ATTENTION RESIDUALS

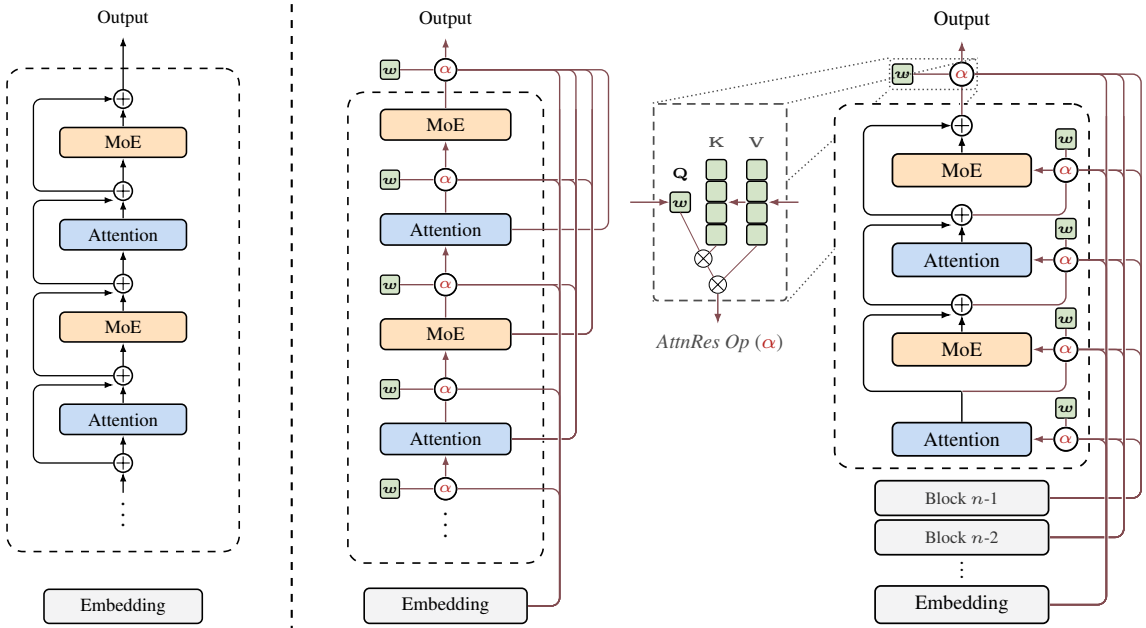
TECHNICAL REPORT OF ATTENTION RESIDUALS

Kimi Team
https://github.com/MoonshotAI/Attention-Residuals

ABSTRACT

Residual connections [12] with PreNorm [60] are standard in modern LLMs, yet they accumulate all layer outputs with fixed unit weights. This uniform aggregation causes uncontrolled hidden-state growth with depth, progressively diluting each layer's contribution [27]. We propose *Attention Residuals (AttnRes)*, which replaces this fixed accumulation with softmax attention over preceding layer outputs, allowing each layer to selectively aggregate earlier representations with learned, input-dependent weights. To address the memory and communication overhead of attending over all preceding layer outputs for large-scale model training, we introduce *Block AttnRes*, which partitions layers into blocks and attends over block-level representations, reducing the memory footprint while preserving most of the gains of full AttnRes. Combined with cache-based pipeline communication and a two-phase computation strategy, Block AttnRes becomes a practical drop-in replacement for standard residual connections with minimal overhead.

Scaling law experiments confirm that the improvement is consistent across model sizes, and ablations validate the benefit of content-dependent depth-wise selection. We further integrate AttnRes into the Kimi Linear architecture [69] (48B total / 3B activated parameters) and pre-train on 1.4T tokens, where AttnRes mitigates PreNorm dilution, yielding more uniform output magnitudes and gradient distribution across depth, and improves downstream performance across all evaluated tasks.



(a) Standard Residuals (b) Full Attention Residuals (c) Block Attention Residuals

Figure 1: Overview of Attention Residuals. **(a)** Standard Residuals: standard residual connections with uniform additive accumulation. **(b)** Full AttnRes: each layer selectively aggregates all previous layer outputs via learned attention weights. **(c)** Block AttnRes: layers are grouped into blocks, reducing memory from $O(Ld)$ to $O(Nd)$.

1 Introduction

Standard residual connections [12] are the *de facto* building block of modern LLMs [35, 51, 9]. The update $\boldsymbol{h}_l = \boldsymbol{h}_{l-1} + f_{l-1}(\boldsymbol{h}_{l-1})$ is widely understood as a *gradient highway* that lets gradients bypass transformations via identity mappings, enabling stable training at depth. Yet residuals also play a second role that has received less attention. Unrolling the recurrence shows that every layer receives the same uniformly-weighted sum of all prior layer outputs; residuals define how information aggregates across depth. Unlike sequence mixing and expert routing, which now employ learnable input-dependent weighting [53, 20, 9], this depth-wise aggregation remains governed by fixed unit weights, with no mechanism to selectively emphasize or suppress individual layer contributions.

In practice, PreNorm [60] has become the dominant paradigm, yet its unweighted accumulation causes hidden-state magnitudes to grow as $O(L)$ with depth, progressively diluting each layer's relative contribution [27]. Early-layer information is buried and cannot be selectively retrieved; empirically, a significant fraction of layers can be pruned with minimal loss [11]. Recent efforts such as scaled residual paths [54] and multi-stream recurrences [72] remain bound to the additive recurrence, while methods that do introduce cross-layer access [36, 56] are difficult to scale. The situation parallels the challenges that recurrent neural networks (RNNs) faced over the sequence dimension before attention mechanism provided an alternative.

We observe a formal duality between depth-wise accumulation and the sequential recurrence in RNNs. Building on this duality, we propose **Attention Residuals (AttnRes)**, which replaces the fixed accumulation $\boldsymbol{h}_l = \sum_i \boldsymbol{v}_i$ with $\boldsymbol{h}_l = \sum_i \alpha_{i\to l} \cdot \boldsymbol{v}_i$, where $\alpha_{i\to l}$ are $\mathrm{softmax}$ attention weights computed from a single learned pseudo-query $\boldsymbol{w}_l \in \mathbb{R}^d$ per layer. This lightweight mechanism enables selective, content-aware retrieval across depth with only one d-dimensional vector per layer. Indeed, standard residual connections and prior recurrence-based variants can all be shown to perform depth-wise *linear* attention; AttnRes generalizes them to depth-wise $\mathrm{softmax}$ attention, completing for depth the same linear-to-$\mathrm{softmax}$ transition that proved transformative over sequences (§6.2, §6.1).

In standard training, Full AttnRes adds negligible overhead, since the layer outputs it requires are already retained for backpropagation. At scale, however, activation recomputation and pipeline parallelism are routinely employed, and these activations must now be explicitly preserved and communicated across pipeline stages. We introduce *Block AttnRes* to maintain efficiency in this regime: layers are partitioned into N blocks, each reduced to a single representation via standard residuals, with cross-block attention applied only over the N block-level summaries. This brings both memory and communication down to $O(Nd)$, and together with infrastructure optimizations (§4), Block AttnRes serves as a drop-in replacement for standard residual connections with marginal training cost and negligible inference latency overhead.

Scaling law experiments confirm that AttnRes consistently outperforms the baseline across compute budgets, with Block AttnRes matching the loss of a baseline trained with $1.25\times$ more compute. We further integrate AttnRes into the Kimi Linear architecture [69] (48B total / 3B activated parameters) and pre-train on 1.4T tokens. Analysis of the resulting training dynamics reveals that AttnRes mitigates PreNorm dilution, with output magnitudes remaining bounded across depth and gradient norms distributing more uniformly across layers. On downstream benchmarks, our final model improves over the baseline across all evaluated tasks.

Contributions

- **Attention Residuals.** We propose AttnRes, which replaces fixed residual accumulation with learned $\mathrm{softmax}$ attention over depth, and its scalable variant Block AttnRes that reduces memory and communication from $O(Ld)$ to $O(Nd)$. Through a unified structured-matrix analysis, we show that standard residuals and prior recurrence-based variants correspond to depth-wise *linear* attention, while AttnRes performs depth-wise $\mathrm{softmax}$ attention.
- **Infrastructure for scale.** We develop system optimizations that make Block AttnRes practical and efficient at scale, including cross-stage caching that eliminates redundant transfers under pipeline parallelism and a two-phase inference strategy that amortizes cross-block attention via online $\mathrm{softmax}$ [31]. The resulting training overhead is marginal, and the inference latency overhead is less than 2% on typical inference workloads.
- **Comprehensive evaluation and analysis.** We validate AttnRes through scaling law experiments, component ablations, and downstream benchmarks on a 48B-parameter model pre-trained on 1.4T tokens, demonstrating consistent improvements over standard residual connections. Training dynamics analysis further reveals that AttnRes mitigates PreNorm dilution, yielding bounded hidden-state magnitudes and more uniform gradient distribution across depth.

2 Motivation

Notation. Consider a batch of input sequences with shape $B \times T \times d$, where B is the batch size, T is the sequence length, and d is the hidden dimension. For clarity, we write formulas for a single token: $\boldsymbol{h}_l \in \mathbb{R}^d$ denotes the hidden state entering layer l, where $l \in \{1, \ldots, L\}$ is the layer index and L is the total number of layers. The token embedding is $\boldsymbol{h}_1$. The function f_l represents the transformation applied by layer l. In Transformer models, we treat each self-attention or MLP as an individual *layer*.

2.1 Training Deep Networks via Residuals

Residual Learning. Residual learning [12] proves to be a critical technique in training deep networks as it allows gradients to bypass transformations. Specifically, each layer updates the hidden state as:

$$\boldsymbol{h}_l = \boldsymbol{h}_{l-1} + f_{l-1}(\boldsymbol{h}_{l-1})$$

Expanding this recurrence, the hidden state at layer l is the sum of the embedding and all preceding layer outputs: $\boldsymbol{h}_l = \boldsymbol{h}_1 + \sum_{i=1}^{l-1} f_i(\boldsymbol{h}_i)$. The key insight behind residual connections is *identity mapping*: each layer preserves a direct path for both information and gradients to flow unchanged. During back-propagation, the gradient with respect to an intermediate hidden state is:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_l} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_L} \cdot \prod_{j=l}^{L-1} \left(\mathbf{I} + \frac{\partial f_j}{\partial \boldsymbol{h}_j} \right)$$

Expanding this product yields $\mathbf{I}$ plus higher-order terms involving the layer Jacobians $\partial f_j / \partial \boldsymbol{h}_j$. The identity term is always preserved, providing a direct gradient path from the loss to any layer regardless of depth.

Generalizing Residuals. While effective, the fixed unit coefficients in the residual update treat every layer's contribution uniformly, offering no mechanism to adapt the mixing across depth. Highway networks [45] relax this by introducing learned element-wise gates:

$$\boldsymbol{h}_l = (1 - \boldsymbol{g}_l) \odot \boldsymbol{h}_{l-1} + \boldsymbol{g}_l \odot f_{l-1}(\boldsymbol{h}_{l-1})$$

where $\boldsymbol{g}_l \in [0, 1]^d$ interpolates between the transformation and the identity path. More generally, both are instances of a weighted recurrence $\boldsymbol{h}_l = \alpha_l \cdot \boldsymbol{h}_{l-1} + \beta_l \cdot f_{l-1}(\boldsymbol{h}_{l-1})$, with residual setting $\alpha_l{=}\beta_l{=}1$ and Highway setting $\alpha_l{=}1{-}\boldsymbol{g}_l$, $\beta_l{=}\boldsymbol{g}_l$.

Limitations. Whether fixed or gated, both approaches share a fundamental constraint: each layer can only access its immediate input $\boldsymbol{h}_{l-1}$, a single compressed state that conflates all earlier layer outputs, rather than the individual outputs themselves. This entails several limitations: (1) *no selective access*: different layer types (e.g., attention vs. MLP) receive the same aggregated state, despite potentially benefiting from different weightings; (2) *irreversible loss*: information lost through aggregation cannot be selectively recovered in deeper layers; and (3) *output growth*: later layers learn increasingly larger outputs to gain influence over the accumulated residual, which can destabilize training. These limitations motivate a mechanism that lets each layer selectively aggregate information from all preceding layers.

3 Attention Residuals: A Unified View of Time and Depth

The limitations discussed above are reminiscent of similar bottlenecks in sequence modeling, suggesting that we seek similar solutions for the depth dimension.

The Duality of Time and Depth. Like RNNs over time, residual connections compress all prior information into a single state $\boldsymbol{h}_l$ over depth. For sequence modeling, the Transformer improved upon RNNs by replacing recurrence with attention [3, 52], allowing each position to selectively access all previous positions with data-dependent weights. We propose the same methodology for depth:

$$\boldsymbol{h}_l = \alpha_{0 \to l} \cdot \boldsymbol{h}_1 + \sum_{i=1}^{l-1} \alpha_{i \to l} \cdot f_i(\boldsymbol{h}_i) \tag{1}$$

where $\alpha_{i \to l}$ are layer-specific attention weights satisfying $\sum_{i=0}^{l-1} \alpha_{i \to l} = 1$. Unlike sequence length (which can reach millions of tokens), network depth is typically modest ($L < 1000$), making $O(L^2)$ attention over depth computationally feasible. We call this approach *Attention Residuals*, abbreviated as *AttnRes*.

3.1 Full Attention Residuals

The attention weights can be written as $\alpha_{i\to l} = \phi(\boldsymbol{q}_l, \boldsymbol{k}_i)$ for a kernel function $\phi\colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_{\geq 0}$, where $\boldsymbol{q}_l$ and $\boldsymbol{k}_i$ are query and key vectors [23, 70]. Different choices of ϕ recover different residual variants (§6.2); we adopt $\phi(\boldsymbol{q}, \boldsymbol{k}) = \exp\left(\boldsymbol{q}^\top \mathrm{RMSNorm}(\boldsymbol{k})\right)$ [66] with normalization, yielding $\mathrm{softmax}$ attention over depth:

$$\alpha_{i\to l} = \frac{\phi\left(\boldsymbol{q}_l, \boldsymbol{k}_i\right)}{\sum_{j=0}^{l-1} \phi\left(\boldsymbol{q}_l, \boldsymbol{k}_j\right)} \tag{2}$$

For each layer l, we define:

$$\boldsymbol{q}_l = \boldsymbol{w}_l, \qquad \boldsymbol{k}_i = \boldsymbol{v}_i = \begin{cases} \boldsymbol{h}_1 & i = 0 \\ f_i(\boldsymbol{h}_i) & 1 \leq i \leq l-1 \end{cases} \tag{3}$$

where the query $\boldsymbol{q}_l = \boldsymbol{w}_l$ is a layer-specific learnable vector in $\mathbb{R}^d$. The $\mathrm{RMSNorm}$ inside ϕ prevents layers with large-magnitude outputs from dominating the attention weights. The input to layer l is then:

$$\boldsymbol{h}_l = \sum_{i=0}^{l-1} \alpha_{i\to l} \cdot \boldsymbol{v}_i \tag{4}$$

We call this form *full attention residuals*. For each token, Full AttnRes requires $O(L^2 d)$ arithmetic and $O(Ld)$ memory to store layer outputs. Since depth is far smaller than sequence length, the arithmetic cost is modest.

Overhead. The $O(Ld)$ memory overlaps entirely with the activations already retained for backpropagation, so Full AttnRes introduces no additional memory overhead in vanilla training. At scale, however, activation recomputation and pipeline parallelism are widely adopted: layer outputs that would otherwise be freed and recomputed must now be kept alive for all subsequent layers, and under pipeline parallelism each must further be transmitted across stage boundaries. Both the memory and communication overhead then grow as $O(Ld)$.

Blockwise optimization. A deliberate design choice in Full AttnRes is that the *pseudo*-query $\boldsymbol{w}_l$ is a learned parameter decoupled from the layer's forward computation. This independence means that attention weights for any group of layers can be computed in parallel without waiting for their sequential outputs, and in particular permits grouping the L layers into N blocks of S layers each and batching the attention computation within each block, reducing per-layer memory I/O from $O(Ld)$ to $O((S+N)d)$ (we defer the detailed two-phase strategy to §4). Under current distributed training regimes, however, the dominant cost is not local memory bandwidth but cross-stage communication under pipeline parallelism: every layer output must still be transmitted between stages, and this $O(Ld)$ communication overhead cannot be alleviated by local batching. This motivates the Block AttnRes variant introduced below, which reduces the number of cross-stage representations from L to N. We anticipate that future interconnect improvements will make the full $O(Ld)$ communication practical, fully realizing the potential of Full AttnRes.

3.2 Block Attention Residuals

We propose *Block Attention Residuals*, which partitions the L layers into N blocks: within each block, the layer outputs are reduced to a single representation via summation, and across blocks, we apply full attention over only N block-level representations and the token embedding. This reduces both memory and communication overhead from $O(Ld)$ to $O(Nd)$.

Intra-Block Accumulation. Specifically, we divide the L layers into N blocks of $S = L/N$ layers each, assuming L is divisible by N; otherwise, the last block contains the remaining $L \bmod N$ layers. Let $\mathcal{B}_n$ denote the set of layer indices in block n ($n = 1, \ldots, N$). To form a block, we sum all of its layer outputs:

$$\boldsymbol{b}_n = \sum_{j \in \mathcal{B}_n} f_j(\boldsymbol{h}_j) \tag{5}$$

We further denote $\boldsymbol{b}_n^i$ as the partial sum over the first i layers in $\mathcal{B}_n$, so that $\boldsymbol{b}_n = \boldsymbol{b}_n^S$. When L is not divisible by N, the final partial sum is taken as the last block's representation. As in Full AttnRes, the $\mathrm{RMSNorm}$ inside ϕ prevents magnitude differences between complete blocks and partial sums from biasing the attention weights.

```python
def block_attn_res(blocks: list[Tensor], partial_block: Tensor, proj: Linear, norm: RMSNorm) -> Tensor:
    """
    Inter-block attention: attend over block reps + partial sum.
    blocks:
        N tensors of shape [B, T, D]: completed block representations for each previous block
    partial_block:
        [B, T, D]:    intra-block partial sum (b_n^i)
    """
    V = torch.stack(blocks + [partial_block])  # [N+1, B, T, D]
    K = norm(V)
    logits = torch.einsum('d, n b t d -> n b t', proj.weight.squeeze(), K)
    h = torch.einsum('n b t, n b t d -> b t d', logits.softmax(0), V)
    return h

def forward(self, blocks: list[Tensor], hidden_states: Tensor) -> tuple[list[Tensor], Tensor]:
    partial_block = hidden_states
    # apply block attnres before attn
    # blocks already include token embedding
    h = block_attn_res(blocks, partial_block, self.attn_res_proj, self.attn_res_norm)

    # if reaches block boundary, start new block
    # block_size counts ATTN + MLP; each transformer layer has 2
    if self.layer_number % (self.block_size // 2) == 0:
        blocks.append(partial_block)
        partial_block = None

    # self-attention layer
    attn_out = self.attn(self.attn_norm(h))
    partial_block = partial_block + attn_out if partial_block is not None else attn_out

    # apply block attnres before MLP
    h = block_attn_res(blocks, partial_block, self.mlp_res_proj, self.mlp_res_norm)

    # MLP layer
    mlp_out = self.mlp(self.mlp_norm(h))
    partial_block = partial_block + mlp_out

    return blocks, partial_block
```

Figure 2: PyTorch-style pseudo code for Block Attention Residuals. `block_attn_res` computes softmax attention over block representations using a learned pseudo-query $\boldsymbol{w}_l$; `forward` is a single-layer pass that maintains `partial_block` ($\boldsymbol{b}_n^i$, intra-block residual) and `blocks` ($[\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1}]$, inter-block history).

Inter-Block Attention. In Full AttnRes, the input to layer l is computed by attending over all outputs up to $f_{l-1}(\boldsymbol{h}_{l-1})$. The block-wise variant replaces these individual outputs with block representations, defining $\boldsymbol{b}_0 = \boldsymbol{h}_1$ so that the token embedding is always included as a source. For the i-th layer in block n, the value matrix is:

$$\mathbf{V} = \begin{cases} [\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}]^\top & \text{if } i = 1 \text{ (first layer of block } n) \\ [\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}, \boldsymbol{b}_n^{i-1}]^\top & \text{if } i \geq 2 \text{ (subsequent layers)} \end{cases} \tag{6}$$

Keys and attention weights follow Eq. 3 and Eq. 2. The input of the very first layer of the network is the token embeddings, i.e. $\boldsymbol{b}_0 = \boldsymbol{h}_1$. In each block, the first layer receives the previous block representations and the token embeddings, and the subsequent layers additionally attend to the partial sum $\boldsymbol{b}_n^{i-1}$. The final output layer aggregates all N block representations. Fig. 2 provides PyTorch-style pseudocode for Block AttnRes.

Efficiency. Since each layer now attends over N block representations rather than L individual outputs, memory reduces from $O(L)$ to $O(N)$ and computation from $O(L^2)$ to $O(N^2)$. The block count N interpolates between two extremes: $N = L$ recovers Full AttnRes, while $N = 1$ reduces to standard residual connections with the embedding isolated as $\boldsymbol{b}_0$. Empirically, we find that $N \approx 8$ recovers most of the benefit across model scales, requiring only eight stored hidden states per token (see § 5).

Beyond memory and computation, the block structure also benefits inference latency: block boundaries define the dispatch granularity for the blockwise optimization described in §3, and the fixed block count N bounds the KV cache size. The parallel inter-block results are merged with the sequential intra-block partial sums via online softmax [31], preserving exact equivalence (§4).

4 Infrastructure Design

Block AttnRes introduces additional system challenges compared to standard residual connections. For large-scale model training, block representations must be propagated across pipeline stages, causing heavy communication in a

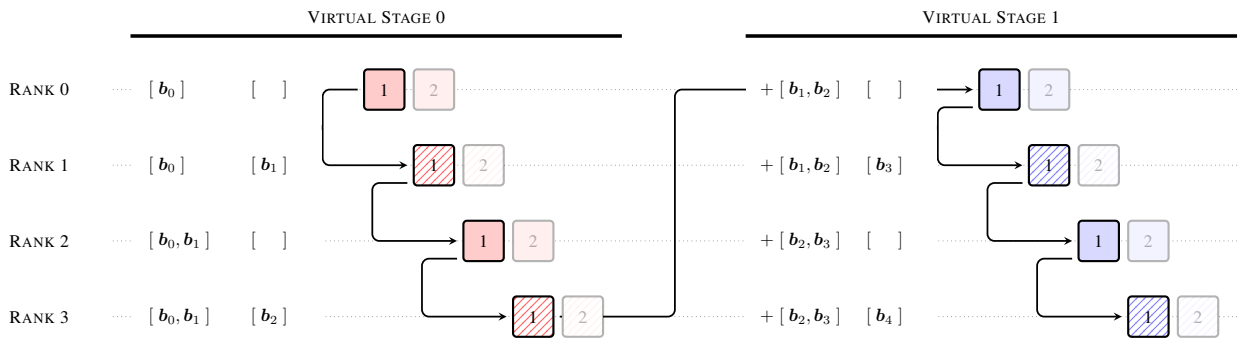


Figure 3: Cache-based pipeline communication example with 4 physical ranks and 2 virtual stages per rank, where hatched boxes denote end of AttnRes blocks. Numbers indicate micro-batch indices. Each rank caches previously received blocks; stage transitions only transmit incremental blocks ($+[\boldsymbol{b}_1, \boldsymbol{b}_2]$) instead of the full history.

naïve implementation. During inference, repeated access to accumulated block representations increases latency, while long-context prefilling amplifies the memory cost of caching block representations. We address these challenges with cross-stage caching in training, and with a two-phase computation strategy together with a memory-efficient prefilling scheme in inference.

4.1 Training

For small-scale training, AttnRes adds a tiny computation overhead and no extra memory usage, as the activations need to be saved for backpropagation regardless. Under large-scale distributed training, pipeline parallelism poses the primary infrastructure challenge for AttnRes. Full AttnRes requires all L layer outputs to be transmitted across stages; Block AttnRes reduces this to N block representations, and the optimizations below further minimize the remaining overhead.

Pipeline communication. With standard residual connections, pipeline parallelism [18] transfers a fixed-size hidden state between adjacent stages, independent of pipeline depth. Block AttnRes requires all accumulated block representations at each stage for inter-block attention, and naïvely transmitting the full history at every transition incurs redundant communication.

Consider an interleaved pipeline schedule [33] with P physical stages and V virtual stages per physical stage. For simplicity, assume each physical stage produces on average N_p block representations of dimension d per token.[1] With $C = PV$ total chunks (each physical stage in each virtual stage), the j-th chunk accumulates jN_p blocks. Naïvely transmitting all accumulated blocks at every transition incurs per-token communication cost:

$$\text{Comm}_{\text{naïve}} = \sum_{j=1}^{C-1} jN_p \cdot d = \frac{C(C-1)}{2} N_p d. \tag{7}$$

Cross-stage caching. Since each physical stage processes multiple virtual stages in succession, we can eliminate this redundancy by caching blocks locally: blocks received during earlier virtual stages remain in local memory and need not be re-transmitted. The first virtual stage ($v = 1$) has no cache and accumulates normally; for $v \geq 2$, each transition conveys only the $\sim PN_p$ incremental blocks accumulated since the receiver's corresponding chunk in the previous virtual stage. Total communication reduces to:

$$\text{Comm}_{\text{cached}} = \underbrace{\frac{P(P-1)}{2} N_p d}_{\text{first virtual stage}} + \underbrace{(V-1)\, P^2\, N_p d}_{\text{subsequent virtual stages}}. \tag{8}$$

Caching reduces peak per-transition cost from $O(C)$ to $O(P)$, a $V\times$ improvement that enables full overlap with computation during steady-state 1F1B. The backward pass benefits from the same scheme. Fig. 3 illustrates this optimization with $P{=}4$ and $V{=}2$: for the second virtual stage, caching eliminates 6 redundant block transmissions.

[1] In practice, block boundaries need not align with physical stage boundaries. For example, in Fig. 3, each block spans two physical stages, so only every other transition involves a newly completed block.

Algorithm 1: Two-phase computation for block n

Input: Pseudo queries $\{\boldsymbol{w}_l\}_{l\in\mathcal{B}_n}$, block representations $\{\boldsymbol{b}_0,\ldots,\boldsymbol{b}_{n-1}\}$

/* Phase 1: Parallel inter-block attention */
$\mathbf{Q}\leftarrow[\boldsymbol{w}_l]_{l\in\mathcal{B}_n}$ // $[S,d]$
$\mathbf{K},\mathbf{V}\leftarrow[\boldsymbol{b}_0;\ldots;\boldsymbol{b}_{n-1}]$ // $[n,d]$
$\{\boldsymbol{o}_l^{(1)},m_l^{(1)},\ell_l^{(1)}\}_{l\in\mathcal{B}_n}\leftarrow\textsc{AttnWithStats}(\mathbf{Q},\mathbf{K},\mathbf{V})$ // Return LSE

/* Phase 2: Sequential intra-block attention + Online softmax merge */
$i\leftarrow 0$
for $l\in\mathcal{B}_n$ **do**
  if $i=0$ **then**
    $\boldsymbol{h}_l\leftarrow\boldsymbol{o}_l^{(1)}/\ell_l^{(1)}$ // Inter-block only
  else
    $\boldsymbol{o}_l^{(2)},m_l^{(2)},\ell_l^{(2)}\leftarrow\textsc{AttnWithStats}(\boldsymbol{w}_l,\boldsymbol{b}_n^i,\boldsymbol{b}_n^i)$ // Intra-block
    $m_l\leftarrow\max(m_l^{(1)},m_l^{(2)})$
    $\boldsymbol{h}_l\leftarrow\dfrac{e^{m_l^{(1)}-m_l}\boldsymbol{o}_l^{(1)}+e^{m_l^{(2)}-m_l}\boldsymbol{o}_l^{(2)}}{e^{m_l^{(1)}-m_l}\ell_l^{(1)}+e^{m_l^{(2)}-m_l}\ell_l^{(2)}}$ // Online softmax merge
  $i\leftarrow i+1$
  $\boldsymbol{b}_n^i\leftarrow\boldsymbol{b}_n^{i-1}+f_l(\boldsymbol{h}_l)$ // Update partial sum; $\boldsymbol{b}_n^0\coloneqq\mathbf{0}$
return $\{\boldsymbol{h}_l\}_{l\in\mathcal{B}_n}$

Memory overhead. With cross-stage caching, each block is stored exactly once across all V virtual stages, which becomes negligible relative to standard per-layer activation cache. Crucially, the per-layer activation footprint remains identical to standard architectures, as activation checkpointing eliminates all inter-block attention intermediates, and the checkpointed input $\boldsymbol{p}_l$ matches the memory size of the hidden state $\boldsymbol{h}_l$ it replaces.

In terms of wall-clock time, Block AttnRes adds negligible training overhead when pipeline parallelism is not enabled; under pipeline parallelism, the measured end-to-end overhead is less than 4%.

4.2 Inference

The two-phase computation strategy described below applies to both Full and Block AttnRes: in either case, layers are grouped into blocks of size S, with Phase 1 batching the inter-block queries and Phase 2 handling sequential intra-block lookback. For Full AttnRes, this reduces per-layer I/O from $O(Ld)$ to $O((S+N)d)$ (detailed derivation shown in Appendix B); Block AttnRes further reduces the stored representations from L to N, since each block is compressed into a single vector. In what follows, we focus on Block AttnRes and detail the two-phase computation strategy together with a sequence-sharded prefilling scheme for long-context inputs.

Two-phase computation strategy. The layer-wise attention computation of Block AttnRes resembles autoregressive decoding, where block representations serve as a shared KV cache reused across layers. A naïve implementation computes the attention residual at every layer, each requiring a full pass over all preceding blocks, resulting in $O(L\cdot N)$ memory accesses. Since the pseudo-query vectors are decoupled from the forward computation (§3), all $S=L/N$ queries within a block can be batched into a single matrix multiplication, amortizing memory access from S reads to 1.

Algorithm 1 instantiates a two-phase computation strategy exploiting this property.

- **Phase 1** computes inter-block attention for all S layers simultaneously via a single batched query against the cached block representations, returning both outputs and softmax statistics (max and log-sum-exp). This amortizes the memory access cost, reducing reads from S times to just once per block.
- **Phase 2** computes intra-block attention sequentially for each layer using the evolving partial sum, then merges with Phase 1 outputs through online softmax [31]. Because the online-softmax merge is elementwise, this phase naturally admits kernel fusion with surrounding operations, further reducing I/O overhead.

With the two-phase design, Phase 2 preserves an I/O footprint similar to that of standard residual connections, whereas the main additional cost arises from Phase 1 inter-block attention. Because these inter-block reads are amortized across

all layers in a block through batching, the total per-layer memory access cost remains only $(\frac{N}{S}+3)d$ reads and $2d$ writes (Table 1). This is substantially lower than the residual-stream I/O of prior residual generalizations such as (m)HC under typical settings. In practice, Phase 1 can also partially overlap with the computation of the first layer in the block, further reducing its wall-clock impact. As a result, the end-to-end inference latency overhead is less than 2% on typical inference workloads.

Table 1: Memory access cost per token per layer incurred by the residual mechanism under each scheme. The internal I/O of the layer function f_l is excluded. For AttnRes, both Full and Block variants use the two-phase inference schedule described in Appendix B; amortized costs are averaged over N layers within a block. Typical values: L=128, N=8, S=L/N=16, m=4.

		Operation	Read	Write	Total I/O	
					Symbolic	Typical
Standard Residuals		Residual Merge	$2d$	d	$3d$	$3d$
mHC (m streams)		Compute $\boldsymbol{\alpha}_l$, $\boldsymbol{\beta}_l$, $\mathbf{A}_l$ Apply $\boldsymbol{\alpha}_l$ Apply $\boldsymbol{\beta}_l$ Apply $\mathbf{A}_l$ Residual Merge	md $md{+}m$ $d{+}m$ $md{+}m^2$ $2md$	$m^2{+}2m$ d md md md	$(8m{+}2)d{+}2m^2{+}4m$	$34d$
AttnRes	Full	Phase 1 (amortized) Phase 2	$(N{-}1)d$ $(S{-}1)d$	d d	$(S{+}N)d$	$24d$
	Block	Phase 1 (amortized) Phase 2	$\frac{N}{S}d$ $3d$	d d	$\left(\frac{N}{S}{+}5\right)d$	$5.5d$

Memory-efficient prefilling. Storing block representations during prefilling requires $N \cdot T \cdot d$ elements, which incurs 15 GB of memory for a 128K-token sequence with 8 blocks. We mitigate this by sharding these representations along the sequence dimension across P tensor-parallel devices, allowing Phase 1 to execute independently on local sequence shards. The Phase 2 online-softmax merge then integrates into the standard TP all-reduce communication path: the output is reduce-scattered, merged locally, and reconstructed via all-gather, naturally admitting kernel fusion with operations like RMSNorm. This reduces the per-device memory footprint to $N \cdot (T/P) \cdot d$—lowering the 128K-context example from 15 GB to roughly 1.9 GB per device. Combined with chunked prefill (e.g., 16K chunk size), the overhead further reduces to under 0.3 GB per device.

5 Experiments

Architecture Details. Our architecture is identical to Kimi Linear [69], a Mixture-of-Experts (MoE) Transformer following the Moonlight [28] / DeepSeek-V3 [9] design, which interleaves Kimi Delta Attention (KDA) and Multi-Head Latent Attention (MLA) layers in a 3:1 ratio, each followed by an MoE feed-forward layer. The only modification is the addition of AttnRes to the residual connections; all other components (model depth, hidden dimensions, expert routing, and MLP structure) remain unchanged. AttnRes introduces only one RMSNorm and one pseudo-query vector $\boldsymbol{w}_l \in \mathbb{R}^d$ per layer, amounting to a negligible fraction of the total parameter count. Crucially, all pseudo-query vectors must be initialized to zero. This ensures that the initial attention weights $\alpha_{i \to l}$ are uniform across source layers, which reduces AttnRes to an equal-weight average at the start of training and prevents training volatility, as we validated empirically.

5.1 Scaling Laws

We sweep five model sizes (Table 2) and train three variants per size: a PreNorm baseline, Full AttnRes, and Block AttnRes with ≈ 8 blocks. They are trained with an 8192-token context window and a cosine learning rate schedule. Within each scaling law size group, all variants share identical hyperparameters selected under the baseline to ensure fair comparison; this setup intentionally favors the baseline and thus makes the comparison conservative. Following standard practice, we fit power-law curves of the form $\mathcal{L} = A \times C^{-\alpha}$ [22, 15], where $\mathcal{L}$ is validation loss and C is compute measured in PFLOP/s-days.

Scaling Behavior. Fig. 4 presents the fitted scaling curves. The Baseline follows $\mathcal{L} = 1.891 \times C^{-0.057}$, while Block AttnRes fits $\mathcal{L} = 1.870 \times C^{-0.058}$, and Full AttnRes fits $\mathcal{L} = 1.865 \times C^{-0.057}$. All three variants exhibit a similar slope, but AttnRes consistently achieves lower loss across the entire compute range. Based on the fitted curves, at 5.6

Table 2: Baseline vs Block AttnRes ($N = 8$) vs Full AttnRes vs mHC(-lite) [64]: Model configurations, Hyperparameters, and Validation Loss.

# Act. Params†	Tokens	L_b	H	d_{model}	d_{ff}	lr	batch size‡	Val. Loss			
								Baseline	Block AttnRes	Full AttnRes	mHC(-lite)
194M	38.7B	12	12	896	400	2.99×10^{-3}	192	1.931	1.909	**1.899**	1.906
241M	45.4B	13	13	960	432	2.80×10^{-3}	256	1.895	1.875	1.874	**1.869**
296M	62.1B	14	14	1024	464	2.50×10^{-3}	320	1.829	1.809	**1.804**	1.807
436M	87.9B	16	16	1168	528	2.20×10^{-3}	384	1.766	1.746	**1.737**	1.747
528M	119.0B	17	17	1264	560	2.02×10^{-3}	432	1.719	1.693	**1.692**	1.694

† Denotes the number of activated parameters in our MoE models, excluding embeddings.
‡ All models were trained with a context length of 8192.
⋆ $L_b = L/2$ denotes the number of Transformer blocks.

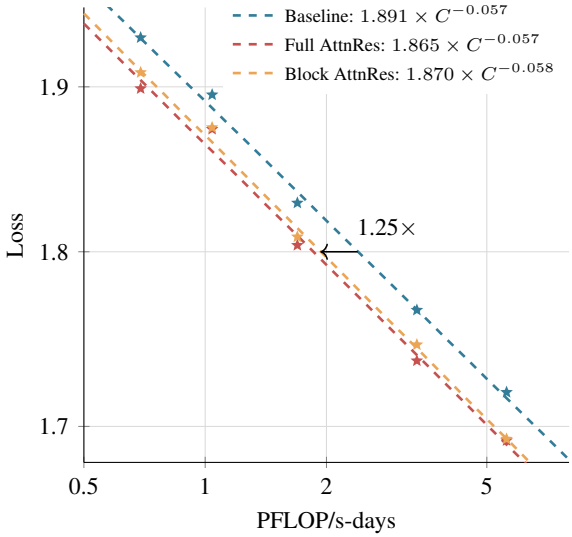


Figure 4: Scaling law curves for Attention Residuals. Both Full and Block AttnRes consistently outperform the baseline across all scales. Block AttnRes closely tracks Full AttnRes, recovering most of the gain at the largest scale.

PFLOP/s-days, Block AttnRes reaches 1.692 versus the Baseline's 1.714, equivalent to a $1.25\times$ compute advantage. The gap between Full and Block AttnRes narrows with scale, shrinking to just 0.001 at the largest size. We also list mHC(-lite) [64] in Table 2 for reference. Full AttnRes outperforms mHC, while Block AttnRes matches it at lower memory I/O per layer: $5.5d$ versus $34d$ for mHC with m=4 streams (Table 1).

5.2 Main Results

Training recipe. The largest models we study are based on the full Kimi Linear 48B configuration: 27 Transformer blocks (54 layers) with 8 out of 256 routed experts plus 1 shared expert, yielding 48B total and 3B activated parameters. This model applies Block AttnRes with 6 layers per block, producing 9 blocks plus the token embedding for a total of 10 depth-wise sources.

We follow the same data and training recipe as the Kimi Linear 1.4T-token runs [69]: all models are pre-trained with a 4096-token context window, the Muon optimizer [28], and a WSD (Warmup–Stable–Decay) learning rate schedule [16], with a global batch size of 8M tokens. Training of the final model proceeds in two stages: (i) a WSD pre-training phase on 1T tokens, followed by (ii) a mid-training phase on $\approx$400B high-quality tokens, following the annealing recipe of Moonlight [28].

After mid-training, we continue training with progressively longer sequence length of 32K tokens. Since our architecture uses hybrid KDA/MLA attention [69], where MLA operates without positional encodings (NoPE) [61], context extension requires no modifications such as YaRN [37] or attention temperature rescaling.

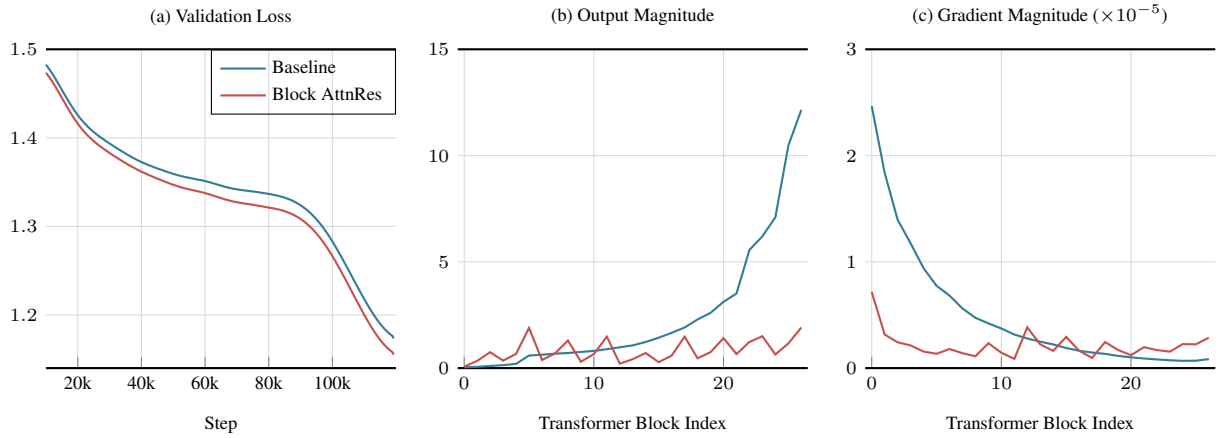


Figure 5: Training dynamics of Baseline and Block AttnRes. **(a)** Validation loss during training. **(b)** Each transformer block's output magnitude at the end of training. **(c)** Each transformer block's gradient magnitude.

Training dynamics. We compare the training dynamics of our final Baseline and Block AttnRes models over 1T tokens in Fig. 5.

- **Validation loss:** AttnRes achieves consistently lower validation loss throughout training, with the gap widening during the decay phase and resulting in a notably lower final loss.
- **Output magnitude:** The Baseline suffers from the PreNorm dilution problem [60, 27]: as hidden-state magnitudes grow monotonically with depth, deeper layers are compelled to learn increasingly large outputs from fixed-scale normalized inputs to remain influential. Block AttnRes confines this growth within each block, as selective aggregation at block boundaries resets the accumulation, yielding a bounded periodic pattern.
- **Gradient magnitude:** With all residual weights fixed to 1, the Baseline provides no means of regulating gradient flow across depth, leading to disproportionately large gradients in the earliest layers. The learnable $\mathrm{softmax}$ weights in Block AttnRes (Fig. 8) introduce competition among sources for probability mass, resulting in a substantially more uniform gradient distribution.

Table 3: Performance comparison of AttnRes with the baseline, both after the same pre-training recipe. Best per-row results are **bolded**.

		Baseline	AttnRes
General	MMLU	73.5	**74.6**
	MMLU-Pro	**52.2**	**52.2**
	GPQA-Diamond	36.9	**44.4**
	BBH	76.3	**78.0**
	ARC-Challenge	64.6	**65.7**
	HellaSwag	83.2	**83.4**
	TriviaQA	69.9	**71.8**
Math & Code	GSM8K	81.7	**82.4**
	MGSM	64.9	**66.1**
	Math	53.5	**57.1**
	CMath	84.7	**85.1**
	HumanEval	59.1	**62.2**
	MBPP	72.0	**73.9**
Chinese	CMMLU	82.0	**82.9**
	C-Eval	79.6	**82.5**

Downstream performance. Following the evaluation protocol of Kimi Linear [69], we assess both models across three areas (Table 3):

Table 4: Ablation on key components of AttnRes (16-layer model).

Variant		Loss
Baseline (PreNorm)		1.766
DenseFormer [36]		1.767
mHC [59]		1.747
AttnRes	Full	1.737
	w/ input-dependent query	1.731
	w/ input-independent mixing	1.749
	w/ sigmoid	1.741
	w/o RMSNorm	1.743
	SWA ($W = 1 + 8$)	1.764
	Block ($S = 4$)	1.746
	w/ multihead ($H = 16$)	1.752
	w/o RMSNorm	1.750

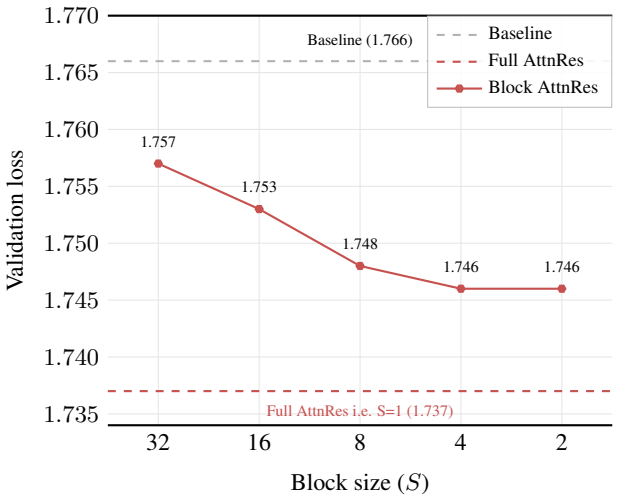


Figure 6: Effect of block size on validation loss (16-layer model).

- **Language understanding and reasoning**: MMLU [13], MMLU-Pro Hard [55], GPQA-Diamond [41], BBH [48], ARC-Challenge [6], HellaSwag [65], and TriviaQA [21].
- **Reasoning (Code and Math)**: GSM8K [7], MGSM [44], Math [25], CMath [14], HumanEval [5], and MBPP [1].
- **Chinese language understanding**: CMMLU [26] and C-Eval [19].

As shown in Table 3, Block AttnRes matches or outperforms the baseline on all benchmarks. The improvements are particularly pronounced on multi-step reasoning tasks such as GPQA-Diamond (+7.5) and Minerva Math (+3.6), as well as code generation such as HumanEval (+3.1), while knowledge-oriented benchmarks such as MMLU (+1.1) and TriviaQA (+1.9) also show solid gains. This pattern is consistent with the hypothesis that improved depth-wise information flow benefits compositional tasks, where later layers can selectively retrieve and build upon earlier representations.

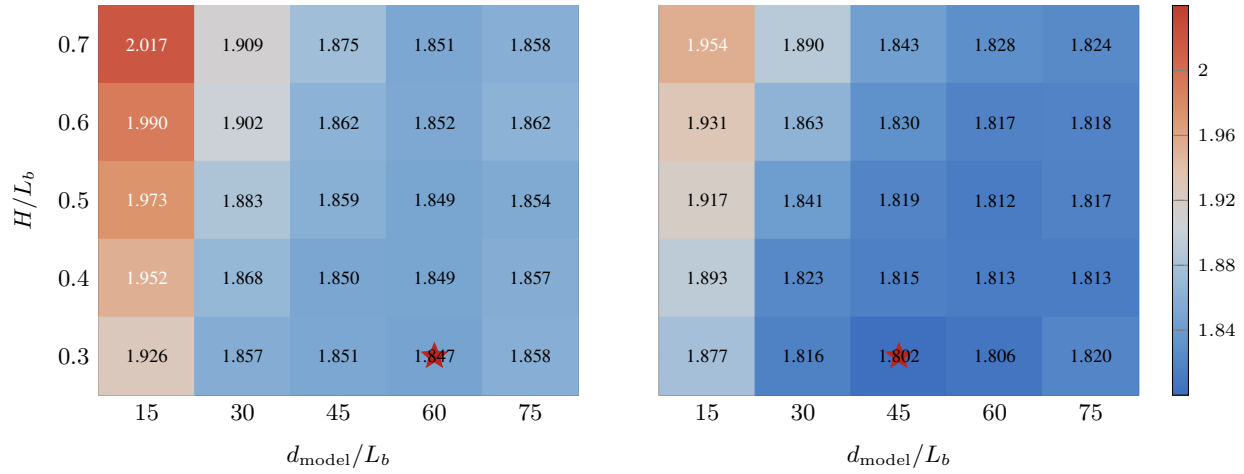
5.3 Ablation Study

We conduct ablation studies on the 16-head model from Table 2 to validate key design choices in AttnRes (Table 4). All models share identical hyperparameters and compute budget.

Comparison with prior methods. We compare AttnRes against the PreNorm baseline (loss 1.766) and two representative methods that generalize residual connections. DenseFormer [36] grants each layer access to all previous outputs but combines them with fixed, input-independent scalar coefficients; it shows no gain over the baseline (1.767), highlighting the importance of input-dependent weighting. mHC [59] introduces input dependence through m parallel streams with learned mixing matrices, improving to 1.747. AttnRes takes this further with explicit content-dependent selection via softmax attention: Full AttnRes achieves 1.737 and Block AttnRes 1.746, outperforming both methods with only a single query vector per layer.

Cross-layer access. We compare three granularities of cross-layer access. Full AttnRes follows directly from the time–depth duality (§ 3), applying attention over all previous layers, and achieves the lowest loss (1.737). A simple way to reduce its memory cost is sliding-window aggregation (SWA), which retains only the most recent $W{=}8$ layer outputs plus the token embedding; it improves over baseline (1.764) but falls well short of both Full and Block AttnRes, suggesting that selectively accessing distant layers matters more than attending to many nearby ones.

Block AttnRes offers a better trade-off: with block size $S{=}4$ it reaches 1.746 while keeping memory overhead constant per layer. Fig. 6 sweeps S across the full spectrum from $S{=}1$ (i.e. Full AttnRes) to increasingly coarse groupings. Loss degrades gracefully as S grows, with $S{=}2, 4, 8$ all landing near 1.746 while larger blocks ($S{=}16, 32$) move toward baseline. In practice, we fix the number of blocks to ≈ 8 for infrastructure efficiency (§ 4). As future hardware alleviates memory capacity constraints, adopting finer-grained block sizes or Full AttnRes represents a natural pathway to further improve performance.



(a) Baseline (b) Attention Residuals

Figure 7: Architecture sweep under fixed compute ($\approx 6.5 \times 10^{19}$ FLOPs, $\approx 2.3 \times 10^{8}$ active parameters). Each cell reports validation loss for a $(d_{\text{model}}/L_b,\ H/L_b)$ configuration, where $L_b = L/2$ is the number of Transformer blocks; the star marks the optimum.

Component design. We further ablate individual components of the attention mechanism:

- **Input-dependent query.** A natural extension is to make the query input-dependent by projecting it from the current hidden state. This further lowers loss to 1.731, but introduces a $d \times d$ projection per layer and requires sequential memory access during decoding, so we default to the learned query.
- **Input-independent mixing.** We removed the query and key and replaced them with learnable, input-independent scalars to weigh previous layers, which hurts performance (1.749 vs. 1.737).
- $\mathrm{softmax}$ **vs.** $\mathrm{sigmoid}$**.** Replacing $\mathrm{softmax}$ with $\mathrm{sigmoid}$ degrades performance (1.741). We attribute this to $\mathrm{softmax}$'s competitive normalization, which forces sharper selection among sources.
- **Multihead attention.** We test per-head depth aggregation (H=16) on Block AttnRes, allowing different channel groups to attend to different source layers. This hurts performance (1.752 vs. 1.746), indicating that the optimal depth-wise mixture is largely uniform across channels: when a layer's output is relevant, it is relevant as a whole.
- $\mathrm{RMSNorm}$ **on keys.** Removing $\mathrm{RMSNorm}$ degrades both Full AttnRes (1.743) and Block AttnRes (1.750). For Full AttnRes, it prevents individual layers with naturally larger outputs from dominating the $\mathrm{softmax}$. This becomes even more critical for Block AttnRes, as block-level representations accumulate over more layers and can develop large magnitude differences; $\mathrm{RMSNorm}$ prevents these from biasing the attention weights.

5.4 Analysis

5.4.1 Optimal Architecture

To understand how AttnRes reshapes optimal architectural scaling, we perform a controlled capacity reallocation study under a fixed compute and parameter budget. Our central question is whether AttnRes alters the preferred depth–width–attention trade-off, and in particular, given its potential strength on the depth dimension, whether it favors deeper models compared to conventional Transformer design heuristics. To isolate structural factors directly coupled to depth, we fix the per-expert MLP expansion ratio based on internal empirical observations ($d_{\text{ff}}/d_{\text{model}} \approx 0.45$). We further fix total training compute (FLOPs $\approx 6.5 \times 10^{19}$) and active parameters ($\approx 2.3 \times 10^{8}$), ensuring that any performance variation arises purely from architectural reallocation rather than overall capacity differences. Under this constrained budget, we enumerate 25 configurations on a 5×5 grid over $d_{\text{model}}/L_b \in \{15, 30, 45, 60, 75\}$ and $H/L_b \in \{0.3, 0.4, 0.5, 0.6, 0.7\}$, where $L_b = L/2$ is the number of Transformer blocks and H the number of attention heads. The results are shown in Fig. 7.

Both heatmaps exhibit a shared pattern: loss decreases with growing d_{model}/L_b and shrinking H/L_b, and both methods reach their optima at $H/L_b \approx 0.3$. Despite this shared trend, AttnRes achieves a lower loss than the baseline in each of the 25 configurations, by 0.019–0.063. The most apparent difference lies in the location of the optimum: the baseline achieves its lowest loss at $d_{\text{model}}/L_b \approx 60$ (1.847), whereas AttnRes shifts it to $d_{\text{model}}/L_b \approx 45$ (1.802). Under a fixed

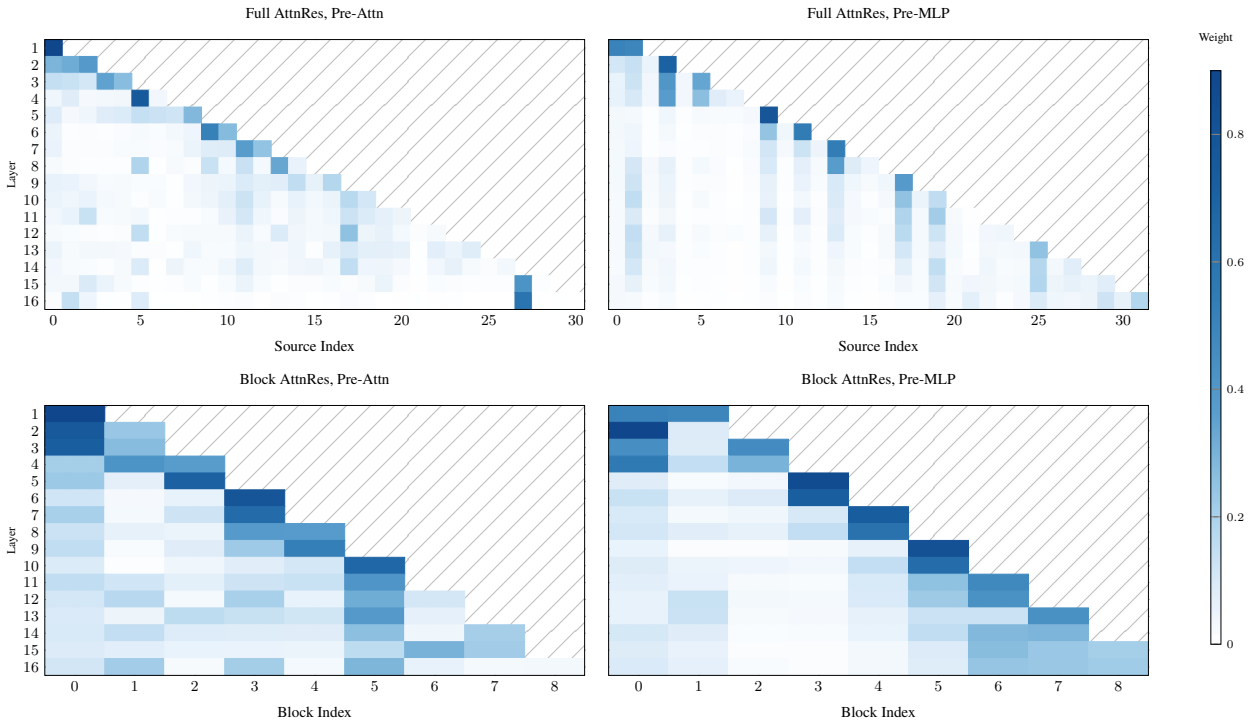


Figure 8: Depth-wise attention weight distributions for a 16-head model with full (top) and block (bottom) Attention Residuals, averaged over tokens. The model has 16 attention and 16 MLP layers. Each row shows how the lth attention (left) or MLP (right) layer distributes weight over previous sources. Diagonal dominance indicates locality remains the primary information pathway, while persistent weights on source 0 (embedding) and occasional off-diagonal concentrations reveal learned skip connections. Block attention ($N = 8$) recovers the essential structure with sharper, more decisive weight distributions.

parameter budget, a lower d_{model}/L_b corresponds to a deeper, narrower network, suggesting that AttnRes can exploit additional depth more effectively. We note that this preference for depth does not directly translate to a deployment recommendation, as deeper models generally incur higher inference latency due to their sequential computation [39]. Rather, this sweep serves as a diagnostic that reveals where AttnRes benefits most, and this depth preference can be factored into the architecture selection alongside inference cost.

5.4.2 Analyzing Learned AttnRes Patterns

We visualize the learned weights $\alpha_{i \to l}$ in Fig. 8 for the 16-head model (from Table 2) with both full and block (N=8) AttnRes. Each heatmap shows how the lth attention or MLP layer (rows) allocates its attention over previous sources (columns), with pre-attention and pre-MLP layers shown separately. We highlight three key observations:

- **Preserved locality.** Each layer attends most strongly to its immediate predecessor, yet selective off-diagonal concentrations emerge (e.g., layer 4 attending to early sources, layers 15–16 reaching back under the block setting), indicating learned skip connections beyond the standard residual path.

- **Layer specialization.** The embedding $\boldsymbol{h}_1$ retains non-trivial weight throughout, especially in pre-attention layers. Pre-MLP inputs show sharper diagonal reliance on recent representations, while pre-attention inputs maintain broader receptive fields, consistent with attention routing information across layers and MLPs operating locally.

- **Block AttnRes preserves structure.** Diagonal dominance, embedding persistence, and layer specialization all transfer from the full to the block variant, suggesting that block-wise compression acts as implicit regularization while preserving the essential information pathways.

Table 5: Comparison of residual update mechanisms. *Weight*: whether the mixing coefficients are architecture-fixed, learned-static (fixed after training), or input-dependent (dynamic). *Source*: which earlier representations layer l can access. Normalization is omitted from most formulas for clarity.

Method		Update rule	Weight	Source
Single-state recurrence: layer l receives only $\boldsymbol{h}_{l-1}$				
Residual [12]		$\boldsymbol{h}_l = \boldsymbol{h}_{l-1} + f_{l-1}(\boldsymbol{h}_{l-1})$	Fixed	$\boldsymbol{h}_{l-1}$
ReZero [2]		$\boldsymbol{h}_l = \boldsymbol{h}_{l-1} + \alpha_l \cdot f_{l-1}(\boldsymbol{h}_{l-1})$	Static	$\boldsymbol{h}_{l-1}$
LayerScale [50]		$\boldsymbol{h}_l = \boldsymbol{h}_{l-1} + \mathrm{diag}(\boldsymbol{\lambda}_l) \cdot f_{l-1}(\boldsymbol{h}_{l-1})$	Static	$\boldsymbol{h}_{l-1}$
Highway [45]		$\boldsymbol{h}_l = (1-\boldsymbol{g}_l) \odot \boldsymbol{h}_{l-1} + \boldsymbol{g}_l \odot f_{l-1}(\boldsymbol{h}_{l-1})$	Dynamic	$\boldsymbol{h}_{l-1}$
DeepNorm [54]		$\boldsymbol{h}_l = \mathrm{Norm}(\alpha\, \boldsymbol{h}_{l-1} + f_{l-1}(\boldsymbol{h}_{l-1}))$	Fixed	$\boldsymbol{h}_{l-1}$
KEEL [4]		$\boldsymbol{h}_l = \mathrm{Norm}(\alpha\, \boldsymbol{h}_{l-1} + f_{l-1}(\mathrm{Norm}(\boldsymbol{h}_{l-1})))$	Fixed	$\boldsymbol{h}_{l-1}$
Multi-state recurrence: layer l receives m streams				
SiameseNorm [27]		$\boldsymbol{h}_l^1 = \mathrm{Norm}(\boldsymbol{h}_{l-1}^1 + \boldsymbol{y}_{l-1});\ \boldsymbol{h}_l^2 = \boldsymbol{h}_{l-1}^2 + \boldsymbol{y}_{l-1}$	Fixed	2 streams
HC/mHC [72, 59]		$\mathbf{H}_l = \mathbf{H}_{l-1}\mathbf{A}_l + f_{l-1}(\mathbf{H}_{l-1}\boldsymbol{\alpha}_{l-1})\,\boldsymbol{\beta}_{l-1}^\top$	Dynamic	m streams
DDL [67]		$\mathbf{H}_l = (\mathbf{I} - \beta_l \boldsymbol{k}_l \boldsymbol{k}_l^\top)\mathbf{H}_{l-1} + \beta_l \boldsymbol{k}_l \boldsymbol{v}_l^\top$	Dynamic	d_v streams
Cross-layer access: layer l can access individual earlier-layer outputs				
DenseNet [17]		$\boldsymbol{h}_l = \mathrm{ConvPool}([\boldsymbol{h}_1;\ f_1(\boldsymbol{h}_1);\ \ldots;\ f_{l-1}(\boldsymbol{h}_{l-1})])$	Static	$[\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{l-1}]$
DenseFormer [36]		$\boldsymbol{h}_l = \alpha_{0\to l}\, \boldsymbol{h}_1 + \sum_{i=1}^{l-1} \alpha_{i\to l}\, f_i(\boldsymbol{h}_i)$	Static	$[\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{l-1}]$
MRLA [10][1]		$\boldsymbol{h}_l = \sum_{i=1}^{l-1} \sigma\big(\mathrm{ConvPool}(f_{l-1}(\boldsymbol{h}_{l-1}))\big)^\top \sigma\big(\mathrm{ConvPool}(f_i(\boldsymbol{h}_i))\big)\, \mathrm{Conv}(f_i(\boldsymbol{h}_i))$	Dynamic	$[\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{l-1}]$
AttnRes (ours)	Full[2]	$\boldsymbol{h}_l \propto \sum_{i=0}^{l-1} \phi(\boldsymbol{w}_l, \boldsymbol{k}_i)\, \boldsymbol{v}_i$	Dynamic	$[\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{l-1}]$
	Block[3]	$\boldsymbol{h}_l \propto \sum_{i=0}^{n-1} \phi(\boldsymbol{w}_l, \boldsymbol{k}_i)\, \boldsymbol{v}_i + \phi(\boldsymbol{w}_l, \boldsymbol{k}_n^j)\, \boldsymbol{v}_n^j$	Dynamic	$[\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{n-1}, \boldsymbol{b}_n^j]$

[1] ConvPool: pooling operation followed by convolution (channel projection).
[2] $\phi(\boldsymbol{q}, \boldsymbol{k}) = \exp\big(\boldsymbol{q}^\top \mathrm{RMSNorm}(\boldsymbol{k})\big)$; $\boldsymbol{k}_i = \boldsymbol{v}_i$; $\boldsymbol{v}_0 = \boldsymbol{h}_1$, $\boldsymbol{v}_{i\geq 1} = f_i(\boldsymbol{h}_i)$. softmax jointly normalized over all sources.
[3] Same ϕ and normalization as Full; $\boldsymbol{v}_i = \boldsymbol{b}_i$, $\boldsymbol{v}_n^j = \boldsymbol{b}_n^j$.

6 Discussions

6.1 Sequence-Depth Duality

Residual connections propagate information over depth via a fixed recurrence $\boldsymbol{h}_l = \boldsymbol{h}_{l-1} + f_{l-1}(\boldsymbol{h}_{l-1})$, much as RNNs propagate information over time. Test-Time Training (TTT) [46] formalizes the sequence side of this analogy (cf. Fast Weight Programmers [43, 32]), casting each recurrent step as gradient descent on a self-supervised loss:

$$\mathbf{W}_t = \mathbf{W}_{t-1} - \eta\, \nabla \ell(\mathbf{W}_{t-1};\, \boldsymbol{x}_t), \tag{9}$$

where a slow network parameterizes ℓ and the state $\mathbf{W}$ is updated once per token. When f is linear, this reduces to vanilla linear attention $\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{k}_t \boldsymbol{v}_t^\top$. The standard residual exhibits the same additive form along depth, with $\boldsymbol{h}_l$ serving as the state and each layer f_l acting as one "gradient step."

As noted by [4], this duality extends to richer variants (Table 5). Data-dependent gates on the sequence side [47, 63] correspond to Highway networks [45] on the depth side; the delta rule [42, 62, 69] corresponds to DDL [67]; and MRLA [10] mirrors GLA's [63] gated linear attention. These methods all refine the recurrent update while remaining within the recurrence paradigm. AttnRes goes a step further and replaces depth-wise recurrence with direct cross-layer attention, just as Transformers replaced temporal recurrence with self-attention. Since the number of layers in current architectures remains well within the practical regime of softmax attention, we adopt vanilla depth-wise attention. Incorporating more expressive yet memory-efficient (e.g. linear-complexity) alternatives is a natural direction for future work.

6.2 Residual Connections as Structured Matrices

The residual variants discussed above can all be viewed as weighted aggregations over previous layer outputs. We formalize this with a *depth mixing matrix* $\mathbf{M} \in \mathbb{R}^{L\times L}$, where $\mathbf{M}_{i\to l}$ is the weight that layer l assigns to the output of layer i. The variants differ in how these weights arise (fixed, learned, or input-dependent) and whether $\mathbf{M}$ is constrained to low rank or allowed to be dense. The semiseparable rank of $\mathbf{M}$ [8] offers a unified lens for comparing them.

Concretely, the input to layer l is $\boldsymbol{h}_l = \sum_{i=0}^{l-1} \mathbf{M}_{i\to l}\, \boldsymbol{v}_i$, where $\boldsymbol{v}_0 = \boldsymbol{h}_1$ (embedding) and $\boldsymbol{v}_i = f_i(\boldsymbol{h}_i)$ for $i \geq 1$. Fig. 9 visualizes $\mathbf{M}$ for representative methods; we derive each below.

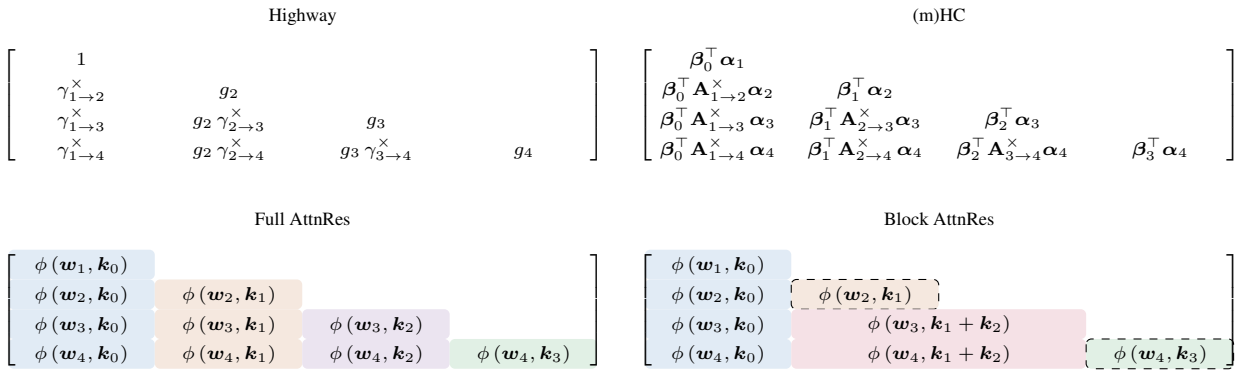


Figure 9: Depth mixing matrices $\mathbf{M}$ for four residual variants (L=4; Block AttnRes uses block size S=2). Highway is shown with scalar gates for clarity. AttnRes panels show unnormalized ϕ scores; background colors group entries that share the same source (Full AttnRes) or the same source block (Block AttnRes).

- Standard residual [12], $\boldsymbol{h}_l = \boldsymbol{h}_{l-1} + f_{l-1}(\boldsymbol{h}_{l-1})$. Expanding gives $\boldsymbol{h}_l = \sum_{i=0}^{l-1} \boldsymbol{v}_i$, so $\mathbf{M}_{i\to l} = 1$ for all $i \le l$ and $\mathbf{M}$ is an all-ones lower-triangular matrix:

$$\begin{bmatrix} \boldsymbol{h}_1 \\ \boldsymbol{h}_2 \\ \vdots \\ \boldsymbol{h}_L \end{bmatrix} = \begin{bmatrix} 1 & & & \\ 1 & 1 & & \\ \vdots & \vdots & \ddots & \\ 1 & 1 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_0 \\ \boldsymbol{v}_1 \\ \vdots \\ \boldsymbol{v}_{L-1} \end{bmatrix}$$

- Highway [45], $\boldsymbol{h}_l = (1-g_l)\,\boldsymbol{h}_{l-1} + g_l\, f_{l-1}(\boldsymbol{h}_{l-1})$ (written here with scalar gates for clarity; the element-wise extension is straightforward). Defining the carry product $\gamma^{\times}_{i\to l} := \prod_{j=i+1}^{l}(1-g_j)$, the weights are $\mathbf{M}_{0\to l} = \gamma^{\times}_{1\to l}$ for the embedding and $\mathbf{M}_{i\to l} = g_{i+1}\,\gamma^{\times}_{i+1\to l}$ for $i \ge 1$. Since the cumulative products factor through scalar gates, $\mathbf{M}$ is 1-semiseparable [8], the same rank as the standard residual but with input-dependent weights. The weights sum to one by construction, making Highway a softmax-free depth-wise instance of stick-breaking attention [49].
- (m)HC [72, 59] maintain m parallel streams $\mathbf{H}_l \in \mathbb{R}^{d\times m}$, updated via

$$\mathbf{H}_l = \mathbf{H}_{l-1}\mathbf{A}_l + f_{l-1}(\mathbf{H}_{l-1}\boldsymbol{\alpha}_{l-1})\,\boldsymbol{\beta}_{l-1}^{\top},$$

  where $\mathbf{A}_l \in \mathbb{R}^{m\times m}$ is a learned transition matrix, $\boldsymbol{\alpha}_{l-1} \in \mathbb{R}^m$ mixes streams into a single input for f_{l-1}, and $\boldsymbol{\beta}_{l-1} \in \mathbb{R}^m$ distributes the output back across streams. Unrolling the recurrence gives the effective weight

$$\mathbf{M}_{i\to l} = \boldsymbol{\beta}_i^{\top}\,\mathbf{A}^{\times}_{i+1\to l}\,\boldsymbol{\alpha}_l, \tag{10}$$

  where $\mathbf{A}^{\times}_{i\to j} := \prod_{k=i+1}^{j}\mathbf{A}_k$. The $m\times m$ transitions render $\mathbf{M}$ m-semiseparable [8]. mHC [59, 64] further constrains each $\mathbf{A}_l$ to be doubly stochastic, stabilizing the cumulative products across depth.
- Full AttnRes computes $\mathbf{M}_{i\to l} = \alpha_{i\to l}$ via $\phi(\boldsymbol{w}_l, \boldsymbol{k}_i) = \exp\left(\boldsymbol{w}_l^{\top}\,\mathrm{RMSNorm}(\boldsymbol{k}_i)\right)$ with normalization, where $\boldsymbol{k}_i = \boldsymbol{v}_i$ are input-dependent layer outputs, yielding a dense, rank-L $\mathbf{M}$.
- Block AttnRes partitions layers into N blocks $\mathcal{B}_1, \ldots, \mathcal{B}_N$. For sources i in a completed earlier block $\mathcal{B}_n$, all share the block-level key/value $\boldsymbol{b}_n$, so $\mathbf{M}_{i\to l} = \alpha_{n\to l}$ for every $i \in \mathcal{B}_n$. Within the current block, each layer additionally attends over the evolving partial sum $\boldsymbol{b}_n^{i-1}$, introducing one extra distinct source per intra-block position. The effective rank of $\mathbf{M}$ therefore lies between N and $N+S$ (where S is the block size), interpolating between standard residual (N=1) and Full AttnRes (N=L).

Practicality. The structured-matrix perspective serves two purposes. First, it enables analytical insights that are not apparent from the recurrence form alone. The input-dependent $\mathbf{M}$ of AttnRes, for instance, reveals depth-wise attention sinks (§5.4.2), where certain layers consistently attract high weight regardless of input, mirroring the same phenomenon in sequence-wise attention [57]. Second, it informs new designs by exposing which properties of the kernel ϕ matter. For example, when ϕ decomposes as $\phi(\boldsymbol{q}, \boldsymbol{k}) = \varphi(\boldsymbol{q})^{\top}\varphi(\boldsymbol{k})$ for some feature map φ [23], depth-wise attention collapses into a recurrence—precisely the structure underlying the MRLA–GLA and DDL–DeltaNet correspondences noted above.

Prior Residuals as Depth-Wise Linear Attention The structured-matrix perspective further relates to the sequence-depth duality by showing that existing residual variants are, in effect, instances of *linear* attention over the depth axis. For example, the unrolled (m)HC weight $\mathbf{M}_{i\to l} = \boldsymbol{\beta}_i^\top \mathbf{A}^{\times}_{i+1\to l}\boldsymbol{\alpha}_l$ (Eq. 10) admits a natural attention interpretation in which $\boldsymbol{\alpha}_l$ plays the role of a query issued by layer l, $\boldsymbol{\beta}_i$ serves as a key summarizing the contribution of layer i, and the cumulative transition $\mathbf{A}^{\times}_{i+1\to l}$ acts as a depth-relative positional operator [69] governing the query–key interaction across intervening layers. Notably, the m parallel streams correspond to state expansion [40, 29] along the depth axis, expanding the recurrent state from d to $d\times m$ and thereby increasing the semiseparable rank of $\mathbf{M}$. [58] show that replacing $\mathbf{A}^{\times}_{i+1\to l}$ with the identity matrix still yields competitive performance, highlighting the role of state expansion. Through this lens, methods like (m)HC thus act as depth-wise *linear* attention with matrix-valued states, while AttnRes acts as depth-wise $\mathrm{softmax}$ attention.

7 Related Work

Normalization, Scaling, and Depth Stability. The standard residual update $\boldsymbol{h}_{l+1} = \boldsymbol{h}_l + f_l(\boldsymbol{h}_l)$ [12] presents a fundamental tension between *normalization placement* and *gradient propagation*. PostNorm [52] maintains bounded magnitudes but distorts gradients, as repeated normalization on the residual path compounds into gradient vanishing at depth [60]. PreNorm [34, 60] restores a clean identity path yet introduces unbounded magnitude growth: since $\|\boldsymbol{h}_l\|$ grows as $O(L)$, each layer's relative contribution shrinks, compelling deeper layers to produce ever-larger outputs and limiting effective depth [27]. Subsequent work reconciles both desiderata via scaled residual paths [54], hybrid normalization [73], amplified skip connections [4], or learned element-wise gates [45] (see Table 5). AttnRes sidesteps this tension by replacing the additive recurrence with selective aggregation over individual earlier-layer outputs, avoiding both the cumulative magnitude growth of PreNorm and the repeated scale contraction of PostNorm.

Multi-State Recurrence. All single-state methods above condition layer l only on $\boldsymbol{h}_{l-1}$, from which individual earlier-layer contributions cannot be selectively retrieved. Several methods address this by widening the recurrence to multiple parallel streams: Hyper-Connections [72] and its stabilized variant mHC [59] maintain m streams with learned mixing matrices; DDL [67] maintains a matrix state updated via a delta-rule erase-and-write mechanism; SiameseNorm [27] maintains two parameter-shared streams—one PreNorm and one PostNorm—to preserve identity gradients and bounded representations. While these methods alleviate information compression, they still condition on the immediate predecessor's state; AttnRes is orthogonal, providing selective access to individual earlier-layer outputs while remaining compatible with any normalization or gating scheme. We discuss the formal connection to Hyper-Connections in § 6.2.

Cross-Layer Connectivity. A separate line of work bypasses the single-state bottleneck by giving each layer direct access to individual earlier-layer outputs. The simplest approach uses static weights: DenseNet [17] concatenates all preceding feature maps; ELMo [38] computes a $\mathrm{softmax}$-weighted sum of layer representations with learned scalar weights; DenseFormer [36] and ANCRe [68] assign learned per-layer scalar coefficients fixed after training. For input-dependent aggregation, MUDDFormer [56] generates position-dependent weights via a small MLP across four decoupled streams; MRLA [10] applies element-wise sigmoid gating over all previous layers, though its separable query–key product is closer to linear attention than $\mathrm{softmax}$-based retrieval. Other methods trade full cross-layer access for more targeted designs: Value Residual Learning [71] accesses only a single earlier layer; LAuReL [30] augments the residual with low-rank projections over the previous k activations; Dreamer [24] combines sequence attention with depth attention and sparse experts. AttnRes combines $\mathrm{softmax}$-normalized, input-dependent weights with selective access to all preceding layers through a single d-dimensional pseudo-query per layer, and introduces a block structure reducing cost from $O(L^2)$ to $O(LN)$. Cache-based pipeline communication and a two-phase computation strategy (§ 4) make Block AttnRes practical at scale with negligible overhead.

Conclusion

Inspired by the duality between sequence and depth, we introduce AttnRes, which replaces fixed, uniform residual accumulation with learned, input-dependent depth-wise attention. We validate the method through ablation studies and scaling law experiments, showing that its gains persist across scales. Because Full AttnRes must access all preceding layer outputs at every layer, the memory footprint of cross-layer aggregation grows as $O(Ld)$, which is prohibitive for large-scale models on current hardware. We therefore introduce Block AttnRes, which partitions layers into N blocks and attends over block-level representations. Empirically, using about 8 blocks recovers most of the gains of Full AttnRes, while finer-grained blocking remains a promising direction as future hardware constraints relax. Together with cross-stage caching and a two-phase computation strategy, Block AttnRes is practical at scale, incurring only marginal training overhead and minimal inference overhead.

References

[1] Jacob Austin et al. *Program Synthesis with Large Language Models*. 2021. arXiv: 2108.07732 [cs.PL]. URL: https://arxiv.org/abs/2108.07732.

[2] Thomas Bachlechner et al. *ReZero is All You Need: Fast Convergence at Large Depth*. 2020. arXiv: 2003.04887 [cs.LG]. URL: https://arxiv.org/abs/2003.04887.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL]. URL: https://arxiv.org/abs/1409.0473.

[4] Chen Chen and Lai Wei. *Post-LayerNorm Is Back: Stable, ExpressivE, and Deep*. 2026. arXiv: 2601.19895 [cs.LG]. URL: https://arxiv.org/abs/2601.19895.

[5] Mark Chen et al. *Evaluating Large Language Models Trained on Code*. 2021. arXiv: 2107.03374 [cs.LG]. URL: https://arxiv.org/abs/2107.03374.

[6] Peter Clark et al. "Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge". In: *arXiv:1803.05457v1* (2018).

[7] Karl Cobbe et al. *Training Verifiers to Solve Math Word Problems*. 2021. arXiv: 2110.14168 [cs.LG]. URL: https://arxiv.org/abs/2110.14168.

[8] Tri Dao and Albert Gu. "Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality". In: *CoRR* abs/2405.21060 (2024). DOI: 10.48550/ARXIV.2405.21060. arXiv: 2405.21060. URL: https://doi.org/10.48550/arXiv.2405.21060.

[9] DeepSeek-AI et al. *DeepSeek-V3 Technical Report*. 2025. arXiv: 2412.19437 [cs.CL]. URL: https://arxiv.org/abs/2412.19437.

[10] Yanwen Fang et al. *Cross-Layer Retrospective Retrieving via Layer Attention*. 2023. arXiv: 2302.03985 [cs.CV]. URL: https://arxiv.org/abs/2302.03985.

[11] Andrey Gromov et al. *The Unreasonable Ineffectiveness of the Deeper Layers*. 2025. arXiv: 2403.17887 [cs.CL]. URL: https://arxiv.org/abs/2403.17887.

[12] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[13] Dan Hendrycks et al. *Measuring Massive Multitask Language Understanding*. 2021. arXiv: 2009.03300 [cs.CY]. URL: https://arxiv.org/abs/2009.03300.

[14] Dan Hendrycks et al. *Measuring Mathematical Problem Solving With the MATH Dataset*. 2021. arXiv: 2103.03874 [cs.LG]. URL: https://arxiv.org/abs/2103.03874.

[15] Jordan Hoffmann et al. *Training Compute-Optimal Large Language Models*. 2022. arXiv: 2203.15556 [cs.CL]. URL: https://arxiv.org/abs/2203.15556.

[16] Shengding Hu et al. *MiniCPM: Unveiling the Potential of Small Language Models with Scalable Training Strategies*. 2024. arXiv: 2404.06395 [cs.CL]. URL: https://arxiv.org/abs/2404.06395.

[17] Gao Huang et al. *Densely Connected Convolutional Networks*. 2018. arXiv: 1608.06993 [cs.CV]. URL: https://arxiv.org/abs/1608.06993.

[18] Yanping Huang et al. "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism". In: *Advances in NeurIPS*. 2019.

[19] Yuzhen Huang et al. "C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models". In: *Advances in NeurIPS* 36 (2023), pp. 62991–63010.

[20] Robert A. Jacobs et al. "Adaptive Mixtures of Local Experts". In: *Neural Computation* 3.1 (1991), pp. 79–87. DOI: 10.1162/neco.1991.3.1.79.

[21] Mandar Joshi et al. "Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension". In: *arXiv preprint arXiv:1705.03551* (2017).

[22] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. 2020. arXiv: 2001.08361 [cs.LG]. URL: https://arxiv.org/abs/2001.08361.

[23] Angelos Katharopoulos et al. "Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention". In: *Proceedings of ICML*. Ed. by Hal Daumé III and Aarti Singh. PMLR, 2020, pp. 5156–5165. URL: https://proceedings.mlr.press/v119/katharopoulos20a.html.

[24] Jonas Knupp et al. *Depth-Recurrent Attention Mixtures: Giving Latent Reasoning the Attention it Deserves*. 2026. arXiv: 2601.21582 [cs.AI]. URL: https://arxiv.org/abs/2601.21582.

[25] Aitor Lewkowycz et al. *Solving Quantitative Reasoning Problems with Language Models*. 2022. arXiv: 2206.14858 [cs.CL]. URL: https://arxiv.org/abs/2206.14858.

[26] Haonan Li et al. "CMMLU: Measuring massive multitask language understanding in Chinese". In: *Findings of the Association for Computational Linguistics: ACL 2024*. Ed. by Lun-Wei Ku, Andre Martins, and Vivek Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 11260–11285. DOI: 10.18653/v1/2024.findings-acl.671. URL: https://aclanthology.org/2024.findings-acl.671/.

[27] Tianyu Li et al. *SiameseNorm: Breaking the Barrier to Reconciling Pre/Post-Norm*. 2026. arXiv: 2602.08064 [cs.LG]. URL: https://arxiv.org/abs/2602.08064.

[28] Jingyuan Liu et al. *Muon is Scalable for LLM Training*. 2025. arXiv: 2502.16982 [cs.LG]. URL: https://arxiv.org/abs/2502.16982.

[29] Brian Mak and Jeffrey Flanigan. *Residual Matrix Transformers: Scaling the Size of the Residual Stream*. 2025. arXiv: 2506.22696 [cs.LG]. URL: https://arxiv.org/abs/2506.22696.

[30] Gaurav Menghani, Ravi Kumar, and Sanjiv Kumar. *LAuReL: Learned Augmented Residual Layer*. 2025. arXiv: 2411.07501 [cs.LG]. URL: https://arxiv.org/abs/2411.07501.

[31] Maxim Milakov and Natalia Gimelshein. *Online normalizer calculation for softmax*. 2018. arXiv: 1805.02867 [cs.PF]. URL: https://arxiv.org/abs/1805.02867.

[32] Tsendsuren Munkhdalai et al. "Metalearned Neural Memory". In: *ArXiv* abs/1907.09720 (2019). URL: https://api.semanticscholar.org/CorpusID:198179407.

[33] Deepak Narayanan et al. *Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM*. 2021. arXiv: 2104.04473 [cs.CL]. URL: https://arxiv.org/abs/2104.04473.

[34] Toan Q. Nguyen and Julian Salazar. "Transformers without Tears: Improving the Normalization of Self-Attention". In: *Proceedings of IWSLT*. Ed. by Jan Niehues et al. 2019. URL: https://aclanthology.org/2019.iwslt-1.17/.

[35] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774.

[36] Matteo Pagliardini et al. *DenseFormer: Enhancing Information Flow in Transformers via Depth Weighted Averaging*. 2024. arXiv: 2402.02622 [cs.CL]. URL: https://arxiv.org/abs/2402.02622.

[37] Bowen Peng et al. "Yarn: Efficient context window extension of large language models". In: *arXiv preprint arXiv:2309.00071* (2023).

[38] Matthew E. Peters et al. "Deep Contextualized Word Representations". In: *Proceedings of NAACL*. 2018, pp. 2227–2237. URL: https://aclanthology.org/N18-1202/.

[39] Reiner Pope et al. *Efficiently Scaling Transformer Inference*. 2022. arXiv: 2211.05102 [cs.LG].

[40] Zhen Qin et al. *HGRN2: Gated Linear RNNs with State Expansion*. 2024. arXiv: 2404.07904 [cs.CL].

[41] David Rein et al. "Gpqa: A graduate-level google-proof q&a benchmark". In: *First Conference on Language Modeling*. 2024.

[42] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. "Linear Transformers Are Secretly Fast Weight Programmers". In: *Proceedings of ICML*. Ed. by Marina Meila and Tong Zhang. PMLR, 2021, pp. 9355–9366. URL: https://proceedings.mlr.press/v139/schlag21a.html.

[43] Jürgen Schmidhuber. "Learning to control fast-weight memories: An alternative to dynamic recurrent networks". In: *Neural Computation* 4.1 (1992), pp. 131–139.

[44] Freda Shi et al. *Language Models are Multilingual Chain-of-Thought Reasoners*. 2022. arXiv: 2210.03057 [cs.CL]. URL: https://arxiv.org/abs/2210.03057.

[45] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. *Highway Networks*. 2015. arXiv: 1505.00387 [cs.LG]. URL: https://arxiv.org/abs/1505.00387.

[46] Yu Sun et al. "Learning to (Learn at Test Time): RNNs with Expressive Hidden States". In: *ArXiv* abs/2407.04620 (2024). URL: https://api.semanticscholar.org/CorpusID:271039606.

[47] Yutao Sun et al. *Retentive Network: A Successor to Transformer for Large Language Models*. 2023. arXiv: 2307.08621 [cs.CL].

[48] Mirac Suzgun et al. "Challenging big-bench tasks and whether chain-of-thought can solve them". In: *arXiv preprint arXiv:2210.09261* (2022).

[49] Shawn Tan et al. "Scaling Stick-Breaking Attention: An Efficient Implementation and In-depth Study". In: *Proceedings of ICLR*. 2025.

[50] Hugo Touvron et al. *Going deeper with Image Transformers*. 2021. arXiv: 2103.17239 [cs.CV]. URL: https://arxiv.org/abs/2103.17239.

[51] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[52] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in NeurIPS*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[53] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in NeurIPS*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[54] Hongyu Wang et al. *DeepNet: Scaling Transformers to 1,000 Layers*. 2022. arXiv: 2203.00555 [cs.CL]. URL: https://arxiv.org/abs/2203.00555.

[55] Yubo Wang et al. "Mmlu-pro: A more robust and challenging multi-task language understanding benchmark". In: *Advances in NeurIPS* 37 (2024), pp. 95266–95290.

[56] Da Xiao et al. "MUDDFormer: Breaking Residual Bottlenecks in Transformers via Multiway Dynamic Dense Connections". In: *Proceedings of ICML*. 2025.

[57] Guangxuan Xiao et al. "Efficient streaming language models with attention sinks". In: *arXiv preprint arXiv:2309.17453* (2023).

[58] Tian Xie. *Your DeepSeek mHC Might Not Need the "m"*. Zhihu blog post. 2026. URL: https://zhuanlan.zhihu.com/p/2010852389670908320.

[59] Zhenda Xie et al. *mHC: Manifold-Constrained Hyper-Connections*. 2026. arXiv: 2512.24880 [cs.CL]. URL: https://arxiv.org/abs/2512.24880.

[60] Ruibin Xiong et al. *On Layer Normalization in the Transformer Architecture*. 2020. arXiv: 2002.04745 [cs.LG]. URL: https://arxiv.org/abs/2002.04745.

[61] Bowen Yang et al. *Rope to Nope and Back Again: A New Hybrid Attention Strategy*. 2025. arXiv: 2501.18795 [cs.CL]. URL: https://arxiv.org/abs/2501.18795.

[62] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. "Gated Delta Networks: Improving Mamba2 with Delta Rule". In: *Proceedings of ICLR*. 2025. URL: https://openreview.net/forum?id=r8H7xhYPwz.

[63] Songlin Yang et al. "Gated Linear Attention Transformers with Hardware-Efficient Training". In: *Proceedings of ICML*. PMLR, 2024.

[64] Yongyi Yang and Jianyang Gao. *mHC-lite: You Don't Need 20 Sinkhorn-Knopp Iterations*. 2026. arXiv: 2601.05732 [cs.LG]. URL: https://arxiv.org/abs/2601.05732.

[65] Rowan Zellers et al. "HellaSwag: Can a Machine Really Finish Your Sentence?" In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019.

[66] Biao Zhang and Rico Sennrich. "Root mean square layer normalization". In: *Advances in NeurIPS* 32 (2019).

[67] Yifan Zhang et al. *Deep Delta Learning*. 2026. arXiv: 2601.00417 [cs.LG]. URL: https://arxiv.org/abs/2601.00417.

[68] Yilang Zhang et al. *ANCRe: Adaptive Neural Connection Reassignment for Efficient Depth Scaling*. 2026. arXiv: 2602.09009 [cs.LG]. URL: https://arxiv.org/abs/2602.09009.

[69] Yu Zhang et al. *Kimi Linear: An Expressive, Efficient Attention Architecture*. 2025. arXiv: 2510.26692 [cs.CL].

[70] Shu Zhong et al. *Understanding Transformer from the Perspective of Associative Memory*. 2025. arXiv: 2505.19488 [cs.LG]. URL: https://arxiv.org/abs/2505.19488.

[71] Zhanchao Zhou et al. "Value Residual Learning". In: *Proceedings of ACL*. Ed. by Wanxiang Che et al. Vienna, Austria, 2025, pp. 28341–28356. URL: https://aclanthology.org/2025.acl-long.1375/.

[72] Defa Zhu et al. *Hyper-Connections*. 2025. arXiv: 2409.19606 [cs.LG]. URL: https://arxiv.org/abs/2409.19606.

[73] Zhijian Zhuo et al. *HybridNorm: Towards Stable and Efficient Transformer Training via Hybrid Normalization*. 2025. arXiv: 2503.04598 [cs.CL]. URL: https://arxiv.org/abs/2503.04598.

A Contributions

The authors are listed in order of the significance of their contributions, with those in project leadership roles appearing last.

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
Xinyu Zhou

* Equal contribution

B Optimized Inference I/O for Full Attention Residuals

A naïve implementation of Full AttnRes scans all preceding layer outputs at every layer, so memory traffic scales linearly with depth. As noted in §4.2, however, the pseudo-query $\boldsymbol{w}_l$ is a learned parameter independent of both the input and the hidden state. We can therefore batch inter-block accesses across layers in a two-phase schedule, bringing total I/O well below the naïve bound.

Note that the block partition introduced below is purely an inference scheduling device. Unlike Block AttnRes, it leaves the model architecture unchanged and does not replace per-layer sources with block summaries; it simply makes the amortization argument concrete.

Setup Let the model have L layers and hidden dimension d, partitioned into N contiguous blocks of size $S = L/N$. Inference proceeds one block at a time: Phase 1 jointly computes inter-block attention for all S layers in the block against all preceding blocks, and Phase 2 walks through intra-block dependencies sequentially.

Phase 1: Batched Inter-block Attention

Consider block n with its S layers. The queries $\{\boldsymbol{w}_l\}_{l \in \mathcal{B}_n}$ are all known before execution begins, so the $(n{-}1)S$ preceding key–value pairs need only be read once from HBM and reused across all S queries. The read cost for block n is therefore

$$\text{Read}_{\text{inter}}^{(n)} = 2(n-1)Sd, \tag{11}$$

where the factor of 2 accounts for both keys and values. Summing over all N blocks and using $SN = L$:

$$\text{Read}_{\text{inter}} = \sum_{n=1}^{N} 2(n-1)Sd = 2Sd \cdot \frac{N(N-1)}{2} = dL(N-1). \tag{12}$$

Phase 1 also writes one d-dimensional output per layer, giving $\text{Write}_{\text{inter}}^{(n)} = Sd$ per block and

$$\text{Write}_{\text{inter}} = Ld \tag{13}$$

in total.

Phase 2: Sequential Intra-block Attention

Phase 1 covers all sources before the current block. Within the block, however, each layer depends on those before it, so these must be handled in order. Layer t ($1 \le t \le S$) reads $t{-}1$ intra-block key–value pairs at a cost of $2(t{-}1)d$. Summing over one block:

$$\text{Read}_{\text{intra}}^{(n)} = \sum_{t=1}^{S} 2(t-1)d = S(S-1)d. \tag{14}$$

Phase 2 also writes one output per layer, so $\text{Write}_{\text{intra}}^{(n)} = Sd$.

Total Amortized I/O per Layer

Summing both phases over all N blocks:

$$\text{Read}_{\text{total}} = dL(N-1) + N \cdot S(S-1)d, \qquad \text{Write}_{\text{total}} = 2Ld. \tag{15}$$

Dividing by L and using $SN = L$:

$$\text{Read per layer} = (N-1)d + (S-1)d = (S+N-2)d, \qquad \text{Write per layer} = 2d, \tag{16}$$

$$\boxed{\text{Total I/O per layer} = (S+N)\,d.} \tag{17}$$

Batching inter-block reads thus brings per-layer I/O from $\mathcal{O}(L)$ down to $\mathcal{O}(S{+}N)$. The schedule follows the same two-phase split as Block AttnRes: inter-block attention accounts for the bulk of the traffic, while sequential computation stays local within each block.