ENHANCING DEEP RESIDUAL LLMS WITH DEPTH-RECURRENT MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Residual connections have been a foundational design principle in large language models (LLMs), enabling the residual stream to accumulate layer-wise information across depth through unit-weight additions. Although this accumulation provides a stable information pathway, its fixed additive form offers limited explicit and input-dependent control over the evolving residual content. Recent studies have introduced constrained mixing or attention mechanisms to make residual information flow more adaptive. However, they still rely on the residual stream or stored intermediate layer representations as the carrier of cross-layer information, without an independent layer-wise memory and an explicit mechanism for controlling what to write, retain, and expose across depth. To fill this gap, we propose Depth-Recurrent Memory (DRM), a recurrent memory module whose state evolves along model depth. DRM maintains two fixed-width memory states per token, one for the attention pathway and one for the MLP pathway, and injects their gated readouts into the residual stream as additive updates. This provides explicit recurrent memory management while preserving the standard backbone sublayers and residual pathways. We instantiate the recurrent transitions with LSTM-style gating and co-design lightweight input projections with a parallel-residual implementation to control parameter and computational overhead and enable concurrent execution with backbone sublayers. Experimental results show that DRM outperforms the standard residual baseline on 16 of 21 benchmarks, with the strongest gains in math and code. In addition, the ablation studies validate the effectiveness of core components of DRM.

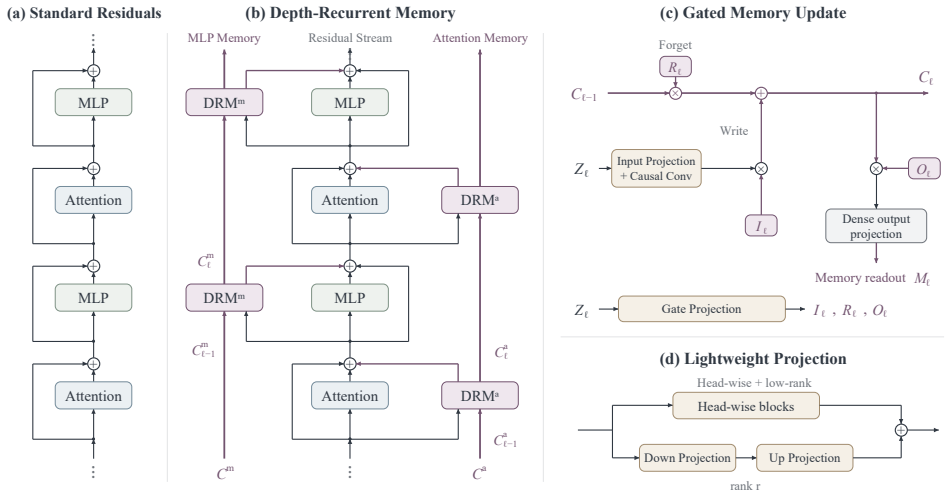


Figure 1: **Overview of DRM.** *(a)* Standard residuals aggregate updates with fixed weights. *(b)* DRM propagates two compact per-token states along depth for attention and MLP pathways. *(c)* LSTM-style gates control state writing, retention, and exposure. *(d)* Head-wise low-rank projections with dense output projection.

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.

1 INTRODUCTION

Residual connections are central to deep networks and have been widely adopted in Transformers and LLMs, as their identity shortcuts ease optimization while preserving an information pathway across depth (He et al., 2016; Vaswani et al., 2017; Touvron et al., 2023). Beyond serving as a gradient highway, the residual stream also acts as an implicit information accumulator across depth. Given a hidden representation h_l, a standard residual layer updates it as $h_{l+1} = h_l + \mathcal{F}_l(h_l)$. Unrolling this recurrence gives $h_l = h_0 + \sum_{i=0}^{l-1} \mathcal{F}_i(h_i)$, showing that the hidden state aggregates all preceding layer updates through fixed unit-weight additions. Although each transformation $\mathcal{F}_i$ can implicitly modify the residual stream, the architecture provides no explicit and input-dependent mechanism for deciding what information should be written, retained, or exposed across depth. Consequently, the same residual stream must simultaneously preserve previously acquired information and accommodate new layer updates, without a dedicated persistent state for controlling memory.

An effective mechanism for managing information across depth should therefore satisfy three requirements. First, it should preserve the original residual pathway and its optimization benefits. Second, it should provide explicit and input-dependent control over memory writing, retention, and exposure. Third, it should maintain a fixed-size state across layers, without storing or aggregating representations from multiple preceding layers, so that the cross-layer memory footprint and communication cost remain bounded as model depth increases. This bounded-state property is important for scaling to deeper and larger LLMs, where retaining a history of intermediate representations can incur increasing memory, bandwidth, and synchronization overhead.

Recent work has made residual information flow more adaptive through multi-stream mixing (e.g., hyper-connections (Zhu et al., 2025) and mHC (Xie et al., 2026b)) or attention-based aggregation over representations from different depths (e.g., attention residuals (AttnRes) (Team et al., 2026) and mixture-of-depths attention (Zhu et al., 2026)). Multi-stream methods enlarge the residual space and improve cross-depth exchange, while attention-based methods provide content-dependent access to prior layer states. However, the former still relies on residual streams as the information carrier, and the latter requires maintaining and aggregating histories of layer- or block-level representations. Explicit memory control that preserves the original residual pathway while propagating only a fixed-size state across layers remains underexplored.

To address these challenges, we introduce Depth-Recurrent Memory (DRM), a lightweight recurrent memory module that maintains a fixed-size state and updates it recurrently along model depth. As illustrated in Figure 1, DRM preserves the standard residual pathway while augmenting each layer with two DRM branches, one for the attention pathway and the other for the multi-layer perceptron (MLP) pathway. Specifically, DRM maintains two compact states for each token and propagates them independently across LLM layers, thereby forming two recurrent state chains over depth. In our implementation, each recurrent transition uses LSTM-style input, forget, and output gates to determine what information should be written, retained, or exposed. These gates instantiate the state-transition rule of DRM rather than define its recurrent-memory formulation. To reduce computational and parameter overhead, the gate and candidate-write projections are implemented with head-wise projections augmented by grouped low-rank bypasses, while the output projection remains a standard dense linear. Furthermore, a parallel-residual design allows DRM to be computed alongside the backbone modules. Different from attention-based aggregation over previous layers, each DRM pathway depends only on its corresponding compact state from the immediately preceding layer, avoiding complex cross-layer interactions as well as the storage and aggregation of multiple intermediate representations. Our contributions are summarized as follows:

- We propose DRM, a low-cost and scalable enhancement to residual information flow that propagates two compact recurrent states along model depth, without constraining residual updates or introducing complex cross-layer interactions.
- We instantiate DRM with a gated state transition and develop head-wise low-rank gate and candidate-write projections with a parallel-residual design, reducing parameter and computational overhead while enabling concurrent execution with backbone modules.
- We demonstrate the effectiveness of DRM through scaling-law experiments and the pretraining of 7.9B Ling-3.0-tiny (inclusionAI, 2026). Ablation studies further validate the recurrent-state organization, lightweight projection, gated update, and parallel design.

2 Related Work

2.1 Cross-Layer Connectivity

Cross-layer connectivity has been widely studied for improving optimization and information flow in deep networks (Xie et al., 2026a). Existing methods can be roughly grouped into three categories. First, local residual designs modify the interaction between the residual branch and the identity path, such as Highway Networks (Srivastava et al., 2015), ResNet (He et al., 2016), and DenseNet (Huang et al., 2017). Second, multi-stream methods enlarge the residual pathway and learn cross-stream mixing, including hyper-connections (Zhu et al., 2025) and mHC (Xie et al., 2026b). Third, non-local aggregation methods allow layers to access representations from other depths, such as transparent attention (Bapna et al., 2018), RealFormer (He et al., 2021), AttnRes (Team et al., 2026), and mixture-of-depths attention (Zhu et al., 2026). While effective, constrained mixing may limit representational flexibility, and non-local aggregation increases cross-layer dependencies and activation storage. In contrast, DRM preserves unrestricted residual updates and recurrently propagates only a compact state between adjacent layers, enabling adaptive memory control with better scalability.

2.2 Recurrent Architectures

Recurrent architectures can be broadly divided into sequence-wise recurrence over token positions and depth-wise recurrence over layers or computation steps. Sequence-wise models, including LSTM (Hochreiter & Schmidhuber, 1997), GRU (Cho et al., 2014), xLSTM (Beck et al., 2024), Gated DeltaNet (Yang et al., 2025), and Titans (Behrouz et al., 2025), maintain compact states to summarize token histories and improve long-context modeling. Depth-wise approaches, such as universal transformers (Dehghani et al., 2019) and deep equilibrium models (Bai et al., 2019), apply recurrent refinement across layers or computation steps, often through shared transformations or fixed-point computation. DRM belongs to the depth-wise recurrent family, i.e., its recurrent index is the layer index, and each memory state at layer ℓ is computed from the state at layer $\ell - 1$ together with the current layer input. Unlike recurrent-depth models based on repeated or shared transformations, DRM augments a standard Transformer with layer-specific recurrent memory transitions while preserving the original backbone computation. In addition, because recurrence is performed across depth rather than token positions, DRM preserves token-level parallelism. The LSTM-style gates used in this work are a particular implementation of this depth-recurrent transition.

3 Method

3.1 Overall Framework

DRM augments the residual backbone with dedicated memory states that evolve along model depth. The residual stream remains the working representation processed by the backbone, whereas the memory states selectively accumulate layer-wise writes and expose their contents through additive residual updates. DRM maintains separate memory chains for the attention and MLP pathways. Both branches reuse the normalized input of their corresponding backbone sublayers and contribute only an additive residual update. Consequently, DRM preserves the original Transformer computation and extends it with explicit recurrent states propagated from one layer to the next.

Let L denote the number of LLM layers, T the sequence length, d the hidden width, and d_m the memory width. We use $\boldsymbol{X}_\ell \in \mathbb{R}^{T\times d}$ to denote the residual stream entering layer ℓ. We first compute $\widetilde{\boldsymbol{X}}_\ell = \mathrm{Norm}^a_\ell(\boldsymbol{X}_\ell)$ and $(\boldsymbol{M}^a_\ell, \boldsymbol{C}^a_\ell) = \mathcal{M}^a_\ell(\widetilde{\boldsymbol{X}}_\ell, \boldsymbol{C}^a_{\ell-1})$. The attention-side residual update is

$$\boldsymbol{U}_\ell = \boldsymbol{X}_\ell + \mathrm{Attn}_\ell\left(\widetilde{\boldsymbol{X}}_\ell\right) + \gamma^a_\ell \boldsymbol{M}^a_\ell. \tag{1}$$

Then, we compute $\widehat{\boldsymbol{U}}_\ell = \mathrm{Norm}^m_\ell(\boldsymbol{U}_\ell)$ and $(\boldsymbol{M}^m_\ell, \boldsymbol{C}^m_\ell) = \mathcal{M}^m_\ell(\widehat{\boldsymbol{U}}_\ell, \boldsymbol{C}^m_{\ell-1})$. The MLP-side residual update is

$$\boldsymbol{X}_{\ell+1} = \boldsymbol{U}_\ell + \mathrm{MLP}_\ell\left(\widehat{\boldsymbol{U}}_\ell\right) + \gamma^m_\ell \boldsymbol{M}^m_\ell. \tag{2}$$

Here, $\boldsymbol{C}^a_\ell, \boldsymbol{C}^m_\ell \in \mathbb{R}^{T\times d_m}$ are the attention-side and MLP-side memory states, while $\boldsymbol{M}^a_\ell, \boldsymbol{M}^m_\ell \in \mathbb{R}^{T\times d}$ are their additive residual updates. The two branches are separated in their recurrent memory

flow. The attention-side branch receives memory information only from $\boldsymbol{C}^a_{\ell-1}$, and the MLP-side branch receives memory information only from $\boldsymbol{C}^m_{\ell-1}$, with no cross-branch state transfer. Their parameters are also independent, both across branches and across layers, so each $\mathcal{M}^a_\ell$ and $\mathcal{M}^m_\ell$ is a layer-specific transition. Thus, for each pathway, the layer index ℓ serves as the recurrent step, i.e., the state at layer ℓ is computed from the current layer input and the state propagated from the same pathway at layer $\ell-1$. The learnable scalars γ^a_ℓ and γ^m_ℓ control the contribution of each branch.

The backbone attention and MLP sublayers are not modified internally. Each memory branch only reads the same normalized activation as its host sublayer and joins the backbone computation at the final residual addition. This parallel-residual structure preserves the original residual pathway and enables concurrent execution between each memory branch and its corresponding sublayer. PyTorch-style pseudocode of DRM is provided in Appendix A.

3.2 Depth-Recurrent Memory Update

Next, we describe the recurrent state transition of a generic DRM branch. Let $\boldsymbol{Z}_\ell \in \mathbb{R}^{T\times d}$ denote the normalized input at recurrent step ℓ, where the recurrent steps correspond to model layers. For the attention-side branch, $\boldsymbol{Z}_\ell = \widetilde{\boldsymbol{X}}_\ell$, whereas for the MLP-side branch, $\boldsymbol{Z}_\ell = \widehat{\boldsymbol{U}}_\ell$. Our implementation instantiates this transition with LSTM-style input, forget, and output gates together with a candidate write. Specifically, the gate logits are computed as $\left[\boldsymbol{Q}^i_\ell; \boldsymbol{Q}^f_\ell; \boldsymbol{Q}^o_\ell\right] = \Phi^{\text{gate}}_\ell(\boldsymbol{Z}_\ell)$, and the raw candidate write is $\boldsymbol{G}^{\text{raw}}_\ell = \Phi^{\text{in}}_\ell(\boldsymbol{Z}_\ell)$. Because the recurrent state evolves across depth, the candidate write would otherwise be formed independently at each token position. Therefore, we apply a lightweight depthwise causal convolution along the sequence dimension to inject short-range local context into the write signal, while keeping the operation channel-wise, causal, and inexpensive. The locally contextualized candidate is $\overline{\boldsymbol{G}}_\ell = \mathrm{DWConv}_K\left(\boldsymbol{G}^{\text{raw}}_\ell\right)$. Here, $\boldsymbol{Q}^i_\ell, \boldsymbol{Q}^f_\ell, \boldsymbol{Q}^o_\ell, \boldsymbol{G}^{\text{raw}}_\ell, \overline{\boldsymbol{G}}_\ell \in \mathbb{R}^{T\times d_m}$. The projection $\Phi^{\text{gate}}_\ell : \mathbb{R}^d \to \mathbb{R}^{3d_m}$ jointly produces the three gate logits, while $\Phi^{\text{in}}_\ell : \mathbb{R}^d \to \mathbb{R}^{d_m}$ produces the candidate write. Both projections are implemented using the lightweight head-wise low-rank projections introduced in Section 3.3.

We use lowercase bold symbols to denote the rows corresponding to token position t. For example, $\boldsymbol{q}^i_{\ell,t} \in \mathbb{R}^{d_m}$ and $\boldsymbol{g}^{\text{raw}}_{\ell,t} \in \mathbb{R}^{d_m}$ are the t-th rows of $\boldsymbol{Q}^i_\ell$ and $\boldsymbol{G}^{\text{raw}}_\ell$, respectively. The depthwise causal convolution operates along the sequence dimension with a small kernel size K, and its output at position t is denoted by $\overline{\boldsymbol{g}}_{\ell,t}$.

Instead of directly applying a sigmoid to the forget-gate logits, we parameterize the retention factor using an input-dependent exponential decay. In particular, we first compute $\boldsymbol{\Delta}_{\ell,t} = \mathrm{softplus}(\boldsymbol{q}^f_{\ell,t} + \boldsymbol{b}_{\Delta,\ell})$. The retention factor is then parameterized as

$$\boldsymbol{R}_{\ell,t} = \exp\left(-\exp(\boldsymbol{a}_\ell) \odot \boldsymbol{\Delta}_{\ell,t}\right). \tag{3}$$

Here, $\boldsymbol{a}_\ell, \boldsymbol{b}_{\Delta,\ell} \in \mathbb{R}^{d_m}$ are learnable channel-wise parameters. Because both $\exp(\boldsymbol{a}_\ell)$ and $\boldsymbol{\Delta}_{\ell,t}$ are positive, the resulting retention factor satisfies $\boldsymbol{R}_{\ell,t} \in (0,1)^{d_m}$. This parameterization separates a learned static decay rate from an input-dependent forgetting signal and guarantees a bounded retention factor.

Under this gated instantiation, the input gate is $\boldsymbol{I}_{\ell,t} = \sigma(\boldsymbol{q}^i_{\ell,t})$. The memory state is then updated by

$$\boldsymbol{c}_{\ell,t} = \boldsymbol{R}_{\ell,t} \odot \boldsymbol{c}_{\ell-1,t} + \boldsymbol{I}_{\ell,t} \odot \tanh(\overline{\boldsymbol{g}}_{\ell,t}). \tag{4}$$

The exposed memory representation is $\boldsymbol{h}_{\ell,t} = \sigma(\boldsymbol{q}^o_{\ell,t}) \odot \tanh(\boldsymbol{c}_{\ell,t})$, and the residual update is $\boldsymbol{m}_{\ell,t} = \Phi^{\text{out}}_\ell(\boldsymbol{h}_{\ell,t})$, where $\boldsymbol{c}_{\ell-1,t}, \boldsymbol{c}_{\ell,t} \in \mathbb{R}^{d_m}$ are the incoming and updated recurrent states, respectively. $\Phi^{\text{out}}_\ell : \mathbb{R}^{d_m} \to \mathbb{R}^d$ maps the exposed state back to the residual width. Unlike the gate and candidate-write projections, Φ^{out}_ℓ is implemented as a standard dense output projection. In this implementation, the input gate controls writing, the retention factor controls preservation, and the output gate controls which part of the recurrent state is exposed to the residual stream.

Moreover, the attention-side and MLP-side state chains are initialized independently: $\boldsymbol{C}^a_0 = \boldsymbol{C}^m_0 = \boldsymbol{0}$. Unrolling Eq. (4) along the model depth gives:

$$\boldsymbol{c}_{\ell,t} = \sum_{j=1}^{\ell} \left(\prod_{k=j+1}^{\ell} \boldsymbol{R}_{k,t} \right) \odot \boldsymbol{I}_{j,t} \odot \tanh\left(\overline{\boldsymbol{g}}_{j,t}\right), \tag{5}$$

where the product is element-wise and an empty product is defined as one. Therefore, the state summarizes writes from all preceding layers through adaptive channel-wise retention, while maintaining a fixed memory width d_m.

The recurrence in Eq. (4) is performed across network layers rather than sequence positions. It therefore introduces no sequential scan over the token dimension during training, i.e., all token states within a layer can be updated in parallel. The only additional dependency along the sequence dimension is the short causal convolution, which is parallelizable during training and requires only a fixed-size buffer during autoregressive decoding. At inference time, each branch maintains one state tensor of size $O(Td_m)$, independent of the number of preceding layers.

3.3 Lightweight and Parallel Co-Design

DRM is designed to be lightweight not only in parameter count and FLOPs, but also in its execution dependency structure. On the algorithmic side, the input-side gate and candidate-write projections, namely $\Phi_\ell^{\mathrm{gate}}$ and Φ_ℓ^{in}, are implemented using a head-wise block-diagonal projection augmented with a grouped low-rank bypass. The block-diagonal path provides inexpensive head-specific transformations, while the low-rank path restores cross-head communication. On the systems side, each memory branch is organized as a parallel residual path, i.e., it consumes the same normalized activation as its corresponding backbone sublayer and synchronizes with the backbone only at the final residual addition. This co-design reduces the intrinsic cost of the memory branch and also makes much of its computation and communication overlapable with the original Transformer sublayers.

The output projection Φ_ℓ^{out} is not factorized and remains a standard dense linear layer, because it maps the exposed memory state back to the residual stream and directly determines the expressive capacity of the memory update.

3.3.1 Head-Wise Block-Diagonal Projection

For the lightweight input-side projections used by $\Phi_\ell^{\mathrm{gate}}$ and Φ_ℓ^{in}, consider a projection from $\mathbb{R}^{d_{\mathrm{in}}}$ to G output groups. The width of each group is d_{out}. A dense implementation contains $Gd_{\mathrm{in}}d_{\mathrm{out}}$ parameters. In our implementation, we divide the input and every output group into H heads: $\boldsymbol{x} = [\boldsymbol{x}_1; \ldots; \boldsymbol{x}_H]$, $\boldsymbol{x}_h \in \mathbb{R}^{\delta_{\mathrm{in}}}$, $\delta_{\mathrm{in}} = d_{\mathrm{in}}/H$. Similarly, each output group is partitioned into slices of width $\delta_{\mathrm{out}} = d_{\mathrm{out}}/H$. For output group g and head h, the block-diagonal path computes

$$\boldsymbol{y}_{g,h}^{\mathrm{hw}} = \boldsymbol{W}_{g,h}^{\mathrm{hw}} \boldsymbol{x}_h, \qquad \boldsymbol{W}_{g,h}^{\mathrm{hw}} \in \mathbb{R}^{\delta_{\mathrm{out}} \times \delta_{\mathrm{in}}}. \tag{6}$$

The complete main-path output is obtained by concatenating the head slices within each output group. This design reduces the parameter count from $Gd_{\mathrm{in}}d_{\mathrm{out}}$ to $\Theta_{\mathrm{hw}} = Gd_{\mathrm{in}}d_{\mathrm{out}}/H$.

Because the effective fan-in of each output channel decreases by a factor of H, we scale the block-diagonal weights by $\sqrt{H}$ relative to the corresponding dense-layer initialization. This preserves the initial output variance under the same input distribution.

3.3.2 Grouped Low-Rank Bypass

For these input-side projections, a purely block-diagonal projection prevents direct information exchange between different heads. To this end, we restore cross-head communication through a grouped low-rank bypass:

$$[\boldsymbol{z}_1; \ldots; \boldsymbol{z}_G] = \boldsymbol{W}^{\downarrow} \boldsymbol{x}, \qquad \boldsymbol{W}^{\downarrow} \in \mathbb{R}^{Gr \times d_{\mathrm{in}}}, \tag{7}$$

$$\overline{\boldsymbol{y}}_g = \boldsymbol{U}_g \boldsymbol{z}_g, \qquad \boldsymbol{U}_g \in \mathbb{R}^{d_{\mathrm{out}} \times r}, \tag{8}$$

$$\boldsymbol{y}_{g,h} = \boldsymbol{y}_{g,h}^{\mathrm{hw}} + \mathrm{slice}_h\left(\overline{\boldsymbol{y}}_g\right). \tag{9}$$

Here, r is the low rank, $\boldsymbol{z}_g \in \mathbb{R}^r$ is the latent representation associated with output group g, and $\mathrm{slice}_h(\cdot)$ extracts the components corresponding to head h.

The dense down projection observes all input heads and compresses their information into one rank-r latent for each output group. Then, the group-specific up projection maps this latent back to the output space. This allows different gates and candidate groups to learn distinct cross-head interactions. We initialize each $\boldsymbol{U}_g$ to zero, such that the low-rank path initially contributes no output.

Table 1: Parameter count of one DRM branch when $d_{\text{in}} = d_{\text{out}} = d_m = d$. Bias terms are omitted.

Component	**Dense**	**DRM**
Three-gate projection	$3d^2$	$3d^2/H + 6rd$
Candidate projection	d^2	$d^2/H + 2rd$
Output projection	d^2	d^2
Candidate convolution	-	Kd
Total	$5d^2$	$d^2 + 4d^2/H + 8rd + Kd$

Consequently, the projection starts as a purely head-wise transformation, while cross-head mixing is introduced gradually during training. The parameter and computational cost of this projection are analyzed in Section 3.4.

3.3.3 Parallel-Residual Execution

The lightweight projection reduces the cost of each DRM branch, while the parallel-residual structure reduces its impact on the critical execution path. For the attention-side pathway, both the attention sublayer and the DRM branch consume the same normalized activation $\widetilde{\boldsymbol{X}}_\ell$. The two outputs are combined only at the residual addition in Eq. (1). Similarly, for the MLP-side pathway, both the MLP sublayer and the DRM branch consume $\widehat{\boldsymbol{U}}_\ell$ and meet only at Eq. (2). Therefore, before the final residual addition, the backbone sublayer and the corresponding memory branch have no producer-consumer dependency.

This parallel-residual execution structure enables communication-computation overlap in distributed LLM pre-training. The training commonly employs context parallelism (CP) and expert parallelism (EP) to reduce per-device activation and parameter memory, but these strategies introduce substantial inter-device communication. Since each DRM branch depends only on the shared normalized input and incoming memory state, rather than the internal computation of its host attention or MLP sublayer, its lightweight gate, candidate, convolution, point-wise gating, and output-projection operations can be scheduled while CP or EP communication is in flight. In particular, attention-side DRM can overlap with communication among context partitions, while MLP-side DRM can overlap with token dispatch and combine in expert-parallel MoE layers. Therefore, DRM can utilize communication-induced idle windows to hide much of its additional computation, reducing the exposed overhead when asynchronous overlap is implemented.

The same co-design is also compatible with tensor parallelism. The head-wise block-diagonal path can be evaluated locally after partitioning memory heads across tensor-parallel ranks. Cross-head communication is restricted to the grouped low-rank bypass, whose communicated latent dimension is Gr per token, much smaller than the full output width Gd_{out} when $r \ll d_{\text{out}}$. Moreover, the low-rank communication can be launched once the local down-projection is available and overlapped with the local head-wise computation. Consequently, DRM combines low parameter cost, bounded recurrent state size, and an execution pattern that is naturally parallel with the backbone.

3.4 Complexity Analysis

We summarize the parameter cost of one DRM branch in Table 1. For clarity, the table considers the common setting $d_{\text{in}} = d_{\text{out}} = d_m = d$ and omits bias terms. A dense implementation of the three gates, the candidate-write projection, and the output projection would require $5d^2$ parameters. In DRM, only the output projection remains dense, while the gate and candidate-write projections use the head-wise low-rank design.

The input-side projection ratio relative to a dense projection is $1/H + r\,(1/d_{\text{in}} + 1/d_{\text{out}})$, where the first term comes from the head-wise block-diagonal path and the second term comes from the grouped low-rank bypass. The computational cost follows the same scaling as the parameter count up to the token and batch dimensions, while the depthwise causal convolution adds only $O(Kd_m)$ operations per token per branch. For example, when $d = 4096$, the memory head dimension is 128 and hence $H = 32$, the low rank is $r = 64$, and the convolution kernel size is $K = 4$, each

Table 2: Final training losses across model scales. "Act. Params." denotes the number of activated parameters in the MoE models.

Act. Params.	Loss	
	Baseline	DRM
26M	2.092	**2.064**
47M	1.944	**1.908**
92M	1.801	**1.777**
199M	1.648	**1.626**
368M	1.551	**1.534**

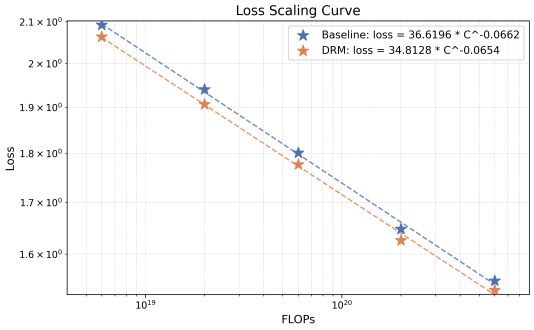


Figure 2: Scaling-law curves.

head-wise low-rank input-side projection contains 6.25% as many parameters as its dense counterpart. The gate and candidate-write projections are reduced from approximately 67.1M parameters to 4.19M parameters, corresponding to a $16\times$ reduction for these input-side projections. Because the output projection remains dense, the projection parameters of one branch are reduced from approximately 83.9M to 21.0M, corresponding to about a $4.0\times$ reduction. The depthwise causal convolution contributes only 0.016M parameters.

4 Experiments

4.1 Experimental Settings

Training Settings. In order to comprehensively evaluate different kinds of residual connections, i.e., standard residual connection (He et al., 2016), mHC (Xie et al., 2026b), AttnRes (Team et al., 2026), and DRM, we incorporate them into MoE models following the Ling-3.0 architecture (inclusionAI, 2026). Moreover, the LLMs adopted for pre-training are of different model sizes. Specifically, we have adopted a series of small-scale LLMs for scaling law experiments. Besides, we have also performed experiments on Ling-3.0-tiny (inclusionAI, 2026) with 7.9B parameters for a large number of GPU hours. For mHC, we replace its parameter-free input normalization with learnable-scale RMSNorm and use a numerically stable log-space Sinkhorn implementation based on LogSumExp. For AttnRes, following (Team et al., 2026), we use blocks of four sublayers (two attention and two MLP). All shared training hyperparameters are identical across methods. Detailed model architecture and training hyperparameters of Ling-3.0-tiny (inclusionAI, 2026) are provided in Appendix B.

Evaluation Benchmarks. Following common LLM evaluation protocols (Team et al., 2025; Team, 2026), we evaluate all models on 21 benchmarks covering three capability categories: world knowledge, language and reasoning, and math and code. The full benchmark list, category assignment, and results are reported in Table 3. Benchmark references are provided in Appendix C.

4.2 Scaling Laws

We conduct scaling-law experiments by training MoE models with either the standard residual connection or the DRM module. The activated parameter counts of the models range from 26M to 368M. After training, we fit the final training loss as a power-law function of training compute, i.e., $\mathcal{L}(C) = aC^b$, where C denotes the number of FLOPs. The detailed model configurations and training hyperparameters are provided in Appendix D.

As shown in Table 2, DRM achieves a lower loss than the baseline at every evaluated scale. Furthermore, Figure 2 presents the fitted loss scaling curves. The two curves exhibit similar scaling exponents, and the curve of DRM remains below the curve of the baseline. These results demonstrate that DRM provides a consistent and scalable improvement over the standard residual connection across different model sizes and compute budgets. Moreover, to separate the architectural benefit of DRM from the effect of its additional parameters, we also construct a parameter-matched standard-residual baseline by enlarging only the hidden size of the LLM. This controlled comparison is reported in Appendix E.

Table 3: Main evaluation results on 21 benchmarks grouped into world knowledge, language and reasoning, and math and code. All methods are pretrained on 1T tokens using Ling-3.0-tiny with 7.9B total parameters. The best result is in bold and the second-best result is underlined.

Types	Benchmarks	Baseline	mHC	AttnRes	DRM
World Knowledge	AGIEval	52.71	**56.95**	<u>55.20</u>	55.04
	MMLU	<u>70.73</u>	**71.08**	70.72	70.67
	MMLU-Pro	39.97	<u>42.28</u>	**42.53**	41.00
	C-Eval	70.84	<u>72.20</u>	**72.93**	72.05
	CMMLU	69.52	**70.59**	<u>70.47</u>	70.16
	SuperGPQA	26.39	<u>26.44</u>	25.29	**26.51**
	ARC-challenge	<u>91.53</u>	**92.54**	85.42	89.49
	Average	60.24	**61.73**	60.37	<u>60.70</u>
Language & Reasoning	BBH	70.95	**72.60**	<u>72.52</u>	72.43
	BBH-ZH	63.28	<u>65.42</u>	65.32	**65.92**
	HellaSwag	68.84	<u>69.91</u>	69.36	**70.78**
	WinoGrande	**69.85**	**69.85**	<u>69.53</u>	<u>69.53</u>
	PIQA	78.18	78.67	**79.00**	<u>78.84</u>
	WorldSense	**56.45**	54.37	54.19	<u>55.72</u>
	Average	67.93	<u>68.47</u>	68.32	**68.87**
Math & Code	HumanEval	71.34	73.78	<u>74.39</u>	**77.44**
	BIRD-SQL	20.40	21.74	<u>23.21</u>	**23.37**
	CRUXEval	49.88	<u>52.31</u>	<u>52.31</u>	**53.81**
	LiveCodeBench	24.84	18.63	**28.27**	<u>26.96</u>
	GSM8K	81.50	**86.43**	84.46	<u>84.69</u>
	MATH	55.02	<u>56.26</u>	**57.56**	53.68
	GSM-ZH	67.20	70.00	**71.60**	<u>71.20</u>
	CMATH	87.70	<u>89.16</u>	87.98	**90.26**
	Average	57.24	58.54	<u>59.97</u>	**60.18**

4.3 Main Results

We compare the standard residual baseline, mHC, AttnRes, and DRM by pretraining Ling-3.0-tiny variants for 1T tokens under the same setup. As shown in Table 3, DRM outperforms the baseline on 16 of 21 benchmarks and achieves the best result on seven. In language and reasoning, it outperforms the baseline on four of six benchmarks and ranks first on BBH-ZH and HellaSwag. In math and code, DRM improves over the baseline on seven of eight benchmarks, ranking first on four and second on three. Its GSM-ZH score increases from 67.20 to 71.20, only 0.40 below the best result. Its world-knowledge gains are more modest, i.e., DRM outperforms the baseline on five of seven benchmarks and ranks first on SuperGPQA.

We have further conducted additional analyses to show that DRM learns a depth-dependent write-retention balance, i.e., lower layers contribute more new information, while deeper layers increasingly reuse the accumulated state. DRM also maintains stable gradient norms during pre-training. Complete visualizations are provided in Appendix F.

4.4 Ablation Studies

We conduct ablation studies for the lightweight projection, state organization, memory branches, gate parameterization, residual scales, and parallel execution. The lightweight-projection, state-organization, branch-contribution, gate-parameterization, and parallel-execution experiments use the 47M-activated-parameter model. The main results are summarized in Table 4 and Table 5, while the residual-scale and input normalization ablations are provided in Appendix G.1 and G.2.

Table 4 validates the main architectural and implementation choices of DRM. The combined head-wise and low-rank design for the gate and candidate-write projections matches the dense input-

Table 4: Ablations of the main DRM components. In the lightweight-projection block, "HW" and "LR" denote the head-wise and grouped low-rank paths applied to the gate and candidate-write projections; the output projection is kept dense in all variants. The default designs are marked by †. Losses are comparable only within each block.

Component	Variant	Configuration	Loss ↓
Lightweight projection			
	Dense projection	Dense	**1.908**
	Low-rank only	LR	1.910
	Head-wise only	HW	1.919
	Default†	HW + LR	**1.908**
State organization and branch contribution			
	Unified state	Attn. + MLP	1.915
	Separate states†	Attn. + MLP	**1.908**
	Attention only	Attn.	1.919
	MLP only	MLP	1.918
	Both branches†	Attn. + MLP	**1.908**
Input/forget-gate parameterization			
	Exp. / Sigmoid	Input / Forget	1.918
	Exp. / Exp.	Input / Forget	1.919
	Sigmoid / Sigmoid	Input / Forget	1.917
	Sigmoid / Exp.†	Input / Forget	**1.908**

side projection variant while outperforming either isolated path, showing that the two paths provide complementary head-local and cross-head transformations. Maintaining separate attention-side and MLP-side states outperforms a unified state, and using both branches is better than retaining either branch alone. Finally, within the gated recurrent transition used in our implementation, the sigmoid input gate combined with exponential forgetting achieves the lowest loss among the four gate configurations.

Parallel-Sequential Formulation Trade-off. We also compare the default parallel-residual execution with a sequential variant under the same model and training configuration. The key difference is that the sequential variant feeds the output of the host backbone sublayer into the corresponding memory branch, which creates a producer-consumer dependency and prevents the memory computation from being decoupled from the backbone computation. As shown in Table 5, the two variants obtain very similar losses, i.e., 1.908 for the parallel design and 1.905 for the sequential design. Although the sequential variant achieves a slight loss advantage of 0.003, we adopt the parallel design as the default under the co-design objective of DRM. This is because it offers a more practical execution pattern and better opportunities for overlapping the memory branch with the corresponding backbone sublayer.

Table 5: Comparison between parallel and sequential memory formulations.

Design	Loss ↓	Difference
Parallel	1.908	+0.003
Sequential	**1.905**	–

5 Conclusion

In this work, we introduce a lightweight depth-recurrent memory module called DRM for enhancing residual information flow in LLMs. DRM maintains separate attention-side and MLP-side states and uses learnable gates to adaptively control memory writing, retention, and exposure, while preserving the original residual pathway. Experiments across model scales show consistent performance improvements, especially in math-and-code tasks and several language-reasoning benchmarks. Ablation studies further validate the effectiveness of the proposed lightweight projections, separate memory pathways, gating scheme, residual scaling, and parallel execution.

AI USE STATEMENT

In this work, we used generative AI tools for providing critical ingredients for proving mathematical claims and implementing methods. We have not used generative AI tools for generating synthetic data sets, helping develop theoretical models or conceptual frameworks, formulating mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, assisting with translation, cleaning and reformatting dataset, supporting qualitative and thematic data analysis, and interpreting results. We have reviewed all AI-assisted work. In particular, all AI-assisted mathematical arguments were independently verified, and all AI-assisted code was reviewed and tested for correctness by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

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

Ankur Bapna, Mia Xu Chen, Orhan Firat, Yuan Cao, and Yonghui Wu. Training deeper neural machine translation models with transparent attention. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 3028–3033, 2018.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xLSTM: Extended long short-term memory. *Advances in Neural Information Processing Systems*, 37: 107547–107603, 2024.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *Advances in Neural Information Processing Systems*, 38:113506–113543, 2025.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 7432–7439, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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

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, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

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

Xeron Du, Yifan Yao, Kaijing Ma, Bingli Wang, Tianyu Zheng, King Zhu, Minghao Liu, Yiming Liang, Xiaolong Jin, Zhenlin Wei, et al. SuperGPQA: Scaling LLM evaluation across 285 graduate disciplines. *Advances in Neural Information Processing Systems*, 38, 2025.

Alex Gu, Baptiste Roziere, Hugh James Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida Wang. CRUXEval: A benchmark for code reasoning, understanding and execution. In *International Conference on Machine Learning*, pp. 16568–16621, 2024.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Ruining He, Anirudh Ravula, Bhargav Kanagal, and Joshua Ainslie. RealFormer: Transformer likes residual attention. In *Findings of the Association for Computational Linguistics*, pp. 929–943, 2021.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021a. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021b.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Jack Hong, Shilin Yan, Jiayin Cai, Xiaolong Jiang, Yao Hu, and Weidi Xie. WorldSense: Evaluating real-world omnimodal understanding for multimodal LLMs. In *International Conference on Learning Representations*, volume 2026, pp. 52423–52443, 2026.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2261–2269, 2017.

Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Yao Fu, et al. C-Eval: A multi-level multi-discipline chinese evaluation suite for foundation models. *Advances in Neural Information Processing Systems*, 36: 62991–63010, 2023.

inclusionAI. Ling-3.0-tiny. https://huggingface.co/inclusionAI/Ling-3.0-tiny, 2026.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Haonan Li, Yixuan Zhang, Fajri Koto, Yifei Yang, Hai Zhao, Yeyun Gong, Nan Duan, and Timothy Baldwin. CMMLU: Measuring massive multitask language understanding in Chinese. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11260–11285, 2024.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can LLM already serve as a database interface? a big bench for large-scale database grounded Text-to-SQLs. *Advances in Neural Information Processing Systems*, 36:42330–42357, 2023.

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.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. Language models are multilingual chain-of-thought reasoners. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=fR3wGCk-IXp.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, 2023.

Kimi Team, Guangyu Chen, Yu Zhang, Jianlin Su, Weixin Xu, Siyuan Pan, Yaoyu Wang, Yucheng Wang, Guanduo Chen, Bohong Yin, et al. Attention residuals. *arXiv preprint arXiv:2603.15031*, 2026.

Ling Team. Ling and ring 2.6 technical report: Efficient and instant agentic intelligence at trillion-parameter scale. *arXiv preprint arXiv:2606.15079*, 2026.

Ling Team, Ang Li, Ben Liu, Binbin Hu, Bing Li, Bingwei Zeng, Borui Ye, Caizhi Tang, Changxin Tian, Chao Huang, et al. Every activation boosted: Scaling general reasoner to 1 trillion open language foundation. *arXiv preprint arXiv:2510.22115*, 2025.

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, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

Tianwen Wei, Jian Luan, Wei Liu, Shuang Dong, and Bin Wang. CMATH: Can your language model pass chinese elementary school math test? *arXiv preprint arXiv:2306.16636*, 2023.

Shufang Xie, Huishuai Zhang, Junliang Guo, Xu Tan, Jiang Bian, Hany Hassan Awadalla, Arul Menezes, Tao Qin, and Rui Yan. Residual: Transformer with dual residual connections. *Artificial Intelligence*, pp. 104602, 2026a.

Zhenda Xie, Yixuan Wei, Huanqi Cao, Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Mingyu Xu, Kuai Yu, Liang Zhao, Shangyan Zhou, Zhean Xu, Zhengyan Zhang, Wangding Zeng, Shengding Hu, Yuqing Wang, Jingyang Yuan, Lean Wang, and Wenfeng Liang. mHC: Manifold-constrained hyper-connections. In *International Conference on Machine Learning*, 2026b. URL https://openreview.net/forum?id=mDhyxu8WRb.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations*, volume 2025, pp. 29687–29707, 2025.

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

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. AGIEval: A human-centric benchmark for evaluating foundation models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 2299–2314, 2024.

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-connections. In *International Conference on Learning Representations*, volume 2025, pp. 97183–97219, 2025.

Lianghui Zhu, Yuxin Fang, Bencheng Liao, Shijie Wang, Tianheng Cheng, Zilong Huang, Chen Chen, Lai Wei, Yutao Zeng, Ya Wang, et al. Mixture-of-depths attention. *arXiv preprint arXiv:2603.15619*, 2026.

APPENDIX

A PYTORCH-STYLE PSEUDOCODE FOR DRM

Listing 1 summarizes the DRM state transition and its integration into a Transformer. Here, `HWLRLinear` denotes the head-wise block-diagonal projection with a grouped low-rank bypass described in Section 3.3. The attention-side and MLP-side branches have independent, layer-specific parameters. For clarity, dropout and distributed implementation details are omitted.

Listing 1: PyTorch-style pseudocode for DRM. Recurrence is performed over model layers, while all tokens are updated in parallel.

```
class DRMBranch(nn.Module):
    def __init__(self, d_model, d_mem, heads, rank, K):
        super().__init__()
        # Lightweight input-side projections
        self.gate = HWLRLinear(d_model, 3 * d_mem,
                               heads, rank)
        self.write = HWLRLinear(d_model, d_mem,
                                heads, rank)
        self.conv = CausalDepthwiseConv1d(d_mem, K)

        # Dense output projection
        self.out = nn.Linear(d_mem, d_model, bias=False)

        # Channel-wise retention parameters
        self.a = nn.Parameter(torch.zeros(d_mem))
        self.b_delta = nn.Parameter(torch.zeros(d_mem))

    def forward(self, z, c_prev):
        # z: [B, T, d_model]; c_prev: [B, T, d_mem]
        q_i, q_f, q_o = self.gate(z).chunk(3, dim=-1)
        g = self.conv(self.write(z))

        delta = F.softplus(q_f + self.b_delta)
        retention = torch.exp(-torch.exp(self.a) * delta)

        c = (retention * c_prev
             + torch.sigmoid(q_i) * torch.tanh(g))
        h = torch.sigmoid(q_o) * torch.tanh(c)
        m = self.out(h)
        return m, c


def drm_transformer_forward(x, layers):
    # Independent attention-side and MLP-side states
    c_a = zeros(B, T, d_mem)
    c_m = zeros(B, T, d_mem)

    for layer in layers:  # recurrence over model depth
        # Attention and DRM-a consume the same normalized input
        z_a = layer.norm_a(x)
        attn_update = layer.attn(z_a)
        m_a, c_a = layer.drm_a(z_a, c_a)
        u = x + attn_update + layer.gamma_a * m_a

        # MLP and DRM-m consume the same normalized input
        z_m = layer.norm_m(u)
        mlp_update = layer.mlp(z_m)
        m_m, c_m = layer.drm_m(z_m, c_m)
        x = u + mlp_update + layer.gamma_m * m_m

    return x
```

Within each pathway, the backbone sublayer and DRM branch consume the same normalized activation and can thus be executed concurrently. Only the fixed-width states `c_a` and `c_m` are propagated across layers, and no history of intermediate-layer representations is stored.

B MODEL ARCHITECTURE AND TRAINING HYPERPARAMETERS

Table 6 summarizes the model architecture and training hyperparameters used for the Ling-3.0-tiny (inclusionAI, 2026) experiments. The table is provided for reproducibility.

Table 6: Model architecture and training hyperparameters of Ling-3.0-tiny (inclusionAI, 2026) used in the main experiments.

Hyperparameter	**Value**
Total Parameters	7.9B
Active Parameters	1.3B
Layers	24
Routed Experts	128
Active Experts	8
Hidden Size	1536
Experts Hidden Size	512
Number of Attention Heads	16
Attention Dimension	128
DRM Memory width d_m	1536
DRM Head Dimension	128
DRM Rank	64
Sequence Length	8192
Vocab Size	157184
Batch Size	8192
Base Learning Rate	0.000339
Wight Decay	0.1

C Evaluation Benchmarks

Table 7 summarizes the evaluation benchmarks used in our main experiments and their corresponding categories.

Table 7: Benchmark categories and references used in the main evaluation.

Category	**Benchmark**	**Reference**
World Knowledge	AGIEval	(Zhong et al., 2024)
	MMLU	(Hendrycks et al., 2021a)
	MMLU-Pro	(Wang et al., 2024)
	C-Eval	(Huang et al., 2023)
	CMMLU	(Li et al., 2024)
	SuperGPQA	(Du et al., 2025)
	ARC-challenge	(Clark et al., 2018)
Language & Reasoning	BBH	(Suzgun et al., 2023)
	BBH-ZH	(Suzgun et al., 2023)
	HellaSwag	(Zellers et al., 2019)
	WinoGrande	(Sakaguchi et al., 2021)
	PIQA	(Bisk et al., 2020)
	WorldSense	(Hong et al., 2026)
Math & Code	HumanEval	(Chen et al., 2021)
	BIRD-SQL	(Li et al., 2023)
	CRUXEval	(Gu et al., 2024)
	LiveCodeBench	(Jain et al., 2025)
	GSM8K	(Cobbe et al., 2021)
	MATH	(Hendrycks et al., 2021b)
	GSM-ZH	(Shi et al., 2023)
	CMATH	(Wei et al., 2023)

D SCALING-LAW EXPERIMENTAL DETAILS

Table 8 reports the model configurations and training hyperparameters used in the scaling-law experiments. The activated parameter count is included as the model identifier so that each configuration can be matched to the corresponding loss reported in Table 2. All models are trained with a sequence length of 4,096.

Table 8: Model configurations and training hyperparameters used in the scaling-law experiments. "Act. Params." denotes the number of activated parameters in the MoE models. All models are trained with a sequence length of 4,096.

Act. Params.	Layers	Hidden Size	Learning Rate	Batch Size
26M	10	512	0.00156	128
47M	12	640	0.00130	192
92M	16	768	0.00110	256
199M	20	1024	0.00091	448
368M	24	1280	0.00077	640

E PARAMETER-MATCHED STANDARD-RESIDUAL BASELINE

DRM introduces additional parameters relative to the standard-residual baseline. Therefore, we conduct a parameter-matched control experiment to examine whether its lower training loss can be explained solely by the increased model capacity. This experiment uses the same 47M-activated-parameter model configuration as the main component ablations. Starting from the standard-residual baseline, we remove all DRM branches and enlarge only the hidden size of the model, such that its total parameter count approximately matches that of the DRM model. All other architectural and training hyperparameters are kept unchanged.

Table 9: Comparison among the original standard-residual baseline, its parameter-matched variant, and DRM.

Variant	Training Loss ↓
Standard residual	1.944
Standard residual, parameter-matched	1.935
DRM	**1.908**

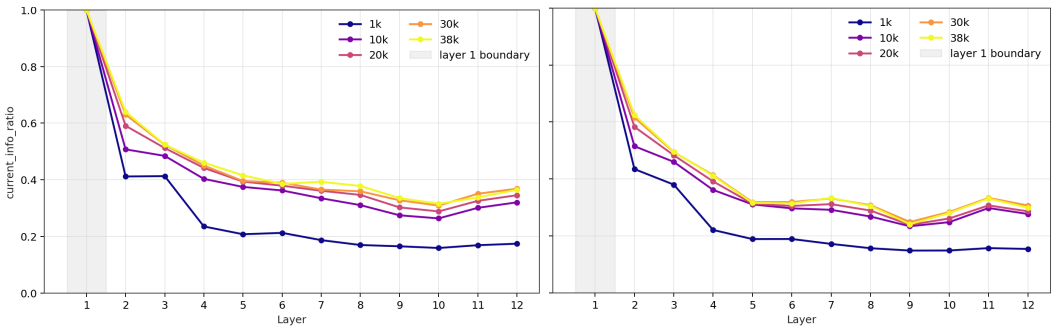
As shown in Table 9, the parameter-matched standard-residual baseline obtains a final training loss of 1.935, whereas DRM obtains 1.908. DRM reduces the loss by 0.027. These results indicate that the improvement of DRM is not explained solely by an increase in total parameter count, and is consistent with the benefit of introducing explicit depth-recurrent memory.

F ADDITIONAL VISUALIZATIONS AND ANALYSIS

F.1 VISUALIZATION FOR THE INFORMATION PROPORTION

In this part, we visualize how the gated transition of DRM balances writing new information and retaining the recurrent state across model depth. The visualizations are conducted based on the 47M-activated-parameter model in our scaling law experiments. For each layer ℓ, we quantify the relative contribution of the current write as $p_\ell = \left\|\boldsymbol{I}_\ell \odot \tanh\left(\overline{\boldsymbol{G}}_\ell\right)\right\|_F / \left(\left\|\boldsymbol{I}_\ell \odot \tanh\left(\overline{\boldsymbol{G}}_\ell\right)\right\|_F + \|\boldsymbol{R}_\ell \odot \boldsymbol{C}_{\ell-1}\|_F\right)$, where $\|\cdot\|_F$ denotes the Frobenius norm over tokens and memory channels. A larger p_ℓ indicates that the updated state relies more heavily on information written by the current layer, whereas a smaller value indicates a larger contribution from the retained historical state. Figure 3 presents the current-information proportion regarding the attention-side and MLP-side branches at different training steps. Please note that

the incoming memory state of the first layer is initialized to zero. Its retention pathway carries no historical information and is marked as a boundary case.



(a) Attention-Side DRM Branch (b) MLP-Side DRM Branch

Figure 3: Current-information proportion across model depth for attention-side and MLP-side DRM branches. The model has 47M activated parameters. The proportion is computed from the norm of the input-gated current write relative to the sum of the norms of the current write and the retained historical state. The first layer is highlighted because its incoming memory state is initialized to zero.

As can be seen from Figure 3, both branches exhibit a sharp decrease in the first few layers, followed by a more gradual decline through the middle layers. Moreover, the proportions generally increase as training proceeds and become similar at 30k and 38k steps, suggesting that the write-retention balance gradually stabilizes. These phenomena are consistent with the recurrent-state dynamics of DRM along model depth. Specifically, lower layers must populate an initially empty state, while deeper layers can reuse information accumulated from multiple preceding layers and thus rely more on retention. Overall, the visualization shows that the two independent DRM branches learn depth-dependent strategies for controlling information flow.

F.2 Training Stability Analysis

To examine whether DRM introduces optimization instability, we compare the step-wise gradient norms of the standard residual baseline, mHC, AttnRes, and DRM under the same pre-training setting. Figure 4 reports the gradient norms from 30k to 60k training steps. All methods exhibit a rapid decrease at the beginning of the displayed interval, because of the introduction of the augmented data. After that, the optimization becomes relatively stable.

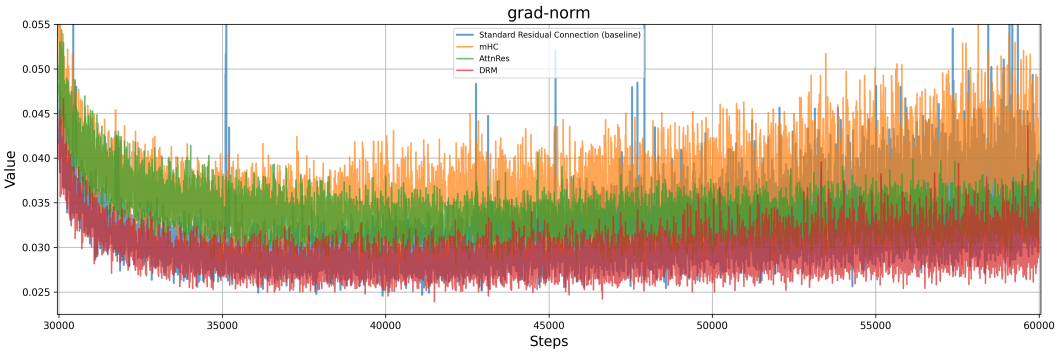


Figure 4: Step-wise gradient norms of the standard residual baseline, mHC, AttnRes, and DRM from 30k to 60k pre-training steps.

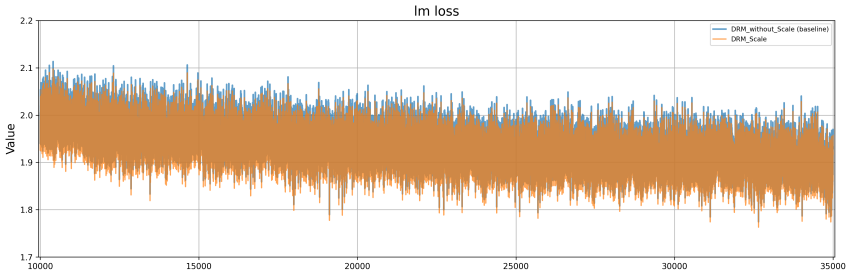
Despite the similar overall trend, the methods differ noticeably in the magnitude and variability of their gradient norms. The baseline maintains a relatively low gradient norm but exhibits several significant spikes. The gradient norms of mHC become increasingly dispersed during the later

stage and reach comparatively high values, while AttnRes shows an intermediate level of variation. In contrast, DRM generally remains within the lowest and most concentrated range throughout the displayed interval, with fewer extreme fluctuations than the other methods. Furthermore, its gradient norm does not show a sustained upward drift as training proceeds. These observations suggest that introducing recurrent memory along model depth does not impair the optimization stability of the backbone model.

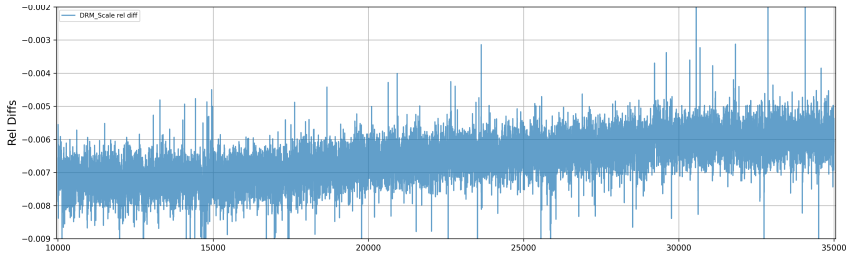
G Additional Ablation Results

G.1 Residual Scale

As shown in Eqs. (1) and (2), DRM introduces learnable residual scales γ_ℓ^a and γ_ℓ^m for the attention-side and MLP-side memory updates, respectively. These scales allow each layer and pathway to calibrate the magnitude of the information exposed by the memory branch before it is added to the residual stream. To evaluate their effect, we compare the full design against an ablation in which the scales are removed. The ablated variant directly adds each memory output to the residual stream, which is equivalent to fixing all residual scales to one. The experimental results are presented in Figure 5.



(a) Training Loss



(b) Loss Difference

Figure 5: Ablation of the learnable residual scales in DRM. (a) Step-wise training loss of DRM with learnable residual scales and the no-scale ablation. (b) Relative loss difference computed as $(\mathcal{L}_{\text{scale}} - \mathcal{L}_{\text{no-scale}})/\mathcal{L}_{\text{no-scale}}$. Negative values indicate that introducing learnable residual scales yields a lower training loss.

Figure 5a shows that the two variants follow similar optimization trajectories, while the model equipped with learnable residual scales consistently achieves a lower training loss. Because the two loss curves largely overlap at the displayed scale, we additionally report their step-wise relative difference, i.e., $(\mathcal{L}_{\text{scale}} - \mathcal{L}_{\text{no-scale}})/\mathcal{L}_{\text{no-scale}}$. As shown in Figure 5b, the relative loss difference remains negative throughout the displayed training interval, with the loss reduction generally ranging from approximately 0.005 to 0.008. These results indicate that directly injecting memory outputs with a fixed unit weight is suboptimal. The learnable residual scales provide an effective mechanism for calibrating the contributions of different memory branches and layers, thereby improving the optimization of DRM.

G.2 Input Normalization

In the default DRM design, each memory branch reuses the normalized input of its corresponding backbone sublayer. Specifically, the attention-side branch takes $\widetilde{\boldsymbol{X}}_\ell = \mathrm{Norm}_\ell^a(\boldsymbol{X}_\ell)$ as input, while the MLP-side branch takes $\widehat{\boldsymbol{U}}_\ell = \mathrm{Norm}_\ell^m(\boldsymbol{U}_\ell)$. Therefore, DRM does not introduce an additional normalization layer, and it directly reuses the normalization already computed by the host backbone sublayer.

To evaluate the importance of this input normalization, we compare the default formulation with a variant in which the DRM branches receive the corresponding unnormalized residual activations. In this variant, the attention-side and MLP-side DRM inputs are changed to $\boldsymbol{X}_\ell$ and $\boldsymbol{U}_\ell$, respectively, while the inputs to the backbone attention and MLP sublayers remain normalized. Therefore, this ablation changes only the inputs to DRM and leaves the backbone computation and all other DRM components unchanged. Both variants are evaluated using the model with 47M activated parameters under the same training configuration.

Table 10: Ablation of input normalization for DRM using the model with 47M activated parameters. The default DRM design reuses the normalized input of each host backbone sublayer.

DRM Input	**Training Loss** $\downarrow$
Unnormalized residual activation	1.920
Normalized host-sublayer input†	**1.908**

As shown in Table 10, removing input normalization increases the training loss from 1.908 to 1.920, corresponding to an absolute degradation of 0.012. This result indicates that normalized activations provide a more suitable input to the depth-recurrent transition. Because the gate logits and candidate writes are computed from these activations at every layer, controlling their input scale helps maintain consistent gate and write distributions along the recurrent depth chain. These results support reusing the normalized host-sublayer inputs in the default DRM design.

H Limitations and Future Work

The limitations and future work are mainly from four aspects. We summarize them as follows:

- Despite the lightweight design in DRM, it still increases the number of activated parameters and FLOPs compared with standard residual connections. Further reducing this overhead through more efficient projections, parameter sharing, or hardware-aware implementations remains an important direction.
- The current DRM implementation uses vector-valued recurrent states and LSTM-style gating as its state-transition mechanism. These are implementation choices rather than restrictions of the general DRM formulation. Matrix-valued memories or alternative recurrent transitions may provide richer state dynamics and stronger expressive capacity, and are worth exploring in future work.
- The information capacity of DRM is closely related to its memory width d_m. Simply increasing d_m becomes costly when scaling to larger models due to the resulting training and activation overhead. Future work should investigate more scalable memory expansion strategies to increase capacity without proportional growth in training cost.
- Our current empirical results show that the gains of DRM on knowledge-intensive benchmarks are less pronounced than those on language reasoning, math, and code tasks. This suggests that the proposed method may be especially effective at improving multi-step transformation, intermediate-state reuse, and structured problem solving, while its effect on factual knowledge acquisition, storage, or recall remains less clear. In future work, we will further investigate the underlying mechanisms behind this discrepancy.