MULTI-HEAD ATTENTION RESIDUALS

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformers propagate information across depth through a single additive residual stream. *Attention residuals* relax this by letting each sublayer attend, through a learned softmax, over all earlier sublayer outputs—but that read shares one query across the entire width, forcing every feature subspace through the same depth distribution, at a cost that grows with model width. We introduce **Multi-Head Attention Residuals (MHAR)**, which splits the routing query into H per-subspace heads, each with its own softmax over depth. The split adds no parameters and negligible compute, and $H{=}1$ recovers attention residuals exactly. Trained from scratch on a STEM- and code-heavy corpus we construct, MHAR improves validation loss over a standard Transformer by $0.061/0.149/0.140$ at 100M/350M/1B, achieving the lowest loss among four methods in every setting. Improvements also appear in held-out perplexity and LAMBADA accuracy and remain under a document-disjoint re-evaluation and at a compute-optimal token budget. Validation loss is U-shaped in H with a flat optimum at $H{=}4$–8; we adopt $H{=}8$. Temperature-matched controls attribute most of the gain to flatter per-head softmax distributions, with a small residual consistent with per-subspace routing. A single query's depth softmax becomes overly sharp as width grows. Fused Triton kernels lift attention-residual training from 0.2–$0.5\times$ to 0.55–$0.88\times$ baseline throughput at near-baseline memory, and an identity-preserving conversion brings MHAR to 8B mid-training ($+3.2$ GSM8K, $+3.1$ GPQA).

1 INTRODUCTION

Transformers (Vaswani et al., 2017) propagate information across depth through a single residual stream (He et al., 2016): layer ℓ reads $\mathbf{h}_{\ell-1}$ and writes $\mathbf{h}_\ell = \mathbf{h}_{\ell-1} + f_\ell(\mathbf{h}_{\ell-1})$. The running sum couples every sublayer's input to the immediately preceding output, even though useful features may have been computed many layers earlier. *Attention residuals* relax this by treating the set of all prior sublayer outputs as a memory and letting each sublayer retrieve a learned, softmax-weighted combination of them, an attention over depth rather than over tokens (Kimi Team, 2026).

This retrieval makes the depth history addressable, but only at the granularity of the *entire width*. A single learned query $\mathbf{q} \in \mathbb{R}^d$ (Kimi Team, 2026) scores the $N{=}2L{+}1$ prior sources and collapses them into one softmax distribution α over depth; all d coordinates of the retrieved mixture are then read through that same α: every feature dimension is forced to read from the same layers in the same proportions (Figure 1, zoom left).

Multi-head self-attention allows feature subspaces to use different distributions over tokens. The same motivation applies to depth routing: different subspaces may prefer different prior layers, while a single routing query requires them to share one distribution.

We therefore propose **Multi-Head Attention Residuals (MHAR)**, a strict generalization that splits the routing query into H per-subspace heads, each routing independently over the depth history, and recovers attention residuals exactly at $H{=}1$ (Figure 1, zoom right). The split preserves the number of routing parameters and adds negligible compute. Depth routing incurs memory traffic because every sublayer re-reads the depth history, with repeated reads in reference implementations. We provide fused Triton routing kernels that remove most of this overhead (Appendix F).

A single query shares one softmax distribution across subspaces with different depth preferences, a restriction we call the *forced compromise*. Its logits are inner products over the full width, so their scale grows with width and sharpens the shared softmax. Temperature-matched controls (Appendix E)

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

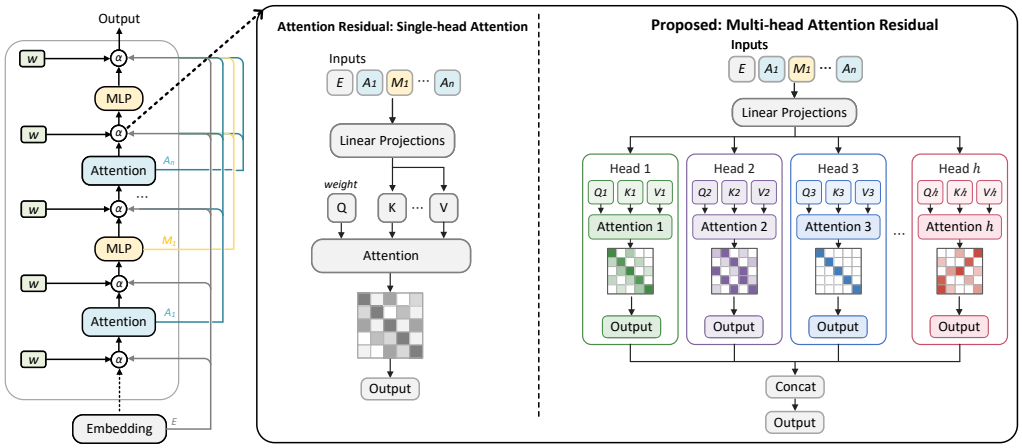

Figure 1: **Single-head and multi-head attention over depth. Left:** attention residuals (Kimi Team, 2026) feed every sublayer a learned mixture of the depth history. **Zoom left:** one routing site is *single-head* attention over that history—one depth map shared by all d channels (Eq. 2). **Zoom right:** MHAR gives each of H subspaces its own map (Eq. 3); $H{=}1$ recovers attention residuals.

attribute most of the measured loss difference to over-sharpening, with a small residual consistent with subspace disagreement. Splitting the query into per-subspace heads makes the read block-diagonal, while the d/H-dimensional logits produce flatter softmax distributions. We examine the width dependence through loss (§3) and direct measurements of the trained queries (Appendix E).

Our contributions:

- **Multi-head attention residuals (MHAR).** We split the single attention-residual routing query into H per-subspace heads (§2): a reshape that adds no parameters or FLOPs and recovers attention residuals exactly at $H{=}1$.
- **Fused kernels for efficient MHAR.** Fused Triton routing kernels lift attention-residual training from $\sim$0.2–0.5$\times$ to 0.55–0.88$\times$ baseline throughput at near-baseline peak memory (Appendix F), making depth routing practical at scale.
- **Scaling MHAR.** Trained from scratch at 100M–1B, MHAR is the lowest-loss method among the four at every scale, with larger improvements over the baseline at 350M and 1B than at 100M (-0.061/-0.149/-0.140; §3), and the validation loss varies with routing-head count in a U-shaped pattern, with minima at $H{=}4$ or $H{=}8$ (cf. the 8 KV groups of GQA (Ainslie et al., 2023); we adopt $H{=}8$ at scale; §4). An identity-preserving conversion (Luo et al., 2026) extends MHAR to 8B mid-training, adding $+3.2$ GSM8K (paired $p{=}0.004$) over a schedule-matched control (§3.3).

2 Method: Multi-Head Attention Residuals

2.1 Background: The Residual Stream as Depth Memory

A pre-norm decoder Transformer (Vaswani et al., 2017; Xiong et al., 2020) of width d with L blocks maintains a single running state and updates it additively (Figure 2, left). Writing the two sublayers of block ℓ (attention and MLP) as f_ℓ and g_ℓ,

$$\mathbf{h}'_\ell = \mathbf{h}_{\ell-1} + f_\ell\big(\mathrm{LN}(\mathbf{h}_{\ell-1})\big), \qquad \mathbf{h}_\ell = \mathbf{h}'_\ell + g_\ell\big(\mathrm{LN}(\mathbf{h}'_\ell)\big). \tag{1}$$

Because the state is a running sum, every sublayer reads only the most recent state $\mathbf{h}_{\ell-1}$. Features computed many layers earlier are still present in the sum, but are no longer individually addressable: the network cannot selectively re-read the output of, say, layer 3 when computing layer 30 without that information first surviving every intervening additive update.

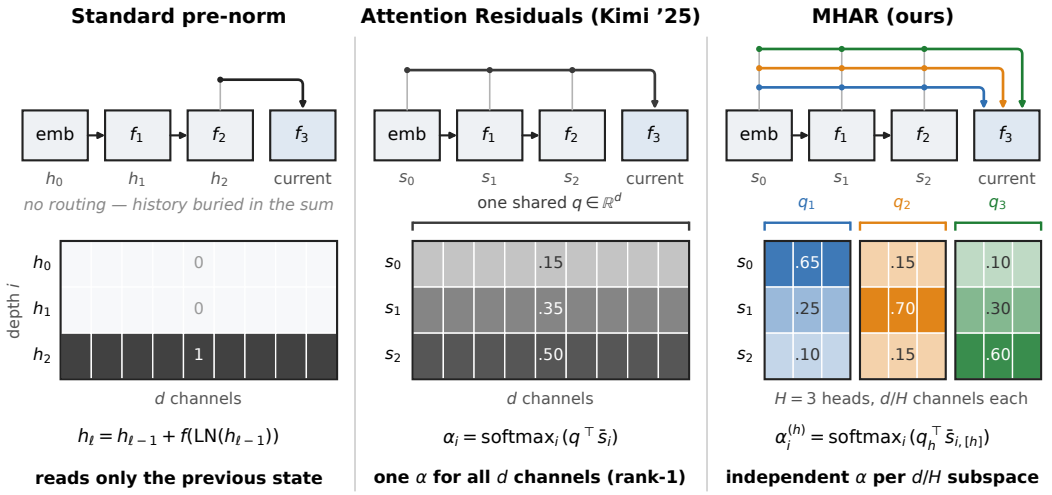

Figure 2: **Access to prior states and sublayer outputs across depth.** Top: forward stack; bottom: illustrative depth-read weights (entry $[i, c]$: how strongly channel c reads depth row i; $\bar{\mathbf{s}}_i$=RMSNorm($\mathbf{s}_i$)). **Pre-norm:** a one-hot read of the previous state. **Attention residuals:** one shared query, every column identical (rank-1). **MHAR (ours):** each subspace reads depth independently.

2.2 Attention Residuals

Attention residuals (Kimi Team, 2026) make past sublayer outputs individually addressable by treating them as a memory and *routing* over them. Let the network expose the ordered list of *sources* $\mathcal{S} = (\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_{N-1})$ of $N{=}2L{+}1$ sources, where $\mathbf{s}_0$ is the token embedding and each later source is the raw output of one attention or MLP sublayer. In place of the additive update (Eq. 1), the input to a sublayer is a learned, softmax-weighted combination of *all* sources produced so far. With a per-sublayer learned pseudo-query $\mathbf{q} \in \mathbb{R}^d$ and an RMSNorm applied to the sources as keys,

$$\alpha_i = \frac{\exp\big(\mathbf{q}^\top \mathrm{RMSNorm}(\mathbf{s}_i)\big)}{\sum_{j=0}^{M} \exp\big(\mathbf{q}^\top \mathrm{RMSNorm}(\mathbf{s}_j)\big)}, \qquad \tilde{\mathbf{h}} = \sum_{i=0}^{M} \alpha_i \, \mathbf{s}_i, \tag{2}$$

where M indexes the sources available at that point. This is an attention over *depth* (the source/layer axis) rather than over tokens; the analogue of keys are the normalized sources and the analogue of values are the sources themselves, with a single learned query per routing site (Figure 2, middle).

2.3 Multi-head routing

A single query (Eq. 2) forces *all* d coordinates of the routed mixture to share one distribution α over depth: every feature dimension must read from the same layers in the same proportions. Different feature subspaces, however, may be best served by different layers. We remove this constraint with **multi-head routing (MHAR)** (Figure 2, right): partition the coordinates into H contiguous heads of width d/H, give each head its own query, and let each head route independently. For head h with query $\mathbf{q}_h \in \mathbb{R}^{d/H}$ and writing $\mathbf{x}_{[h]}$ for the h-th slice of a vector $\mathbf{x}$,

$$\alpha_i^{(h)} = \mathrm{softmax}_i\Big(\mathbf{q}_h^\top \, \mathrm{RMSNorm}(\mathbf{s}_i)_{[h]}\Big), \qquad \tilde{\mathbf{h}}_{[h]} = \sum_i \alpha_i^{(h)} \, \mathbf{s}_{i,[h]}, \tag{3}$$

and the H routed slices are concatenated to width d. Each head scores only its own slice of each source and mixes only that slice, so the H softmaxes are fully independent and different subspaces can attend to different layers (implementation in Figure 3).

Properties. *(i) Parameter-matched.* The H queries $\{\mathbf{q}_h\}_{h=1}^{H}$ together form an $H \times (d/H)$ matrix with exactly d parameters—the same as the single d-dimensional query. Multi-head routing therefore

```python
def route(self, sources: list[Tensor], proj: Linear, norm: RMSNorm):
    """Multi-head depth routing: split the (D,) query into self.H
    heads and run H independent softmaxes over the sources. This
    kernel is the ONLY change from attention residuals (Kimi 2025)."""
    V = torch.stack(sources)                    # [N, B, T, D]
    N, B, T, D = V.shape
    q = proj.weight.view(self.H, D // self.H)   # (D,) query -> H heads
    Vh = V.view(N, B, T, self.H, D // self.H)   # H slices per source
    logits = einsum('he,nbthe->nbth', q, norm(V).view_as(Vh))
    alpha = logits.softmax(0)                   # H softmaxes over
    depth
    mixed = einsum('n b t h, n b t h e -> b t h e', alpha, Vh)
    return mixed.reshape(B, T, D)

def forward(self, sources):  # IDENTICAL to attention residuals (Kimi
    2025)
    # --- Attention sublayer ---
    h = self.route(sources, self.attn_proj, self.attn_norm)
    sources.append(self.attn(norm(h)))   # append output as a new
    source

    # --- MLP sublayer ---
    h = self.route(sources, self.mlp_proj, self.mlp_norm)
    sources.append(self.mlp(norm(h)))    # replacement: append, don't
    add

    return sources
```

Figure 3: **MHAR pseudocode.** The forward pass is *identical* to attention residuals (Kimi Team, 2026): each sublayer reads a routed mix of the prior sources and *appends* its output as a new source. The *sole* change is the routing operation, which splits the d-dimensional query into H heads with independent depth softmaxes (zero added parameters); $H{=}1$ recovers attention residuals exactly.

adds *zero* parameters over single-head routing; H is a reshape, not a widening. *(ii) FLOP-matched.* The scoring and mixing in Eq. 3 touch each source coordinate exactly once, as in Eq. 2; the only overhead is computing H small softmaxes over the depth axis (length $\leq 2L{+}1$) instead of one, which is negligible next to the attention and MLP sublayers. *(iii) Strict generalization of attention residuals.* At $H{=}1$ the route reduces to the single shared query of Eq. 2 run through the identical forward, so the *single-head* method in our tables *is* the original attention-residual formulation of Kimi Team (2026)—making MHAR-vs-single-head a strict iso-parameter, iso-forward control.

Fused routing kernels. Depth routing is memory-bound: every sublayer re-reads the whole depth history ($O(L^2\,B\,T\,d)$ traffic per step) with no data reuse, and reference implementations pay this traffic several times over. Our fused, deterministic Triton kernels cut per-microbatch routing time by $2\times$ over the compiled PyTorch reference and $6.4\times$ over the eager reference ($\sim$30% end-to-end mid-training throughput on $8\times$H100; algorithm and verification in Appendix F).

3 Experiments

3.1 Setup

Models and training. We train decoder-only Transformers from scratch on our corpus (Appendix J.1)—a deduplicated, Nemotron-based mix (NVIDIA, 2025b) (quality-filtered, synthetic-, STEM-, and code-heavy), the same corpus used for the 8B mid-training experiments of §3.3 (composition in Appendix Table 15)—for 20K steps with the AdamW optimizer (Loshchilov & Hutter, 2019), a cosine learning-rate schedule with 1,000 warmup steps decaying to $0.1\times$ peak, and bfloat16. (A replication of the same four-method comparison on a second, web-generic corpus is reported in Appendix Table 13.) The three scales are: 100M ($d{=}512$, $L{=}12$, 8 heads, 4 KV heads, FFN 1536), 350M ($d{=}1024$, $L{=}24$, 16 heads, 8 KV heads, FFN 4096), and 1B ($d{=}1280$, $L{=}36$, 16 heads, 8 KV heads, FFN 5120), all with head dimension 64–80 and a Qwen3-style architecture (Qwen, 2025). The 100M models use sequence length 2048 and global batch 64; the 350M/1B models use

sequence length 1024 and global batch 32. Over the 20K-step budget this is $\approx$2.6 B training tokens at 100M and $\approx$0.66 B at 350M and 1B (roughly 26, 1.9, and 0.7 tokens per parameter)—deliberately far short of compute-optimal budgets at the larger scales; Appendix I repeats the comparison at $3\times$ this budget and finds the gaps preserved. We compare four methods at each scale: a standard Transformer (*baseline*), hyper-connections (Zhu et al., 2024), single-head attention residuals (H=1, i.e. the published attention-residual formulation of Kimi Team (2026)), and multi-head attention residuals (*MHAR*) with H=8 routing heads at every scale (equal to the KV-head count at 350M and 1B). Following Kimi Team (2026), all routing pseudo-queries are *zero-initialized*, so every depth softmax starts as a uniform average over its sources (the appendix replication uses randomly initialized queries; see the caveat there). All four methods share architecture, data, schedule, optimizer, and data-parallel degree, so the routing mechanism is the only difference. Table 1 reports all methods at a unified peak learning rate of 5×10^{-4}; a per-method learning-rate sweep (Appendix Table 13) shows the method ranking is not an artifact of sharing one rate. MHAR is parameter-matched to baseline up to the $O(d)$ routing queries and adds only 0.5–1.2% routing FLOPs (Table 4). Full training and reproducibility details (optimizer hyperparameters, tokenizer, positional scheme, precision, and the validation protocol) are given in Appendix J.

Metric. A single validation pass is noisy ($\sim\pm0.07$ at 1B; Appendix E) because each streaming pass samples a small window of the corpus, so we report the *tail-mean* of the last eleven evaluations (steps 15–20k) and compare methods *paired* (same data order within a scale), which resolves cross-method Δs far more finely. Tables report one fixed seed (Appendix J); a *document-disjoint* re-evaluation of all four models trained on our corpus reproduces the Table 1 ranking and gap sizes (Table 11).

3.2 PRETRAINING

Table 1: From-scratch validation loss (↓; tail-mean, §3) after 20K steps on our corpus (Appendix J.1) at a unified peak learning rate of 5×10^{-4}; Δ is relative to the baseline. The last column is an *over-splitting control* (H=16, past the H=4–8 optimum). Second-corpus replication: Appendix Table 13.

Scale	d/L/KV	Baseline	Hyper-conn.	Single-head	MHAR (H=8)	MHAR (H=16)
100M	512/12/4	3.031	2.999 (−0.032)	2.970 (−0.060)	**2.969** (−0.061)	2.996 (−0.035)
350M	1024/24/8	2.997	2.931 (−0.066)	2.876 (−0.121)	**2.848** (−0.149)	2.895 (−0.102)
1B	1280/36/8	2.894	2.807 (−0.086)	2.759 (−0.134)	**2.754** (−0.140)	2.825 (−0.069)

Comparison to hyper-connections. A learned alternative to the additive residual is hyper-connections (Zhu et al., 2024), which replace the single stream with n parallel streams mixed by per-layer learned depth and width connections. We use the *static* (input-independent) variant at expansion rate n=4, initialized at the standard residual (stream-mixing matrix at identity, width connection one-hot on the first stream, uniform depth softmax) plus 0.02 symmetry-breaking noise. Trained in the same sweep, they improve over the baseline at all three scales (−0.032/−0.066/−0.086; Table 1), but MHAR achieves lower loss at every scale (by 0.029, 0.083, and 0.053 at 100M/350M/1B). In these comparisons, learned depth routing within a single stream is more effective than learned connections across streams, with lower parameter and memory-traffic costs (Table 4).

Over-splitting control (H=16). The last column of Table 1 reshapes the same query into H=16 heads, past the H=4–8 optimum: it reduces MHAR's gain by a third to a half at 350M/1B ($-0.149 \rightarrow -0.102$, $-0.140 \rightarrow -0.069$), a cost that grows with scale (+0.026/+0.047/+0.071 over the optimum), and has higher loss than single-head routing at every scale—loss is U-shaped in H (§4).

Compute-equivalent gain. The loss deltas of Table 1 can also be expressed as compute-equivalent gain (Davidson et al., 2023): the factor by which baseline training FLOPs would have to grow to match MHAR's loss, under a power law $L=AC^{-\alpha}$ fitted to the lower envelope of the three baseline training curves in the style of Kaplan et al. (2020) (α=0.089, a *conservative* exponent since the cosine schedule makes mid-run points pessimistic). Because MHAR's routing adds only 0.5–1.2% FLOPs, the loss gain translates into a 1.3×/1.8×/1.7× compute equivalence at 100M/350M/1B (1.15–2.6× across plausible exponents; Appendix I).

Table 2: Zero-shot downstream evaluation (single run) of the models trained on our corpus in Table 1. Each model is evaluated at its training context (100M: 2048; 350M/1B: 1024), so perplexity is comparable only *within* a scale; HellaSwag uses 200 samples (±3pt noise).

Scale	Method	WikiText-2 PPL ↓	LAMBADA ↑	HellaSwag ↑
100M	Baseline	60.0	9.6%	35.0%
100M	Hyper-conn.	57.0	10.2%	33.0%
100M	Single-head	**54.5**	11.6%	**36.0%**
100M	MHAR	**54.5**	**12.0%**	33.0%
350M	Baseline	64.9	8.8%	34.5%
350M	Hyper-conn.	58.4	10.6%	33.0%
350M	Single-head	53.8	**14.4%**	33.5%
350M	MHAR	**52.0**	12.6%	**36.0%**
1B	Baseline	56.4	8.8%	30.5%
1B	Hyper-conn.	49.6	12.2%	34.5%
1B	Single-head	**45.0**	14.0%	**35.0%**
1B	MHAR	45.4	**16.0%**	33.0%

Downstream evaluation. To check that the loss improvement is not an artifact of the training distribution, we evaluate all four models trained on our corpus zero-shot (single run) on held-out WikiText-2 perplexity, LAMBADA (Paperno et al., 2016), and HellaSwag (Zellers et al., 2019) (Table 2). The benchmarks are web-derived, so this is *cross-distribution* transfer, and it tracks the validation ranking: MHAR improves perplexity ($-9\%/-20\%/-19\%$) and LAMBADA ($+2.4/+3.8/+7.2$ points) over the baseline at every scale, the two attention-residual variants lead every perplexity column, and hyper-connections sit in between. On an extended suite (ARC-Easy, PIQA, WinoGrande, BoolQ; Appendix Table 12), the methods are indistinguishable within the ± 2.2-point sampling noise. The improvements in perplexity and LAMBADA do not extend to measurable gains in multiple-choice accuracy in these experiments.

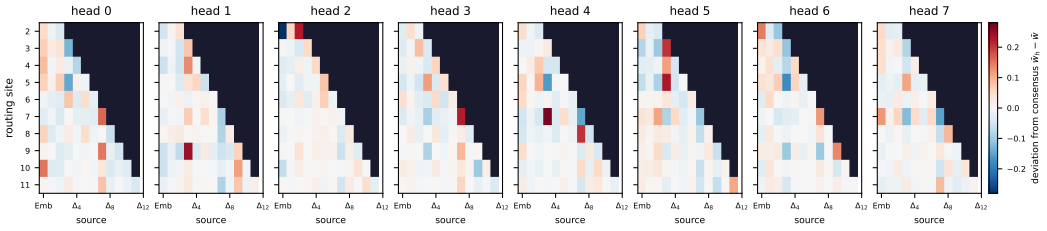

Figure 4: **Distinct depth-routing patterns across heads.** Each head's token-averaged deviation $\bar{w}_h - \bar{w}$ from the head-consensus routing at the first ten routing sites of the trained 1B MHAR model (dark cells: source not yet available). Deviations reach ± 0.28 vs. 0.067 under a matched-norm random query, replicate on disjoint evaluation text (r=0.77), and are near-uncorrelated across heads.

Effect of multiple routing heads. A single routing query shares one depth distribution across all d channels—a forced compromise whose measured cost is driven chiefly by the softmax oversharpening that comes with width, with a residual from subspace disagreement (Appendix E). The cost is strongly *data-dependent*: on our corpus a zero-initialized single query already captures most of the routing gain (Table 1; MHAR−single-head $= -0.001/-0.028/-0.006$), whereas in the web-corpus replication single-head routing degrades from helpful at 100M to *worse than the plain residual* at 1B while MHAR keeps its full gain (Appendix Table 13). A temperature-matched control clarifies the source of the difference on our corpus: at 350M, single-head routing with its logits scaled by $1/\sqrt{8}$—matching the per-head softmax flatness of H=8 without splitting the read—matches MHAR's loss, so the residual increment on our corpus is a softmax-temperature effect rather than subspace independence (Appendix E). MHAR achieves the lowest validation loss at every tested scale on both corpora. Splitting the query preserves its parameter count, sets the routing temperature through the head dimension, and allows different subspaces to use different depth distributions. Figure 4 shows that the trained heads learn distinct depth distributions, with stable deviations from the mean routing distribution across heads.

Table 3: Downstream accuracy (↑) after 8B mid-training on our corpus (Appendix J.1; final EMA checkpoints; control and MHAR are schedule-matched).

	Task	Base	Plain CPT	+MHAR
General	MMLU	0.554	0.643	**0.645**
	GPQA	0.266	0.315	**0.346**
Math & Coding	GSM8K	0.190	0.470	**0.502**
	MATH	0.053	**0.191**	**0.191**
	HumanEval	0.122	0.409	**0.415**
	MBPP	0.148	0.386	**0.392**

3.3 MID-TRAINING

The from-scratch experiments isolate the routing mechanism at 100M–1B. We next evaluate MHAR in 8B *mid-training* (continual pretraining), where a pretrained base model is trained further on a capability-oriented corpus. We use an identity-preserving conversion to add multi-head attention residuals to the pretrained Transformer, then decay the learning rate to convergence.

Data construction. We assemble our English-only training corpus of $\approx 1.9\,\mathrm{T}$ tokens from public deduplicated, quality-filtered corpora. The blend emphasizes synthetic, STEM, and code data, building on the Nemotron-CC data pipeline (Su et al., 2024). The blend draws on the Nemotron-CC-v2 web crawl and its synthetic rephrasing and diverse-QA variants (NVIDIA, 2025b), the Nemotron pretraining SFT/STEM/code sets (NVIDIA, 2025c), Nemotron-CC-Math (NVIDIA, 2025a), FinePDFs restricted to its top three of twenty quality bins (Hugging Face, 2025a), Stack-Edu code (Hugging Face, 2025b), arXiv, and English Wikipedia, shuffled at the document level into 2,048 uniform shards. Every document retains its source tag. Table 15 (Appendix J.1) reports per-source text-byte shares over a uniform random sample of 48 shards.

Base model and schedule. We start from Marin-8B[1] (checkpoint version phoenix), a Llama-3-8B-architecture model (8.03B parameters, d=4096, L=32, 32/8 grouped-query heads) released at its Warmup–Stable–Decay *pre-decay plateau*—the natural resume point for decay-phase mid-training. The *plain-CPT* control cools from peak 4×10^{-4} to 4×10^{-6} over 9,500 steps at a global batch of $\approx 1.05\,\mathrm{M}$ tokens ($\approx 10\,\mathrm{B}$ tokens total, $\sim 0.5\%$ of the corpus); the MHAR run replicates the control's schedule exactly—the same peak rate, decay shape, batch, data order, and budget—giving a *schedule-matched* pair that isolates the routing effect. Optimizer and systems details (cold-start AdamW with re-warmup, FSDP, EMA, z-loss) are in Appendix J.

Identity-preserving conversion. We convert with the *delta* (additive-stream) form of attention residuals (Luo et al., 2026), which *adds* the routed term to the intact residual stream (h=$h_{\mathrm{res}} + \alpha\tilde{h}$, where h_{res} is the residual stream and $\tilde{h}$ the routed mixture) rather than *replacing* it as the replacement-based routing used in the from-scratch experiments does (Appendix F); closing the gate therefore leaves the pretrained computation untouched. The routed sources are *block-aggregated*: a learnable null source plus one delta per block of four consecutive layers (eight blocks for the 32-layer model), so each routing site attends over at most nine sources with H=8 heads. MHAR is grafted onto the pretrained model with a zero-initialized output gate (α=0 at init), so the converted model is *numerically identical* to the base at step 0 (maximum logit deviation 0 in fp32), and the routing opens during mid-training. This lets us continue-train an existing checkpoint without a loss spike: on the schedule-matched pair (identical seed and data order) both runs start from the same loss (1.8616 vs 1.8615 at the first logged step), and the per-step gap stays 36× below batch noise across all 9,500 steps.

Downstream protocol. We evaluate six capability-oriented tasks with the LM Evaluation Harness (Gao et al., 2023), grouped as *General* —MMLU (Hendrycks et al., 2021a) (57-subject average) and GPQA (Rein et al., 2024) (main split, 0-shot)—and *Math & Coding*: GSM8K (Cobbe et al., 2021) (5-shot, strict exact match), MATH (Hendrycks et al., 2021b) (Minerva 4-shot (Lewkowycz

[1] https://huggingface.co/marin-community/marin-8b-base

Table 4: **Training speed and memory** at the settings of Table 1: MHAR with compiled PyTorch reference kernels vs. our fused kernels; throughput relative to the same-scale baseline. Single-head routing is cost-identical to MHAR. Full breakdown: Appendix G.

	Throughput (vs. base)			**Peak mem.** (GB)		
Method	100M	350M	1B	100M	350M	1B
Baseline Transformer	$1.00\times$	$1.00\times$	$1.00\times$	41.5	19.4	19.0
MHAR	$0.54\times$	$0.32\times$	$0.23\times$	52.4	40.1	47.5
MHAR + fused kernels (ours)	$\mathbf{0.88}\times$	$\mathbf{0.71}\times$	$\mathbf{0.55}\times$	**42.0**	**20.0**	**20.1**

Table 5: **Routing-operation speedup**: wall-clock of all routing calls in one microbatch (forward+backward, bf16, one H100); the 8B column is the mid-training delta variant. End-to-end gains (Table 4) are smaller since routing is only part of a step (Appendix G).

Routing per microbatch (ms)	**100M**	**350M**	**1B**	**8B (delta)**
Compiled PyTorch	30.2	53.1	153.2	651
Fused Triton (ours)	**5.7**	**11.4**	**42.4**	**323**
Speedup	$5.3\times$	$4.7\times$	$3.6\times$	$2.0\times$

et al., 2022)), and HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) (pass@1, greedy). MATH is scored with a format-robust checker rather than the harness's template-strict extractor: annealed models increasingly answer in bold style while abandoning the few-shot "Final Answer" template, which strict extraction mistakes for failure despite mathematically correct solutions. For generative tasks we report paired per-item exact McNemar tests between the schedule-matched arms. Table 3 reports final (EMA) checkpoints.

Mid-training on our corpus produces large downstream gains (GSM8K more than doubles, $19.0 \rightarrow 47.0$; MATH more than triples, $5.3 \rightarrow 19.1$; MMLU rises $+8.9$) while the schedule-matched comparison isolates the contribution of routing: a gain of $+3.2$ GSM8K (123 vs. 81 discordant items, paired exact McNemar p=0.004; 95% paired-bootstrap CI $[+1.1, +5.3]$ points) and $+3.1$ GPQA (27 vs. 13, p=0.038; CI $[+0.5, +6.0]$), with MMLU, MATH, and code statistically unchanged. The routing gain at this scale is thus modest and concentrated on GSM8K and GPQA; how it interacts with a more aggressive re-training schedule is left to future work. We will release the mid-trained MHAR model and its schedule-matched control upon publication.

3.4 Parameter and compute cost

Parameter cost. MHAR adds only $+0.02\%$ parameters (per-layer routing queries and RMSNorms: $+100$K at 350M, $+187$K at 1B), with validation-loss changes of -0.06 to -0.15 nats relative to the baseline (Table 1). Because the head split only reshapes the query (§2), MHAR and single-head routing have identical parameters, throughput, and peak memory (Appendix G), so every MHAR-vs-single-head comparison is iso-wall-clock as well as iso-parameter.

Equal-compute comparison. The one comparison that is *not* iso-compute is the attention-residual family vs. the plain baseline: routing over the list of past sublayer outputs has a mechanism cost inherited from Kimi Team (2026), so Table 1 compares at equal steps. Our fused kernels (Appendix F; Table 5) remove most of this cost: MHAR trains at $0.88/0.71/0.55\times$ baseline throughput at near-baseline memory (Table 4). Giving each baseline the matching wall-clock budget (22,700/28,200/36,400 steps at 100M/350M/1B, cosine schedule stretched, all else as in Table 1) shrinks MHAR's advantage from $-0.061/-0.149/-0.140$ to $-0.006/-0.080/-0.018$: under this deliberately under-trained budget, equal-cost superiority is clear at 350M but within the single-seed noise floor at 100M and 1B. The comparison matches GPU-hours for the same hardware and software configuration rather than FLOPs.

Compute-optimal training budget. The 20K-step budget is deliberately under-trained, so the gain could be a low-token-budget artifact (Hoffmann et al., 2022). We retrain a matched baseline–MHAR pair at 350M and 1B at the Chinchilla-optimal ≈ 20 tokens per parameter (7.0B and 21.6B tokens, $10.7\times$ and $33\times$ the main-table budget; cosine schedule stretched, all else fixed). The loss gap narrows

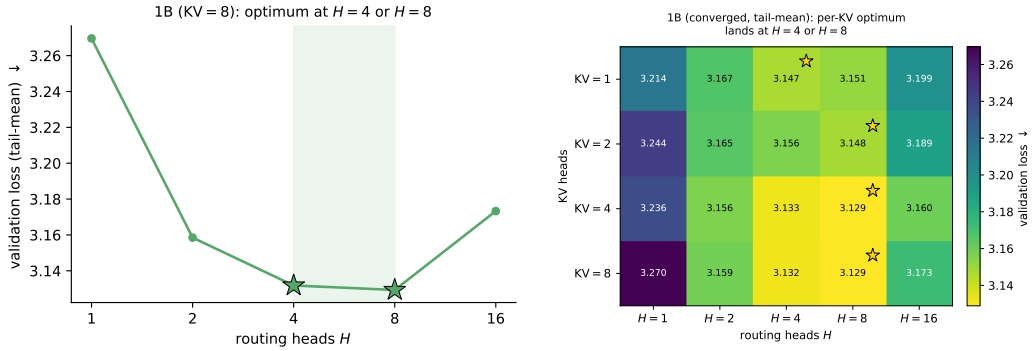

Figure 5: **Head count at 1B: a flat H=4–8 optimum. (a)** The KV = 8 row (tail-mean): loss falls from single-head (H=1) into a flat basin at H=4–8—the optimum, 3.132 and 3.129, essentially tied—then rises past it: H=16 (3.173) gives back roughly a third of the multi-head gain, though it stays well below H=1. **(b)** The full KV×H grid: multi-head routing helps at every KV, the per-KV optimum (gold stars) lands at H=4 or H=8, and the H=16 column overshoots the basin in every row while staying below H=1.

but remains nonzero—$-0.149 \rightarrow -0.119$ at 350M and $-0.140 \rightarrow -0.081$ at 1B (≥ 50 standard errors; Appendix Table 10)—and the 1B gap is flat from 10% of the budget onward. The improvement is thus not an under-training artifact, though it shrinks with budget and these runs are single-seed.

4 Ablations

The head count has a two-sided optimum. Figure 5a sweeps the routing-head count at 1B (KV = 8) with everything else fixed: loss falls sharply from H=1 (3.270) into a flat basin at the KV granularity, reaching its optimum at H=4 (3.132) and H=8 (3.129)— the two are essentially tied, within tail-mean noise—then rises past it: H=16 (3.173) sits 0.044 above the optimum, beyond the tail-mean noise ($\sim \pm 0.02$), and the same overshoot appears in every KV row of the grid (+0.031–0.052). The two failure modes are granularity mismatches with the stream's consumers, though asymmetric ones: too few heads force unrelated subspaces through one depth distribution (costing 0.14 at H=1), while too many split what a KV group consumes coherently (costing 0.03–0.05 at H=16, still well below single-head). The size of the overshoot is corpus-dependent: on our corpus (Table 1) it is larger and H=16 has higher loss than single-head routing. The lowest loss occurs at H=4 or H=8 at every tested scale, with little difference between these settings. We use H=8 for the large-scale models.

5 Conclusion

Multi-head attention residuals address softmax sharpening with increasing width and the use of a shared depth distribution across feature subspaces. Each subspace receives a separate routing head whose temperature is determined by its dimension, without adding parameters relative to single-head attention residuals. Trained from scratch at 100M–1B, MHAR achieves the lowest loss among four methods in every setting on both corpora we test. The gains extend to held-out perplexity and LAMBADA and remain under a document-disjoint re-evaluation. The head count has a flat H=4–8 optimum (we adopt H=8). Fused routing kernels bring training to 0.55–0.88× baseline throughput at near-baseline memory, and an identity-preserving conversion extends MHAR to 8B mid-training (+3.2 GSM8K, paired p=0.004). Understanding the H=4–8 optimum and closing the equal-compute gap at the largest scales remain open questions; we discuss limitations in Appendix B.

AI use statement

In this work, we used generative AI tools (including Claude) for three tasks that require disclosure: helping *implement methods*—writing, running, and debugging the training, evaluation, and fused-kernel code; providing *feedback on the experiments*, including an internal reviewer-style pass that re-derived reported numbers from raw logs and flagged missing artifacts; and helping *produce the*

training data: our corpus used for all main experiments (Appendix J.1) was built with AI assistance, and more than half of its text bytes are LLM-generated or LLM-rewritten data from public releases (about 38% Nemotron-CC synthetic rephrasings and diverse QA, plus about 15% Nemotron SFT-style, synthetic math/code and rewritten-Wikipedia subsets; Table 15). AI agents also helped plan, monitor, and summarize experiment sweeps; the method, its hypotheses, and the final interpretation of results are the authors'. We have not used generative AI tools to propose the method or its hypotheses; the remaining required-disclosure tasks (theoretical frameworks, mathematical claims or proofs, translation, and qualitative data analysis) are not applicable to this work. Additionally, we used generative AI tools to draft parts of the text (including appendix analyses written up from our measurements), edit and condense the manuscript, reformat it into the conference template, and check and correct references. We checked the reported numbers against raw training and evaluation artifacts. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Reproducibility statement

Section 2 defines MHAR completely (it adds no parameters over attention residuals). Section 3 and Appendix J give model configurations, optimizer and schedule settings, tokenizer, data pipeline, and evaluation protocol for every run; Appendix J.1 describes how we construct our corpus from public data and reports its measured composition (Table 15). Appendix I reports seed and validation-split robustness, Appendix F describes the fused Triton routing kernels, and Appendix G their compute and memory cost. Training code, the fused Triton kernels, and their correctness tests are included (anonymized) in the supplementary material; the mid-trained 8B checkpoints will be released upon publication.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *EMNLP*, 2023.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Thomas Bachlechner, Bodhisattwa Prasad Majumder, Huanru Henry Mao, Garrison W Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. *UAI*, 2021.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

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

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022.

Tom Davidson, Jean-Stan Denain, Pablo Villalobos, and Guillem Bas. AI capabilities can be significantly improved without expensive retraining. *arXiv preprint arXiv:2312.07413*, 2023.

Nelson Elhage, Neel Nanda, Catherine Olsson, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *JMLR*, 2022.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 2023. URL https://zenodo.org/records/10256836.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Ruining He, Anirudh Ravula, Bhargav Kanagal, and Joshua Ainslie. Realformer: Transformer likes residual attention. *ACL Findings*, 2021.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *ICLR*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *NeurIPS Datasets and Benchmarks Track*, 2021b.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

Hugging Face. Finepdfs. https://huggingface.co/datasets/HuggingFaceFW/finepdfs, 2025a.

Hugging Face. Stack-edu. https://huggingface.co/datasets/HuggingFaceTB/stack-edu, 2025b.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kimi Team. Attention residuals. *arXiv preprint arXiv:2603.15031*, 2026.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems*, 2022.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *ICLR*, 2019.

Cheng Luo, Zefan Cai, and Junjie Hu. Delta attention residuals. *arXiv preprint arXiv:2605.18855*, 2026.

Gaurav Menghani, Ravi Kumar, and Sanjiv Kumar. Laurel: Learned augmented residual layer. *arXiv preprint arXiv:2411.07501*, 2024.

NVIDIA. Nemotron-cc-math-v1. https://huggingface.co/datasets/nvidia/Nemotron-CC-Math-v1, 2025a.

NVIDIA. Nemotron-cc-v2. https://huggingface.co/datasets/nvidia/Nemotron-CC-v2, 2025b.

NVIDIA. Nemotron-pretraining-sft-v1. https://huggingface.co/datasets/nvidia/Nemotron-Pretraining-SFT-v1, 2025c.

Matteo Pagliardini, Amirkeivan Mohtashami, François Fleuret, and Martin Jaggi. Denseformer: Enhancing information flow in transformers via depth weighted averaging. *arXiv preprint arXiv:2402.02622*, 2024.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *ACL*, 2016.

Guilherme Penedo, Hynek Kydlíček, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *arXiv preprint arXiv:2406.17557*, 2024.

Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. *arXiv preprint arXiv:2505.06708*, 2025.

Qwen. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Anton Razzhigaev, Matvey Mikhalchuk, Elizaveta Goncharova, Nikolai Gerasimenko, Ivan Oseledets, Denis Dimitrov, and Andrey Kuznetsov. Your transformer is secretly linear. In *ACL*, 2024.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof Q&A benchmark. In *Conference on Language Modeling (COLM)*, 2024.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V Le, Geoffrey E Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *ICLR*, 2017.

Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. *NeurIPS*, 2015.

Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-CC: Transforming Common Crawl into a refined long-horizon pretraining dataset. *arXiv preprint arXiv:2412.02595*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. *NeurIPS*, 2016.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. Deepnet: Scaling transformers to 1,000 layers. *arXiv preprint arXiv:2203.00555*, 2022.

Da Xiao, Qingye Meng, Shengping Li, and Xingyuan Yuan. Muddformer: Breaking residual bottlenecks in transformers via multiway dynamic dense connections. *ICML*, 2025.

Zhenda Xie et al. mHC: Manifold-constrained hyper-connections. *arXiv preprint arXiv:2512.24880*, 2025.

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. *ICML*, 2020.

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

Yifan Zhang. Residual stream duality in modern transformer architectures. *arXiv preprint arXiv:2603.16039*, 2026.

Yifan Zhang, Yifeng Liu, Mengdi Wang, and Quanquan Gu. Deep delta learning. *arXiv preprint arXiv:2601.00417*, 2026.

Zhanchao Zhou, Tianyi Wu, Zhiyun Jiang, Fares Obeid, and Zhenzhong Lan. Value residual learning. *arXiv preprint arXiv:2410.17897*, 2024.

Defa Zhu et al. Hyper-connections. *arXiv preprint arXiv:2409.19606*, 2024.

A ADDITIONAL ABLATIONS

Routing-head count and convergence. How many routing heads help, and where the optimum sits, depends on how converged the router is. In a deliberately *under-trained* grid (100M on the web corpus at its tuned LR 1×10^{-3} but only 5K steps, vs. the 20K used elsewhere; Figure 7), the largest difference occurs in the most over-subscribed row, KV $= 1$: there the single shared query ($H{=}1$) is worst and $H{=}8$ best, a *monotone* 0.028 drop, well above the ~ 0.006 single-seed noise floor (the higher-KV rows are within noise). In this under-trained grid, increasing the head count does not measurably increase loss. Trained to convergence, the optimum instead *saturates* into a flat $H{=}4$–8 basin: the full grids at 100M (Appendix D) and 1B (Figure 5b) put every per-KV optimum at $H{=}4$ or $H{=}8$ (the over-subscribed KV $\in \{1, 2\}$ rows still prefer more heads even at convergence), which supports $H{=}8$ at scale.

Head-aligned routing. A natural hypothesis is that routing should be *aligned* to the model's attention structure rather than to arbitrary coordinate slices. We test this with ***MHAR-HW***, a head-*wise* variant that gives each grouped-query-attention KV head group its own full-width routing query and feeds that group's routed mixture to its $Q/K/V$ projections; with one full-width query per group, its routing cost is $G\times$ that of MHAR (G the number of KV groups) but still negligible next to the sublayers. Table 6 compares the two across scales: head-alignment yields *no detectable* improvement over arbitrary subspace routing: all differences are within the ± 0.07 single-pass eval noise (the $+0.016$ at 1B is a single final-step eval). Being single-seed, this is a *failure to find* a benefit from head-alignment, not evidence of equivalence.

Table 6: Head-alignment ablation (FineWeb-Edu; validation loss ↓, final-step eval): routing over arbitrary subspaces (MHAR) vs. aligned to KV head groups (MHAR-HW), with matched head count; single-seed, from a separate matched-recipe batch, so read only the within-table Δ (MHAR-HW$-$MHAR).

Scale	MHAR	MHAR-HW	Δ loss
100M	3.330	3.333	$+0.003$
350M	3.225	3.223	-0.002
1B	3.213	3.229	$+0.016$

B LIMITATIONS

(i) **Scale, budget, and seeds.** The from-scratch evidence spans 100M–1B on two corpora (§3). The main table uses a 20K-step budget, but the gaps are not a short-schedule artifact: a $3\times$-budget control preserves them and a Chinchilla-optimal ≈ 20 tokens/parameter retraining still leaves $-0.119/-0.081$ at 350M/1B (Table 10). Caveats remain: a 7B run under the web recipe gains only -0.010 (within noise) and is excluded from the scaling claim, the MHAR$-$baseline gain is non-monotonic in scale, and the compute-optimal and several grid/ablation cells are single-seed. The three scales also differ in depth, KV heads, sequence length, batch size, and tokens per parameter, so the scale trend is not attributable to width alone; only the single-seed, fixed-depth widening control at 100M (Appendix H) isolates width. (ii) **Equal compute.** The headline claims are *step-matched*. At a compute-optimal token budget the advantage persists (Table 10), but the orthogonal *equal-wall-clock* reading—which charges MHAR for its lower throughput—is single-seed, matches GPU-hours rather than FLOPs, and leaves the margin clear only at 350M ($-0.006/-0.080/-0.018$); an equal-wall-clock run at a compute-optimal budget is untested. (iii) **Validation.** Streaming validation re-shuffles the training corpus; the document-disjoint re-evaluation (Table 11) reproduces the gaps but is a single window set, and all downstream suites are single-run. (iv) **Mechanism.** Temperature-matched controls attribute the dominant share of the gain to per-head softmax flatness (Appendix E); subspace disagreement is a partially-supported hypothesis (correlational probe, residual benefit at the single-seed noise floor), we offer no theory for the $H{=}4$–8 optimum, and the head-alignment ablation is within noise, so "arbitrary subspace heads suffice" is unfalsified rather than established. (v) **Systems.** The $O(L^2\,B\,T\,d)$ re-read of the depth history is inherent to dense depth routing; the fused kernels reduce but do not remove it, and decode-time cost is unmeasured. (vi) **Retrofit.** The 8B evidence is a single schedule-matched pair, single-seeded, using the *delta* form over $N{=}9$ block-aggregated sources rather than the full

mechanism; its gains concentrate on GSM8K and GPQA. (vii) **Baselines.** We train three alternatives to the plain residual (hyper-connections, single-head attention residuals, and MHAR). DenseFormer, MUDDFormer, and delta routing (Luo et al., 2026) are discussed (§C) but not trained as from-scratch baselines under our recipe (the 8B retrofit uses the delta form for MHAR itself, not as a competitor), so MHAR's ranking against them is untested.

C Related Work

Residual and dense connections. Residual connections (He et al., 2016) and their gated predecessor, highway networks (Srivastava et al., 2015), enable very deep training by providing additive identity paths; Veit et al. (2016) characterize residual networks as ensembles of paths of varying length. DenseNets (Huang et al., 2017) instead concatenate all previous feature maps, giving each layer direct access to every earlier one. Our sources list $\mathcal{S}$ is dense in this spirit, but rather than concatenating (which grows width) or summing (which is the standard residual stream), we *route* over the sources with a learned softmax, keeping width fixed while making each past output individually addressable.

Cross-layer aggregation in Transformers. Several works move beyond the single additive stream. DenseFormer (Pagliardini et al., 2024) adds a learned depth-weighted average of all previous block outputs; MUDDFormer (Xiao et al., 2025) learns multi-way, input-dependent dense connections between layers; RealFormer (He et al., 2021) carries a residual path on the attention scores. LAuReL (Menghani et al., 2024) generalizes the residual connection itself with lightweight learned linear maps, and value-residual learning (ResFormer) (Zhou et al., 2024) adds a residual on the attention *values* from the first layer; both enrich what the stream carries, whereas we enrich how each sublayer *reads* it. ReZero (Bachlechner et al., 2021) and DeepNet (Wang et al., 2022) instead rescale the standard residual branch for stable deep training. Attention residuals differ by replacing the additive stream with an explicit softmax *retrieval* over past sublayer outputs: unlike DenseFormer's fixed depth-weighted average, the retrieval is input-adaptive and—crucially for us—*splittable* into per-subspace heads. Making that retrieval multi-headed at no parameter or FLOP cost is our contribution.

Routing over layer outputs. Closest to our setting are attention residuals (Kimi Team, 2026), which introduce a learned attention over the history of layer outputs. We build directly on this formulation and identify its single routing query as a bottleneck: splitting it into independent per-subspace heads (Eq. 3) improves quality while staying parameter- and FLOP-matched. Its one multi-head ablation ($H{=}16$, Block AttnRes) hurts (1.752 vs. 1.746), consistent with our finding that $H{=}16$ over-splits while $H{=}4$–8 helps. Hyper-connections (Zhu et al., 2024) generalize residual connections into a learnable width-n set of parallel streams, and manifold-constrained hyper-connections (Xie et al., 2025) add geometric constraints to their mixing; these enrich the *connection topology* between layers, whereas we keep a single stream and enrich the *routing* that reads the depth history—our heads act on routing distributions, not on separate streams. MHAR outperforms hyper-connections at matched recipe across all three scales (Table 1). Concurrent work changes a different part of the mechanism: Delta Attention Residuals (Luo et al., 2026) keep the residual stream and *add* the routed mixture of per-sublayer deltas to it, rather than replacing the stream with the mixture as attention residuals do—while Deep Delta Learning (Zhang et al., 2026) edits the residual operator itself with a gated rank-1 update, and Zhang (2026) analyze residual-stream structure. We instead split the routing query across feature subspaces, which our temperature-matched controls show acts chiefly by flattening the width-sharpened routing softmax, with a small residual from per-subspace independence (Appendix E); the two fixes are complementary.

Conditional computation and mixtures. Mixture-of-experts routing (Shazeer et al., 2017; Fedus et al., 2022) and mixture-of-depths (Raposo et al., 2024) also use learned routing, but route *tokens* to experts or layers to save or allocate compute. Our routing is over the *depth axis* of already-computed sublayer outputs and is applied to every token; it changes what each sublayer reads, not which tokens are processed.

Multi-head mechanisms and interpretability. Multi-head attention (Vaswani et al., 2017) lets different heads attend to different token-level patterns; we apply the same intuition one level up, letting different heads attend to different *layers*. Mechanistic analyses of the residual stream as

a shared communication channel that many components read from and write to (Elhage et al., 2021)—and evidence that transformer layers act near-linearly on the stream (Razzhigaev et al., 2024)—motivate giving different subspaces independent read access to depth, which is precisely what multi-head routing provides. Methodologically, validating a lightweight, parameter-cheap architectural change through systematic from-scratch experiments across scale follows recent studies of attention modifications such as gated attention (Qiu et al., 2025).

D ROUTING-HEAD SWEEP AT 100M

Figure 6 shows the full 4×4 routing-head sweep at 100M ($H, \mathrm{KV} \in \{1, 2, 4, 8\}$), the converged counterpart of the under-trained grid in Figure 7. These grids train on the web replication corpus (FineWeb-Edu, randomly initialized routing queries; Appendix I), so their absolute losses are not comparable to those on our corpus in Table 1. Multi-head routing helps at every KV setting; the $\mathrm{KV} = 4$ and $\mathrm{KV} = 8$ optima sit on the $H =$ KV diagonal, while the over-subscribed $\mathrm{KV} \in \{1, 2\}$ rows prefer more heads, up to H=8. The same sweep at 350M (LR 5×10^{-4}, 20K steps) again puts the $\mathrm{KV} \geq 4$ optima on the diagonal. At 1B (Figure 5b) the grid argmin is the off-diagonal $\mathrm{KV} = 4$, H=8 (3.129), but the four lowest cells (KV4/H8, KV8/H8, KV8/H4, KV4/H4) lie within ~ 0.004 of one another, inside the tail-mean noise of these single-seed sweeps; H=8 at 1B is thus within noise of optimal rather than the strict grid minimum.

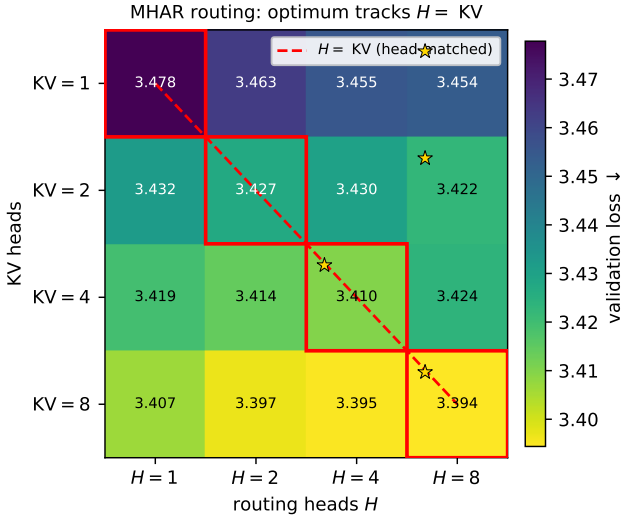

Figure 6: Validation loss of MHAR at 100M as a function of routing heads H (x) and KV heads (y). Lower is better. Gold stars mark the best H per KV; the red dashed line is the head-matched diagonal $H =$ KV. The 100M architecture of Table 1, trained on FineWeb-Edu at 5×10^{-4}.

E MECHANISTIC ANALYSIS OF SINGLE-HEAD ROUTING

We probe the web-data checkpoints (Table 13), where single-head routing crosses from helpful at 100M to harmful at 1B while MHAR keeps its gain. A single d-dimensional query routes the entire width through one softmax α (Eq. 2), serving every subspace with one depth distribution; we call the resulting penalty a *forced compromise*. We consider two candidate factors: (i) how differently subspaces would route if freed, which grows with model *width* and which we isolate below, and (ii) how *collinear* the sources are, which we do not cleanly isolate. The temperature-matched controls at the end of this appendix identify a third factor that accounts for most of the measured *loss* difference: the shared softmax's logit scale grows with the width it spans, over-sharpening the read, which the multi-head split also removes.

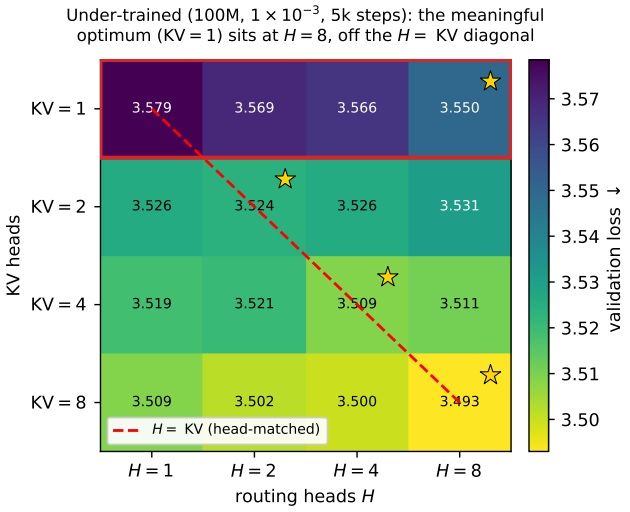

Figure 7: **Under-trained 100M KV$\times H$ grid** (web replication, 1×10^{-3}, only 5K steps; tail-mean 3500–5000)—the under-trained counterpart of Figure 6. In this grid, increasing the head count does not measurably increase loss. The largest difference occurs in the row KV $= 1$ (red outline; a monotone 0.028 drop from H=1 to H=8, above the ~ 0.006 single-seed noise floor), while the higher-KV rows are within noise. Gold stars: per-KV optimum; red dashed line: $H =$ KV.

Probe of trained routing queries. We measure both factors directly on the trained single-head (H=1) checkpoints, with *no* new training (Table 7). At each routing site we split the trained query $\mathbf{q}$ into S contiguous slices and compute, for each slice, the depth distribution it would prefer alone, $a_s = \mathrm{softmax}_n \langle \mathbf{q}_{[s]}, \mathrm{RMSNorm}(\mathbf{s}_n)_{[s]} \rangle$, relative to the shared α the single softmax actually uses. The mean $\mathrm{KL}(a_s \,\|\, \alpha)$ is the latent *width-disagreement*. We use three controls.

(i) The disagreement is learned, not geometric. A matched-norm *random* query produces only ~ 0.03–0.10 of the disagreement, versus 0.27–0.70 for the trained query, so the signal is overwhelmingly learned rather than an artifact of source geometry. The learned excess (trained minus null) grows *monotonically* across all four scales ($0.235 \to 0.281 \to 0.512 \to 0.606$ from 100M to 1B) and is largest at 1B—exactly the scale where single-head routing *regresses below baseline*, so the probe is measured directly at the regression scale. *(ii) Width is the driver.* Widening at *fixed* depth and KV-head count (d from 512 to 768, L=12, four KV heads) raises the trained-query KL by 14% ($0.273 \to 0.311$) while the null stays flat, so the learned excess rises 20% ($0.235 \to 0.281$)—and here d alone changes. *(iii) Not a "more sources" artifact.* A matched-source-count control leaves a $\sim 2\times$ median KL growth, and contiguous vs. head-interleaved slicing give identical results (echoing the head-alignment null of Table 6). Source collinearity is *non-monotonic* across scale—it rises from 100M to 350M ($0.060 \to 0.114$) but then *drops* at 1B (0.087), partly a dimension effect (cosine shrinks at larger d)—so it is *not* a clean cross-scale driver.

The same widening also doubles MHAR's loss advantage over single-head routing ($-0.010 \to -0.020$; single-seed, Appendix H), so disagreement and loss co-vary. *Scope:* the slices are sub-parts of *one* trained query rather than independently trained routers, so the KL is an upper-bound proxy for the disagreement of fully independent heads.

Temperature-matched single-head control. Splitting the query changes two things at once: each subspace routes *independently*, and each head's logits are computed from d/H coordinates rather than d, so its softmax is flatter (an effective temperature of roughly $\sqrt{H}$). To separate the two we trained a *temperature-matched single head* at 350M on our corpus: the unmodified single-head route with its logits scaled by a fixed $1/\sqrt{8}$ (applied through the query, so gradients scale consistently), matching the per-head logit scale of H=8 without splitting the read; everything else is identical to Table 1. On our corpus, the temperature-matched single head reaches 2.847 (tail-mean)—indistinguishable from MHAR's 2.848 and 0.028 below plain single-head routing. A same-data-order paired decomposition

Table 7: Direct probe of the trained *single-head* routing queries (no new training; $S{=}4$ slices). *Width-disagreement* is the mean $\mathrm{KL}(a_s \,\|\, \alpha)$ between each query slice's preferred depth distribution and the shared α; a matched-norm *random* query gives the null; the learned excess is trained$-$null. The 124M column widens d at *fixed* depth, KV heads, and source count N. Source collinearity is the mean pairwise cosine of the $N{=}2L{+}1$ sources.

Probe quantity ($S{=}4$); $d/L/\mathrm{KV}$	**100M** $512/12/4$	**124M** *(width ctrl)* $768/12/4$	**350M** $1024/24/8$	**1B** $1280/36/8$
Width-disagreement KL (trained)	0.273	0.311	0.570	0.701
random-query null	0.038	0.030	0.058	0.095
learned excess (trained$-$null)	**0.235**	**0.281**	**0.512**	**0.606**
Source collinearity (mean cos)	0.060	0.063	0.114	0.087
N sources ($= 2L{+}1$)	25	25	49	73

attributes $-0.025{\pm}0.001$ of that gap to the temperature alone and -0.003 to differences in the routing implementation, leaving no measurable residual attributable to subspace independence. The increment of MHAR over single-head routing on our corpus (Table 1) is therefore a *softmax-temperature* effect, not evidence of subspace disagreement. We repeated this control on web data at 1B, where single-head routing regresses below the plain baseline and MHAR's advantage is largest. Temperature scaling reverses the regression: the $1/\sqrt{8}$-scaled single head reaches 3.152 (tail-mean, using the same training settings as the routing-head sweep), 0.045 *below* the baseline (3.197) that plain single-head routing sits $+0.08$ above, recovering 0.118 of the 0.141 single-to-multi-head gap; MHAR (3.129) retains a residual -0.023 edge (single-seed, near the tail-noise floor; the effect of implementation differences is bounded using a matched control without logit scaling, $\tau{=}1$, with a loss change of -0.003). These controls indicate that the dominant driver of both the increment on our corpus and the web-corpus regression is the *sharpness* of the shared depth softmax—a single width-d query's logit scale grows with width, over-sharpening the softmax—rather than subspace disagreement as such, and the probe's width-growing disagreement (Table 7) is best read as a correlate of the same width-driven logit growth. The multi-head reshape supplies the $1/\sqrt{H}$ routing temperature through the head dimension, without a separate temperature parameter. It also allows independent routing across subspaces, whose residual contribution is small but non-negative at both tested points.

F FUSED ROUTING KERNELS

Depth routing is memory-bound for two reasons. First, it does almost no arithmetic per byte: scoring a source row against the query and folding it into the mix (Eq. 3) is a constant number of operations per element, with none of the data reuse that lets the attention and MLP matrix multiplies run compute-bound, so throughput is set by memory bandwidth. Second, the traffic is inherently quadratic in depth: sublayer s scores and mixes all $N_s{=}s{+}1$ prior states, so a training step must move $\sum_s N_s\, B\, T\, d = O(L^2\, B\, T\, d)$ of source data no matter how it is implemented. A reference implementation pays this bound several times over (per-sublayer stacking of the source list, a materialized normalized key tensor, separate softmax/mix kernels, thousands of gradient-accumulation launches in backward) and additionally retains the per-sublayer stacked sources for autograd, the dominant activation cost of the mechanism (Appendix G). We provide a fused Triton implementation that reaches (approximately) the bound once per pass: a shared source buffer plus one kernel per routing call in each direction, with deterministic reductions throughout (no atomics), so runs are bitwise reproducible.

Shared source buffer. Each forward pass allocates one $[2L{+}1, B, T, d]$ buffer and writes every residual state into it exactly once as it is produced. Routing call s reads rows $[0, N_s)$ directly, eliminating the per-sublayer source stacking (an $O(N_s)$ copy per call, $O(L^2\, B\, T\, d)$ in total).

Forward: online softmax over depth with the norm fused in. One program per position (b, t) holds a $[H, d/H]$ accumulator in registers and loops over the N source rows. For each row it computes the RMS statistic and the per-head logits $\ell_{n,h} = \langle q_h, \widehat{s}_{n,h} \rangle$ on the fly (fp32 arithmetic, with the normalized key rounded to the storage precision at the same stage as in the reference implementation), and folds the row into the accumulator with the standard running-max/renormalization update, as in

```
def route_fwd(V, q, g, partial, out, W):  # one program per position p
    m = full([H], -inf); l = zeros([H]); acc = zeros([H, K])
    for n in range(N):                    # single pass over the depth
    sources
        v = load(V[n, p])                 # [H, K] tile; the only source
    read
        r = rsqrt(mean(v * v) + eps)      # RMS statistic over the full d
    -row
        khat = round(v * r) * g           # fused RMSNorm; no key tensor
    made
        logit = sum_k(q * khat)           # per-head scores  [H]
        W[n, p] = logit                   # raw logit; normalized after
    loop
        m_new = maximum(m, logit)         # online softmax: rescale, fold
     in
        acc = acc * exp(m - m_new) + exp(logit - m_new) * v
        l   = l   * exp(m - m_new) + exp(logit - m_new)
        m = m_new
    out[p] = acc / l + (partial[p] if DELTA else 0)
    for n in range(N):                    # tiny second loop: N*H scalars
        W[n, p] = exp(W[n, p] - m) / l # exact weights, saved for
    backward
```

Figure 8: **Fused routing forward.** One program (thread block) per position $p{=}(b,t)$ with $[H,K]$ register tiles, $K{=}d/H$. Online softmax over the depth sources with the RMSNorm fused in: each source row is read once, nothing is materialized except the $[N,B,T,H]$ routing weights (a factor d/H smaller than the stacked sources a reference path retains for autograd).

online-softmax attention kernels (Dao et al., 2022). The normalized keys are never materialized, and the only activation saved for backward is the routing-weight tensor $w \in \mathbb{R}^{N\times B\times T\times H}$—a factor d/H smaller than the stacked sources a reference path retains. In the delta-routing variant the kernel adds the residual stream to the mix before writing the output.

Backward: one kernel, norms recomputed, gradients accumulated in place. Let y denote the routing output. Given $\partial\mathcal{L}/\partial y$ and the saved w, the kernel makes two passes over the source rows. The first accumulates the softmax coupling term $S_h = \sum_n w_{n,h}\,\langle \partial y_h, s_{n,h}\rangle$; the second recomputes the RMS statistic from the buffer (the row is being re-read anyway, so recomputation is free relative to saving keys), forms $\partial\ell_{n,h} = w_{n,h}(\langle \partial y_h, s_{n,h}\rangle - S_h)$, and emits per-source gradients combining the value path and the key path through the norm: the contribution added to ∂s_n is

$$r\,(g \odot \partial\widehat{k}_n) - \tfrac{r^3}{d}\,\langle g \odot \partial\widehat{k}_n,\, s_n\rangle\, s_n + w_n\,\partial y, \qquad \partial\widehat{k}_n = \partial\ell_n \cdot q,\;\; r = \left(\tfrac{1}{d}\|s_n\|^2 + \varepsilon\right)^{-1/2},$$

with query and norm-weight gradients accumulated in registers and reduced by a deterministic two-stage tree. Because every source feeds every later routing call, a reference backward fans these contributions in through thousands of small tensor additions; the fused backward instead accumulates them into one shared fp32 buffer *in place*. This is safe because autograd executes the routing calls' backwards in strictly reverse forward order—every later call is a graph descendant of the current call's output through the residual chain—so when call s runs, rows $[0, N_s)$ already contain all contributions from calls $s' > s$. The backward pass for the final routing operation initializes the buffer, and subsequent backward passes accumulate into it in reverse order. Only the row owned by the call's own residual input is handed back to autograd.

Delta variant (Luo et al., 2026). The mid-training configuration (§3.3) routes over a short list—a learnable null source plus at most eight block-aggregated deltas—so cross-call buffer sharing provides little benefit. We use separate source and gradient buffers for each routing operation and add the residual stream to its output. This supports FSDP and activation checkpointing. At the 8B configuration ($d{=}4096$, $H{=}8$, $N{=}9$, per-GPU microbatch 2×4096, per-call checkpointing) routing drops from 651 ms (compiled PyTorch) to 323 ms per microbatch ($2.0\times$; $6.4\times$ over the eager reference), raising end-to-end mid-training throughput by $\sim$30% on 8×H100. The Table 5 from-scratch columns use the Table 1 depth/width at reduced batch so the eager reference also fits in

```
def route_bwd(V, W, dout, dV, dq_part, dg_part):  # one program per pos. p
    do = load(dout[p])                          # [H, K]
    S = zeros([H])
    for n in range(N):                          # pass 1: softmax coupling term
        v = load(V[n, p]); w = W[n, p]
        S += w * sum_k(do * v)
    dq = zeros([H, K]); dg = zeros([H, K])
    for n in range(N):                          # pass 2: all gradients
        v = load(V[n, p]); w = W[n, p]
        dlogit = w * (sum_k(do * v) - S)        # softmax backward  [H]
        r = rsqrt(mean(v * v) + eps)            # norm recomputed, not saved
        khat = round(v * r)
        dq += dlogit * (khat * g)               # query grad, register-resident
        dK = dlogit * q                         # grad wrt normalized key
        dg += dK * khat                         # norm-weight grad
        dkp = dK * g                            # grad wrt v * r
        dv = r * dkp - (r**3 / d) * sum(dkp * v) * v + w * do
        dV[n, p] = dv if FIRST_CALL else dV[n, p] + dv  # in-place accum.
    dq_part[p] = dq; dg_part[p] = dg    # per-position partials, host-summed
```

Figure 9: **Fused routing backward.** Two passes over the source rows: the first accumulates the softmax coupling term S, the second recomputes the norm from the buffer (the row is re-read anyway) and emits the closed-form key- and value-path source gradients, accumulating them across routing calls in place (the reverse order of backpropagation permits this accumulation).

memory (100M: B=2, 350M/1B: B=1; ratios are batch-invariant); the eager reference kernels are a further 1.5–2.5× slower than the compiled PyTorch reference (7.9–11.5× slower than fused).

Verification. In fp32 the fused path matches the reference exactly up to reduction order: on routing chains and full models (with and without activation checkpointing), losses are identical and every parameter gradient agrees to $\leq 2.5 \times 10^{-6}$ maximum relative error. In bf16 the two paths round differently; measured against fp32 ground truth, the fused gradients are on average *closer* to the truth than the reference's (gradient-error ratio geometric mean $\approx$0.5 across batches; gate-scalar gradients are cancellation-dominated in both paths). Outputs are bitwise identical across repeated runs. Measured cost across scales is reported in Appendix G (Table 8).

G Compute and Memory Cost

Table 8 reports training throughput and peak memory at the three settings of Table 1 on a single H100 for three implementations of the routing; single-head attention residuals are cost-identical to MHAR (§2), so all overhead is inherited from the attention-residual mechanism. The reference kernels pay the $O(L^2\,B\,T\,d)$ source traffic several times over (Appendix F) and retain the stacked sources of every sublayer for backward, which exceeds 80 GB at 1B; PyTorch compilation fuses the arithmetic but not the stacking, retention, or backward accumulation. The fused kernels reach 1.6–2.4× the compiled path at approximately baseline peak memory; the remaining gap is the inherent source re-read, which they run at roughly HBM bandwidth.

H Width-control: isolating width in loss

This is the loss-level counterpart of probe control (ii) (Appendix E): a correlation, not a causal test. We train all three methods at d=768 ("124M"; L=12, four KV heads) and at the matched d=512 100M architecture of Table 1 (only d, the head count, and the FFN width change) on FineWeb-Edu

Table 8: Training compute and memory at the three model settings of Table 1 (single H100; per-GPU microbatch as in training: batch 8 / sequence 2048 at 100M, 4/1024 at 350M, 2/1024 at 1B; identical node and software for all cells; 120-step runs). Each cell is the median of three steady-state windows (within $\pm 1\%$). Throughput is relative to the same-scale baseline (absolute baseline medians: 113.9k / 45.0k / 19.7k tokens/s). Parameters are matched to the baseline up to the $O(d)$ routing queries ($+0.02\%$). †Single-head attention residuals are cost-identical to MHAR by construction and not listed separately.

	100M		350M		1B	
Method†	thr.	mem. (GB)	thr.	mem. (GB)	thr.	mem. (GB)
Baseline (additive residual)	$1.00\times$	41.5	$1.00\times$	19.4	$1.00\times$	19.0
MHAR, reference kernels	$0.31\times$	68.8	$0.16\times$	70.9	OOM (>80 GB)	
MHAR, compiled PyTorch	$0.54\times$	52.4	$0.32\times$	40.1	$0.23\times$	47.5
MHAR, fused Triton (ours)	$\mathbf{0.88}\times$	**42.0**	$\mathbf{0.71}\times$	**20.0**	$\mathbf{0.55}\times$	**20.1**

under a unified 5×10^{-4} recipe (20K steps, sequence 2048, global batch 64), so the d=512 deltas differ slightly from Table 13's tuned-rate numbers. Within each width all methods share the same data-parallel degree (Table 9).

Table 9: Width-control (FineWeb-Edu, tail-mean validation loss, steps 15–20k). Holding depth (L=12) and KV (4) fixed and only widening d from 512 to 768: single-head routing's edge over the baseline *erodes* ($-0.041 \to -0.033$) while MHAR holds ($-0.050 \to -0.053$), so MHAR's advantage over single-head *doubles* ($-0.010 \to -0.020$). Single-seed; the single-head deficit shift (0.008) is near the per-seed noise floor, so the robust signal is the widening MHAR$-$single gap.

Width	**Baseline**	**Single-head**	**MHAR**	**MHAR$-$single**
100M (d=512)	3.462	3.421 (-0.041)	3.411 (-0.050)	-0.010
124M (d=768)	3.220	3.187 (-0.033)	3.167 (-0.053)	$\mathbf{-0.020}$

I Robustness of the from-scratch comparison

Compute-equivalent gain: exponent sensitivity. The $1.3\times/1.8\times/1.7\times$ compute-equivalent gain of §3 uses α=0.089, fitted to the lower envelope of the three baseline training curves. Fitting only the three final points gives α=0.053—nearly identical to the 0.057 that Kimi Team (2026) fit on their own ladder—and would give 1.5–2.6×, while a steeper Chinchilla-like α=0.15 gives 1.15–1.4×.

Compute-optimal budget. Table 10 details the Chinchilla-optimal retraining summarized in §3.

Table 10: **Validation loss at a compute-optimal budget.** MHAR$-$baseline validation-loss gap (↓, tail-mean of the last eleven paired same-step evaluations) at the 20K-step main-table budget versus the Chinchilla-optimal $\approx$ 20 tokens/parameter budget (7.0B tokens at 350M, 21.6B at 1B; our corpus, single seed). The loss advantage narrows at the larger token budget.

Scale	**20K-step (Δ)**	**Compute-optimal (Δ)**	**Tokens**
350M	-0.149	-0.119 ± 0.002	0.66B $\to$ 7.0B
1B	-0.140	-0.081 ± 0.001	0.66B $\to$ 21.6B

Document-disjoint validation. The streaming validation protocol (Appendix J) re-shuffles the training corpus, so Table 1's gaps could in principle reflect train–validation document overlap. To rule this out we re-evaluated all four models trained on our corpus on a *document-disjoint* fixed window set: we deterministically replayed each data-parallel worker's document stream (using the same shuffle seeds and streaming procedure) and recorded MD5 hashes of a strict *superset* of every document read by any worker (a 2× margin in the number of characters replayed accounts

for variation caused by stream reconnections during training; 18.0M hashes), then built 400 fixed windows per training context exclusively from documents outside that superset, evaluated identically for every model at a scale (Table 11); the ranking and gap sizes of Table 1 reproduce.

Table 11: Document-disjoint fixed-window validation loss (↓) of the models trained on our corpus in Table 1: 400 windows per scale (at the training context; 100M: 2048, 350M/1B: 1024), built only from documents verified (by digest) to lie outside a replay-derived superset of every document consumed by any data-parallel worker; every model at a scale sees the identical windows. Δ vs. the same-scale baseline. Both the ranking and the gap sizes of Table 1 reproduce.

Scale	Baseline	Hyper-conn.	Single-head	MHAR (H=8)
100M	2.985	2.954 (−0.031)	2.924 (−0.061)	**2.924** (−0.061)
350M	2.951	2.891 (−0.060)	2.836 (−0.115)	**2.810** (−0.141)
1B	2.832	2.769 (−0.064)	2.701 (−0.131)	**2.695** (−0.138)

Table 12: Extended downstream suite (zero-shot, n=500 per task, single run) for the four models trained on our corpus in Table 1: ARC-Easy (test), PIQA (val), WinoGrande-XL (val, full-sentence scoring), BoolQ (val). Sampling noise is $\approx \pm 2.2$ points at n=500; the arms are statistically inseparable on these tasks at these scales, and WinoGrande is near chance throughout (the 100M single-head BoolQ outlier is a yes/no calibration artifact common at this scale).

Scale	Method	ARC-E	PIQA	WinoGrande	BoolQ
100M	Baseline	38.8	61.4	52.0	52.2
100M	Hyper-conn.	40.2	59.4	52.8	57.6
100M	Single-head	40.2	61.8	53.8	40.6
100M	MHAR	41.2	59.6	53.2	51.6
350M	Baseline	43.6	61.0	53.0	61.8
350M	Hyper-conn.	41.8	61.0	51.0	59.8
350M	Single-head	44.8	62.4	50.2	62.6
350M	MHAR	42.4	62.2	51.6	63.0
1B	Baseline	42.0	59.6	53.6	63.0
1B	Hyper-conn.	41.2	61.0	49.0	63.0
1B	Single-head	42.2	64.4	49.2	62.6
1B	MHAR	42.2	63.2	54.6	63.4

Web-data comparison and learning-rate robustness (FineWeb-Edu). Table 13 repeats the four-method comparison of Table 1 on web-generic FineWeb-Edu (Penedo et al., 2024), with each method at its own best peak learning rate from an independent per-method sweep over $\{1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}\}$ (1×10^{-4} is uniformly too low: validation loss 4–5 at 20K steps). Baseline, hyper-connections, and MHAR prefer 1×10^{-3}; single-head routing, whose single shared query is destabilized by the higher rate, prefers 5×10^{-4} at 350M and 1B. These runs, like the other web-data runs in this appendix, the head grids of Appendix D, and the checkpoints probed in Appendix E, use randomly initialized routing queries (deviating from Kimi Team (2026); on this corpus zero-init moves the 1B single-head delta from $+0.082$ to $+0.018$ at 5×10^{-4} but does not change its sign at the tuned rate). The single-head column reverses relative to our corpus: helpful at 100M, above baseline at 350M–1B even at its own best rate, while MHAR has the lowest loss at every scale—the distribution-dependence discussed in §3.

Robustness to training budget. At the fixed 20K-step budget the models are under-trained, which raises the question of whether MHAR's advantage is merely an under-training artifact. We therefore train baseline, single-head, and MHAR for 3× longer (60K steps) at 350M and 1B and read the baseline-vs-method gap at both budgets (within-run; the longer cosine schedule means the 20K read here is not comparable to Table 1, only the gap-vs-budget trend is). Tripling the budget lowers validation loss by $\sim$0.25, while the differences between methods persist: MHAR stays -0.056 (350M) / -0.062 (1B) ahead of the baseline at 60K (vs. -0.069 / -0.068 at 20K), and single-head routing's 1B regression persists ($+0.016$ at 20K, $+0.018$ at 60K). These differences remain observable at the 60K-step budget.

Table 13: Web-data (FineWeb-Edu) validation loss (↓, tail-mean), each method at its own best peak learning rate (1×10^{-3} unless marked *: 5×10^{-4}); Δ vs. the baseline; $\text{CEG}_{\text{FLOPs}}$ is the compute-equivalent gain of MHAR over baseline. At 1×10^{-3}, single-head is $+0.055$/$+0.140$ at 350M/1B. Randomly initialized routing queries (see text).

Scale	d/L/KV	Baseline	Hyper-conn.	Single-head	MHAR	$\text{CEG}_{\text{FLOPs}}$
100M	512/12/4	3.336	3.322 (-0.014)	3.297 (-0.039)	**3.287** (-0.049)	$1.27\times$
350M	1024/24/8	3.290	3.260 (-0.030)	3.328^* ($+0.038$)	**3.210** (-0.080)	$1.49\times$
1B	1280/36/8	3.175	3.172 (-0.002)	3.279^* ($+0.105$)	**3.111** (-0.063)	$1.38\times$

Seed robustness (all scales). We train *three seeds per method at every scale* under a single shared 5×10^{-4} recipe, so that the three methods differ *only* in seed and routing mechanism. Because seed s fixes the data order across methods, each method-vs-baseline comparison is *paired*, cancelling cross-seed noise. MHAR improves over the baseline at all three scales, with the paired delta exceeding ten standard errors everywhere (-0.045/-0.090/-0.071 nats at 100M/350M/1B; Table 14). The single-head and hyper-connection patterns also hold under seeds. Single-head routing moves from a significant *help* at 100M (-0.043) through -0.005 at 350M to a significant *regression* at 1B ($+0.093$), and is by far the highest-variance method: its three 1B paired deltas span $+0.04$ to $+0.16$, versus MHAR's ± 0.002. Hyper-connections help at 100M/350M but fall within noise at 1B (-0.016).

Table 14: Seed robustness (FineWeb-Edu): paired per-seed delta vs. baseline (validation loss ↓, tail-mean of the last eleven evaluations; three seeds per method per scale, shared 5×10^{-4} recipe; $\pm$ is the per-seed standard deviation). Seed s fixes the data order across methods, so the comparison is paired. MHAR improves at every scale with $|\Delta|/\text{SE} \geq 10$; single-head crosses from help to harm and is the highest-variance method.

Scale	Hyper-conn.	Single-head	MHAR
100M	-0.031 ± 0.003	-0.043 ± 0.003	$\mathbf{-0.045 \pm 0.007}$
350M	-0.028 ± 0.009	-0.005 ± 0.015	$\mathbf{-0.090 \pm 0.004}$
1B	-0.016 ± 0.015	$+0.093 \pm 0.049$	$\mathbf{-0.071 \pm 0.002}$

J Training and reproducibility details

All runs use the same training procedure; the settings below are common to every method and scale unless noted.

Mid-training (8B). The base checkpoint is released at a constant peak rate of 1.7×10^{-3} (published before cooldown); our control's peak 4×10^{-4} is this plateau rate scaled down by $\sqrt{\text{batch ratio}}$. Because no 8B plateau checkpoint ships loadable optimizer moments, we cold-start AdamW (Loshchilov & Hutter, 2019) and use a short linear re-warmup ($0 \rightarrow$peak) to rebuild the second moment before decaying. Training uses FSDP full-shard, bfloat16, and activation checkpointing on $8\times$H100-80GB, sequence length 4096, weight-EMA (β=0.999, evaluated), a logit z-loss (10^{-4}), and weight decay 0.05 (excluding embeddings, norms, biases).

Tokenizer and data. We tokenize our corpus (Appendix J.1; composition in Table 15) with the Qwen3 byte-level BPE tokenizer from Qwen3-0.6B. Documents are streamed from the corpus shards, joined with the end-of-text token (id 151,645), and packed into contiguous sequences of length T with no padding. The token-embedding and output projection are tied with model vocabulary size 151,936. The supplementary web replication (Appendix I) runs the identical pipeline on FineWeb-Edu (Penedo et al., 2024).

Architecture. Every model follows the Qwen3 recipe (Qwen, 2025): pre-norm decoder blocks with RMSNorm (ϵ=10^{-6}), SwiGLU (SiLU) MLPs, grouped-query attention with per-head query/key RMSNorm, and rotary position embeddings (RoPE, θ=10^4; no learned or absolute positional em-

beddings), with a maximum configured sequence length of $2T$. Attention biases are disabled and weights are initialized from $\mathcal{N}(0, 0.02^2)$. There is no dropout (attention and hidden dropout are 0).

Optimization. AdamW (Loshchilov & Hutter, 2019) with β_1=0.9, β_2=0.95, ϵ=10^{-8}, and weight decay 0.1 applied uniformly to all parameters. Gradients are clipped to global norm 1.0. The learning rate warms up linearly over $1{,}000$ steps to the peak and then follows a cosine decay to $0.1\times$ peak over the full 20K-step budget. Training uses bfloat16 (parameters and compute) under distributed data parallelism at all scales up to 1B.

Batch and seed. Sequence length and global batch are as in §3; the base random seed is 42 (offset per worker).

Validation protocol. There is no explicit held-out split: validation draws from the *same* corpus under a different shuffle order of the stream (a $+9999$ seed offset from training, plus a per-worker offset), which reduces rather than eliminates train/validation overlap. Every 500 steps we run one evaluation pass: the token-level cross-entropy is averaged over the pass's evaluation batches and then averaged across data-parallel workers. Reported numbers are the *tail-mean* of the last eleven evaluations (steps 15–20K); the single-pass ± 0.07 noise and the paired-comparison rationale are discussed in §3 (Metric).

J.1 Data construction

Table 15 gives the measured composition of our corpus used for the from-scratch and 8B mid-training experiments (§3). We assemble it from public, quality-filtered corpora and shuffle documents into 2,048 uniform parquet shards (≈ 8 TB of text, ≈ 1.9 T tokens); every document retains its corpus of origin and upstream quality metadata. We compute source shares from these records in the sample described below.

Table 15: Composition of our corpus (Appendix J.1): share of text bytes per source, measured over a uniform random sample of 48 of the 2,048 shards (187 GB of text; group rows sum their constituents, so rounded columns may differ in the last digit). The corpus is English-only; FinePDFs is restricted to its top three of twenty quality bins (verified from the quality-bin assignment recorded for each document).

Source	**% bytes**
Synthetic web rephrasings (Nemotron-CC-v2/v2.1)	24.3
Nemotron-CC v2: high-quality synthetic rephrasings	14.9
Nemotron-CC v2.1: high-quality synthetic rephrasings	6.0
Nemotron-CC v2.1: high-quality synthetic rephrasings (translated English)	3.5
Raw high-quality web (Nemotron-CC-v2/v2.1)	16.6
Nemotron-CC v2: high-quality web text	12.5
Nemotron-CC v2.1: high-quality web text translated into English	2.4
Nemotron-CC v2.1: high-quality web text	1.6
Web PDFs	15.9
FinePDFs (top 3/20 quality bins)	15.9
Synthetic diverse QA (Nemotron-CC-v2/v2.1)	14.1
Nemotron-CC v2: diverse question answering	8.3
Nemotron-CC v2: translated diverse question answering	5.3
Nemotron-CC v2.1: high-quality diverse question answering	0.4
Code	11.9
Stack-Edu (Dolma collection)	6.7
Nemotron: code instruction data	2.8
Nemotron-CC Code v1	2.0
Nemotron v1.1: code concepts	0.4
Nemotron: scientific coding	0.1
SFT / STEM / reasoning	9.7
Nemotron: general instruction data	4.6
Nemotron: STEM instruction data	4.1
Nemotron: RQA	0.7
Nemotron: Infinibyte reasoning	0.2
Nemotron v1.1: multiple-choice questions	0.1
Nemotron v1.1: formal logic	$<$0.1
Nemotron v1.1: unconditional algorithmic data	$<$0.1
Math / arXiv	6.9
Nemotron Math: subset 3	2.5
Nemotron Math: subset 4+	1.6
Nemotron Math: subset 4+ (MIND)	1.1
Proof-Pile arXiv (Dolma, RPJ)	0.9
Nemotron: mathematics textbooks	0.8
Nemotron: mathematics instruction data	0.1
Wikipedia	0.6
Nemotron: rewritten Wikipedia	0.3
FineWiki: English	0.3