T^{2}MLR: TRANSFORMER WITH TEMPORAL MIDDLE-LAYER RECURRENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformer reasoning is limited by autoregressive decoding, which repeatedly compresses rich hidden computation through token space and makes it difficult for intermediate reasoning states to persist across time. We introduce Transformers with Temporal Middle-Layer Recurrence (T^{2}MLR), a Transformer-based latent reasoning architecture that fuses a cached middle-layer representation from the previous token directly into an earlier layer of the current token position, enabling abstract intermediate computation to persist across decoding steps with little inference overhead. Across natural-language pretraining and multi-hop reasoning finetuning, middle-layer T^{2}MLR variants outperform data- and parameter-matched Transformer baselines in every setting we test, with gains that persist when scaling to 1B parameters or 50B pretraining tokens. Moreover, applying recurrence to only a localized middle-layer block (as little as 20% of the network) often outperforms full-layer recurrence. Importantly, T^{2}MLR does not require pretraining from scratch: retrofitting the recurrent pathway into an existing pretrained 1.7B Transformer and briefly finetuning substantially improves math reasoning, lowering the barrier to practical adoption. These results suggest that effective latent reasoning in Transformers does not require recurrence through all layers, as in prior work, but can instead emerge more strongly from targeted middle-layer recurrence.

1 INTRODUCTION

Large language models now solve challenging multi-step problems in mathematics and science (Wei et al., 2022; Cobbe et al., 2021; OpenAI, 2023), which requires intermediate reasoning states to be refined over many steps before an answer is produced. Yet the Transformer (Vaswani et al., 2017) remains token-centric: at each decoding step, rich high-dimensional hidden states are projected onto a single sampled token. Attention lets the next token read earlier states only layer by layer, so the deep representations computed for one token never reach the shallow layers of the next except through that token. This deep-to-shallow bottleneck limits how latent reasoning states can persist across time.

Latent reasoning methods relax this bottleneck by letting computation persist in continuous space, either by reasoning directly with hidden states (Hao et al., 2024; Shen et al., 2025) or by feeding back mixtures of token embeddings (Zhang et al., 2025; Zhuang et al., 2025; Yue et al., 2025; Tang et al., 2026). These methods route the recurrent signal from the final layer back through the input embedding, so it bypasses the middle layers, where abstract reasoning is most prominent (Tenney et al., 2019; Geva et al., 2021; Meng et al., 2022; Saunshi et al., 2024; Atanas & Liu, 2025). Looped Transformers instead add depth, re-applying layers to each token several times (Saunshi et al., 2025; Geiping et al., 2025; Zhu et al., 2025b), so their decoding cost grows with the number of loops.

We introduce Transformers with Temporal Middle-Layer Recurrence (T^{2}MLR),[1] which, instead of confining recurrence to the token or embedding space or adding inference-time depth by looping, lets abstract representations computed in the middle layers persist and evolve across decoding steps. A gated pathway injects the representation from a deeper layer at the previous token into an earlier layer at the current token. T^{2}MLR keeps standard autoregressive decoding with one pass per token, dense token-level supervision, and nearly the inference cost of a standard Transformer.

[1] Code is included in the supplementary material and will be released publicly upon publication.

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

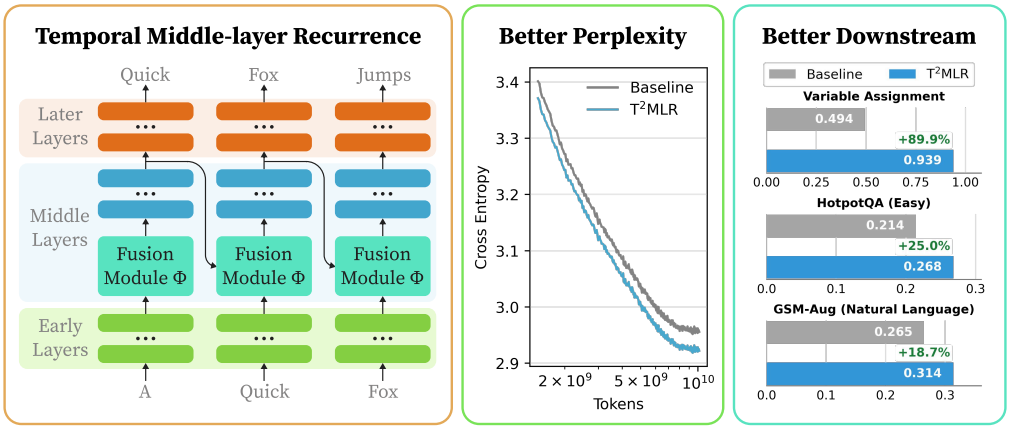

Figure 1: T^{2}MLR fuses the representation from a deep layer at the previous token into a shallow layer at the current token (left). It improves pretraining perplexity (middle; Section 4.2) and downstream reasoning (right, with relative gains over the baseline; Section 4.4).

A central practical challenge for latent-reasoning architectures is scalable teacher-forced training: recurrent dependencies across decoding steps break standard sequence-parallel training, making pretraining and finetuning much less scalable. We address it with an approximate temporal-parallel scheme (Section 2.4), so that T^{2}MLR can be pretrained and finetuned with dense teacher forcing under scalable sequence parallelism.

On data- and parameter-matched comparisons, our main findings are:

- Shallow T^{2}MLR reaches near-perfect exact match within the training lengths on S5-RETRIEVAL, a synthetic task that combines non-solvable group state tracking with in-context retrieval, where Transformers and recurrent models each fail (Section 3.2).
- T^{2}MLR improves pretraining perplexity and zero-shot accuracy over the Transformer baseline, with the best results from recurrence over only 20% of the layers (Section 4.2). Gains persist at 361M and 1B parameters and 50B tokens, where the 361M model raises the zero-shot average from 52.8 to 54.8 (Section 4.1), with at most $\sim$8% per-token inference overhead (Section B.5).
- Finetuned on GSM-Aug (Deng et al., 2023), ProsQA (Hao et al., 2024), HotpotQA (Yang et al., 2018), and variable assignment (Saunshi et al., 2024), T^{2}MLR outperforms baselines, and its $D = 22, 14$ middle-layer variants outperform full-layer recurrence on all tasks (Section 4.4).
- T^{2}MLR can be *retrofitted*: adding the recurrent pathway to a pretrained SmolLM2-1.7B-Instruct and briefly finetuning improves GSM8K from 35.8 to 39.9 and MATH500 from 12.8 to 18.0 over identical finetuning without it (Section 4.5).

2 TRANSFORMER WITH TEMPORAL MIDDLE-LAYER RECURRENCE

2.1 MOTIVATION: MIDDLE LAYERS AND THE INFORMATION BOTTLENECK

Mechanistic studies find that middle layers are the primary locus of abstract reasoning in Transformers, while early layers handle lexical processing and late layers project onto the vocabulary (Tenney et al., 2019; Geva et al., 2021; Meng et al., 2022; Saunshi et al., 2024; Atanas & Liu, 2025). This suggests that the most valuable reasoning-related computation happens in the middle layers, and that the representations they produce encode rich information about the model's current reasoning state.

Yet in a decoder-only Transformer, the shallow layers of the current token cannot directly read the deep representations already computed for the previous token: that information must either be rebuilt through the same depth and retrieved via attention, or squeezed through the unembed–sample–embed bottleneck. As a result, an evolving state is pushed ever deeper into the layer stack at each step and becomes unavailable to shallow layers (Mozer et al., 2026a). To relax this bottleneck without changing the autoregressive interface, T^{2}MLR adds a lightweight recurrent pathway that exposes this cached representation as an explicit input at the next step.

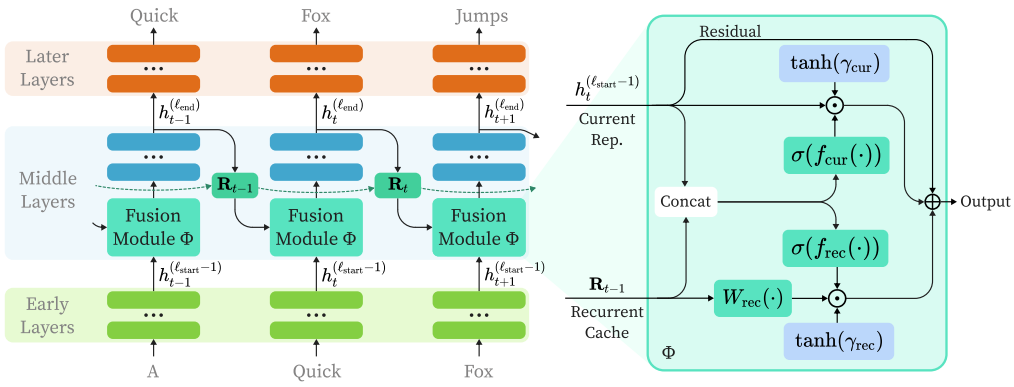

Figure 2: The T^{2}MLR architecture (left) and the representation fusion module Φ (right). At token t, the representation after layer ℓ_{end} updates the recurrent cache $\mathbf{R}_t$; at token $t+1$, $\mathbf{R}_t$ is fused with the representation before layer ℓ_{start}.

2.2 Temporal Middle-Layer Recurrence

We start from a standard L-layer decoder-only Transformer with hidden dimension d, and work at the level of Transformer blocks with KV caches (Ott et al., 2019), abstracting each token's key and value at each layer as a single cache vector in $\mathbb{R}^{d_{\text{KV}}}$. Let $\boldsymbol{h}_t^{(0)} \in \mathbb{R}^d$ be the input embedding of the t-th token x_t, and let $\mathbf{KV}_{1:t-1}^{(\ell)} \in \mathbb{R}^{(t-1)\times d_{\text{KV}}}$ be the KV cache of layer $\ell \in [L]$ up to step $t-1$. The ℓ-th block maps the current representation and past cache to an updated representation and cache:

$$\left(\boldsymbol{h}_t^{(\ell)}, \mathbf{KV}_{1:t}^{(\ell)}\right) = \mathcal{F}_\ell\left(\boldsymbol{h}_t^{(\ell-1)}, \mathbf{KV}_{1:t-1}^{(\ell)}\right), \qquad \ell = 1, 2, \ldots, L. \tag{2.1}$$

Temporal middle-layer recurrence uses layer indices $1 \le \ell_{\text{start}} \le \ell_{\text{end}} \le L$ and a learnable fusion module $\Phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$. A constant-size *recurrent cache* $\mathbf{R}_t \in \mathbb{R}^d$ is written after layer ℓ_{end} at step t and read before layer ℓ_{start} at step $t+1$. For $t \ge 1$, block ℓ_{start} of (2.1) becomes

$$\left(\boldsymbol{h}_t^{(\ell_{\text{start}})}, \mathbf{KV}_{1:t}^{(\ell_{\text{start}})}\right) = \mathcal{F}_{\ell_{\text{start}}}\left(\Phi\left(\boldsymbol{h}_t^{(\ell_{\text{start}}-1)}, \mathbf{R}_{t-1}\right), \mathbf{KV}_{1:t-1}^{(\ell_{\text{start}})}\right), \tag{2.2}$$

while all other layers are unchanged. This creates a direct pathway through which representations computed by the middle layers at the previous step are used early in the current step. After layer ℓ_{end}, the cache $\mathbf{R}_t$ is computed from $\boldsymbol{h}_t^{(\ell_{\text{end}})}$ and $\mathbf{R}_{t-1}$ for the next step.

2.3 Gated Fusion and Recurrent Cache Update

The gated module fuses the current representation $\boldsymbol{h}_t^{(\ell_{\text{start}}-1)}$ with the recurrent cache $\mathbf{R}_{t-1}$:

$$\begin{aligned}\Phi\left(\boldsymbol{h}_t^{(\ell_{\text{start}}-1)}, \mathbf{R}_{t-1}\right) = \boldsymbol{h}_t^{(\ell_{\text{start}}-1)} \\ + \tanh\left(\gamma_{\text{cur}}\right)\, \sigma\left(f_{\text{cur}}\left(\left[\boldsymbol{h}_t^{(\ell_{\text{start}}-1)}, \mathbf{R}_{t-1}\right]\right)\right) \odot \boldsymbol{h}_t^{(\ell_{\text{start}}-1)} \\ + \tanh\left(\gamma_{\text{rec}}\right)\, \sigma\left(f_{\text{rec}}\left(\left[\boldsymbol{h}_t^{(\ell_{\text{start}}-1)}, \mathbf{R}_{t-1}\right]\right)\right) \odot \boldsymbol{W}_{\text{rec}}\mathbf{R}_{t-1},\end{aligned} \tag{2.3}$$

where $f_{\text{cur}}, f_{\text{rec}} : \mathbb{R}^{2d} \to \mathbb{R}^d$ are learnable linear layers on the concatenated inputs, $\boldsymbol{W}_{\text{rec}} : \mathbb{R}^d \to \mathbb{R}^d$ is a learnable projection, and σ is the element-wise sigmoid. The scalars $\tanh\left(\gamma_{\text{cur}}\right)$ and $\tanh\left(\gamma_{\text{rec}}\right)$ are global, input-independent gates that select between pathways at the model level, while the sigmoid terms are local, input-dependent gates that select features per token. Initializing the γ's to zero (and the gate weights randomly) makes the module start as the identity, so training begins from the backbone Transformer's behavior, which we find important for early training stability. After layer ℓ_{end}, the recurrent cache is updated as

$$\mathbf{R}_t = \text{RMSNorm}\left(\boldsymbol{h}_t^{(\ell_{\text{end}})} + \mathbf{R}_{t-1}\right). \tag{2.4}$$

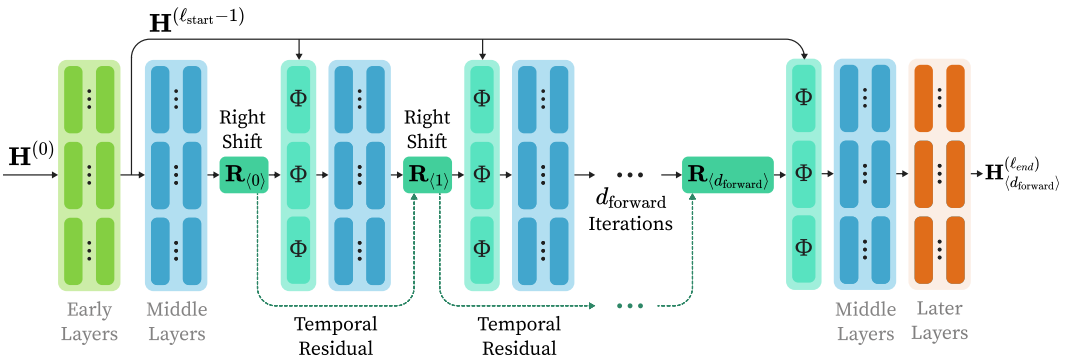

Figure 3: Approximated training scheme of T^{2}MLR. To train on all tokens in parallel, the ground-truth recurrent cache $\mathbf{R}^*$ is approximated by d_{forward} Jacobi iterations through the middle layers.

2.4 Approximated Training of T^{2}MLR with Temporal Parallelism

Like other models that recur over continuous representations, T^{2}MLR cannot directly use token-parallel training. We instead approximate $\mathbf{R}$ for all tokens in parallel with a constant number of Jacobi fixed-point iterations, similar to Wu et al. (2025) (Figure 3; Algorithm 1 in Section B).

We first run a standard forward pass without recurrence and take the (shifted) layer-ℓ_{end} representations as an initial cache $\mathbf{R}^{\langle 0 \rangle}$. We then fuse this cache before layer ℓ_{start}, rerun the middle layers up to ℓ_{end}, and obtain a refined cache $\mathbf{R}^{\langle 1 \rangle}$. After d_{forward} such iterations, $\mathbf{R}^{\langle d_{\text{forward}} \rangle}$ is used in a final pass through all layers. Because the procedure is differentiable, we back-propagate through the last d_{backward} iterations, analogous to truncated BPTT (Williams & Zipser, 2013). We use d_{forward}=16 and d_{backward}=4 unless stated otherwise, balancing approximation quality and training cost (Section B.3). In controlled runs at 135M, 360M, and 1B parameters, Jacobi-trained models match exact sequential recurrent training to within 0.005 validation loss (Table 6), so the approximation does not measurably degrade the learned model. These iterations are a training-time device only: during decoding, each token passes through the network once, adding only the fusion module.

3 Best of Both Worlds: State Tracking and Retrieval

We first isolate the inductive bias of T^{2}MLR on S_5-Retrieval, a synthetic task that separates it from standard Transformers, which excel at in-context retrieval but struggle with state tracking at small depth, and recurrent models, which track state but must compress retrieval into a fixed-size state.

3.1 The S_5-Retrieval Task

In the S_5 state-tracking task (Liu et al., 2023), the input is a sequence $(a_1, \ldots, a_N)$ of elements of S_5, the permutation group on 5 elements ($|S_5| = 120$), and the output is the sequence of cumulative compositions $(a_1, a_1 \circ a_2, \ldots, \Pi_{i=1}^{N} a_i)$. The i-th output is the i-th state, which transitions to the $(i{+}1)$-th state by composing with a_{i+1}. Under standard precision assumptions and the conjecture $\mathsf{TC}^0 \neq \mathsf{NC}^1$, fixed-depth Transformers cannot solve it exactly for arbitrary sequence lengths (Merrill et al., 2024), and a known associative-scan construction solves it with $\lceil \log_2 N \rceil$ layers (Li et al., 2025). RNNs solve it with a constant number of layers because their circuit depth grows with time (Merrill et al., 2024), but they fail at key-value retrieval, which must be compressed into a fixed-dimensional state.

S_5-Retrieval stress-tests the composition of state tracking and in-context retrieval (examples in Table 14). Given an alphabet $\aleph$ and a random mapping $\mathcal{D} : S_5 \to \aleph^K$, the input serializes the dictionary $\{(a, \mathcal{D}(a)) : a \in S_5\}$, followed by a delimiter and the action sequence $(a_1, \ldots, a_N)$, padded so that tokens align with the output. The target interleaves each state with its retrieved value:

$$\left(a_1,\ \mathcal{D}(a_1),\ a_1 \circ a_2,\ \mathcal{D}(a_1 \circ a_2),\ \ldots,\ \Pi_{i=1}^{N} a_i,\ \mathcal{D}\left(\Pi_{i=1}^{N} a_i\right)\right).$$

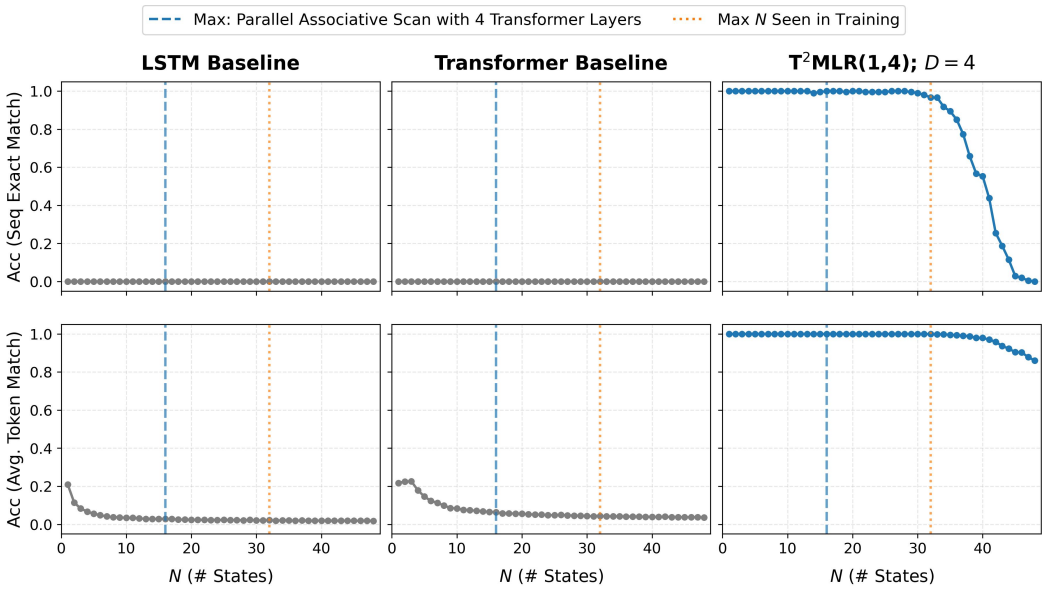

Figure 4: S_5-Retrieval accuracy vs. number of states for parameter-matched 4-layer LSTM, Transformer, and T^{2}MLR models, trained for 200k steps on up to 32 states. Top: sequence exact match, which is sensitive to compounding errors over long sequences; bottom: average per-token accuracy. Blue dashed line: the most states a 4-layer Transformer can track with the parallel associative-scan circuit (Li et al., 2025). Orange dotted line: the most states seen in training. T^{2}MLR attains near-perfect exact match within the training range and non-trivial token accuracy beyond it, while both baselines fail to learn the task.

3.2 Empirical Separation from Transformers and RNNs

We compare a 4-layer LSTM, a 4-layer, 6-head Llama Transformer (Touvron et al., 2023), and a T^{2}MLR model built on the same Llama backbone with recurrence over all four layers ($\ell_{\text{start}} = 1$, $\ell_{\text{end}} = 4$), all with matched parameter counts. We use $K = 4$, sample N uniformly from $\{1, \ldots, 32\}$ during training, and resample $\mathcal{D}$ for every sequence so that retrieval must happen in context. Models are trained for 200k steps with learning rate 5×10^{-4}, a cosine schedule, and batch size 64, and evaluated on held-out sequences with N from 1 to 48.

As Figure 4 shows, T^{2}MLR reaches near-perfect exact match within the training lengths and retains non-trivial token accuracy beyond them, while the LSTM and Transformer baselines keep only modest token accuracy and collapse on exact match, which compounds errors over up to $48 \times 5 = 240$ tokens, so it can fall to zero for the baselines even when they are partially correct; per-token accuracy gives the finer-grained picture. This matches the intended inductive bias: the recurrent pathway propagates the evolving state, while the Transformer backbone preserves access to the dictionary for retrieval.

4 Experiments

Setup All models build on SmolLM2 (Allal et al., 2025), a LLaMA-like decoder-only Transformer, and are pretrained on FineWeb-Edu (Penedo et al., 2024) with identical hyperparameters (Section E.1). We denote a variant by T^{2}MLR $(\ell_{\text{start}}, \ell_{\text{end}})$, with recurrence depth $D = \ell_{\text{end}} - \ell_{\text{start}} + 1$. Each T^{2}MLR model is compared with a Transformer of matching size trained on the same data; to offset the fusion module's parameters we widen the baseline (e.g., from 576 to 584 hidden units at 135M), so the baseline is slightly larger, which is conservative. We evaluate zero-shot with lm-eval-harness (Gao et al., 2024) on ARC-Challenge/Easy (ARC-C/E), HellaSwag (HS), OpenBookQA (OBQA), PIQA, SciQ, and Winogrande (WG), reporting normalized accuracy where available. Since most continuous latent-reasoning methods lack sequence-parallel training and do not extend to pretraining, our main baseline is the standard Transformer, with looped and pause-token models in Section 4.3.

Setting	Model	Zero-shot ARC-C	ARC-E	HS	OBQA	PIQA	SciQ	WG	Avg.	Finetuned GSM-NL	GSM-Sym
135M / 10B	Baseline	**24.74**	44.28	29.81	**30.20**	**61.53**	60.80	48.46	42.83	26.5	38.1
	T^{2}MLR (9,22)	24.15	**45.24**	**30.40**	29.20	61.15	**66.40**	**51.54**	**44.01**	**31.4**	**40.9**
361M / 10B	Baseline	27.13	**52.65**	37.18	**34.00**	65.51	71.00	51.85	48.48	31.08	39.35
	T^{2}MLR (9,24)	**28.84**	52.48	**38.66**	33.40	**66.00**	**73.10**	**52.09**	**49.22**	**34.12**	**42.46**
1B / 10B	Baseline	30.97	**55.85**	42.91	33.40	67.74	**77.10**	51.70	51.38	32.37	43.97
	T^{2}MLR (9,24)	**32.00**	55.30	**45.11**	**35.40**	**68.99**	75.60	**54.22**	**52.37**	**36.69**	**44.96**
135M / 50B	Baseline	**27.82**	**52.53**	36.82	33.60	65.67	72.40	49.64	48.35	26.46	38.67
	T^{2}MLR (9,22)	27.56	51.30	**38.74**	**34.00**	**66.10**	**75.20**	**53.04**	**49.42**	**30.48**	**41.55**
361M / 50B	Baseline	31.23	**59.64**	46.28	34.60	**70.08**	76.00	52.01	52.83	40.18	42.53
	T^{2}MLR (9,24)	**33.28**	58.88	**48.57**	**36.80**	69.42	**80.30**	**56.20**	**54.78**	**44.35**	**46.63**

Table 1: Results at larger model and data scales (setting = parameters / FineWeb-Edu tokens), comparing T^{2}MLR with parameter-matched Transformers trained on the same data. Zero-shot: accuracy on the benchmarks of Section 4; GSM-NL and GSM-Sym are accuracies after finetuning on the natural-language and symbolic variants of GSM-Aug. T^{2}MLR improves the zero-shot average and both GSM-Aug variants in every setting; HotpotQA results and relative gains are in Table 9.

Comparisons are matched on parameters, data, and inference compute but not on training compute (Section 2.4); Section C.3 reports a training-compute-matched baseline.

4.1 Pretraining at Scale

We first test T^{2}MLR in our largest settings, where benchmarks are furthest from chance. We scale along two axes: model size, pretraining 361M and 1B models (32 layers, T^{2}MLR (9,24)) on 10B tokens, and data, extending pretraining to 50B tokens at 135M (T^{2}MLR (9,22)) and 361M (T^{2}MLR (9,24)). The 135M, 10B-token pair is the reference setting of Section 4.2. Each checkpoint is also finetuned on GSM-Aug (Section 4.4).

T^{2}MLR improves the zero-shot average in every setting (Table 1), and the gain is largest for the most-trained model. At 361M parameters and 50B tokens, the setting furthest from chance level, the average rises from 52.83 to 54.78, with gains of 4.3 points on SciQ, 4.2 on Winogrande, and 2.3 on HellaSwag. The same model trained on only 10B tokens gains 0.74 points, so at 361M more pretraining data widens rather than erodes the advantage. After finetuning, T^{2}MLR improves both GSM-Aug variants in every setting, by about 10% relative at 361M with 50B tokens ($40.18 \rightarrow 44.35$ and $42.53 \rightarrow 46.63$), and HotpotQA-Easy by 14–16% relative at 361M and 1B (Table 9). ARC-Easy is the one benchmark on which the baseline stays slightly ahead in all four larger settings. The advantage of T^{2}MLR is therefore not an artifact of the small, near-chance regime.

4.2 Where Should Recurrence Live?

At 135M parameters and 10B tokens, we map the design space of the recurrent block (Table 2, left): five blocks centered in the network, from full recurrence ($D = 30$) down to $D = 2$, and, at fixed widths $D \in \{6, 14\}$, blocks placed over the earliest or latest layers instead.

Two findings emerge. First, a narrow middle block is best: T^{2}MLR (13,18), which recurs over only 6 of 30 layers (20%), attains the highest average (44.14 vs. 42.83 for the baseline), while full recurrence ($D = 30$), as used by prior latent-reasoning methods, gains only 0.53 points. Second, location matters at fixed width: at both $D = 6$ and $D = 14$, the middle block beats early and late placement by 0.9 to 1.8 points, and early-layer recurrence falls below the baseline. Validation loss, in contrast, generally improves with block width apart from the two extremes (Figure 7), which suggests an implicit bias of middle-layer recurrence toward reasoning that perplexity does not capture, as Saunshi et al. (2024) observe for middle-layer stacking.

D	Block location Early	Middle	Late
30		43.36 (1,30)	
22	–	43.73 (5,26)	–
14	42.21 (1,14)	44.01 (9,22)	43.14 (17,30)
6	42.74 (1,6)	**44.14** (13,18)	42.87 (25,30)
2	–	42.89 (15,16)	–
Transformer: 42.83			

Model	Decode cost	Zero-shot avg.
Pause-token $\times 2$	$2\times$	43.31
Full-looped $\times 2$	$2\times$	42.99
Middle-looped $\times 3$	$1.93\times$	42.68
T^{2}MLR (9,22)	$1\times$	44.01
T^{2}MLR (13,18)	$1\times$	**44.14**
Transformer	$1\times$	42.83

Table 2: Zero-shot results of 135M models trained on 10B tokens. Each cell is the zero-shot average over the seven benchmarks, with the recurrence boundaries $(\ell_{\text{start}}, \ell_{\text{end}})$ in parentheses; per-benchmark results are in Table 10. *Left:* recurrent blocks of width D over the middle, earliest, or latest layers. A narrow middle block is best, and at fixed width the middle placement beats early and late placement. *Right:* baselines that spend extra compute at every decoding step (middle-looped $\times 3$ loops the same layers 9–22 that T^{2}MLR (9,22) recurs over); decode cost counts layer evaluations per generated token relative to the Transformer, to which T^{2}MLR adds only the fusion module.

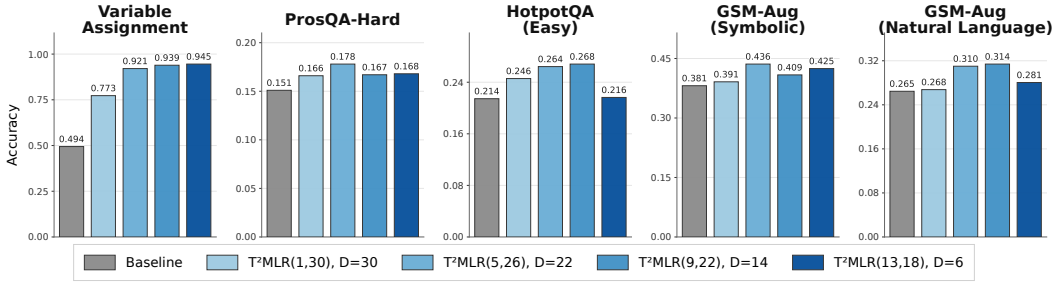

Figure 5: Finetuning accuracy of 135M T^{2}MLR variants and the parameter- and data-matched Transformer on multi-hop reasoning and grade-school math tasks. We report pass@1 accuracy for all experiments. In all five tasks, T^{2}MLR outperforms parameter and data-matched transformer baselines. The middle-layer variants with $D = 22$ and $D = 14$ outperform full-layer recurrence ($D = 30$) on all five tasks, and $D = 6$ on four of five.

4.3 Comparison to Looped and Pause-Token Baselines

As reference points for spending extra computation, we also train three parameter-matched 135M models on the same 10B tokens: a $2\times$ pause-token model (Goyal et al., 2024), a $2\times$ full-looped Transformer, and a $3\times$ middle-looped Transformer (layers 9–22). T^{2}MLR (13,18) attains the best average (Table 2, right), although these baselines spend about twice the compute at every decoding step, whereas T^{2}MLR spends its extra compute only in training. Coconut-style methods (Hao et al., 2024) lack a scalable teacher-forced pretraining procedure, so we do not compare against them.

4.4 Finetuning T^{2}MLR on Reasoning Downstream Tasks

Perplexity can understate changes in a model's internal computation, so we test latent reasoning directly by finetuning the 135M checkpoints of Section 4.2 on four tasks, made harder where the baseline saturates the original (details in Section E.2): variable assignment at depth 5 (Saunshi et al., 2024); ProsQA-Hard, a version of ProsQA (Hao et al., 2024) with 60-node graphs and 8-step paths; the easy subset of HotpotQA (Yang et al., 2018), with GPT-4o-mini (OpenAI, 2024) chain-of-thought data; and both variants of GSM-Aug (Deng et al., 2023).

On all tasks, T^{2}MLR outperforms the parameter- and data-matched baseline (Figure 5), and the $D = 22$ and $D = 14$ middle-layer variants beat full-layer recurrence on every task, highlighting the importance of placing recurrence in the middle layers to boost latent reasoning.

	GSM8K		MATH500	
	Baseline	T^{2}MLR (5,20)	Baseline	T^{2}MLR (5,20)
pass@1	35.78	**39.88** (+11.5%)	12.80	**18.00** (+40.6%)

Table 3: Retrofitting results for SmolLM2-1.7B-Instruct: T^{2}MLR (5,20) vs. identical finetuning on OpenMathReasoning without the recurrent pathway (relative gains in parentheses).

4.5 Retrofitting a Pretrained Transformer into T^{2}MLR

Can T^{2}MLR be adopted without training from scratch, by grafting the recurrent pathway onto an existing pretrained model? In our largest-scale experiment, we insert it (T^{2}MLR (5,20)) into the instruction-tuned SmolLM2-1.7B-Instruct (24 layers) and finetune for one epoch on OpenMathReasoning (Moshkov et al., 2025), comparing against identical finetuning without the pathway. Retrofitting improves GSM8K from 35.78 to 39.88 ($+11.5\%$ relative) and MATH500 from 12.80 to 18.00 ($+40.6\%$; Table 3). Simply adding the pathway to an off-the-shelf model and briefly finetuning thus delivers substantial reasoning gains, so existing Transformers can be upgraded at a fraction of the cost of pretraining while keeping nearly the same inference cost.

5 Related Work

Deep-to-shallow temporal recurrence The Feedback Transformer (Fan et al., 2020) first let lower layers attend to higher-level representations of past tokens, but its fully sequential training confines it to small models. T^{2}MLR revisits this idea with a single middle-to-middle residual pathway that can be pretrained with dense teacher forcing under sequence parallelism (Section 2.4). A growing line of recent work builds on the same principle of feeding a deeper representation of the previous token into shallower computation of the current one: the Recurrent Transformer lets each layer attend to keys and values computed from its own outputs (Oncescu et al., 2026); the Latent Recurrent Transformer reuses an upper-layer state through residual and KV injection (Huang et al., 2026b;a); the Full-bandwidth Transformer fuses the final-layer state into the next input embedding (Wang et al., 2026); WhiteMatter routes mixtures of all past-token layer states into per-layer KV channels (Zhang & Ren, 2026); and Recirculation leaks deep activations into shallow layers of frozen pretrained models at inference time (Mozer et al., 2026b). Within this family, T^{2}MLR injects the recurrent state directly into the residual stream at a *middle* layer, trains the pathway end-to-end with truncated BPTT, and shows that confining recurrence to a localized middle block outperforms full-depth recurrence (Sections 4.2 and 4.4); see Section A for more.

Latent reasoning Coconut (Hao et al., 2024) and CODI (Shen et al., 2025) internalize chain-of-thought into continuous hidden states, and related methods replace discrete reasoning tokens with soft token mixtures (Su et al., 2025; Zhang et al., 2025; Zhuang et al., 2025; Tang et al., 2026). These methods carry latent information across steps through the input embedding, typically from the final layer; HRPO (Yue et al., 2025) does so in an RL setting without gradient flow through past tokens. T^{2}MLR instead relaxes the bottleneck through *middle-layer recurrence*, preserving reasoning states where abstract computation occurs: it keeps the discrete token interface, trains under dense token-level supervision, and optimizes the recurrent pathway across tokens with BPTT (Table 4).

Looped Transformers recur over a different axis Recurrent Transformer variants can be organized by their axis of recurrence, depth or time (Mozer et al., 2026a). Looped and depth-recurrent Transformers (Dehghani et al., 2019; Giannou et al., 2023; Saunshi et al., 2025; Geiping et al., 2025; Zhu et al., 2025b) recur over *depth*: every token is passed through a shared block several times, so per-token decoding compute and latency grow with the number of loops. Geiping et al. (2025) and Zhu et al. (2025b) also reuse KV caches from later loops in earlier ones, another way of shortcutting deeper representations to shallower layers. T^{2}MLR recurs over *time*. At decoding time, each token passes through every layer once and its recurrent input is already computed for the previous token, so the only added cost is the fusion module (Section B.5). The Jacobi iterations of Section 2.4 parallelize training over the sequence rather than adding depth per token.

State-space, hybrid, and recurrent-memory models Another body of work adds recurrence along a different architectural axis. State-space and hybrid models such as Mamba (Gu & Dao,

2023), RetNet (Sun et al., 2023), and Griffin (De et al., 2024) replace or interleave attention with recurrent, linear-time sequence mixers, typically for long-context efficiency, while recurrent-memory variants such as Transformer-XL (Dai et al., 2019) carry memory across segments beyond the context window. T^{2}MLR instead keeps the Transformer backbone and standard attention intact and adds a deep-to-shallow recurrent residual pathway across decoding steps, targeting reasoning within the context window rather than long-context memory (Section A).

6 Discussion, Limitations, and Future Work

Training cost The main limitation of T^{2}MLR is training cost. The Jacobi refinement through the recurrent block makes training roughly 2–4$\times$ slower in wall-clock time than a parameter-matched Transformer, depending on the block size. Under a *training-wall-clock-matched* budget, a longer-trained Transformer surpasses the 135M T^{2}MLR on zero-shot NLP (Table 11). Our parameter-, data-, and inference-compute-matched comparisons therefore isolate the *architectural and inference-side* contribution of middle-layer recurrence, which adds at most $\sim$8% per-token inference overhead, rather than a training-compute win. Reducing this cost is a key direction: fewer refinement steps may suffice, and on-policy RL already produces exact recurrent states during rollouts (Section B.2).

Scale and transfer Our from-scratch models reach 1B parameters and 50B tokens (Section 4.1), and retrofitting a pretrained 1.7B model yields substantial gains (Section 4.5), which makes the architecture easy to adopt. Scaling further, testing on more demanding reasoning benchmarks, and estimating variance across seeds remain important future work.

Architecture design and mechanistic understanding Better fusion parameterizations and recurrence boundaries may yield stronger and more efficient variants. Another direction is to study mechanistically what information the recurrent pathway carries across decoding steps. Initial evidence is encouraging: probes on T^{2}MLR's intermediate layers predict the next token with lower loss than probes on the baseline (Section D.2), and in our runs the learned global gates converge to a positive recurrent gate and a negative current-input gate, so the model treats recurrence as an additive refinement of the residual stream rather than a replacement for it (Section D.3).

Conclusion T^{2}MLR routes recurrence through the middle layers, carrying a middle-layer state across decoding steps at nearly the inference cost of a standard Transformer. Across pretraining and downstream reasoning tasks it outperforms parameter- and data-matched baselines, localized middle-layer recurrence often beats full-depth recurrence, and the pathway can be retrofitted. These results suggest that the value of recurrence in Transformers depends not only on enabling iterative latent computation but on placing it at the right depth. We hope this work serves as a starting point for a broader investigation of *where* recurrence should live in latent-reasoning architectures.

Reproducibility Statement

Architectural details and training hyperparameters are given in Sections B and E.1, and the synthetic data generation procedures for each task are described alongside the corresponding experiments. Our training and inference code is in the supplementary material.

AI Use Statement

We used GPT-4o-mini to generate the chain-of-thought finetuning data for HotpotQA (Section 4.4). The core training and inference code, as well as the experiment infrastructure (job scripts, evaluation harnesses, and plotting), were written with the assistance of coding agents. We also used generative AI tools to find related literature and to edit the paper for readability. We reviewed and tested all AI-assisted code, checked every suggested reference against the original paper, and verified all AI-edited text; we take full responsibility for the content of this paper.

REFERENCES

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. SmolLM2: When smol goes big – data-centric training of a small language model, 2025. URL https://arxiv.org/abs/2502.02737.

Adam Atanas and Kai Liu. A modular dataset to demonstrate LLM abstraction capability, 2025. URL https://arxiv.org/abs/2503.17645.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time, 2024. URL https://arxiv.org/abs/2501.00663.

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.

Aydar Bulatov, Yuri Kuratov, and Mikhail S. Burtsev. Recurrent memory transformer, 2022. URL https://arxiv.org/abs/2207.06881.

Natasha Butt, Ariel Kwiatkowski, Ismail Labiad, Julia Kempe, and Yann Ollivier. Soft tokens, hard truths, 2025. URL https://arxiv.org/abs/2509.19170.

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

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

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context, 2019. URL https://arxiv.org/abs/1901.02860.

Soham De, Samuel L. Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, Guillaume Desjardins, Arnaud Doucet, David Budden, Yee Whye Teh, Razvan Pascanu, Nando De Freitas, and Caglar Gulcehre. Griffin: Mixing gated linear recurrences with local attention for efficient language models, 2024. URL https://arxiv.org/abs/2402.19427.

Mostafa Dehghani, Stephan Gouws, et al. Universal transformers. In *International Conference on Learning Representations*, 2019.

Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. Implicit chain of thought reasoning via knowledge distillation, 2023. URL https://arxiv.org/abs/2311.01460.

Yuntian Deng, Yejin Choi, and Stuart Shieber. From explicit CoT to implicit CoT: Learning to internalize CoT step by step, 2024. URL https://arxiv.org/abs/2405.14838.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing some limitations of transformers with feedback memory, 2020. URL https://arxiv.org/abs/2002.09402.

Tianyu Fu, Yichen You, Zekai Chen, Guohao Dai, Huazhong Yang, and Yu Wang. Think-at-hard: Selective latent iterations to improve reasoning language models. *arXiv preprint arXiv:2511.08577*, 2025.

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. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Jonas Geiping, Sean Michael 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 *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=S3GhJooWIC.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. *EMNLP*, 2021.

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 (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pp. 11398–11442. PMLR, 2023. URL https://arxiv.org/abs/2301.13196. arXiv:2301.13196.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens, 2024. URL https://arxiv.org/abs/2310.02226.

Riccardo Grazzi, Julien Siems, Arber Zela, Jörg K. H. Franke, Frank Hutter, and Massimiliano Pontil. Unlocking state-tracking in linear RNNs through negative eigenvalues. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=UvTo3tVBk2.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2023. URL https://arxiv.org/abs/2312.00752.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces, 2021. URL https://arxiv.org/abs/2111.00396.

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, 2024. URL https://arxiv.org/abs/2412.06769.

Zeyi Huang, Xuehai He, Yong Jae Lee, and Yelong Shen. Trading depth for time in recurrent transformers, 2026a. URL https://arxiv.org/abs/2609.21605.

Zeyi Huang, Xuehai He, Liliang Ren, Yiping Wang, Baolin Peng, Hao Cheng, Shuohang Wang, Pengcheng He, Jianfeng Gao, Yong Jae Lee, and Yelong Shen. Latent recurrent transformer: Architecture exploration, training strategies, and scaling behavior, 2026b. URL https://arxiv.org/abs/2605.26797.

Samy Jelassi, David Brandfonbrener, Sham M. Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2402.01032.

Deqian Kong, Minglu Zhao, Dehong Xu, Bo Pang, Shu Wang, Edouardo Honig, Zhangzhang Si, Chuan Li, Jianwen Xie, Sirui Xie, and Ying Nian Wu. Latent thought models with variational bayes inference-time computation, 2025. URL https://arxiv.org/abs/2502.01567.

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 (ICLR)*, 2020. URL https://arxiv.org/abs/1909.11942. arXiv:1909.11942.

Belinda Z. Li, Zifan Carl Guo, and Jacob Andreas. (how) do language models track state? In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=8SXosAVIFH.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=De4FYqjFueZ.

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

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. *NeurIPS*, 2022.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=QZgo9JZpLq.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Ivan Moshkov, Darragh Hanley, Ivan Sorokin, Shubham Toshniwal, Christof Henkel, Benedikt Schifferer, Wei Du, and Igor Gitman. Aimo-2 winning solution: Building state-of-the-art mathematical reasoning models with openmathreasoning dataset, 2025. URL https://arxiv.org/abs/2504.16891.

Michael C. Mozer, Shoaib Ahmed Siddiqui, and Rosanne Liu. The topological trouble with transformers, 2026a. URL https://arxiv.org/abs/2604.17121.

Michael C. Mozer, Shoaib Ahmed Siddiqui, Danny Sawyer, Sunny Sanyal, and Rosanne Liu. Recirculation, 2026b. URL https://arxiv.org/abs/2608.17981.

Eshaan Nichani, Jason D. Lee, and Alberto Bietti. Understanding factual recall in transformers via associative memories. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2412.06538.

Costin-Andrei Oncescu, Depen Morwani, Samy Jelassi, Alexandru Meterez, Mujin Kwun, and Sham Kakade. The recurrent transformer: Greater effective depth and efficient decoding, 2026. URL https://arxiv.org/abs/2604.21215.

OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

OpenAI. GPT-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In Waleed Ammar, Annie Louis, and Nasrin Mostafazadeh (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pp. 48–53, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-4009. URL https://aclanthology.org/N19-4009/.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y. Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models, 2023. URL https://arxiv.org/abs/2302.10866.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling, 2019. URL https://arxiv.org/abs/1911.05507.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Nikunj Saunshi, Stefani Karp, Shankar Krishnan, Sobhan Miryoosefi, Sashank Jakkam Reddi, and Sanjiv Kumar. On the inductive bias of stacking towards improving reasoning. *Advances in Neural Information Processing Systems*, 37:71437–71464, 2024.

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 (ICLR)*, 2025. URL https://arxiv.org/abs/2502.17416. arXiv:2502.17416.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. CODI: Compressing chain-of-thought into continuous space via self-distillation, 2025. URL https://arxiv.org/abs/2502.21074.

Oscar Skean, Md Rifat Arefin, Dan Zhao, Niket Patel, Jalal Naghiyev, Yann LeCun, and Ravid Shwartz-Ziv. Layer by layer: Uncovering hidden representations in language models. In *Forty-second International Conference on Machine Learning (ICML)*, 2025. URL https://openreview.net/forum?id=WGXb7UdvTX.

DiJia Su, Hanlin Zhu, Yingchen Xu, Jiantao Jiao, Yuandong Tian, and Qinqing Zheng. Token assorted: Mixing latent and text tokens for improved language model reasoning, 2025. URL https://arxiv.org/abs/2502.03275.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models, 2023. URL https://arxiv.org/abs/2307.08621.

Jihoon Tack, Jack Lanchantin, Jane Yu, Andrew Cohen, Ilia Kulikov, Janice Lan, Shibo Hao, Yuandong Tian, Jason Weston, and Xian Li. LLM pretraining with continuous concepts, 2025. URL https://arxiv.org/abs/2502.08524.

Yao Tang, Li Dong, Yaru Hao, Qingxiu Dong, Furu Wei, and Jiatao Gu. Multiplex thinking: Reasoning via token-wise branch-and-merge, 2026. URL https://arxiv.org/abs/2601.08808.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In *ACL*, 2019.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Xi Wang, Ziyang Cai, Zheng Zhan, Harry Dong, Ying Fan, Gustavo de Rosa, Tim Pearce, and John Langford. Full-bandwidth transformer, 2026. URL https://arxiv.org/abs/2608.08888.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

Johannes Welbl, Nelson F Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. *arXiv preprint arXiv:1707.06209*, 2017.

Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. In *Backpropagation*, pp. 433–486. Psychology Press, 2013.

Haoyi Wu and Kewei Tu. Layer-condensed KV cache for efficient inference of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. URL https://arxiv.org/abs/2405.10637.

Haoyi Wu, Zhihao Teng, and Kewei Tu. Parallel continuous chain-of-thought with jacobi iteration. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 914–926, 2025.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *Advances in neural information processing systems*, 37: 115491–115522, 2024.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Zhenrui Yue, Bowen Jin, Huimin Zeng, Honglei Zhuang, Zhen Qin, Jinsung Yoon, Lanyu Shang, Jiawei Han, and Dong Wang. Hybrid latent reasoning via reinforcement learning, 2025. URL https://arxiv.org/abs/2505.18454.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Wenbo Zhang and Xiang Ren. WhiteMatter: All-to-all cross-layer connections via KV mixing, 2026. URL https://arxiv.org/abs/2608.18486.

Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, Shuohang Wang, Yelong Shen, and Xin Eric Wang. Soft thinking: Unlocking the reasoning potential of LLMs in continuous concept space. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=ByQdHPGKgU.

Hanlin Zhu, Shibo Hao, Zhiting Hu, Jiantao Jiao, Stuart Russell, and Yuandong Tian. Reasoning by superposition: A theoretical perspective on chain of continuous thought. *arXiv preprint arXiv:2505.12514*, 2025a. URL https://arxiv.org/abs/2505.12514.

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

Yufan Zhuang, Liyuan Liu, Chandan Singh, Jingbo Shang, and Jianfeng Gao. Mixture of inputs: Text generation beyond discrete token sampling. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=l6C6Pw30Gl.

Jiaru Zou, Xiyuan Yang, Ruizhong Qiu, Gaotang Li, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, James Zou, Mengdi Wang, and Ling Yang. Latent collaboration in multi-agent systems, 2025. URL https://arxiv.org/abs/2511.20639.

APPENDIX CONTENTS

A Extended Related Work **16**

B Training T^{2}MLR with Temporal Parallelism **17**

B.1 Temporal-Parallel Approximated Training 18

B.2 Computational Cost 18

B.3 How Deep Must the Approximation Be? 19

B.4 Jacobi Approximation vs. Exact Recurrence in Training 20

B.5 Inference Overhead 21

C Additional Empirical Results **22**

C.1 Scaling to Larger Models and More Data 22

C.2 Full 135M Results and Recurrence Location 22

C.3 Training-Compute-Matched Baseline 24

D Analysis of T^{2}MLR **24**

D.1 T^{2}MLR as a Generalized Transformer and Recurrent Model 24

D.2 Probing Intermediate Representations 24

D.3 Learned Gating Behavior 25

E Experimental Setup **25**

E.1 Pretraining and Zero-Shot Evaluation 25

E.2 Finetuning Data 26

A Extended Related Work

Deep-to-shallow temporal recurrence In a standard decoder-only Transformer, the representation at a given layer can access only lower-layer representations of past tokens, even though higher-level ones have already been computed. The Feedback Transformer (Fan et al., 2020) removes this restriction by letting every layer attend to a learned mixture of all layer states of past tokens. This yields strong small, shallow models, but training must proceed token by token. Layer-condensed KV caching (Wu & Tu, 2024) lets all layers attend to top-layer keys and values and recovers parallel training through iterative refinement, but its motivation is KV-cache compression rather than reasoning. Several recent works pursue the same deep-to-shallow principle, differing in which representation is fed back, where it re-enters the network, and how training is parallelized:

- *Recurrent Transformer* (Oncescu et al., 2026): each layer attends to keys and values from its own outputs, with efficient tiled prefill and training; pretrained at 150M–300M on C4.
- *Latent Recurrent Transformer* (Huang et al., 2026b): reuses an upper-layer hidden state of the previous token through both gated KV projections and additive residual injection, trained with interleaved parallel updates over disjoint position subsets; 1.3B–2.1B models. Huang et al. (2026a) add "thought tokens" to trade depth for temporal computation.
- *Full-bandwidth Transformer* (Wang et al., 2026): fuses the final-layer state of the previous token with the next token embedding through a gated linear unit, trained with a scheduled multi-pass objective; 1B models on 400B tokens.
- *WhiteMatter* (Zhang & Ren, 2026): a router mixes all layer states of each past token into a small number of KV channels, and each layer attends to one channel; recurrence enters only through attention, trained with Gauss–Seidel iterations.
- *Recirculation* (Mozer et al., 2026b): a training-free, inference-time leak of a normalized fraction of deep-layer activations into a shallower layer at the next step of frozen pretrained models, with an adaptive variant that learns only the mixing coefficients.

T^{2}MLR shares the principle and differs in three respects: (i) the recurrent state leaves a middle layer ℓ_{end} and re-enters the residual stream at an earlier middle layer ℓ_{start}, rather than the input embedding or an attention memory; (ii) the pathway is trained end-to-end with dense teacher forcing, Jacobi-style temporal parallelism, and truncated BPTT through the recurrence; and (iii) we find that restricting the recurrent span to a localized middle block (as little as 20% of layers) outperforms full-depth recurrence. Like all of these methods, and unlike looped Transformers, T^{2}MLR keeps one forward pass per token at decoding time.

Latent Reasoning Many recent works study or exploit *latent computation* in transformers to improve reasoning without relying on long explicit chain-of-thought (CoT).

On the modeling side, recent work has explored increasing reasoning capacity without relying on long explicit chain-of-thought traces, by shifting computation into latent or continuous representations. One line of work focuses on *internalizing* chain-of-thought into hidden states, via distillation or progressively removing explicit intermediate steps while preserving final-answer performance (Deng et al., 2023; 2024; Hao et al., 2024; Shen et al., 2025). A second line explicitly allocates additional *inference-time computation* to latent processes, such as iterative latent refinement, token-wise branch-and-merge reasoning, or latent communication in multi-agent settings (Kong et al., 2025; Fu et al., 2025; Tang et al., 2026; Zou et al., 2025). Finally, several approaches replace discrete reasoning tokens with *soft or continuous* alternatives, including mixed latent/text token training and continuous mixtures in embedding space, which can shorten or eliminate verbose reasoning traces (Su et al., 2025; Tack et al., 2025; Zhang et al., 2025; Butt et al., 2025; Zhuang et al., 2025). On the theory side, Zhu et al. (2025a) prove that continuous chain-of-thought helps on graph reasoning.

Among latent-reasoning methods, the closest to our work is HRPO (Yue et al., 2025), which projects the last-layer latent state of the previous token back into token space before the first layer and trains the resulting architecture with RL, improving mathematical reasoning. Crucially, gradients in HRPO do not flow to past tokens, whereas we use supervised training and backpropagate through time; Section B.3 shows that this gradient flow matters most for the fusion module. We emphasize *middle-layer* recurrence without confining recurrent information to embedding space.

Model	Recurrent Information	Injection Location	Fusion Module	BPTT
Transformer	Discrete token	Input embedding	–	N/A
Coconut	Last-layer hidden state	Post-embedding	Identity	✓
CODI	Last-layer hidden state	Post-embedding	2-layer MLP	✓
Multiplex-Thinking	Soft token mixture	Post-embedding	Identity	×
HRPO	Soft token mixture + discrete token	Post-embedding (soft token) + Input embedding (discrete token)	Gated fusion	×
T^2MLR (ours)	Middle-layer hidden state + Discrete token	Earlier middle layer (hidden state) + Input embedding (discrete token)	Gated fusion	✓

Table 4: Comparison with related architectures. Standard transformers communicate across time only through discrete tokens. Prior recurrent or latent-reasoning variants additionally propagate either last-layer hidden states or soft token mixtures, usually through input-level or post-embedding pathways. T^2MLR adds a recurrent middle-layer hidden-state pathway, fused into an earlier middle layer of the next token, while preserving the ordinary discrete token pathway and supporting end-to-end supervised training with a constant-depth BPTT.

Looped Transformers Looped (recurrent-depth) Transformers trade new parameters for repeated computation by repeatedly applying the same Transformer block. Universal Transformers introduce explicit recurrence over depth, optionally with adaptive halting (Dehghani et al., 2019). ALBERT popularizes cross-layer parameter sharing in Transformers to reduce parameters (Lan et al., 2020). On the expressivity side, Giannou et al. (2023) show that looping expands the expressive power of Transformers, and Saunshi et al. (2025) provide theoretical evidence that looping can approximate the benefits of much deeper Transformers, yielding large effective depth with fewer parameters.

Geiping et al. (2025) demonstrate empirically that increasing test-time compute via recurrent depth improves reasoning. Most recently, ByteDance's Ouro (Zhu et al., 2025b) looped LMs scale this design to billion-parameter models, reporting performance that rivals much larger-parameter Transformers. Both Geiping et al. (2025) and Zhu et al. (2025b) use the KV caches of the last loops when computing the forward pass for the shallower loops. This is another shortcut that lets shallower layers use more refined deeper representations.

Recurrent memory for long-context modeling. A line of work uses recurrence primarily as a *memory mechanism* to improve long-context performance, rather than to change per-token computation. Transformer-XL reuses hidden states across segments to extend effective context (Dai et al., 2019), while Compressive Transformer retains older history via a compressed memory (Rae et al., 2019). Recurrent Memory Transformer passes information across segments using dedicated memory tokens (Bulatov et al., 2022). More recently, Titans (Behrouz et al., 2024) augments standard attention over a fixed-length current context (motivated by attention's quadratic cost) with a neural long-term memory module that stores and retrieves information from much longer histories.

State space and hybrid models. State space models offer subquadratic alternatives to attention by maintaining and updating a compact state over the sequence, with representative examples including S4 (Gu et al., 2021), Hyena (Poli et al., 2023), and Mamba (Gu & Dao, 2023). Some hybrid architectures interleave attention with recurrent/SSM-style blocks to trade off content-based retrieval and efficiency, e.g., RetNet (Sun et al., 2023), Griffin (De et al., 2024), and DeltaNet (Yang et al., 2024). In contrast, we do not introduce a new recurrent/SSM layer or replace attention; instead, we add a token-to-token recurrent connection that injects a previous token's later-layer residual stream into an earlier-layer residual stream of the current token.

B Training T^2MLR with Temporal Parallelism

This section gives the full training procedure sketched in Section 2.4, analyzes its cost, and validates it from three angles: how close its gradients are to those of a much deeper approximation (Section B.3), whether models trained with it match models trained with exact sequential recurrence (Section B.4), and what it costs at inference (Section B.5).

Algorithm 1: Temporal-Parallel Approximated Forward for T^{2}MLR

Input: d_{forward}, d_{backward}, input embeddings $\boldsymbol{H}^{(0)} \in \mathbb{R}^{T\times d}$
Output: $\boldsymbol{H}^{(L)}_{\langle d_{\text{forward}}\rangle} \in \mathbb{R}^{T\times d}$

$\boldsymbol{H}^{(\ell_{\text{start}}-1)} \leftarrow \mathcal{F}_{1:\ell_{\text{start}}-1}\big(\boldsymbol{H}^{(0)}\big)$ // exact input to layer ℓ_{start}
$\boldsymbol{H}^{(\ell_{\text{end}})}_{\langle 0\rangle} \leftarrow \mathcal{F}_{\ell_{\text{start}}:\ell_{\text{end}}}\big(\boldsymbol{H}^{(\ell_{\text{start}}-1)}\big)$ // single pass to seed cache
$\mathbf{R}_{\langle 0\rangle} \leftarrow \mathrm{ShiftRight}\Big(\boldsymbol{H}^{(\ell_{\text{end}})}_{\langle 0\rangle}\Big)$ // initialize cache
for $k = 0$ **to** $d_{forward} - 1$ **do**
  $\boldsymbol{H}^{(\ell_{\text{end}})}_{\langle k+1\rangle} \leftarrow \mathcal{F}_{\ell_{\text{start}}:\ell_{\text{end}}}\Big(\Phi\left(\boldsymbol{H}^{(\ell_{\text{start}}-1)}, \mathbf{R}_{\langle k\rangle}\right)\Big)$ // recurrent refinement
  if $k < d_{forward} - d_{backward}$ **then**
    $\boldsymbol{H}^{(\ell_{\text{end}})}_{\langle k+1\rangle} \leftarrow \mathrm{StopGrad}\Big(\boldsymbol{H}^{(\ell_{\text{end}})}_{\langle k+1\rangle}\Big)$ // truncate gradients
  $\mathbf{R}_{\langle k+1\rangle} \leftarrow \mathrm{RMSNorm}\left(\mathrm{ShiftRight}\Big(\boldsymbol{H}^{(\ell_{\text{end}})}_{\langle k+1\rangle} + \mathbf{R}_{\langle k\rangle}\Big)\right)$ // update cache
$\boldsymbol{H}^{(L)}_{\langle d_{\text{forward}}\rangle} \leftarrow \mathcal{F}_{\ell_{\text{start}}:L}\Big(\Phi\left(\boldsymbol{H}^{(\ell_{\text{start}}-1)}, \mathbf{R}_{\langle d_{\text{forward}}\rangle}\right)\Big)$ // final pass
return $\boldsymbol{H}^{(L)}_{\langle d_{forward}\rangle}$

B.1 Temporal-Parallel Approximated Training

Let $\boldsymbol{H}^{(0)} \in \mathbb{R}^{T\times d}$ be the input embeddings of a sequence of T tokens and, with slight abuse of notation, let $\mathcal{F}_{\ell_1:\ell_2} : \mathbb{R}^{T\times d} \to \mathbb{R}^{T\times d}$ be the map implemented by layers ℓ_1 through ℓ_2 on the whole sequence. Because recurrence only enters at layer ℓ_{start}, the input to that layer, $\boldsymbol{H}^{(\ell_{\text{start}}-1)} := \mathcal{F}_{1:\ell_{\text{start}}-1}(\boldsymbol{H}^{(0)})$, is exact and is computed once. We write subscripts in angle brackets for the iteration index of the recurrent cache and of the representations it induces.

Initialization. A standard forward pass without recurrence gives $\boldsymbol{H}^{(\ell_{\text{end}})}_{\langle 0\rangle} := \mathcal{F}_{\ell_{\text{start}}:\ell_{\text{end}}}(\boldsymbol{H}^{(\ell_{\text{start}}-1)})$, and its right-shifted version is the initial cache, $\mathbf{R}_{\langle 0\rangle} := \mathrm{ShiftRight}(\boldsymbol{H}^{(\ell_{\text{end}})}_{\langle 0\rangle})$. The shift aligns the state computed at token $t-1$ with position t, where it is consumed.

Refinement. We fuse $\mathbf{R}_{\langle k\rangle}$ into the exact layer-ℓ_{start} input and rerun only the recurrent block,

$$\boldsymbol{H}^{(\ell_{\text{end}})}_{\langle k+1\rangle} := \mathcal{F}_{\ell_{\text{start}}:\ell_{\text{end}}}\Big(\Phi\left(\boldsymbol{H}^{(\ell_{\text{start}}-1)}, \mathbf{R}_{\langle k\rangle}\right)\Big), \tag{B.1}$$

and form the next iterate with the cache-update rule of (2.4),

$$\mathbf{R}_{\langle k+1\rangle} = \mathrm{RMSNorm}\big(\mathrm{ShiftRight}(\boldsymbol{H}^{(\ell_{\text{end}})}_{\langle k+1\rangle} + \mathbf{R}_{\langle k\rangle})\big). \tag{B.2}$$

After d_{forward} iterations, $\mathbf{R}_{\langle d_{\text{forward}}\rangle}$ is used in a final pass through all remaining layers. Without the temporal cache residual, the update reduces to $\mathbf{R}_{\langle k+1\rangle} = \mathrm{ShiftRight}(\boldsymbol{H}^{(\ell_{\text{end}})}_{\langle k+1\rangle})$.

Why it converges. The cache at position t depends only on tokens before t, and each iteration propagates exact information one position further through the shift. The iteration is therefore a Jacobi fixed-point scheme in which each iteration extends the prefix of exact positions by one; the remaining positions are approximations whose quality we measure in Sections B.3 and B.4.

Truncated backpropagation. The whole procedure is differentiable, but backpropagating through all d_{forward} iterations is expensive. Analogous to truncated BPTT in RNN training (Williams & Zipser, 2013), we stop gradients through the layer outputs of all but the last d_{backward} iterations (Algorithm 1).

B.2 Computational Cost

Let N be the sequence length, L the number of layers, d the hidden width, and $D = \ell_{\text{end}} - \ell_{\text{start}} + 1$ the size of the recurrent block. In full-sequence training, each layer costs $O(N^2 d + N d^2)$, from self-attention and from the projections and MLPs respectively; in autoregressive decoding with a KV

Model	Training (per sequence)	Decoding (per token)
Transformer	$O\big(L\,(N^2d + Nd^2)\big)$	$O\big(L\,(Nd + d^2)\big)$
Full-looped ($\times K$)	$O\big(KL\,(N^2d + Nd^2)\big)$	$O\big(KL\,(Nd + d^2)\big)$
Middle-looped ($\times K$)	$O\big((L + (K-1)D)\,(N^2d + Nd^2)\big)$	$O\big((L + (K-1)D)\,(Nd + d^2)\big)$
T^{2}MLR	$O\big((L + d_{\text{forward}}D)\,(N^2d + Nd^2)\big)$	$\approx O\big(L\,(Nd + d^2)\big)$

Table 5: Asymptotic compute of T^{2}MLR and looped Transformers. N: sequence length; L: number of layers; d: hidden width; $D = \ell_{\text{end}} - \ell_{\text{start}} + 1$: size of the recurrent (or looped) middle block; K: number of loops. T^{2}MLR pays its extra cost only in training, and only on the recurrent block.

cache, each layer costs $O(Nd + d^2)$ per generated token. Table 5 compares T^{2}MLR with looped Transformers that apply the whole network, or a middle block of size D, K times.

Training. The Jacobi refinement reruns only the recurrent block, not the whole network, so training costs $O((L + d_{\text{forward}}D)(N^2d + Nd^2))$. The overhead therefore scales with the block size D and the depth d_{forward}, which is one practical reason to prefer localized middle-layer recurrence: at $D = 6$ of $L = 30$ layers, each refinement step costs a fifth of a full forward pass. Moreover, only the last d_{backward} iterations are kept in the autograd graph.

Decoding. At inference, no refinement is needed: the previous token's cache is exact, so each token takes a single forward pass plus one fusion-module application, and the per-token cost stays $O(L(Nd + d^2))$ up to a small constant. Looped models multiply decoding cost by the loop count.

Reinforcement learning. The trade-off is particularly favorable for on-policy RL. During rollout generation, the model already computes the exact recurrent states token by token, so these can be cached and reused during policy optimization; the temporal-parallel approximation is only needed for prefill, teacher-forced training, or replayed trajectories. And because the cache values themselves are exact, d_{forward} only needs to be large enough to build the computation graph for BPTT, so one can set $d_{\text{forward}} = d_{\text{backward}}$ and substantially reduce the overhead relative to supervised pretraining.

B.3 How Deep Must the Approximation Be?

Exactness at every position of a 2048-token sequence would require 2048 iterations. We show that far fewer suffice for training, by comparing gradients computed under different $(d_{\text{forward}}, d_{\text{backward}})$ with those of a much deeper anchor.

Setup. We take an intermediate checkpoint of the T^{2}MLR (5,26) FineWeb-Edu pretraining run (trained with $d_{\text{forward}} = 16$, $d_{\text{backward}} = 4$), sample a training batch of 2048-token sequences, and compute gradients with Algorithm 1 for every $d_{\text{backward}} \le d_{\text{forward}}$ with $d_{\text{forward}} \in \{1, 2, 4, 8, 16, 32\}$. We use $(d_{\text{forward}}, d_{\text{backward}}) = (32, 32)$ as the anchor and report the cosine similarity and the ℓ_2 distance normalized by the anchor norm (relative ℓ_2 distance), for all parameters, for the fusion module only, and for a single middle layer (Figure 6).

Forward depth is the critical knob. With $d_{\text{forward}} \le 4$, the gradients are far from the anchor: the cosine similarity over all parameters is at most 0.68 and the relative ℓ_2 distance exceeds 0.8. For the fusion module, $d_{\text{forward}} \le 2$ even yields a *negative* cosine similarity (-0.21 to -0.27), so an overly shallow approximation would push the recurrent pathway in the wrong direction. From $d_{\text{forward}} = 8$ on, the agreement jumps to a cosine similarity of at least 0.95 over all parameters whenever $d_{\text{backward}} \ge 1$, and increasing d_{forward} further brings diminishing returns.

Backward depth matters for the recurrent pathway. Given a sufficient forward depth, the backward depth mostly affects the fusion module, which learns to use the recurrent state. Truncating backpropagation through all refinement iterations ($d_{\text{backward}} = 0$) leaves the all-parameter cosine similarity at 0.88 but drops the fusion module to 0.62 with a relative ℓ_2 distance of 0.81, even at $d_{\text{forward}} = 32$. With $d_{\text{backward}} \ge 4$, the fusion-module gradient matches the anchor (cosine similarity at least 0.99, relative ℓ_2 distance at most 0.22) for every $d_{\text{forward}} \ge 8$. This supports BPTT, unlike latent-reasoning methods that do not propagate gradients to past tokens (Yue et al., 2025).

Our default $(d_{\text{forward}}, d_{\text{backward}}) = (16, 4)$ gives cosine similarity 1.00 and relative ℓ_2 distance 0.06 over all parameters (0.10 for the fusion module), backpropagating through layers in four iterations.

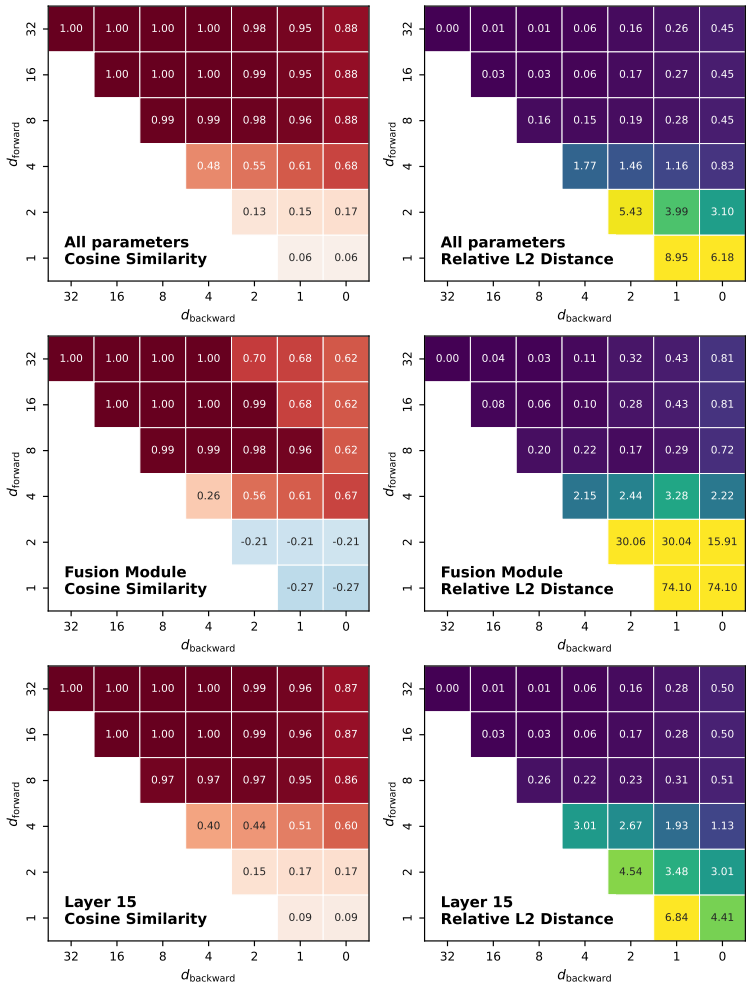

Figure 6: Gradient agreement with the $(d_{\text{forward}}, d_{\text{backward}}) = (32, 32)$ anchor for a T^{2}MLR (5,26) checkpoint, for all parameters (top), the fusion module (middle), and layer 15 (bottom). Left: cosine similarity (higher is better); right: relative ℓ_2 distance (lower is better). The forward depth d_{forward} must reach 8 for the gradients to agree; given that, $d_{\text{backward}} \geq 4$ is needed for the fusion-module gradient. Our default $(16, 4)$ reaches cosine similarity 1.00 in all three groups.

B.4 Jacobi Approximation vs. Exact Recurrence in Training

Gradient agreement at one checkpoint does not guarantee that the approximation leads to the same model after training. We therefore compare Jacobi-approximation training directly against *exact* sequential recurrent training, in which the recurrent cache is computed token by token as at inference. We train T^{2}MLR models with $\ell_{\text{start}} = 8$ at 135M, 360M, and 1B parameters on the first 1B tokens of FineWeb-Edu. Jacobi training uses $(d_{\text{forward}}, d_{\text{backward}}) = (16, 4)$ as in our main runs; exact training with full 2048-token backpropagation through time is intractable, so it uses the same truncated-BPTT budget $d_{\text{backward}} = 4$ for a controlled comparison. We evaluate each model both with exact sequential recurrence and, for Jacobi-trained models, with the Jacobi forward.

Table 6 shows three things. First, Jacobi training matches exact recurrent training at every scale: the Jacobi-trained models are within 0.0010, 0.0042, and 0.0045 validation loss of the exact-trained models at 135M, 360M, and 1B, and in fact marginally better in these single-seed runs. Second, a Jacobi-trained model gives the same loss whether it is evaluated with the Jacobi forward or with exact sequential recurrence (differences of at most 0.0001), so the approximation used in training is faithful to the computation performed at inference. Third, the benefit of the recurrent pathway

Size	Transformer	T^{2}MLR (training / evaluation)		
		Exact / exact	Jacobi / exact	Jacobi / Jacobi
135M	3.9514 (52.01)	3.9083 (49.81)	**3.9073** (49.76)	3.9073 (49.77)
360M	3.3803 (29.38)	3.3246 (27.79)	**3.3204** (27.67)	3.3205 (27.68)
1B	3.1562 (23.48)	3.1148 (22.53)	**3.1103** (22.43)	3.1102 (22.43)

Table 6: Jacobi-approximation vs. exact recurrent training ($\ell_{\text{start}} = 8$, first 1B FineWeb-Edu tokens, $d_{\text{backward}} = 4$ in both cases). Each cell is validation loss, with validation perplexity in parentheses. Jacobi-trained models match exact-trained ones at all three scales, and evaluating them with the Jacobi forward gives the same loss as exact sequential evaluation.

appears early: after only 1B tokens, T^{2}MLR improves on the parameter-matched Transformer by 0.044–0.060 validation loss at all three scales.

B.5 Inference Overhead

A central property of T^{2}MLR is that, unlike looped or pause-token models, it adds little per-token inference cost. Because the recurrent state is a single vector per sequence, all T^{2}MLR variants add less than 0.1% to peak GPU memory at inference.

Autoregressive generation. We generate from the BOS token up to 512, 1024, and 2048 tokens and report wall-clock time relative to the parameter-matched Transformer (Table 7). The overhead is between 4.1% and 8.2% in all settings. It does not grow with the recurrent block size: T^{2}MLR (1,30), which recurs over all 30 layers, is no slower than T^{2}MLR (9,22), because the only added work is the fusion module, whose cost is independent of D. The overhead generally shrinks with model size, to 4.1% for the 1B model at 2048 tokens, as the constant cost of the fusion module is increasingly dominated by attention and MLP compute.

Generated tokens	135M			361M	1B
	T^{2}MLR (1,30)	T^{2}MLR (5,26)	T^{2}MLR (9,22)	T^{2}MLR (9,24)	T^{2}MLR (9,24)
512	$1.070\times$	$1.076\times$	$1.082\times$	$1.065\times$	$1.067\times$
1024	$1.064\times$	$1.068\times$	$1.078\times$	$1.057\times$	$1.057\times$
2048	$1.067\times$	$1.076\times$	$1.078\times$	$1.056\times$	$1.041\times$

Table 7: Autoregressive generation time of T^{2}MLR relative to the parameter-matched Transformer ($1.0\times$). Overhead is at most 8.2%, independent of block size, and generally shrinks with model size.

Prefill. The results in the main paper use exact sequential prefill of the prompt. The Jacobi approximation can also be applied to prefill, which keeps prefill parallel over the prompt at the cost of a small approximation error. Table 8 reports the zero-shot average under Jacobi prefill with different d_{forward}, with the relative prefill time in parentheses. From $d_{\text{forward}} = 8$ on, Jacobi prefill recovers the exact-prefill average to within 0.2 points for every variant, while $d_{\text{forward}} = 4$ costs up to 0.7 points.

Model	d_{forward}=4	d_{forward}=8	d_{forward}=16	d_{forward}=32	Exact
T^{2}MLR (5,26)	43.6 ($3.6\times$)	43.6 ($6.2\times$)	43.5 ($11.1\times$)	43.6 ($20.5\times$)	43.7
T^{2}MLR (9,22)	43.6 ($2.7\times$)	44.1 ($4.6\times$)	43.9 ($8.0\times$)	44.0 ($14.5\times$)	44.0
T^{2}MLR (13,18)	43.4 ($1.8\times$)	44.0 ($2.7\times$)	44.1 ($4.3\times$)	44.1 ($7.7\times$)	44.1
T^{2}MLR (15,16)	43.2 ($1.3\times$)	43.0 ($1.7\times$)	43.0 ($2.5\times$)	42.9 ($4.0\times$)	42.9
Transformer	–	–	–	–	42.8

Table 8: Jacobi vs. exact prefill at 135M. Each cell is the zero-shot average (%) over the seven benchmarks of Section 4, with the relative prefill time in parentheses. From $d_{\text{forward}} = 8$ on, Jacobi prefill matches exact prefill to within 0.2 points.

C Additional Empirical Results

We report per-benchmark scaling results from Section 4.1 (Section C.1), the full 135M results including the fixed-width location ablation (Section C.2), and a compute-matched baseline (Section C.3).

C.1 Scaling to Larger Models and More Data

We test whether the gains of middle-layer recurrence persist beyond the 135M, 10B-token setting of the main text along two axes: model size (361M and 1B parameters on 10B tokens) and pretraining data (135M and 361M parameters on 50B tokens). The 361M and 1B models have 32 layers and use T^{2}MLR (9,24); the 135M model uses T^{2}MLR (9,22); all use $(d_{\text{forward}}, d_{\text{backward}}) = (16, 4)$. The baselines have 367.8M and 996.9M parameters against 361.8M and 981.5M for T^{2}MLR. Every T^{2}MLR model is compared with a Transformer of matching size trained on the same data. Table 1 in the main text gives per-benchmark zero-shot and GSM-Aug results; Table 9 adds HotpotQA-Easy and the relative improvements.

Zero-shot. T^{2}MLR improves the zero-shot average in all four settings: by 0.74 and 0.99 points at 361M and 1B on 10B tokens, and by 1.07 and 1.95 points at 135M and 361M on 50B tokens (Table 1). The gains are not driven by a single benchmark. HellaSwag improves in every setting (by 1.5 to 2.3 points) and so does Winogrande (by up to 4.2 points), whereas ARC-Easy is the one benchmark on which the baseline is ahead in all four settings, by 0.2 to 1.2 points. Longer pretraining does not erode the advantage: the largest zero-shot gain is at 361M with 50B tokens.

Setting	Model	HotpotQA-Easy	GSM-Aug-NL	GSM-Aug-Sym
361M / 10B	Baseline	24.43	31.08	39.35
	T^{2}MLR (9,24)	**28.28** (+15.8%)	**34.12** (+9.8%)	**42.46** (+7.9%)
1B / 10B	Baseline	23.28	32.37	43.97
	T^{2}MLR (9,24)	**26.52** (+13.9%)	**36.69** (+13.3%)	**44.96** (+2.3%)
135M / 50B	Baseline	–	26.46	38.67
	T^{2}MLR (9,22)	–	**30.48** (+15.2%)	**41.55** (+7.4%)
361M / 50B	Baseline	–	40.18	42.53
	T^{2}MLR (9,24)	–	**44.35** (+10.4%)	**46.63** (+9.6%)

Table 9: Accuracy after finetuning the scaled checkpoints of Table 1 on HotpotQA-Easy and on the natural-language (NL) and symbolic (Sym) variants of GSM-Aug, with relative improvement over the baseline in parentheses. T^{2}MLR improves on the baseline on every task and in every setting.

Reasoning after finetuning. The gains after finetuning are larger in relative terms (Table 9): $+14$ to $+16\%$ on HotpotQA-Easy, $+10$ to $+15\%$ on the natural-language GSM-Aug, and $+2$ to $+10\%$ on its symbolic variant, in every setting where each task was evaluated. As in the main text, the downstream reasoning gains exceed what the zero-shot and perplexity differences alone would suggest, consistent with the view that perplexity understates changes in the model's internal computation (Section 4.4).

Perplexity. T^{2}MLR also lowers validation perplexity after pretraining on 10B tokens, from 22.64 to 21.54 (-4.9%) at 135M and from 16.12 to 15.79 (-2.0%) at $\sim$370M.

C.2 Full 135M Results and Recurrence Location

Table 10 gives per-benchmark zero-shot results for every 135M model summarized in Table 2: the five centered recurrence blocks, which change the width D and the location of the block together; fixed-width blocks of $D \in \{6, 14\}$ placed over the earliest or latest layers, which isolate location; and the looped and pause-token baselines of Section 4.3. All models have about 136M parameters and are pretrained on the same 10B tokens.

At both widths, middle placement is best: 44.14 versus 42.74 (early) and 42.87 (late) at $D = 6$, and 44.01 versus 42.21 and 43.14 at $D = 14$. The other placements do not merely help less. Early-layer recurrence falls *below* the baseline (42.83) at both widths, and late-layer recurrence improves on it by at most 0.3 points. This matches the functional picture motivating T^{2}MLR (Section 2.1). Early

Group	Model	ARC-C	ARC-E	HS	OBQA	PIQA	SciQ	WG	Avg.
–	Transformer	24.74	44.28	29.81	30.20	61.53	60.80	48.46	42.83
Centered	T^{2}MLR (1,30), D=30	24.49	43.48	29.98	30.00	60.72	62.60	**52.25**	43.36
	T^{2}MLR (5,26), D=22	23.98	**46.13**	**30.75**	29.80	60.77	62.80	51.85	43.73
	T^{2}MLR (9,22), D=14	24.15	45.24	30.40	29.20	61.15	**66.40**	51.54	44.01
	T^{2}MLR (13,18), D=6	24.23	45.50	29.95	31.20	**61.70**	64.20	52.17	**44.14**
	T^{2}MLR (15,16), D=2	24.40	45.83	30.11	29.20	59.63	60.20	50.83	42.89
D=6	T^{2}MLR (1,6), early	24.66	45.08	29.83	30.00	60.07	61.00	48.54	42.74
	T^{2}MLR (25,30), late	**25.09**	45.24	29.65	28.20	60.12	61.20	50.59	42.87
D=14	T^{2}MLR (1,14), early	23.38	44.70	28.59	29.00	59.25	59.50	51.07	42.21
	T^{2}MLR (17,30), late	23.98	44.02	29.97	31.40	60.28	61.90	50.43	43.14
Baselines	Pause-token ×2	24.74	44.57	29.51	**31.60**	60.23	61.90	50.59	43.31
	Full-looped ×2	24.83	44.95	29.84	30.80	60.23	60.70	49.57	42.99
	Middle-looped ×3	23.04	45.16	30.12	30.20	60.17	59.80	50.28	42.68

Table 10: Per-benchmark zero-shot results for all 135M models (10B FineWeb-Edu tokens), summarized in Table 2. Bold marks the best value in each column. The centered blocks change width and location together; the D=6 and D=14 rows place blocks of the same width as T^{2}MLR (13,18) and T^{2}MLR (9,22) over the earliest or latest layers.

layers perform lexical processing, so injecting a state there plausibly perturbs computation that should depend mainly on the current token. Late layers are specialized toward the output vocabulary, so a state taken from them likely carries little beyond the next-token distribution, much like the embedding-level feedback of prior latent-reasoning methods. Middle layers host the most abstract computation (Skean et al., 2025; Saunshi et al., 2024) and benefit most from temporal state propagation.

Validation loss across recurrence boundaries. Figure 7 shows the validation loss of the 135M T^{2}MLR variants as a function of the recurrence start layer; each variant recurs over a block centered in the network, so a smaller ℓ_{start} means a wider block. Most configurations attain lower loss than the baseline, except $\ell_{\text{start}} = 1$ (full recurrence with $D = 30$) and $\ell_{\text{start}} = 15$ (recurrence over only $D = 2$ layers). Apart from these two exceptions, the loss generally improves with the number of recurrent layers, a different trend from the downstream results. This suggests a *distinct implicit bias of middle layers* on reasoning that perplexity may not capture, resembling the observation of Saunshi et al. (2024) for middle-layer stacking.

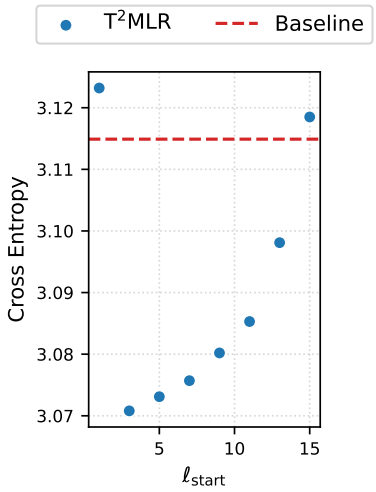

Figure 7: Validation loss vs. recurrence start layer ℓ_{start} for 135M T^{2}MLR runs on FineWeb-Edu at 16k steps (dashed: Transformer baseline). Smaller ℓ_{start} means a wider block.

C.3 Training-Compute-Matched Baseline

Our main comparisons are matched on parameters, data, and inference compute, but *not* on training compute: the Jacobi approximation makes T^{2}MLR slower to train (Section B.2). For a training-compute-matched comparison, we train the Transformer baseline for 2.24 epochs over the same 10B tokens, matching the $2.24\times$ training wall-clock of T^{2}MLR (13,18) (Table 11).

Model	ARC-C	ARC-E	HS	OBQA	PIQA	SciQ	WG	Avg.
Transformer (1 epoch)	24.74	44.28	29.81	30.20	61.53	60.80	48.46	42.83
T^{2}MLR (13,18) (1 epoch)	24.23	45.50	29.95	31.20	61.70	64.20	52.17	44.14
Transformer (2.24 epochs)	**25.68**	**46.30**	**32.15**	**31.80**	**62.40**	**66.50**	**52.25**	**45.30**

Table 11: Training-compute-matched comparison at 135M. The Transformer trained for 2.24 epochs matches the training wall-clock of T^{2}MLR (13,18) and surpasses it on zero-shot NLP; the advantage of T^{2}MLR holds at fixed parameter, data, and inference-compute budgets.

Under this matched training budget, the longer-trained Transformer surpasses T^{2}MLR (13,18) on zero-shot NLP (45.30 vs. 44.14). This result delimits our claims. T^{2}MLR is an *inference-side and architectural* contribution: at fixed parameter, data, and inference-compute budgets, it adds a temporal latent pathway with little decoding cost (at most $\sim$8% per token, Section B.5) and gains on state tracking and multi-hop reasoning, at the cost of additional training compute. The longer-trained baseline also sees each training token more than twice, so this comparison trades data matching for compute matching. Reducing the training cost of T^{2}MLR is our main future direction (Section 6).

D Analysis of T^{2}MLR

This section collects analyses of what T^{2}MLR can represent and what it learns: its relation to Transformers and recurrent networks (Section D.1), a probing study of its intermediate representations (Section D.2), and the behavior of its learned gates (Section D.3).

D.1 T^{2}MLR as a Generalized Transformer and Recurrent Model

T^{2}MLR contains both recurrent neural networks and the standard Transformer as special cases. When learned self-attention attends only to the current token, token interactions occur exclusively through the recurrent pathway and T^{2}MLR becomes a recurrent model whose state is the cache $\mathbf{R}_t$. Conversely, when the global gates $\tanh(\gamma_{\text{cur}})$ and $\tanh(\gamma_{\text{rec}})$ are zero, the recurrent pathway is disabled and T^{2}MLR recovers the standard Transformer, its exact state at initialization (Section 2.3).

T^{2}MLR can therefore interpolate between recurrent and attention-based computation. Transformers excel at in-context retrieval (Jelassi et al., 2024; Nichani et al., 2025), while recurrent models are well suited to state tracking (Merrill et al., 2024; Grazzi et al., 2025); T^{2}MLR can use both within one network. The S_5-Retrieval experiment (Section 3) tests whether gradient-based training actually finds such a combined solution, and it does: within the training lengths, a 4-layer T^{2}MLR reaches near-perfect exact match on a task that neither a parameter-matched Transformer nor an LSTM learns.

D.2 Probing Intermediate Representations

Temporal middle-layer recurrence creates an explicit temporal shortcut: the recurrent cache from token t is fused into the residual stream for token $t{+}1$, so the loss at position $t{+}1$ backpropagates through this pathway into token t's intermediate representation. This should encourage the model to maintain latent states that anticipate upcoming tokens. To test this, we extract the representation of token t from each of the top layers and train a two-layer MLP probe to predict the next token x_{t+1}. We probe two T^{2}MLR models whose recurrence ends 3 and 5 layers below the LM head, and a parameter-matched Transformer. Negative indices count back from the LM head: $\ell_{\text{end}} = -k$ places the recurrence endpoint k layers below the head, and ℓ_{-j} denotes the j-th layer from the top.

The T^{2}MLR representations yield lower probe loss than the baseline at every probed layer and for both recurrence endpoints (Table 12), by 0.024 to 0.047 nats. The gap is not confined to the layer where

	T^{2}MLR with $\ell_{\text{end}} = -3$			T^{2}MLR with $\ell_{\text{end}} = -5$				
Probed layer	ℓ_{-1}	ℓ_{-2}	ℓ_{-3}	ℓ_{-1}	ℓ_{-2}	ℓ_{-3}	ℓ_{-4}	ℓ_{-5}
T^{2}MLR	**5.091**	**5.088**	**5.110**	**5.097**	**5.096**	**5.109**	**5.126**	**5.158**
Transformer	5.135	5.131	5.145	5.135	5.131	5.145	5.173	5.182

Table 12: Next-token probe loss (↓) from the top layers of two T^{2}MLR models and a parameter-matched Transformer, whose probes are shared across column groups. Each probe is a two-layer MLP trained for 20,000 steps on FineWeb-Edu. T^{2}MLR has lower loss at every probed layer.

the recurrent state is read out: it is already present at the top layer ℓ_{-1}, whose representation feeds the LM head. This suggests that the recurrent pathway shapes the model's intermediate representations toward anticipating upcoming tokens, rather than adding a side channel the output layer ignores.

D.3 Learned Gating Behavior

The fusion module in (2.3) learns two scalar, input-independent gates: γ_{cur} scales the input-stream correction and γ_{rec} scales the recurrent contribution, both relative to the residual branch $\boldsymbol{h}_t^{(\ell_{\text{start}}-1)}$. We track these gates over training to see how the model allocates the two streams.

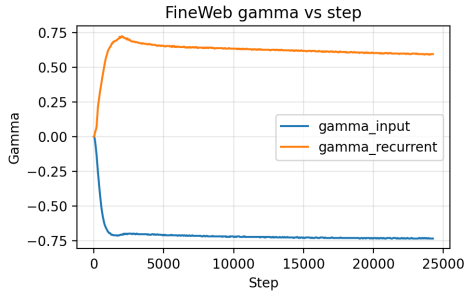

Figure 8: Global gates γ_{cur} (input) and γ_{rec} (recurrent) over FineWeb-Edu pretraining. Both start at zero, so training begins from the backbone Transformer; within the first few thousand steps γ_{rec} becomes positive and γ_{cur} negative, and both then change only slowly.

Across all tasks we study, γ_{rec} converges to positive values and γ_{cur} to negative values; Figure 8 shows this for FineWeb-Edu pretraining. The asymmetry follows from the structure of (2.3). Since the input representation $\boldsymbol{h}_t^{(\ell_{\text{start}}-1)}$ is already added unconditionally through the residual connection, the γ-gated terms do not need to reintroduce it; instead they *modulate deviations from the identity map*. A negative γ_{cur} suppresses redundant amplification of the input stream, and a positive γ_{rec} injects information from previous steps where it adds predictive value beyond the current token. T^{2}MLR thus learns to treat recurrence as an additive refinement of the residual stream rather than a competing source of information: it keeps the Transformer computation as the default and adds temporally accumulated latent information when useful.

Because both gates are initialized to zero, the fusion module starts as the identity and T^{2}MLR starts training exactly as a standard Transformer, departing from it only as the gates adapt. This stabilizes optimization and lets recurrent computation be introduced gradually rather than imposed at initialization; it also makes retrofitting a pretrained model natural (Section 4.5), since inserting the pathway leaves the pretrained function unchanged at the start of finetuning.

E Experimental Setup

E.1 Pretraining and Zero-Shot Evaluation

Pretraining. All from-scratch models are pretrained on FineWeb-Edu (Penedo et al., 2024) with AdamW (Loshchilov & Hutter, 2019) ($\beta_1 = 0.9$, $\beta_2 = 0.98$, weight decay 0.01) and a cosine

learning-rate schedule that decays to 0.001 times the peak learning rate. We pack the data into sequences of 2048 tokens and train with batch size 256, so one epoch over the 10B-token subset takes about 19k steps. Unless stated otherwise, T^{2}MLR uses $(d_{\text{forward}}, d_{\text{backward}}) = (16, 4)$. All models are trained on $4\times$H100 GPUs. Table 13 lists the per-task hyperparameters.

Zero-shot evaluation. We evaluate pretrained models with lm-eval-harness (Gao et al., 2024) on ARC-Challenge and ARC-Easy (ARC-C/E; Clark et al., 2018), HellaSwag (HS; Zellers et al., 2019), OpenBookQA (OBQA; Mihaylov et al., 2018), PIQA (Bisk et al., 2020), SciQ (Welbl et al., 2017), and Winogrande (WG; Sakaguchi et al., 2021), a standard suite in the pretraining literature. We report normalized accuracy where available and accuracy otherwise.

Task	Model	LR	Schedule	Steps
S_5-Retrieval	Transformer: 4 layers, 6 heads, width 384 LSTM: 4 layers, hidden size 456	5e-4	Cosine	200k
FineWeb-Edu pretraining	SmolLM2-135M	5e-4	Cosine w/ min LR	$\sim$20k
GSM-Aug finetuning	SmolLM2-135M	5e-5	Cosine	8000
HotpotQA finetuning	SmolLM2-135M	1e-4	Cosine w/ min LR	5000

Table 13: Training hyperparameters. The S_5 Transformer uses RoPE. Finetuning runs use AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and weight decay 0.01.

E.2 Finetuning Data

HotpotQA. To construct SFT data with step-by-step multi-hop reasoning, we take a subset of HotpotQA (Yang et al., 2018) and generate 19k chain-of-thought traces with GPT-4o-mini (OpenAI, 2024), using the prompt below. The prompt includes the gold answer so that the generated reasoning chain ends at the correct answer. We restrict our experiments to the *easy* subset: in preliminary experiments, both the baselines and T^{2}MLR struggle on the medium and hard subsets at this model scale, precluding meaningful comparisons.

Prompt for generating HotpotQA chains of thought

According to the context, answer the question in a multi-hop reasoning process. Be sure to include a detailed and clear logic chain in your reasoning. Detail your thought process at each step. Put your final answer within \boxed{}. Output example: \boxed{Knox County Regional Airport}.
Question: {*question*}
Context: {*context*}
Note that the correct final answer is \boxed{*answer*}.

Variable assignment. From the reasoning-primitives suite of Saunshi et al. (2024): given a chain of variable assignments, the model outputs the value of a query variable. Baselines saturate the original depth-2 task, so we use depth 5.

ProsQA-Hard. ProsQA (Hao et al., 2024) provides a directed acyclic graph and asks which of two candidate end nodes a start node connects to; the output walks the connecting path. Baselines saturate the original data, so we regenerate it with 60 nodes and average path length 8.

GSM-Aug. Synthetic GSM8K-style grade-school math (Deng et al., 2023) in two variants: a symbolic one containing only the arithmetic, and a natural-language one closer to ordinary chain-of-thought. Table 14 shows one example per dataset.

Dataset	Details
ProsQA-Hard	**Input:** Every zolufibus is a conarus. Every conarus is a tanirus. Every tanirus is a gedatus. Every cemapus is a lovaperus. Every lovaperus is a zucirarus. Every zucirarus is a pilevus. Every sasarus is a madusurus. Every madusurus is a cidodivus. Every cidodivus is a bumobus. Every bumobus is a nuronus. Every nuronus is a metuvelus. Every metuvelus is a resamus. Every resamus is a gedatus. Jordan is a sasarus. Is Jordan a gedatus or a pilevus? **Steps:** Jordan is a sasarus. Every sasarus is a madusurus. Every madusurus is a cidodivus. Every cidodivus is a bumobus. Every bumobus is a nuronus. Every nuronus is a metuvelus. Every metuvelus is a resamus. Every resamus is a gedatus. **Output:** Jordan is a gedatus.
S5 Retrieval	**Input:** `<A_32514>F4Nd <A_12543>9O8W <A_54213>Jccq	<A_54213> <A_43125> <A_52314>` **Target:** `<A_32514>F4Nd <A_12543>9O8W <A_54213>Jccq	<A_54213>Jccq <A_12543>9O8W <A_32514>F4Nd`
GSM-Aug Natural Language	**Question:** Out of 600 employees in a company, 30% got promoted while 10% received bonus. How many employees did not get either a promotion or a bonus? **Steps:** 600 x 30/100 = 180 employees were promoted. 600 x 10/100 = 60 employees received a bonus. So a total of 180+60=240 employees received a promotion or a bonus. Therefore, 600 - 240 = 360 employees did not get either a promotion or a bonus. **Answer:** 360
Variable Assignment	**Input:** Variable assignment. Follow the assignments and fill in the blank. a=5 b=2 c=8 x=c y=x z=y w=b v=w z=___ Answer: | **Outputs:** 8
HotpotQA	**Input:** Which magazine was founded first, *The Economist* or *The Atlantic*? **Context:** Document 1: *The Economist* is a weekly newspaper founded in 1843. Document 2: *The Atlantic* is a magazine founded in 1857. **Steps:** The Economist was founded in 1843. The Atlantic was founded in 1857. 1843 is earlier than 1857. **Answer:** The Economist

Table 14: Examples for each dataset used.