LEARNING FROM RECURRENCE: INTERNALIZING DEPTH-INDUCED PREFERENCES IN LOOPED LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recurrent language models can improve predictions by repeatedly applying the same parameters, making depth a property of computation rather than model size. We ask whether behavior exposed by additional recurrence can be written back into the shared computation itself, a transfer we call *preference write-back*. Trajectories generated by deeper execution then self-distill only the shared recurrent block, without an additional correctness signal. On a 1B model retrofitted with recurrent depth, this update substantially improves both one-pass and full-depth execution on mathematical reasoning, across models trained for mean depths from 4 to 32: GSM8K accuracy at a model's own depth rises from 56.2% to 62.7% at mean depth 4, with comparable gains at 8 and 32. Moreover, updating the block through one recurrent pass consistently outperforms training on the same targets through the full recurrent depth, while requiring less compute and memory; the resulting update also remains useful when the block is reused at later passes. A controlled comparison of looping recipes shows that this advantage of shallow training is a property of how the recurrent model was trained, not of weight tying alone. These results show that recurrent depth is not only an inference-time resource: it can generate self-supervision that improves the shared operator in both single-pass and multi-pass execution.

1 INTRODUCTION

Looped Transformer language models scale inference-time computation by reapplying recurrent blocks before emitting each token, decoupling computation depth from layer depth (Dehghani et al., 2019; Saunshi et al., 2025; Geiping et al., 2025). This makes recurrent depth attractive both for serving—via parameter sharing, depth-wise batching, and KV-cache reuse (Lan et al., 2020; Bae et al., 2025a;b; Geiping et al., 2025)—and for latent test-time computation, where the model iterates in hidden-state space before emitting a token (Geiping et al., 2025; McLeish et al., 2026; Koishekenov et al., 2025; Zhu et al., 2025). Empirically, such models can improve with additional recurrent depth and, in some settings, rival language models several times larger (Saunshi et al., 2025; Geiping et al., 2025; Zhu et al., 2025).

The additional computation, however, is not free. Although recurrent KV state can in some settings be reused or compressed, each additional pass still incurs sequential computation before the next token is emitted (Geiping et al., 2025; Bae et al., 2025b; Zhu et al., 2025; O'Neill & Reid, 2026). Recent work therefore treats recurrent depth as an explicit inference-time compute budget (Jeddi et al., 2026; Bae et al., 2025b). This raises a natural question: *can behavior revealed by additional recurrence be learned back into the shared computation itself?*

A central uncertainty is what that additional computation is doing. Recurrent models can learn genuine iterative computation in controlled settings, including repeated algorithmic operations and computing intermediate results in one recurrent pass that are used in subsequent passes (Schwarzschild et al., 2021; Bansal et al., 2022; Saunshi et al., 2025; Jolicoeur-Martineau, 2025; Kohli et al., 2026; Huang et al., 2026). But these results do not determine the role of additional passes in pretrained language models retrofitted with recurrence (Bae et al., 2025a; McLeish et al., 2026; Koishekenov

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.

et al., 2025): improvements with depth may reflect new intermediate computation, refinement of an already-formed predictive state, or a combination of the two.

We therefore examine how the next-token distribution changes with recurrent depth. In the models we study, additional recurrence primarily redistributes probability among tokens that already rank highly after shallow execution, rather than promoting previously low-ranked tokens (the top-ranked token is unchanged across four passes at 95.6% of generated positions; Section 2.2). This observation connects to recent results on simple self-distillation, where changes in a model's own predictive distribution can provide useful training signal (Zhang et al., 2026). Here, deeper recurrent execution itself produces such a change, suggesting that its greedy trajectories could supervise the shared recurrent block without an additional correctness signal.

This paper tests this possibility by using trajectories generated at deeper recurrent depth as supervision for the same model, while training only the shared recurrent block through a shallower execution. Because the updated block is reused at every recurrent pass, this setup asks whether behavior exposed by additional computation can be internalized into the shared computation itself. Specifically, we make the following **contributions**. We show that, in looped language models retrofitted from a pretrained model, additional recurrence revises relative preference among continuations that are already plausible (Section 2.2), and we turn this depth-induced change into supervision through self-distillation on greedy deeper trajectories, without reference answers or filtering (Section 2.3). By separating teacher, training and inference depth, we find that learning these trajectories through a single pass improves deep recurrent inference at every mean depth we study, from 4 to 32, raising GSM8K accuracy (Cobbe et al., 2021) at the mean depth by 8.9–11.6% relative to the model before write-back (Section 3.2). One-pass learning outperforms learning through the full depth, which we trace to deeper backpropagation weakening and redirecting the gradient that reaches the first-pass state (Section 3.2), and later recurrent passes reuse the update it writes (Section 3.3). This advantage holds in the input-injected models we study (Section 3.4); the signal comes from deeper teachers, transfers across question pools, and leaves 17 non-mathematical tasks (Gu et al., 2025; Team OLMo et al., 2025) essentially unchanged (Section 3.5).

2 Learning from recurrence

2.1 Looped language models: depth as repeated computation

A shared computation across depth. Looped language models increase execution depth by repeatedly applying the same recurrent core to an evolving latent state. We take this shared core as the object of learning: updating its parameters changes the computation used at every recurrent pass.

Input-conditioned recurrent computation. We retrofit the OLMo-2 1B stage-1 checkpoint (Team OLMo et al., 2025) into a non-recurrent input stack P, a shared recurrent core F_θ, and a non-recurrent output stack C, following prior recurrent architectures (Geiping et al., 2025) and prior retrofits of pretrained models (McLeish et al., 2026). The input and output stacks are termed the *prelude* and *coda* in prior work. We assign the first seven Transformer blocks to P, the middle four to F_θ, and the final five to C, following the partition used for this model family (Koishekenov et al., 2025). This partition places recurrence in a middle region where the angular changes between consecutive hidden representations have stabilized, retaining separate input and output transformations.

Let $x_{1:n} \in \mathcal{V}^n$ be a prefix of n tokens from vocabulary $\mathcal{V}$. The input stack, including token embeddings, produces $e = P(x_{1:n}) \in \mathbb{R}^{n\times d}$, where d is the model width. Starting from an initial state $s_0 \in \mathbb{R}^{n\times d}$, an execution of depth r computes

$$s_i = F_\theta(A_\psi([s_{i-1}; e])), \qquad i = 1, \dots, r.$$

Here, $s_i \in \mathbb{R}^{n\times d}$ is the state after pass i, and $[\cdot;\cdot]$ denotes concatenation along the hidden dimension. The recurrent core $F_\theta : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$ and the position-wise linear projection $A_\psi : \mathbb{R}^{2d} \to \mathbb{R}^d$ reuse the same parameters θ and ψ at every pass. This *input injection* makes the same representation e available throughout the evolution of the recurrent state. After the final pass, the output stack $C : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d}$ and language-model head $W \in \mathbb{R}^{|\mathcal{V}|\times d}$ produce the next-token distribution $p_r(\cdot \mid x_{1:n}) = \mathrm{softmax}(W[C(s_r)]_n)$, where $[\cdot]_n \in \mathbb{R}^d$ selects the final-position representation. The input and output stacks are each applied once.

Learning across recurrent depths. We adapt the pretrained parameters to recurrent execution through continued causal language-model training on the Dolmino mixture (Team OLMo et al., 2025). During this stage, the execution depth is sampled once per training step from a Poisson-lognormal depth distribution, following prior recurrent-depth training (Geiping et al., 2025). Let

$$r \sim \pi_t, \qquad \mathbb{E}_{r\sim\pi_t}[r] = \mu_t,$$

where r is the sampled number of recurrent passes and π_t denotes the Poisson-lognormal depth distribution at training step t. Rather than training immediately at the final depth distribution, we follow the recurrence curriculum of McLeish et al. (2026) and gradually increase its mean μ_t toward a target value m. The language-model objective is therefore optimized over recurrent computations of varying depth, training the shared core to operate across the depth distribution.

We call m the model's *mean depth*, using target values of 4, 8, and 32. Mean depth characterizes the recurrent-depth distribution under which the model is learned, whereas the *execution depth* r specifies the number of recurrent passes used in an individual execution. Once mid-training is complete, r remains independently selectable for a fixed checkpoint, so shallow and deep executions are obtained by applying the same learned recurrent computation for different numbers of passes. Initialization and curriculum details are provided in Appendix C, and Appendix A shows that recurrence already improves accuracy during mid-training. The resulting checkpoint supplies both the model to be updated and the deeper executions from which we construct supervision.

2.2 Recurrence-induced preferences as supervision

Once mid-training is complete, execution depth can be varied while the learned parameters remain fixed. This raises a natural question: *what does additional recurrence contribute to the prediction of a fixed model?* We answer this at the level of the next-token distribution, asking how repeated applications of the shared core reshape the model's preference among candidate continuations. We first characterize this change, then use the preference of deeper execution as supervision.

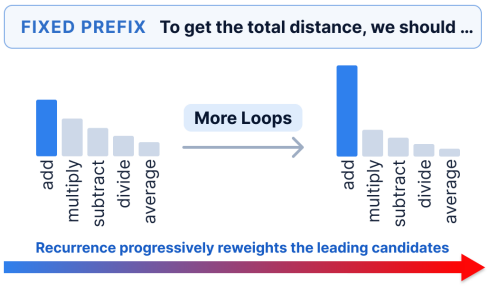

Figure 1: **Recurrence revises predictive preference.** At a fixed prefix, successive recurrent passes redistribute probability among leading candidates, occasionally changing which continuation is preferred.

Recurrence refines predictive preference. The behavioral contribution of additional recurrence is primarily a refinement of preference among already plausible continuations. We expose this change using intermediate readouts along a fixed deep-generation path, decoding each recurrent state with the same output stack while leaving the generation itself unchanged. In the mean-depth-4 model, the top-ranked token remains unchanged across four passes at 95.6% of generated positions, even as probability shifts toward the leading token while total top-10 mass changes little. When the final top-ranked token does change, 99.4% of those choices are already contained in the first-pass top five. The dominant effect of recurrence is therefore a redistribution of probability within the head of the predictive distribution, occasionally strong enough to change the selected token. We take this depth-refined preference as the behavioral signal to be preserved as supervision. Figure 1 illustrates this effect; Appendix B details the readout protocol and analysis.

From predictive preference to behavioral supervision. The remaining question is how a distribution-level preference revealed by deeper recurrence should be represented as a learnable target. We use trajectory construction as this bridge: a decoding rule turns the deeper next-token distribution into a complete autoregressive continuation. This preserves the token preferred at each step and the contexts induced by those choices, through which subsequent preferences are expressed.

In our default construction, we freeze the pre-update checkpoint and choose a teacher depth r_{T}. For a prompt x, greedy decoding produces a continuation $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_T)$ according to

$$\hat{y}_t = \arg\max_{v\in\mathcal{V}} p^{(0)}_{r_{\mathrm{T}}}(v \mid x, \hat{y}_{<t}), \qquad t = 1, \ldots, T,$$

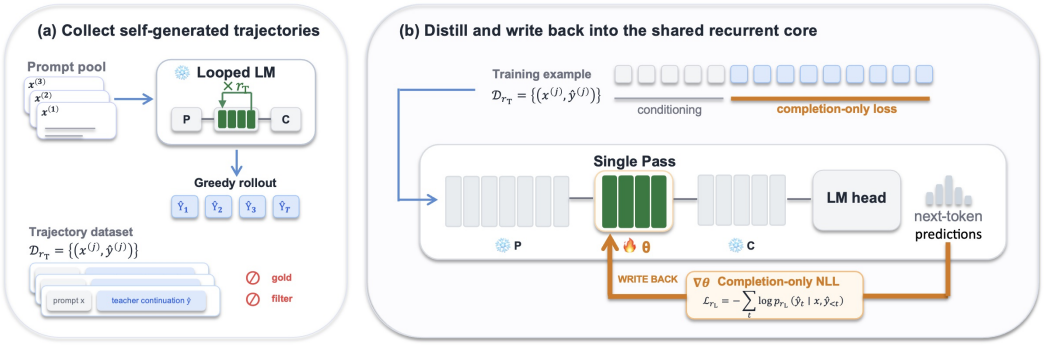

Figure 2: **Preference write-back.** (a) The frozen model generates one greedy continuation per prompt at teacher depth r_{T}, without reference answers or filtering. (b) These trajectories train only the shared recurrent core through a single pass (r_{L}=1); the updated core is reused at every pass.

where $p^{(0)}_{r_{\mathrm{T}}}$ is the predictive distribution of the frozen pre-update model after r_{T} recurrent passes. The resulting trajectory is a sequence-level realization of the preference expressed by deeper execution: it preserves the behavior selected by that distribution while discarding its exact probability margins.

For N training prompts, we collect one such continuation per prompt to form

$$\mathcal{D}_{r_{\mathrm{T}}} = \left\{ \left(x^{(j)}, \hat{y}^{(j)} \right) \right\}_{j=1}^{N}.$$

Each target retains the complete generated continuation, including its reasoning and final answer. Reference answers to the target questions are neither provided during generation nor used to filter trajectories. Teacher depth determines which recurrent behavior is exposed, while the trajectory-construction rule determines which realization of that behavior is retained as supervision. How this supervision is written back into the shared computation is a separate choice, discussed next.

2.3 Writing preferences back into the shared recurrent core

Once deeper behavior has been converted into supervision, the remaining problem is to turn a transient effect of additional computation into a persistent change in the computation itself. In a weight-tied model, this means writing the behavior back into the shared operator that produced it and that subsequent recurrence will reuse. Crucially, the execution that reveals the behavior need not be the execution through which the operator learns it. We refer to this transfer from depth-induced behavior into reusable computation as *preference write-back* (Figure 2; Algorithm 1 in Appendix C.3).

Decoupling behavior exposure from incorporation. Preference write-back separates the source of supervision from the computation through which that supervision is learned. The *teacher depth* r_{T} specifies the execution whose behavior is retained as supervision, while the *training depth* r_{L} specifies the recurrent computation through which that supervision acts on the shared operator. For a fixed target set $\mathcal{D}_{r_{\mathrm{T}}}$, changing r_{L} leaves both the behavioral target and the shared parameterization unchanged; it changes only the computational path connecting that target to the shared parameters. Behavior exposed at one depth can thus be learned through another execution of the same operator.

Writing behavior into the reusable operator. The supervision is behavioral, but its write-back destination is the shared recurrent operator. Let θ_0 denote the pre-update parameters of the recurrent core, and let ϕ collect the remaining model parameters, which are held fixed; ϕ includes the injection projection A_ψ, so input injection acts in the forward computation but is not itself updated. Given $\mathcal{D}_{r_{\mathrm{T}}}$, we initialize $\theta = \theta_0$ and optimize the recurrent core through an execution of depth r_{L} using completion-only next-token likelihood:

$$\mathcal{L}_{r_{\mathrm{L}}}(\theta) = -\frac{1}{\sum_j T_j} \sum_{j=1}^{N} \sum_{t=1}^{T_j} \log p_{r_{\mathrm{L}}}\left(\hat{y}^{(j)}_t \mid x^{(j)}, \hat{y}^{(j)}_{<t}; \theta, \phi \right).$$

Here, $p_{r_{\mathrm{L}}}(\cdot\,;\theta,\phi)$ denotes the predictive distribution obtained after r_{L} applications of the recurrent core, in the notation of Section 2.1. Only the shared recurrent core is updated; all remaining model parameters stay fixed. The resulting parameters θ_1 store the deep-execution behavior in the same transformation that recurrence will invoke again. Training depth controls the path by which supervision reaches the operator, while the object being changed remains the reusable computation.

Reusing the written-back computation. After optimization, the updated core F_{θ_1} becomes the recurrent operator used at every subsequent pass, while the remaining parameters ϕ remain unchanged. At inference depth r_{E}, the model repeatedly applies F_{θ_1} as the recurrent state evolves. The update is therefore not tied to the execution through which it was learned: once written into the shared operator, it is available wherever that operator is reused.

Preference write-back thus separates three roles that are otherwise coupled by recurrent depth: where behavior is exposed, how it is incorporated, and where the resulting computation is reused. These roles are indexed by teacher depth r_{T}, training depth r_{L}, and inference depth r_{E}, respectively, while all three operate on the same shared recurrent computation. The mechanism turns behavior revealed by additional computation into feedback for improving the computation that will be used again.

3 Preference write-back in looped language models

Experimental setup. We study looped language models obtained by continuing mid-training an OLMo-2-1B base model, with mean recursive inference depths ranging from 4 to 32 recurrent passes. Unless otherwise stated, self-distillation follows the recipe introduced in Section 2: each model generates greedy trajectories using its standard deeper execution, without correctness filtering, and only the shared recurrent block is updated through a single recurrent pass. We evaluate the same model before and after self-distillation, primarily on mathematical reasoning tasks under greedy decoding, reporting both shallow execution and the model's standard inference depth. Architecture, training, and evaluation details are given in Section 2 and Appendix C.

3.1 Depth-induced preferences can be written back into recurrent computation

We first ask whether the preference exposed by deeper recurrence can become part of the shared computation itself. Section 2 showed that recurrence primarily reweights continuations that are already plausible after the first pass. Our self-distillation target is a hard, sequence-level realization of this change: greedy decoding selects the continuation preferred by the deeper execution, and that trajectory is used to update the shared recurrent block. Despite discarding the full probability shift, this signal is sufficient to improve the computation that produced it. On the mean-depth-4 model, self-distillation improves GSM8K accuracy by 7.66 percentage points after one pass and 6.52 points after four passes (Table 1). Crucially, the updated model continues to benefit from recurrence at inference: running four passes instead of one still adds 14.18 points (48.52% to 62.70%), close to the 15.32-point depth gain of the model before self-distillation.

The same write-back effect extends to substantially deeper recurrent models. Single-pass self-distillation improves deployed GSM8K accuracy by 6.52 points for the mean-depth-8 model and 5.08 points for the mean-depth-32 model. The latter is particularly informative: its single-pass execution initially achieves only 4.09%, yet that execution still provides a viable route for learning an update that improves 32-pass inference. A shallow execution therefore need not itself be a strong inference procedure to absorb behavior revealed by deeper recurrence. The same pattern holds on GSM1K, while SVAMP shows that the effect is not uniform across tasks. Together, these results show that behavior exposed by additional recurrent computation can be absorbed into the shared operator and reused when that operator is applied repeatedly.

3.2 Shallow learning writes deep behavior into the shared operator

Once deeper recurrence has revealed useful behavior, at what depth should it be written back into the shared operator? We isolate this question by fixing the trainable parameters and optimization, and varying the number of recurrent passes used during learning (Table 3). Learning four-pass targets through one pass improves subsequent four-pass inference by 5.84 percentage points

Table 1: **Self-distillation through one-pass training improves deep recurrent inference across model depths.** Exact match (%) under greedy decoding at depth 1 and at the mean depth m. *Mean depth* rows are the Looped LMs before self-distillation, which trains only the shared recurrent block on greedy depth-m trajectories (50,000 OpenMath questions for m=4, 10,000 otherwise). Parenthesized values and the last column give the relative change at depth m against the non-looped control.

Model	GSM8K		GSM1K		SVAMP		GSM-Symbolic		ASDiv		AQuA-RAT		Avg.
Depth	1	m	1	m	1	m	1	m	1	m	1	m	vs. control
Non-looped control	44.05	–	39.83	–	56.50	–	37.42	–	59.39	–	10.24	–	–
Mean depth 4	40.86	56.18 (+27.5%)	36.93	51.12 (+28.3%)	52.40	63.30 (+12.0%)	31.82	43.56 (+16.4%)	55.44	64.82 (+9.1%)	14.17	16.93 (+65.4%)	+26.5%
+ Self-distillation	48.52	62.70 (+42.3%)	44.65	54.36 (+36.5%)	56.40	64.70 (+14.5%)	36.92	47.64 (+27.3%)	59.09	66.55 (+12.1%)	12.20	18.11 (+76.9%)	+34.9%
Mean depth 8	22.59	56.33 (+27.9%)	20.50	52.53 (+31.9%)	39.90	66.00 (+16.8%)	15.76	45.06 (+20.4%)	46.33	66.20 (+11.5%)	10.24	23.62 (+130.8%)	+39.9%
+ Self-distillation	37.38	62.85 (+42.7%)	32.37	56.02 (+40.6%)	43.60	65.20 (+15.4%)	27.94	48.42 (+29.4%)	51.84	67.90 (+14.3%)	5.91	22.44 (+119.2%)	+43.6%
Mean depth 32	4.09	57.01 (+29.4%)	3.24	53.36 (+34.0%)	19.70	67.70 (+19.8%)	2.68	44.90 (+20.0%)	20.39	66.55 (+12.1%)	0.79	19.29 (+88.5%)	+34.0%
+ Self-distillation	31.84	62.09 (+41.0%)	26.39	54.36 (+36.5%)	36.20	65.70 (+16.3%)	20.66	48.08 (+28.5%)	44.95	67.07 (+12.9%)	2.76	24.80 (+142.3%)	+46.2%

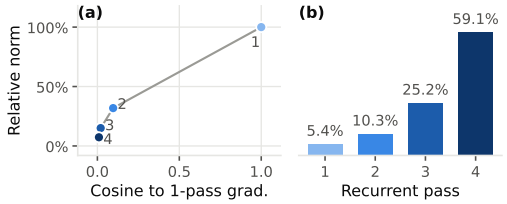

Figure 3: **Recurrent depth weakens and redirects the gradient reaching the first-pass state.** (a) Cosine similarity to the one-pass gradient and relative norm at the first-pass state, with the loss placed after one to four passes. (b) Per-pass share of the parameter-gradient norm under four-pass training. Setup in Appendix D.1.

Table 2: **The shallow-learned update is useful mainly in later recurrent passes.** GSM8K exact match (%) at four-pass inference when each pass uses the original (✗) or the self-distilled (✓) shared block. Intervals and per-question counts are in Table 11.

Pass				
1	2	3	4	Accuracy (%)
✗	✗	✗	✗	56.18
✓	✗	✗	✗	56.25 (+0.1%)
✗	✓	✓	✓	62.47 (+11.2%)
✓	✓	✓	✓	62.70 (+11.6%)

on GSM8K, 3.32 points on GSM1K and 2.50 points on SVAMP relative to learning them through four passes. To understand why training depth matters, we trace how the gradient changes as the same supervision is placed deeper in the recurrent computation. As the loss moves from one to four passes, the gradient reaching the same first-pass hidden state rapidly shrinks and changes direction; after four passes, its norm is only 7.3% of the direct one-pass gradient and its cosine similarity is 0.009 (Figure 3a). At the parameter level, four-pass training is likewise concentrated toward later uses of the shared block: the share of the summed per-pass gradient norms rises from 5.4% at the first pass to 59.1% at the fourth (Figure 3b). Together, these measurements suggest that shallow self-distillation changes where supervision acts: it places behavior revealed by deeper computation directly on a single reusable application of the shared operator, whereas full-depth training routes the same supervision primarily through its later recurrent uses. Appendix D.2 extends this comparison across inference depths and to a mean-depth-2 model.

3.3 Shallow-learned updates remain useful in later recurrence

When the same operator is reused across recurrence, does an update learned from its first use remain useful as the recurrent state evolves? We test this directly by choosing, at each of four recurrent passes, whether to use the original or self-distilled shared block, while keeping the rest of the model fixed (Table 2). Using the updated block only on the first pass leaves GSM8K accuracy where it started. In contrast, keeping the original block on the first pass and using the updated block on passes 2–4 recovers almost all of the gain obtained when the updated block is used throughout. The learned change is therefore not specific to the state on which it was directly trained:

Table 3: **Self-distillation separates the depth that generates targets from the depth that learns them.** Exact match (%) at depth 1 and at the mean depth m. Each block trains on the model's own greedy depth-m trajectories (*Teacher*) and varies the depth the student learns through (*Training*; $4^{\dagger}$: four passes, gradients through the last only). Changes are relative to the block's *Mean depth* row.

Model	Depth		GSM8K		GSM1K		SVAMP		Avg.
	Teacher	Training	Depth 1	Depth m	Depth 1	Depth m	Depth 1	Depth m	gain
Mean depth 4	–	–	40.86	56.18	36.93	51.12	52.40	63.30	–
+ Self-distillation	4	1	48.52	**62.70** (+11.6%)	44.65	**54.36** (+6.3%)	56.40	**64.70** (+2.2%)	+6.7%
	4	4	37.83	56.86 (+1.2%)	37.18	51.04 (−0.2%)	52.00	62.20 (−1.7%)	−0.2%
	4	$4^{\dagger}$	37.53	56.63 (+0.8%)	35.44	49.96 (−2.3%)	51.50	61.30 (−3.2%)	−1.5%
Mean depth 8	–	–	22.59	56.33	20.50	52.53	39.90	**66.00**	–
+ Self-distillation	8	1	37.38	**62.85** (+11.6%)	32.37	**56.02** (+6.6%)	43.60	65.20 (−1.2%)	+5.7%
	8	8	22.59	58.68 (+4.2%)	19.67	51.87 (−1.3%)	39.90	65.70 (−0.5%)	+0.8%
Mean depth 32	–	–	4.09	57.01	3.24	53.36	19.70	**67.70**	–
+ Self-distillation	32	1	31.84	**62.09** (+8.9%)	26.39	**54.36** (+1.9%)	36.20	65.70 (−3.0%)	+2.6%
	32	32	4.93	58.76 (+3.1%)	3.57	53.28 (−0.2%)	19.80	65.30 (−3.5%)	−0.2%

Table 4: **The advantage of shallow training depends on input injection.** Exact match (%) for matched mean-depth-4 models with and without input injection, self-distilled through one or four passes on their own greedy four-pass trajectories (*own*) or on those of the injected model (*injected*). Changes are relative to the *Mean depth 4* row with the same injection setting.

Model	Injection	Trajectories	Training depth	GSM8K		GSM1K		SVAMP		Avg.
				Depth 1	Depth m	Depth 1	Depth m	Depth 1	Depth m	gain
Mean depth 4	✓	–	–	40.86	56.18	36.93	51.12	52.40	63.30	–
+ Self-distillation	✓	own	1	48.52	**62.70** (+11.6%)	44.65	**54.36** (+6.3%)	56.40	**64.70** (+2.2%)	+6.7%
	✓	own	4	37.83	56.86 (+1.2%)	37.18	51.04 (−0.2%)	52.00	62.20 (−1.7%)	−0.2%
Mean depth 4	✗	–	–	43.90	50.80	38.42	45.31	54.90	58.70	–
+ Self-distillation	✗	own	1	47.16	52.39 (+3.1%)	42.90	45.31 (+0.0%)	54.60	56.50 (−3.7%)	−0.2%
	✗	own	4	44.58	50.72 (−0.1%)	40.41	46.72 (+3.1%)	53.90	59.00 (+0.5%)	+1.2%
	✗	injected	1	48.98	54.74 (+7.8%)	41.91	46.39 (+2.4%)	56.40	56.80 (−3.2%)	+2.3%
	✗	injected	4	45.26	54.81 (+7.9%)	39.75	47.63 (+5.1%)	56.20	59.60 (+1.5%)	+4.9%

an update learned through shallow execution remains useful when the same operator is applied to states produced later by recurrence. More broadly, where an update is learned and where it becomes most useful need not coincide.

3.4 When does shallow learning help deep inference?

What determines whether learning through shallow execution produces a better model at deep inference? Sections 3.1–3.3 show that a shared operator can learn from deeper behavior through a single pass and remain useful when reused later in recurrence. We next ask whether this property follows from weight tying alone or depends on how the looping model is trained. In our OLMo-2 retrofit, the main architectural choice beyond sharing the recurrent block is input injection, which reintroduces the same input representation at every pass. We therefore keep the dynamic-depth training setup fixed and compare otherwise matched models with and without injection (Table 4). With input injection, learning four-pass targets through one pass outperforms learning through four passes by 5.84 percentage points on GSM8K and 3.32 points on GSM1K at four-pass inference. Without injection, one-pass training still improves the model, but no longer gives the better deep-inference result: on the injected model's trajectories, four-pass training matches it on GSM8K and performs better on GSM1K and SVAMP. The ability to learn at shallow depth is therefore distinct from the advantage of learning there: weight tying can make the same operator reusable, but how that operator is trained determines whether shallow updates remain especially effective at deep inference.

3.5 How broadly does self-distillation apply?

How much recurrent computation is needed before the teacher exposes behavior worth writing back? A one-pass teacher provides a useful self-training baseline, but it does not include any behav-

Table 5: **Useful self-distillation signals arise across teacher depths and trajectory construction choices.** Exact match (%) with one-pass training. Panels vary the teacher depth, the question pool and the target decoding (*Setting*); the last two panels use the mean-depth-4 model, and all other settings follow the default recipe. Changes are relative to the matching *Mean depth* row.

Model	Setting	GSM8K		GSM1K		SVAMP		Avg.
		Depth 1	Depth m	Depth 1	Depth m	Depth 1	Depth m	gain
Mean depth 4	–	40.86	56.18	36.93	51.12	52.40	63.30	–
+ Self-distillation	teacher depth 4	48.52	**62.70** (+11.6%)	44.65	**54.36** (+6.3%)	56.40	**64.70** (+2.2%)	+6.7%
	teacher depth 1	38.06	59.29 (+5.5%)	36.18	51.95 (+1.6%)	53.10	63.60 (+0.5%)	+2.5%
Mean depth 32	–	4.09	57.01	3.24	53.36	19.70	**67.70**	–
+ Self-distillation	teacher depth 32	31.84	62.09 (+8.9%)	26.39	**54.36** (+1.9%)	36.20	65.70 (−3.0%)	+2.6%
	teacher depth 8	27.90	**62.85** (+10.2%)	23.90	53.69 (+0.6%)	34.70	65.10 (−3.8%)	+2.3%
+ Self-distillation	GSM8K train (7,473)	47.92	61.26 (+9.0%)	40.66	52.95 (+3.6%)	54.90	64.60 (+2.1%)	+4.9%
	OpenMath (10,000)	48.52	62.62 (+11.5%)	42.16	53.94 (+5.5%)	52.40	61.20 (−3.3%)	+4.6%
	OpenMath, curated (10,000)	46.78	61.56 (+9.6%)	41.83	54.27 (+6.2%)	54.10	63.90 (+0.9%)	+5.6%
	OpenMath (30,000)	50.11	**63.38** (+12.8%)	43.49	53.61 (+4.9%)	54.60	63.40 (+0.2%)	+5.9%
	OpenMath (50,000)	48.52	62.70 (+11.6%)	44.65	**54.36** (+6.3%)	56.40	**64.70** (+2.2%)	+6.7%
+ Self-distillation	greedy	48.52	**62.62** (+11.5%)	42.16	**53.94** (+5.5%)	52.40	**61.20** (−3.3%)	+4.6%
	sampled, T=0.7, k=10	45.11	60.50 (+7.7%)	41.24	53.03 (+3.7%)	52.10	59.80 (−5.5%)	+2.0%

ior revealed by additional recurrence. On the mean-depth-4 model, self-distillation from one-pass trajectories yields only modest gains at four-pass inference, whereas four-pass trajectories roughly double the relative improvement on GSM8K and produce larger gains on GSM1K and SVAMP (Table 5). This shows that recurrence itself contributes useful supervision. At larger depth, however, the useful signal can emerge well before the teacher reaches the largest inference depth we evaluate. On the mean-depth-32 model, trajectories generated after eight passes already yield nearly the same overall improvement at 32-pass inference as trajectories generated after 32 passes, with neither teacher uniformly better across tasks. The value of teacher depth therefore lies in whether additional recurrence reveals behavior worth writing back. Once that behavior has emerged, further teacher computation may no longer be necessary to obtain an effective training signal.

How carefully must the self-distillation signal be constructed once deeper recurrence has exposed useful behavior? We vary both the question pool used to generate teacher trajectories and the way those trajectories are decoded. The effect transfers readily across question sources: using GSM8K or OpenMath prompts produces similar gains, showing that the signal is not tied to a particular dataset. Target construction can also remain simple. On the same OpenMath questions, one greedy trajectory per prompt produces more than twice the average relative gain of truncated sampling. Data quality can partly substitute for quantity as well: a curated 10,000-question subset retains about 84% of the gain obtained with 50,000 questions, using only one fifth as many prompts. Useful behavior revealed by recurrence can therefore be learned from varied question pools with a simple deterministic teacher path. In our setting, exposing the right behavior matters more than constructing a more elaborate sampling distribution around it.

Does writing deeper reasoning behavior back into the shared operator disturb capabilities outside the self-distillation domain? Across 17 non-mathematical OLMES tasks covering knowledge, exams, commonsense reasoning, science QA, reading, and open-domain QA, the overall performance remains essentially unchanged after self-distillation (Table 6). Averaged across all 17 tasks, the change is within 0.2 points for each of the mean-depth-4, mean-depth-8, and mean-depth-32 models, while several task groups and individual benchmarks improve slightly after the update. The gains in mathematical reasoning do not come at the cost of the model's broader capabilities; the shared-operator update remains compatible with behavior beyond the self-distillation questions.

4 Related Work

Recurrent depth and looped language models. Weight sharing across depth dates back to Universal Transformers (Dehghani et al., 2019) and ALBERT (Lan et al., 2020); looped Transformers can emulate iterative algorithms (Giannou et al., 2023) and approach the reasoning ability of deeper models (Saunshi et al., 2025). Recent work pre-trains recurrent-depth language models at scale

Table 6: **Broader non-mathematical capabilities remain stable after self-distillation.** OLMES averages at the mean depth over 17 tasks in six groups; percentages are relative to the non-looped control, averaged within each group. Per-task results are in Table 12.

Model	Knowledge (2)	Exams (2)	Commonsense (4)	Science QA (3)	Reading (3)	Open QA (3)	All (17)
Non-looped control	29.63	35.69	69.65	61.26	61.42	46.56	53.94
Mean-depth-4 Looped LM	32.39 (+9.0%)	39.97 (+12.0%)	70.04 (+0.6%)	66.01 (+8.6%)	67.20 (+10.5%)	47.45 (+1.5%)	56.88 (+6.3%)
+ Self-distillation	32.72 (+10.5%)	40.07 (+12.3%)	69.87 (+0.3%)	65.91 (+8.6%)	67.95 (+12.3%)	47.59 (+2.4%)	57.02 (+6.9%)
Mean-depth-8 Looped LM	34.34 (+17.5%)	41.65 (+16.7%)	69.91 (+0.4%)	67.27 (+10.8%)	64.94 (+6.8%)	46.32 (−2.6%)	56.90 (+6.8%)
+ Self-distillation	34.36 (+17.0%)	41.62 (+16.6%)	70.28 (+1.0%)	67.47 (+11.1%)	65.26 (+7.7%)	46.43 (−2.1%)	57.09 (+7.1%)
Mean-depth-32 Looped LM	34.04 (+16.5%)	41.86 (+17.3%)	70.14 (+0.8%)	67.02 (+10.5%)	63.56 (+3.1%)	45.89 (−3.3%)	56.58 (+6.0%)
+ Self-distillation	33.79 (+14.9%)	41.01 (+14.9%)	70.18 (+0.8%)	66.60 (+9.8%)	63.18 (+3.1%)	46.37 (−2.0%)	56.40 (+5.6%)

(Geiping et al., 2025; Zhu et al., 2025), converts pretrained models into looped form (Bae et al., 2025a; McLeish et al., 2026; Koishekenov et al., 2025), and adapts the number of passes to each token (Bae et al., 2025b) or to a compute budget (Jeddi et al., 2026). These works shape how recurrence is trained and spent; we instead write the behavior that extra passes expose in a trained model back into the recurrent operator, without reference answers.

Turning inference-time computation into parameters. Extra inference-time computation improves reasoning (Snell et al., 2025), and several lines of work fold it back into the model: sequence-level distillation of search outputs (Kim & Rush, 2016) and born-again self-distillation (Furlanello et al., 2018), internalizing chain-of-thought into latent reasoning (Hao et al., 2025; Shen et al., 2025), and training on a model's own generations, usually filtered for correctness or consistency (Zelikman et al., 2022; Huang et al., 2023; Singh et al., 2024) and sometimes without any filter (Zhang et al., 2026; Agarwal et al., 2025). Our objective is sequence-level self-distillation, but its teacher is the same recurrent operator run for more passes rather than another model or an explicit reasoning trace, and the update lands in the operator that every pass reuses, so it improves inference at every depth.

Supervision across depth and training through partial computation. Intermediate depths are often supervised directly, through losses on hidden layers (Lee et al., 2015), deep-to-shallow self-distillation (Zhang et al., 2019) or early-exit losses (Elhoushi et al., 2024). Weight-tied iterative models are often trained through part of their computation: implicit models use implicit or approximate gradients, down to a single operator application (Bai et al., 2019; Geng et al., 2021; Fung et al., 2022); recurrent networks that extrapolate to more iterations (Schwarzschild et al., 2021) backpropagate through a random suffix (Bansal et al., 2022); and recurrent-depth language models truncate backpropagation (Geiping et al., 2025), as in truncated backpropagation through time (Williams & Peng, 1990). There, partial gradients buy cheaper or more stable training. We separate the depth that exposes behavior from the depth that learns it, and find that learning through a single pass is also more effective for deep inference.

5 Conclusion

Running a looped language model for more passes is usually treated as a way to spend inference compute. Our results show that those passes also produce a training signal. Deeper execution revises the preferences that the first pass already holds, and its greedy trajectories, generated without reference answers, can be written back into the shared recurrent block. The write-back works best through a single pass, and the updated block then helps the later passes that reuse it. Self-distillation raises four-pass GSM8K accuracy of the mean-depth-4 model from 56.18% to 62.70%, carries over to mean depths 8 and 32, and leaves 17 non-mathematical tasks essentially unchanged. Input injection is what gives the one-pass route this advantage.

AI use statement

In this work, we used generative AI tools for implementing methods. We have not used generative AI tools for research ideation, hypothesis formulation, theoretical or mathematical development, or data

generation. Translation and qualitative data analysis are not applicable to this work. Additionally, we used generative AI tools for coding assistance, figure preparation, identifying related work, and drafting and editing parts of the paper. We also used generative AI to sharpen existing argumentation and to critique and refine the design and interpretation of possible follow-up analyses that are not part of the experimental results reported in this submission.

We have reviewed all AI-assisted work. All research ideas are our own; we verified all AI-assisted code, text, and figures and checked every reported number against the underlying evaluation outputs. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

Ethics statement

This work does not involve human subjects or personal data, and it does not release new datasets. All experiments use publicly available pretrained models and datasets, and self-distillation trains each model only on its own outputs for questions drawn from these datasets. We are not aware of ethical concerns arising from this work beyond those common to language model research.

Reproducibility statement

We have made every effort to ensure that our results are reproducible. Section 2 specifies the looped architecture, the recurrent-depth training and the self-distillation objective, and the setup paragraph of Section 3 states the default self-distillation recipe. Appendix C gives the complete implementation details. It covers the architecture (C.1), mid-training (C.2), the self-distillation procedure with pseudocode in Algorithm 1 (C.3), the evaluation protocol (C.4) and compute (C.5). Appendices D and E describe the setup of each additional analysis and report the full results. All pretrained models and datasets we use are publicly available, and we will release our code upon publication.

References

Shivam Agarwal, Zimin Zhang, Lifan Yuan, Jiawei Han, and Hao Peng. The unreasonable effectiveness of entropy minimization in LLM reasoning. In *Advances in Neural Information Processing Systems*, volume 38, pp. 107150–107180, 2025. URL https://openreview.net/forum?id=UfFTBEsLgI.

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://openreview.net/forum?id=WwpYSOkkCt.

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. In *Advances in Neural Information Processing Systems*, volume 38, pp. 96572–96617, 2025b. URL https://openreview.net/forum?id=QuqsEIVWIG.

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

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20232–20242, 2022.

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. *arXiv preprint arXiv:2110.14168*, 2021.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. LayerSkip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, 2024. doi: 10.18653/v1/2024.acl-long.681.

Samy Wu Fung, Howard Heaton, Qiuwei Li, Daniel McKenzie, Stanley Osher, and Wotao Yin. JFB: Jacobian-free backpropagation for implicit networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 6648–6656, 2022. doi: 10.1609/aaai.v36i6.20619.

Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1607–1616. PMLR, 2018. URL https://proceedings.mlr.press/v80/furlanello18a.html.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, volume 38, pp. 41340–41391, 2025. URL https://openreview.net/forum?id=S3GhJooWIC.

Zhengyang Geng, Xin-Yu Zhang, Shaojie Bai, Yisen Wang, and Zhouchen Lin. On training implicit models. In *Advances in Neural Information Processing Systems*, volume 34, pp. 24247–24260, 2021.

Mor Geva, Avi Caciularu, Kevin Wang, and Yoav Goldberg. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 30–45, 2022. doi: 10.18653/v1/2022.emnlp-main.3.

Angeliki Giannou, Shashank Rajput, Jy-Yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 11398–11442. PMLR, 2023.

Yuling Gu, Oyvind Tafjord, Bailey Kuehl, Dany Haddad, Jesse Dodge, and Hannaneh Hajishirzi. OLMES: A standard for language model evaluations. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5020–5048, 2025.

Danny Halawi, Jean-Stanislas Denain, and Jacob Steinhardt. Overthinking the truth: Understanding how language models process false demonstrations. In *International Conference on Learning Representations*, 2024.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Itxz7S4Ip3.

Benhao Huang, Zhengyang Geng, and Zico Kolter. Equilibrium reasoners: Learning attractors enables scalable reasoning. In *International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=lh95PnOlpM.

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1051–1068, 2023. doi: 10.18653/v1/2023.emnlp-main.67. URL https://aclanthology.org/2023.emnlp-main.67/.

Stanisław Jastrzębski, Devansh Arpit, Nicolas Ballas, Vikas Verma, Tong Che, and Yoshua Bengio. Residual connections encourage iterative inference. In *International Conference on Learning Representations*, 2018.

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://openreview.net/forum?id=RzYXb5YWBs.

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

Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1317–1327, 2016. doi: 10.18653/v1/D16-1139. URL https://aclanthology.org/D16-1139/.

Harsh Kohli, Srinivasan Parthasarathy, Huan Sun, and Yuekun Yao. Loop, think, & generalize: Implicit reasoning in recurrent-depth transformers. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=8fz7WRThKL.

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

Vedang Lad, Jin Hwa Lee, Wes Gurnee, and Max Tegmark. Remarkable robustness of LLMs: Stages of inference? In *Advances in Neural Information Processing Systems*, volume 38, pp. 130050–130083, 2025.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1eA7AEtvS.

Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pp. 562–570. PMLR, 2015.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 158–167, 2017.

Sean McLeish, Ang Li, John Kirchenbauer, Dayal Singh Kalra, Brian R. Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Jonas Geiping, Tom Goldstein, and Micah Goldblum. Teaching pretrained language models to think deeper with retrofitted recurrence. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=PXVQTHYwgt.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 975–984, 2020.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-Symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

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

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2080–2094, 2021.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=din0lGfZFd.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? Generalizing from easy to hard problems with recurrent networks. In *Advances in Neural Information Processing Systems*, volume 34, pp. 6695–6706, 2021.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. CODI: Compressing chain-of-thought into continuous space via self-distillation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 677–693, 2025. doi: 10.18653/v1/2025.emnlp-main.36. URL https://aclanthology.org/2025.emnlp-main.36/.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron T Parisi, Abhishek Kumar, Alexander A Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Fathy Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura A Culp, Lechao Xiao, Maxwell Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=lNAyUngGFK. Expert Certification.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Allyson Ettinger, Michal Guerquin, David Heineman, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Jake Poznanski, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 OLMo 2 furious, 2025. URL https://arxiv.org/abs/2501.00656.

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. OpenMathInstruct-2: Accelerating AI for math with massive open-source instruction data. In *The Thirteenth International Conference on Learning Representations*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

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.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488, 2022.

Hugh Zhang, Jeff Da, Dean Lee, Vaughn Robinson, Catherine Wu, Will Song, Tiffany Zhao, Pranav Raja, Charlotte Zhuang, Dylan Slack, Qin Lyu, Sean Hendryx, Russell Kaplan, Michele Lunati, and Summer Yue. A careful examination of large language model performance on grade school arithmetic. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, volume 37, pp. 46819–46836, 2024.

Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3713–3722, 2019.

Ruixiang Zhang, Richard He Bai, Huangjie Zheng, Navdeep Jaitly, Ronan Collobert, and Yizhe Zhang. Embarrassingly simple self-distillation improves code generation. *arXiv preprint arXiv:2604.01193*, 2026.

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. *arXiv preprint arXiv:2510.25741*, 2025. URL https://arxiv.org/abs/2510.25741.

A Recurrence during mid-training

Before asking what additional recurrence contributes to a trained model, we check that recurrence pays off during mid-training itself. Figure 4 follows the Looped LMs of this paper on GSM8K and GSM-Symbolic as mid-training proceeds from the stage-1 checkpoint (Appendix C.2), as their gain over the non-looped control trained on the same 50B tokens. Every Looped LM finishes with a positive gain on both tasks, and on GSM8K each one crosses zero within 24B tokens, less than half of the control's training. Recurrent computation is therefore already an effective part of these models before any self-distillation; the remaining appendices and Sections 2 and 3 study what it adds and how to write it back into the shared block.

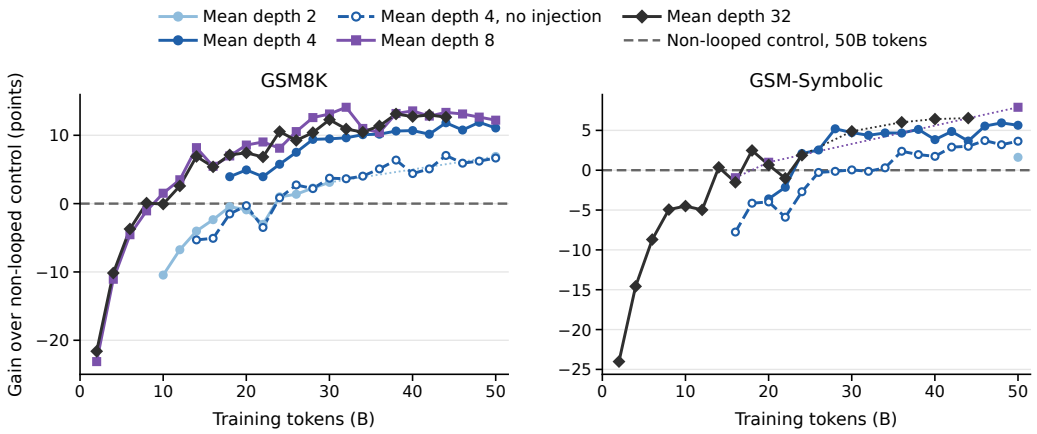

Figure 4: Gain in exact match (points) over the non-looped control after 50B tokens (dashed line at zero) during mid-training, for the Looped LMs of this paper, each evaluated at a fixed depth (its mean depth; eight passes for the mean-depth-32 model). Dotted segments join checkpoints more than 4B tokens apart.

B Recurrence primarily reweights the top of the distribution

Reading out along a depth-4 trajectory. We analyze the mean-depth-4 Looped LM of Table 1. During an ordinary greedy generation at depth 4, we additionally decode the state s_k after each pass $k = 1, \ldots, 4$ with the same output stack and language-model head, giving the intermediate readouts

$$q_k(\cdot \mid x, \hat{y}_{<t}) = \mathrm{softmax}(W[C(s_k)]_n)\,, \qquad k = 1, \ldots, 4.$$

The readouts are diagnostic only: each decodes from a copy of the output-stack cache, none feeds back into the recurrence, and tokens are selected from q_4 alone, so the generation is exactly the ordinary depth-4 generation. Because every q_k is conditioned on the depth-4 trajectory, and its output stack attends to the depth-4 states of earlier positions, it describes an intermediate state of that execution and differs in general from p_k, the distribution of an independent execution of depth k (Section 2.1). Figure 5 summarizes how the readouts change from pass to pass.

Recurrence primarily reshapes the head of the distribution. Additional passes mainly reshape the relative probabilities of candidates already near the top after the first pass, rather than promote

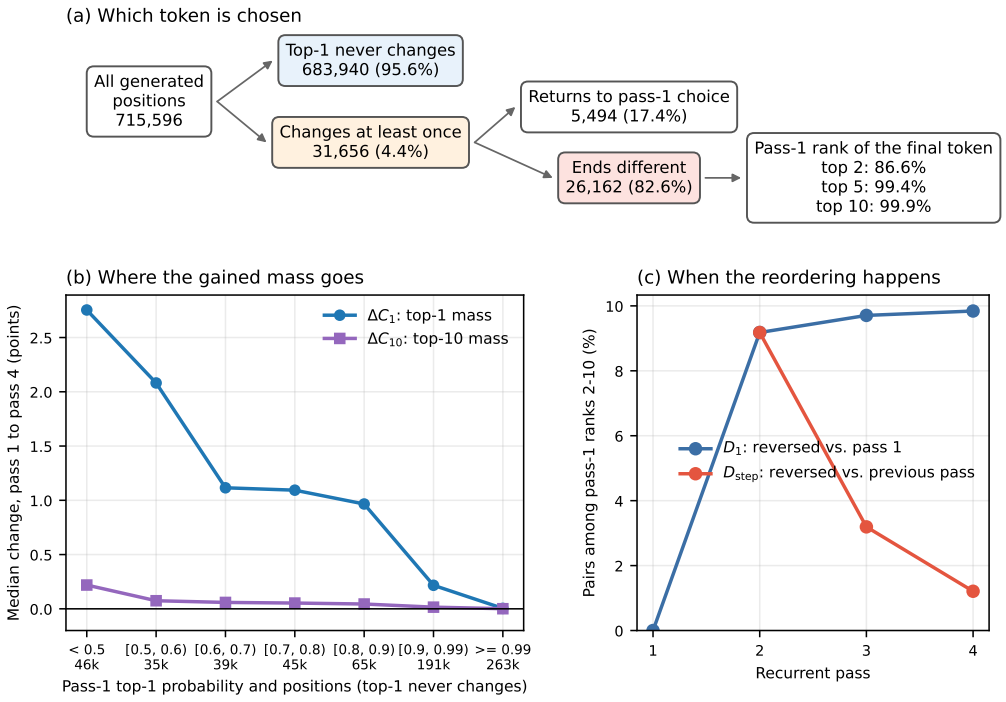

Figure 5: Effect of passes 2–4 on the readout distribution over 715,596 generated positions of GSM8K training problems. (a) Stability of the top-1 token across passes, and the pass-1 rank of the final token when it differs; each percentage is relative to the box it branches from. (b) Median change from q_1 to q_4 in top-1 probability (ΔC_1) and top-10 mass (ΔC_{10}), binned by the pass-1 top-1 probability, at positions whose top-1 never changes. (c) Pairwise reordering among the pass-1 ranks 2–10, relative to pass 1 (D_1) and to the preceding pass (D_{step}).

previously low-ranked candidates. The effect varies in magnitude: at many positions the probabilities change while the ordering is preserved, at others nearby candidates are reordered, and only occasionally does the top-1 prediction change.

We apply this diagnostic to all 7,473 GSM8K training problems under greedy depth-4 generation, yielding 715,596 generated-token positions. (a) The top-1 prediction is stable at most positions: it is the same at all four passes at 95.6% of positions, and when the final top-1 token differs from the pass-1 choice, it was already among the pass-1 top five 99.4% of the time (Figure 5a); (b) Where the top-1 remains fixed, recurrence mainly redistributes probability within the head of the distribution: the top-1 probability increases while the total top-10 mass changes little, with the largest shifts where the first pass is uncertain and the effect vanishing as it becomes confident (Figure 5b); (c) When recurrence does reorder nearby candidates, most of that change occurs early: pass 2 already accounts for 93% of the pairwise reordering among the pass-1 ranks 2–10 observed by pass 4, while later passes make progressively smaller changes (Figure 5c).

From readout to supervision. What the additional passes contribute is therefore mainly a sharper preference among continuations that are already plausible after one pass, rather than a new candidate. Greedy decoding at the deeper execution turns this preference into a complete continuation, and these continuations are the self-distillation targets of Section 2.2.

Relation to layer-wise readouts. Decoding intermediate states with the model's own output head is a common probe of residual networks, whose layers refine a prediction iteratively (Jastrzębski et al., 2018): feed-forward updates promote candidate tokens in vocabulary space (Geva et al., 2022), late layers sharpen the next-token distribution (Lad et al., 2025), and intermediate predictions can be more accurate than final ones (Halawi et al., 2024). In a looped model every pass shares one

state space and one operator, so the same output stack reads out each pass directly, without learned per-layer probes.

C IMPLEMENTATION DETAILS

This appendix gives the implementation behind Sections 2 and 3: the looped architecture (Appendix C.1), the mid-training that produces each Looped LM (Appendix C.2), the self-distillation recipe (Appendix C.3), and the evaluation protocol (Appendix C.4).

Notation in the tables. The tables name the three depths of Section 2.3 directly: *Teacher* is the teacher depth r_{T}, *Training* is the training depth r_{L}, and the *Depth 1* and *Mean depth* columns give the inference depth r_{E}. The mean depth m is the target of the depth curriculum during mid-training (Appendix C.2), and it is also the inference depth at which each Looped LM is used.

C.1 ARCHITECTURE

Partition of OLMo-2 1B. OLMo-2 1B has 16 Transformer blocks of width $d = 2048$ with 16 attention heads (Team OLMo et al., 2025); its released configuration specifies a SwiGLU feed-forward layer of width 8,192 and a vocabulary of 100,352 tokens. The input stack P consists of blocks 1–7, the shared recurrent core F_θ, called the shared recurrent block in Section 3, of blocks 8–11, and the output stack C of blocks 12–16. F_θ is a single module invoked once per recurrent pass, so an execution of depth r applies the same 268,468,224 parameters r times; the complete model has 1.49B parameters.

Input injection and the initial state. The injection projection A_ψ is one bias-free linear map from $\mathbb{R}^{2d}$ to $\mathbb{R}^d$, applied at each position, with no gate, normalization or activation. The same input-stack output e is concatenated with the recurrent state at every pass. During training, the initial state s_0 is drawn from a truncated normal distribution with standard deviation 0.0087. At evaluation, s_0 is drawn with a seed derived from the prompt tokens, so executions of different depth on the same prompt start from the same state.

Model without input injection. The model without input injection in Table 4 removes both the concatenation and the separate initial state:

$$s_0 = e, \qquad s_i = F_\theta(s_{i-1}), \qquad i = 1, \ldots, r.$$

It is therefore a different recurrence rather than injection with a zeroed projection. Its partition, output stack and mid-training procedure match those of the injected mean-depth-4 model.

C.2 MID-TRAINING OF THE LOOPED LMS

Common setup. Every Looped LM and the non-looped control start from the OLMo-2 1B stage-1 checkpoint and continue causal language-model training on the Dolmino mixture (Team OLMo et al., 2025) for up to 50B tokens, with context length 4,096, a global batch of 2,097,152 tokens (512 sequences), the same data order, BF16 precision and fully sharded data parallelism. Table 7 lists the optimizer and the gradient path of each model. All models use a peak learning rate of 7.4×10^{-5}, decayed linearly to zero over 23,842 steps without warmup, $\beta = (0.9, 0.95)$, weight decay 0.033, gradient clipping at norm 1.0 and a z-loss coefficient of 10^{-5}. Muon updates the matrix parameters with its learning rate rescaled to the update scale of AdamW, and AdamW updates the embeddings, normalization parameters and language-model head.

Depth sampling and curriculum. At each training step, one depth r is drawn for the whole batch from the Poisson-lognormal distribution of Geiping et al. (2025), with log-normal scale $\sigma = 0.5$ and parameterized by the current curriculum mean μ_t of Section 2.1. Over the first 75% of the T training steps, μ_t rises from 1 to the target mean depth m along a one-minus-square-root schedule, and it is held at m afterwards:

$$\mu_t = \max\Big(1,\ \Big\lceil m\left(1 - \sqrt{1 - \min(t/t_{\mathrm{w}},\, 1)}\right)\Big\rceil\Big), \qquad t_{\mathrm{w}} = 0.75\,T.$$

Table 7: Mid-training of the Looped LMs and the non-looped control. *Backpropagation* gives the passes that carry gradient at each training step.

Model	Input injection	Optimizer	Backpropagation
Non-looped control	✗	AdamW	single fixed pass
Mean-depth-2 Looped LM	✓	Muon	every pass
Mean-depth-4 Looped LM	✓	Muon	every pass
Mean-depth-4 Looped LM, no injection	✗	Muon	every pass
Mean-depth-8 Looped LM	✓	AdamW	last eight passes
Mean-depth-32 Looped LM	✓	AdamW	last eight passes

Gradient path through the recurrence. For the mean-depth-8 and mean-depth-32 models, all passes before the last eight of a sampled depth run without gradient, so backpropagation covers at most eight passes, following Geiping et al. (2025); this bounds the length of the gradient path through the recurrence and keeps optimization stable at large depth. Because e is injected at every pass, the input stack still receives gradient through the differentiated passes.

Non-looped control. The non-looped control of Tables 1 and 6 is trained with the same data and setup, but with a single fixed recurrent pass and no input injection. With $s_0 = e$ and one pass, it computes $C(F_\theta(P(x)))$, which is the original 16-block OLMo-2 architecture. Appendix A follows the accuracy of the Looped LMs through this stage.

C.3 Self-distillation

Teacher trajectories. The teacher is the frozen Looped LM before the update. For each training question it generates one continuation at teacher depth r_{T} by greedy decoding, using the fixed eight-shot chain-of-thought prompt of OLMES for GSM8K (Gu et al., 2025; Team OLMo et al., 2025) and at most 512 new tokens. Generation ends at the first stop string or end-of-sequence token. The generated token ids, including the stop token, are the target without further editing; a continuation that reaches the token limit is kept as generated. Targets are generated once and reused in every epoch, and the reference answer of the training question is never used. The sampled targets in Table 5 apply temperature $T = 0.7$ and then top-k truncation with $k = 10$ at each step.

Objective and updated parameters. A training sequence is the eight-shot prompt followed by the target continuation. The loss is the completion-only next-token cross-entropy of Section 2.3, normalized by the total number of target tokens in the batch; no distillation term on the teacher probabilities is added. Only the 32 parameter tensors of F_θ are updated. The embeddings, input stack, injection projection A_ψ, output stack and language-model head are frozen, leaving 18% of the parameters trainable.

Algorithm 1 summarizes the procedure.

Training depth. Training at depth r_{L} runs r_{L} recurrent passes forward, reads the loss out after the last pass, and backpropagates through all r_{L} passes; because F_θ is shared, the gradient accumulates contributions from each of its uses. The $4^\dagger$ row of Table 3 runs the first three passes without gradient and backpropagates through the fourth only. For the depth-8 and depth-32 training rows of Table 3, activations are recomputed pass by pass to fit in memory, which leaves the gradient unchanged.

Optimization. Table 8 lists the optimization settings, which are shared by every self-distillation run in the paper. Runs differ only in the teacher depth, training depth, question pool and target decoding stated in the corresponding table.

Question pools. The default pool contains question text from the `augmented_gsm8k` portion of OpenMathInstruct-2 (Toshniwal et al., 2025). Only the problem statements are read; the released solutions and answers are not used. Questions are normalized for Unicode and whitespace, deduplicated, and screened against the GSM8K test, GSM1K and SVAMP questions and the development questions of Appendix C.4. A lexical audit against these 4,214 questions (word five-gram Jaccard

Algorithm 1 Preference write-back by self-distillation

Input: Looped LM with shared core θ_0 and remaining parameters ϕ; prompts $\{x^{(j)}\}_{j=1}^{N}$;
teacher depth r_{T}; training depth r_{L}; number of epochs E

Expose: the frozen model generates targets at teacher depth
for $j = 1, \ldots, N$ **do**
  $\hat{y}^{(j)} \leftarrow$ greedy continuation of $x^{(j)}$ under $p^{(0)}_{r_{\mathrm{T}}}$, at most 512 tokens, kept unfiltered
$\mathcal{D}_{r_{\mathrm{T}}} \leftarrow \{(x^{(j)}, \hat{y}^{(j)})\}_{j=1}^{N}$
Incorporate: update only the shared core through r_{L} passes
$\theta \leftarrow \theta_0$
for epoch $= 1, \ldots, E$ **do**
  for each batch $B \subset \mathcal{D}_{r_{\mathrm{T}}}$ of 16 sequences **do**
    $\theta \leftarrow \mathrm{AdamW}\left(\theta, \nabla_\theta \mathcal{L}_{r_{\mathrm{L}}}(\theta; B)\right)$ *completion-only cross-entropy; ϕ frozen*
Reuse: the updated core serves every pass at any inference depth
return $\theta_1 \leftarrow \theta$, used with ϕ at inference depth r_{E}

Table 8: Self-distillation settings shared by all runs, and the size of the OpenMath question pools.

Setting	Value
Updated parameters	shared recurrent block F_θ (268M of 1.49B)
Objective	completion-only next-token cross-entropy
Optimizer	AdamW, $\beta = (0.9, 0.95)$, no weight decay
Learning rate	3×10^{-6}; 10 linear warmup steps, then cosine decay
Batch	16 sequences
Gradient clipping	global norm 1.0
Epochs	3 (OpenMath pools), 6 (GSM8K training questions)
Precision	BF16 computation, FP32 parameters
Targets	one teacher continuation per question, at most 512 tokens
OpenMath question pools (targets of the mean-depth-4 model)	
10,000 questions	992,459 target tokens per epoch
10,000 questions, curated	1,019,832 target tokens per epoch
30,000 questions	2,973,623 target tokens per epoch
50,000 questions	4,953,856 target tokens per epoch

Table 9: Mathematical evaluation sets. All prompts are fixed chain-of-thought demonstrations shared by every question of a set.

Benchmark	Questions	Shots	Token limit
GSM8K test (Cobbe et al., 2021)	1,319	8	512
GSM1K (Zhang et al., 2024)	1,205	8	512
SVAMP (Patel et al., 2021)	1,000	8	512
GSM-Symbolic, main (Mirzadeh et al., 2025)	5,000	8	1,024
ASDiv (Miao et al., 2020)	2,305	8	512
AQuA-RAT (Ling et al., 2017)	254	4	512

similarity of at least 0.8, or numeric-template similarity of at least 0.9) finds no near match in the final pools. The 10,000-, 30,000- and 50,000-question pools are nested. The curated 10,000-question pool of Table 5 is selected from the 50,000-question pool using the question text alone: it matches the length distribution of the random 10,000-question pool, favors rare problem types, and keeps at most one question from each cluster of near-duplicates, and it reuses the targets already generated for those questions. The GSM8K pool consists of all 7,473 training questions (Cobbe et al., 2021).

Checkpoint selection. Each run is trained for a fixed number of epochs, and a single checkpoint per run, chosen by greedy accuracy at the model's mean depth, is used for every evaluation.

C.4 Evaluation protocol

Mathematical reasoning. Table 9 lists the mathematical evaluation sets. Every model is evaluated with greedy decoding in BF16 at depth 1 and at its mean depth. Generation stops at the first stop string or end-of-sequence token, and a response that reaches the token limit remains in the denominator. For GSM8K, GSM1K and SVAMP the answer is the number after the last `####` if present and otherwise the last number in the response, normalized for commas, trailing zeros and sign; GSM-Symbolic uses the answer extraction of the OLMES GSM8K task; for ASDiv it is the number after "The final answer is", falling back to the last number; for AQuA-RAT it is the last explicitly labelled option letter. Accuracy is exact match of the extracted answer with the reference answer.

GSM8K and GSM-Symbolic use the OLMES eight-shot prompt (Gu et al., 2025; Team OLMo et al., 2025); GSM1K and SVAMP use the eight arithmetic chain-of-thought demonstrations of Wei et al. (2022), and ASDiv uses the same eight demonstrations in a problem–solution format. AQuA-RAT uses four fixed demonstrations with their rationales, drawn from its development split. The non-mathematical tasks of Tables 6 and 12 are evaluated with the OLMES harness at each model's mean depth (depth 1 for the non-looped control), with the shots and metrics listed in Table 12.

Summary columns. Relative changes in the tables are per-task changes divided by the reference value, the Looped LM before self-distillation or, where stated, the non-looped control. The *Avg.* columns of Tables 1 and 3–5 are unweighted means of these per-task relative changes. In Table 6, each group score is the mean of its task scores and its percentage the mean of the per-task relative changes.

Development set and intervals. The development set consists of 690 questions from the GSM8K training split and is used only to select checkpoints. The OpenMath pools are screened against it as described above. Intervals are 95% question-paired bootstrap intervals over 10,000 resamples of the evaluation questions. Table 10 gives them for the comparisons of Table 1: every Looped LM is above the non-looped control on all six tasks, and the gains of self-distillation on GSM8K and GSM-Symbolic exclude zero for all three models.

C.5 Compute

Mid-training runs on NVIDIA A100 80GB GPUs, and self-distillation, including the generation of teacher trajectories, on NVIDIA A800 80GB GPUs. Self-distillation is small next to mid-training.

Table 10: Paired differences in exact match (points) behind Table 1, at each model's mean depth, with 95% question-paired bootstrap intervals. The non-looped control is evaluated at depth 1.

Comparison	GSM8K	GSM1K	SVAMP	GSM-Symbolic	ASDiv	AQuA-RAT
+ Self-distillation minus Looped LM						
Mean depth 4	+6.5 [4.2, 8.8]	+3.2 [1.1, 5.5]	+1.4 [−1.1, 3.9]	+4.1 [3.0, 5.3]	+1.7 [0.5, 3.0]	+1.2 [−3.9, 6.3]
Mean depth 8	+6.5 [4.4, 8.6]	+3.5 [1.2, 5.8]	−0.8 [−3.0, 1.4]	+3.4 [2.3, 4.5]	+1.7 [0.5, 3.0]	−1.2 [−6.7, 4.3]
Mean depth 32	+5.1 [2.7, 7.6]	+1.0 [−1.6, 3.6]	−2.0 [−4.6, 0.6]	+3.2 [1.9, 4.4]	–	+5.5 [0.0, 11.0]
Looped LM minus non-looped control						
Mean depth 4	+12.1 [9.6, 14.6]	+11.3 [8.7, 13.8]	+6.8 [4.0, 9.7]	+6.1 [4.9, 7.3]	+5.4 [3.9, 6.9]	+6.7 [2.0, 11.8]
Mean depth 8	+12.3 [9.8, 14.8]	+12.7 [10.2, 15.2]	+9.5 [6.7, 12.3]	+7.6 [6.4, 8.9]	+6.8 [5.3, 8.3]	+13.4 [7.9, 19.3]
Mean depth 32	+13.0 [10.4, 15.5]	+13.5 [11.0, 16.2]	+11.2 [8.3, 14.2]	+7.5 [6.2, 8.7]	+7.2 [5.7, 8.7]	+9.1 [3.5, 15.0]

Even at teacher depth 32, generating the trajectories for 10,000 questions takes 18 GPU-hours, and three epochs of training on them at depth 1 take 2 GPU-hours.

D Further evidence on training depth

This appendix supports Sections 3.2 and 3.3. It details the gradient measurements of Figure 3 (Appendix D.1), follows the two training depths across inference depths and model families (Appendix D.2), and gives the full record of the pass-wise block replacement of Table 2 (Appendix D.3).

D.1 Recurrent pass-gradient diagnostic

Setup. Both panels of Figure 3 use the mean-depth-4 Looped LM before self-distillation, the point from which training at depth 1 and training at depth 4 start with the same parameters and the same targets. We take 64 fixed teacher trajectories from the OpenMath question pool (6,892 target tokens) and evaluate the completion-only loss of Section 2.3 in FP32.

Panel (a): gradient at the same first-pass state. For each trajectory, let s_1 be the state after the first recurrent pass and L_k the loss read out after k passes. We compute $g_k = \partial L_k / \partial s_1$ over the full state tensor for $k = 1, \ldots, 4$; the state s_1 is verified to be identical across the four depths. The panel plots the cosine between g_k and the one-pass gradient g_1, and the ratio $\|g_k\|/\|g_1\|$, each computed per trajectory and averaged over the 64 trajectories; error bars are 95% bootstrap intervals over trajectories. Most of the change occurs at the first additional pass: with two passes, the cosine is already 0.10 and the norm 32% of the one-pass gradient.

Panel (b): parameter gradient by pass. To attribute the gradient of four-pass training to individual uses of the shared block, we untie the block into four copies with identical weights, one per pass. Their gradients $g^{(1)}, \ldots, g^{(4)}$ sum to the gradient of the shared parameters, with a maximum relative error of 1.3×10^{-6}. The panel shows the share $\|g^{(i)}\| / \sum_j \|g^{(j)}\|$ for each pass, computed on four groups of 16 trajectories, the batch size of self-distillation, and averaged over the groups.

Together, the two panels suggest that training at depth 1 applies the direct signal to the first use of the shared block, whereas training at depth 4 reaches it only as a small, rotated remnant and updates the block mainly through its uses on late recurrent states.

D.2 Training depth across inference depths and models

Figure 6 follows each injected Looped LM and its two self-distilled models across the inference depths we evaluated, for mean depths 2, 4, 8 and 32. On GSM8K and GSM1K, training at depth 1 improves the model at every inference depth in all four families, and the gain from recurrence is retained: the mean-depth-4 model self-distilled at depth 1 rises from 48.52% on GSM8K at one pass to 59.06% at two and 62.70% at four. The gain at one pass is largest where that execution starts weakest: the mean-depth-32 model moves from 4.09% to 31.84% at one pass while also improving at 8, 16 and 32 passes. Training on the same targets through the full mean depth ends below training at depth 1 at the mean depth on both sets in every family.

The mean-depth-2 model repeats the controlled comparison of Table 3 on a second injected checkpoint, mid-trained toward mean depth 2: the teacher generates one greedy trajectory at depth 2 for each of the 10,000 OpenMath questions, and the same targets train the shared block through one pass or through two, with all other settings as in Appendix C.3. At the deployed depth, training at depth 1 is ahead by 4.2, 2.3 and 3.8 points on GSM8K, GSM1K and SVAMP, and every 95% CI of these paired differences excludes zero. Training at depth 1 is also the cheaper update: on the same trajectories and hardware, three epochs of the mean-depth-4 model take 0.72 GPU-hours at depth 1 against 1.14 at depth 4, with 14% less peak memory.

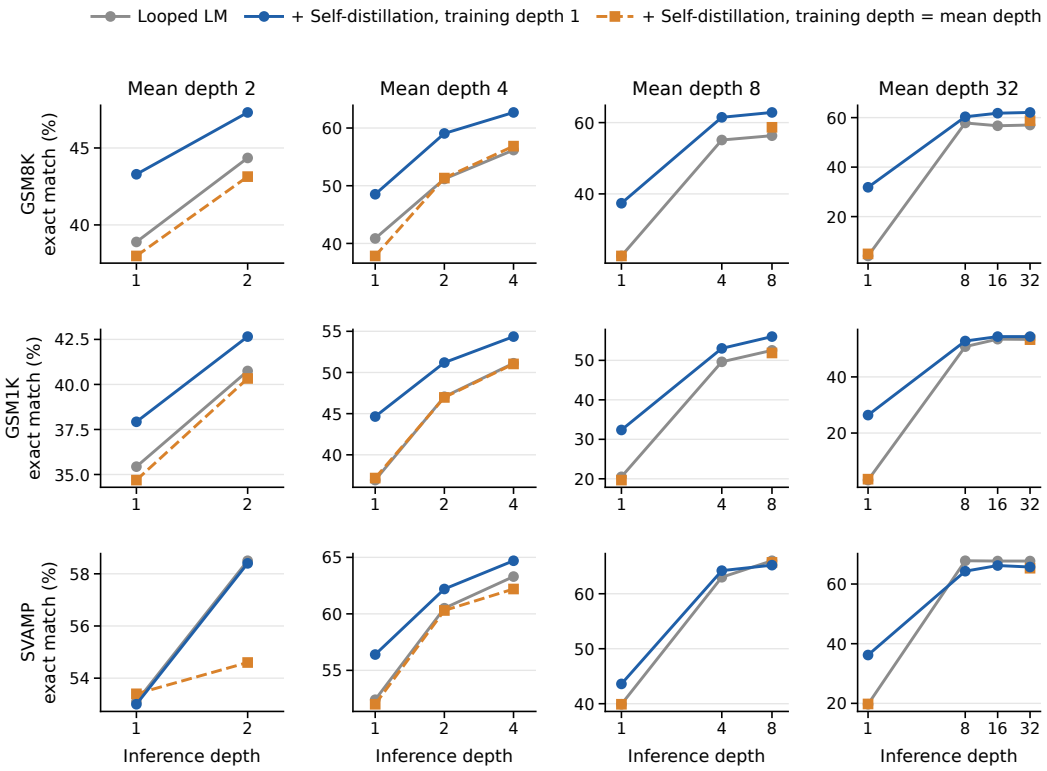

Figure 6: Exact match (%) across inference depths for four injected Looped LMs (columns) on GSM8K, GSM1K and SVAMP (rows). Each Looped LM is self-distilled from its own greedy mean-depth trajectories through one pass or through its full mean depth; lines connect adjacent evaluated depths. Inference depth is on a logarithmic axis, and the vertical range differs between panels. The values at depth 1 and at the mean depth of the mean-depth-4, 8 and 32 models are those of Table 3.

D.3 Pass-wise block replacement

The intervention of Table 2 executes the mean-depth-4 Looped LM for four passes and chooses, for each pass, either the original shared block or the block of the self-distilled model of Table 1. All other parameters are shared by the two models; only the 32 tensors of the recurrent block differ. Each pattern is a complete execution: generation and the cache are rebuilt under that pattern, so the comparison locates where the updated block is needed rather than splitting the gain into additive per-pass contributions. On eight check examples, the two pure patterns reproduce ordinary inference of the two models exactly, in generated tokens, next-token logits and cache contents.

Table 11 gives the record of the four patterns on GSM8K. Using the updated block on passes 2–4 recovers 97% of the gain obtained by using it throughout. Using it at the first pass alone does change the model's answers, but symmetrically: as many questions become correct as become incorrect, and accuracy does not move.

Table 11: Pass-wise replacement of the shared block on GSM8K (1,319 questions) at four-pass inference. ✓: the self-distilled block is used at that pass; ✗: the original block. Change is in points relative to the original model; repaired / broken count the questions that become correct / incorrect.

Pass							
1	2	3	4	Accuracy (%)	Change	95% CI	Repaired / broken
✗	✗	✗	✗	56.18	–	–	–
✓	✗	✗	✗	56.25	+0.1	[−1.2, 1.4]	39 / 38
✗	✓	✓	✓	62.47	+6.3	[4.0, 8.6]	164 / 81
✓	✓	✓	✓	62.70	+6.5	[4.2, 8.8]	165 / 79

E RETENTION BEYOND MATHEMATICS

Table 12 gives the per-task scores behind Table 6. Self-distillation leaves individual tasks close to where they were: 47 of the 51 task scores of the three self-distilled models move by less than one point, and the mean-depth-8 model stays within one point on every task. The advantage that looping gives over the non-looped control is preserved as well: after self-distillation, each Looped LM remains above the control on at least 13 of the 17 tasks.

Table 12: Scores (%) on the 17 non-mathematical OLMES tasks, grouped as in Table 6: the non-looped control at depth 1 and, for each Looped LM, the model before and after self-distillation at its mean depth. The self-distilled models are those of Table 1; shots and metrics follow the OLMES task definitions (Appendix C.4), and AGIEval is evaluated in both its 3-shot and its 1-shot configuration.

			Control	Mean-depth-4		Mean-depth-8		Mean-depth-32	
Task	Shots	Metric	Depth 1	Looped LM	+ Self-dist.	Looped LM	+ Self-dist.	Looped LM	+ Self-dist.
Knowledge									
MMLU	5	accuracy	43.92	48.17	48.44	50.14	50.43	49.69	49.66
MMLU-Pro	5	accuracy	15.33	16.61	16.99	18.54	18.28	18.38	17.91
Exams									
AGIEval	3	accuracy	36.10	40.02	40.07	42.67	42.69	42.58	41.83
AGIEval	1	accuracy	35.28	39.92	40.07	40.63	40.54	41.14	40.20
Commonsense									
CommonsenseQA	5	accuracy	66.50	66.42	66.18	67.08	67.49	66.67	67.08
HellaSwag	5	accuracy	70.00	70.60	70.70	70.00	70.10	70.10	70.10
PIQA	5	accuracy	75.50	75.20	74.90	74.30	75.20	74.90	74.80
WinoGrande	5	accuracy	66.61	67.96	67.72	68.27	68.35	68.90	68.75
Science QA									
ARC-Challenge	5	accuracy	51.79	59.13	59.13	60.32	59.81	61.26	59.81
ARC-Easy	5	accuracy	78.60	80.50	79.60	81.70	82.00	80.80	80.40
OpenBookQA	5	accuracy	53.40	58.40	59.00	59.80	60.60	59.00	59.60
Reading									
CoQA	0	F1	67.14	74.50	74.10	68.05	68.28	66.59	64.44
SQuAD	5	F1	82.41	87.31	88.28	87.56	87.45	88.40	88.16
DROP	5	F1	34.72	39.80	41.46	39.21	40.04	35.69	36.94
Open QA									
Jeopardy	5	F1	63.81	65.42	65.54	64.83	64.88	64.59	65.50
NaturalQuestions	5	F1	20.84	20.83	21.48	18.77	19.02	18.77	19.28
TriviaQA	5	F1	55.02	56.11	55.75	55.36	55.39	54.32	54.34