HIERARCHICAL RESIDUALS: A MINIMALIST DECOUPLING OF RESIDUAL WIDTH FROM COMPUTE WIDTH

Anonymous authors
Paper under double-blind review

ABSTRACT

Residual connections underpin Transformers, but they implicitly tie the residual width carried through depth to the compute width used inside each layer. We introduce *Hierarchical Residuals* (HiRes), a minimalist architectural change that decouples these dimensions, widening the residual state while keeping per-layer compute width fixed. HiRes groups layers into blocks: standard d-dimensional PreNorm layers operate inside each block, while a wider D-dimensional Post-Norm outer residual connects blocks. Learned read/write projections bridge the two spaces at block boundaries, leaving individual Transformer layers unchanged. Across different model sizes and depths, HiRes consistently outperforms vanilla residual addition, with fitted scaling laws indicating a $1.30\times$ improvement in FLOPs efficiency. On OLMoE-1B-7B pretraining for 500B tokens, HiRes achieves 0.0208 lower training loss and 1.14-point higher downstream accuracy. HiRes is also competitive with other residual expansion methods such as VWN, mHC and AttnRes with less activation memory and I/O costs. Built from only dense projections and normalizations, it requires neither specialized kernels nor infrastructure-level optimizations. HiRes thus exposes residual width as a practical scaling variable using only standard operators.

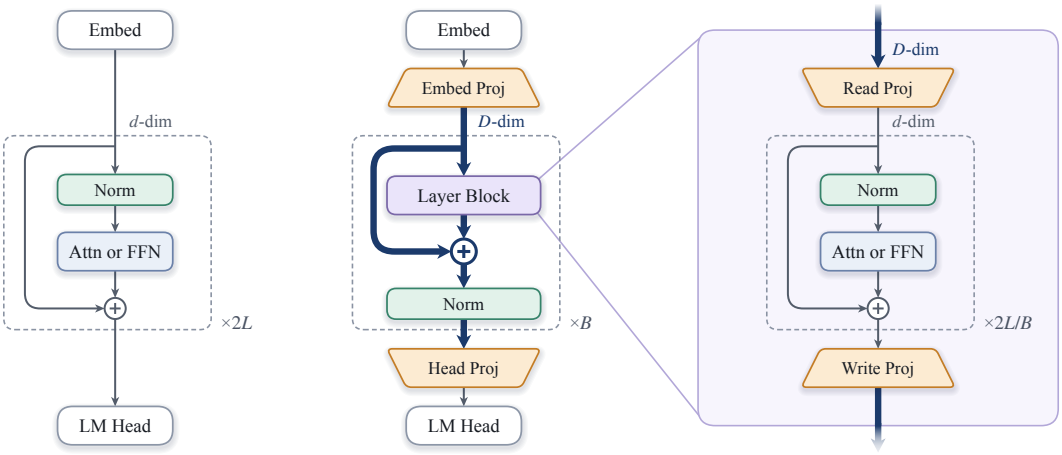
1 INTRODUCTION

Residual connections (He et al., 2016) are the backbone of modern Transformers. A standard PreNorm layer reads from a normalized copy of the current hidden state, computes an update, and adds that back to the original state (Vaswani et al., 2017; Xiong et al., 2020). Repeating this operation yields a simple yet effective transport rule: one d-dimensional state is carried through depth along an identity path, while every attention and feed-forward module contributes an additive update. This design makes deep optimization practical, but it also ties the width of the state carried through depth to the width of the working space of every layer.

Recent residual architectures relax this constraint in different ways. Alternating Updates (AltUp) maintains multiple residual streams, with each layer computing one stream and predicting the remainder (Baykal et al., 2023). Hyper-Connections (HC) likewise uses multiple streams but learns how layers read from, write to, and combine them with lightweight projectors (Zhu et al., 2025a); Virtual Width Networks (VWN) generalizes this formulation by splitting each stream into finer granularity (Seed et al., 2025); Manifold-Constrained Hyper-Connections (mHC) stabilizes HC by constraining its residual mixing operators to a doubly stochastic manifold (Xie et al., 2026). Attention Residuals instead retrieves representations from earlier layers using attention over depth (Kimi Team et al., 2026). These methods introduce different residual operators with different update rules. Nevertheless, they share two recurring departures from vanilla residual addition: (i) the effective residual state is expanded, and (ii) residual accumulation becomes explicitly regulated, keeping the residual scale in check. This suggests that residual-state capacity is an architectural variable in its own right, rather than a quantity that must always equal model width.

However, their per-layer expanded states and intricate mixing add activation traffic and memory-bound residual operators, typically requiring fusion, recomputation, overlap, or specialized kernels to recover throughput. If the benefit stems from the wider state itself rather than particular operators, a natural question arises: *can Transformers benefit from a wider residual state with as little architectural change as possible?*

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.



(a) Vanilla Transformer. (b) HiRes. (c) Inside one layer block.

Figure 1: **HiRes uses a two-level residual hierarchy.** (a) Vanilla PreNorm carries one d-dimensional residual stream. (b) HiRes carries a D-dimensional PostNorm outer residual across layer blocks. (c) Each block reads $D \to d$, runs $2L/B$ vanilla sublayers on the d-dimensional inner residual, and writes back $d \to D$. Gray and navy paths have width d and D; orange trapezoids are the added dense interfaces, and dashed frames mark repeated residual units.

To this end, we introduce *Hierarchical Residuals* (HiRes). HiRes groups adjacent Transformer layers into layer blocks and organizes residual computation at two levels. Inside each block, the model runs ordinary d-dimensional PreNorm Transformer layers. Across blocks, it maintains a D-dimensional outer residual state with the familiar additive update ($D > d$). A read projection maps the outer state into a block, and after the inner residual accumulation, a write projection maps the resulting inner state back. The block write is added to the outer residual and the sum is then post-normalized, which regulates the residual scale. Thus, the outer recurrence retains the familiar PostNorm residual, while the inner Transformer layers are unchanged. In this way, the residual expansion ratio D/d can be selected independently of the compute width d for each inner layer. Because read/write operations occur only at block boundaries, the additional parameters and compute costs are amortized over depth. Figure 1 places the two levels side by side: the outer recurrence is the residual motif of a vanilla PostNorm Transformer, with a whole layer block in place of a sublayer.

Empirically, HiRes consistently outperforms vanilla residual addition across model sizes and depths. Under fitted loss-compute scaling laws, it is $1.30\times$ more FLOP-efficient. As depth scales up at fixed width, HiRes continues to improve through 128 layers where the baseline exhibits diminishing returns. Against Hyper-Connections, mHC, VWN, and Attention Residuals, HiRes remains comparable in quality at lower activation memory and I/O costs. On the pretraining task of OLMoE-1B-7B MoE model for 500B tokens (Section 3.5), HiRes achieves 0.0208 lower training loss and 1.14-point higher downstream accuracy. Overall, HiRes provides a minimalist way to expand the residual state without widening per-layer compute. It shows that decoupling residual width from compute width is beneficial, and that this decoupling is available with standard operators.

2 Method

2.1 Vanilla Residual Addition

Consider a decoder-only Transformer with L layers and hidden width d. Each layer contains an attention module and an FFN module, giving $2L$ residual sublayers. Let $x_{i-1} \in \mathbb{R}^d$ denote the residual state before sublayer i. A standard PreNorm Transformer can be written as

$$x_i = x_{i-1} + F_i(\mathrm{RMSNorm}_i(x_{i-1})), \qquad i = 1, \ldots, 2L, \tag{1}$$

where $F_i : \mathbb{R}^d \to \mathbb{R}^d$ is an attention or FFN transformation. Thus the state carried through depth (residual width, dimension of x_i) and the state processed by every sublayer (compute width, input and output dimension of F_i) share the same width d.

2.2 Decoupling Residual Width and Compute Width

To decouple residual width and compute width, we introduce a wider residual stream $h_i \in \mathbb{R}^D$ with $D \geq d$ and $n := \frac{D}{d}$ denotes the residual expansion ratio. Since every F_i keeps operating at compute width d, there needs a read map $R_i : \mathbb{R}^D \to \mathbb{R}^d$ to project the carried state into the working space, and a write map $W_i : \mathbb{R}^d \to \mathbb{R}^D$ to return the sublayer output to the residual space. So the vanilla residual in Equation (1) becomes

$$x_i = R_i h_{i-1}, \qquad h_i = h_{i-1} + W_i F_i(\mathrm{RMSNorm}_i(x_i)), \qquad i = 1, \ldots, 2L. \tag{2}$$

Setting $D = d$ and $R_i = W_i = I$ recovers the vanilla recurrence.

Residual-expansion methods typically introduce lightweight interfaces by sharing mixing weights across streams or coordinate groups. HiRes instead adopts the simplest choice: dense projections $R_i \in \mathbb{R}^{d \times D}$ and $W_i \in \mathbb{R}^{D \times d}$. At fixed D and d these are the most general linear interfaces, imposing no structural inductive biases like weight sharing, providing full connectivity between the expanded residual state and the working space. Empirically, HiRes achieves competitive performance at a smaller expansion ratio than methods with structured interfaces, while incurring lower activation memory and I/O costs (Section 3.4).

2.3 Hierarchical Residual Recurrence

Applying dense interfaces at every sublayer would add $\Theta(LDd)$ parameters and FLOPs, which is comparable to the $\Theta(Ld^2)$ cost of the Transformer layers themselves and thus unacceptable. HiRes therefore invokes them only at block boundaries, yielding a hierarchical structure.

HiRes partitions the L layers into B consecutive blocks $\mathcal{B}_1, \ldots, \mathcal{B}_B$. Block b contains ℓ_b layers, where $\sum_{b=1}^{B} \ell_b = L$. For simplicity, we use equal-sized contiguous blocks with $\ell_b = L/B$.

Inside each block b, HiRes keeps the inner PreNorm recurrence unchanged as the vanilla residual in Equation (1):

$$x_{b,i} = x_{b,i-1} + F_{b,i}(\mathrm{RMSNorm}_{b,i}(x_{b,i-1})), \qquad i = 1, \ldots, 2\ell_b. \tag{3}$$

We write Φ_b for the composition of these recurrences, so that $x_{b,2\ell_b} = \Phi_b(x_{b,0})$.

Between blocks, HiRes introduces an outer residual and normalizes it in PostNorm style, transforming the layerwise recurrence in Equation (2) into a blockwise one:

$$x_{b,0} = R_b h_{b-1}, \qquad h_b = \mathrm{RMSNorm}_b(h_{b-1} + W_b \Phi_b(x_{b,0})), \qquad b = 1, \ldots, B. \tag{4}$$

Equations (3) and (4) define HiRes' two-level residual hierarchy: a d-dimensional PreNorm inner residual and a D-dimensional PostNorm outer residual, illustrated in Figure 1. Grouping layers into blocks amortizes the dense-interface overhead across depth, reducing it from $\Theta(LDd)$ to $\Theta(BDd)$. Layers within each block thus share the same boundary interfaces and d-dimensional coordinate system: the block pulls a subspace of the outer residual state, transforms it through several layers, and commits the result back.

To connect the embedding and the language modeling head to the outer residual state, we use an embedding write projection $E \in \mathbb{R}^{D \times d}$ and a head read projection $H \in \mathbb{R}^{d \times D}$,

$$h_0 = E\ \mathrm{Embed}(t), \qquad x_{\mathrm{final}} = H\, h_B, \tag{5}$$

where t is the input token. Appendix C details the exact parameter, FLOP, memory, and I/O overheads, and Appendix Algorithm 1 provides PyTorch-style pseudocode for the forward pass.

2.4 Design Choices

Normalization at block boundaries. Boundary normalization provides residual regulation in HiRes: without it, the outer residual stream would be unbounded as it accumulates block outputs.

Keeping PreNorm within blocks and applying PostNorm between them combines PreNorm's optimization stability during the backward pass (Wang et al., 2019; Xiong et al., 2020) with PostNorm's periodic control of the outer residual scale during the forward pass (Kim et al., 2025).

Setting $D = d$ and $R_b = W_b = I$ in Equation (4) and removing the outer residual accumulation yields hierarchical normalization (HiNorm):

$$x_{b+1,0} = \mathrm{RMSNorm}_b(\Phi_b(x_{b,0})), \qquad b = 1, \ldots, B, \tag{6}$$

which inserts one RMSNorm into the residual stream every L/B layers. HiNorm isolates regulation from residual expansion and serves as a control in our experiments.

HiNorm alone beats the baseline and reaches minimum loss at an intermediate $B = 4$ (Appendix Figure 6a). In Appendix Figure 6b, the residual scale grows monotonically in the vanilla baseline and in HiRes without boundary normalization. HC, VWN, and AttnRes (Zhu et al., 2025a; Seed et al., 2025; Kimi Team et al., 2026) regulate the scale implicitly through residual mixing learned at every layer, whereas HiRes regulates it explicitly with boundary normalizations: the scale grows only within each block and is reset at every boundary.

Interface stabilization. We empirically observe that randomly initialized interfaces degrade performance early in training (Appendix A.3.2), which we hypothesize arises from a seesaw dynamic: the interfaces learn each block's working coordinate system while the inner layers simultaneously learn to transform representations within that evolving space. Motivated by prior identity-biased residual gates (Bachlechner et al., 2021; Touvron et al., 2021), we stabilize the interfaces by interpolating between identity and learned mappings using learnable gating scalars:

$$R_b = \big(1 - \sigma(\alpha_b^R)\big)\Pi + \sigma(\alpha_b^R)\widetilde{R}_b, \qquad W_b = \big(1 - \sigma(\alpha_b^W)\big)\Pi^\top + \sigma(\alpha_b^W)\widetilde{W}_b, \tag{7}$$

where $\Pi = \big[I_{d\times d}, \mathbf{0}_{d\times(D-d)}\big] \in \mathbb{R}^{d\times D}$ truncates the outer state to its first d coordinates, $\Pi^\top$ zero-pads back to width D, and σ is the logistic sigmoid. Each interface has an independent learned scalar logit initialized such that $\sigma(\alpha)$ is close to 0 (we set it to 0.1 in the experiments).

This parameterization initializes the interfaces near identity, providing the inner layers with a relatively stable coordinate system early in training. The interfaces can then gradually specialize and make increasing use of the expanded residual state parameterized by $\widetilde{R}_b$ and $\widetilde{W}_b$.

3 Experiments

3.1 Setup

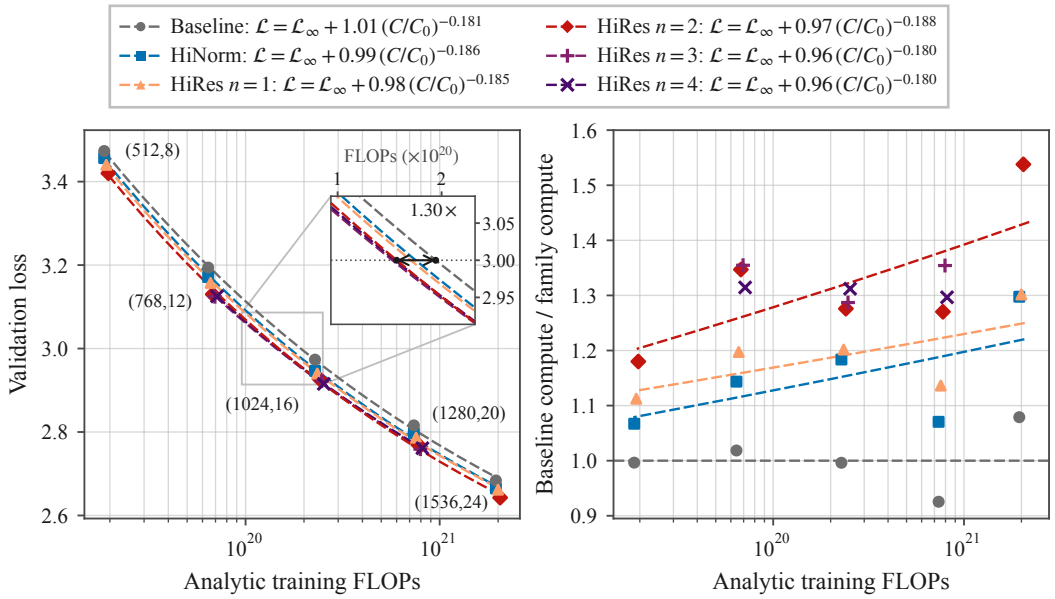
We conduct the primary architecture study using the OLMoE codebase (Muennighoff et al., 2025). All models use RMSNorm (Zhang & Sennrich, 2019), RoPE (Su et al., 2024), grouped-query attention (GQA) (Ainslie et al., 2023), and a SwiGLU feed-forward network (Shazeer, 2020). We train with AdamW (Loshchilov & Hutter, 2019) and a cosine learning-rate schedule with warmup, use a 4096-token context, and initialize weights with standard deviation 0.02.

The default HiRes configuration uses residual expansion ratio $n = D/d = 2$ and block count $B = 4$. We ablate both choices in Section 4.1.

For architecture comparisons, we report training loss averaged over the final 1000 optimization steps. Validation loss and downstream accuracy are averaged over the final 3 evaluations, spaced 1000 steps apart. Validation loss is averaged over 11 held-out domains, and downstream accuracy is averaged over 12 tasks, detailed in Appendix A.

3.2 Scaling on Model Size

We evaluate model-size scaling over five model sizes, jointly increasing inner width and depth from $(d, L) = (512, 8)$ to $(1536, 24)$ with the parameter count spanning from 70M to 626M. The detailed scaling ladder can be found in Table 5. We compare six fixed architecture families: the vanilla residual baseline; HiRes with $n = 2$, $n = 3$, and $n = 4$; and two reduced forms of HiRes, namely HiNorm, which keeps only the boundary normalization, and equal-width HiRes with $n = 1$, which adds the interfaces and the outer residual without widening. All HiRes variants use $B = 4$.



(a) Fitted scaling laws.

(b) Compute multiplier at matched loss.

Figure 2: **HiRes achieves better scaling properties.** (a) Validation loss versus training FLOPs with fitted laws. At $\mathcal{L} = 3.00$ (inset), the baseline needs $1.30\times$ the FLOPs of HiRes with $n = 2$. (b) Compute multiplier at matched loss: baseline FLOPs over family FLOPs, at runs (markers) and along the fitted laws (lines).

Since HiRes introduces modest parameter overhead, we compare models by fitting a loss-compute scaling law (Kaplan et al., 2020; Hoffmann et al., 2022) in analytic training FLOPs: $C = T(6P + 12Lsd)$, where P is the total parameter count, T is the number of training tokens, L is the number of layers, s is the sequence length, and d is the inner width. We jointly fit the observations with the scaling law $\mathcal{L}_f(C) = \mathcal{L}_\infty + A_f(C/C_0)^{-\alpha_f}$, with a shared irreducible loss and family-specific amplitude and exponent. Here $C_0 = 10^{20}$ FLOPs is the reference compute and $\mathcal{L}_\infty = 2.101$ is the fitted irreducible loss.

Figure 2 separates the two sources of the HiRes gain: residual regulation and residual expansion. HiNorm consistently shifts the vanilla residual scaling curve downward. Widening the residual state ($n = 2$) brings a further gain at every scale. Equal-width HiRes ($n = 1$) remains close to HiNorm, indicating that the hierarchy interfaces alone add little benefit. A clearer improvement appears when the outer state expands from $D = d$ to $D = 2d$, while the $n = 3$ and $n = 4$ trajectories remain close to $n = 2$, indicating diminishing returns. Regulation and widening thus contribute complementary gains. HiRes with $n = 2$ combines both, and the fitted scaling laws estimate that the baseline requires $1.30\times$ the FLOPs of HiRes to reach a validation loss of 3.00.

3.3 Scaling on Depth

We next examine depth scaling properties of HiRes by fixing the inner width at $d = 512$ and the training budget at 30B tokens while increasing the number of Transformer layers from 8 to 128.

Figure 3 plots validation loss against depth on a logarithmic axis, where ideal compute scaling should

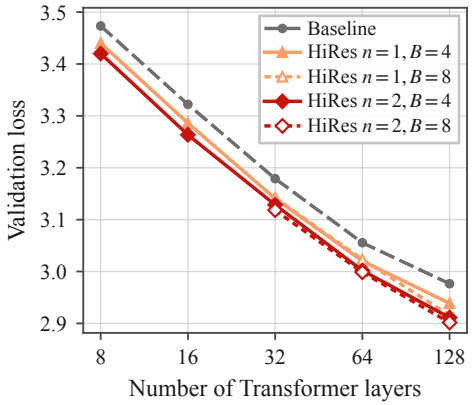


Figure 3: **HiRes scales better with depth.** HiRes wins at every depth and bends less at $L = 128$; $B = 8$ stays nearly straight.

Table 1: **HiRes versus other residual-expansion methods at $(d, L) = (768, 12)$ and 50B tokens.** Suffixes give the expansion rate for HC/mHC/VWN and the block count for Block AttnRes. With a $2\times$ wider residual state, HiRes is competitive even with the $4\times$ variants of HC, mHC, and VWN. **Bold**/underlined: best/second-best among residual-expansion methods. Appendix B details the configurations and lists their parameters and training FLOPs.

Metric	Baseline	HC-2	HC-4	mHC-2	mHC-4	VWN-2	VWN-4	Full AttnRes	Block AttnRes-4	HiRes
Train loss	3.2228	3.1773	3.1870	3.1941	3.1625	3.1584	**3.1578**	3.1938	3.1943	3.1645
Val. loss	3.1941	3.1649	3.1533	3.1871	3.1459	3.1400	**3.1228**	3.1566	3.1649	3.1300
Avg. Acc. (%)	46.91	47.25	46.26	46.66	**47.33**	47.03	45.86	46.17	46.13	47.15

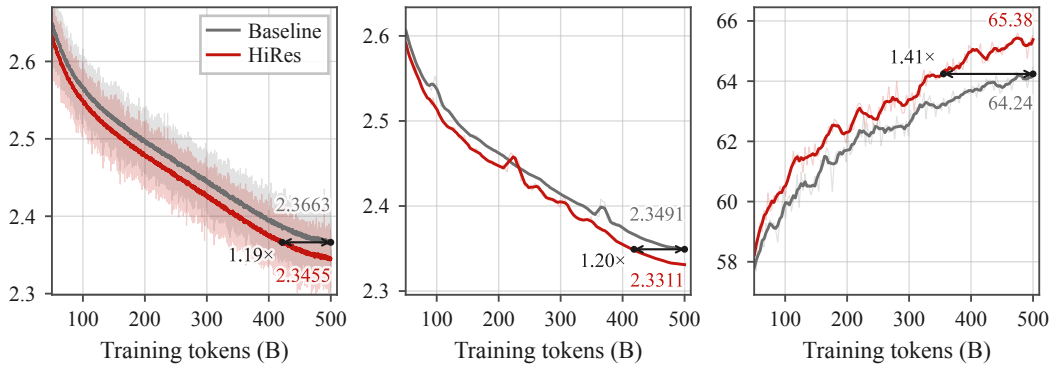
Table 2: **HiRes stays competitive across scales.** Activation I/O is the analytic count per token per sublayer for optimized implementations (Appendix C); peak memory and throughput are measured with each method's optimized Triton implementation (Appendix D), whereas HiRes is nearly as fast with vanilla PyTorch implementation (Appendix Table 13). **Bold**/underlined: best/second-best among residual-expansion methods.

Backbone	Metric	Baseline	mHC-4	VWN-4	Block AttnRes-4	HiRes
$(768, 12)$	Train loss	3.2228	3.1625	**3.1578**	3.1943	3.1645
	Val. loss	3.1941	3.1459	**3.1228**	3.1649	3.1300
	Avg. Acc. (%)	46.91	**47.33**	45.86	46.13	47.15
	Activation I/O	$3d$	$17d$	$17d$	$5.25d$	$\mathbf{4.58d}$
	Peak memory (GiB)	16.31	19.71	20.89	17.18	**16.86**
	Throughput (M tok/s)	0.682	0.590	0.556	**0.639**	0.625
$(1024, 16)$	Train loss	3.0108	**2.9633**	2.9678	2.9912	2.9663
	Val. loss	2.9734	**2.9205**	2.9231	2.9547	2.9271
	Avg. Acc. (%)	48.76	49.16	**49.57**	49.45	49.15
	Activation I/O	$3d$	$17d$	$17d$	$5.19d$	$\mathbf{4.19d}$
	Peak memory (GiB)	21.49	27.28	28.84	22.96	**22.22**
	Throughput (M tok/s)	0.419	0.370	0.354	**0.398**	0.389
$(1280, 20)$	Train loss	2.8524	2.8157	**2.8068**	2.8366	2.8097
	Val. loss	2.8158	2.7671	**2.7533**	2.7871	2.7697
	Avg. Acc. (%)	51.90	52.49	52.30	51.48	**52.76**
	Activation I/O	$3d$	$17d$	$17d$	$5.15d$	$\mathbf{3.95d}$
	Peak memory (GiB)	28.07	36.89	38.85	30.33	**28.99**
	Throughput (M tok/s)	0.279	0.240	0.224	**0.263**	0.259

appear as a straight line. HiRes not only consistently outperforms the vanilla residual baseline at every depth, but also maintains better scaling properties. The baseline bends away from its scaling curve at the last doubling from $L = 64$ to $L = 128$. With $B = 4$, HiRes gives up less of its per-doubling reduction than the baseline, especially at $n = 2$, indicating that regulating residual accumulation and widening the residual state can hold more layers. Raising the block count to $B = 8$ further preserves the scaling close to a straight line through $L = 128$, suggesting more blocks for deeper models.

3.4 Comparison with Other Residual Expansion Methods

We compare HiRes with other residual expansion methods with the recipe of $(d, L) = (768, 12)$ for 50B-token training, namely Hyper-Connections (HC) (Zhu et al., 2025a), mHC (Xie et al., 2026), Virtual Width Networks (VWN) (Seed et al., 2025), and Attention Residuals (Kimi Team et al., 2026). Specifically, we test $2\times$ and $4\times$ wider residual states for HC/mHC/VWN, full and 4-block AttnRes, and $n = 2, B = 4$ HiRes. We then test mHC-4, VWN-4, Block AttnRes-4, and HiRes configurations on larger backbones. Appendix B gives the detailed implementations and complete results.



(a) Training loss. (b) Validation loss. (c) Average accuracy (%).

Figure 4: **HiRes on 500B-token OLMoE-1B-7B pretraining.** HiRes ends with 0.0208 lower training loss, 0.0180 lower validation loss, and a 1.14-point higher average accuracy. See Appendix A.2 for details.

At $(d, L) = (768, 12)$, all residual-expansion methods improve the vanilla residual baseline in loss, and HiRes achieves comparable performance to mHC, VWN, and AttnRes at a lower residual expansion rate, as shown in Table 1.

Table 2 compares the vanilla residual baseline, mHC, VWN, Block AttnRes, and HiRes across three backbone scales: $(d, L) = (768, 12), (1024, 16), (1280, 20)$. HiRes remains within the leading quality range of these recent residual-expansion methods. The same table reports the analytic I/O at each scale (Appendix C), together with throughput and memory from a unified measurement of each method's highly optimized Triton implementation (Appendix D). HiRes has the lowest activation I/O among residual-expansion methods and closest to the baseline's $3d$. Despite additional parameters, HiRes retains 91.7–93.1% of baseline throughput at 3.3–3.4% memory overhead, which is on par with optimized AttnRes and more efficient than mHC/VWN. Moreover, HiRes' standard PyTorch implementation already reaches near-final performance, while other methods rely on mechanism-specific kernel fusion. Appendix D reports the agentic optimization campaign and the latency decomposition behind these numbers.

3.5 Long-Horizon MoE Pretraining

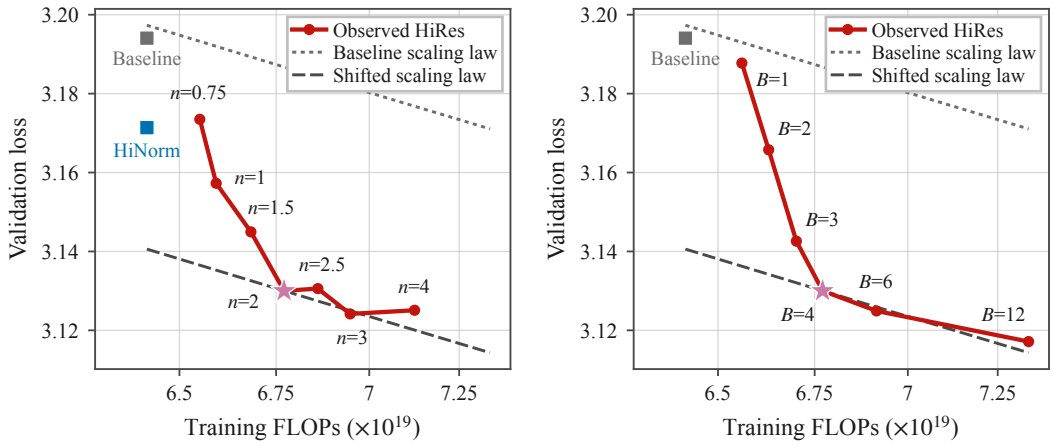
To scale up, we evaluate HiRes on the OLMoE-1B-7B pretraining recipe (Muennighoff et al., 2025), a sparse Mixture-of-Experts model (Shazeer et al., 2017; Fedus et al., 2022). The model contains 6.9B total parameters and 1.3B active parameters per token, and is trained for 500B tokens. We compare the vanilla residual baseline with HiRes using the $n = 2$, $B = 4$ configuration, which adds 83.9M parameters (6.5% of the active parameters and 1.2% of the total parameters); Appendix Table 7 lists the full configuration.

As shown in Figure 4, HiRes achieves 0.0208 lower training loss and 0.0180 lower validation loss than the vanilla residual baseline. It also performs better on most downstream tasks and improves the downstream average accuracy by 1.14 points. These results establish that the HiRes design transfers to long-horizon sparse-MoE pretraining.

4 Ablation Studies

4.1 Ablation on Block Count and Expansion Ratio

We ablate the two structural hyperparameters of HiRes: the residual expansion ratio n and the number of blocks B. All runs use $(d, L) = (768, 12)$ and 50B training tokens. The expansion sweep fixes $B = 4$ and varies $n = D/d$ from 0.75 to 4, while the block sweep fixes $n = 2$ and varies B from 1 to 12. Figure 5 reports both sweeps.



(a) Expansion ratio n at $B = 4$.

(b) Block count B at $n = 2$.

Figure 5: **Ablation on expansion ratio n and block count B in HiRes.** Beyond the default $n = 2$, $B = 4$ (star), gains roughly follow the shifted scaling law (dashed), which translates the baseline scaling law (dotted) through the star, accounting for the benefit from increasing parameters.

Expansion ratio. In Figure 5a, validation loss rapidly decreases as the expansion ratio increases to $n = 2$, after which the slope slows. The wider $n = 3$ and $n = 4$ settings provide marginal gains, largely following the shifted scaling law, which accounts for the benefit from extra parameters. Thus $n = 2$ is the empirical knee of the quality-overhead curve: it captures most of the benefit of residual expansion while keeping the interface cost moderate. On the other side, at $n = 0.75$ the outer state is narrower than the layer compute width, and it is the only point in the sweep that fails to improve over HiNorm, even with more parameters. This result identifies residual width, rather than the interfaces themselves, as the main source of the gain.

Number of blocks. In Figure 5b, increasing B also improves validation loss, with the largest gains occurring as the hierarchy is refined from one to four blocks. Beyond $B = 4$, more frequent outer-state updates show diminishing returns similar to those of the expansion ratio. We therefore retain $B = 4$ as the empirical knee.

4.2 Architecture Ablations

We subtract one component at a time from the complete $n = 2, B = 4$ HiRes model, testing boundary normalization, outer residual, and interface learning against the same reference architecture, as shown in Table 3; HiNorm and equal-width HiRes ($n = 1$) serve as reduced references. All ablations use $(d, L) = (768, 12)$ and 50B training tokens.

The ablations support the following conclusions. (i) The interfaces must be learnable. Freezing the read/write projections raises validation loss by 0.031, so widening residual with fixed maps does not work. (ii) Widening helps even without normalization. Removing boundary RMSNorm raises validation loss by 0.025, yet this widening variant still improves on the baseline by 0.039. (iii) Outer residual is necessary. Removing the outer residual raises validation loss by 0.032, because each write-read pair between adjacent blocks then collapses into a single $d \to D \to d$ MLP with boundary RMSNorm as its activation, leaving the residual state effectively unexpanded. (iv) Interface sta-

Table 3: **Ablations on HiRes components.** Removing or freezing any one component of HiRes ($n = 2, B = 4$) hurts.

Variant	Train loss	Val. loss
Baseline	3.2228	3.1941
HiNorm	3.1956	3.1714
HiRes $n = 1, B = 4$	3.1869	3.1573
HiRes $n = 2, B = 4$	3.1645	3.1300
– Frozen random R/W	3.1905	3.1614
– w/o boundary RMSNorm	3.1874	3.1552
– w/o outer residual	3.1899	3.1616
– w/o interface stabilization	3.1686	3.1375

bilization gives a modest final gain. Disabling it raises validation loss by 0.008, a relatively modest gap. Nevertheless, its advantage is much larger early in training (Appendix A.3.2), which is the main reason we adopt it.

5 Related Work

Residual-state expansion. The closest methods expand the residual state, but typically by replacing vanilla addition with a learned, per-layer operator. AltUp widens the representation into sub-blocks and updates one sub-block at backbone width while a lightweight predictor updates the rest (Baykal et al., 2023). Hyper-Connections (HC) maintains multiple residual streams with learned layer-dependent read, write, and residual mixing weights (Zhu et al., 2025a); Virtual Width Networks (VWN) generalizes HC by constructing fine-grained fractional streams, enabling more expressive mixing (Zhu et al., 2025b; Seed et al., 2025); mHC constrains the residual stream transition to a doubly stochastic manifold for stability (Xie et al., 2026). Attention Residuals instead learns content-dependent attention over earlier layer or block representations (Kimi Team et al., 2026). Relative to this family, HiRes is simpler in mechanism: one wide outer state, dense read/write only at block boundaries, and additive residual transport. It maintains more expressive interfaces between the expanded residual and the fixed-width inner state, and amortizes the cost across depth.

Normalization in Transformers. The original Transformer normalizes after each residual addition (PostNorm), whereas modern language models typically use PreNorm to keep an uninterrupted additive path through depth (Vaswani et al., 2017; Ba et al., 2016; Nguyen & Salazar, 2019; Xiong et al., 2020; Zhang & Sennrich, 2019). There exists a well-known trade-off between PreNorm and PostNorm: PostNorm controls activation scale but can attenuate gradients, while PreNorm eases optimization but lets the residual state grow unbounded (Wang et al., 2024). Hybrid placements pursue both, adding normalization inside the residual branches (Shleifer et al., 2021), around each sublayer (Kim et al., 2025), mixed within a sublayer (Zhuo et al., 2025), across depth (Li et al., 2025), or across coupled PreNorm-like and PostNorm-like streams (Li et al., 2026). HiRes differs in granularity: complete PreNorm layers run inside a block, and PostNorm is applied only after the block-level outer residual addition, preserving ordinary inner updates while periodically resetting the outer-state scale.

Cross-layer connectivity and state aggregation. Prior work enriches information flow across depth through gated shortcuts, identity paths, or dense connectivity (Srivastava et al., 2015; He et al., 2016; Huang et al., 2017). Transformer variants further aggregate earlier representations using static weights, element-wise gates, reused values, or low-rank projections (Pagliardini et al., 2024; Fang et al., 2023; Zhou et al., 2025; Menghani et al., 2025), while MUDDFormer mixes decoupled streams and Deep Delta Learning maintains a matrix-valued state (Xiao et al., 2025; Zhang et al., 2026). HiRes instead uses hierarchical connectivity, combining a persistent wide outer state across blocks with narrow inner states within each block.

6 Conclusion

We introduced Hierarchical Residuals (HiRes), a deliberately minimal architecture that decouples residual-state width from per-layer compute width. HiRes carries a wide persistent state across layer blocks, performs ordinary Transformer computation in a narrower inner space, and connects the two through learned read/write projections at block boundaries. This hierarchical design uses only dense linear maps, normalization, and residual addition, requiring neither specialized kernels nor changes to the inner Transformer layers.

Across model sizes and depths, HiRes consistently outperforms the vanilla residual baseline and is competitive with other residual expansion methods including HC, VWN, mHC, and AttnRes. These results demonstrate that residual-state width need not be tied to per-layer compute width and that their conventional equality can be suboptimal. HiRes therefore establishes residual width as a practical scaling variable that can be exploited through a simple yet effective hierarchical design.

AI USE STATEMENT

In this work, we used generative AI tools to implement systems evaluation, with Claude Opus 5 agents rewriting HiRes and the compared residual mechanisms into optimized PyTorch and Triton implementations for the systems evaluation (Appendix D). We also used generative AI tools to create and edit analysis and plotting code, create and modify figures, to help articulate and refine our interpretation of the results, to revise the analytic cost accounting (Appendix C), and to assist with rewording and polishing the paper for readability. We have not used generative AI tools to generate research ideas, to propose or enhance the method, or to design or run the training experiments. All AI-assisted text, code, figures were reviewed by the authors and revised where needed.

REPRODUCIBILITY STATEMENT

Equations (3)–(5) and (7) fully specify HiRes, and Algorithm 1 gives its forward pass as PyTorch-style pseudocode. Appendix A describes the training recipe, data, tokenizer, and evaluation suites, with scale-dependent hyperparameters in Table 4 and per-run results of the scaling, depth, MoE, and ablation experiments in Appendix A.2. Appendix B gives the configurations of the compared residual methods, Appendix C derives their parameter, FLOP, memory, and I/O costs, and Appendix D describes the throughput and memory measurement protocol. All experiments build on the public OLMoE codebase (Muennighoff et al., 2025).

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.298.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Henry Mao, Gary Cottrell, and Julian McAuley. Rezero is all you need: fast convergence at large depth. In Cassio de Campos and Marloes H. Maathuis (eds.), *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proceedings of Machine Learning Research*, pp. 1352–1361. PMLR, 27–30 Jul 2021.

Cenk Baykal, Dylan Cutler, Nishanth Dikkala, Nikhil Ghosh, Rina Panigrahy, and Xin Wang. Alternating updates for efficient transformers. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 76718–76736. Curran Associates, Inc., 2023. doi: 10.52202/075280-3352.

Yonatan Bisk, Rowan Zellers, Ronan Le bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7432–7439, April 2020. ISSN 2159-5399. doi: 10.1609/aaai.v34i05.6239.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300.

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

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruizhe Pan, Runxin Xu, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Size Zheng, T. Wang, Tian Pei, Tian Yuan, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Liu, Xin Xie, Xingkai Yu, Xinnan Song, Xinyi Zhou, Xinyu Yang, Xuan Lu, Xuecheng Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Zheng, Yichao Zhang, Yiliang Xiong, Yilong Zhao, Ying He, Ying Tang, Yishi Piao, Yixin Dong, Yixuan Tan, Yiyuan Liu, Yongji Wang, Yongqiang Guo, Yuchen Zhu, Yuduan Wang, Yuheng Zou, Yukun Zha, Yunxian Ma, Yuting Yan, Yuxiang You, Yuxuan Liu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhewen Hao, Zhihong Shao, Zhiniu Wen, Zhipeng Xu, Zhongyu Zhang, Zhuoshu Li, Zihan Wang, Zihui Gu, Zilin Li, and Ziwei Xie. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Peter Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin,

Rodolphe Jenatton, Lucas Beyer, Michael Tschannen, Anurag Arnab, Xiao Wang, Carlos Riquelme Ruiz, Matthias Minderer, Joan Puigcerver, Utku Evci, Manoj Kumar, Sjoerd Van Steenkiste, Gamaleldin Fathy Elsayed, Aravindh Mahendran, Fisher Yu, Avital Oliver, Fantine Huot, Jasmijn Bastings, Mark Collier, Alexey A. Gritsenko, Vighnesh Birodkar, Cristina Nader Vasconcelos, Yi Tay, Thomas Mensink, Alexander Kolesnikov, Filip Pavetic, Dustin Tran, Thomas Kipf, Mario Lucic, Xiaohua Zhai, Daniel Keysers, Jeremiah J. Harmsen, and Neil Houlsby. Scaling vision transformers to 22 billion parameters. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 7480–7512. PMLR, 23–29 Jul 2023.

Venmugil Elango, Nidhi Bhatia, Roger Waleffe, Rasoul Shafipour, Tomer Asida, Abhinav Khattar, Nave Assaf, Maximilian Golub, Joey Guman, Tiyasa Mitra, Ritchie Zhao, Ritika Borkar, Ran Zilberstein, Mostofa Patwary, Mohammad Shoeybi, and Bita Rouhani. Latentmoe: Toward optimal accuracy per flop and parameter in mixture of experts, 2026.

Yanwen Fang, Yuxi CAI, Jintai Chen, Jingyu Zhao, Guangjian Tian, and Guodong Li. Cross-layer retrospective retrieving via layer attention. In *The Eleventh International Conference on Learning Representations*, 2023.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020.

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 (CVPR)*, June 2016.

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*, 2021.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Thomas Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karén Simonyan, Erich Elsen, Oriol Vinyals, Jack Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 30016–30030. Curran Associates, Inc., 2022. doi: 10.52202/068431-2176.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020.

Jeonghoon Kim, Byeongchan Lee, Cheonbok Park, Yeontaek Oh, Beomjun Kim, Taehwan Yoo, Seongjin Shin, Dongyoon Han, Jinwoo Shin, and Kang Min Yoo. Peri-LN: Revisiting normalization layer in the transformer architecture. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 30400–30436. PMLR, 13–19 Jul 2025.

Kimi Team, Guangyu Chen, Yu Zhang, Jianlin Su, Weixin Xu, Siyuan Pan, Yaoyu Wang, Yucheng Wang, Guanduo Chen, Bohong Yin, Yutian Chen, Junjie Yan, Ming Wei, Y. Zhang, Fanqing

Meng, Chao Hong, Xiaotong Xie, Shaowei Liu, Enzhe Lu, Yunpeng Tai, Yanru Chen, Xin Men, Haiqing Guo, Y. Charles, Haoyu Lu, Lin Sui, Jinguo Zhu, Zaida Zhou, Weiran He, Weixiao Huang, Xinran Xu, Yuzhi Wang, Guokun Lai, Yulun Du, Yuxin Wu, Zhilin Yang, and Xinyu Zhou. Attention residuals, 2026.

Pengxiang Li, Lu Yin, and Shiwei Liu. Mix-ln: Unleashing the power of deeper layers by combining pre-ln and post-ln. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 98244–98258, 2025.

Tianyu Li, Dongchen Han, Zixuan Cao, Haofeng Huang, Mengyu Zhou, Ming Chen, Erchao Zhao, Xiaoxi Jiang, Guanjun Jiang, and Gao Huang. Siamesenorm: Breaking the barrier to reconciling pre/post-norm, 2026.

Kyle Lo, Lucy Lu Wang, Mark Neumann, Rodney Kinney, and Daniel Weld. S2ORC: The semantic scholar open research corpus. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4969–4983, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.447.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Ian Magnusson, Akshita Bhagia, Valentin Hofmann, Luca Soldaini, Ananya Harsh Jha, Oyvind Tafjord, Dustin Schwenk, Evan Walsh, Yanai Elazar, Kyle Lo, Dirk Groeneveld, Iz Beltagy, Hannaneh Hajishirzi, Noah A. Smith, Kyle Richardson, and Jesse Dodge. Paloma: A benchmark for evaluating language model fit. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 64338–64376. Curran Associates, Inc., 2024. doi: 10.52202/079017-2052.

Gaurav Menghani, Ravi Kumar, and Sanjiv Kumar. LAuReL: Learned augmented residual layer. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 43826–43836. PMLR, 13–19 Jul 2025.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260.

Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax, 2018.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, Noah Smith, Pang Wei Koh, Amanpreet Singh, and Hanna Hajishirzi. Olmoe: Open mixture-of-experts language models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 62061–62121, 2025.

Toan Q. Nguyen and Julian Salazar. Transformers without tears: Improving the normalization of self-attention. In Jan Niehues, Rolando Cattoni, Sebastian Stüker, Matteo Negri, Marco Turchi, Thanh-Le Ha, Elizabeth Salesky, Ramon Sanabria, Loic Barrault, Lucia Specia, and Marcello Federico (eds.), *Proceedings of the 16th International Conference on Spoken Language Translation*, Hong Kong, November 2-3 2019. Association for Computational Linguistics.

Matteo Pagliardini, Amirkeivan Mohtashami, Francois Fleuret, and Martin Jaggi. Denseformer: Enhancing information flow in transformers via depth weighted averaging. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 136479–136508. Curran Associates, Inc., 2024. doi: 10.52202/079017-4336.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Machel Reid, Victor Zhong, Suchin Gururangan, and Luke Zettlemoyer. M2D2: A massively multi-domain language modeling dataset. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 964–975, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.63.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S. Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *Logical Formalizations of Commonsense Reasoning: Papers from the AAAI Spring Symposium*, volume SS-11-06 of *AAAI Technical Report*, pp. 90–95, 2011.

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, August 2021. ISSN 1557-7317. doi: 10.1145/3474381.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQa: Commonsense reasoning about social interactions. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4463–4473, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1454.

Seed, Baisheng Li, Banggu Wu, Bole Ma, Bowen Xiao, Chaoyi Zhang, Cheng Li, Chengyi Wang, Chengyin Xu, Chi Zhang, Chong Hu, Daoguang Zan, Defa Zhu, Dongyu Xu, Du Li, Faming Wu, Fan Xia, Ge Zhang, Guang Shi, Haobin Chen, Hongyu Zhu, Hongzhi Huang, Huan Zhou, Huanzhang Dou, Jianhui Duan, Jianqiao Lu, Jianyu Jiang, Jiayi Xu, Jiecao Chen, Jin Chen, Jin Ma, Jing Su, Jingji Chen, Jun Wang, Jun Yuan, Juncai Liu, Jundong Zhou, Kai Hua, Kai Shen, Kai Xiang, Kaiyuan Chen, Kang Liu, Ke Shen, Liang Xiang, Lin Yan, Lishu Luo, Mengyao Zhang, Ming Ding, Mofan Zhang, Nianning Liang, Peng Li, Penghao Huang, Pengpeng Mu, Qi Huang, Qianli Ma, Qiyang Min, Qiying Yu, Renming Pang, Ru Zhang, Shen Yan, Shen Yan, Shixiong Zhao, Shuaishuai Cao, Shuang Wu, Siyan Chen, Siyu Li, Siyuan Qiao, Tao Sun, Tian Xin, Tiantian Fan, Ting Huang, Ting-Han Fan, Wei Jia, Wenqiang Zhang, Wenxuan Liu, Xiangzhong Wu, Xiaochen Zuo, Xiaoying Jia, Ximing Yang, Xin Liu, Xin Yu, Xingyan Bin, Xintong Hao, Xiongcai Luo, Xujing Li, Xun Zhou, Yanghua Peng, Yangrui Chen, Yi Lin, Yichong Leng, Yinghao Li, Yingshuan Song, Yiyuan Ma, Yong Shan, Yongan Xiang, Yonghui Wu, Yongtao Zhang, Yongzhen Yao, Yu Bao, Yuehang Yang, Yufeng Yuan, Yunshui Li, Yuqiao Xian, Yutao Zeng, Yuxuan Wang, Zehua Hong, Zehua Wang, Zengzhi Wang, Zeyu Yang, Zhengqiang Yin, Zhenyi Lu, Zhexi Zhang, Zhi Chen, Zhi Zhang, Zhiqi Lin, Zihao Huang, Zilin Xu, Ziyun Wei, and Zuo Wang. Virtual width networks, 2025.

Noam Shazeer. Glu variants improve transformer, 2020.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

Sam Shleifer, Jason Weston, and Myle Ott. Normformer: Improved transformer pretraining with extra normalization, 2021.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew Peters, Abhilasha Ravichander, Kyle Richardson, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Evan Walsh, Luke Zettlemoyer, Noah Smith, Hannaneh Hajishirzi, Iz Beltagy, Dirk Groeneveld, Jesse Dodge, and Kyle Lo. Dolma: an open corpus of three trillion tokens for language model pretraining research. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp.

15725–15788, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.840.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks, 2015.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, February 2024. ISSN 0925-2312. doi: 10.1016/j.neucom.2023.127063.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1421.

Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 32–42, October 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Evan Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Allyson Ettinger, Michal Guerquin, David Heineman, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James Validad Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Jake Poznanski, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 OLMo 2 furious (COLM's version). In *Second Conference on Language Modeling*, 2025.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. Deepnet: Scaling transformers to 1,000 layers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(10):6761–6774, October 2024. ISSN 1939-3539. doi: 10.1109/tpami.2024.3386927.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. Learning deep transformer models for machine translation. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 1810–1822, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1176.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In Leon Derczynski, Wei Xu, Alan Ritter, and Tim Baldwin (eds.), *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-4413.

Da Xiao, Qingye Meng, Shengping Li, and Xingyuan Yuan. MUDDFormer: Breaking residual bottlenecks in transformers via multiway dynamic dense connections. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 68440–68458. PMLR, 13–19 Jul 2025.

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 *Forty-third International Conference on Machine Learning*, 2026.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 10524–10533. PMLR, 13–18 Jul 2020.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Yifan Zhang, Yifeng Liu, Mengdi Wang, and Quanquan Gu. Deep delta learning, 2026.

Zhanchao Zhou, Tianyi Wu, Zhiyun Jiang, Fares Obeid, and Zhenzhong Lan. Value residual learning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28341–28356, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1375.

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-connections. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 97183–97219, 2025a.

Defa Zhu, Hongzhi Huang, Jundong Zhou, Zihao Huang, Yutao Zeng, Banggu Wu, Qiyang Min, and Xun Zhou. Frac-connections: Fractional extension of hyper-connections, 2025b.

Zhijian Zhuo, Yutao Zeng, Ya Wang, Sijun Zhang, Jian Yang, Xiaoqing Li, Xun Zhou, and Jinwen Ma. Hybridnorm: Towards stable and efficient transformer training via hybrid normalization. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 25106–25147. Curran Associates, Inc., 2025. doi: 10.52202/085713-0844.

A Experimental Details

Algorithm 1 PyTorch-style pseudocode of the HiRes forward pass.

```
def forward(self, input_ids):
    x = self.embed(input_ids)
    x_outer = self.embed_proj(x)  # up-project d -> D
    for b in range(self.num_blocks):
        # Read at block entry, down-project D -> d
        x_inner = self.read_proj[b](x_outer)
        # Standard PreNorm inner residual
        for i in range(self.num_layers // self.num_blocks):
            x_inner = x_inner + self.attn[b][i](self.attn_norm[b][i](x_inner))
            x_inner = x_inner + self.ffn[b][i](self.ffn_norm[b][i](x_inner))
        # Write at block exit, up-project d -> D
        x_outer = self.outer_norm[b](x_outer + self.write_proj[b](x_inner))
    x = self.lm_head_proj(x_outer)  # down-project D -> d
    logits = self.lm_head(self.final_layernorm(x))
    return logits
```

A.1 Training Setup

All dense experiments use decoder-only Transformers implemented in the OLMoE codebase (Muennighoff et al., 2025). Models use weight-tied input/output embeddings, RMSNorm (Zhang & Sennrich, 2019), RoPE (Su et al., 2024), SwiGLU (Shazeer, 2020) with preactivation width $d_{\text{ff}} = 4d$, grouped-query attention (Ainslie et al., 2023) with head dimension 64 and 4 KV heads, and per-head QK-norm (Dehghani et al., 2023). Training uses AdamW (Loshchilov & Hutter, 2019) with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, weight decay 0.1, and gradient-norm clipping at 1.0. The learning rate uses cosine decay to 0.1 of the peak rate after a linear warmup over 2% of training tokens. Models use a 4,096-token sequence length and initialization standard deviation 0.02. Table 4 lists the scale-dependent settings.

The training corpus is the OLMo2 stage-1 mixture, tokenized with the Dolma2 tokenizer and its 100,352-token vocabulary (Walsh et al., 2025). All compared architectures at a given scale reuse the same data order and global sample-index array.

Validation loss is averaged over 11 held-out domains from Paloma (Magnusson et al., 2024): C4 (Raffel et al., 2020), Dolma Books, Dolma Common Crawl, Dolma Pes2o, Dolma Reddit, Dolma Stack, Dolma Wiki (Soldaini et al., 2024), ICE, M2D2 S2ORC (Reid et al., 2022; Lo et al., 2020), Pile (Gao et al., 2020), and WikiText-103 (Merity et al., 2017). Downstream accuracy is averaged over 12 tasks: MMLU (Hendrycks et al., 2021), ARC-Challenge and ARC-Easy (Clark et al., 2018), BoolQ (Clark et al., 2019), CommonsenseQA (Talmor et al., 2019), COPA (Roemmele et al., 2011), HellaSwag (Zellers et al., 2019), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), SciQ (Welbl et al., 2017), SocialIQA (Sap et al., 2019), and WinoGrande (Sakaguchi et al., 2021).

Table 4: **Scale-dependent training recipes.**

Configuration (d, L)	$(512, 8)$	$(768, 12)$	$(1024, 16)$	$(1280, 20)$	$(1536, 24)$
Attention heads	8	12	16	20	24
KV heads	4	4	4	4	4
FFN width	2,048	3,072	4,096	5,120	6,144
Baseline params	70.264M	138.433M	245.403M	403.757M	626.078M
Global batch size	128	128	256	384	512
Peak learning rate	8.0×10^{-4}	6.5×10^{-4}	5.7×10^{-4}	5.1×10^{-4}	4.6×10^{-4}
Training tokens	30B	50B	100B	200B	350B

Table 5: **At every model size, loss falls from the baseline to HiNorm, HiRes $n = 1$, and HiRes $n = 2$.** Parameters, training FLOPs (Section 3.2, $s = 4096$), and losses at each scale's token budget. HiNorm and HiRes use $B = 4$; dashes denote unexecuted runs.

					HiRes			
(d, L)	Tokens	Metric	Baseline	HiNorm	$n = 1$	$n = 2$	$n = 3$	$n = 4$
$(512, 8)$	30B	Params	70.264M	70.266M	72.888M	75.511M	–	–
		FLOPs (10^{19})	1.8691	1.8691	1.9163	1.9635	–	–
		Train loss	3.4768	3.4617	3.4515	3.4314	–	–
		Val. loss	3.4733	3.4564	3.4401	3.4201	–	–
$(768, 12)$	50B	Params	138.433M	138.436M	144.334M	150.235M	156.136M	162.037M
		FLOPs (10^{19})	6.4186	6.4187	6.5957	6.7727	6.9497	7.1267
		Train loss	3.2228	3.1956	3.1869	3.1645	3.1586	3.1591
		Val. loss	3.1941	3.1714	3.1573	3.1300	3.1242	3.1251
$(1024, 16)$	100B	Params	245.403M	245.407M	255.892M	266.382M	276.872M	287.362M
		FLOPs (10^{19})	22.780	22.780	23.409	24.039	24.669	25.299
		Train loss	3.0108	2.9827	2.9758	2.9663	2.9589	2.9539
		Val. loss	2.9734	2.9466	2.9401	2.9271	2.9219	2.9153
$(1280, 20)$	200B	Params	403.757M	403.762M	420.146M	436.535M	452.924M	469.313M
		FLOPs (10^{19})	73.623	73.623	75.590	77.556	79.522	81.488
		Train loss	2.8524	2.8338	2.8245	2.8097	2.8015	2.8065
		Val. loss	2.8158	2.7972	2.7865	2.7697	2.7590	2.7612
$(1536, 24)$	350B	Params	626.078M	626.084M	649.677M	673.276M	–	–
		FLOPs (10^{19})	194.91	194.91	199.86	204.82	–	–
		Train loss	2.7295	2.7084	2.7019	2.6907	–	–
		Val. loss	2.6838	2.6647	2.6618	2.6427	–	–

A.2 Detailed Experimental Results

Model size scaling. Following the model-size study in Section 3.2, Table 5 provides the complete results at every evaluated scale.

Depth scaling. In Section 3.3 we study depth scaling at fixed width; Table 6 reports the corresponding results and the follow-up sweep over the number of HiRes blocks.

Long-horizon MoE. For the long-horizon experiment in Section 3.5, Table 7 summarizes the MoE configuration and 500B-token results.

Expansion ratio and block count. Complementing Section 4.1, Table 8 reports the expansion-ratio and block-count ablations.

A.3 Other Architecture Ablations

Table 9 extends the architecture ablations in Section 4.2 with interface, stabilization, write-rule, and transport variants.

Boundary RMSNorm and the outer residual each improve over the baseline when introduced separately, and their combination performs best in the identity-interface factorial. Stabilization improves both learned and frozen interfaces, with the larger gain for frozen random maps. Boundary RMSNorm yields a similar improvement with and without stabilization, so its contribution is not contingent on the scalar gates. The increment-only variant replaces the default write $W_b\Phi_b(x_{b,0})$ with $W_b[\Phi_b(x_{b,0}) - x_{b,0}]$, committing only the update accumulated inside the block, so the outer skip is exactly the identity rather than $I + W_bR_b$; its result is close to full-output writing. At equal residual and compute width, both one-sided variants underperform the complete interface, indicating that learned transport in both directions is preferable.

Table 6: **HiRes beats the baseline at every depth, and $B = 8$ helps most at $L = 128$.** All models use $d = 512$ and 30B tokens. HiRes crosses $n \in \{1, 2\}$ with $B \in \{4, 8\}$; dashes denote unexecuted runs. Parameters include tied embeddings.

			HiRes			
(d, L)	Metric	Baseline	$n = 1, B = 4$	$n = 1, B = 8$	$n = 2, B = 4$	$n = 2, B = 8$
$(512, 8)$	Params	70.264M	72.888M	–	75.511M	–
	FLOPs (10^{19})	1.8691	1.9163	–	1.9635	–
	Train loss	3.4768	3.4515	–	3.4314	–
	Val. loss	3.4733	3.4401	–	3.4201	–
$(512, 16)$	Params	89.148M	91.771M	–	94.395M	–
	FLOPs (10^{19})	2.8126	2.8598	–	2.9071	–
	Train loss	3.3334	3.2934	–	3.2774	–
	Val. loss	3.3221	3.2860	–	3.2635	–
$(512, 32)$	Params	126.915M	129.539M	131.638M	132.162M	136.360M
	FLOPs (10^{19})	4.7004	4.7476	4.7854	4.7948	4.8704
	Train loss	3.1949	3.1521	3.1551	3.1421	3.1364
	Val. loss	3.1790	3.1407	3.1421	3.1285	3.1182
$(512, 64)$	Params	202.449M	205.073M	207.172M	207.696M	211.895M
	FLOPs (10^{19})	8.4759	8.5232	8.5609	8.5704	8.6459
	Train loss	3.0695	3.0323	3.0318	3.0185	3.0162
	Val. loss	3.0555	3.0202	3.0232	3.0019	2.9986
$(512, 128)$	Params	353.518M	356.142M	358.241M	358.765M	362.963M
	FLOPs (10^{19})	16.027	16.074	16.112	16.121	16.197
	Train loss	2.9918	2.9536	2.9258	2.9232	2.9141
	Val. loss	2.9765	2.9387	2.9154	2.9111	2.9017

Table 7: **On OLMoE-1B-7B, HiRes improves both losses and 9 of 12 downstream tasks after 500B tokens. (a)** Model and training settings inherited from OLMoE (Muennighoff et al., 2025), plus the HiRes configuration. **(b)** Losses and 12-task downstream accuracy (%); validation loss and downstream accuracies are averaged over the final three evaluations. **Bold**: the better of the two.

(a) Model and training configuration

Setting	Value
Total parameters	6.9B
Active parameters	1.3B
Transformer layers	16
Model width d	2,048
Attention heads	16
Context length	4,096
MoE experts	64
MoE top-k	8
Expert hidden size	2,048
Global batch size	768
Peak learning rate	4.0×10^{-4}
Warmup ratio	2%
Training tokens	500B
HiRes outer width D	4,096
HiRes blocks B	4
HiRes added parameters	83.9M
Interface initialization	$\sigma(\alpha) = 0.1$

(b) Training and downstream results

Metric	Baseline	HiRes ($n = 2, B = 4$)
Train loss	2.3663	**2.3455**
Val. loss	2.3491	**2.3311**
MMLU	41.58	**42.19**
ARC-Challenge	44.70	**49.61**
ARC-Easy	76.43	**78.54**
BoolQ	66.39	**70.09**
CommonsenseQA	**49.88**	49.66
COPA	**87.67**	86.67
HellaSwag	73.19	**74.29**
OpenBookQA	44.33	**44.80**
PIQA	78.65	**79.67**
SciQ	93.13	**93.60**
SocialIQA	47.78	**48.43**
WinoGrande	**67.17**	67.01
Avg. Acc.	64.24	**65.38**

A.3.1 Boundary Normalization

Figure 6 shows the boundary-frequency ablation and the per-layer residual scale discussed in Section 2.4.

Table 8: **Loss improves quickly up to $n = 2$ and $B = 4$ and slowly beyond.** All runs use $(d, L) = (768, 12)$ and 50B tokens. The upper section varies n at $B = 4$; the lower varies B at $n = 2$.

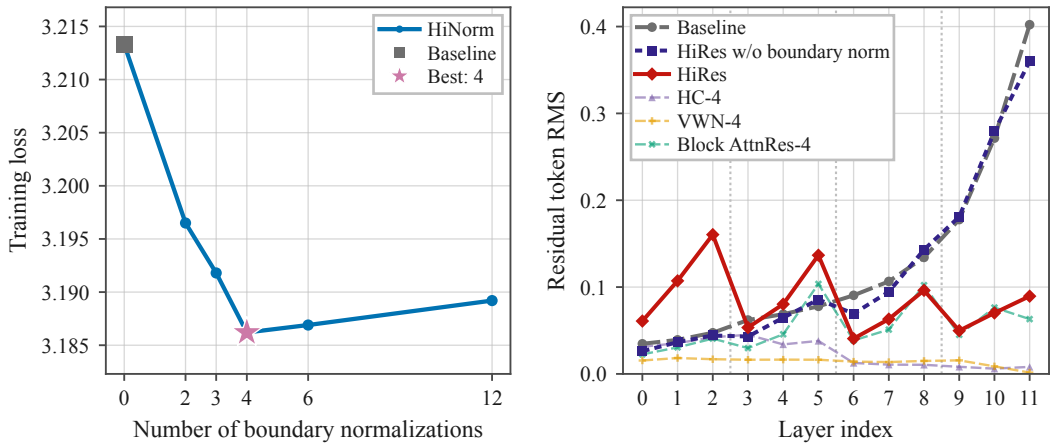
Configuration	Params	Train loss	Val. loss
Expansion ratio at $B = 4$			
$n = 0.75$	142.9M	3.2034	3.1735
$n = 1$	144.3M	3.1869	3.1573
$n = 1.5$	147.3M	3.1738	3.1450
$n = 2$	150.2M	3.1645	3.1300
$n = 2.5$	153.2M	3.1631	3.1306
$n = 3$	156.1M	3.1586	3.1242
$n = 4$	162.0M	3.1591	3.1251
Block count at $n = 2$			
$B = 1$	143.2M	3.2170	3.1877
$B = 2$	145.5M	3.1950	3.1658
$B = 3$	147.9M	3.1747	3.1426
$B = 4$	150.2M	3.1645	3.1300
$B = 6$	155.0M	3.1557	3.1249
$B = 12$	169.1M	3.1540	3.1172

Table 9: **Supplementary interface ablations at $(d, L) = (768, 12)$.** All models train for 50B tokens. Boundary RMSNorm and the outer residual each help and work best together; stabilization helps learned and, more, frozen interfaces; RMSNorm helps with or without stabilization; increment-only writing is close to full-output writing; and both one-sided interfaces trail the complete one.

Variant	Train loss	Val. loss
Boundary normalization × outer residual		
Baseline	3.2228	3.1941
Boundary RMSNorm only (HiNorm)	3.1956	3.1714
Identity R/W + outer residual	3.2118	3.1807
Identity R/W + outer residual + RMSNorm	3.1943	3.1686
Learned interface × scalar stabilization		
Learned R/W, stabilized	3.1645	3.1300
Learned R/W, direct	3.1686	3.1375
Frozen random R/W, stabilized	3.1905	3.1614
Frozen random R/W, direct	3.2181	3.1927
Boundary RMSNorm × scalar stabilization		
With RMSNorm, stabilized	3.1645	3.1300
Without RMSNorm, stabilized	3.1874	3.1552
With RMSNorm, direct	3.1686	3.1375
Without RMSNorm, direct	3.1938	3.1591
Write rule at $n = 2, B = 4$		
Full-output write	3.1645	3.1300
Increment-only write	3.1653	3.1341
Read/write direction at $n = 1, B = 4$		
Complete read/write interface	3.1869	3.1573
Read-only interface	3.1902	3.1630
Write-only interface	3.1897	3.1614

A.3.2 Interface Stabilization Dynamics

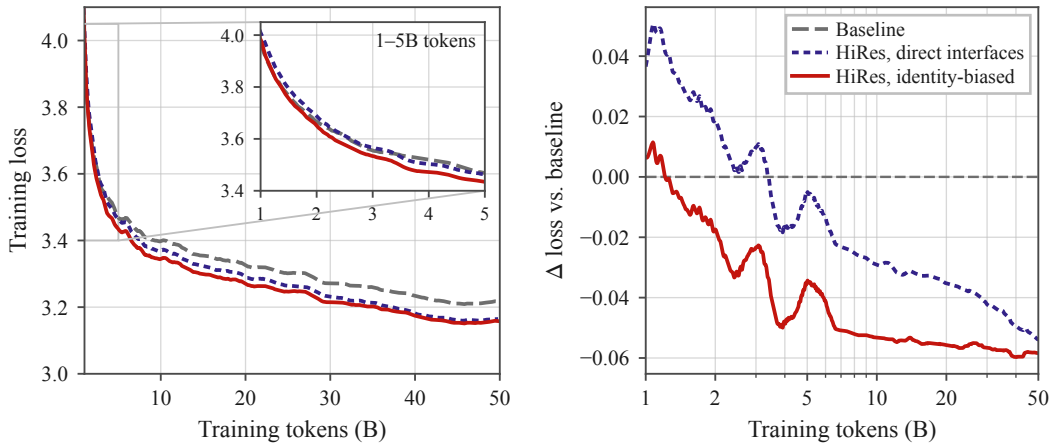
The scalar gates in Equation (7) bias each read/write map toward the coordinate-aligned identity at initialization. We compare the default initialization, $\sigma(\alpha) = 0.1$, with direct dense interfaces. Figure 7 shows that direct interfaces initially train worse than the baseline and only later cross below it. The identity-biased parameterization starts closer to the baseline and remains better throughout the early optimization phase. Stabilization therefore prevents newly initialized interfaces from immedi-



(a) Boundary-frequency ablation.

(b) Per-layer residual activation scale.

Figure 6: **Boundary normalization lowers loss and regulates residual scale at** $(d, L) = (768, 12)$**. (a)** HiNorm training loss under different boundary normalization frequencies. **(b)** Boundary normalization regulates residual accumulation as in other residual-expansion methods; dotted lines mark the HiRes block boundaries ($B = 4$).



(a) Training trajectories and early-token inset.

(b) Loss difference from baseline.

Figure 7: **Identity-biased initialization stabilizes early interface training.** Training loss of the baseline and of HiRes with direct or identity-biased interfaces under one development-scale recipe. **(a)** Full trajectories with a 1–5B-token inset. **(b)** Difference from the baseline; negative favors HiRes. Direct interfaces start worse and cross below the baseline only later, whereas identity-biased ones start near it and stay ahead of direct ones.

ately disrupting residual transport. The final results in Table 9 further show that stabilization is an optimization aid rather than a requirement for the architecture.

B Residual-Method Comparison Details

Configurations. We detail the configurations compared in Section 3.4. HC-m and mHC-m maintain m width-d residual streams. VWN-m has the same total virtual width md but uses finer-grained streams: $2m$ streams of width $d/2$. We evaluate $m \in \{2, 4\}$. Full AttnRes retrieves from all preceding sublayer outputs, whereas Block AttnRes-4 partitions the $2L$ residual sublayers into 4 equal blocks and retains one representation per completed block. HiRes uses expansion ratio

Table 10: **Parameters and training FLOPs of the compared residual methods.** FLOPs, in units of 10^{19}, follow $C = T(6P + 12Lsd)$ from Section 3.2 at each scale's token budget and exclude parameter-free mixing operations. Dashes mark methods evaluated only at $(768, 12)$.

	$(768, 12)$		$(1024, 16)$		$(1280, 20)$	
Method	Params	FLOPs	Params	FLOPs	Params	FLOPs
Baseline	138.4M	6.419	245.4M	22.78	403.8M	73.62
HC-2	138.5M	6.421	–	–	–	–
HC-4	138.6M	6.423	–	–	–	–
mHC-2	138.8M	6.429	–	–	–	–
mHC-4	140.3M	6.474	248.7M	22.98	408.9M	74.24
VWN-2	140.9M	6.492	–	–	–	–
VWN-4	143.3M	6.564	254.0M	23.30	417.2M	75.24
Full AttnRes	138.5M	6.420	–	–	–	–
Block AttnRes-4	138.5M	6.420	245.5M	22.78	403.9M	73.64
HiRes	150.2M	6.773	266.4M	24.04	436.5M	77.56

$n = D/d = 2$ and block count $B = 4$ throughout. Table 10 lists the parameters and training FLOPs of each configuration.

Per-task downstream accuracy. Table 11 reports the complete downstream-task breakdown for the four residual-expansion methods transferred across all three backbone scales.

Across three backbone scales, HiRes improves downstream accuracy over the vanilla baseline by 0.24, 0.39, and 0.86 percentage points from the smallest to the largest model, achieving the highest aggregate accuracy among the compared methods at the largest scale. Individual tasks favor different methods, but HiRes delivers consistent aggregate gains and is competitive with other residual expansion methods.

C Parameters, Compute, Memory, and I/O Costs

Parameters, compute, and memory. HiRes leaves all attention and FFN modules unchanged. Its additional parameters come from $2B$ block interfaces, two endpoint projections, B outer RMSNorm, and $2B + 2$ stabilization scalars:

$$\Delta P = \underbrace{2BdD}_{R_b, W_b} + \underbrace{2dD}_{E, H} + \underbrace{BD}_{\text{Norm}} + \underbrace{(2B + 2)}_{\text{scalar gates}}. \tag{8}$$

For the OLMoE-1B-7B model in Section 3.5, where $(d, L, D, B) = (2048, 16, 4096, 4)$, this amounts to 83.9M additional parameters, approximately 6.5% of the model's 1.3B active parameters and 1.2% of its 6.9B total parameters. The dominant additional training compute comes from the forward and backward passes through the $2(B + 1)$ dense interface projections. Over N_{train} training tokens, this amounts to

$$\Delta C_{\text{train}} \approx 12 N_{\text{train}} (B + 1) dD \quad \text{FLOPs}, \tag{9}$$

excluding the lower-order normalization and scalar-gate operations. Relative to the vanilla residual baseline, HiRes additionally stores only the outer states at the $(B + 1)$ block boundaries. For a micro-batch of N_{mb} tokens, this activation-memory overhead is

$$\Delta A = N_{\text{mb}} (B + 1) D \quad \text{activation elements}. \tag{10}$$

Activation I/O. Let $f_l : \mathbb{R}^d \to \mathbb{R}^d$ denote one sublayer; an L-layer Transformer has $2L$ sublayers. We count forward activation elements per token, excluding parameter reads and f_l's internal traffic. For HiRes, the count includes its two endpoint maps. Following AttnRes (Kimi Team et al., 2026), an input constructed only for f_l is passed to it directly rather than written, whereas residual states reused by later merges are written. Table 12 summarizes the result for each method, and the paragraphs below derive each row.

Table 11: **Tasks favor different methods, but HiRes raises the average over the baseline at every scale.** The gains are 0.24, 0.39, and 0.86 points from the smallest to the largest backbone, where HiRes ranks first. Each method keeps its residual configuration across scales; mHC-4 at $(768, 12)$ uses its available endpoint evaluation. Best and second-best values among residual-expansion methods are **bold** and underlined; the baseline is not ranked. Accuracies are in %.

Backbone	Task	Baseline	mHC-4	VWN-4	Block AttnRes-4	HiRes
$(768, 12)$	MMLU	27.87	27.24	27.28	**27.40**	26.59
	ARC-Challenge	25.98	**24.75**	22.97	23.63	**24.75**
	ARC-Easy	50.94	51.93	50.41	**52.05**	51.11
	BoolQ	59.67	55.84	51.27	**56.06**	55.18
	CommonsenseQA	32.05	34.07	31.59	31.61	**34.34**
	COPA	67.00	68.00	65.00	62.33	**70.67**
	HellaSwag	33.42	34.83	34.83	34.25	**35.26**
	OpenBookQA	30.60	29.40	29.07	**30.53**	29.27
	PIQA	64.62	65.61	65.61	65.52	**65.96**
	SciQ	76.10	**81.50**	78.83	77.30	78.83
	SocialIQA	42.73	42.84	42.39	42.15	**43.09**
	WinoGrande	51.91	**51.93**	51.12	50.70	50.72
	Avg. Acc.	46.91	**47.33**	45.86	46.13	47.15
$(1024, 16)$	MMLU	28.20	28.82	**28.89**	28.52	28.40
	ARC-Challenge	26.09	25.31	24.86	26.20	**26.98**
	ARC-Easy	56.43	54.56	56.43	**56.96**	55.73
	BoolQ	57.28	58.28	59.94	**62.04**	56.38
	CommonsenseQA	35.54	33.58	35.05	34.21	**35.93**
	COPA	69.00	69.00	69.67	**70.67**	66.00
	HellaSwag	40.18	42.02	**42.14**	40.74	42.03
	OpenBookQA	30.60	31.87	**32.00**	30.93	**32.00**
	PIQA	65.92	67.99	67.27	66.47	**68.61**
	SciQ	82.03	**82.73**	82.20	81.60	81.57
	SocialIQA	43.06	43.52	43.50	42.53	**43.72**
	WinoGrande	50.80	52.20	**52.93**	52.49	52.51
	Avg. Acc.	48.76	49.16	**49.57**	49.45	49.15
$(1280, 20)$	MMLU	30.49	30.75	30.89	30.56	**30.95**
	ARC-Challenge	29.32	**29.10**	28.65	28.65	27.65
	ARC-Easy	59.59	58.83	61.29	58.48	**62.81**
	BoolQ	58.92	**60.71**	57.96	58.37	59.71
	CommonsenseQA	37.37	37.37	35.54	38.36	**39.53**
	COPA	73.33	**73.00**	72.00	72.00	72.67
	HellaSwag	47.48	49.59	**50.39**	48.00	50.13
	OpenBookQA	32.20	**35.47**	33.20	31.73	33.07
	PIQA	70.44	70.86	70.78	69.33	**71.13**
	SciQ	85.37	84.67	**87.40**	85.10	85.53
	SocialIQA	43.36	43.36	**44.23**	43.88	44.17
	WinoGrande	54.91	**56.14**	55.25	53.33	55.75
	Avg. Acc.	51.90	52.49	52.30	51.48	**52.76**

Residual and HiRes. A vanilla merge moves $3d$ elements per sublayer, or $6Ld$ in total. HiRes retains this cost for inner merges. At each block boundary, the read projection reads D elements and writes d elements; fusing the write projection, outer addition, and RMSNorm then reads $D + d$ elements and writes D elements. Each block therefore incurs $3D + 2d$ elements of traffic, and the two endpoint maps move another $2(D + d)$, giving the overhead

$$\Delta\mathrm{IO} = B(3D + 2d) + 2(D + d) \quad \text{elements per token} \tag{11}$$

and the total

$$\mathrm{IO}_{\mathrm{HiRes}} = [6L + B(3n + 2) + 2(n + 1)]\, d, \qquad n = D/d. \tag{12}$$

HC, mHC and VWN. HC (Zhu et al., 2025a) maintains m width-d residual streams, $X_l \in \mathbb{R}^{m\times d}$. Its three connections are generated from the current state,

$$\left(H_l^{\mathrm{pre}}, H_l^{\mathrm{post}}, H_l^{\mathrm{res}}\right) = G_l(X_l), \tag{13}$$

Table 12: **HiRes has the lowest activation I/O among residual-expansion methods.** Analytic reads and writes per token for optimized implementations, excluding attention/FFN internals, parameter reads, and lower-order coefficient traffic. m is the HC/mHC/VWN expansion rate, Q the number of AttnRes blocks, and $n = D/d$ and B the HiRes expansion ratio and block count. The fourth column divides total I/O by the $2L$ sublayers; typical values use $L = 16$, $Q = B = 4$, $m = 4$, and $n = 2$.

Method	Total reads	Total writes	Average I/O per sublayer	Typical values
Baseline	$4Ld$	$2Ld$	$3d$	$3d$
HC (fused)†	$2L(3m+1)d$	$2Lmd$	$(4m+1)d$	$17d$
mHC (fused)†	$2L(3m+1)d$	$2Lmd$	$(4m+1)d$	$17d$
VWN (fused)†	$2L(3m+1)d$	$2Lmd$	$(4m+1)d$	$17d$
Full AttnRes (two-phase)	$L\left(Q+\frac{2L}{Q}\right)d$	$2Ld$	$\frac{1}{2}\left(Q+\frac{2L}{Q}+2\right)d$	$7d$
Block AttnRes (two-phase)	$\left[6L+\frac{Q(Q-1)}{2}\right]d$	$4Ld$	$\left[5+\frac{Q(Q-1)}{4L}\right]d$	$5.2d$
HiRes (fused)	$\boldsymbol{[4L+B(2n+1)+n+1]d}$	$\boldsymbol{[2L+B(n+1)+n+1]d}$	$\boldsymbol{\left[3+\frac{B(3n+2)+2(n+1)}{2L}\right]d}$	$\mathbf{4.2}\boldsymbol{d}$

† Leading terms; ignoring items lower-order in d.

with pre-connection $H_l^{\text{pre}} \in \mathbb{R}^{1\times m}$, post-connection $H_l^{\text{post}} \in \mathbb{R}^{m\times 1}$, and a residual connection $H_l^{\text{res}} \in \mathbb{R}^{m\times m}$:

$$x_l = H_l^{\text{pre}} X_l, \qquad y_l = f_l(x_l), \qquad X_{l+1} = H_l^{\text{res}} X_l + H_l^{\text{post}} y_l. \tag{14}$$

Following the dynamic HC parameterization, HC constructs three connections by self-gated projections. Taking H_l^{pre} as an example:

$$H_l^{\text{pre}} = a_l^{\text{pre}} \cdot \tanh(\text{Norm}_l^{\text{pre}}(X_l) W_l^{\text{pre}})^\top + b_l^{\text{pre}}, \tag{15}$$

where $a_l^{\text{pre}} \in \mathbb{R}$ is a scaling factor, $b_l^{\text{pre}} \in \mathbb{R}^{1\times m}$ is a bias, and $W_l^{\text{pre}} \in \mathbb{R}^{d\times 1}$ is the projection weight.

In mHC (Xie et al., 2026), G_l normalizes $\text{vec}(X_l)$, jointly projects it to $m^2 + 2m$ logits, applies sigmoid transforms to the pre/post logits, and applies Sinkhorn–Knopp iterations to H_l^{res}.

In VWN (Seed et al., 2025), each stream is split into r parts of width d/r, forming $X_l \in \mathbb{R}^{mr\times d/r}$. The remaining operations are similar to HC.

The fused schedules below are our analytic constructions from these published recurrences rather than traffic reported by the source papers. For each method, we can pack the three projections in (13) and fold (15) into one kernel, reading md elements once without materializing the normalized and intermediate states. Because H_l^{pre} is available only after the projection has reduced over all of X_l, the subsequent pre-connection in (14) makes a second md read and writes d. After f_l, the residual connection and post-connection are fused into one kernel that reads $md + d$ and writes md. Thus the activation traffic for each sublayer is

$$R = (3m+1)d, \qquad W = md, \qquad \text{IO} = (4m+1)d, \tag{16}$$

and over the network,

$$\text{IO}_{\text{HC}} = \text{IO}_{\text{mHC}} = \text{IO}_{\text{VWN}} = 2L(4m+1)d. \tag{17}$$

This omits the lower-order $m^2 + 2m$ coefficient traffic.

AttnRes. Full AttnRes (Kimi Team et al., 2026) stores every preceding layer output $v_i \in \mathbb{R}^d$ and uses a learned query $q_l \in \mathbb{R}^d$ to retrieve across depth:

$$\alpha_{i\to l} = \frac{\exp\left(q_l^\top \text{RMSNorm}(v_i)\right)}{\sum_{j<l} \exp\left(q_l^\top \text{RMSNorm}(v_j)\right)}, \qquad h_l = \sum_{i<l} \alpha_{i\to l} v_i. \tag{18}$$

Block AttnRes instead stores one completed representation per architecture block and includes the current block's partial sum as an additional source.

For the two-phase schedule, partition the $2L$ sublayers into Q execution blocks of $S = 2L/Q$ sublayers. Phase 1 batches the S fixed queries in a block and attends to completed history once, returning one d-wide output and softmax statistics per sublayer. Phase 2 attends to the sequentially available intra-block history and merges both phases using online softmax (Milakov & Gimelshein, 2018). For Full AttnRes, execution block $q \in \{0, \ldots, Q-1\}$ has qS completed outputs. Phase 1 reads these once for all S queries, contributing qSd elements per block, and writes one d-wide result per sublayer. Within a block, sublayer j reads j earlier intra-block outputs in Phase 2, which also reads back its Phase 1 result for the merge and passes h_l directly to f_l. Summing both triangular histories over depth gives

$$\begin{aligned} R_{\text{FullAttnRes}} &= \left[S \sum_{q=0}^{Q-1} q + Q \sum_{j=0}^{S-1} j + 2L \right] d = L(Q+S)d, \\ W_{\text{FullAttnRes}} &= 2Ld, \\ \overline{\text{IO}}_{\text{FullAttnRes}} &= \frac{Q+S+2}{2} d. \end{aligned} \tag{19}$$

Here the two phases each write one d-wide result per sublayer. Block AttnRes instead reads only the q completed block summaries in Phase 1. Summing this history gives $Q(Q-1)d/2$ reads; its fused partial-sum attention and online-softmax merge additionally read $3d$ and write the d-wide partial sum per sublayer. Thus

$$\begin{aligned} R_{\text{BlockAttnRes}} &= \left[6L + \frac{Q(Q-1)}{2} \right] d, \\ W_{\text{BlockAttnRes}} &= 4Ld, \\ \overline{\text{IO}}_{\text{BlockAttnRes}} &= \left[5 + \frac{Q-1}{2S} \right] d. \end{aligned} \tag{20}$$

D Systems Evaluation of Residual Mechanisms

To assess the practical efficiency of each residual expansion mechanism, we measure both its final training efficiency and the optimization required to reach it, comparing framework-level PyTorch and mechanism-specific Triton implementations.

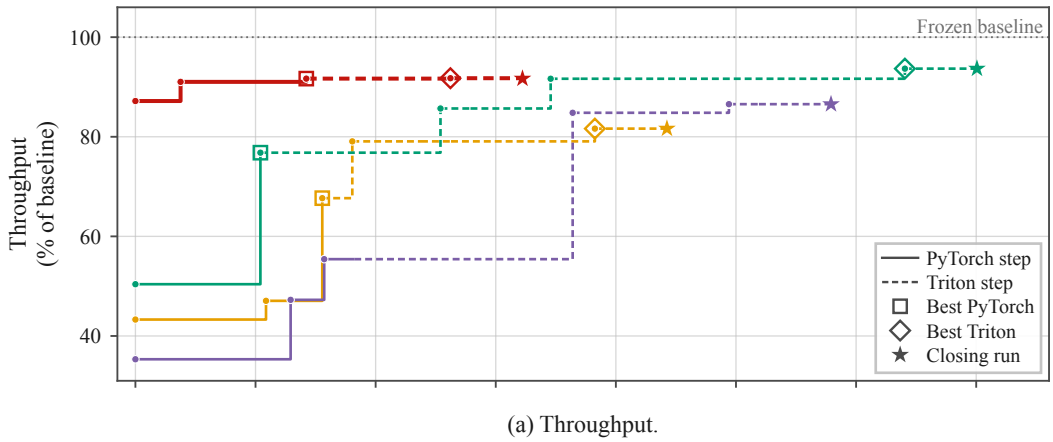
D.1 Experimental Protocol

We evaluate $(d_{\text{model}}, L) \in \{(768, 12), (1024, 16), (1280, 20)\}$ on eight GPUs using FSDP `FULL_SHARD`, bfloat16, sequence length 4096, and per-GPU micro-batch 4. Every measurement includes forward, backward, and the optimizer update; peak memory is reported per GPU. The attention, MLP, normalization, optimizer, precision, and measurement harness are shared, and only mechanism-specific code may change.

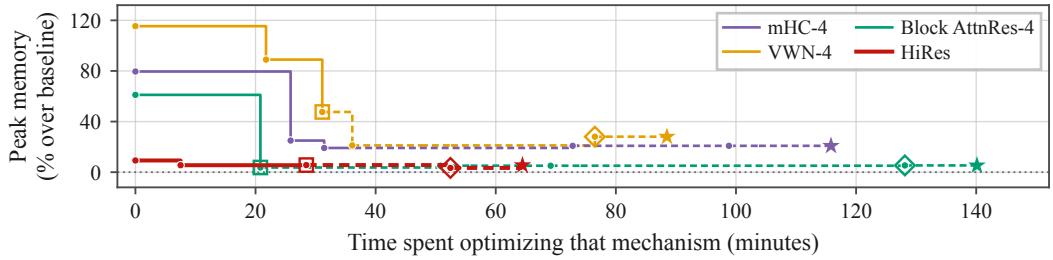
Starting from functionally equivalent naive implementations, independent Claude Opus 5 agents receive the same prompt and optimize HiRes, VWN-4, mHC-4, and Block AttnRes-4 in two stages: the PyTorch stage that may leverage graph transformations, `torch.compile`, batching, and layout changes; and the Triton stage that adds mechanism-specific fused kernels. Every candidate must match the reference outputs and gradients at all three scales. Candidates are measured against the current best and retained only if throughput improves by at least 0.2% at every scale. We report the best implementation from each stage. The final optimized Triton implementation is used to measure the efficiency of each residual expansion mechanism in Table 2.

D.2 Optimization Trajectory

We visualize the optimization campaign in Figure 8. Three patterns emerge. First, HiRes maintains the highest throughput among the residual-expansion methods throughout most of the campaign, until the fully optimized Block AttnRes overtakes it near the end. Second, HiRes reaches its best implementation in less optimization time than the other methods in our campaign. Third, HiRes' PyTorch implementation already matches its final Triton throughput, with custom kernels primarily reducing memory. Other methods instead rely on Triton for substantial throughput gains.



(a) Throughput.



(b) Peak-memory overhead.

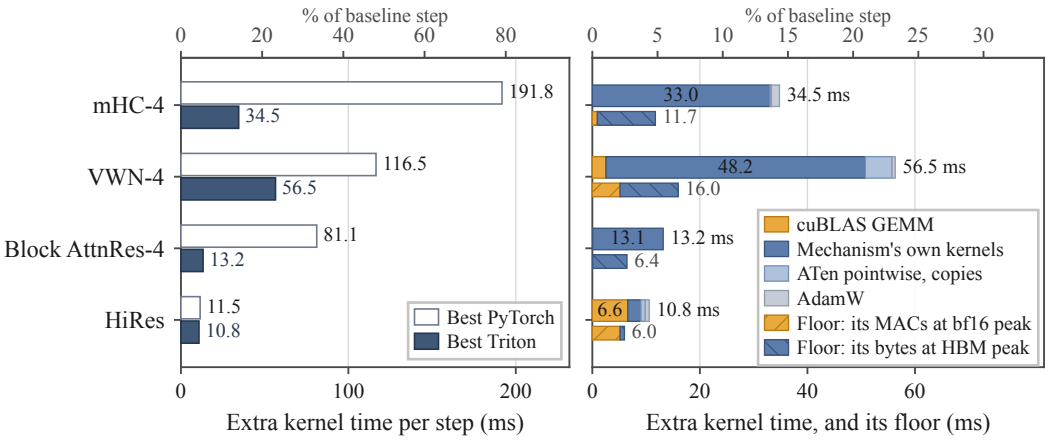
Figure 8: **HiRes reaches its best throughput early and without custom kernels. (a)** Throughput (% of the frozen baseline) and **(b)** peak-memory overhead at $(d, L) = (768, 12)$ versus optimization time. Dots are accepted improvements; solid and dashed segments are the PyTorch and Triton stages; square, diamond, and star mark the best PyTorch, best Triton, and closing implementations. HiRes leads for most of the campaign until fully fused Block AttnRes-4 overtakes it.

Table 13: **HiRes leads in throughput without custom kernels and in memory with them.** Best PyTorch- and Triton-stage implementation of each mechanism; throughput and peak-memory overhead are relative to the frozen baseline. Within each stage, best/second-best values are **bold**/underlined.

		Throughput (% of baseline)			Peak memory (% over baseline)		
Method	Stage	$(768, 12)$	$(1024, 16)$	$(1280, 20)$	$(768, 12)$	$(1024, 16)$	$(1280, 20)$
mHC-4	PyTorch	58.1	56.4	55.2	+19.1	+25.2	+29.7
	Triton	86.5	88.2	86.1	+20.8	+27.0	+31.4
VWN-4	PyTorch	68.3	69.3	69.2	+47.6	+58.7	+66.2
	Triton	81.6	84.4	80.5	+28.1	+34.2	+38.4
Block AttnRes-4	PyTorch	80.3	77.4	75.1	**+3.7**	**+4.9**	+5.8
	Triton	**93.7**	**94.9**	**94.5**	+5.3	+6.9	+8.0
HiRes	PyTorch	**91.7**	**92.7**	**93.0**	+5.5	+5.7	**+5.5**
	Triton	91.7	92.8	93.1	**+3.4**	**+3.4**	**+3.3**

D.3 Practical Efficiency

Table 13 compares the best PyTorch-stage and fused Triton implementations across scales. Without custom kernels, HiRes already achieves 91.7–93.0% of baseline throughput—the highest among the residual-expansion methods—with only 5.5–5.7% peak-memory overhead. Triton fusion preserves this throughput while reducing the memory overhead to 3.3–3.4%. The resulting implementation



(a) What fusion removes. (b) What the fused overhead is, against its floor.

Figure 9: **HiRes overhead is mostly dense GEMMs, so fusion barely changes it.** Extra kernel time per step at $(1280, 20)$ over the shared frozen backbone (242.5 ms). **(a)** Fusion (best Triton versus best PyTorch) cuts the overhead of the other methods by 2–6× but leaves HiRes nearly unchanged. **(b)** Fused overhead by kernel type (upper bars) and its compute- and bandwidth-limited floors at the GPU's advertised bf16 and HBM peaks (lower bars): HiRes spends most of it in cuBLAS GEMMs, the others in their own mixing kernels.

approaches the throughput of Block AttnRes-4, exceeds that of mHC-4 and VWN-4, and has the lowest memory overhead among the compared methods.

D.4 Latency Decomposition

Figure 9 separates measured kernel time from analytic compute- and bandwidth-limited floors. HiRes spends most of its overhead in the $2B + 2$ dense interface GEMMs, whereas the comparison methods spend most of theirs moving and mixing expanded states at every sublayer. This explains why fusion materially benefits the latter, while HiRes is already close to its fused performance with framework-level optimizations.

E A Latent View

Besides expanding the residual state width, HiRes also admits a dual interpretation: relative to a D-dimensional persistent hidden state, its attention and FFN computation occurs in a d-dimensional latent space. Expanding residual width by $n = D/d$ is therefore equivalent to compressing compute width by a factor of n; the wide state preserves information across blocks, while the narrow state performs the transformations within each block.

This view connects HiRes to module-specific latent designs. MLA compresses the attention KV representation (DeepSeek-AI et al., 2024), while LatentMoE performs routed expert computation in a compressed space (Elango et al., 2026). HiRes extends the same separation between state width and compute width to the residual hierarchy. Its key distinction is interaction granularity: instead of exchanging information between hidden and latent spaces at every sublayer, HiRes reads once at block entry and writes once at block exit. Layer grouping therefore reduces hidden-latent transport from per-sublayer to per-block while allowing computation to evolve for multiple layers in the latent space.

F Limitations

Four boundaries of our evidence are worth stating explicitly. First, the structural hyperparameters are tuned once: the sweeps over n and B in Section 4.1 use $(d, L) = (768, 12)$, and the resulting

default $n = 2, B = 4$ is carried to every other scale without re-tuning. The depth study already indicates that this default is not scale-invariant, since $B = 8$ preserves depth scaling better than $B = 4$ at $L = 128$. Second, each configuration is trained once, so we report no seed variance; the small gaps between residual-expansion methods in Table 2 should not be read as reliable orderings. Third, HiRes does not recover baseline throughput. It retains 91.7–93.1% of it, and a fully optimized Block AttnRes is faster, so our efficiency claim is specifically about activation memory and I/O rather than wall-clock speed; the analytic I/O costs we report for the compared methods (Appendix C) are our own constructions from their published recurrences and are not measured. Finally, our results establish that a wider residual state helps, but not why, and we evaluate pretraining loss and few-shot downstream accuracy only, without studying post-training, long-context behavior, or inference-time cost.