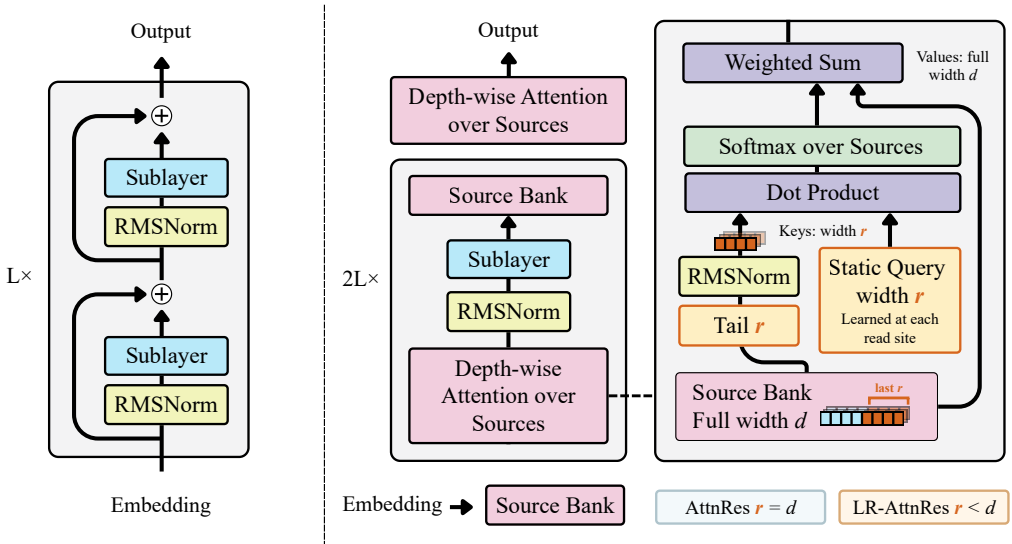
LOW-RANK ATTENTION RESIDUALS

Anonymous authors
Paper under double-blind review

ABSTRACT

Attention Residuals (ATTNRES) replace the fixed residual sum with depth-wise attention over previous sub-layer outputs in Large Language Models (LLMs), but use each output as both a full-dimensional key and value. This couples routing with representation and makes the cost of computing depth-routing scores scale with hidden width d. We propose Low-Rank Attention Residuals (LR-ATTNRES), which keep full-dimensional residual values while using r-dimensional keys, with $r < d$, for routing. LR-ATTNRES uses the last r dimensions of each value as the routing key, reducing total residual-side FLOPs while still improving performance. Comprehensive sweeps across the number of blocks (N) and r show that depth-wise routing can be effective with far fewer dimensions than the model width. At both 1B and 4B parameters with $r = d/4$, LR-ATTNRES achieves lower final validation loss, higher average downstream accuracy, and higher measured training-step throughput than standard ATTNRES. We also provide a fused kernel supporting standard and low-rank routing. We release all code, the kernel, and all trained models to facilitate future research.



(a) Standard Residuals (b) AttnRes and LR-AttnRes

Figure 1: Residual and routing paths. (a) Standard PreNorm residuals. (b) Shared architecture for ATTNRES ($r = d$) and LR-ATTNRES ($r < d$). In full routing, the source bank retains the embedding and appends each sub-layer output. Keys use the last r coordinates of each source before RMSNorm (values retain all d coordinates). Queries are learned, input-independent, and specific to each read site. Softmax is over sources. The L Transformer blocks contain $2L$ sub-layers, with an additional depth-wise attention read before the output head. During block routing, the current block's partial sum is included as an additional source. Once the block is complete, its sum is retained as a separate source in the source bank, while a new partial sum accumulates for the next block.

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

1 INTRODUCTION

Residual connections are central to the optimization of deep networks (He et al., 2016). This is especially true in Transformer-based large language models (Vaswani et al., 2017), where residual streams allow information to accumulate across depth and also serve as gradient highways for optimization. In a PreNorm decoder (Xiong et al., 2020), each attention or feed-forward sub-layer reads a normalized version of the current stream and writes an output back into it. This fixed additive rule is simple and stable, but it gives every previous output a fixed unit coefficient. The model can learn what each sub-layer writes, yet it cannot directly choose which previous writes should be emphasized when constructing the next hidden state. **Attention Residuals** (**ATTNRES**) address this limitation by replacing fixed accumulation with attention over depth (Kimi Team et al., 2026).

At each residual site, the hidden state is formed as a learned weighted combination of earlier sub-layer outputs. This is analogous to token attention, but the "tokens" are previous sub-layer outputs. The full version attends over all previous sub-layer outputs, while **BLOCK ATTNRES** groups several outputs into block summaries to reduce storage and communication.

The standard ATTNRES design makes each sub-layer output $\boldsymbol{o}_i \in \mathbb{R}^d$ serve two roles simultaneously: it is both the value mixed into the residual stream and, after normalization, the key used to determine how strongly that source should be selected. These roles need not require the same representation. A residual output should remain a full-capacity carrier of information for downstream layers, whereas a routing key only needs to distinguish among a small set of depth sources.

This disparity is striking at production scale. Kimi K3 ($d = 7168$) groups its 93 layers into 8 ATTNRES blocks, using full-width queries and keys to select among at most 9 sources per depth read, including the embedding (Kimi Team, 2026). Its token context extends to 10^6 positions, roughly five orders of magnitude more candidates. Although depth-wise and sequence-wise attention solve different problems, source cardinality alone offers little justification for full-width depth routing, and the observed query concentration suggests that narrower routing representations may suffice.

Across Kimi K3's 186 active ATTNRES queries, folding the RMSNorm affine weights into the query projections gives a mean Shannon coordinate effective dimension of 602.6, just 8.4% of their width. Moreover, 45.5% of folded-query coordinates are exactly zero under the released implementation's FP32 arithmetic, almost entirely because of underflow during folding. Appendix E gives the definition and layerwise results. These observations motivate separating routing-key width from value width.

We introduce **Low-Rank Attention Residuals** (**LR-ATTNRES**), which retain full-dimensional residual values while computing depth-routing scores with r-dimensional keys, where $r < d$. LR-ATTNRES uses the final r value dimensions as keys, yielding a net reduction in added FLOPs compared to ATTNRES. Section 4.8 gives conditions for converting ATTNRES checkpoints to full-width-RMS LR-ATTNRES with identical outputs in exact arithmetic, with potential applicability to Kimi K3. Figure 1 illustrates the construction. We also explore **Projected LR-AttnRes**, which generates exogenous routing keys through auxiliary projections. This alternative is discussed in Appendix C.

We evaluate 32 configurations at 0.5B parameters and width $d = 1024$, spanning full routing, block counts $N \in \{4, 8, 16\}$, and ranks $r \in \{16, 32, 64, 128, 256, 512, 768, 1024\}$. The best configuration uses $N = 8, r = 128 = d/8$. At 1B and 4B, LR-ATTNRES with $r = d/4$ achieves lower final validation loss than ATTNRES and increases average accuracy across eight downstream benchmarks by 0.40 and 0.57 percentage points, respectively.

Our contributions are:

- We introduce LR-ATTNRES, a depth-routing mechanism that retains full-dimensional values while reducing the routing-key width from d to $r < d$.
- We develop **FAST-ATTNRES**, a fused kernel supporting both standard ATTNRES and LR-ATTNRES (Appendix I).
- We characterize static-query expressivity under full-width normalization and local routing feedback under both normalization choices, and evaluate rank, grouping, seed replication, and normalization effects.

2 Related Work

A broad line of work modifies the residual pathway. DenseNet concatenates earlier outputs, while DenseFormer combines layer outputs with learned static depth-wise weights (Huang et al., 2017 and Pagliardini et al., 2024). Other methods introduce more expressive cross-layer access, including retrospective layer attention, learned augmented residual layers, multi-stream or hyper-connection mechanisms, and input-dependent depth mixing (Fang et al., 2023, Menghani et al., 2025, Zhu et al., 2025, Xie et al., 2025, and Xiao et al., 2025).

AttnRes is the most direct predecessor of our work (Kimi Team et al., 2026). In standard AttnRes, however, the same full-dimensional tensor is used both as the residual value and as the routing key. LR-AttnRes keeps the cross-layer routing framework but isolates a more specific design axis: the representation and dimensionality used to score depth sources. We show that low-dimensional routing keys can decrease validation loss while reducing FLOPs.

3 Method

3.1 Depth residual notation

Consider a decoder-only Transformer with L blocks and $M = 2L$ residual-writing sub-layers: one attention sub-layer and one feed-forward sub-layer per block. For clarity, equations omit batch and sequence dimensions. Let $\boldsymbol{o}_i \in \mathbb{R}^d$ denote the output written by sub-layer i, and let $\boldsymbol{o}_0 \in \mathbb{R}^d$ denote the token embedding. We index residual read sites by $t \in \{1, \ldots, M+1\}$, where site t reads from sources written before that site. A standard residual stream supplies the fixed sum

$$\boldsymbol{h}_t = \boldsymbol{o}_0 + \sum_{i=1}^{t-1} \boldsymbol{o}_i \tag{1}$$

to residual site t.

AttnRes replaces this fixed accumulation with learned attention over depth. At residual site t, the hidden state is formed as a weighted mixture over a source set $\mathcal{S}_t \subseteq \{0, \ldots, t-1\}$:

$$\alpha_{t,i} = \mathrm{softmax}_{i \in \mathcal{S}_t} \left(\boldsymbol{q}_t^\top \, \mathrm{RMSNorm}(\boldsymbol{o}_i) \right), \tag{2}$$

$$\boldsymbol{h}_t = \sum_{i \in \mathcal{S}_t} \alpha_{t,i} \boldsymbol{o}_i. \tag{3}$$

Here $\boldsymbol{q}_t \in \mathbb{R}^d$ is a learned pseudo-query associated with residual read site t. Thus, unlike a standard residual stream, the model can learn which previous sub-layer outputs to read from rather than always summing all previous outputs equally. The same tensor $\boldsymbol{o}_i$ is used as the value in Eq. 3 and, after normalization, as the key in Eq. 2.

Full AttnRes. The full version attends to every previous residual-writing source:

$$\mathcal{S}_t^{\mathrm{full}} = \{0, 1, \ldots, t-1\}. \tag{4}$$

Full AttnRes gives the maximum depth-routing flexibility, because every residual site can directly select from the embedding and all earlier attention and feed-forward outputs. The cost is that all previous full-dimensional outputs must be retained as attention sources.

Block AttnRes. Block AttnRes reduces the number of stored sources by grouping contiguous sub-layer outputs into block summaries. At a residual site t, the model attends to the embedding, all completed block summaries, and the current partial block if one exists. The partial block contains the outputs that have been written since the most recent completed block boundary but have not yet accumulated enough outputs to form a completed block.

Let B be a completed block containing $c = |B|$ sub-layer outputs. The block summary is the sum

$$\boldsymbol{b}_B^{\mathrm{sum}} = \sum_{i \in B} \boldsymbol{o}_i. \tag{5}$$

Now consider a residual site t that lies inside an unfinished block. Let $P_t \subseteq \{1, \ldots, t-1\}$ denote the current partial block: the set of residual-writing sub-layers after the last completed block boundary and before site t. If P_t is nonempty, its partial summary is

$$\boldsymbol{b}_{P_t}^{\text{sum}} = \sum_{i \in P_t} \boldsymbol{o}_i. \tag{6}$$

3.2 Low-rank routing with full-dimensional values

LR-AttnRes keeps the same full-dimensional value path as AttnRes but computes depth-routing scores in a smaller key space. Each source is represented as

$$\text{source } i: \qquad (\boldsymbol{o}_i, \boldsymbol{k}_i), \qquad \boldsymbol{o}_i \in \mathbb{R}^d, \quad \boldsymbol{k}_i \in \mathbb{R}^r, \quad r < d. \tag{7}$$

The value $\boldsymbol{o}_i$ remains the ordinary full-dimensional sub-layer output. The key $\boldsymbol{k}_i$ is an r-dimensional routing descriptor used only to select among depth sources. At residual site t, a learned static query $\boldsymbol{q}_t \in \mathbb{R}^r$ scores normalized keys:

$$\bar{\boldsymbol{k}}_i = \mathrm{RMSNorm}(\boldsymbol{k}_i), \tag{8}$$

$$\alpha_{t,i} = \mathrm{softmax}_{i \in \mathcal{S}_t} \left(\boldsymbol{q}_t^\top \bar{\boldsymbol{k}}_i \right), \tag{9}$$

$$\boldsymbol{h}_t = \sum_{i \in \mathcal{S}_t} \alpha_{t,i} \boldsymbol{o}_i. \tag{10}$$

This changes only the routing descriptor. The output mixed into the residual stream is still the full d-dimensional $\boldsymbol{o}_i$, so LR-AttnRes does not impose a low-rank bottleneck on residual values. LR-AttnRes obtains $\boldsymbol{k}_i$ by slicing the value, as described below.

3.3 Routing keys

LR-AttnRes uses the last r dimensions of the residual value as the key, without a learned key-projection path:

$$\boldsymbol{k}_i^{\mathrm{S}} = \mathrm{tail}_r(\boldsymbol{o}_i) = (\boldsymbol{o}_i)_{d-r+1:d} \in \mathbb{R}^r, \qquad \bar{\boldsymbol{k}}_i^{\mathrm{S}} = \mathrm{RMSNorm}(\boldsymbol{k}_i^{\mathrm{S}}). \tag{11}$$

The embedding source uses the same rule:

$$\boldsymbol{o}_0 = \mathrm{Embed}(x), \qquad \boldsymbol{k}_0^{\mathrm{S}} = \mathrm{tail}_r(\boldsymbol{o}_0), \qquad \bar{\boldsymbol{k}}_0^{\mathrm{S}} = \mathrm{RMSNorm}(\boldsymbol{k}_0^{\mathrm{S}}). \tag{12}$$

The last r dimensions are a convention: any fixed subset is equivalent up to a global permutation of hidden coordinates. Let P_r select these coordinates. For a sub-layer output $\boldsymbol{o}_i = W_i^O \boldsymbol{a}_i$ with projection input $\boldsymbol{a}_i$, the key $P_r \boldsymbol{o}_i = (P_r W_i^O)\boldsymbol{a}_i$ is learned through the selected rows of the value projection, without an auxiliary projection. It can be implemented as a view into the cached value. When $r = d$, LR-AttnRes reduces to AttnRes.

Partial-block routing. Each sub-layer output updates the full-width partial sum $\boldsymbol{b} \leftarrow \boldsymbol{b} + \boldsymbol{o}_i$. Its normalized key is $\mathrm{RMSNorm}(\mathrm{tail}_r(\boldsymbol{b}))$: accumulation precedes slicing and normalization. At a block boundary, the partial sum becomes a completed source and the partial accumulator resets to empty.

Figure 11 in Appendix H summarizes full and block routing for both AttnRes ($r = d$) and LR-AttnRes ($r < d$) in a shared implementation using PyTorch pseudocode.

3.4 Efficient inference compatibility

LR-AttnRes preserves the conditions required by the efficient inference strategy used by base AttnRes. The two-phase strategy for Block AttnRes relies on two properties. First, residual-site queries are input-independent. Second, the inter-block and intra-block attention results can be merged with an online-softmax update. LR-AttnRes preserves both properties. It changes only the key representation used for routing, while keeping the residual values in $\mathbb{R}^d$.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Architecture. The 0.5B sweep uses a 24-layer decoder-only Transformer with hidden dimension $d = 1024$, 16 attention heads, SwiGLU hidden width $m = 2816$, RoPE (Su et al., 2024) base $\theta = 500{,}000$, QK normalization (Henry et al., 2020), PreNorm, FlashAttention (Dao et al., 2022), untied input and output embeddings, and a vocabulary size of 100,277. The context length is 2048 tokens. Each Transformer block has one attention sub-layer and one MLP sub-layer, giving $2L = 48$ residual writers. Depth-wise pseudo-queries are initialized to zero and key normalization is enabled. At 1B and 4B, paired standard ATTNRES and LR-ATTNRES models use $N = 8$, matching Kimi K3 (Kimi Team, 2026). We fix $r/d = 1/4$ at both scales to test transfer without rank tuning: this reduces key width by 75% while retaining a larger fraction of coordinates than the best-performing 0.5B ratio, $r/d = 1/8$ (Section 4.2).

Training. The sweep models are trained from scratch on Ultra-FineWeb (Wang et al., 2025) for 10B tokens with document masking. All scales use a global batch size of 262,144 tokens. We optimize with Muon (Jordan et al., 2024) and Adam (Kingma & Ba, 2015). Muon uses learning rate 10^{-3}, momentum 0.95, and weight decay 0.1. Adam uses learning rate 3×10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.95$, and weight decay 0. We include a z-loss regularizer with coefficient 10^{-5}. Gradients are clipped to norm 1.0. We use a linear schedule with 2000 warmup steps and 20% warmdown steps. Final validation reports token-averaged cross-entropy on 100M held-out tokens. Two sweep configurations are each trained three times in total (Section 4.4). The 1B and 4B pairs use 20B training tokens.

4.2 ROUTING-KEY RANK AND VALIDATION LOSS

We evaluate all 32 combinations of eight ranks and four routing settings. Appendix A reports all 32 sweep configurations.

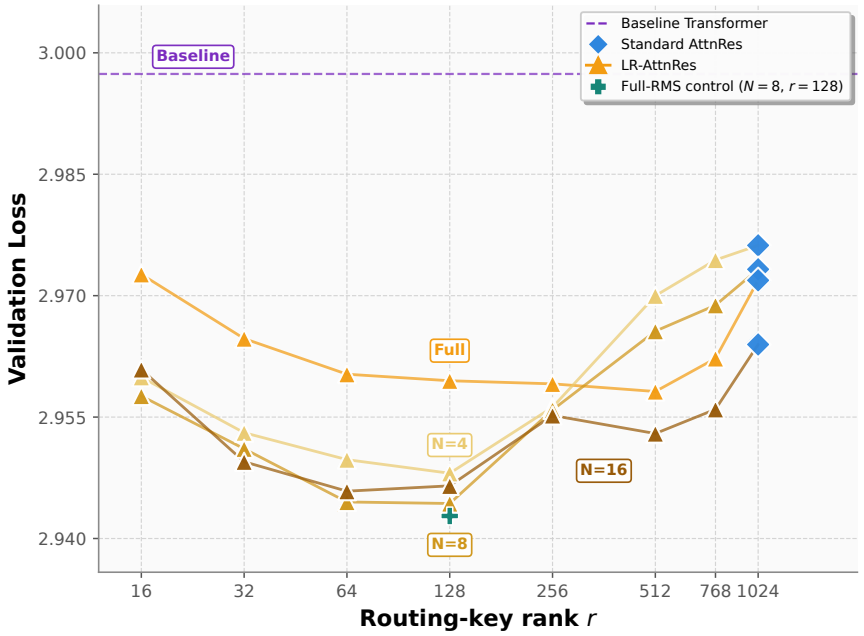


Figure 2: Validation loss versus routing-key rank r (base-2 log axis) at 0.5B. Triangles denote LR-ATTNRES. Blue diamonds denote standard ATTNRES. Only the $N = 8, r = 128$ triangle and $N = 8, r = d = 1024$ diamond plot means across three training seeds; all other sweep points show single runs. The teal plus is the full-width-RMS control. The purple dashed line is the baseline. Appendix A gives numerical results.

All 32 routed configurations improve on the baseline loss of 2.997. 27 of the 28 low-rank configurations improve on their $r = d$ endpoint (full routing at $r = 16$ is the exception). The lowest observed loss is 2.944 at $N = 8, r = 128$, improving by 0.03074 over standard $N = 8$ ATTNRES and by 0.02035 over the best standard setting, $N = 16$. At $N = 8, r = 64$, loss is only 8.978×10^{-4} higher. These minima are based on a single training run per configuration.

The preferred rank depends on source grouping. Full LR-ATTNRES improves from 2.973 at $r = 16$ to 2.958 at $r = 512$, then worsens to 2.962 at $r = 768$ and 2.972 at $r = d$. All three block settings instead attain their minima at $r = 64$ or 128. Neither the smallest nor the largest routing space is uniformly best.

The $N = 8, r = 128$ configuration also reduces added residual-side FLOPs by approximately 44% relative to standard $N = 8$ ATTNRES, from 0.1754% to 0.09865% of the non-embedding Transformer core. Appendix A presents the complete validation–FLOPs comparison, and Appendix F details the computational accounting.

4.3 BLOCK GRANULARITY

Block granularity and rank interact: $N = 8$ performs best at $r = 16, 64, 128$, and $N = 16$ at the other tested ranks. The minima for $N = 4, 8, 16$ are 2.948, 2.944, and 2.946. Increasing N expands the source set but does not consistently improve validation loss.

At $r = 256$, block losses span only 0.001016, a near-tie that could reflect training-seed variability. At $r = 512$, the spread grows to 0.01702: $N = 16$ improves while $N = 4, 8$ worsen.

The lack of a consistent benefit from larger N may reflect changes in the sources themselves. Changing N changes which outputs are summed together and assigned a shared routing weight. These outputs can reinforce or cancel before normalization, changing the routing keys as well as the number of sources (Appendix B.4). These results favor joint selection of N and r, rather than a universal U-shaped rank response or optimal block count.

4.4 SEED REPLICATION AND NORMALIZATION CONTROL

We replicate the selected $N = 8, r = 128$ configuration and standard $N = 8$ ATTNRES with two additional seeds. Figure 2 shows the three-seed means. Mean $\pm$ sample standard deviation is 2.973 ± 0.0009852 for ATTNRES and 2.944 ± 0.001777 for LR-ATTNRES. The mean paired reduction is 0.02893, with improvements in all three pairs. Per-seed losses are in Appendix A.

To test whether slice-based normalization is necessary for the gain, we train a full-width-RMS control. Let P_r select the final r coordinates of a value $\boldsymbol{o}_i \in \mathbb{R}^d$. The two constructions are

$$\begin{aligned} \bar{\boldsymbol{k}}_i^{\text{slice}} &= \text{RMSNorm}(P_r \boldsymbol{o}_i) = \frac{P_r \boldsymbol{o}_i}{\sqrt{\|P_r \boldsymbol{o}_i\|_2^2/r + \epsilon}}, \\ \bar{\boldsymbol{k}}_i^{\text{control}} &= P_r \, \text{RMSNorm}(\boldsymbol{o}_i) = \frac{P_r \boldsymbol{o}_i}{\sqrt{\|\boldsymbol{o}_i\|_2^2/d + \epsilon}}. \end{aligned} \tag{13}$$

The default method slices first, then applies RMSNorm to the r selected coordinates. The control applies RMSNorm to all d coordinates first, then slices. Both score sources with $\boldsymbol{q}_t^\top \bar{\boldsymbol{k}}_i$ and mix full-width values. The control reaches 2.943: 0.03157 below standard ATTNRES and 8.345×10^{-4} below slice-normalized LR-ATTNRES. Thus the gain persists with full-width normalization. Section 4.7 characterizes the resulting routing feedback, and Section 4.8 examines expressivity through frozen-checkpoint conversion.

4.5 RESULTS AT 1B AND 4B PARAMETERS

The 1B models use 24 layers, width $d = 1536$, 24 heads, and SwiGLU width 4224. The 4B models use 44 layers, $d = 2560$, 40 heads, and SwiGLU width 7040. Table 1 shows validation-loss reductions of 0.01946 and 0.03204, with eight-task mean accuracy gains of 0.40 and 0.57 percentage points. Appendix G gives the evaluation protocol and per-task scores.

Table 1: Results after 20B training tokens. Accuracy averages eight downstream tasks.

Model	Method	Avg. accuracy (%) ↑	Final val. loss ↓
1B	AttnRes	41.87	2.800
1B	LR-AttnRes	**42.27**	**2.780**
4B	AttnRes	42.97	2.738
4B	LR-AttnRes	**43.54**	**2.706**

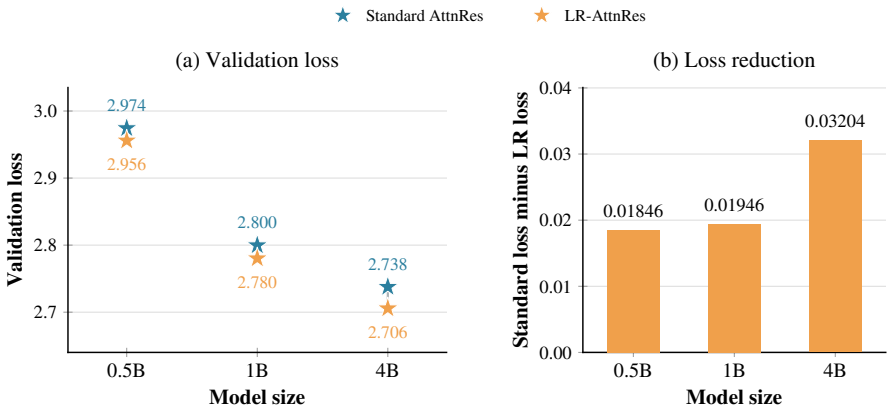


Figure 3: Validation loss across model sizes with block routing $N = 8$ and LR rank $r = d/4$ ($r = 256, 384, 640$). Left: final validation loss. Right: standard AttnRes minus LR-AttnRes loss. The 0.5B pair is trained for 10B tokens, and the 1B and 4B pairs for 20B tokens.

4.6 Effective number of sources

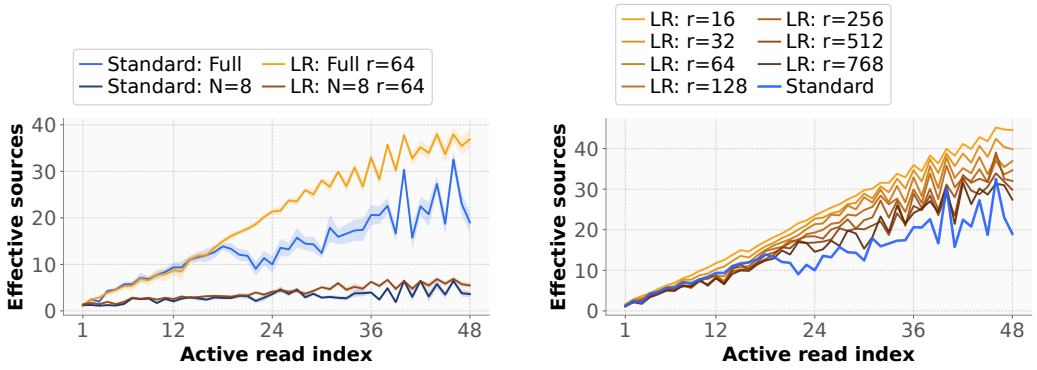


Figure 4: Effective number of residual sources across residual read sites. Left: comparison between full and block routing for standard AttnRes (Standard) and LR-AttnRes (LR). Right: rank sweep for full LR-AttnRes, with standard full AttnRes shown in blue. Lines show token means; shading in the left panel indicates ± 1 standard deviation across 1M held-out tokens. Lower values indicate sharper source selection.

We next analyze how many residual sources are effectively used by each depth-wise attention distribution. For a residual read site t with source set $\mathcal{S}_t$, let $\boldsymbol{\alpha}_t(x)$ denote the source-attention distribution for token position x, where x ranges over the batch and sequence positions used in the

probe. We define the effective number of sources as the entropy perplexity of this distribution:

$$N_{\text{eff}}(t,x) = \exp\left(H(\boldsymbol{\alpha}_t(x))\right), \qquad H(\boldsymbol{\alpha}_t(x)) = -\sum_{i\in\mathcal{S}_t} \alpha_{t,i}(x)\log\alpha_{t,i}(x). \tag{14}$$

This equals 1 for selection of a single source and k for uniform attention over k sources. We average this quantity over 1M held-out tokens at each of 48 query-bearing reads. Diagnostic figures number these active reads 1–48, corresponding to $t = 2, \ldots, 49$ in Section 3.1. At the final read, replacing standard keys with rank-64 keys raises the effective source count from 19.02 to 36.90 under full routing and from 3.62 to 5.47 under $N = 8$ block routing. Within block settings, low-rank keys produce less concentrated attention distributions, spreading weight across more residual sources.

Broader routing is not itself a predictor of lower loss. In the full-routing rank sweep, the final-site effective count decreases from 44.57 at $r = 16$ to 29.85 at $r = 512$, even as validation loss improves from 2.973 to 2.958. At $r = 768$, the count decreases further to 27.42.

4.7 Routing feedback

Slicing changes how a source influences both its contribution and its selection. For one read, write z_i for its source-local score, $\alpha_i = \mathrm{softmax}_i(z)$, and $\boldsymbol{g} = \nabla_{\boldsymbol{h}}\mathcal{L}$. Treating the source slots as independent inputs, the gradient through this read decomposes as

$$\nabla_{\boldsymbol{o}_i}\mathcal{L} = \underbrace{\alpha_i \boldsymbol{g}}_{\text{direct value gradient}} + \underbrace{\alpha_i\langle \boldsymbol{g}, \boldsymbol{o}_i - \boldsymbol{h}\rangle \nabla_{\boldsymbol{o}_i} z_i}_{\text{routing feedback}}. \tag{15}$$

For finite logits, the local source-to-output Jacobian has rank at least $d-1$, irrespective of r, so narrow scoring does not impose a rank-r bottleneck on local value transmission. Let $\Pi_r = P_r^\top P_r$ and $\rho_m(\boldsymbol{u}) = \sqrt{\|\boldsymbol{u}\|_2^2/m + \epsilon}$. The two normalization choices in Eq. 13 give

$$\begin{aligned}(I-\Pi_r)\nabla_{\boldsymbol{o}_i} z_i^{\text{slice}} &= 0,\\ (I-\Pi_r)\nabla_{\boldsymbol{o}_i} z_i^{\text{control}} &= -\frac{z_i^{\text{control}}}{d\,\rho_d(\boldsymbol{o}_i)^2}(I-\Pi_r)\boldsymbol{o}_i.\end{aligned} \tag{16}$$

Thus unselected coordinates receive only the direct value contribution under slice normalization. A perturbation $\boldsymbol{u}$ with $P_r\boldsymbol{u} = 0$ leaves the routing weights unchanged and changes the mixed output by $\alpha_i\boldsymbol{u}$. The control retains dependence on the norm of these coordinates, so complete local isolation is not necessary for the gain observed in Section 4.4. These properties only concern the current read. This local separation does not establish that routing feedback harms learning. Appendix B.3 gives the derivation. Gradient-norm measurements are reported in Appendices C.5 and G.1.

4.8 Static-query span and normalization

Input-independent queries give depth-wise routing a useful structural property: all query vectors lie in a fixed subspace. Let $Q \in \mathbb{R}^{d\times T}$ contain the query vectors at the T active standard AttnRes read sites, where d is the residual width. Any coordinate-wise scaling from the routing RMSNorm is absorbed into these vectors. The rank of Q counts the independent directions needed to represent all queries. Each routing score depends only on the projection of the normalized key onto the subspace spanned by these directions.

This observation gives an exact conversion from standard AttnRes to LR-AttnRes with full-width RMS normalization under two assumptions. First, keys are normalized using all d residual coordinates before selecting the r routing coordinates. Second, the underlying Transformer allows its weights to be reparameterized consistently under an orthogonal change of residual coordinates. Such a change preserves vector lengths and hence full-width RMS normalization (Ashkboos et al., 2024a). We can therefore align the query subspace with r retained coordinates whenever $r \geq \mathrm{rank}(Q)$, preserving every routing distribution and model output while keeping residual values at width d. Our 24-layer model has $T = 48$, so there are at most 48 independent query directions and $r \geq 48$ suffices regardless of source grouping. Appendix B states the precise assumptions and gives the proof.

We convert six AttnRes checkpoints to LR-AttnRes without retraining (Table 2). The largest FP64 logit difference is 8.251×10^{-13}. Paired BF16 evaluations over 100M tokens yield cross-entropy (CE) increases ranging from 6.086×10^{-5} to 8.865×10^{-5}. Appendix B.2 details verification.

The basis change is absorbed into existing weights, adding no inference-time rotations or projections. An implementation exploiting the reduced query width lowers score dot products from Sd to Sr multiply-adds per read over S sources. Thus conversion permits lower routing FLOPs without additional projection FLOPs.

This conversion could also be applied to large models such as Kimi K3: its 186 active folded queries have joint rank at most 186 despite value width $d = 7168$ (Appendix E). A whole-model conversion would therefore permit scoring width $r = 186$ with mathematically identical routing distributions and model outputs. We have not converted Kimi K3 or measured its inference speed due to computational and budget constraints.

Table 2: Conversion without retraining. FP64 logit errors use one 2048-token sequence. Paired CE uses 100M tokens with BF16 model computation, FP32 per-token cross-entropy, and FP64 summation. ΔCE is converted minus original before rounding.

Model	Routing	$d \to r$	FP64 max. error ($\times 10^{-13}$)	Original CE	Converted CE	ΔCE ($\times 10^{-5}$)
0.5B	Full	$1024 \to 64$	7.407	2.972	2.972	+6.086
0.5B	$N = 4$	$1024 \to 64$	5.853	2.976	2.976	+8.865
0.5B	$N = 8$	$1024 \to 64$	3.852	2.974	2.974	+7.768
0.5B	$N = 16$	$1024 \to 64$	3.300	2.964	2.964	+7.908
1B	$N = 8$	$1536 \to 64$	4.126	2.800	2.800	+6.398
4B	$N = 8$	$2560 \to 128$	8.251	2.738	2.738	+6.998

4.9 Training throughput

Table 3 reports complete training-step throughput over five paired repeats. Flash Linear Attention (FLA, Yang & Zhang, 2024) is an open-source GPU-kernel library. Its fused standard AttnRes kernel is the fastest baseline we tested. Reducing rank from d to $d/4$ with $N = 8$ under Fast-AttnRes gives mean paired gains of 0.59–1.20%. The table reports their 95% confidence intervals. Combined kernel and rank gains over FLA are 1.55% and 1.54% at 1B and 4B on H100, and 2.87% and 2.97% on B200. Appendix I gives the protocol, uncertainty calculation, and correctness checks.

Table 3: Complete training-step throughput (tokens/s, mean $\pm$ sample SD, five repeats). Rank gains compare $r = d/4$ with $r = d$ under Fast-AttnRes (paired mean and 95% CI).

GPU	Model	FLA $r = d$	Fast $r = d$	Fast $r = d/4$	Rank gain (%)	95% CI (%)
H100	1B	$47{,}436 \pm 34$	$47{,}887 \pm 48$	$48{,}171 \pm 179$	0.594	$[0.030, 1.157]$
H100	4B	$15{,}097 \pm 14$	$15{,}208 \pm 17$	$15{,}329 \pm 17$	0.795	$[0.552, 1.038]$
B200	1B	$66{,}489 \pm 17$	$67{,}933 \pm 358$	$68{,}400 \pm 309$	0.688	$[0.591, 0.786]$
B200	4B	$22{,}856 \pm 8$	$23{,}256 \pm 36$	$23{,}536 \pm 15$	1.203	$[1.060, 1.345]$

5 Conclusion

We introduced LR-AttnRes, which computes depth-routing scores from a low-dimensional coordinate slice while preserving full-width values, without auxiliary key projections. In the 32-setting 0.5B sweep, 27 of 28 low-rank configurations improve on their matched standard endpoint while reducing the counted routing FLOPs. The $N = 8, r = 128$ configuration improves loss across three seeds, and the gain persists in a full-width-RMS control. Our analysis establishes a sufficient width for preserving standard expressivity under full-width normalization, supported by numerical checks on six converted checkpoints, and characterizes the local routing feedback under both normalization choices. At 1B and 4B, $r = d/4$ yields lower validation loss and higher average accuracy across eight benchmarks. We also provide Fast-AttnRes, a fused implementation of standard and low-rank routing. These findings support selecting routing-key width independently of value width.

AI USE STATEMENT

In this work, we used generative AI tools for implementing methods, providing feedback on experimental methodology, developing theoretical arguments, and interpreting results. We have not used generative AI tools for originating research ideas, generating synthetic data sets, translation, or directly cleaning and reformatting data sets, and qualitative and thematic analyses are not applicable to this work. Additionally, we used generative AI tools for creating and editing software code, preparing scientific figures, drafting and editing the manuscript, and identifying relevant literature. All AI-assisted code, analyses, figures, and text underwent human review. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the complete LR-ATTNRES and FAST-ATTNRES codebases as anonymized supplementary material. Section 3 and Appendix H describe the routing mechanisms and provide implementation pseudocode, while Sections 4.1 and 4.5 specify the model architectures and training setup. Appendix B states the assumptions underlying our theoretical results and provides proofs and numerical verification details. Appendix F describes the FLOPs, memory, and parameter accounting, and Appendix G specifies the downstream evaluation protocol. Appendix I documents the FAST-ATTNRES implementation, numerical correctness checks, and hardware throughput measurement procedure.

REFERENCES

Saleh Ashkboos, Maximilian L. Croci, Marcelo Gennari Do Nascimento, Torsten Hoefler, and James Hensman. SliceGPT: Compress large language models by deleting rows and columns. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024a. URL https://openreview.net/forum?id=vXxardq6db.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. QuaRot: Outlier-free 4-bit inference in rotated LLMs. In Amir Globerson, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024b. URL http://papers.nips.cc/paper_files/paper/2024/hash/b5b939436789f76f08b9d0da5e81af7c-Abstract-Conference.html.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html.

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, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=pvgEL1yS3Ql.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90.

Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for transformers. In Trevor Cohn, Yulan He, and Yang Liu (eds.), *Findings of the*

Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020, Findings of ACL, pp. 4246–4253. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.FINDINGS-EMNLP.379. URL https://doi.org/10.18653/v1/2020.findings-emnlp.379.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 2261–2269. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.243. URL https://doi.org/10.1109/CVPR.2017.243.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Kimi Team. Kimi K3: open frontier intelligence. *CoRR*, abs/2607.24653, 2026. doi: 10.48550/ARXIV.2607.24653. URL https://doi.org/10.48550/arXiv.2607.24653.

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. URL https://arxiv.org/abs/2603.15031.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

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.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/menghani25a.html.

Matteo Pagliardini, Amirkeivan Mohtashami, François Fleuret, and Martin Jaggi. DenseFormer: Enhancing information flow in transformers via depth weighted averaging. In Amir Globerson, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/f67449c7ab72f441d3a713b046c6818c-Abstract-Conference.html.

Jianlin Su, Murtadha H. M. Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. doi: 10.1016/J.NEUCOM.2023.127063. URL https://doi.org/10.1016/j.neucom.2023.127063.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Yudong Wang, Zixuan Fu, Jie Cai, Peijun Tang, Hongya Lyu, Yewei Fang, Zhi Zheng, Jie Zhou, Guoyang Zeng, Chaojun Xiao, Xu Han, and Zhiyuan Liu. Ultra-FineWeb: Efficient data filtering and verification for high-quality LLM training data. *CoRR*, abs/2505.05427, 2025. doi: 10.48550/ARXIV.2505.05427. URL https://doi.org/10.48550/arXiv.2505.05427.

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.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/xiao25d.html.

Zhenda Xie, Yixuan Wei, Huanqi Cao, Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Jiang Chang, 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. *CoRR*, abs/2512.24880, 2025. doi: 10.48550/ARXIV.2512.24880. URL https://doi.org/10.48550/arXiv.2512.24880.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

Songlin Yang and Yu Zhang. FLA: A Triton-based library for hardware-efficient implementations of linear attention mechanism, January 2024. URL https://github.com/fla-org/flash-linear-attention.

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-connections. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=9FqARW7dwB.

A VALIDATION–FLOPS

Table 4 presents validation losses and added routing FLOPs across full routing and block settings $N \in \{4, 8, 16\}$ at eight ranks $r \in \{16, 32, 64, 128, 256, 512, 768, 1024\}$. At $r = d = 1024$, each setting is standard ATTNRES. All runs use a 100M-token held-out validation set. Table 5 includes baseline and projected variants.

Table 4: Validation losses and added routing FLOPs for all 32 configurations in Figure 2. Added FLOPs are reported as a percentage of the non-embedding Transformer core. Bold marks the lowest loss.

	Full		Block $N = 4$		Block $N = 8$		Block $N = 16$	
Rank r	FLOPs (%)	Val. loss	FLOPs (%)	Val. loss	FLOPs (%)	Val. loss	FLOPs (%)	Val. loss
16	0.4129	2.973	0.05668	2.960	0.08906	2.958	0.1538	2.961
32	0.4193	2.965	0.05755	2.953	0.09043	2.951	0.1562	2.949
64	0.4320	2.960	0.05929	2.950	0.09317	2.945	0.1609	2.946
128	0.4574	2.959	0.06278	2.948	0.09865	**2.944**	0.1704	2.947
256	0.5082	2.959	0.06975	2.956	0.1096	2.956	0.1893	2.955
512	0.6099	2.958	0.08371	2.970	0.1315	2.966	0.2272	2.953
768	0.7115	2.962	0.09766	2.974	0.1535	2.969	0.2651	2.956
1024	0.8131	2.972	0.1116	2.976	0.1754	2.974	0.3029	2.964

Seed repeats and normalization control. Across the three paired runs with different seeds, standard $N = 8$ ATTNRES ($r = 1024$) has validation losses $(2.974, 2.972, 2.973)$, and slice-normalized $N = 8, r = 128$ LR-ATTNRES has losses $(2.944, 2.946, 2.943)$. All runs use the 0.5B architecture, 10B training tokens, and the same 100M-token validation set. The full-width-RMS control reaches 2.943. The sweep tables retain single-run sweep values.

B STATIC-QUERY EXPRESSIVITY AND NORMALIZATION

B.1 A SUFFICIENT WIDTH FROM THE JOINT QUERY SPAN

The following argument applies the orthogonal computational invariance of RMSNorm Transformers established by SliceGPT (Ashkboos et al., 2024a), also used in QuaRot (Ashkboos et al., 2024b), to static depth-routing queries. It concerns full-width RMS normalization, as in the control in Eq. 13, while retaining all d value coordinates.

Affine normalization. Routing RMSNorm gains can be folded into the static query: $\boldsymbol{q}_t = \boldsymbol{\gamma}_t \odot \mathbf{w}_t$ gives

$$\mathbf{w}_t^\top \left(\boldsymbol{\gamma}_t \odot \frac{\boldsymbol{o}}{\rho_d(\boldsymbol{o})} \right) = \frac{\boldsymbol{q}_t^\top \boldsymbol{o}}{\rho_d(\boldsymbol{o})}, \qquad \rho_d(\boldsymbol{o}) = \sqrt{\|\boldsymbol{o}\|_2^2/d + \epsilon}. \tag{17}$$

Likewise, a backbone RMSNorm gain followed by a linear map W can be absorbed into $\widehat{W} = W \operatorname{diag}(\boldsymbol{\gamma})$, leaving scalar RMS normalization. This reduction requires that every consumer admit the absorption without violating parameter-sharing constraints. The proposition applies after these absorptions. Q contains the effective routing queries.

Proposition 1 (Static-query span). Consider a deterministic ATTNRES decoder with scalar residual RMSNorm, unrestricted residual-facing linear projections, and T input-independent routing queries. Assume that its embeddings, vocabulary head, and remaining residual-coordinate operations admit the same orthogonal reparameterization. Let $Q = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_T] \in \mathbb{R}^{d \times T}$ and $k = \operatorname{rank}(Q)$. For every $1 \le r \le d$ with $r \ge k$, there is a consistent orthogonal reparameterization whose queries use only the final r coordinates and whose routing distributions and vocabulary logits equal those of the original model for every input in exact arithmetic. The routing denominator remains full-width. The result holds for both full routing and the block sums defined in Section 3.

Proof. Let $P_r \in \mathbb{R}^{r \times d}$ select the final r coordinates. Choose an orthogonal $R \in \mathbb{R}^{d \times d}$ that maps the column span of Q into this coordinate subspace, and set $\boldsymbol{q}'_t = P_r R \boldsymbol{q}_t$ and $\boldsymbol{o}'_i = R\boldsymbol{o}_i$. With $\rho_d(\boldsymbol{o}) = \sqrt{\|\boldsymbol{o}\|_2^2/d + \epsilon}$, orthogonality gives

$$\rho_d(R\boldsymbol{o}) = \rho_d(\boldsymbol{o}), \qquad \mathrm{RMSNorm}(R\boldsymbol{o}) = R\,\mathrm{RMSNorm}(\boldsymbol{o}). \tag{18}$$

Since $P_r^\top P_r R \boldsymbol{q}_t = R\boldsymbol{q}_t$, every routing score is preserved:

$$\frac{(\boldsymbol{q}'_t)^\top P_r \boldsymbol{o}'_i}{\rho_d(\boldsymbol{o}'_i)} = \frac{(R\boldsymbol{q}_t)^\top P_r^\top P_r R \boldsymbol{o}_i}{\rho_d(\boldsymbol{o}_i)} = \frac{\boldsymbol{q}_t^\top \boldsymbol{o}_i}{\rho_d(\boldsymbol{o}_i)}. \tag{19}$$

Thus the softmax weights are unchanged and the mixed value transforms as $\boldsymbol{h}'_t = R\boldsymbol{h}_t$. Block accumulation also commutes with R, since $\sum_{i \in B} R\boldsymbol{o}_i = R \sum_{i \in B} \boldsymbol{o}_i$.

For column-vector activations, transform each residual-facing input projection as $W'_{\mathrm{in}} = W_{\mathrm{in}} R^\top$ and each output projection as $W'_{\mathrm{out}} = R W_{\mathrm{out}}$. Rotate embeddings and any residual-output biases by R, and transform the vocabulary head as $W'_{\mathrm{head}} = W_{\mathrm{head}} R^\top$. Equation 18 then preserves the inputs to internal token-attention and SwiGLU computations, including QK normalization and RoPE. Their outputs rotate back into the new residual basis. Induction over residual reads therefore preserves every routing distribution and final vocabulary logit. All transformations are absorbed into existing parameters. □

For a fixed checkpoint, its joint query rank k suffices. Across the stated architecture, $k \le \min(T, d)$ gives a uniform sufficient width. Conversely, a full-width query can reproduce any reduced query by zero-padding it, so the full-width-RMS function classes coincide for $r \ge \min(T, d)$. Our 24-layer decoder has $T = 48$ active queries. The embedding-only read needs none. This bound counts queries, not candidate sources, and does not predict the validation-optimal rank. It also retains full-width normalization cost. Kimi K3's 186 active folded queries similarly have joint rank at most 186. Applying the whole-model construction additionally requires the stated backbone assumptions.

Minimal shared score projection. For fixed queries and arbitrary source vectors, k is also necessary for an exact shared linear score projection retaining the full-width denominator. If $U \in \mathbb{R}^{d \times r}$ and

reduced queries $\boldsymbol{a}_t$ satisfy $\boldsymbol{a}_t^\top U^\top \boldsymbol{o}/\rho_d(\boldsymbol{o}) = \boldsymbol{q}_t^\top \boldsymbol{o}/\rho_d(\boldsymbol{o})$ for every $\boldsymbol{o}$, then $U\boldsymbol{a}_t = \boldsymbol{q}_t$. Hence $\mathrm{col}(Q) \subseteq \mathrm{col}(U)$ and $r \geq k$. An orthonormal basis of $\mathrm{col}(Q)$ attains this bound. This characterizes exact score projection on arbitrary sources, not the minimum width of a retrained network with different representations.

B.2 Numerical verification

The joint query spans have dimension 48 for the 0.5B and 1B checkpoints and 88 for 4B. A shared orthogonal basis change places these spans within the final $r = 64$ or 128 coordinates, respectively, while retaining full-width values and the full-width RMS denominator. We verify FP64 logits on one complete 2048-token sequence per model, then export and strictly reload the BF16 checkpoints. Their native routing implementations match the reference on this sequence. Paired original and converted evaluations use the same 100M validation tokens, with FP32 per-token cross-entropy and FP64 summation. Table 2 reports numerical fidelity without additional training.

B.3 Local value and routing gradients

Fix one read with $\boldsymbol{h} = \sum_i \alpha_i \boldsymbol{o}_i$, $\boldsymbol{\alpha} = \mathrm{softmax}(\boldsymbol{z})$, and source-local scores $z_i = z_i(\boldsymbol{o}_i)$ with the static query held fixed. All derivatives below are contributions through this read: source slots are treated as independent graph inputs. If a tensor also participates in other reads or in constructing later sources, its total gradient includes those additional paths.

Lemma 2 (Value and routing contributions). For upstream gradient $\boldsymbol{g} = \nabla_{\boldsymbol{h}}\mathcal{L}$, Eq. 15 holds. Indeed, the softmax Jacobian gives

$$\frac{\partial \boldsymbol{h}}{\partial z_i} = \alpha_i(\boldsymbol{o}_i - \boldsymbol{h}), \qquad \mathrm{d}\boldsymbol{h} = \alpha_i\,\mathrm{d}\boldsymbol{o}_i + \alpha_i(\boldsymbol{o}_i - \boldsymbol{h})\,\mathrm{d}z_i. \tag{20}$$

Taking the inner product with $\boldsymbol{g}$ yields the decomposition. Equivalently, the local value Jacobian is $\partial \boldsymbol{h}/\partial \boldsymbol{o}_i = \alpha_i I + \alpha_i(\boldsymbol{o}_i - \boldsymbol{h})(\nabla_{\boldsymbol{o}_i} z_i)^\top$. □

Corollary 2.1 (Slice-normalized routing). Let $\boldsymbol{k}_i = P_r \boldsymbol{o}_i$, $\rho_i = \rho_r(\boldsymbol{k}_i)$, and $z_i^{\mathrm{slice}} = \boldsymbol{q}^\top \boldsymbol{k}_i/\rho_i$. Direct differentiation gives

$$\nabla_{\boldsymbol{o}_i} z_i^{\mathrm{slice}} = P_r^\top \left[\frac{\boldsymbol{q}}{\rho_i} - \frac{(\boldsymbol{q}^\top \boldsymbol{k}_i)\boldsymbol{k}_i}{r\rho_i^3}\right]. \tag{21}$$

Its support lies in the selected coordinates. Thus $(I - \Pi_r)\nabla_{\boldsymbol{o}_i}\mathcal{L} = \alpha_i(I - \Pi_r)\boldsymbol{g}$. More strongly, any finite perturbation $\boldsymbol{u} \in \ker P_r$ of source i leaves all scores and weights at this read unchanged, while the output becomes $\boldsymbol{h} + \alpha_i \boldsymbol{u}$. This is local invariance to the unselected value coordinates. It does not prevent later layers from mixing them with selected coordinates.

Corollary 2.2 (Full-width-RMS routing). For $z_i^{\mathrm{control}} = \boldsymbol{q}^\top P_r \boldsymbol{o}_i/\rho_d(\boldsymbol{o}_i)$,

$$\nabla_{\boldsymbol{o}_i} z_i^{\mathrm{control}} = \frac{P_r^\top \boldsymbol{q}}{\rho_d(\boldsymbol{o}_i)} - \frac{z_i^{\mathrm{control}} \boldsymbol{o}_i}{d\,\rho_d(\boldsymbol{o}_i)^2}. \tag{22}$$

Projection onto the unselected coordinates gives Eq. 16. Their contribution to the score is through their norm. The corresponding derivative is parallel to $(I - \Pi_r)\boldsymbol{o}_i$. The control therefore preserves a radial dependence outside the slice.

Corollary 2.3 (Local value transmission). For finite logits, $\alpha_i > 0$. By Lemma 2, the local source-to-output Jacobian satisfies

$$\mathrm{rank}\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{o}_i}\right) = \mathrm{rank}\big[I_d + (\boldsymbol{o}_i - \boldsymbol{h})(\nabla_{\boldsymbol{o}_i} z_i)^\top\big] \geq d - 1, \tag{23}$$

since the bracket is the identity plus a matrix of rank at most one. Thus narrow scoring does not impose a rank-r bottleneck on local value transmission under either normalization. This statement treats source slots independently at one read. It does not guarantee good conditioning or a full-rank end-to-end Jacobian.

These identities establish gradient support, not a reduction in feedback magnitude or harmful interference. The inner product between the direct and routing contributions is $\alpha_i^2\langle \boldsymbol{g}, \boldsymbol{o}_i - \boldsymbol{h}\rangle\langle \boldsymbol{g}, \nabla_{\boldsymbol{o}_i} z_i\rangle$, which can have either sign. Standard full-width routing also has structured derivatives: $\nabla_{\boldsymbol{o}_i} z_{t,i}$ lies in $\mathrm{span}(Q, \boldsymbol{o}_i)$. Slicing fixes the coordinate support of the numerator's dependence, and slice normalization additionally removes the outside-slice norm dependence. Neither property alone determines validation loss.

B.4 Block aggregation and routing geometry

At a fixed read, a completed block is exposed only through $\boldsymbol{b}_B = \sum_{j\in B} \boldsymbol{o}_j$. Perturbations whose sum within that block is zero leave its value and routing score unchanged. Every constituent write has the same explicit mixture coefficient α_B and receives the same adjoint through the block-sum node. These statements hold for that read with the other source slots fixed. They do not identify the complete gradients of the original writes across the network.

The normalization of a block depends on cross terms between its writes:

$$\|P_r \boldsymbol{b}_B\|_2^2 = \sum_{j,k\in B} (P_r \boldsymbol{o}_j)^\top (P_r \boldsymbol{o}_k), \qquad \|\boldsymbol{b}_B\|_2^2 = \sum_{j,k\in B} \boldsymbol{o}_j^\top \boldsymbol{o}_k. \tag{24}$$

Alignment and cancellation can differ between the selected coordinates and the full value space. Grouping therefore changes both the distinctions available to the router and the selected-to-full RMS ratio. The number of active queries, and hence the uniform sufficient-width bound in Proposition 1, is unchanged by grouping. Fewer candidate sources alone do not imply a smaller necessary query span or predict the best rank of a model trained from scratch.

C Projected Low-Rank Attention Residuals

Projected LR-AttnRes explores exogenous routing keys generated by auxiliary output projections. Here, exogenous means separate from the residual-value coordinates. The keys still depend on the same sub-layer activations. This provides an interesting test of dedicated routing capacity, but its additional projections, parameters, and key storage make it a less practical choice for LLM training than LR-AttnRes.

C.1 Projected low-rank keys

Projected LR-AttnRes uses a learned low-rank key emitted from the same output projection that produces the residual value. Let $\boldsymbol{a}_i \in \mathbb{R}^{m_i}$ be the activation passed to the output projection of residual-writing sub-layer i. For an attention sub-layer, this is the activation passed to the attention output projection, so typically $m_i = d$. For a SwiGLU feed-forward sub-layer, this is the activation passed to the second/output projection of the FFN, so m_i is the SwiGLU hidden width. If the ordinary output projection is $W_i^O \in \mathbb{R}^{d\times m_i}$, the standard residual contribution is

$$\boldsymbol{o}_i = W_i^O \boldsymbol{a}_i. \tag{25}$$

Projected LR-AttnRes adds a low-rank key projection and computes the output and key together:

$$\begin{bmatrix} \boldsymbol{o}_i \\ \boldsymbol{k}_i^{\mathrm{P}} \end{bmatrix} = \begin{bmatrix} W_i^O \\ W_i^K \end{bmatrix} \boldsymbol{a}_i, \qquad W_i^O \in \mathbb{R}^{d\times m_i}, \qquad W_i^K \in \mathbb{R}^{r\times m_i}, \qquad \boldsymbol{o}_i \in \mathbb{R}^d, \qquad \boldsymbol{k}_i^{\mathrm{P}} \in \mathbb{R}^r. \tag{26}$$

In the attention sub-layer, the fused matrix augments the attention output projection. In the SwiGLU feed-forward sub-layer, it augments the second/output projection that maps the gated hidden activation back to the model dimension.

The embedding source has no sub-layer output projection. The default Projected LR-AttnRes embedding key is therefore produced by a small low-rank projection:

$$\boldsymbol{o}_0 = \mathrm{Embed}(x), \qquad \boldsymbol{k}_0^{\mathrm{P}} = W_0^K \boldsymbol{o}_0, \qquad \bar{\boldsymbol{k}}_0^{\mathrm{P}} = \mathrm{RMSNorm}(\boldsymbol{k}_0^{\mathrm{P}}), \qquad W_0^K \in \mathbb{R}^{r\times d}. \tag{27}$$

Projected LR-AttnRes is an exploratory alternative to the primary method. Because the key has its own learned projection, the model can use the full value vector $\boldsymbol{o}_i$ for residual-stream content while using a separate representation for routing. Its best block configuration achieves a validation loss of 2.944. Its drawback is that the auxiliary key projections add projection FLOPs and parameters. The detailed accounting is given in Appendix F.

C.2 Inference compatibility and practical costs

Projected LR-AttnRes preserves the same input-independent queries and online-softmax merge as LR-AttnRes, so it also satisfies the mathematical conditions for the predecessor's two-phase inference strategy. Its practical costs differ: the auxiliary projections add parameters and computation, and the retained source representation includes separate keys. Under a pipeline layout that forwards each key alongside its value, the payload per source is $d + r$ rather than d elements, an increase of 3.125% for $d = 1024, r = 32$. Appendix F gives the full cost accounting.

C.3 Complete validation–FLOPs comparison

Table 5: Validation loss and routing cost for the baseline, standard AttnRes, and all low-rank configurations. Bold marks the lowest loss within each method. Added FLOPs are reported as a percentage of the non-embedding Transformer core. For Projected LR-AttnRes, this column includes the auxiliary learned key-projection path. For AttnRes and LR-AttnRes, it is the depth-wise residual kernel cost because no projected-key path is present. The default validation split contains 100M held-out tokens.

Method	Block count N	Rank r	Added FLOPs (%)	Val. loss
Baseline Transformer	–	–	0	2.997
Full AttnRes	Full	1024	0.8131	2.972
Block AttnRes	4	1024	0.1116	2.976
Block AttnRes	8	1024	0.1754	2.974
Block AttnRes	16	1024	0.3029	**2.964**
Full Projected LR-AttnRes	Full	16	0.8966	2.957
Full Projected LR-AttnRes	Full	32	1.387	2.950
Full Projected LR-AttnRes	Full	64	2.367	2.947
Block Projected LR-AttnRes	4	32	1.025	2.945
Block Projected LR-AttnRes	8	32	1.058	**2.944**
Block Projected LR-AttnRes	16	32	1.123	2.951
Full LR-AttnRes	Full	16	0.4129	2.973
Full LR-AttnRes	Full	32	0.4193	2.965
Full LR-AttnRes	Full	64	0.4320	2.960
Full LR-AttnRes	Full	128	0.4574	2.959
Full LR-AttnRes	Full	256	0.5082	2.959
Full LR-AttnRes	Full	512	0.6099	2.958
Full LR-AttnRes	Full	768	0.7115	2.962
Block LR-AttnRes	4	16	0.05668	2.960
Block LR-AttnRes	4	32	0.05755	2.953
Block LR-AttnRes	4	64	0.05929	2.950
Block LR-AttnRes	4	128	0.06278	2.948
Block LR-AttnRes	4	256	0.06975	2.956
Block LR-AttnRes	4	512	0.08371	2.970
Block LR-AttnRes	4	768	0.09766	2.974
Block LR-AttnRes	8	16	0.08906	2.958
Block LR-AttnRes	8	32	0.09043	2.951
Block LR-AttnRes	8	64	0.09317	2.945
Block LR-AttnRes	8	128	0.09865	**2.944**
Block LR-AttnRes	8	256	0.1096	2.956
Block LR-AttnRes	8	512	0.1315	2.966
Block LR-AttnRes	8	768	0.1535	2.969
Block LR-AttnRes	16	16	0.1538	2.961
Block LR-AttnRes	16	32	0.1562	2.949
Block LR-AttnRes	16	64	0.1609	2.946
Block LR-AttnRes	16	128	0.1704	2.947
Block LR-AttnRes	16	256	0.1893	2.955
Block LR-AttnRes	16	512	0.2272	2.953
Block LR-AttnRes	16	768	0.2651	2.956
Block LR-AttnRes (full RMS)	8	128	0.09865	**2.943**

The best PROJECTED LR-ATTNRES configuration uses block routing with $N = 8$ and $r = 32$, reaching 2.944 validation loss at 1.058% added FLOPs. At the same rank, projected block losses are 2.945 for $N = 4$ and 2.951 for $N = 16$. Full projected routing improves from 2.957 to 2.947 as rank increases from 16 to 64, while added FLOPs rise from 0.8966% to 2.367%. Thus the projected results also depend on both rank and source grouping.

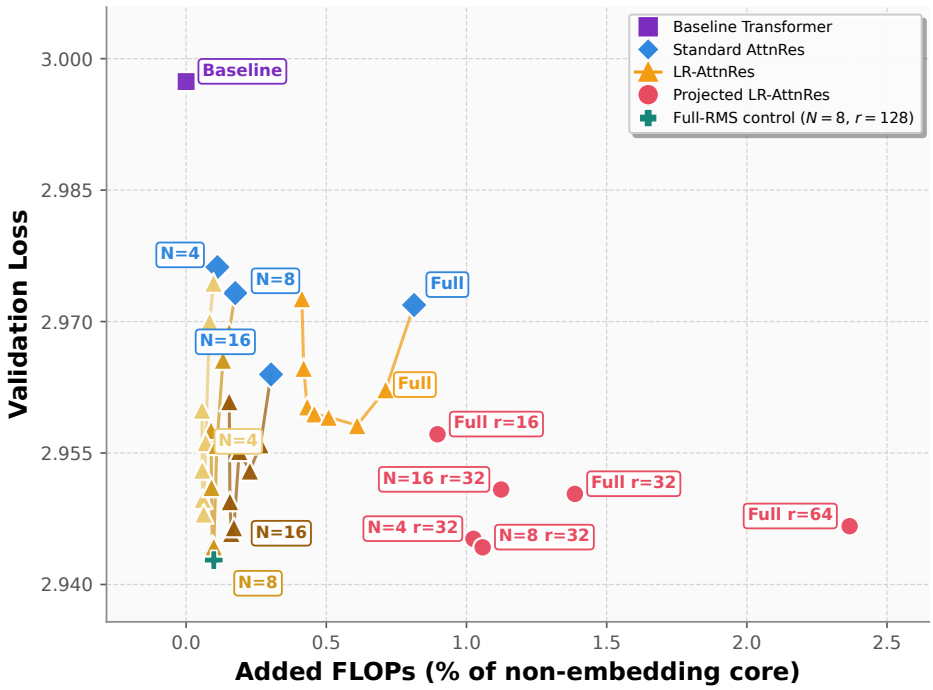


Figure 5: Validation loss versus added FLOPs relative to the non-embedding Transformer core at 0.5B, including the full-width-RMS control (teal plus). Only the $N = 8, r = 128$ LR-ATTNRES triangle and $N = 8, r = d = 1024$ standard ATTNRES diamond plot means across three training seeds, as in Figure 2; all other points show single runs. Projected variants include auxiliary key-projection FLOPs. Other variants use the depth-wise score-and-value accounting in Appendix F. Tables 5 and 4 retain single-run sweep values.

The sliced sweep identifies $N = 8, r = 128$ as its lowest-loss configuration: 2.944 validation loss at 0.09865% added FLOPs. Its loss is 6.263×10^{-4} lower than the best projected result, with 10.72× lower added residual-side FLOPs. The sliced $N = 8, r = 64$ alternative has loss only 8.978×10^{-4} higher than the sliced minimum, at 0.09317% added FLOPs. It remains a lower-compute near-tie. At matched full-routing rank $r = 64$, however, PROJECTED LR-ATTNRES reaches 2.947 compared with 2.960 for LR-ATTNRES, a loss difference of 0.01365. Dedicated keys therefore have lower observed loss in this full-routing comparison, while the block sweep offers a similar best loss at much lower routing cost (Table 5).

C.4 PROJECTED ROUTING DIAGNOSTICS

Figure 6 compares effective source counts across standard, projected, and sliced routing configurations, using the definition in Eq. 14. These routing diagnostics use 1,007,616 held-out tokens. The validation losses in Table 5 come from a separate 100M-token cross-entropy evaluation with the same checkpoint settings.

The matched comparisons show broader source use with low-rank keys, but broader routing does not consistently predict lower validation loss (Section 4.6). Our interpretation is that restricting the routing representation may limit sensitivity to incidental value features while preserving useful source distinctions.

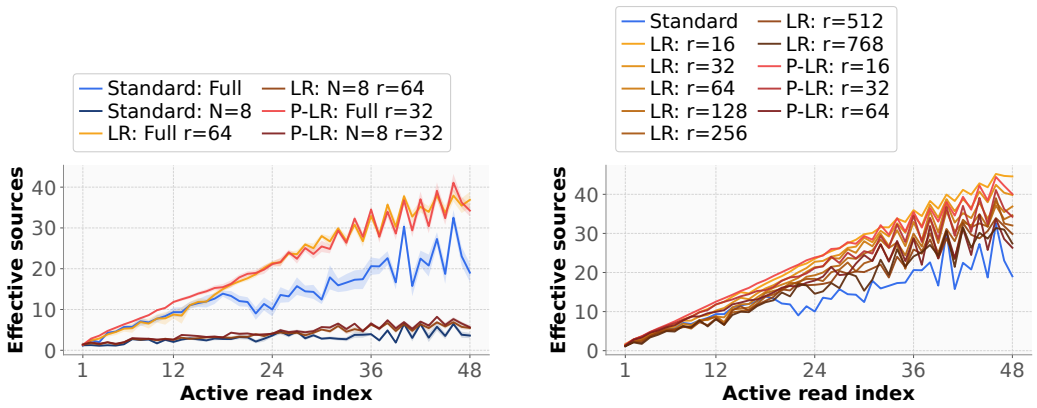


Figure 6: Effective number of residual sources across residual read sites. Left: comparison between standard AttnRes (Standard), LR-AttnRes (LR), and Projected LR-AttnRes (P-LR) under full and block routing. Right: rank sweep for full LR-AttnRes and full Projected LR-AttnRes, with standard full AttnRes shown in blue. Lines show token means; shading in the left panel indicates ± 1 standard deviation across 1M held-out tokens. Higher values indicate broader aggregation.

C.5 Gradient norms across routing configurations

Figure 7 compares mean gradient norms across routing-key ranks for projected and sliced routing, including the full-width-RMS control.

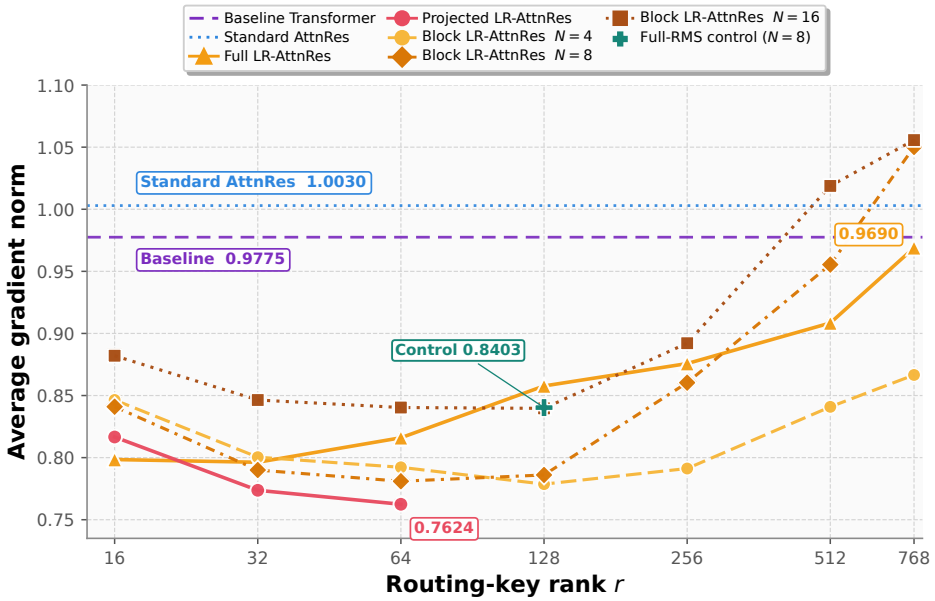


Figure 7: Average gradient norm by routing-key rank for the baseline, full AttnRes, full low-rank variants, and block LR-AttnRes. The teal plus marks the $N = 8, r = 128$ full-width-RMS control. The purple dashed line shows the baseline. The blue dotted line shows full standard AttnRes. Curves connect ranks within the same low-rank construction and block count N.

For full Projected LR-AttnRes, the mean gradient norm decreases from 0.8166 to 0.7737 and 0.7624 at ranks 16, 32, and 64, respectively, compared with 1.003 for full standard AttnRes. These three means are also below the baseline mean of 0.9775. The sliced sweep has a different response: full routing reaches its smallest mean at $r = 32$, whereas block settings $N = 4, 8, 16$ reach theirs at

$r = 128, 64, 128$. The sliced $N = 8$ validation minimum is instead at $r = 128$. Gradient magnitude and final validation loss therefore need not select the same rank.

For full sliced routing, the mean norm rises from 0.7963 at $r = 32$ to 0.9690 at $r = 768$. At $r = 768$, the $N = 8$ and $N = 16$ means reach 1.050 and 1.056, exceeding full routing. The nearly equal block losses at $r = 256$ accompany distinct mean norms of 0.7912, 0.8604, and 0.8921 for $N = 4, 8, 16$. Similar final losses therefore need not imply identical gradient statistics.

The full-routing validation minimum occurs at $r = 512$, compared with $r = 32$ for its mean gradient norm. The full-width-RMS control has a higher mean norm than sliced $N = 8, r = 128$ (0.8403 versus 0.7861), despite its lower loss. Lower gradient norms accompany some validation gains, but do not necessarily explain them.

Our interpretation is that a separate projected key may ease the demands placed on the routing representation, while reusing value coordinates may introduce competing routing and representation requirements. These are hypotheses about routing difficulty and representational tension, not quantities measured by gradient norms. In particular, the non-monotonic sliced curves do not support a general claim that increasing r necessarily increases this tension.

D Additional Routing Analyses

D.1 Mean attention heatmaps

To complement the entropy-based effective-source analysis, we also visualize the mean depth-wise attention weights directly. For each residual read site, we average the source-attention distribution over validation tokens and plot the resulting source-by-depth pattern. Figure 8 shows full LR-AttnRes at several ranks in the first row and block variants with $N = 8$ in the second row.

The full LR-AttnRes heatmaps show a clear rank-dependent smoothness pattern. At very small rank, especially $r = 16$, the attention map is comparatively smooth: probability mass changes gradually across residual read sites, and the model tends to form broad depth preferences rather than highly source-specific peaks.

As the rank increases to $r = 128$ and $r = 512$, the heatmaps become visibly less smooth. Checkerboard-like patterns, localized spikes, and sharper structures appear. This accompanies the reduction in effective-source count for the full sliced rank sweep (Figure 4). Validation loss improves from $r = 16$ to $r = 512$ despite the less smooth mean patterns.

All three block heatmaps show stage-dependent source preferences. At the final read, standard $N = 8$ AttnRes, projected $N = 8, r = 32$, and sliced $N = 8, r = 64$ routing have token-averaged effective-source counts of 3.62, 5.65, and 5.47, respectively. The heatmaps average attention weights across tokens; they do not establish smoother routing for individual tokens or a causal explanation for validation-loss differences.

E Kimi K3 folded-query utilization

E.1 Query construction and selection

Kimi K3 has hidden width $d = 7168$ and 93 Transformer layers partitioned into $N = 8$ AttnRes blocks (Kimi Team, 2026). At each residual-routing site, the implementation compares the current within-block prefix with the retained summaries of completed blocks and the input embedding. The deepest routing operations therefore select among at most nine source vectors.

The checkpoint stores one pre-attention query and one pre-MLP query for each layer, plus one final-output query, giving 187 stored query sites. The first layer's pre-attention query is structurally unused because only one source (input embedding) exists and the official forward pass skips that routing call. We therefore exclude it and analyze 186 active queries: 92 pre-attention queries, 93 pre-MLP queries, and one final-output query.

The implementation folds the RMSNorm affine weight into the learned query projection before computing routing scores. If $\boldsymbol{\gamma}_t \in \mathbb{R}^d$ is the RMSNorm weight and $\mathbf{w}_t \in \mathbb{R}^d$ is the projection row at

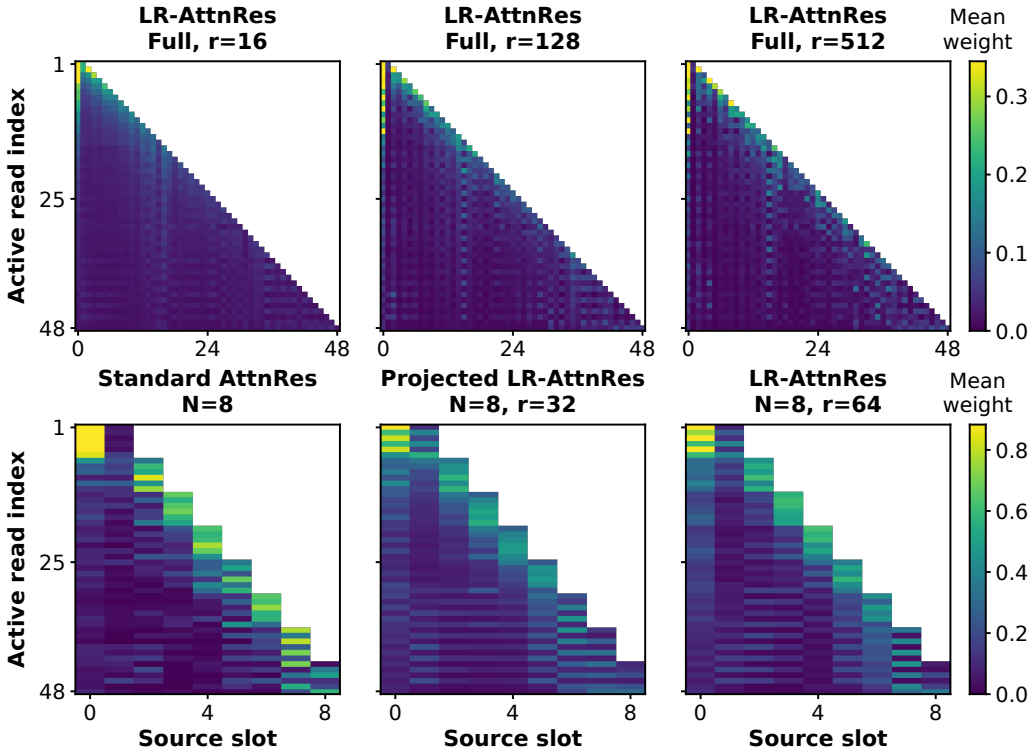


Figure 8: Mean depth-wise attention heatmaps over 1M held-out tokens. Top row: full LR-AttnRes with $r = 16$, $r = 128$, and $r = 512$. Bottom row: standard AttnRes, Projected LR-AttnRes ($r = 32$), and LR-AttnRes ($r = 64$), all with $N = 8$. The rows use separate color scales, with fixed upper limits of 0.345 and 0.883, respectively.

site t, the operational score vector is

$$\widetilde{\mathbf{q}}_t = \boldsymbol{\gamma}_t \odot \mathbf{w}_t. \tag{28}$$

Both factors are explicitly cast to FP32 before the elementwise product in the released implementation. All results below use this folded FP32 vector.

E.2 Coordinate effective dimension and exact zeros

For each folded query, we convert squared coordinate magnitudes into a probability distribution,

$$p_{t,j} = \frac{\widetilde{q}_{t,j}^2}{\sum_{k=1}^{d} \widetilde{q}_{t,k}^2}, \tag{29}$$

and define its Shannon coordinate effective dimension as

$$D_{\text{eff}}(\widetilde{\mathbf{q}}_t) = \exp\left(-\sum_{j=1}^{d} p_{t,j} \log p_{t,j} \right). \tag{30}$$

This is the entropy perplexity of the coordinate-energy distribution: it equals d when energy is uniform over all coordinates and equals 1 when one coordinate contains all energy.

We separately measure the fraction of each folded query's $d = 7168$ coordinates that are exactly equal to zero, using no tolerance threshold. The layerwise results show that the folded static queries use substantially less than their full coordinate width on average, but D_{eff} should not be interpreted as a recommended low-rank width.

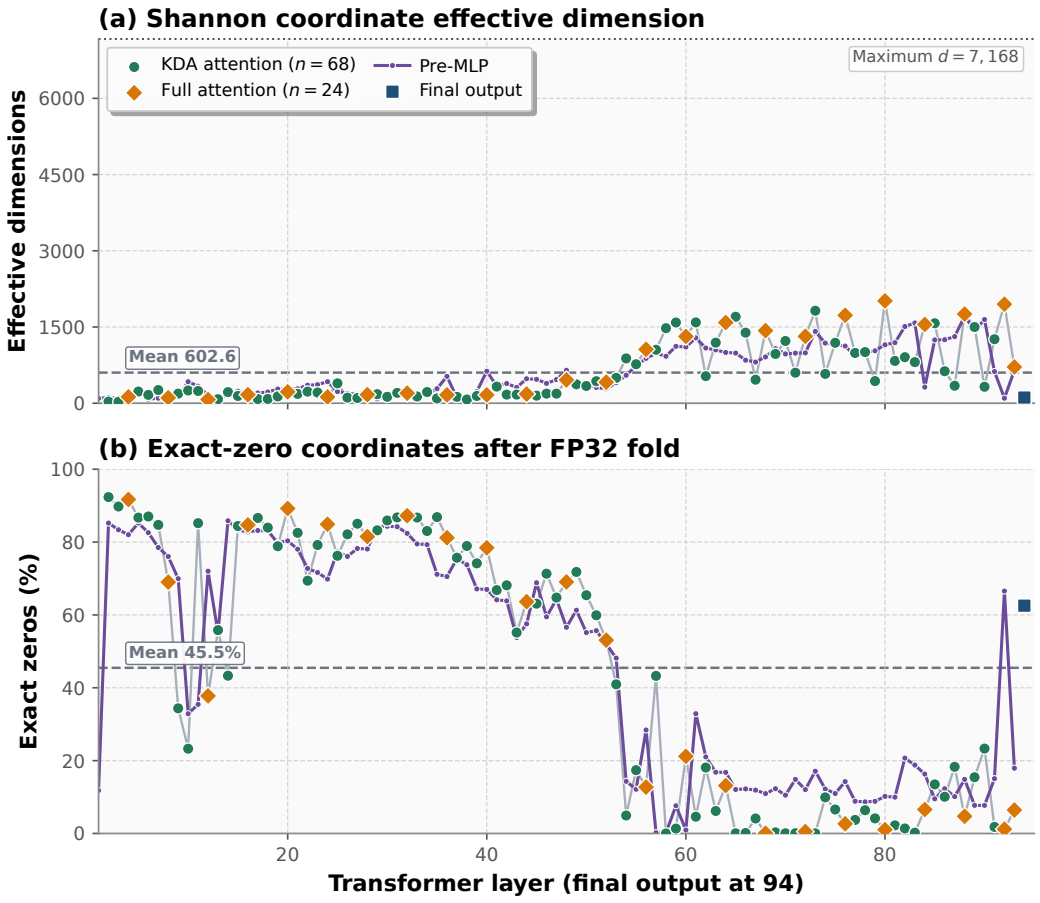


Figure 9: Layerwise utilization of Kimi K3's 186 active folded AttnRes queries. Top: Shannon coordinate effective dimension from Equation 30, shown on the full 0–7168 width scale. Bottom: fraction of coordinates exactly zero after the FP32 fold. Dashed lines show the all-query means of 602.6 effective dimensions and 45.5% exact zeros.

Table 6: Folded-query utilization by site type. KDA denotes Kimi Delta Attention. Full denotes full token attention. Effective-dimension statistics are computed per query. Zero fractions are averaged over queries in each row.

Query group	Count	Mean D_{eff}	Median D_{eff}	Mean zeros (%)
KDA pre-attention	68	546.2	325.7	45.3
Full pre-attention	24	791.5	437.3	43.4
Pre-MLP	93	600.4	402.3	46.0
Final output	1	111.8	111.8	62.5
All active queries	186	602.6	369.7	45.5

E.3 Numerical precision and limitations

Across the $186 \times 7168 = 1{,}333{,}248$ folded coordinates, 606,351 are exactly zero in the reconstructed FP32 product, giving the reported 45.4792%. Of those zeros, 606,264 occur even though both BF16-decoded factors are nonzero (they arise from underflow in the FP32 multiplication). Recomputing the same products in FP64 leaves only 87 exact zeros, confirming that almost all observed zeros arise during the FP32 fold.

Many learned folded-query coordinates are therefore exactly zero under the released implementation's arithmetic and make no direct contribution to the routing-score numerator. This is consistent with optimization suppressing some coordinates' contribution to source selection, suggesting that full-width routing queries may be unnecessary. Realizing memory or speed benefits requires an implementation that exploits this sparsity.

F FLOPs, ACTIVATION MEMORY, AND PARAMETER ACCOUNTING

This appendix gives the detailed accounting for standard ATTNRES, projected low-rank keys, and sliced low-rank keys. The main distinction is between two types of cost. The *depth-wise residual kernel* is the attention-over-depth computation itself: it scores retained sources and mixes their values. The *projected-key path* is the extra work needed only by PROJECTED LR-ATTNRES to produce learned low-rank keys.

Table 7: Residual-side cost summary. $S_{\mathrm{src}} = \sum_{t\in\mathcal{T}} |\mathcal{S}_t|$ is the total number of source reads across residual sites, and K_{src} is the number of retained source activations.

Method	Total residual-side FLOPs	Source-cache activations	Added parameters
ATTNRES	$2dS_{\mathrm{src}}$	$K_{\mathrm{src}}d$	$2Ld$
PROJECTED LR-ATTNRES	$(d+r)S_{\mathrm{src}} + r[L(d+m)+d]$	$K_{\mathrm{src}}(d+r)$	$r[L(d+m)+d+2L]$
LR-ATTNRES	$(d+r)S_{\mathrm{src}}$	$K_{\mathrm{src}}d$	$2Lr$

Let $\mathcal{T}$ be the set of residual read sites and let

$$S_{\mathrm{src}} = \sum_{t\in\mathcal{T}} n_t, \qquad n_t = |\mathcal{S}_t|. \tag{31}$$

The source-count term S_{src} depends on whether the method uses full or block sources.

F.1 DEPTH-WISE RESIDUAL KERNEL FLOPS

Ordinary ATTNRES uses the same d-dimensional tensor as both key and value. At residual site t, the score computation costs $n_t d$ multiply-adds and the weighted value mixture costs another $n_t d$ multiply-adds:

$$C_{\mathrm{AttnRes}}(t) = n_t d + n_t d = 2n_t d. \tag{32}$$

Summing over residual read sites gives

$$C_{\mathrm{AttnRes}} = 2dS_{\mathrm{src}}. \tag{33}$$

Both PROJECTED LR-ATTNRES and LR-ATTNRES use an r-dimensional key and a full d-dimensional value. Therefore the depth-wise residual kernel cost is

$$C_{\mathrm{LR,kernel}}(t) = n_t r + n_t d = n_t(d+r), \tag{34}$$

and

$$C_{\mathrm{LR,kernel}} = (d+r)S_{\mathrm{src}}. \tag{35}$$

The percentage reduction in the depth-wise residual kernel relative to ordinary ATTNRES is

$$\%\,\Delta C_{\mathrm{kernel}} = 100\left(1 - \frac{C_{\mathrm{LR,kernel}}}{C_{\mathrm{AttnRes}}}\right) \tag{36}$$

$$= 100\left(1 - \frac{(d+r)S_{\mathrm{src}}}{2dS_{\mathrm{src}}}\right) \tag{37}$$

$$= 100\left(\frac{d-r}{2d}\right). \tag{38}$$

The full-width-RMS control retains the same r-dimensional dot products and d-dimensional value mixtures, so it has the same $(d+r)S_{\mathrm{src}}$ cost as slice-normalized LR-ATTNRES under this accounting.

Its RMS statistic instead uses all d coordinates. This normalization work is outside the score-and-value count plotted in Figure 5.

The source-count term cancels, so the same kernel percentage applies to full and block source sets. These counts include score dot products and value mixing and exclude normalization and softmax arithmetic. Slice normalization additionally reduces the normalization width from d to r, whereas full-width-RMS conversion retains the original normalization width.

F.2 PROJECTED-KEY FLOPS

PROJECTED LR-ATTNRES produces a learned low-rank key by adding r rows to output projections that already exist. Let m_i be the input width of the output projection for residual-writing sub-layer i. The extra projected-key cost is

$$C_{\mathrm{P,key\text{-}proj}} = r \sum_{i=1}^{2L} m_i + rd, \tag{39}$$

where the final rd term is the embedding key projection. For a Transformer block with one attention output projection of input width d and one SwiGLU output projection of input width m,

$$\sum_{i=1}^{2L} m_i = L(d+m), \tag{40}$$

so

$$C_{\mathrm{P,key\text{-}proj}} = r\big[L(d+m)+d\big]. \tag{41}$$

Thus the total residual-side cost of PROJECTED LR-ATTNRES is

$$C_{\mathrm{P,total}} = (d+r)S_{\mathrm{src}} + r\big[L(d+m)+d\big]. \tag{42}$$

Relative to ordinary ATTNRES, the net percentage change is

$$\%\,\Delta C_{\mathrm{P,total}} = 100\left(1 - \frac{(d+r)S_{\mathrm{src}} + r\big[L(d+m)+d\big]}{2dS_{\mathrm{src}}}\right). \tag{43}$$

Therefore, PROJECTED LR-ATTNRES reduces the depth-wise residual kernel cost, but it reduces *total* residual-side FLOPs only when

$$r\big[L(d+m)+d\big] < (d-r)S_{\mathrm{src}}. \tag{44}$$

This condition can fail, especially in block mode where S_{src} is small. In that case, PROJECTED LR-ATTNRES improves validation loss but does not reduce total added FLOPs once projected-key computation is included.

F.3 SLICED-KEY FLOPS

LR-ATTNRES obtains its key by slicing the last r dimensions of the already-computed residual value:

$$\boldsymbol{k}_i^{\mathrm{S}} = \mathrm{tail}_r(\boldsymbol{o}_i). \tag{45}$$

It therefore has no auxiliary key projection:

$$C_{\mathrm{S,key\text{-}proj}} = 0. \tag{46}$$

Its total residual-side cost is just the low-rank depth-wise residual kernel:

$$C_{\mathrm{S,total}} = (d+r)S_{\mathrm{src}}. \tag{47}$$

Thus LR-ATTNRES reduces total residual-side FLOPs relative to ordinary ATTNRES whenever $r < d$:

$$\%\,\Delta C_{\mathrm{S,total}} = 100\left(1 - \frac{(d+r)S_{\mathrm{src}}}{2dS_{\mathrm{src}}}\right) = 100\left(\frac{d-r}{2d}\right). \tag{48}$$

This is the main efficiency advantage of slicing. PROJECTED LR-ATTNRES and LR-ATTNRES have the same low-rank depth-wise kernel, but only LR-ATTNRES removes the projected-key path entirely.

F.4 ACTIVATION MEMORY

Let K_{src} be the number of retained depth sources. For full ATTNRES, this is the number of previous residual-writing sources. For BLOCK ATTNRES, this is the number of retained block sources. Assume a fused depth-wise attention kernel, analogous to FlashAttention, so the attention-weight vector is not materialized for backward. If each activation element uses b bytes, the dominant source-cache activation memory for ordinary ATTNRES is

$$A_{\mathrm{AttnRes}} = bK_{\mathrm{src}}d. \tag{49}$$

PROJECTED LR-ATTNRES caches the full-dimensional values and the projected low-rank keys:

$$A_{\mathrm{P}} = bK_{\mathrm{src}}(d + r). \tag{50}$$

Therefore the retained source-cache activation memory increases by

$$\%\,\Delta A_{\mathrm{P}} = 100\left(\frac{A_{\mathrm{P}}}{A_{\mathrm{AttnRes}}} - 1\right) = 100\frac{r}{d}. \tag{51}$$

LR-ATTNRES can implement the key as a view into the last r coordinates of the cached value. In that implementation, it does not require a separate key cache:

$$A_{\mathrm{S}} = bK_{\mathrm{src}}d = A_{\mathrm{AttnRes}}. \tag{52}$$

Thus the dominant retained source-cache activation increase is

$$\%\,\Delta A_{\mathrm{S}} = 0. \tag{53}$$

An implementation may optionally cache normalized sliced keys, but this is not required by the method.

F.5 PARAMETERS

The parameter count for PROJECTED LR-ATTNRES consists of projected key rows and residual-site queries. For residual-writing sub-layer i, adding an r-dimensional key to an output projection with input width m_i adds rm_i parameters. The embedding key projection adds rd parameters. The $2L$ query-bearing read sites each have one r-dimensional static query, adding $2Lr$ parameters. Thus

$$\Delta P_{\mathrm{P,key}} = r\sum_{i=1}^{2L} m_i + rd, \tag{54}$$

$$\Delta P_{\mathrm{P,query}} = 2Lr, \tag{55}$$

$$\Delta P_{\mathrm{P}} = r\sum_{i=1}^{2L} m_i + rd + 2Lr. \tag{56}$$

With one attention output projection of width d and one SwiGLU output projection of width m per Transformer block,

$$\Delta P_{\mathrm{P}} = r\big[L(d + m) + d + 2L\big]. \tag{57}$$

LR-ATTNRES has no key-projection parameters. It only needs the low-rank residual-site queries:

$$\Delta P_{\mathrm{S}} = 2Lr. \tag{58}$$

Relative to a base model with P_{base} parameters, the parameter increases are

$$\%\,\Delta P_{\mathrm{P}} = 100\frac{r\big[L(d + m) + d + 2L\big]}{P_{\mathrm{base}}}, \tag{59}$$

$$\%\,\Delta P_{\mathrm{S}} = 100\frac{2Lr}{P_{\mathrm{base}}}. \tag{60}$$

If ordinary ATTNRES is the reference instead of the no-ATTNRES baseline, then ordinary ATTNRES already contains $2Ld$ query parameters under the same parameter-free RMSNorm convention. The parameter differences become

$$\Delta P_{\mathrm{P-vs-AttnRes}} = r\big[L(d + m) + d + 2L\big] - 2Ld, \tag{61}$$

$$\Delta P_{\mathrm{S-vs-AttnRes}} = 2Lr - 2Ld = -2L(d - r). \tag{62}$$

Thus sliced low-rank routing uses fewer query parameters than full-dimensional ATTNRES and adds no key-projection parameters.

F.6 NUMERICAL EXAMPLE FOR THE EXPERIMENTAL MODEL

For an illustrative $r = 32$ configuration of the 0.5B model, $d = 1024$, $L = 24$, and $m = 2816$. The low-rank depth-wise residual kernel reduction is

$$100\left(\frac{1024 - 32}{2 \cdot 1024}\right) = 48.44\%. \tag{63}$$

Both PROJECTED LR-ATTNRES and LR-ATTNRES obtain this kernel reduction.

The projected-key path in PROJECTED LR-ATTNRES adds

$$C_{\mathrm{P,key\text{-}proj}} = 32\,[24(1024 + 2816) + 1024] = 2{,}981{,}888 \tag{64}$$

multiply-adds per token. LR-ATTNRES removes this term.

The parameter increase for PROJECTED LR-ATTNRES is

$$\Delta P_{\mathrm{P}} = 32\,[24(1024 + 2816) + 1024 + 48] = 2{,}983{,}424, \tag{65}$$

while the sliced-key parameter increase is only

$$\Delta P_{\mathrm{S}} = 2 \cdot 24 \cdot 32 = 1{,}536. \tag{66}$$

For retained source-cache activations, PROJECTED LR-ATTNRES increases the value-plus-key cache by

$$100\frac{32}{1024} = 3.125\%, \tag{67}$$

whereas LR-ATTNRES adds no separate source-key cache when implemented as a slice of the cached value.

For a dense SwiGLU Transformer core with

$$C_{\mathrm{base}} = L(4d^2 + 3dm), \tag{68}$$

the projected-key parameter overhead, ignoring embeddings and output heads, is approximately

$$100\frac{32}{1024}\frac{1 + 2816/1024}{4 + 3(2816/1024)} = 0.957\%. \tag{69}$$

The exact projected-key numerator above gives roughly the same scale, while the sliced-key query-only overhead is negligible.

Under the added-FLOP accounting used in Table 5, the projected-key path contributes approximately 0.9673% of the non-embedding Transformer core for $r = 32$. Thus the full projected $r = 32$ row has total added FLOPs $0.4193\% + 0.9673\% \approx 1.387\%$, while block projected $N = 8, r = 32$ has total added FLOPs $0.09043\% + 0.9673\% \approx 1.058\%$. LR-ATTNRES has no projected-key path, so its total added FLOPs equal its depth-kernel compute.

G 1B AND 4B EVALUATION DETAILS

Downstream protocol. We use `lm-evaluation-harness` v0.4.7 with its task prompts, zero demonstrations, the `cl100k_base` tokenizer, and a 2048-token context. A custom adapter scores answer continuations with BF16 model computation and FP32 log-probabilities, using batches of eight and causal right padding. Prompt and answer tokenization preserves boundary whitespace. Overlong inputs are left-truncated while retaining the complete answer, and an end-of-text prefix is used only for an empty prompt. There is no chat template or additional beginning-of-sequence token.

We evaluate the complete test splits of ARC-Challenge (1172 items), ARC-Easy (2376), and OpenBookQA (500), and the validation splits of BoolQ (3270), CommonsenseQA (1221), HellaSwag (10,042), PIQA (1838), and Social-IQA (1954). We report `acc_norm` for ARC-Challenge, ARC-Easy, HellaSwag, OpenBookQA, and PIQA, selecting the answer with the highest log-likelihood divided by its character length. BoolQ, CommonsenseQA, and Social-IQA use `acc`, based on unnormalized answer log-likelihood. The reported average weights the eight task accuracies equally.

Table 8: Zero-shot downstream accuracy (%) for the 1B and 4B pairs. Metrics and evaluation splits are specified above. Bold marks the higher accuracy within each scale.

	1B		4B	
Task	AttnRes	LR-AttnRes	AttnRes	LR-AttnRes
ARC-Challenge	**28.58**	28.24	28.07	**28.67**
ARC-Easy	52.06	**52.10**	53.49	**55.39**
BoolQ	**59.79**	57.89	61.25	**61.41**
CommonsenseQA	18.84	**20.39**	20.39	**20.97**
HellaSwag	39.23	**40.13**	41.10	**42.38**
OpenBookQA	31.20	**32.20**	**33.20**	31.40
PIQA	67.74	**68.06**	68.55	**68.72**
Social-IQA	37.51	**39.10**	37.72	**39.41**
Eight-task mean	41.87	**42.27**	42.97	**43.54**

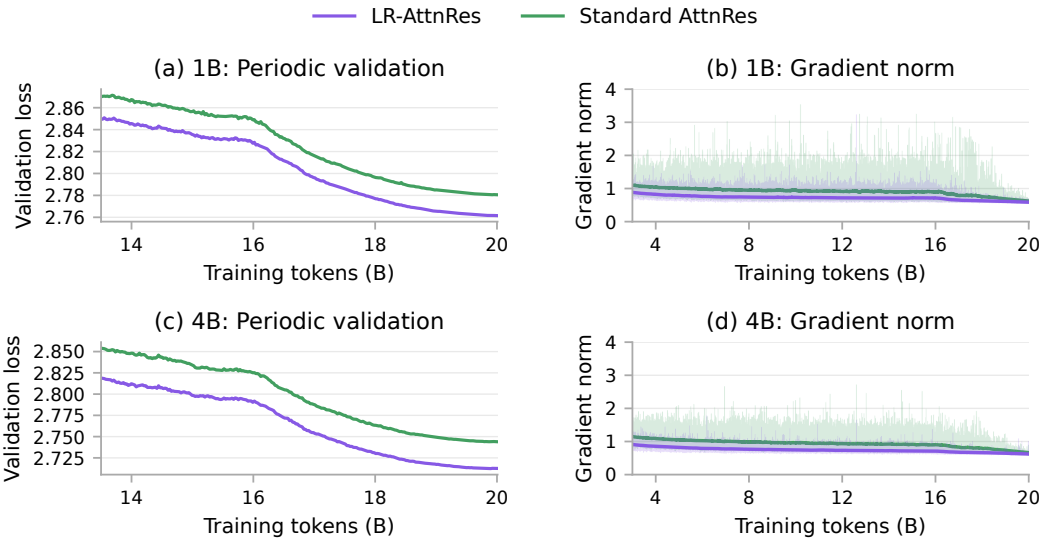


Figure 10: Training dynamics at 1B (top) and 4B (bottom), with $r = d/4$ for LR-AttnRes. Left: periodic validation loss over the final 6.5B training tokens. Right: pre-clipping gradient norms from 3B to 20B tokens on linear axes, capped at 4 for display. Faint traces are individual steps and solid curves are trailing 1,000-step means.

G.1 GRADIENT-NORM STATISTICS

Table 9 summarizes pre-clipping gradient norms over all 76,294 training steps. Across the full training runs, LR-ATTNRES reduces mean gradient norm by 19.5% at 1B and 21.2% at 4B, and population variance by 30.8% and 34.1%. The fraction of logged norms above the clipping threshold of 1 falls from 35.1% to 11.0% and from 38.3% to 9.9%, respectively.

Table 9: Full-run gradient-norm statistics at 1B and 4B. The final column is the percentage of steps with pre-clipping norm above 1.

Model	Method	Mean	Variance	Above 1 (%)
1B	ATTNRES	0.9754	0.1747	35.1
1B	LR-ATTNRES	0.7853	0.1209	11.0
4B	ATTNRES	1.022	0.1679	38.3
4B	LR-ATTNRES	0.8055	0.1106	9.9

H PYTORCH-STYLE PSEUDOCODE

```
def route(sources, q, r):
    V = torch.stack(sources)                    # [S, B, T, d]
    K = rms_norm(V[..., -r:])                   # [S, B, T, r]
    weights = (K * q).sum(-1).softmax(dim=0)
    return (weights[..., None] * V).sum(dim=0)

def lr_attnres(x, sublayers, queries, r, N=None):
    M = len(sublayers)                          # M = 2L
    N = M if N is None else min(N, M)
    ends = {ceil(M * j / N) for j in range(1, N + 1)}
    blocks, partial, h = [x], None, x

    for t, (f, q) in enumerate(zip(sublayers, queries), start=1):
        y = f(rms_norm(h))                      # attention or SwiGLU
        partial = y if partial is None else partial + y
        if t in ends:
            blocks.append(partial)
            partial = None
        sources = blocks + ([] if partial is None else [partial])
        h = route(sources, q, r)
    return rms_norm(h)                          # then the LM head
```

Figure 11: PyTorch-style pseudocode for full and block LR-ATTNRES. `x` is the token embedding. `sublayers` alternates attention and SwiGLU transformations without residual addition. The $2L$ static r-dimensional queries are learned and initialized to zero. `rms_norm` normalizes the last dimension. `N=None` retains individual outputs (full routing), whereas a specified N groups them into block sums. Setting $r = d$ recovers standard ATTNRES with the same source grouping.

I FAST-ATTNRES KERNELS AND TRAINING THROUGHPUT

Implementation. FAST-ATTNRES supports standard routing ($r = d$) and sliced routing ($r < d$) through a shared source-list interface. Building on the source-pointer and tiled online-softmax design of FLA (Yang & Zhang, 2024), our implementation reads the last r coordinates of each source for parameter-free RMS normalization and query scoring, while retaining all d coordinates for the weighted value sum. The same operator serves full and block routing. The model determines which sources are retained. BF16 inputs and outputs use FP32 internal accumulation. The backward pass combines the direct value gradient with the routing gradient on the selected coordinates before writing the BF16 source gradient, and separately reduces the query gradient. Tiling supports unaligned widths and ranks. The source-list path avoids stacking values, saves normalization and softmax statistics, and recomputes the mixed value only for lists of up to three sources. PyTorch custom operators

provide autograd and `torch.compile` integration. The implementation and its FLA attribution are available in the versioned source release.

Numerical correctness. We test the trained 1B and 4B checkpoints on H100 using one 2048-token batch per case. At each routing read, the native operator and independent PyTorch/autograd reference receive identical BF16 inputs and loss-derived upstream gradients. All 408 reads across six cases pass the elementwise BF16 criterion $|x - x_{\mathrm{ref}}| \le 0.05 + 0.05|x_{\mathrm{ref}}|$. Table 10 shows comparable relative errors for FLA and FAST-ATTNRES at full width. Reduced-rank checks cover the non-power-of-two ranks 384 and 640.

Table 10: Maximum relative ℓ_2 errors (%) against PyTorch on trained checkpoints. **All cases pass the BF16 tolerance**. Maxima cover 48 reads at 1B, 88 at 4B, and independent source gradients. FLA uses checkpoint level 1.

Model	Implementation	r	Forward	Source grad.	Query grad.
1B	FLA	1536	0.003309	0.29	0.02017
1B	FAST-ATTNRES	1536	0.004766	0.2899	0.01689
1B	FAST-ATTNRES	384	0.002804	0.2179	0.01557
4B	FLA	2560	0.003682	0.3029	0.01388
4B	FAST-ATTNRES	2560	0.003682	0.3007	0.0127
4B	FAST-ATTNRES	640	0.003156	0.2636	0.03336

Under the timing software stack below, B200 checks of all three configurations at both model widths and source counts $2, 5, 9, 10$ pass the same criterion for outputs and both gradients (24 cases with 16 random token rows each). Separate B200 checks with PyTorch 2.13.0 and Triton 3.7.1 cover unaligned width $d = 513$, repeated or strided sources, strided queries, and changed-input replays. All pass the same criterion.

Matched model comparison. All timings use $N = 8$, with FLA at $r = d$ and FAST-ATTNRES at $r = d$ or $d/4$. Methods share the production Transformer, document masking, vocabulary head, training objective, and Muon/Adam optimizers. The 1B and 4B models use $(L, d, m) = (24, 1536, 4224)$ and $(44, 2560, 7040)$, respectively, with head width 64, context length 2048, and vocabulary size 100,277.

Measurement protocol. Each three-method repeat uses matched shared weights, seed, cached real training tokens, and production document boundaries. Both GPUs use BF16, PyTorch 2.10.0 (CUDA 13.0), Triton 3.6.0, FlashAttention 2.8.3, and `fullgraph=True, dynamic=False`, without CUDA Graph capture. Microbatch size 2 and 64 accumulation steps give 262,144 tokens per update. Synchronized timing includes host-to-device transfer, forward, backward, accumulation, clipping, and optimizer updates. Compilation, storage reads, evaluation, and logging are excluded. Three warmup updates precede eight to thirty measured updates. Timing stops when two consecutive six-update windows each have a coefficient of variation of at most 1% and drift of at most 0.5% between their halves. All measured updates contribute. Each GPU uses five fresh repeats per configuration (30 measurements), with rotated method order and matched production code, data, and shared backbone initialization across GPUs.

Paired uncertainty. For each of the five matched repeats, let $v_j^{(d)}$ and $v_j^{(d/4)}$ denote complete training-step throughput under FAST-ATTNRES. We compute the percentage gain and its approximate 95% Student-t confidence interval as

$$g_j = 100\left(\frac{v_j^{(d/4)}}{v_j^{(d)}} - 1\right), \qquad \bar{g} \pm t_{0.975,4}\frac{s_g}{\sqrt{5}},$$

where s_g is the sample standard deviation of the five gains. Each observation is a matched pair of fresh processes. The interval assumes independent, approximately normally distributed repeat-level gains. Training-throughput gains from changing the kernel at fixed rank, computed from mean throughput at $r = d$, are 0.95% and 0.74% at 1B and 4B on H100, and 2.17% and 1.75% on B200.