ON THE DESIGN OF HYPER-CONNECTIONS: SIMPLIFICATION, STABILITY, AND SCALABILITY

Anonymous authors
Paper under double-blind review

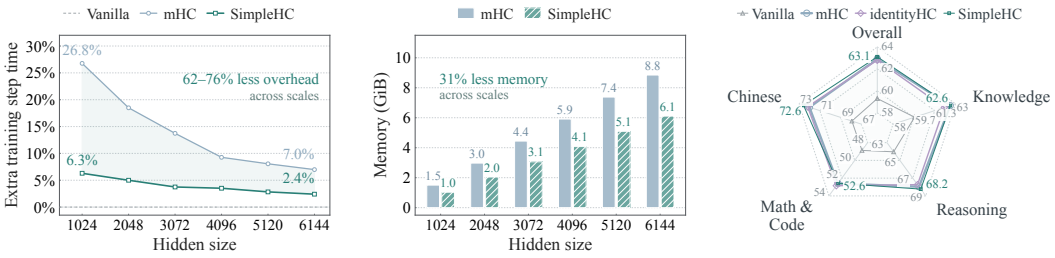
ABSTRACT

Hyper-Connections (HC) architectures are increasingly adopted in frontier language models, yet their components remain insufficiently understood and introduce non-negligible training overhead. In this work, we systematically study the HC architecture to understand its components and identify simpler, more efficient designs. We find that learned read/write gating without mixing and unconstrained mixing with fixed gates form two effective interaction groups: either alone achieves performance comparable to mHC. We also identify repeated access to the expanded residual state as a major source of training overhead, even without mixing. These findings lead to **SimpleHC**, which simplifies the architecture by removing residual mixing and simplifies execution through *shared memory access*, using the resulting additive structure and shared gate inputs. Layer-specific gates share a group-entry state, enabling batched reads and deferred writes; scalar corrections exactly preserve preceding sublayer output contributions. Our analyses support both the retained residual structure and the design of shared memory access. SimpleHC incurs only $\mathbf{2}.4\%$ extra end-to-end training time over a vanilla residual baseline at the largest tested scale. It maintains performance comparable to mHC when scaling from 15B to 30B MoE models, exhibits smoother gradient dynamics during standard training, and remains stable under stress.

1 INTRODUCTION

Residual connections support optimization and information propagation in modern language models (He et al., 2016a;b; Vaswani et al., 2017; Grattafiori et al., 2024; Yang et al., 2025; Liu et al., 2024b), and recent work improves language modeling and depth utilization by enriching this pathway (Pagliardini et al., 2024; Xiao et al., 2025; Team et al., 2026). Hyper-Connections (HC) (Zhu et al., 2025) introduce multiple residual streams with learned read, write, and mixing operations for flexible information routing. The manifold-constrained variant, mHC (Xie et al., 2025), constrains residual mixing to improve training stability at scale. HC-based architectures, particularly mHC, have been adopted in frontier language models (Xu et al., 2026; Zeng et al., 2026). Despite the growing adoption, these architectures remain insufficiently understood and costly to train: their gating and mixing operations introduce architectural complexity and non-negligible execution overhead, even under efficient implementations (Xie et al., 2025).

A key source of this overhead is repeated access to the expanded residual state: at every sublayer, HC-style mappings read and write n streams of width C to generate gates, aggregate inputs, and



(a) Training overhead (b) Saved activation storage (c) Downstream performance

Figure 1: **SimpleHC reduces training overhead while retaining competitive performance.** (a) Additional training overhead relative to baseline with efficient implementations. (b) Storage retained for backward at sublayer boundaries (Section 5.5). (c) Average scores on 30B MoE models (Table 4).

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.

propagate updates. This memory traffic incurs costs not captured by FLOPs alone. In our end-to-end experiments with an efficient implementation, mHC adds $7\% - 27\%$ training time over a vanilla baseline (Figure 3(a)), despite adding less than 1% FLOPs. This motivates reconsidering the underlying read, write, and mixing interactions. We ask: **Can a simpler Hyper-Connections design reduce execution cost while preserving model quality and training stability at scale?**

In this work, we systematically study the Hyper-Connections architecture through three components: read gates, write gates, and residual mixing. We find that unconstrained mixing improves quality, but Sinkhorn constraints substantially reduce its benefit. Learned read/write gating without mixing and unconstrained mixing with fixed gates each achieve quality comparable to mHC, with further gains from combining them. Our analysis also highlights the propagation risks of unconstrained mixing, leading us to select learned read/write gating with identity transitions as a minimal effective structure that retains competitive quality while avoiding these risks.

Without residual mixing, the additive structure lets *scalar corrections* exactly recover preceding output contributions for given gates, without materializing intermediate nC-dimensional states. Gate generation, however, still accesses the updated state at every sublayer. We reduce these accesses by computing layer-specific gates from a common group-entry state, consolidating reads and deferring writes across sublayers. We call this execution scheme *shared memory access*.

Together, these findings lead to **SimpleHC**, which **simplifies the architecture** through learned read/write gates without mixing and **simplifies execution** through shared memory access. We also develop specialized kernels to translate this simplicity into training efficiency. As shown in Figure 1(a), SimpleHC incurs only 2.4% additional training cost at the largest tested scale and halves memory traffic from $18C$ to $9C$ per sublayer relative to fused mHC. It achieves downstream performance comparable to mHC and identityHC on 15B and 30B MoE models (Figure 1(c)). Analyses in Section 6 further support the design of SimpleHC. It also exhibits smoother gradient dynamics during training and remains stable under stress-test settings.

2 PRELIMINARIES

Hyper-Connections (HC, dynamic variant) (Zhu et al., 2025) and Manifold-Constrained HC (mHC) (Xie et al., 2025) share a multi-stream residual structure. The two methods differ in how their coefficients are conditioned and parameterized, and whether residual mixing is constrained.

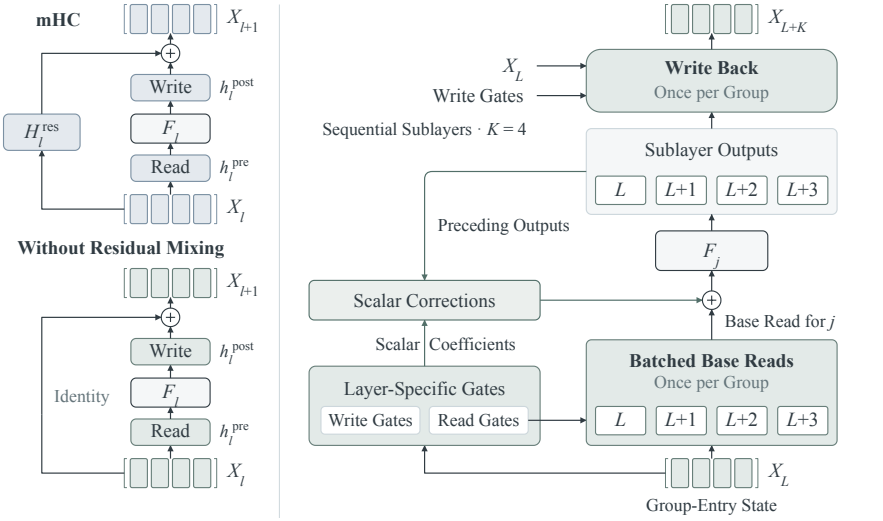
Each Transformer block contains two *sublayers*: attention and FFN/MoE. Let $X_l \in \mathbb{R}^{n\times C}$ contain n width-C residual streams entering sublayer l. All vectors are column vectors; $\mathrm{vec}(X_l) \in \mathbb{R}^{nC\times 1}$ concatenates streams in row order. We omit token and batch indices unless stated; I denotes the $n\times n$ identity matrix. Both methods compute $u_l = X_l^\top h_l^{\mathrm{pre}}$, $o_l = F_l(u_l)$, and $X_{l+1} = H_l^{\mathrm{res}} X_l + h_l^{\mathrm{post}} o_l^\top$. The read and write gates $h_l^{\mathrm{pre}}, h_l^{\mathrm{post}} \in \mathbb{R}^{n\times 1}$ aggregate stream states and distribute sublayer outputs; $H_l^{\mathrm{res}} \in \mathbb{R}^{n\times n}$ mixes streams. Here, $u_l, o_l \in \mathbb{R}^{C\times 1}$, and F_l denotes the sublayer module.

HC projects each normalized C-dimensional stream using $W_l^{\mathrm{pre}}, W_l^{\mathrm{post}} \in \mathbb{R}^{C\times 1}$ and $W_l^{\mathrm{res}} \in \mathbb{R}^{C\times n}$, shared across streams within each sublayer. Scaled $\tanh$ projections are added to learned static coefficients, yielding read/write gates and unconstrained residual mixing.

In contrast, mHC uses the normalized full state $z_l = \mathrm{RMSNorm}(\mathrm{vec}(X_l)) \in \mathbb{R}^{nC\times 1}$ to compute $\ell_l^s = \alpha_l^s (W_l^s)^\top z_l + b_l^s$ for $s \in \{\mathrm{pre}, \mathrm{post}, \mathrm{res}\}$. Here, $W_l^{\mathrm{pre}}, W_l^{\mathrm{post}} \in \mathbb{R}^{nC\times n}$ and $W_l^{\mathrm{res}} \in \mathbb{R}^{nC\times n^2}$. Each α_l^s is a learned scalar; logits and learned biases have n entries for read/write gates and n^2 for mixing. Parameters are independent across operations and sublayers. mHC applies sigmoid to read logits, twice sigmoid to write logits, and Sinkhorn normalization to mixing logits reshaped into an $n \times n$ matrix. Sinkhorn approximately enforces a doubly stochastic H_l^{res} to stabilize propagation across depth. Unless otherwise stated, we use the full-state gate parameterization of mHC.

3 SIMPLEHC

We propose SimpleHC, a Hyper-Connections architecture with a simplified residual structure and execution scheme (Figure 2). It retains multiple residual streams and learned read/write gates while removing residual mixing. Shared memory access reduces repeated state access by generating layer-specific gates from group-entry states, batching reads, and deferring writes.



(a) Residual Simplification **(b) SimpleHC: Grouped Execution with Shared Memory Access**

Figure 2: **Overview of SimpleHC.** (a) Identity transitions replace residual mixing, retaining learned read/write gates. (b) Layer-specific gates share a group-entry state across $K = 4$ sublayers, enabling batched reads and one group-boundary write-back. Scalar corrections preserve sequential output contributions without materializing intermediate expanded states.

3.1 Simplifying the Residual Structure

Prior work observed that removing residual mixing from mHC by setting $H_l^{\text{res}} = I$ (identityHC) largely preserves model quality (Guo et al., 2026). Our study finds gains from unconstrained mixing but little benefit from Sinkhorn-constrained mixing over identity transitions (Table 2). However, unconstrained mixing risks cumulative signal amplification or attenuation across depth (Section 6.1).

We therefore retain learned read/write gates with identity transitions as a minimal effective structure, preserving competitive quality while avoiding residual matrix products, with $X_{l+1} = X_l + h_l^{\text{post}} o_l^\top$. With $z_l = \text{RMSNorm}(\text{vec}(X_l))$ and $\ell_l^s = \alpha_l^s (W_l^s)^\top z_l + b_l^s$ for $s \in \{\text{pre}, \text{post}\}$, SimpleHC uses $h_l^{\text{pre}} = \sigma(\ell_l^{\text{pre}})$ and $h_l^{\text{post}} = 2\sigma(\ell_l^{\text{post}})$. Without residual mixing, updates accumulate additively. Given the state X_L at sublayer L, the residual state at any later sublayer j is

$$X_j = X_L + \sum_{k=L}^{j-1} h_k^{\text{post}} o_k^\top. \tag{1}$$

Each term adds a preceding sublayer's output to the streams according to its write gate, and subsequent residual transitions leave this allocation unchanged.

3.2 Shared Memory Access

Even without residual mixing, conventional execution accesses the nC-dimensional state at every sublayer to generate gates, form inputs, and write updates. We introduce shared memory access across sublayers to reduce this traffic. With additive updates, each preceding output contributes to a later input through the inner product of its write gate and the later read gate. Substituting Equation 1 into the read operation gives:

$$u_j = X_j^\top h_j^{\text{pre}} = X_L^\top h_j^{\text{pre}} + \sum_{k=L}^{j-1} \underbrace{\left((h_j^{\text{pre}})^\top h_k^{\text{post}}\right)}_{\text{scalar coefficient}} o_k. \tag{2}$$

We call $X_L^\top h_j^{\text{pre}}$ the *base read* and the remaining terms *scalar corrections*, each weighting a preceding C-dimensional output by a scalar coefficient. For given gates, this reconstructs the input without writing intermediate updates into the nC-dimensional residual state.

This reconstruction assumes that the gates are already available. However, computing $h_j^s = g_j^s(X_j)$ requires materializing and accessing the updated state at every sublayer, where g_j^s is the gate map-

Table 1: Memory access of residual-stream operations (reads and writes), per token per sublayer.

Operation	mHC (naive)	mHC (fused)	identityHC (fused)	SimpleHC (fused)
Projection and RMS statistics	nC	nC	nC	nC/K
Base reads	$nC + C$	$nC + C$	$nC + C$	$nC/K + C$
Residual write-back	$6nC + C$	$2nC + C$	$2nC + C$	$2nC/K + C$
Total	$8nC + 2C$	$4nC + 2C$	$4nC + 2C$	$4nC/K + 2C + Q_{\mathrm{corr}}(K)$
$n = K = 4$	$34C$	$18C$	$18C$	$9C$

ping defined above for $s \in \{\mathrm{pre}, \mathrm{post}\}$. SimpleHC relaxes this dependency by generating gates for K consecutive sublayers from a common group-entry state X_L:

$$\bar{h}_j^{\mathrm{pre}} = g_j^{\mathrm{pre}}(X_L), \qquad \bar{h}_j^{\mathrm{post}} = g_j^{\mathrm{post}}(X_L), \qquad L \le j < L + K. \tag{3}$$

Within groups, sublayers retain independent gate parameters but share the conditioning state X_L. Relative to per-sublayer conditioning, the only approximation is generating gates from X_L rather than X_j. Grouped gate generation computes layer-specific gates jointly from X_L, reducing state reads from K to 1 and making group gates available upfront. With these gates, Equation 2 becomes

$$u_j = X_L^\top \bar{h}_j^{\mathrm{pre}} + \sum_{k=L}^{j-1} d_{j,k} o_k, \qquad d_{j,k} = (\bar{h}_j^{\mathrm{pre}})^\top \bar{h}_k^{\mathrm{post}}. \tag{4}$$

All K base reads are batched into one scan of X_L, with $d_{j,k}$ precomputed at group entry. Corrections incorporate preceding C-dimensional outputs without intermediate nC-dimensional state writes.

Within each group, sublayers execute sequentially, passing each $o_j = F_j(u_j)$ to later inputs through scalar corrections. A single fused kernel writes back all updates at the group boundary:

$$X_{L+K} = X_L + \sum_{k=L}^{L+K-1} \bar{h}_k^{\mathrm{post}} o_k^\top. \tag{5}$$

This execution is algebraically equivalent to sequential execution with the same group-entry gates (Appendix C). SimpleHC uses $K = 4$, where shared memory access reduces full-state reads for gate generation and input aggregation from 8 to 2 and writes from 4 to 1 relative to original execution. Each full-state read/write accesses nC elements per token. Algorithm 1 summarizes the schedule.

4 Infrastructure Design

To realize the efficiency benefits of the simplified design, we optimize the forward and backward execution of SimpleHC with specialized kernels. Appendix D provides implementation details.

4.1 Forward Fusion with Shared State Access

1) **Grouped reads.** We compute all layer-specific gate projections and RMS statistics from one state read, then fuse RMS scaling with gate activations to avoid materializing a normalized-state buffer. A separate batched-read kernel reuses each state tile for all K base reads. Gate generation and base aggregation each reduce full-state reads from KnC to nC elements per token per group. 2) **Fused corrections and deferred write-back.** Correction kernels directly load preceding C-dimensional outputs and fuse weighting, summation, and base addition, avoiding stacked output copies and expanded updates. One group-boundary kernel fuses all output allocations with residual addition, reducing wide-state update traffic from $2KnC$ to $2nC$ elements per token per group.

4.2 Backward Reuse and Efficient Reductions

We tailor backward kernels to two distinct workloads: wide activation gradients benefit from cross-sublayer reuse, whereas small gate gradients require efficient channel reductions.

1) **Shared activation-gradient access.** We fuse the backward contributions of all K base reads with the identity gradient, avoiding K expanded gradients and their merge. The write-back backward reuses one scan of the group-boundary gradient for all K output gradients, reducing its logical

Table 2: **Architectural design choices in Hyper-Connections.** 15B-A1B MoE models trained on 200B tokens. C/nC denote stream-local/joint-state gate inputs; Fixed uses uniform weights of $1/n = 1/4$ for read gates and 1 for write gates.

Method	Read	Write	Mixing	Loss ↓	MMLU	HellaSwag	MATH	MBPP	C3	Avg.
(1) Vanilla Residual	—	—	—	1.678	56.1	65.8	26.2	37.8	75.9	52.4
Gate inputs and residual mixing										
(2) HC	C	C	Unconstrained	1.660	**60.9**	66.4	**30.4**	39.8	81.6	55.8
(3) HC w/o mixing	C	C	Identity	1.664	59.6	65.5	30.1	37.6	82.9	55.1
(4) mHC w/o Sinkhorn	nC	nC	Unconstrained	**1.657**	60.0	66.4	27.9	**42.6**	**84.2**	**56.2**
(5) mHC	nC	nC	Sinkhorn	1.659	58.8	**67.3**	26.9	39.8	82.2	55.0
(6) identityHC	nC	nC	Identity	1.660	59.6	66.4	29.6	39.6	81.9	55.4
Read- and write-input restrictions										
(7) Restricted read	C	nC	Identity	**1.661**	**59.9**	**66.9**	28.5	39.0	**80.1**	**54.9**
(8) Restricted write	nC	C	Identity	1.663	59.1	66.4	**29.0**	**39.2**	79.9	54.7
Fixed-gate controls										
(9) Fixed read	Fixed	nC	Identity	1.676	57.5	65.5	27.9	38.4	78.5	53.6
(10) Fixed write	nC	Fixed	Identity	1.678	56.2	64.3	27.5	38.0	78.1	52.8
(11) Fixed read and write	Fixed	Fixed	Sinkhorn	1.678	56.5	64.7	26.7	37.4	77.0	52.5
(12) Fixed read and write	Fixed	Fixed	Unconstrained	**1.660**	**61.9**	**67.1**	**29.1**	**40.2**	**81.8**	**56.0**

reads by $K\times$. Fusing projection input gradients with RMS differentiation further avoids a wide intermediate and a separate normalization-backward pass. 2) **Token-packed gate gradients.** Gate gradients reduce C channels to only nK coefficients per token. We pack multiple tokens into a Tensor Core matrix product, accumulate channel reductions on-chip, and store only same-token blocks. This eliminates the partial-gradient scratch buffer and extra reduction launch required by the channel-tiled fallback.

4.3 Memory-Access and Storage Accounting

Table 1 counts logical forward activation reads and writes per token per sublayer, excluding parameters, small coefficient buffers, and attention/FFN internals. Here, $Q_{\mathrm{corr}}(K)$ denotes correction traffic averaged over a group; Appendix D.3 derives the counts.

Larger groups reduce wide-state accesses but require more preceding-output reads for corrections. At $n = 4$, total traffic is $11.5C$, $9C$, $9.25C$, and $12.375C$ for $K = 2, 4, 8, 16$, respectively, so it does not decrease monotonically. Our choice of $K = 4$ halves the $18C$ traffic of fused mHC and identityHC. Removing mixing alone does not reduce these accesses: fused mHC already combines mixing and output allocation in one write-back. The reduction comes from sharing state access across sublayers. Saving one group-entry residual state instead of K sublayer-entry states also reduces saved residual-state memory by 75% at $K = 4$.

5 Experiments

5.1 Setup

Training setup. We conduct experiments on a MoE model with 15B total and 1B activated parameters (15B-A1B), trained on 200B tokens, and evaluate scalability on 30B-A2B trained on 400B tokens. Both use MLA (Liu et al., 2024a), alternating one full-attention with three sliding-window layers, and MoE layers (Dai et al., 2024). Training uses sequence length 8192 and Muon (Liu et al., 2025) on a high-quality mixture of English and Chinese text, code, and mathematics. Unless specified, we use $n = 4$ streams and groups of $K = 4$ sublayers. Appendix B.1 details configurations.

Evaluation. We evaluate knowledge, reasoning, math, code, and Chinese on MMLU (Hendrycks et al., 2020), HellaSwag (Zellers et al., 2019), MATH (Hendrycks et al., 2021), MBPP (Austin et al., 2021), and C3 (Sun et al., 2020) for the 15B-A1B studies (Tables 2 and 3). For 30B-A2B (Table 4), we add MMLU-Pro (Wang et al., 2024), MMLU-Redux (Gema et al., 2025), TriviaQA

Table 3: **Design of shared memory access.** K: sublayers per group. Independent/Reused: layer-specific/shared read/write gates. Row 8 omits corrections but retains group-boundary write-back.

Method	Memory access	Group size K	Mixing	Gates	Loss ↓	MMLU	HellaSwag	MATH	MBPP	C3	Avg.
(1) Vanilla Residual	—	—	—	—	1.678	56.1	65.8	26.2	37.8	75.9	52.4
Reference configurations											
(2) mHC	Per sublayer	1	Sinkhorn	Independent	**1.659**	58.8	**67.3**	26.9	**39.8**	**82.2**	55.0
(3) identityHC	Per sublayer	1	Identity	Independent	1.660	**59.6**	66.4	**29.6**	39.6	81.9	**55.4**
Sharing granularity											
(4) Share 2	Shared	2	Identity	Independent	**1.660**	**59.9**	**67.0**	28.1	39.0	80.1	54.8
(5) SimpleHC (Share 4)	Shared	4	Identity	Independent	**1.660**	58.5	**67.0**	**28.7**	**40.8**	**80.2**	**55.0**
(6) Share 8	Shared	8	Identity	Independent	1.662	59.1	66.2	27.9	39.4	79.1	54.3
Layer-specific gating											
(7) Share 4, reused gates	Shared	4	Identity	Reused	1.667	58.1	67.1	28.4	35.4	78.4	53.5
Within-group update corrections											
(8) Share 4 w/o corrections	Shared	4	Identity	Independent	1.705	52.8	64.4	21.7	32.0	68.8	47.9

(Joshi et al., 2017), DROP (Dua et al., 2019), BBH (Suzgun et al., 2023), GSM8K (Cobbe et al., 2021), HumanEval (Chen et al., 2021), CMMLU (Li et al., 2024), and CEval (Huang et al., 2023), yielding fourteen benchmarks. Appendix B.3 details few-shot settings and loss reporting.

5.2 Understanding the Hyper-Connections Architecture

We examine the roles and interactions of read/write gates and residual mixing in the HC architecture (Table 2). HC and mHC share these components but differ in gate conditioning, parameterization, and mixing constraints. Except for HC variants, gates follow the parameterization in Section 3.1.

We observe: (i) **Unconstrained mixing is effective, but Sinkhorn diminishes its gains.** Retaining unconstrained mixing improves the average score in HC (rows 3 vs. 2). Under the mHC parameterization, unconstrained mixing achieves 56.2, versus 55.0 with Sinkhorn constraints and 55.4 with identity transitions (rows 4–6). Sinkhorn thus limits the benefit of mixing, leaving performance close to identityHC. (ii) **Three components form two effective groups: read/write gating and unconstrained mixing.** Without mixing, jointly learned read/write gates achieve an average score of 55.4; fixing either gate lowers it significantly, supporting their treatment as one group. Unconstrained mixing forms the other: with both gates fixed, it achieves 56.0 (row 12). Either group alone is comparable to mHC; combining them further improves the score (row 4). (iii) **Full-state conditioning improves model quality.** With full-state inputs, loss is 1.660. Restricting read/write gates to stream-local inputs raises loss by 0.001/0.003 (rows 6–8), suggesting slightly greater sensitivity in write gates. Section 6.1 analyzes these findings and their implications for SimpleHC.

5.3 Shared Memory Access with Layer-Specific Gating

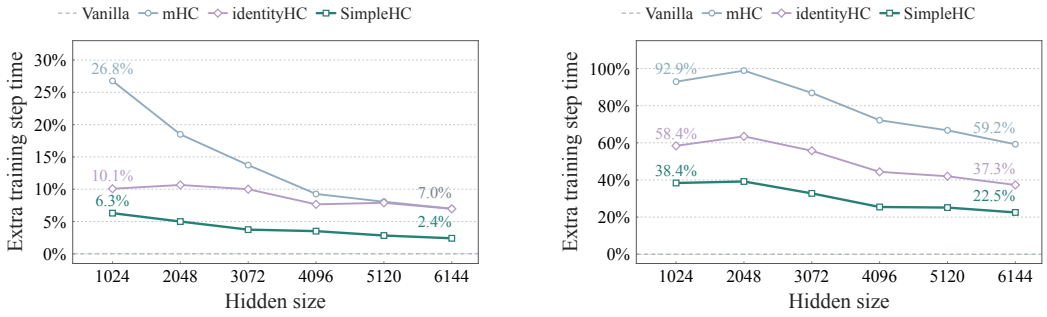
Table 3 ablates group size, gate reuse, and scalar corrections, with identityHC and mHC as references. The results highlight three findings: (i) **Moderate state sharing preserves model quality.** $K = 2, 4$ match the loss of identityHC, while $K = 8$ slightly increases it (rows 3–6). At $K = 4$, the average score of 55.0 is comparable to mHC and identityHC. We choose $K = 4$ for its competitive quality and lowest memory traffic (Section 4.3). (ii) **Layer-specific gates improve quality under shared inputs.** At $K = 4$, reusing one read/write gate pair lowers the average score from 55.0 to 53.5 (rows 5 vs. 7). This supports retaining layer-specific gate parameters. (iii) **Scalar corrections are important for model performance.** Removing corrections raises loss from 1.660 to 1.705 despite retaining all outputs in the final write-back (rows 5 vs. 8). Thus, later sublayers must incorporate preceding outputs even when gate inputs are shared. Section 6.2 analyzes these findings.

5.4 Evaluating SimpleHC at Larger Scale

On the larger 30B-A2B model, SimpleHC matches mHC performance while simplifying both the residual structure and execution. As shown in Table 4, it achieves the same loss of 1.561 as mHC and identityHC and an average benchmark score of 63.1 versus 62.9 for mHC and 62.8 for identityHC. It achieves higher scores on MMLU, MATH, and C3, although it trails mHC on GSM8K and both methods on HumanEval. These results support the scalability of SimpleHC.

Table 4: **Downstream evaluation on 30B MoE models.** We compare Vanilla, mHC, identityHC, and SimpleHC across benchmarks covering Knowledge, Reasoning, Math & Code, and Chinese.

	Benchmark	Vanilla	mHC	identityHC	SimpleHC
	Loss ↓	1.579	**1.561**	**1.561**	**1.561**
Knowledge	MMLU	65.8	67.7	67.9	**68.6**
	MMLU-Pro	45.6	48.6	48.0	**48.9**
	MMLU-Redux	62.4	64.3	64.1	**64.7**
	TriviaQA	65.1	**69.7**	68.4	68.1
Reasoning	HellaSwag	71.3	72.4	72.8	**73.0**
	DROP	61.4	67.2	67.7	**68.1**
	BBH	59.4	**64.3**	62.7	63.6
Math & Code	GSM8K	71.3	**75.0**	74.5	74.5
	MATH	38.4	38.4	38.9	**41.5**
	HumanEval	39.6	47.6	**48.2**	45.1
	MBPP	46.2	49.4	**49.8**	49.2
Chinese	CMMLU	55.1	**60.7**	58.7	59.0
	CEval	60.6	65.7	**68.2**	67.3
	C3	88.1	89.6	89.6	**91.4**
	Average	59.3	62.9	62.8	**63.1**



(a) Efficient implementations (b) PyTorch implementations

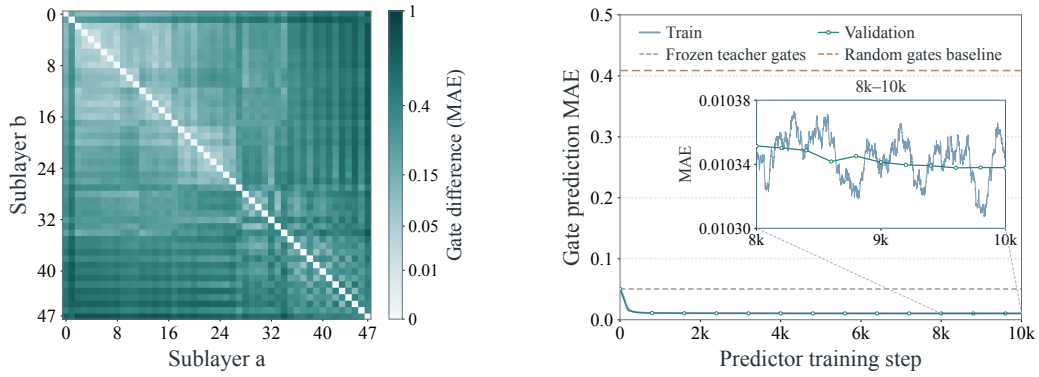
Figure 3: **End-to-end training overhead relative to Vanilla Residual.** SimpleHC achieves the lowest overhead across tested model widths with both (a) efficient and (b) PyTorch implementations.

5.5 Translating Simplicity into Training Efficiency

We compare end-to-end training overhead against the baseline across scales from $C = 1024$ to $C = 6144$ (matching frontier models (Xu et al., 2026)). Details are provided in Appendix B.2.

With the efficient implementations, SimpleHC achieves the lowest overhead across all tested scales (Figure 3(a)). At the smallest scale, overhead is 6.3%, versus 26.8% for mHC and 10.1% for identityHC. Despite matching forward activation traffic, mHC incurs additional mixing and Sinkhorn costs, explaining its larger gap from identityHC at smaller scales. Relative overhead generally decreases with width: HC operations scale linearly at fixed n, whereas expert projections scale quadratically when expert width grows proportionally. At the largest tested scale, SimpleHC cuts overhead from approximately 7.0% for both baselines to 2.4%, eliminating about two-thirds. Specialized kernels require substantial engineering effort, limiting wider adoption. Thus we also evaluate PyTorch versions (Figure 3(b)), where SimpleHC has the lowest overhead across all tested scales.

Saved activation memory. By reusing group-entry states, SimpleHC reduces estimated storage retained for backward at sublayer boundaries and within HC connections by approximately 31% relative to mHC (Figure 1(b)). Appendix B.2 details the accounting.



(a) Write-gate variation across sublayers

(b) Prediction from the group-entry state

Figure 4: **Gates vary across sublayers yet are predictable from shared inputs.** (a) Pairwise gate MAE across identityHC sublayers, averaged over streams and tokens. (b) Train and val. MAE for write-gate prediction at offsets 1–3 from the group-entry state. See Appendix F.3 for details.

6 Analysis

We analyze the design and training stability of SimpleHC. Three conclusions emerge: (i) **learned read/write gating provides a minimal effective residual structure**; (ii) **gating decisions vary across sublayers yet remain predictable from a shared group-entry state**; (iii) **SimpleHC remains stable under stress and exhibits smoother gradient dynamics during standard training**.

6.1 A Minimal Structure for Hyper-Connections

To combine architectural simplicity, competitive performance, and robust scaling, we select learned read/write gating without mixing as a minimal effective residual structure for Hyper-Connections. This avoids the propagation risks of cumulative residual mixing, as analyzed below.

Learned read/write gating and unconstrained mixing form two effective groups: each achieves performance comparable to mHC, with further gains when combined (Table 2). Appendix E compares their weighting mechanisms. However, unconstrained mixing repeatedly transforms earlier contributions through residual matrix products, risking cumulative amplification and training instability (Xie et al., 2025). We observe unconstrained mixing with fixed gates produces larger peak residual-path coefficients than identityHC, consistent with these amplification risks (Figure 6, Appendix F.1).

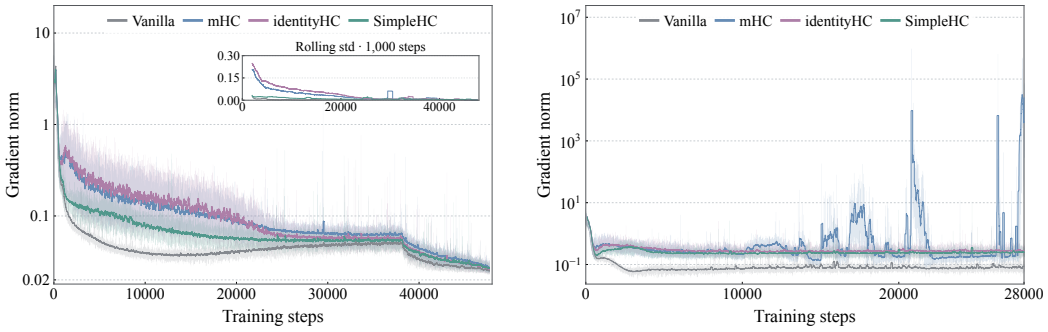
Sinkhorn constraints mitigate these risks (Xie et al., 2025), but constrained mixing adds little benefit over learned read/write gating alone. We therefore retain learned gates without mixing and use full-state inputs for better loss and benchmark scores (Table 2). Full-state and stream-local conditioning incur the same residual-state input traffic, as both read all streams.

6.2 Why Shared Memory Access Is Effective

We examine why shared memory access benefits from layer-specific gates and remains effective with shared inputs through gate variation and predictability analyses.

Gate variation across sublayers. Read and write gates exhibit distinct allocation patterns across sublayers of the 15B identityHC model (Figures 7(a) and 4(a)). These differences motivate retaining layer-specific gates under shared inputs, consistent with their advantage over gate reuse (Table 3).

Predictability from the group-entry state. We examine whether subsequent gates are predictable from the group-entry state to assess the feasibility of sharing gate inputs. We train layer-specific predictors to recover subsequent read/write gates of a frozen 15B identityHC model from the group-entry state. The predictors use the SimpleHC parameterization and are initialized from the corresponding teacher gate parameters. Only predictors are trained. The "Frozen teacher gates" baseline uses these mappings without training. In Figure 4(b), it predicts write gates more accurately than uniform random gates (MAE 0.0504 vs. 0.4086), indicating predictive ability from earlier inputs even without adaptation. Training further reduces MAE to 0.0103, suggesting that layer-specific



(a) Standard 15B training (b) Stress test

Figure 5: **Training stability of SimpleHC.** (a) Inset: 1,000-step rolling gradient-norm std. (b) Under stress, mHC exhibits severe gradient spikes, while SimpleHC and identityHC remain stable.

mappings closely approximate subsequent gates from shared inputs and supporting the shared memory access design. Read gates show the same trend (Figure 7(b)). Appendix F.3 details the protocol.

Sharing gate inputs does not remove the need for sequential output contributions: removing scalar corrections raises loss from 1.660 to 1.705 despite retaining the final write-back.

6.3 Does Simplification Compromise Training Stability?

Training stability is essential for reliable scaling, as instabilities absent in smaller models can emerge at larger scales (Wortsman et al., 2024). We examine whether the simplifications in SimpleHC compromise training stability. During 15B training, SimpleHC shows smaller gradient-norm fluctuations and lower rolling standard deviation than mHC and identityHC early after warmup (Figure 5(a)); all three reach comparable ranges later. Shared-input gating may reduce sensitivity to within-group state fluctuations by conditioning decisions on an earlier state, loosely analogous to stabilization through historical averaging. This could contribute to smoother training of SimpleHC.

To emulate large-scale training stability challenges, we stress-test 230B-A6B MoE models at $10\times$ the reference learning rate. In Figure 5(b), mHC exhibits gradient spikes reaching approximately 1.9×10^6, while SimpleHC and identityHC maintain bounded gradient norms. Appendix G details the settings and analyzes potential causes of mHC instability.

7 Related Work

Hyper-Connections expand the residual pathway into multiple streams with learned read, write, and mixing operations (Zhu et al., 2025). Subsequent work constrains mixing to doubly stochastic matrices (Xie et al., 2025), reduces constraint-enforcement costs (Yang & Gao, 2026; Zhou et al., 2026; Dandachi & Diggs-Galligan, 2026), or explores spectral and orthogonal constraints for stable propagation and stream diversity (Liu et al., 2026; Guo et al., 2026). Other designs modify read/write structures (Qiu et al., 2026a; Zheng et al., 2026) or use sparse updates to support more streams (Zhang et al., 2026). However, HC components remain insufficiently understood, and training overhead remains non-negligible. Our study clarifies their interactions and motivates SimpleHC, which simplifies both residual structure and execution. See Appendix A for additional related work.

8 Conclusion

In this work, we study how the components of the Hyper-Connections architecture contribute to model performance and training cost. Our study identifies learned read/write gating as a minimal effective structure, while unconstrained mixing offers gains with propagation risks. Beyond removing mixing, we introduce shared memory access to reduce repeated residual-state accesses, with gate analyses supporting layer-specific mappings conditioned on shared inputs. Together, they form SimpleHC, simplifying residual structure and execution. SimpleHC matches mHC performance on 15B and 30B MoE models, reduces additional training overhead from 7.0% to 2.4% at the largest tested scale, and exhibits smoother gradient dynamics while remaining stable under stress. SimpleHC thus combines architectural simplicity, execution efficiency, training stability, and scalability.

AI USE STATEMENT

We used generative AI tools to assist with drafting and polishing the manuscript, including improving clarity, organization, and the presentation of experimental analyses. We developed the core research ideas and take responsibility for the final content and conclusions.

ETHICS STATEMENT

This work studies residual architectures and training efficiency for language models. Although it does not target specific downstream applications, more efficient training may facilitate broader deployment, with associated concerns about bias, privacy, and misuse. Responsible deployment requires evaluation beyond the architectural and training analyses presented here.

REPRODUCIBILITY STATEMENT

We describe the SimpleHC formulation and execution scheme in the main text, together with implementation details. Model configurations, training settings, and evaluation protocols are provided in Appendix B. Appendix C establishes execution equivalence, while Appendices F and G document the gate-analysis and stability experiments. We will release the code upon acceptance.

REFERENCES

Ekaterina Alimaskina, Gleb Molodtsov, and Aleksandr Beznosikov. Analyzing stream collapse in hyper-connections: From diagnosis to mitigation. *arXiv preprint arXiv:2606.03483*, 2026.

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

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

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

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1280–1297, 2024.

Torque Dandachi and Sophia Diggs-Galligan. go-m hc: Direct parameterization of manifold-constrained hyper-connections via generalized orthostochastic matrices. *arXiv preprint arXiv:2604.02309*, 2026.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2368–2378, 2019.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, et al. Are we done with mmlu? In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5069–5096, 2025.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Haoqiang Guo, Xuyi Chen, Bo Ke, Yishu Lei, Ziyang Xu, Shikun Feng, Wenhan Luo, et al. ohc: Orthogonal hyper-connections on so (4) via quaternions. *arXiv preprint arXiv:2609.02672*, 2026.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

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

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

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

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, 2017.

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

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024a.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024b.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

Zhaoyi Liu, Haichuan Zhang, and Ang Li. Beyond the birkhoff polytope: Spectral-sphere-constrained hyper-connections. *arXiv preprint arXiv:2603.20896*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Valentijn Oldenburg, Floris de Kam, Bente Zuijdam, Lieve Eberson, Nicky van Zutphen, Stef de Wildt, and Ivo Verhoeven. Manifold-constrained hyper-connections for parameter-efficient finetuning. *arXiv preprint arXiv:2607.18130*, 2026.

Matteo Pagliardini, Amirkeivan Mohtashami, Francois Fleuret, and Martin Jaggi. Denseformer: Enhancing information flow in transformers via depth weighted averaging. *Advances in neural information processing systems*, 37:136479–136508, 2024.

William Peng, Josheev Rai, Kevin Tseng, Siwei Wang, and Sean Wu. Ablate and rescue: A causal analysis of residual stream hyper-connections. *arXiv preprint arXiv:2603.14833*, 2026.

Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bochao Mao, Chengruidong Zhang, et al. On the design of qwen3. 8-next architecture: Evaluation, efficiency, and training stability. *arXiv preprint arXiv:2608.30320*, 2026a.

Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, et al. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. *Advances in Neural Information Processing Systems*, 38:100092–100118, 2026b.

Kai Sun, Dian Yu, Dong Yu, and Claire Cardie. Investigating prior knowledge for challenging chinese machine reading comprehension. *Transactions of the Association for Computational Linguistics*, 8:141–155, 2020.

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

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

Mitchell Wortsman, Peter Liu, Lechao Xiao, Katie Everett, Alexander Alemi, Ben Adlam, John D Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, et al. Small-scale proxies for large-scale transformer training instabilities. In *International Conference on Learning Representations*, volume 2024, pp. 49844–49869, 2024.

Da Xiao, Qingye Meng, Shengping Li, and Xingyuan Yuan. Muddformer: Breaking residual bottlenecks in transformers via multiway dynamic dense connections. *arXiv preprint arXiv:2502.12170*, 2025.

Zhenda Xie, Yixuan Wei, Huanqi Cao, Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Jiang Chang, Kuai Yu, et al. mhc: Manifold-constrained hyper-connections. *arXiv preprint arXiv:2512.24880*, 2025.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Yongyi Yang and Jianyang Gao. mhc-lite: You don't need 20 sinkhorn-knopp iterations. *arXiv preprint arXiv:2601.05732*, 2026.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Xiangdong Zhang, Xiaohan Qin, Sunan Zou, Tuo Dai, Xiaoming Shi, Huaijin Wu, Yebin Yang, Zhuo Xia, Shaofeng Zhang, Lin Yao, et al. xhc: Expanded hyper-connections. *arXiv preprint arXiv:2607.14530*, 2026.

Zhizhan Zheng, Feiyun Zhang, Shuchun Liu, Tian Xia, Xi Liu, Dasheng Hu, and Hongquan Zhou. Multi-gate residuals. *arXiv preprint arXiv:2605.23259*, 2026.

Wuyang Zhou, Yuxuan Gu, Giorgos Iacovides, and Danilo Mandic. Kromhc: Manifold-constrained hyper-connections with kronecker-product residual matrices. *arXiv preprint arXiv:2601.21579*, 2026.

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

A Additional Related Work

Understanding and Simplifying Hyper-Connections. Recent analyses reveal uneven stream utilization, redundancy, and concentrated read/write routing in trained HC models (Peng et al., 2026; Alimaskina et al., 2026). Prior work also reports that removing residual mixing can largely preserve model quality (Guo et al., 2026), with related evidence from finetuning (Oldenburg et al., 2026). These observations motivate examining which interactions account for the benefits. Our study identifies learned read/write gating and unconstrained mixing as two independently effective interaction groups, while showing that Sinkhorn constraints diminish the gains from mixing. These findings explain why learned gating alone remains competitive with mHC (Xie et al., 2025) and motivate an additive structure whose execution we further simplify through shared memory access.

Efficient Execution of Residual Aggregation. Richer residual pathways introduce storage and data-movement costs beyond their arithmetic overhead. Prior work reduces these costs through block-wise aggregation and staged computation (Team et al., 2026), kernel fusion and recomputation (Xie et al., 2025; Zheng et al., 2026), and optimized multi-stream execution (Zhang et al., 2026). SimpleHC further reduces repeated residual-state access across sublayers by sharing gate inputs while retaining layer-specific parameters. Together with the additive residual structure, this permits consolidated reads and deferred writes, with scalar corrections preserving sequential output contributions for the chosen gates.

B Experimental Settings

This section provides additional details on the experimental settings. Tables 2 and 3 use the 15B-A1B setup, and Table 4 uses the 30B-A2B setup. Efficiency measurements use the separate configurations in Appendix B.2. Gate-analysis and stability figures are documented in Appendices F and G, respectively.

B.1 Model and Training Configurations

Table 5 reports the backbone architectures and training recipes for the 15B-A1B design-space studies and the 30B-A2B scaling experiments. Parameter counts are rounded; activated counts exclude embedding and output layers. Both backbones apply headwise sigmoid gating to attention outputs before the output projection (Qiu et al., 2026b).

Training hyperparameters. Table 5 summarizes the model configurations and training hyperparameters. Muon updates backbone matrix weights in attention and MLP/MoE modules; AdamW updates embeddings, output weights, normalization parameters, and HC gate parameters. We use a warmup–stable–decay (WSD) (Hu et al., 2024) learning-rate schedule with 1,000 warmup steps, followed by a constant learning rate and arccos decay over the final 20% of training.

For the gate-reuse variant in Table 3, each group uses one learned read/write gate pair instead of four layer-specific pairs. The gates are computed from the group-entry state and reused across all sublayers.

B.2 Efficiency Benchmark Settings

We evaluate training efficiency across model scales by varying hidden width from 1024 to 6144, as summarized in Table 6. To limit memory requirements and avoid introducing parallelism changes, all benchmark models use eight Transformer layers. This allows comparisons across widths without the resource demands of full-depth models. At each width, residual variants share the backbone, batch sizes, and parallel configuration. Since both residual operations and backbone computation repeat across layers, these measurements provide a representative estimate of relative training overhead in deeper models with the same parallel configuration.

All models use MLA with one full-attention layer followed by three sliding-window layers of window size 512. The first layer has a dense FFN; the remaining seven use 256 routed experts with top-8 routing and one shared expert. We use BF16, a sequence length of 8192, and a global batch size of 256. CUDA Graph is enabled for all methods, with activation recomputation disabled. HC variants use four residual streams, and SimpleHC groups four sublayers.

Table 5: **Model configurations and training hyperparameters.** Residual streams apply to HC variants, and group size applies to SimpleHC, unless varied in the corresponding experiments.

Attribute	**15B-A1B**	**30B-A2B**
Total parameters	15B	30B
Activated parameters	1B	2B
Transformer layers	24	32
Hidden dimension	1792	2048
Vocabulary size	152,064	152,064
Attention	MLA	MLA
Full / sliding-window ratio	1:3	1:3
Sliding-window size	512	512
Attention heads	40	56
Experts (routed / active / shared)	256 / 8 / 1	256 / 8 / 1
Expert intermediate dimension	448	576
Residual streams n	4	4
Group size K (SimpleHC)	4	4
Sequence length	8192	8192
Global batch size	512	768
Training tokens	200B	400B
Optimizer	Muon / AdamW	Muon / AdamW
Peak learning rate	9×10^{-4}	7.4×10^{-4}
Learning-rate schedule	WSD, arccos decay	WSD, arccos decay
Warmup steps	1,000	1,000
Decay fraction	20%	20%
Weight decay	0.1	0.1
Gradient clipping	1.0	1.0
Training precision	BF16	BF16

Table 6: **Model configurations for efficiency measurements.** Each row defines an eight-layer model. FFN widths denote intermediate dimensions; the shared expert uses the same width as each routed expert. Microbatch sizes count sequences and are shared across methods at each width.

Width	Heads	Query rank	Dense FFN	Expert FFN	Microbatch
1024	24	256	2880	320	8
2048	48	384	5728	608	4
3072	80	640	8608	928	2
4096	96	768	11456	1216	1
5120	128	1024	14336	1536	1
6144	160	1280	17216	1856	1

Saved activation storage accounting. We estimate storage retained for backward after one microbatch forward pass, covering sublayer inputs and outputs, residual states and copies, base reads, HC auxiliary tensors, and the decoder output before final normalization (Table 7). Shared storage is counted once, while distinct copies are counted separately. The accounting excludes attention/FFN internals, LM-head/loss buffers, model and optimizer states, temporary workspaces, and allocator/CUDA Graph reservations. These source-level estimates are not total activation or peak device memory. Figure 1(b) uses microbatch size one; both estimates scale linearly with microbatch size, leaving the relative reduction unchanged.

The models use eight layers (16 sublayers), $n = 4$, and no recomputation; SimpleHC uses four groups with $K = 4$. For T local microbatch tokens, let $U = 2TC$ bytes denote one BF16 narrow activation. The mHC residual inputs occupy $U + 15 \times 4U = 61U$: the first uses a broadcast view of the dense input, followed by 15 wide inputs. SimpleHC retains two wide copies for gate projection and base reads in the first group, and one per later group, totaling $8U + 3 \times 4U = 20U$. Each group jointly allocates four base reads; the first sublayer input keeps the entire allocation live through a view. Beyond that input, already counted in the input row, the remaining slices contribute $3U$ per

Table 7: **Estimated activation storage retained for backward at sublayer boundaries and within HC connections.** The example uses eight Transformer layers, $C = 2048$, and one 8192-token sequence; $1\,\mathrm{MiB} = 2^{20}$ bytes. Shared storage is counted once, including allocations retained by views.

	Storage (bytes)		Example (MiB)	
Component	mHC	SimpleHC	mHC	SimpleHC
Pre-normalization sublayer inputs	$16U$	$16U$	512	512
Residual states and input copies	$61U$	$20U$	1952	640
Sublayer branch outputs	$16U$	$16U$	512	512
Additional base-read storage	0	$12U$	0	384
Collapsed decoder output	U	U	32	32
HC auxiliary tensors	$4672T$	$1648T$	36.5	12.875
Total	$94U + 4672T$	$65U + 1648T$	3044.5	2092.875

group, or $12U$ overall. identityHC has a similar saved-storage footprint to mHC, as both retain wide residual states at each sublayer.

HC auxiliary tensors are FP32 and include projections, gates, inverse-RMS statistics, and mixing or correction intermediates. The mHC path retains 73 elements per token per sublayer ($4672T$ bytes overall); SimpleHC retains 103 per token per group ($1648T$ bytes), including contiguous gate copies and correction coefficients. At $C = 2048$ and $T = 8192$, total storage falls from 3044.5 to 2092.875 MiB, a 31.26% reduction. Across the six evaluated widths, reductions range from 31.0% to 31.7%, supporting the rounded 31% summary.

B.3 Evaluation Protocols

We report validation loss on the Pile test split (Gao et al., 2020) in Tables 2, 3, and 4.

For the 15B-A1B component studies, we evaluate MMLU (Hendrycks et al., 2020) in a 5-shot setting, HellaSwag (Zellers et al., 2019) in a 10-shot setting, MATH (Hendrycks et al., 2021) in a 4-shot setting, and MBPP (Austin et al., 2021) and C3 (Sun et al., 2020), both in a 3-shot setting. For the 30B-A2B evaluation, we retain these settings and add MMLU-Pro (Wang et al., 2024), MMLU-Redux (Gema et al., 2025), TriviaQA (Joshi et al., 2017), CMMLU (Li et al., 2024), and C-Eval (Huang et al., 2023) with 5-shot evaluation; BBH (Suzgun et al., 2023) and DROP (Dua et al., 2019) with 3-shot evaluation; GSM8K (Cobbe et al., 2021) with 4-shot evaluation; and HumanEval (Chen et al., 2021) with zero-shot evaluation.

C Exact Reconstruction with Scalar Corrections

Algorithm 1 summarizes the forward pass for one group. Matrix columns use zero-based offsets within the group.

We prove that scalar reconstruction and deferred write-back are equivalent to sequential execution with the same shared-input gates. Both executions use Equation 3, identical parameters, and identical random choices for any stochastic operations. Define the sequential reference by $\widetilde{X}_L = X_L$ and

$$\widetilde{u}_j = \widetilde{X}_j^\top \bar{h}_j^{\mathrm{pre}}, \qquad \widetilde{o}_j = F_j(\widetilde{u}_j), \qquad \widetilde{X}_{j+1} = \widetilde{X}_j + \bar{h}_j^{\mathrm{post}} \widetilde{o}_j^\top . \tag{6}$$

Forward equivalence. At $j = L$, Equation 4 has no correction terms, so $u_L = \widetilde{u}_L$ and $o_L = \widetilde{o}_L$. Assume $o_k = \widetilde{o}_k$ for $L \le k < j$. Expanding the reference updates and applying the read decomposition in Equation 2 gives

$$\widetilde{u}_j = X_L^\top \bar{h}_j^{\mathrm{pre}} + \sum_{k=L}^{j-1} \big((\bar{h}_j^{\mathrm{pre}})^\top \bar{h}_k^{\mathrm{post}} \big) \widetilde{o}_k = u_j. \tag{7}$$

Thus $\widetilde{o}_j = o_j$, completing the induction. Summing the reference updates also yields exactly the group-boundary state in Equation 5.

Algorithm 1 SimpleHC forward pass for one group

Require: Entry state $X_L \in \mathbb{R}^{n\times C}$; group size K; gate functions $g_j^{\mathrm{pre}}, g_j^{\mathrm{post}}$ and sublayers F_j for $L \le j < L+K$
Ensure: Group-boundary state X_{L+K}

$\bar{H}^{\mathrm{pre}} \leftarrow [g_L^{\mathrm{pre}}(X_L), \ldots, g_{L+K-1}^{\mathrm{pre}}(X_L)]$
$\bar{H}^{\mathrm{post}} \leftarrow [g_L^{\mathrm{post}}(X_L), \ldots, g_{L+K-1}^{\mathrm{post}}(X_L)]$ ▷ Independent gate columns, shared input
$V \leftarrow X_L^\top \bar{H}^{\mathrm{pre}}$ ▷ Batched base reads; $V \in \mathbb{R}^{C\times K}$
$D \leftarrow \mathrm{tril}((\bar{H}^{\mathrm{pre}})^\top \bar{H}^{\mathrm{post}}, -1)$ ▷ Strictly lower-triangular read/write coefficients
for $a = 0, \ldots, K-1$ **do**
  $u_{L+a} \leftarrow V_{:,a} + \sum_{b=0}^{a-1} D_{a,b}\, o_{L+b}$ ▷ Empty sum is zero
  $o_{L+a} \leftarrow F_{L+a}(u_{L+a})$
end for
$O \leftarrow [o_L, \ldots, o_{L+K-1}]$
$X_{L+K} \leftarrow X_L + \bar{H}^{\mathrm{post}} O^\top$ ▷ Materialize the group-boundary state
return X_{L+K}

Gradient equivalence. The two executions define the same function of X_L and the parameters, so their derivatives agree wherever this function is differentiable. Gradients must include the dependence of every gate on X_L; detaching gates would invalidate this equivalence. Nondifferentiable operations require consistent derivative conventions. The equivalence holds in exact arithmetic; floating-point results may differ with reduction order and precision. It does not assert equivalence to gates conditioned on each updated sublayer state.

D Implementation and Memory Accounting Details

This section details the forward and backward kernels described in Sections 4.1 and 4.2. These kernels are implemented using DeepGEMM and Triton.

D.1 Forward Fusion with Shared State Access

Grouped reads. Grouped gate generation and batched base reads implement the read side of shared memory access. For $x = \mathrm{vec}(X_L) \in \mathbb{R}^{nC\times 1}$ and independent gate projections concatenated into $W \in \mathbb{R}^{nC\times 2Kn}$, we compute

$$p = W^\top x, \qquad s = x^\top x, \qquad V = X_L^\top \bar{H}^{\mathrm{pre}}. \tag{8}$$

Here $p \in \mathbb{R}^{2Kn\times 1}$, s is scalar, and $V \in \mathbb{R}^{C\times K}$. One kernel computes p, s from the same loaded state. Gate normalization uses affine-free RMSNorm, with $r = (s/(nC) + \epsilon)^{-1/2}$ computed from the input state. For each sublayer j and gate type $t \in \{\mathrm{pre}, \mathrm{post}\}$, the corresponding projection block p_j^t gives logits $\ell_j^t = \alpha_j^t r p_j^t + b_j^t$. A second kernel fuses this scaling and bias addition with gate activations, avoiding a normalized-state buffer. At $n = K = 4$, one $4C \times 32$ projection supplies four read gates and four write gates. A separate batched-read kernel reuses each loaded state tile for all K distinct bases, storing them contiguously per sublayer. Gate generation and base aggregation each reduce full-state reads from KnC to nC elements per token per group.

Fused corrections and deferred write-back. We implement Equations 4 and 5 without stacked output copies or expanded update buffers. Using precomputed coefficients D, each correction kernel loads preceding C-dimensional outputs through separate pointers and fuses weighting, summation, and base addition. At $K = 4$, three correction kernels supply the sequential sublayer inputs; one group-exit kernel then fuses all four output allocations with residual addition. Wide-state update traffic falls from $2KnC$ to $2nC$ elements per token per group.

D.2 Backward Reuse and Efficient Reductions

We detail the backward computations for activation and gate gradients, including shared state access and token-packed reductions.

For loss $\mathcal{L}$, let $G = \nabla_V \mathcal{L} \in \mathbb{R}^{C\times K}$, $O = [o_L, \ldots, o_{L+K-1}] \in \mathbb{R}^{C\times K}$, and $E = \nabla_{X_{L+K}} \mathcal{L} \in \mathbb{R}^{n\times C}$. The following equations give gradient contributions from the base reads and final write-back, with $+=$ denoting accumulation.

Shared activation-gradient access. We compute activation-gradient contributions jointly across the group:

$$\nabla_{X_L} \mathcal{L} \mathrel{+}= E + \bar{H}^{\text{pre}} G^\top, \qquad \nabla_O \mathcal{L} \mathrel{+}= E^\top \bar{H}^{\text{post}}. \tag{9}$$

The read backward fuses all K contributions with the identity gradient E, avoiding K expanded gradients and their merge. Reusing one scan of E for all K output gradients reduces its logical reads from KnC to nC elements per token per group ($4\times$ at $K = 4$). We also fuse projection input gradients with RMS differentiation, avoiding a wide intermediate and a separate normalization-backward pass; weight gradients use a separate GEMM.

Token-packed gate gradients. Gate gradients instead require reductions over the channel dimension:

$$\nabla_{\bar{H}^{\text{pre}}} \mathcal{L} \mathrel{+}= X_L G, \qquad \nabla_{\bar{H}^{\text{post}}} \mathcal{L} \mathrel{+}= EO. \tag{10}$$

At $n = K = 4$, each product reduces C channels to just 16 coefficients per token. We pack four tokens into a 16×16 Tensor Core product, accumulate over channels on-chip, and store only the four same-token 4×4 blocks, eliminating the partial-gradient scratch buffer and extra reduction launch of the channel-tiled fallback for each product.

D.3 Memory-Access and Storage Accounting

The accounting in Table 1 covers logical forward activation reads and writes per token per sublayer, excluding parameters, small coefficient buffers, and attention/FFN internals. Naive mHC materializes mixed states and expanded updates separately; fused mHC combines residual mixing, output allocation, and residual addition.

At group offset $a > 0$, each correction reads one base and a preceding outputs, then writes one input, costing $(a + 2)C$ elements. Offset $a = 0$ uses its base directly. Averaging over K sublayers gives

$$Q_{\text{corr}}(K) = \frac{C}{K} \sum_{a=1}^{K-1} (a + 2) = \frac{C(K - 1)(K + 4)}{2K}, \qquad Q_{\text{corr}}(4) = 3C. \tag{11}$$

Grouped gate generation and base reads each access nC elements per group; final write-back reads and writes the wide state once, adding $2nC$. Together these contribute $4nC/K$ per sublayer. Base writes and output reads for final write-back add $2C$, yielding

$$Q_{\text{SimpleHC}}(K) = \frac{4nC}{K} + 2C + Q_{\text{corr}}(K). \tag{12}$$

At $n = 4$, this gives $11.5C$, $9C$, $9.25C$, and $12.375C$ for $K = 2, 4, 8, 16$, respectively. Thus, $K = 4$ has the lowest logical traffic among these group sizes, rather than over all positive integers. Correction costs grow with K, offsetting the reduced wide-state traffic at larger group sizes.

Without recomputation, retaining one wide group-entry state instead of K sublayer-entry states reduces this storage by $1 - 1/K$, or 75% at $K = 4$. This count concerns sublayer-entry residual states only.

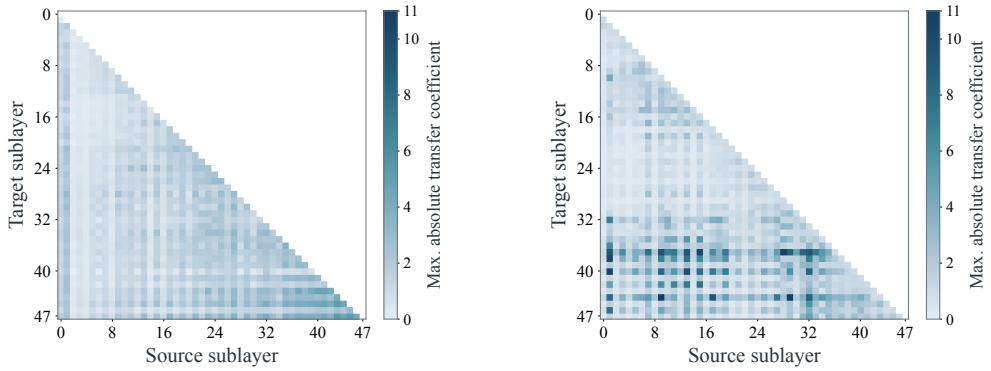
E Read/Write Gating and Unconstrained Mixing

We examine the weighting mechanisms of the two interaction groups in Table 2 to interpret their comparable empirical performance. Both control how earlier outputs contribute to later inputs, but place learnable flexibility at different points along the residual path. For a fixed token, the residual-path coefficient carrying output o_k into a later input u_j is

$$c_{j,k} = (h_j^{\text{pre}})^\top \Phi_{k+1\to j} h_k^{\text{post}}, \qquad \Phi_{k+1\to j} = H_{j-1}^{\text{res}} \cdots H_{k+1}^{\text{res}}, \tag{13}$$

where an empty product is I. With learned read/write gates and identity transitions, this reduces to

$$c_{j,k} = (h_j^{\text{pre}})^\top h_k^{\text{post}}. \tag{14}$$



(a) Learned gates with identity transitions

(b) Unconstrained mixing with fixed gates

Figure 6: **Learned read/write gating avoids the larger coefficient peaks observed with unconstrained mixing.** Blank cells indicate undefined source–target pairs.

For unconstrained mixing with the fixed gates used in our comparisons, $h_j^{\text{pre}} = \mathbf{1}/n$ and $h_k^{\text{post}} = \mathbf{1}$, the same coefficient becomes

$$c_{j,k} = \frac{1}{n}\mathbf{1}^\top \Phi_{k+1\to j}\mathbf{1}. \tag{15}$$

Learned read/write gates jointly determine historical-output weights: the write gate controls how an output is allocated across streams, while a later read gate determines how that allocation contributes to its input. Unconstrained mixing instead modifies the allocation through intervening residual transitions, even with fixed read/write gates.

Under the mHC gate parameterization, read and write gates lie in $(0, 1)$ and $(0, 2)$, respectively, bounding their inner-product coefficients between 0 and $2n$. Unconstrained mixing permits signed coefficients without a comparable parameter-independent bound. It therefore offers additional weighting freedom, but also introduces the propagation risks discussed in the main text.

These mechanisms provide alternative ways to learn effective historical-output weights, offering a structural explanation for why either group can achieve competitive performance. Learning both groups combines these controls, consistent with the further gains in Table 2. Their comparable scores remain an empirical finding rather than a consequence of equivalent expressivity.

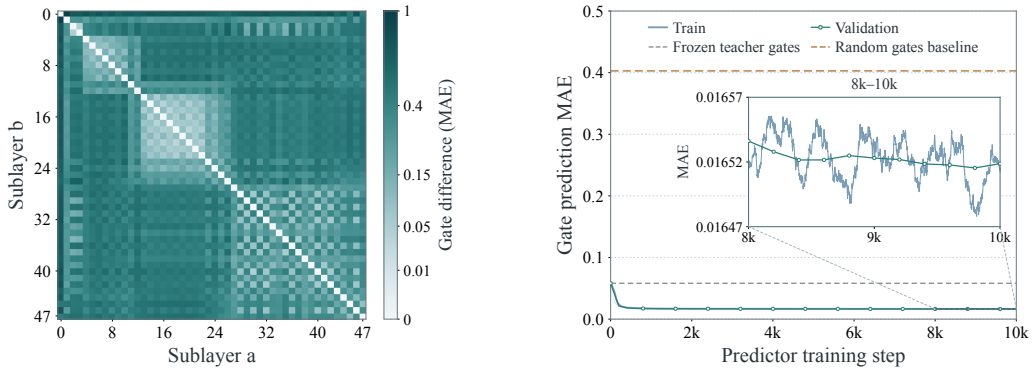
F Gate Analysis Details

This section details the measurements and experimental protocols for the analyses in Sections 6.1 and 6.2.

F.1 Residual-Path Contribution Coefficients

Figure 6 compares the final checkpoints of two 15B-A1B models: unconstrained mixing with fixed read/write gates and learned read/write gates with identity transitions. Both have 24 Transformer blocks, 48 sublayers, and four residual streams (Table 2, rows 12 and 6). We use 1,024 analysis sequences and sample 256 non-special-token positions per sequence with a fixed seed. Both models use the same 262,144 token occurrences. Sublayer indices alternate attention and FFN, from 0 to 47.

For each source a and later target b, we evaluate the coefficient $c_{b,a}$ in Equation 13 and plot $M_{b,a} = \max_{t\in\mathcal{T}} |c_{b,a}(t)|$. The mixing configuration fixes $h^{\text{pre}} = \mathbf{1}/4$ and $h^{\text{post}} = \mathbf{1}$; the identity configuration uses learned gates. These coefficients use raw write gates, without division by two. Each cell may attain its maximum at a different token. These sampled residual-path coefficients are neither matrix norms nor full-network Jacobians and do not alone establish training instability or increasing amplification with propagation distance.



(a) Read-gate variation across sublayers

(b) Prediction from the group-entry state

Figure 7: **Read gates vary across sublayers yet are predictable from shared inputs.** (a) Pairwise read-gate MAE across identityHC sublayers, averaged over streams and matched tokens. (b) Training and validation MAE for predicting read gates of the next three sublayers from the group-entry state, using the same experiment as Figure 4(b). Read gates are evaluated on their original $[0, 1]$ scale; the inset enlarges steps 8k–10k.

F.2 Gate Variation Across Sublayers

We use the same identityHC checkpoint and sampled token occurrences. For sublayers a, b, the plotted difference is

$$D^s(a,b) = \frac{1}{4|\mathcal{T}|} \sum_{t\in\mathcal{T}} \sum_{i=1}^{4} \left| q^s_{a,i}(t) - q^s_{b,i}(t) \right|, \qquad q^{\text{pre}} = h^{\text{pre}}, \quad q^{\text{post}} = h^{\text{post}}/2. \tag{16}$$

Absolute differences are computed for matched token positions and stream indices before averaging. Figure 4(a) shows write gates; Figure 7(a) shows read gates. Both use a power-normalized color scale with $\gamma = 0.4$ over $[0, 1]$; colorbar labels give actual MAE. The matrices are symmetric with zero diagonals and measure gate variation rather than causal propagation.

F.3 Prediction from the Group-Entry State

The prediction experiment freezes the final identityHC checkpoint and groups its 48 sublayers into 12 groups of $K = 4$. The teacher always executes with its own gates. Predictors receive only the same-token group-entry state X_L; their outputs never alter the teacher trajectory.

Predictors and targets. For $L \le j < L + K$ and $s \in \{\text{pre}, \text{post}\}$, the target is $h^s_j = g^s_j(X_j)$ from the frozen teacher. Each predictor uses the SimpleHC gate parameterization with independent parameters for every sublayer and gate type:

$$\begin{aligned} z_L &= \text{RMSNorm}(\text{vec}(X_L)), \\ \widehat{\ell}^s_j &= \widehat{\alpha}^s_j (\widehat{W}^s_j)^\top z_L + \widehat{b}^s_j, \\ \widehat{h}^{\text{pre}}_j &= \sigma(\widehat{\ell}^{\text{pre}}_j), \qquad \widehat{h}^{\text{post}}_j = 2\sigma(\widehat{\ell}^{\text{post}}_j). \end{aligned} \tag{17}$$

The input dimension is $4 \times 1792 = 7168$, with 32 gate values per group and approximately 2.75M trainable parameters in total. Projections, scales, and biases are initialized from the corresponding teacher parameters. Read and write gates are trained jointly; teacher states and targets are detached.

Training and evaluation. We train the predictor for 10,000 steps with MSE loss, a global batch size of 64, and a sequence length of 8,192. We use AdamW (Loshchilov & Hutter, 2017) with $\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$, zero weight decay, and gradient clipping at 1.0. The learning rate increases linearly from zero to 10^{-3} over 500 warmup steps, then follows cosine decay to 10^{-5}. Validation uses a fixed set of 2,048 sequences, excluding special tokens. We evaluate every 200 steps and use the final step-10,000 checkpoint for testing, without dev-based checkpoint selection.

Baselines and metrics. The frozen reference applies each sublayer mapping copied from the teacher to the shared entry state without adaptation. It remains input-dependent and does not reuse one gate

Table 8: **Model and training configuration for the stress test.**

Configuration	Value
Transformer layers	64
Hidden width	2816
Routed experts	512
Experts activated per token	8
Activated parameters	6B
Total parameters	230B
Peak learning rate	6.2×10^{-3}
Global batch size	256
Sequence length	8192
Warmup steps	4000

pair across sublayers. In the plots, this reference is labeled "Frozen teacher gates." The random baseline samples independent $h^{\text{pre}} \sim \mathcal{U}(0,1)$ and $h^{\text{post}} \sim \mathcal{U}(0,2)$ for each token, sublayer, and stream. The constant baseline uses mean gates fitted on training data. MAE averages valid tokens, groups, future offsets 1–3, and streams; write gates are divided by two. Offset 0 is excluded.

Experimental results. On 102,400 test sequences, the final predictor achieves read/write MAE of 0.01653/0.01034, compared with 0.05817/0.05044 for frozen mappings and 0.07345/0.06893 for constant mean gates. Together with the improvements over uniform random gates on the dev set, these results support closely approximating subsequent gates from shared entry states using layer-specific mappings.

G Training Stability Details

This section details the stress-test design, training dynamics, and instability diagnostics supporting Section 6.3.

G.1 Stress-Test Design and Settings

To emulate stability challenges in large-scale training, we combine 64 layers, 512 routed experts with top-8 routing, and a learning rate $10\times$ the reference value. This stresses optimization through a longer propagation path, higher sparsity, and larger updates. A hidden width of 2816 keeps activated parameters near 6B, yielding a 230B-A6B model. The experiment tests robustness under these combined pressures rather than isolating individual factors.

All variants use the backbone in Table 8. MLA alternates full and sliding-window attention in a 1:3 pattern, with 72 heads and window size 512. The first FFN is dense with width 7936; the remaining 63 use experts of width 832 and one shared expert. Training uses BF16, microbatch size 1, and Muon with momentum 0.95, weight decay 0.1, and gradient clipping at 1.0. The WSD schedule uses 4,000 linear warmup steps, followed by the peak learning rate throughout the comparison window. HC variants use $n = 4$; mHC uses 20 Sinkhorn iterations, and SimpleHC uses $K = 4$.

G.2 Training Dynamics

Standard training. During 15B training, SimpleHC shows smaller gradient-norm fluctuations early after warmup than mHC and identityHC, with comparable ranges later (Figure 5(a)). The maximum 1,000-step rolling standard deviation after warmup is 0.032 for SimpleHC, versus 0.206 for mHC and 0.248 for identityHC, indicating smoother standard training.

Stress testing. Under stress, mHC exhibits gradient spikes reaching approximately 1.9×10^6, whereas SimpleHC and identityHC remain bounded (Figure 5(b)). These spikes in mHC are accompanied by loss spikes (Figure 8) and growing branch-output scales, especially in the FFN. SimpleHC and identityHC show no comparable sustained growth (Figure 9), supporting stable training despite the structural and execution simplifications.

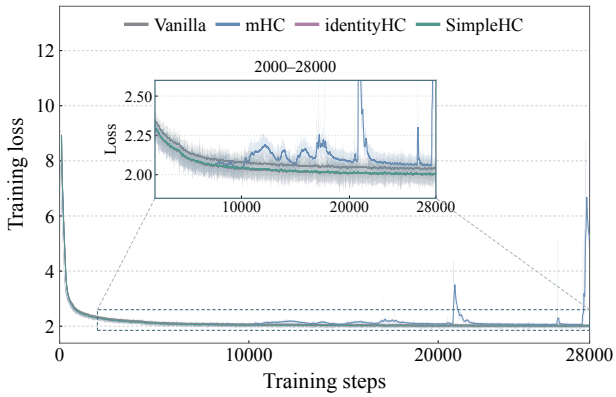
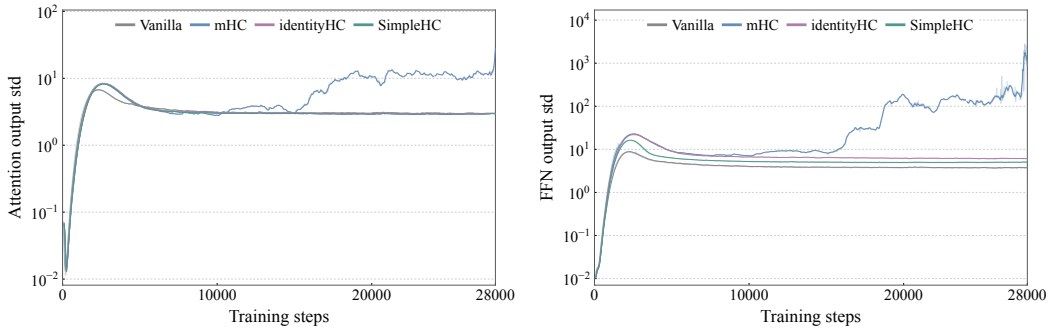


Figure 8: **Training loss under stress.** All runs cover the same 28,000 steps. We show per-step losses and their 100-step moving averages. The inset enlarges the low-loss region over steps 2,000–28,000.



(a) Attention output (b) FFN output

Figure 9: **Attention and FFN output standard deviations under stress.** Lines show 100-step moving averages; shaded bands show the minimum and maximum within each window. Both panels use logarithmic vertical axes and cover the same 28,000 steps.

G.3 Analysis of Stress-Test Results

SimpleHC and identityHC maintain stable optimization under the combined stress setting, indicating that removing residual mixing, with or without shared gate inputs, can preserve training stability. SimpleHC remains comparable to identityHC despite sharing gate inputs across sublayers. In contrast, mHC exhibits instability in both forward activations and backward gradients. To investigate this behavior, we inspect its branch outputs, residual mixing matrices, and gradient paths.

In a captured early-layer MoE event, an input RMS of 0.294 produced a branch-output RMS of 2.31×10^4 before write gating, locating the initial large values in the FFN rather than residual mixing at this sublayer. In the observed stress setting, 20 Sinkhorn iterations did not always suffice to produce a doubly stochastic matrix. In one captured case, row sums remained $(2, 1, 0.5, 0.5)$, increasing same-token RMS from 5.42×10^4 to 7.50×10^4. Mixing could therefore further amplify existing FFN outliers. These observations also suggest a possible rationale for the SwiGLU clamping used in DeepSeek-V4 (Xu et al., 2026): limiting FFN outliers before they enter the residual pathway. Residual mixing also introduces an additional gradient path because H^{res} depends on the input. For $Y = H^{\text{res}}(X)X$, the differential is $\mathrm{d}Y = H^{\text{res}}\mathrm{d}X + (\mathrm{d}H^{\text{res}})X$: the first term propagates through the fixed matrix, while the second captures changes in the mixing matrix itself. In one captured backward spike, the gate-input gradient norm was approximately $9.4\times$ that of the fixed-matrix path in the same call. Bounds computed from the recorded read/write gates and upstream gradients made their contributions negligible, implicating the dynamic mixing path. Whether this anomaly reflects mathematical sensitivity or numerical effects in backward remains unresolved. Identity transitions set $\mathrm{d}H^{\text{res}} = 0$, eliminating this additional path, consistent with the stable training observed in identityHC and SimpleHC.