives the best GSM8K accuracy, while code and reading scores decline slowly through $R = 7$ and fall from $R = 8$. This greedy accuracy (53.68 at $R = 6$) is higher than the sampled pass@1 at temperature 0.8 that Table 19 reports before RL (40.63). Groups of 16 samples contain both a correct and an incorrect response for 78% of the training questions, compared with 65% of the MBPP+ training tasks, so GSM8K gives more learning signal per rollout.

Table 18: $R = 6$ **gives the best GSM8K accuracy for the RL starting checkpoint.** L learned-prior model ($\lambda_H = 0.01$) with visit-specific KV; GSM8K decodes greedily on the full test split.

R	5	6	7	8	10	15	20
L scale: (1.6B, 343.6B tokens)							
GSM8K (greedy)	50.27	**53.68**	52.31	52.01	49.96	48.90	49.73
MBPP+ pass@1	**28.88**	28.75	27.88	25.88	22.50	20.88	20.75
DROP F1	**35.01**	34.67	33.93	32.39	30.45	30.47	29.15

Table 19: **Reusing rollout states cuts scoring and backward time.** Training uses $6{,}400$ responses to 400 GSM8K training questions; evaluation draws eight samples per question on 500 held-out test questions. Time is scoring plus backward, as the median per update (16 replay microbatches), excluding rollout and the optimizer step.

Training	GSM8K (%) ↑		Time (s) ↓
Update method	Pass@1	Pass@8	Update
L scale: (1.6B, 343.6B tokens)			
Before RL	40.63	77.60	—
Full BPTT	63.20	88.00	2.77
Recompute + Neumann-4	60.05	89.60	2.09
Reuse + Neumann-4	61.65	90.20	1.39
Reuse + Neumann-3	61.30	89.80	1.30

Table 20: **Rollout generation dominates RL training time.** Realized counts and times for the GSM8K runs of Table 19. Tokens counts sampled response tokens. Train (min) includes rollout, answer checking, and updates; Update (s) sums scoring, backward, and optimizer time. Prompts whose responses all receive the same reward skip replay, so per-update means fall below the medians in Table 19.

Training	Realized		Time		
Update method	Tokens	Updates	Train	Rollout	Update
L scale: (1.6B, 343.6B tokens)					
Full BPTT	1,009,546	188	65.45	57.42	463.5
Recompute + Neumann-4	880,930	186	61.71	55.50	356.5
Reuse + Neumann-4	886,381	191	57.30	52.97	244.1
Reuse + Neumann-3	858,002	192	58.98	54.95	225.9

Results. We compare full BPTT, Neumann-4 on recomputed states, and Neumann-3 and Neumann-4 on saved rollout states. Reuse speeds up scoring and backward computation by $1.50\times$ over recomputation and $1.99\times$ over full BPTT (Table 19). All methods raise pass@1 from 40.6% to 60–63% and pass@8 from 77.6% to 88–90%. Reuse with Neumann-4 is within 1.6 points of full BPTT on pass@1 and slightly higher on pass@8; dropping to Neumann-3 changes pass@1 by 0.35 points. Rollouts take about 90% of training time (Table 20), so the end-to-end saving is small.

Quality and uncertainty. Pass@1 gains over initialization in Table 19 are 22.58 ($[20.28, 24.90]$) points for full BPTT, 19.43 ($[17.28, 21.58]$) for recomputation, 21.03 ($[18.85, 23.23]$) for Reuse + Neumann-4, and 20.68 ($[18.48, 22.93]$) for Reuse + Neumann-3. Reuse + Neumann-4 differs from full BPTT by -1.55 ($[-3.13, 0.05]$) points and from recomputation by $+1.60$ ($[0.08, 3.18]$);

Neumann-3 differs from Neumann-4 by -0.35 ($[-2.05, 1.33]$). All 95% intervals use $5{,}000$ paired question-bootstrap draws, resampling questions together with all eight of their responses. They condition on one training seed and so exclude seed variance. Reuse and recomputation compute nearly identical gradients (cosine 0.997–0.999, below), so their $+1.60$ difference indicates the size of run-to-run variation; differences of this size between arms do not show an estimator effect.

Paired update cost and diagnostics. Timings in Table 19 are per-update medians from the training runs. A paired probe with fixed actions, masks, actor probabilities, and synthetic advantages on two probe questions (two warmups, five repetitions) gives 169.7, 131.6, and 89.2 ms for full BPTT, recomputation, and Reuse + Neumann-4. Whole-model gradient cosines are 0.997–0.999 between reuse and recomputation and 0.76–0.87 between reuse and full BPTT (0.74–0.86 with Neumann-3).

At $R = 6$, the relative state residual averages 0.034 (maximum 0.041 in the Neumann-4 run), larger than near a converged fixed point, so the finite Neumann series truncates more of the full gradient. Reuse's maximum replay log-probability RMSE is 0.021, and its minimum normalized importance-weight effective sample size is 0.9995. Detached BF16 states stay on the GPU for one batch and use about 2.0 GiB without host transfer; the time to capture them is counted as rollout time. The longest full-BPTT probe peaks at 49.2 GiB after Adam-state allocation.

Total cost and scope. Table 20 reports total costs, which also depend on sampled lengths and nonzero-update counts. Multiple seeds, the full test split, and reuse across updates remain untested.

D Datasets and Evaluation Protocols

We define the shared datasets, target populations, and scoring rules. Appendix C gives each experiment's checkpoint, execution budget, and any departures from these common protocols. Each cached, split, or reused-state path for evaluation or gradients was checked against a reference (uncached scoring, full autograd, or an analytic oracle) on a subset of inputs before use; the experiment descriptions omit these checks. Unless stated otherwise, results use one training seed, intervals quantify evaluation-sample uncertainty, and "batch eight" means a batch size of eight.

D.1 Mixture-Matched Validation

MixVal is a fixed mixture-matched split of 64 packed sequences of length $8{,}192$ from each of 19 TxT360 validation sources, totaling $9{,}961{,}472$ next-token targets. We report $\mathrm{PPL} = \exp(\mathrm{CE})$ from token-weighted teacher-forced cross-entropy, evaluated in BF16 with multi-segment semantics and no generation cache. Document-level separation from the training corpus remains unverified.

D.2 WikiText-103 and Zero-Shot Tasks

Table 21 lists the evaluation tasks.

We concatenate the $2{,}891$ nonempty records of WikiText-103 (Merity et al., 2016) with newlines. Native curves in Figures 3 and 5 score $310{,}315$ next-token targets in 38 windows of $8{,}192$ tokens at every depth 1–64, using batch two, BF16, FlashAttention-3 (FA3), and state seed 42. PPL exponentiates token-averaged cross-entropy; document-level pretraining decontamination was not performed.

Paired WikiText PPL in Figure 5 uses the same windows and example IDs but scores $271{,}403$ continuation targets after a $1{,}024$-token prefill, excluding the prefill prediction and padding. Both layouts start from an identical per-visit prefill at $R_{\mathrm{pre}} = R_{\mathrm{dec}} = 5$, with batch eight and FA3 disabled; terminal sharing retains each physical layer's last bank, reducing twelve banks to four.

The main tables share six zero-shot tasks from the Mamba-3 suite (Lahoti et al., 2026), using complete labeled splits (LAMBADA test, HellaSwag and PIQA validation, and ARC and OpenBookQA test): LAMBADA OpenAI ($5{,}153$), HellaSwag ($10{,}042$), PIQA ($1{,}838$), ARC-Easy ($2{,}376$), ARC-Challenge ($1{,}172$), and OpenBookQA (500). HellaSwag and ARC-Challenge normalize continuation log likelihood by character length; PIQA, ARC-Easy, and OpenBookQA sum it. LAMBADA accuracy requires every final-word subtoken to match its greedy prediction; last-word PPL exponentiates the negative mean final-word log probability given the context, summing over subtokens.

Table 21: **Evaluation tasks.** Choice tasks are zero-shot and score candidates by log likelihood (length-normalized for HellaSwag and ARC-C, summed otherwise). Generation tasks decode greedily except MBPP+, which averages eight samples at temperature 0.8.

Task	Split	Metric	Examples	Shots	Used in
LAMBADA (OpenAI)	test	accuracy, last-word PPL	5,153	0	main tables
HellaSwag	validation	accuracy	10,042	0	main tables
PIQA	validation	accuracy	1,838	0	main tables
ARC-Easy	test	accuracy	2,376	0	main tables
ARC-Challenge	test	accuracy	1,172	0	main tables
OpenBookQA	test	accuracy	500	0	main tables
SciQ	test	accuracy	1,000	0	main tables
WinoGrande	validation	accuracy	1,267	0	Appendix C.1, Appendix C.2
DROP	frozen subset	F1	500	3	main tables
GSM8K	test	accuracy	1,319	8	main tables, Appendix C.7
MBPP+ v0.2.0	frozen subset	pass@1	100	3	main tables, Appendix C.7

Tables 2 and 3 and their no-reuse counterparts add all 1,000 SciQ test questions at $R = 5$, with the support paragraph followed by `Question: {question}` and then `Answer:`. Four answers are scored by summed log likelihood. Table 8 and the entropy-sweep averages in Appendix C.1 instead add the WinoGrande (Sakaguchi et al., 2020) validation split (1,267 examples), scoring the common suffix after each candidate-filled prefix by summed log likelihood. Each seven-task AVG equally weights its accuracies, excluding PPL and generation.

The main distilled-prefill results (Table 4) use this battery, with each prompt split between the prefill path and the teacher decoder (Appendix C.6.1).

Request formatting and pair tokenization use `lm-evaluation-harness v0.4.2` (Gao et al., 2023), commit `4600d6bf`, pinned for this implementation; Mamba-3 does not specify a harness revision. Native scoring uses BF16 with FA3, candidate batch eight, and no BOS, EOS, chat template, or generation cache. State seed 42 and fixed example IDs are shared across candidates and checkpoints. Requests are right-padded into token-determined power-of-two buckets of minimum width 128; batches and bucket shapes stay fixed across models. We keep all ARC questions, including those with three or five choices, and reject rather than truncate inputs over 8,192 tokens.

The reward ablations (Table 10) instead use raw-likelihood scoring and a four-task mean.

D.3 Terminal-KV Evaluation

Tables 2 and 3 and the Teacher row of Table 4 score every checkpoint at $R = 5$ with terminal KV sharing, using the battery's requests and scorers. At $R = 5$, a token visits the four physical attention layers twelve times: the prelude and coda once each, and each of the two core blocks five times. The cache keeps one bank per physical layer. Because the prelude and coda run once per token, their banks are exact; each core bank keeps only the final-recurrence KV of earlier tokens. A forward pass over a chunk of new tokens runs all five recurrences; at every core visit, the chunk attends to that layer's bank and to its own KV from the same visit, and after the final recurrence its KV is appended to the bank. From an empty cache this pass equals the native forward pass and differs only in what it stores; every checkpoint passes this check on MixVal prefixes.

Prefill passes every prompt token but the last as one chunk, then the last prompt token alone. For the seven choice tasks, the context is the prompt, and each candidate token is then passed alone, so every scored token reads earlier tokens only through terminal banks. Generation prefills the prompt the same way and decodes one token at a time with the settings of Appendix D.5. For Val. and Wiki, the prompt is the first 4,096 tokens of each 8,192-token sequence (1,216 MixVal and 38 WikiText sequences), and the remaining 4,096 tokens form one chunk; PPL covers these tokens and the prediction from the last prompt token. Tokens in this chunk read the prefix through its terminal banks but attend to one another as in training, so Val. and Wiki test sharing of the prefix only. Tables 7 and 11 evaluate the same checkpoints without KV reuse: Val. and Wiki are native

PPL over full sequences (Appendix D.1), choice tasks are scored without a cache, and generation uses $R = 5$ prefill and decoding with twelve visit-specific KV banks and a BOS token.

D.4 Cached-Continuation Perplexity

Cached PPL is teacher-forced next-token perplexity from the native incremental decoder. Prefill's final-position logits score the first continuation target; appending each reference token once scores the next target. For the scored positions $\mathcal{T}_i$ of sequence i,

$$\mathrm{CE}_{\mathrm{cache}} = -\frac{1}{\sum_i |\mathcal{T}_i|} \sum_i \sum_{t \in \mathcal{T}_i} \log p_\theta^{\mathrm{inc}}(x_{i,t} \mid x_{i,<t}), \qquad \mathrm{PPL}_{\mathrm{cache}} = \exp(\mathrm{CE}_{\mathrm{cache}})\,. \tag{27}$$

Predicted EOS tokens do not stop scoring.

CacheEval contains 304 frozen single-document MixVal spans (16 per source), each with 1,024 prompt tokens and 512 continuation targets. The cache grows from 1,024 to 1,535 tokens over 511 teacher-forced inputs. Prior, input-injection, and training-depth comparisons score targets 2–512 (155,344 total), using five recurrences for both prefill and decoding; Appendix C.5 varies these budgets independently. Each experiment specifies its cache layout.

D.5 Generation Benchmarks

All generation scores are percentages. DROP (Dua et al., 2019) uses official F1 on 500 frozen examples (three-shot, greedy), GSM8K (Cobbe et al., 2021) uses numeric-answer accuracy on all 1,319 test examples (eight-shot, greedy), and MBPP+ uses estimated pass@1 on 100 frozen v0.2.0 tasks (three-shot, eight samples per task, temperature 0.80). MBPP+ augments MBPP (Austin et al., 2021) with EvalPlus tests (Liu et al., 2023). Generation caps are 64 tokens for DROP and 512 for GSM8K and MBPP+. RL uses the same eight GSM8K demonstrations; Appendix C.7 gives its training split and rollout protocol. Each experiment specifies its model and recurrence budgets.

E Extended Related Work

We position the learned depth prior, OrthoInj, terminal KV sharing, rollout-state reuse, and distilled prefill against related work that Section 6 covers only briefly.

E.1 Learned Depth Distributions and Exits

Ouro, RL-Halting, and our prior all optimize an entropy-regularized expected objective over depths (Zhu et al., 2025; Kuo et al., 2026). Ouro takes the exact expectation with per-token exit gates and reuses the gates for early exit.

Closest related work. The closest related work is RL-Halting (Kuo et al., 2026), which also supervises one sampled depth per example and learns the depth distribution with REINFORCE. Its stopping head reads the detached, average-pooled hidden state after each iteration and outputs a hazard π_r, the probability of stopping at recurrence r given no earlier stop, so $p(R = r \mid x) = \pi_r \prod_{r'<r}(1-\pi_{r'})$. The reward is the negative task loss with a moving-average baseline and entropy coefficient 0.01, and inference stops at the most probable depth. Because the hazards depend on the states along the loop, training unrolls every example to its maximum depth of 20 to form the distribution; the sampled depth selects the supervision but not the compute. A hazard that ignores the state is constant and gives a geometric distribution, so the head cannot start from a prescribed prior such as capped PLN. RL-Halting has no budget term and targets length extrapolation on synthetic tasks; it does not evaluate fixed points or KV sharing.

Our prior is unconditional and used only in training, giving it three advantages over Ouro and RL-Halting. First, it saves compute: it samples one depth per microbatch before the loop and runs only that depth, and the budget term in Equation (5) fixes the expected forward compute at $\bar{R}$. Ouro instead runs every step to maximum depth for the exact expectation, and RL-Halting unrolls to that maximum depth to form its state-dependent distribution. Second, it starts from a strong prior: its logits start at the capped PLN prior, so each learned run begins at the fixed-prior baseline, whereas

a state-dependent hazard has no setting that reproduces a prescribed prior. Third, it deliberately promotes KV sharing: because the prior does not decide where inference stops, its entropy term extends supervision above $\bar{R}$, where settled prefix context supports terminal KV sharing (Appendix C.1). Table 10 compares its inverse-perplexity reward to RL-Halting's raw CE.

Learned exits have two limits for fixed-point training. A single exit distribution both stops inference and weights the per-depth training losses; Popescu et al. (2026) trace gate failures mainly to the trajectory this joint training induces, and find that fixed-prior depth supervision yields useful trajectories. An exit also stops once prediction suffices, so training need not reach depths beyond prefix convergence, where supervision prepares the model for terminal KV sharing (Section 3.2). Depth uncertainty also learns a global depth distribution, but within a Bayesian objective that marginalizes over depths at inference (Antorán et al., 2020).

E.2 Input Injection and Orthogonal Updates

DEQ-Transformer injects embeddings into Q/K/V; DEQ-TrellisNet injects them before convolutional LSTM gates (Bai et al., 2019a, Section 4). TrellisNet combines causal convolution, shared depth, and persistent injection (Bai et al., 2019b). Huginn and Parcae instead inject at the recurrent-core entrance (Geiping et al., 2025; Prairie et al., 2026).

Orthogonal residual updates (Oh et al., 2025) also apply a projection inside a residual block, but to the other operand. They keep only the component of a module's output that is orthogonal to the residual stream, so an update adds new directions without rescaling the stream. OrthoInj (Equation (7)) instead projects the carried-over state off the injected input q_t and then adds q_t, so the component along q_t is fixed at every recurrence while the orthogonal part evolves freely. The two constraints act on different quantities: what a block adds, and what the recurrence retains.

E.3 KV Sharing Across Recurrent Depth

Huginn reports zero-shot cyclic reuse of a fixed historical-KV budget, with later recurrent visits overwriting earlier entries (Geiping et al., 2025, Section 6.2). Ouro's last-loop reuse during decoding largely preserves GSM8K accuracy with a modest MATH-500 (Lightman et al., 2024) decrease, while prefill sharing degrades performance (Zhu et al., 2025, Section 5.4.2, Table 14). Even with visit-specific prompt banks, decode-time sharing represents the generated tokens by shared banks.

MELT instead finds zero scores on its Table 5 reasoning benchmarks for all four combinations of first-/last-loop reuse and preserved/shared prompt caches on Ouro-1.4B-Thinking (Conchello Vendrell et al., 2026, Appendix B). Its LightEval setup permits 32K-token completions. Differences from Ouro's few-shot CoT evaluation, alongside incomplete reproduction of the original results, leave the discrepancy unresolved (Conchello Vendrell et al., 2026, Section 4.1 and Appendix F.1). MELT adapts the model to a shared cache with gated updates, chunk-wise training, a gradual transition, and attention-aligned distillation; it does not directly evaluate Huginn.

Under a common GSM8K setup without retraining, Schwethelm et al. (2026, Appendix C.1, Table 1) find that one shared KV slot per physical layer leaves Huginn accuracy nearly unchanged at 32 recurrences, while Ouro accuracy collapses at four. A separate first-visit slot recovers most of Ouro's accuracy. These results suggest that cache compatibility depends on the model; Huginn under MELT's long-reasoning protocol remains unmeasured. These studies test sharing zero-shot or after adaptation. We instead change only the training depth policy, which decides whether terminal sharing is nearly lossless (Table 2); like them, we do not measure long-form reasoning.

E.4 Fixed-Point Reuse

Implicit models reuse a fixed point to warm-start the solver on a related input. DEQ-Flow and StreamDEQ start each video frame from the previous frame's fixed point (Bai et al., 2022a; Ertenli et al., 2022), and fixed point diffusion models reuse solutions between denoising timesteps (Bai & Melas-Kyriazi, 2024). In each case, the reused state initializes a new forward solve. Rollout-state reuse (Section 3.1) instead carries states between the two passes that RL makes over the same tokens, rollout and the policy update. The update scores the sampled tokens from the endpoints saved during

rollout, so it does not replay the recurrence, and backpropagates through a single recurrence at each endpoint with a truncated Neumann series (Appendix C.7).

E.5 Distillation for Looped Models

Distillation has trained looped students from non-looped teachers (Bae et al., 2025a) and adapted looped models to cheaper KV caches: MELT adapts Ouro to one gated cache per layer with attention-aligned distillation (Conchello Vendrell et al., 2026), and Looped Latent Attention distills a codec that compresses KV across loops (O'Neill & Reid, 2026). LoopFormer trains shorter loop trajectories of the same model to match the final states of the full trajectory (Jeddi et al., 2026). In each case, the student still iterates the recurrent block. Distilled prefill instead trains a separate, non-recurrent student to predict the endpoint of a frozen looped teacher and uses it only for the prompt; decoding keeps the teacher's recurrence. The closest idea is the learned initializer of HyperDEQ, a small network that predicts the starting point of a frozen DEQ's fixed-point solver (Bai et al., 2022b). Our student instead replaces the solver: one teacher recurrence from its prediction builds every KV bank that decoding reads under terminal sharing.