Analyzing Stream Collapse in Hyper-Connections: From Diagnosis to Mitigation

Ekaterina Alimaskina [*1 2 3] **Gleb Molodtsov** [*1 2] **Aleksandr Beznosikov** [1 2 4]

Abstract

Hyper-Connections (HC) replace the single Transformer residual stream with multiple streams, introducing a permutation symmetry over stream indices. We study how this symmetry is resolved in practice: whether streams specialize in a balanced way or exhibit dominant-stream usage. Using fine-grained diagnostics for HC-based language models, we trace how multi-stream representations are actually used. We find that after an early seeding stage, residual mixing often remains close to identity, limiting a core HC mechanism for exchanging information between streams. Moreover, both signal and interpretable features concentrate in a dominant stream, and the nominally multi-stream residual connection can underutilize its capacity, behaving closer to a single-stream residual pathway. Finally, we show that breaking symmetry at stream initialization reduces dominant behavior and improves performance across *m*HC variants. Our code is publicly available[†].

1. Introduction

Weight-space symmetries arise when different parameter configurations represent the same network function. While often viewed as harmless redundancies, they can also shape optimization by making some solutions easier to reach than others. Prior work has often focused on symmetries inside Transformer blocks, such as those involving attention heads or feed-forward layers. In contrast, the symmetries introduced by residual pathways remain underexplored.

Each Transformer block applies attention and feed-forward transforms to a shared residual stream, producing an update:

$$\mathbf{x}_{\ell+1} = \mathbf{x}_\ell + \mathcal{F}_\ell(\mathbf{x}_\ell; \mathbf{W}_\ell), \tag{1}$$

where $\mathbf{x}_\ell$ denotes the residual representation at layer ℓ, and $\mathcal{F}_\ell$ denotes the layer transformation.

Hyper-Connections (HC) replace one residual stream with n parallel ones and learn token-dependent connectivity between them (Zhu et al., 2024). For one token, let $\mathbf{X}_\ell \in \mathbb{R}^{n\times d}$ denote the stream state at layer ℓ, with rows corresponding to streams. An HC layer is

$$\mathbf{X}_{\ell+1} = \mathbf{H}_\ell^{\text{res}}\mathbf{X}_\ell + \left(\mathbf{H}_\ell^{\text{post}}\right)^\top \mathcal{F}_\ell(\mathbf{H}_\ell^{\text{pre}}\mathbf{X}_\ell; \mathbf{W}_\ell)\,. \tag{2}$$

Here the connectivity operators are token-dependent: $\mathbf{H}_\ell^{\text{res}} \in \mathbb{R}^{n\times n}$ mixes the carried stream state in the residual branch, while $\mathbf{H}_\ell^{\text{pre}}, \mathbf{H}_\ell^{\text{post}} \in \mathbb{R}^{1\times n}$ form the read/write interface. The former reads from streams into the block, and the latter writes the block update back. When $n = 1$, Eq. (2) reduces to Eq. (1).

By construction, HC first expands the residual representation into n streams. In the standard setup, this is done by replication, $\mathbf{x} \mapsto [\mathbf{x}; \ldots; \mathbf{x}]$, so all streams start as identical copies. Together with Eq. (2), this creates a permutation symmetry over stream indices: relabeling the streams and corresponding connectivity operators preserves the represented computation. This raises a practical question:

If streams start identical and exchangeable, what makes them acquire different roles during training, and do they actually specialize in practice?

Prior HC variants primarily focus on architecture design, stability, and aggregate language-modeling performance, leaving the internal use of individual streams less characterized. In particular, they address the instability of unconstrained residual mixing: unrolling Eq. (2) replaces the fixed identity path by products of $\mathbf{H}_\ell^{\text{res}}$ matrices, which may cause residual norms to grow or decay rapidly. Manifold-Constrained Hyper-Connections (*m*HC) therefore constrain $\mathbf{H}_\ell^{\text{res}}$ to be approximately doubly stochastic (Xie et al., 2025), i.e., to lie in the Birkhoff polytope

$$\mathcal{B}_n = \left\{\mathbf{H} \in \mathbb{R}^{n\times n} \,\middle|\, \mathbf{H}\mathbf{1} = \mathbf{1}, \mathbf{1}^\top\mathbf{H} = \mathbf{1}^\top, \mathbf{H} \geq 0\right\}. \tag{3}$$

This can be interpreted as mixing streams by convex combinations of permutations (Birkhoff, 1946). Efficient variants

[1]MIRAI, Moscow, Russia [2]BRAIn Lab, Moscow, Russia [3]Yandex Research, Moscow, Russia [4]Innopolis University, Innopolis, Russia. Correspondence to: Ekaterina Alimaskina <alimaskina.ea@yandex.ru>.

Workshop on Weight-Space Symmetries, held in conjunction with the 43rd International Conference on Machine Learning, Seoul, South Korea. 2026. Copyright 2026 by the author(s).

[†]https://github.com/brain-lab-research/hc-stream-collapse

such as *m*HC-lite and KromHC provide cheaper constrained parameterizations (Yang & Gao, 2026; Zhou et al., 2026).

These constraints address the stability of residual mixing, but they do not show whether the learned multi-stream mechanism is used as intended. In particular, the read/write interface remains a separate degree of freedom, and the standard symmetric expansion provides no controlled way to assign stream roles. This creates a possible failure mode: a small early advantage for one stream may be amplified across depth, as that stream grows in norm, contributes more to later reads, and receives more subsequent updates. Rather than producing balanced specialization, training may therefore favor a dominant-stream regime.

We find this collapse pattern across our main HC experiments. Read/write signal, representation norms, and semantic probes identify one dominant stream, while residual mixing often stays close to identity. To test whether this failure mode can be mitigated by controlled symmetry breaking, we study Learned Stream Scaling (LSS), a minimal modification at the expansion interface. LSS replaces exact stream replication with learnable near-identity diagonal scales, adding only nd parameters while leaving the core HC operator unchanged. This helps the model to drift away from the dominant-stream regime and improves perplexity.

Contributions. Our contributions are:

- We identify a stream-level failure mode in HC-style residuals: models with multiple symmetric streams can rely on one dominant stream.
- We show that collapse arises in mechanics and semantics: residual mixing stays near identity, while read/write signal and representation content concentrate in one stream.
- Using Learned Stream Scaling, we show that a small controlled symmetry break reduces collapse and improves *m*HC variants without changing the core HC operator.

2. Stream collapse under symmetric HC initialization

2.1. Experimental protocol

We analyze *m*HC-lite nanoGPT models (Karpathy, 2022) with $n = 4$ streams, following prior HC work (Xie et al., 2025; Yang & Gao, 2026). See Appendix B for other stream counts. Main figures use the medium model trained on OpenWebText. We evaluate perplexity on OpenWebText, WikiText-103, and C4 (Gokaslan & Cohen, 2019; Merity et al., 2016; Raffel et al., 2020). See details in Appendix D.

2.2. Residual mixing is mostly near-identity

Prior HC variants often attribute their gains to the residual mixing operator $\mathbf{H}_\ell^{\text{res}}$: ablations indicate that the largest improvements come from allowing width-wise exchange between streams (Xie et al., 2025). If residual mixing were the main driver of HC gains in the trained model, we would expect substantial cross-stream exchange beyond the earliest layers. We find that this is mostly not the case.

As shown in Fig. 1, the early mixer has substantial off-diagonal mass, while a deeper mixer is close to identity, suggesting that residual mixing becomes a near-bypass after initial seeding; full depth-wise visualizations are provided in Appendix F.

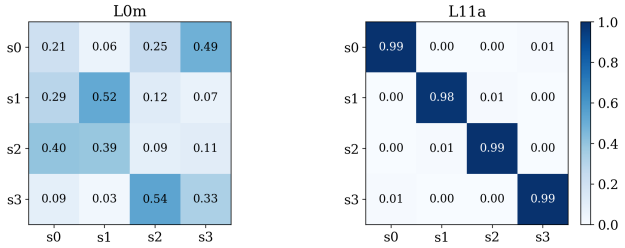

Figure 1. Token-averaged residual-mixing matrices $\mathbf{H}_\ell^{\text{res}}$ in trained *m*HC-lite. Labels $s0$–$s3$ denote streams. The early mixer (`L0m`, left) has substantial off-diagonal mass, while a deeper mixer (`L11a`, right) is close to identity.

A possible interpretation is that early updates differentiate the initially identical streams, so residual mixing is most useful at this stage. Once read/write usage concentrates on one stream, later cross-stream mixing may become less beneficial, consistent with the observed shift from early mixing to near-identity $\mathbf{H}_\ell^{\text{res}}$ matrices.

To test whether sustained residual mixing is critical under the default regime, we freeze most $\mathbf{H}_\ell^{\text{res}}$ matrices to identity and leave only every sixth layer trainable.[1] If cross-stream exchange were a key performance driver, this intervention should substantially degrade perplexity. Instead, performance is mostly preserved, and even improves for the medium model (Table 1). This suggests that baseline models often operate near a bypass regime, where much of the residual mixing mechanism is underutilized.

Table 1. *m*HC-lite vs IDENTITY residual mapping. Perplexity (↓) on validation/test sets.

Size	Model	OWT val	WT val	WT test	C4 test
M	*m*HC-lite	26.22	52.74	51.05	57.32
M	Identity $\mathbf{H}_\ell^{\text{res}}$	**25.93**	**51.86**	**49.65**	**53.94**
L	*m*HC-lite	**24.51**	**46.03**	**43.98**	**50.95**
L	Identity $\mathbf{H}_\ell^{\text{res}}$	24.54	46.30	44.32	55.15

Takeaway. After an early seeding stage, residual mixing often stays close to identity. Stream indices therefore remain approximately aligned across layers, making it meaningful to track per-stream read/write usage, norm growth, and semantic content below.

[1]The strongest mixing appears early, but we keep every sixth mixer trainable to allow limited later exchange.

2.3. Read/write routing favors one stream

Beyond residual mixing, hyper-connections introduce a read-compute-write interface. If multiple streams are used as intended, then several streams should contribute to the block input and receive meaningful updates across depth.

Figure 2 reports per-layer average stream contributions to the read vector $\mathbf{H}_\ell^{\text{pre}}$ and the write vector $\mathbf{H}_\ell^{\text{post}}$. A single stream quickly becomes dominant: it provides most of the block input and receives most of the block update. Thus, the block is repeatedly read from and written back to one persistent stream, while the remaining streams receive substantially weaker update signal.

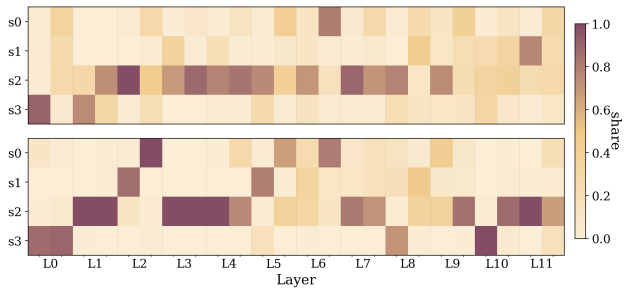

Figure 2. Token-averaged, normalized stream contributions in the read/write interface ($n = 4$ streams). **Top:** average contribution of each stream to the block input (read). **Bottom:** average fraction of the block update written back to each stream (write). Stream s_2 dominates both.

Thus, collapse is visible not only in residual mixing but also at the block interface: input and update signals are repeatedly routed through the same stream. We next show that this read/write imbalance is accompanied by a depth-wise norm imbalance.

2.4. Depth-wise accumulation amplifies the dominant stream

The read/write imbalance above suggests a simple mechanism for signal accumulation. In HC-style blocks, updates are added to streams across depth. If the write operator repeatedly favors one stream, that stream receives larger effective increments and its norm grows faster. A larger-norm stream can then contribute more to later block inputs through both learned read weights and activation scale, reinforcing its advantage over depth. This creates a self-reinforcing loop that amplifies an initially uneven read/write pattern into a dominant-stream regime.

Figure 3 shows the corresponding representation-level imbalance for the *m*HC-lite baseline. In the right panel, one stream accumulates markedly higher representational L_2 norm in deeper layers, while the other streams carry much lower L_2 norm. This provides a representation-level signature of the same collapse as at the read/write interface.

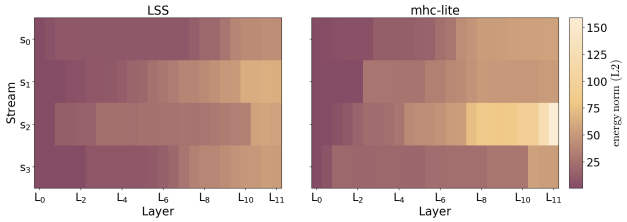

Figure 3. Per-stream representation L_2 norm, averaged across validation tokens. The *m*HC-lite baseline shows a strong norm imbalance, while the LSS variant, introduced in Sec. 3, reduces this imbalance.

2.5. Dominance extends to semantic representation structure

We now ask whether the same stream also carries more stable and interpretable structure. We use two complementary semantic probes: residual curvature and sparse crosscoders. These tools test token-level stream dominance via smoother residual trajectories and through crosscoder features that are preferentially expressed in a particular stream.

Probe 1: residual curvature. Residual curvature measures how smoothly representations evolve along the token sequence. Following the curved-inference view of language models (Hosseini & Fedorenko, 2023), we compute the average turning angle between consecutive token-to-token displacement vectors in hidden-state space. Lower curvature indicates a smoother and more stable representation trajectory; prior work links such measures to representation quality and downstream behavior (Skean et al., 2025). We use this probe to test whether the read/write-dominant stream is also a geometrically more stable stream. Figure 4 shows the same trend: the dominant stream has lower curvature over many layers, suggesting that it is not only larger in norm or preferred by the interface, but also carries a more stable representational signal.

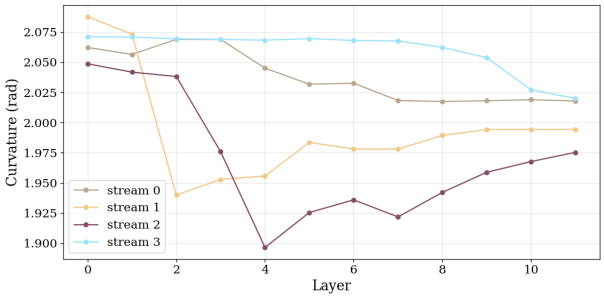

Figure 4. Token-trajectory curvature per stream across layers. Lower curvature indicates smoother token-level representation geometry. The stream that dominates the read/write interface also exhibits the lowest curvature over many layers.

Probe 2: sparse crosscoders. As a second probe, we train sparse crosscoders on hidden states from all four streams at even layers $\{0, 2, 4, 6, 8, 10\}$ (Lindsey et al., 2024). Sparse

Table 2. Comparison of *m*HC and *m*HC-lite variants. Perplexity (↓). IDENTITY replaces all residual mixing matrices by identity except every sixth layer.

	***m*HC**								***m*HC-lite**							
	OWT val		WT val		WT test		C4 test		OWT val		WT val		WT test		C4 test	
Method	M	L	M	L	M	L	M	L	M	L	M	L	M	L	M	L
Baseline	26.44	24.61	53.29	46.35	50.97	44.38	58.62	55.39	26.22	24.51	52.74	46.03	51.05	43.98	57.32	**50.95**
Identity	26.72	24.45	53.15	46.07	50.88	43.95	57.72	51.38	25.93	24.54	51.86	46.30	49.65	44.32	**53.94**	55.15
LSS + Identity	**25.74**	24.35	51.54	45.87	**48.92**	43.80	55.16	53.37	25.88	24.36	51.96	44.70	49.60	42.69	57.04	52.18
LSS	26.08	**22.78**	**51.43**	**45.73**	48.99	**43.66**	**54.91**	**49.14**	**25.65**	**24.29**	**49.41**	**44.64**	**47.92**	**42.62**	54.20	51.23

dictionary methods decompose representations into sparse, often interpretable feature directions; crosscoders learn one shared sparse code for all stream slices at a layer, making them suitable for localizing such features across streams. We assign each recovered feature to the stream with the largest decoder norm; details are given in Appendix A.

Figure 5 shows that recovered feature mass is highly concentrated. Early layers are dominated by s_3, but from layer 4 onward most feature mass concentrates in s_2, matching the stream identified by the read/write and norm probes. This provides feature-level evidence that the dominant stream also captures much of the recovered interpretable structure.

Takeaway. Mechanical and semantic probes consistently identify the same dominant pathway, suggesting that residual capacity is not evenly distributed across streams.

3. Controlled initial symmetry breaking with Learned Stream Scaling

The collapse above suggests a simple test: if exact stream replication contributes to uncontrolled role assignment, then a small perturbation of the replicated initialization may move the model away from the dominant-stream regime. Learned Stream Scaling (LSS) applies this break only at the expansion interface, leaving the HC operator unchanged.

3.1. LSS breaks initial exchangeability

The HC update is equivariant to stream relabeling. For a fixed token, suppressing token dependence, let $\mathbf{P}$ be a permutation matrix acting on the stream axis and define

$$\begin{aligned} \mathbf{X}'_\ell &= \mathbf{P}\mathbf{X}_\ell, & \mathbf{H}'^{\text{res}}_\ell &= \mathbf{P}\mathbf{H}^{\text{res}}_\ell\mathbf{P}^\top, \\ \mathbf{H}'^{\text{pre}}_\ell &= \mathbf{H}^{\text{pre}}_\ell\mathbf{P}^\top, & \mathbf{H}'^{\text{post}}_\ell &= \mathbf{H}^{\text{post}}_\ell\mathbf{P}^\top. \end{aligned} \tag{4}$$

Substituting these into Eq. (2) gives $\mathbf{X}'_{\ell+1} = \mathbf{P}\mathbf{X}_{\ell+1}$, so stream labels have no intrinsic roles: relabeling streams together with their connectivity operators preserves the represented computation.

This symmetry becomes a practical issue because the standard HC expansion also gives streams no initial differences: it simply replicates the same representation, $\mathbf{E}(\mathbf{x}) = [\mathbf{x}; \ldots; \mathbf{x}]$. Since $\mathbf{P}\mathbf{E}(\mathbf{x}) = \mathbf{E}(\mathbf{x})$ for any permutation $\mathbf{P}$, all streams start in the fixed subspace of the symmetry. Thus, role separation must emerge from training dynamics rather than from a controlled signal.

LSS targets this point by replacing exact replication with near-identity per-stream diagonal scaling:

$$\mathbf{x}^{(s)} = \mathbf{x} \odot \mathbf{h}_s, \qquad \mathbf{h}_s \in \mathbb{R}^d, \qquad s = 1, \ldots, n. \tag{5}$$

The scales are initialized near uniform, so streams remain close but are no longer identical. This gives streams distinct initial signals while leaving the HC update unchanged, adding only nd parameters and no extra matmuls.

3.2. LSS changes the stream-usage regime

We test whether LSS moves the model away from the near-bypass regime. First, we repeat the identity-mixing ablation from Sec. 2.2, fixing all residual mixing matrices to identity except every sixth layer. The baseline is almost insensitive to this ablation, consistent with near-bypass behavior. With LSS, the same constraint tends to be more costly, especially in larger models, suggesting that cross-stream mixing becomes more relevant under the LSS regime.

Second, we directly examine the collapse signature. Figure 3 shows that LSS substantially reduces the strong norm imbalance observed in the baseline: the representation norm is less concentrated in a single stream. Thus, LSS appears to change the stream-usage regime, rather than only improving perplexity within the same collapsed solution. Additional scaling results across token budgets are provided in App. C.

4. Conclusion

We show that HC can fail to realize their intended multi-stream behavior. After early mixing, trained models concentrate most signal, representation norm, and interpretable features in a single dominant stream. This suggests that stream-permutation symmetry may bias optimization toward collapsed residual pathways. A minimal symmetry break, Learned Stream Scaling, reduces this collapse and improves perplexity across *m*HC variants.

Acknowledgments

The work was supported by the Ministry of Economic Development of the Russian Federation (agreement No. 139-15-2025-013, dated June 20, 2025, IGK 000000C313925P4B0002).

Impact Statement

This work takes a step toward understanding the limitations of current multi-stream residual architectures: by exposing their failure modes, we highlight the need for further work on robust strategies for symmetry breaking, stream specialization, and sustained cross-stream information exchange.

References

Birkhoff, G. Tres observaciones sobre el algebra lineal. *Universidad Nacional de Tucuman. Revista, Serie A*, 5: 147–151, 1946.

Gokaslan, A. and Cohen, V. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

Hosseini, E. and Fedorenko, E. Large language models implicitly learn to straighten neural sentence trajectories to construct a predictive representation of natural language. *Advances in Neural Information Processing Systems*, 36: 43918–43930, 2023.

Karpathy, A. nanoGPT. https://github.com/karpathy/nanoGPT, 2022.

Lindsey, J., Templeton, A., Marcus, J., Conerly, T., Batson, J., and Olah, C. Sparse crosscoders for cross-layer features and model diffing. *Transformer Circuits Thread*, pp. 3982–3992, 2024.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21 (140):1–67, 2020.

Skean, O., Arefin, M. R., Zhao, D., Patel, N. N., Naghiyev, J., LeCun, Y., and Shwartz-Ziv, R. Layer by layer: Uncovering hidden representations in language models. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=WGXb7UdvTX.

Xie, Z., Wei, Y., Cao, H., Zhao, C., Deng, C., Li, J., Dai, D., Gao, H., Chang, J., Yu, K., et al. mHC: Manifold-constrained hyper-connections. *arXiv preprint arXiv:2512.24880*, 2025.

Yang, Y. and Gao, J. mHC-lite: You don't need 20 sinkhorn-knopp iterations. *arXiv preprint arXiv:2601.05732*, 2026.

Zhou, W., Gu, Y., Iacovides, G., and Mandic, D. KromHC: Manifold-constrained hyper-connections with kronecker-product residual matrices. *arXiv preprint arXiv:2601.21579*, 2026.

Zhu, D., Huang, H., Huang, Z., Zeng, Y., Mao, Y., Wu, B., Min, Q., and Zhou, X. Hyper-connections. *arXiv preprint arXiv:2409.19606*, 2024.

Supplementary Material

A. Sparse crosscoder details

Sparse crosscoders are designed to compare representations across multiple sources using a shared sparse dictionary (Lindsey et al., 2024). This makes them a natural tool for our setting: instead of training separate sparse autoencoders for each stream, we train a crosscoder that reconstructs all stream slices at a given layer from the same sparse activation vector. Thus, a recovered feature can be compared across streams through its stream-specific decoder vectors.

For each selected even layer $\ell \in \{0, 2, 4, 6, 8, 10\}$, we collect hidden states

$$\mathbf{x}_t^{(\ell,s)} \in \mathbb{R}^d, \qquad s \in \{0, 1, 2, 3\},$$

where t indexes tokens and s indexes streams. The crosscoder learns non-negative sparse activations $\mathbf{a}_t^{(\ell)} \in \mathbb{R}^m_{\geq 0}$ and stream-specific decoders $\mathbf{D}^{(\ell,s)} \in \mathbb{R}^{d\times m}$, reconstructing each stream slice as

$$\widehat{\mathbf{x}}_t^{(\ell,s)} = \mathbf{D}^{(\ell,s)}\mathbf{a}_t^{(\ell)}.$$

The same sparse code $\mathbf{a}_t^{(\ell)}$ must reconstruct all streams at that layer, so the learned features are aligned across streams rather than fitted independently.

To localize feature i, we use the norm of its decoder column in each stream:

$$w_i^{(\ell,s)} = \left\| \mathbf{D}_{:,i}^{(\ell,s)} \right\|_2 .$$

Following standard crosscoder attribution, feature i is assigned to the stream with the largest decoder norm,

$$s_i^{(\ell)} = \arg\max_s w_i^{(\ell,s)}.$$

We then aggregate these assignments to obtain the feature share of each stream at layer ℓ.

In Fig. 5, we report the stream-level feature share across selected even layers. The earliest layers are dominated by s_3, but from layer 4 onward the recovered features concentrate in s_2. This agrees with the read/write and norm probes in the main text, indicating that the dominant stream also carries most localized sparse features after the initial transition.

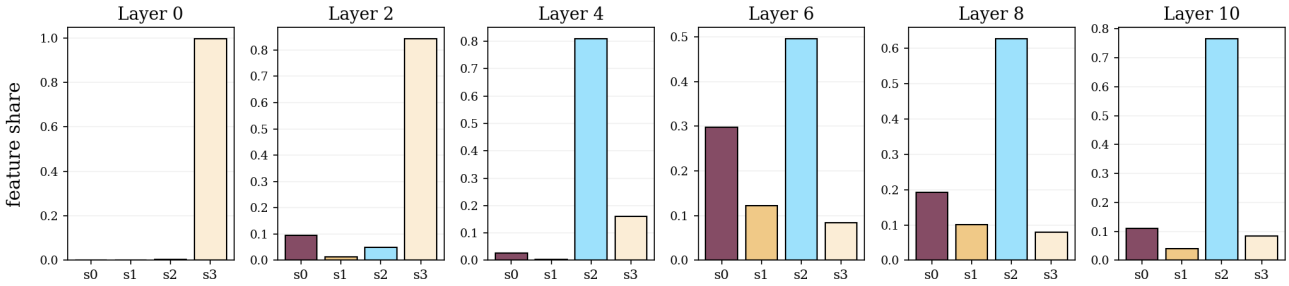

Figure 5. Sparse crosscoder trained on stream states from even layers. Recovered sparse-feature assignments concentrate in one stream at a time, providing interpretable evidence of dominant-stream behavior under HC-style residuals.

B. Stream count analysis

Prior HC work commonly uses $n = 4$ streams, but stream collapse may depend on the number of available streams. We therefore repeat our diagnostics for $n \in \{2, 4, 8, 16\}$. For this sweep, we use *m*HC rather than *m*HC-lite, since the *m*HC-lite permutation-based parameterization scales factorially with the number of streams.

Figure 6 shows read share, write share, and per-stream representation L_2 norm across layers. The imbalance is visible for all stream counts. It is strongest for $n = 2$, where one stream dominates both routing and norm growth. For larger n, especially $n = 16$, the signal is distributed across more streams, but the allocation remains far from uniform.

Figure 7 shows representative residual-mixing matrices for the first, middle, and last blocks. Across stream counts, the learned mixers become mostly near-diagonal after the earliest layers. Overall, larger n softens but does not eliminate the imbalance: learned stream usage remains nonuniform.

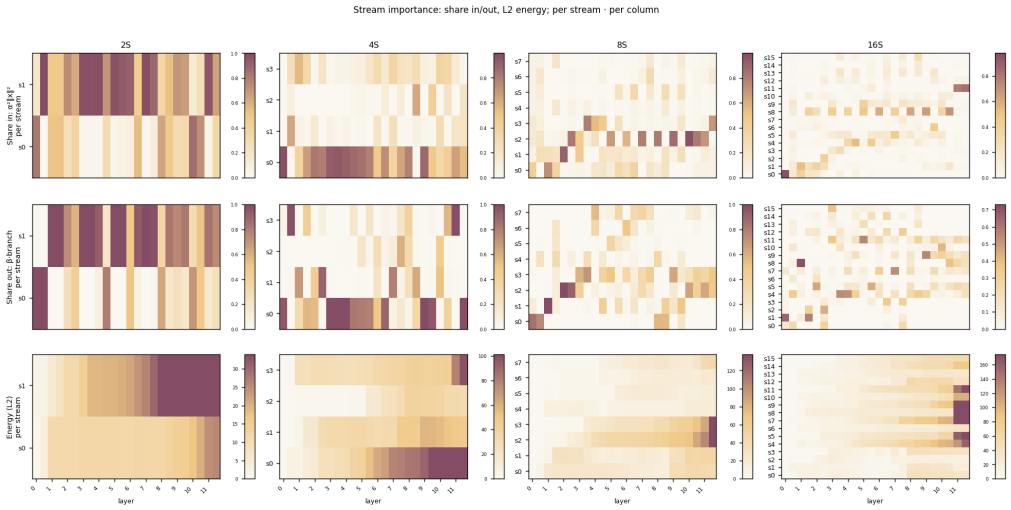

Figure 6. Stream-usage diagnostics for *m*HC models with $n \in \{2, 4, 8, 16\}$ streams. Rows show normalized read share, normalized write share, and per-stream representation L_2 norm across layers. Larger n spreads the signal across more streams, but the allocation remains nonuniform.

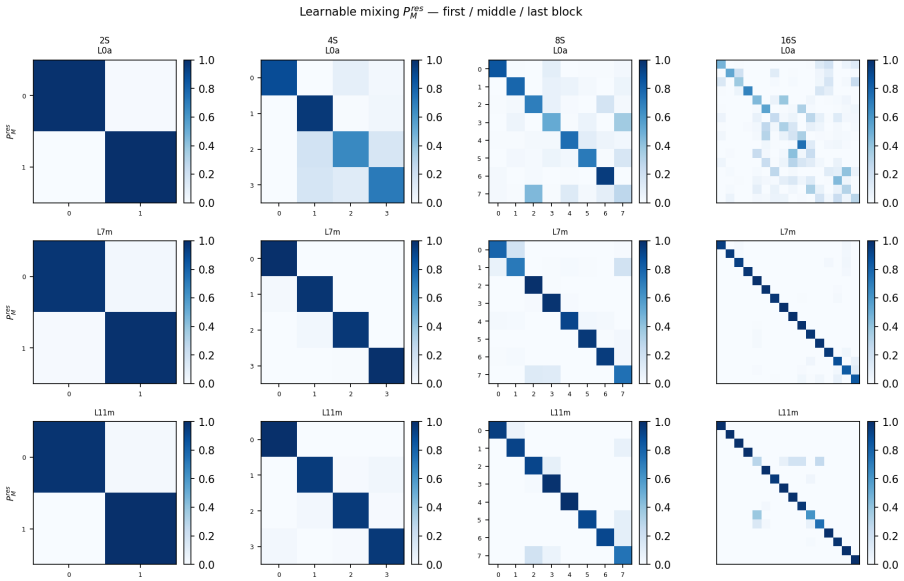

Figure 7. Representative residual-mixing matrices for *m*HC models with $n \in \{2, 4, 8, 16\}$ streams. Residual mixing is strongest in the earliest block and becomes mostly near-diagonal in later layers across stream counts.

C. Scaling across token budgets

Table 3 compares *m*HC-lite and LSS across token-to-parameter regimes corresponding to Chinchilla ratios $0.5/1/2$. LSS improves perplexity consistently across OpenWebText, WikiText-103, and C4, and the gains remain visible as the token budget increases.

Table 3. Perplexity (↓) for *m*HC-lite baseline and LSS across token-to-parameter (T/P) rates.

T/P	Model	OWT val	WT val	WT test	C4 test
10	Base	26.22	52.74	51.05	57.32
10	LSS	**25.65**	**49.41**	**47.92**	**54.20**
20	Base	24.16	48.59	46.32	52.09
20	LSS	**23.63**	**45.66**	**43.57**	**50.39**
40	Base	22.16	43.99	42.07	48.42
40	LSS	**21.76**	**41.31**	**39.63**	**48.06**

D. Experimental details

Unless stated otherwise, ablations use the nanoGPT **M** model (12 layers, ∼0.12B parameters) trained on OpenWebText for ≈ 1.3B tokens (Karpathy, 2022; Gokaslan & Cohen, 2019). We also evaluate the nanoGPT **L** model (24 layers, ∼0.36B parameters). Following prior HC work, we set the number of streams to $n = 4$ (Xie et al., 2025; Yang & Gao, 2026). We report perplexity on OpenWebText validation, WikiText-103 validation/test, and C4 test (Merity et al., 2016; Raffel et al., 2020).

All experiments were run on NVIDIA H200 GPUs. As a reference point, pre-training nanoGPT-M on 1.3B tokens takes approximately 6 GPU-hours, while nanoGPT-L takes approximately 13 GPU-hours. Seed-stability results are reported in Appendix E.

D.1. Hyperparameters

Hyperparameter	**Medium**	**Large**
`n_layer`	12	24
`n_head`	12	16
`n_embd`	768	1024
`dropout`	0.0	0.0
`learning_rate`	6e-4	3e-4
`min_lr`	6e-5	3e-5
`warmup_iters`	2%	2%
`weight_decay`	0.1	0.1
`beta1`	0.9	0.9
`beta2`	0.95	0.95
`grad_clip`	1.0	1.0

Table 4. Training hyperparameters for nanoGPT Medium and Large runs.

D.2. Notation

Symbol	Type / shape	Meaning
ℓ	index	Layer index.
L	scalar	Number of layers.
t	index	Token index.
s	index	Stream index, typically $s \in \{1, \ldots, n\}$.
n	scalar	Number of parallel streams.
d	scalar	Hidden dimension of each stream.
$\mathbf{x}_\ell$	$\in \mathbb{R}^d$	Single-stream token representation at layer ℓ; token index is often omitted.
$\mathbf{X}_\ell$	$\in \mathbb{R}^{n \times d}$	Multi-stream token state at layer ℓ; rows correspond to streams.
$\mathbf{x}^{(s)}$	$\in \mathbb{R}^d$	Representation in stream s, i.e., a row/slice of $\mathbf{X}$.
$\mathbf{x}_t^{(\ell,s)}$	$\in \mathbb{R}^d$	Token t representation at layer ℓ in stream s, used in cross-stream analysis.
$\mathcal{F}_\ell(\cdot; \mathbf{W}_\ell)$	function	Transformer block or residual block at layer ℓ with parameters $\mathbf{W}_\ell$.
$\mathbf{W}_\ell$	parameters	Learnable parameters of block $\mathcal{F}_\ell$.
$\mathbf{H}_\ell^{\text{res}}$	$\in \mathbb{R}^{n \times n}$	Residual mixing operator: mixes the carried stream state in the residual branch.
$\mathbf{H}_\ell^{\text{pre}}$	$\in \mathbb{R}^{1 \times n}$	Read operator: forms the block input by reading from streams.
$\mathbf{H}_\ell^{\text{post}}$	$\in \mathbb{R}^{1 \times n}$	Write operator: distributes the block output back into streams.
$\mathcal{B}_n$	set	Birkhoff polytope: set of doubly stochastic $n \times n$ matrices.
$\mathbf{1}$	vector	All-ones vector used in doubly stochastic constraints.
$\mathbf{P}$	$\in \mathbb{R}^{n \times n}$	Permutation matrix acting on the stream axis.
$\mathbf{E}(\mathbf{x})$	$\in \mathbb{R}^{n \times d}$	Standard HC expansion by replication, $\mathbf{E}(\mathbf{x}) = [\mathbf{x}; \ldots; \mathbf{x}]$.
$\mathbf{h}_s$	$\in \mathbb{R}^d$	Per-stream diagonal scaling vector used in Learned Stream Scaling.
$\odot$	operator	Elementwise product.
$\mathbf{a}_t^{(\ell)}$	$\in \mathbb{R}^m$	Sparse activation/code for token t in the crosscoder at layer ℓ.
m	scalar	Number of sparse features in the crosscoder dictionary.
$\mathbf{D}^{(\ell,s)}$	$\in \mathbb{R}^{d \times m}$	Decoder matrix for layer ℓ, stream s.
$\hat{\mathbf{x}}_t^{(\ell,s)}$	$\in \mathbb{R}^d$	Crosscoder reconstruction, $\hat{\mathbf{x}}_t^{(\ell,s)} = \mathbf{D}^{(\ell,s)} \mathbf{a}_t^{(\ell)}$.
i	index	Sparse feature index.
$w_i^{(\ell,s)}$	scalar	Decoder-column norm, $w_i^{(\ell,s)} = \lVert \mathbf{D}_{:,i}^{(\ell,s)} \rVert_2$.
$s_i^{(\ell)}$	index	Stream assigned to feature i at layer ℓ, $s_i^{(\ell)} = \arg\max_s w_i^{(\ell,s)}$.

Table 5. Notation used throughout the paper.

E. Training Stability Across Random Seeds

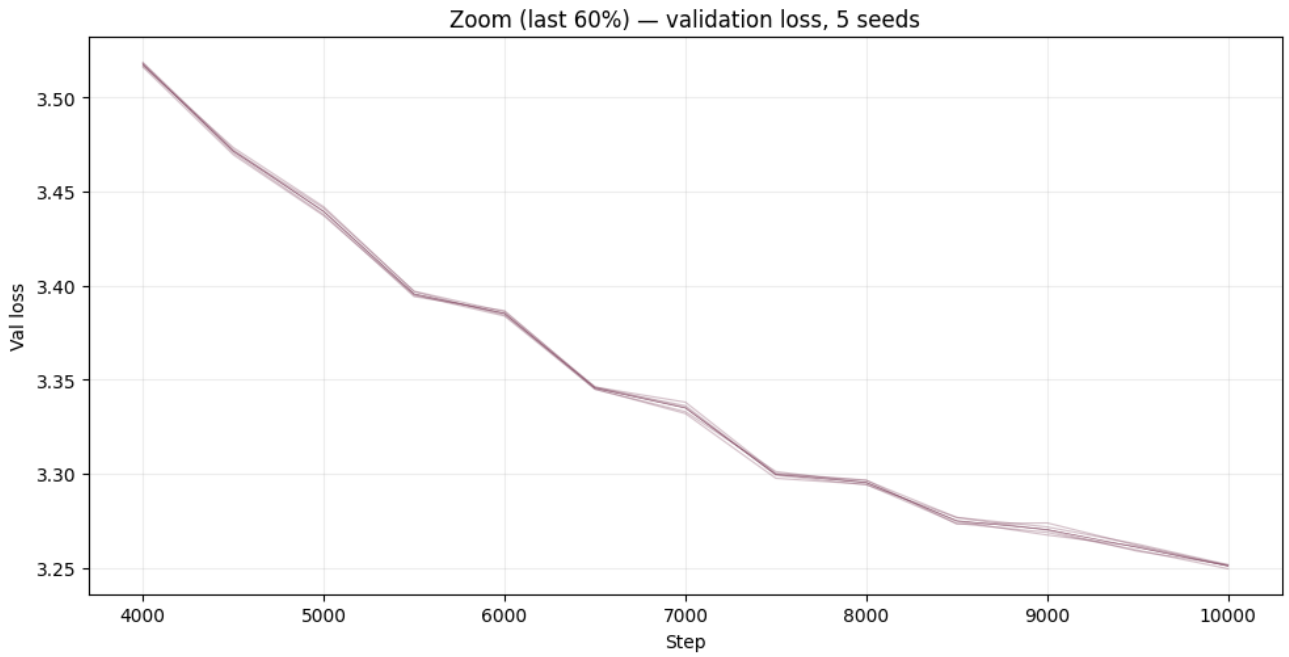

Figure 8. Validation loss across 5 random seeds (zoomed to the last 60% of training), showing tightly clustered trajectories and low seed sensitivity.

We evaluated training stability of medium *m*HC-lite model by repeating the same setup with 5 different random seeds and comparing the resulting training dynamics. Figure 8 shows the validation-loss trajectories over the later part of training (zoomed to emphasize small differences).

Across seeds, the curves are tightly clustered throughout training: the end-of-training validation loss varies within a narrow range (final min–max range $\approx 3.2 \times 10^{-3}$; final std $\approx 8.8 \times 10^{-4}$), and typical pairwise relative deviations stay below $\sim 0.15\%$.

Overall, we observe low sensitivity to random initialization and no divergent runs, suggesting that the qualitative conclusions of our experiments are robust and not driven by seed-specific effects.

F. Cross-stream mixing across token budgets

In this appendix we visualize the learned residual-mixing operators across depth for two variants: the standard *m*HC-lite baseline and the LSS variant. Each page corresponds to a different token budget (1.3B / 2.6B / 5.2B), and shows the per-layer mixing heatmaps for the two models. See Figures 9 - 11 for details.

Overall, strong cross-stream mixing emerges very slowly: for most layers the learned mixing remains close to an identity-like pattern, indicating that sustained exchange between streams is limited and builds up only gradually with training budget. At the same time, compared to the baseline, our scheme starts to exhibit noticeable off-diagonal mass earlier, i.e., the onset of stream mixing appears sooner. Nevertheless, even with larger budgets, the effect remains modest and the dominant regime is still characterized by weak/slowly developing mixing.

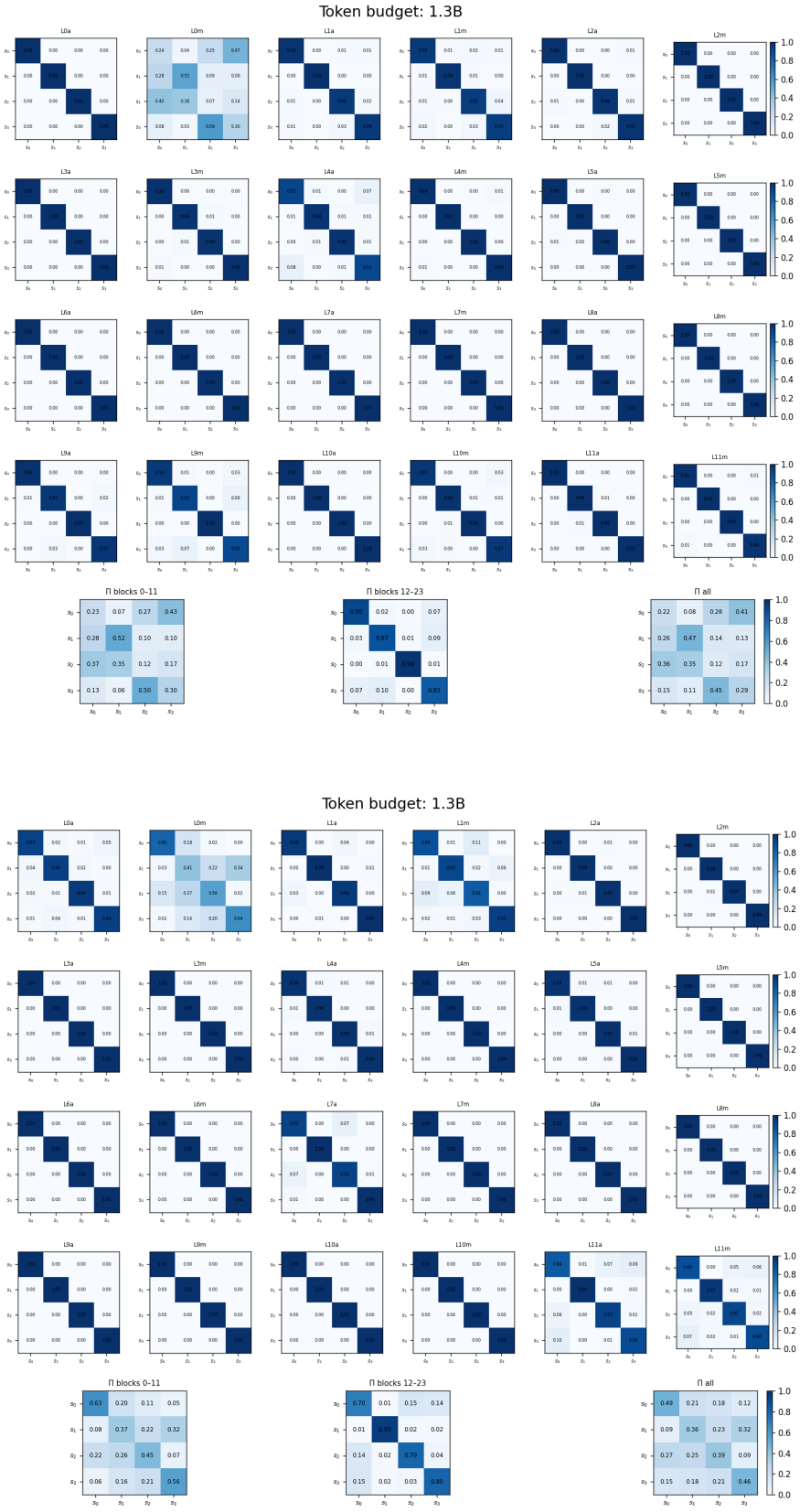

Figure 9. Token budget 1.3B: *m*HC-lite (top), LSS (bottom).

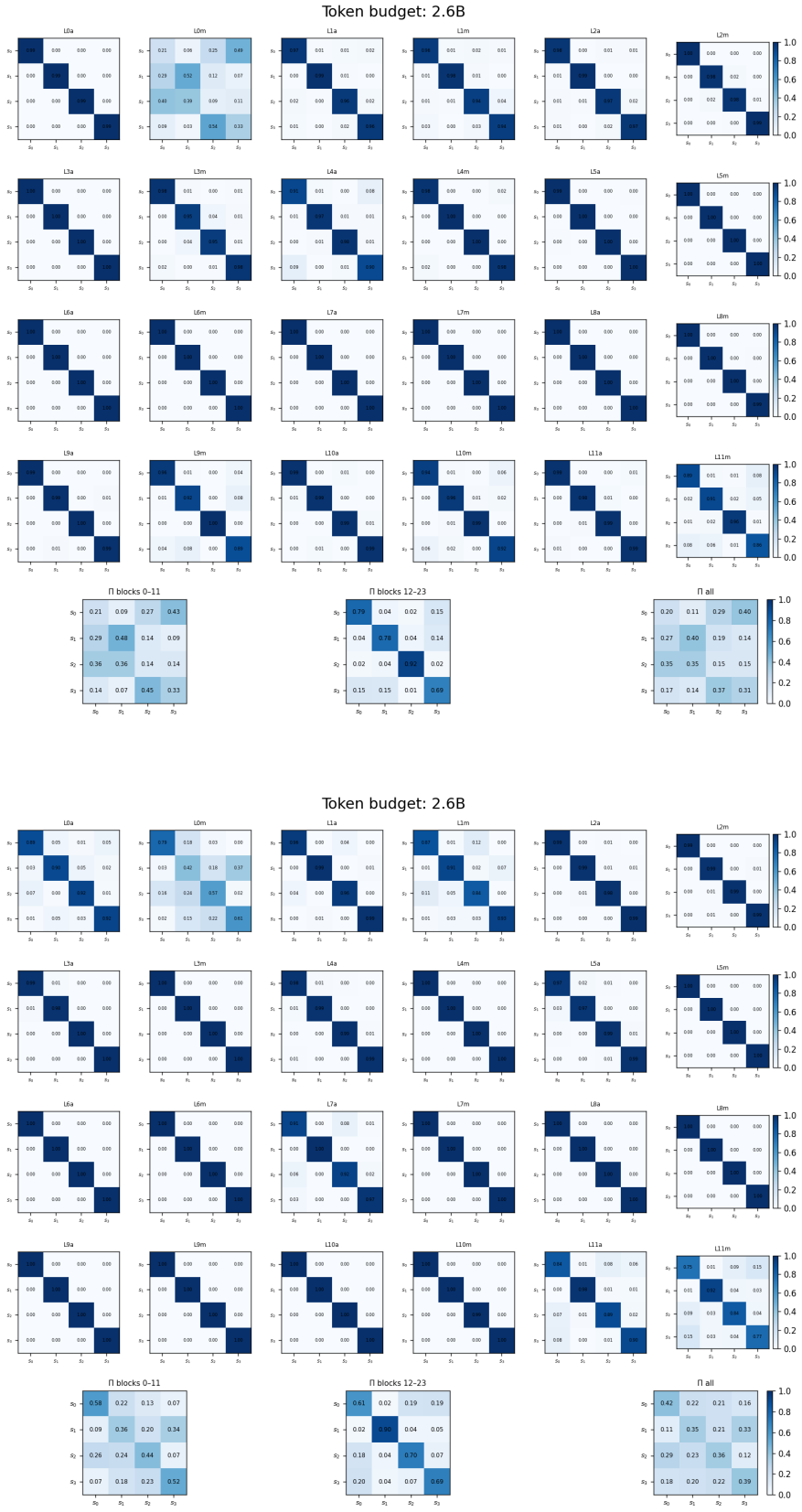

Figure 10. Token budget 2.6B: *m*HC-lite (top), LSS (bottom).

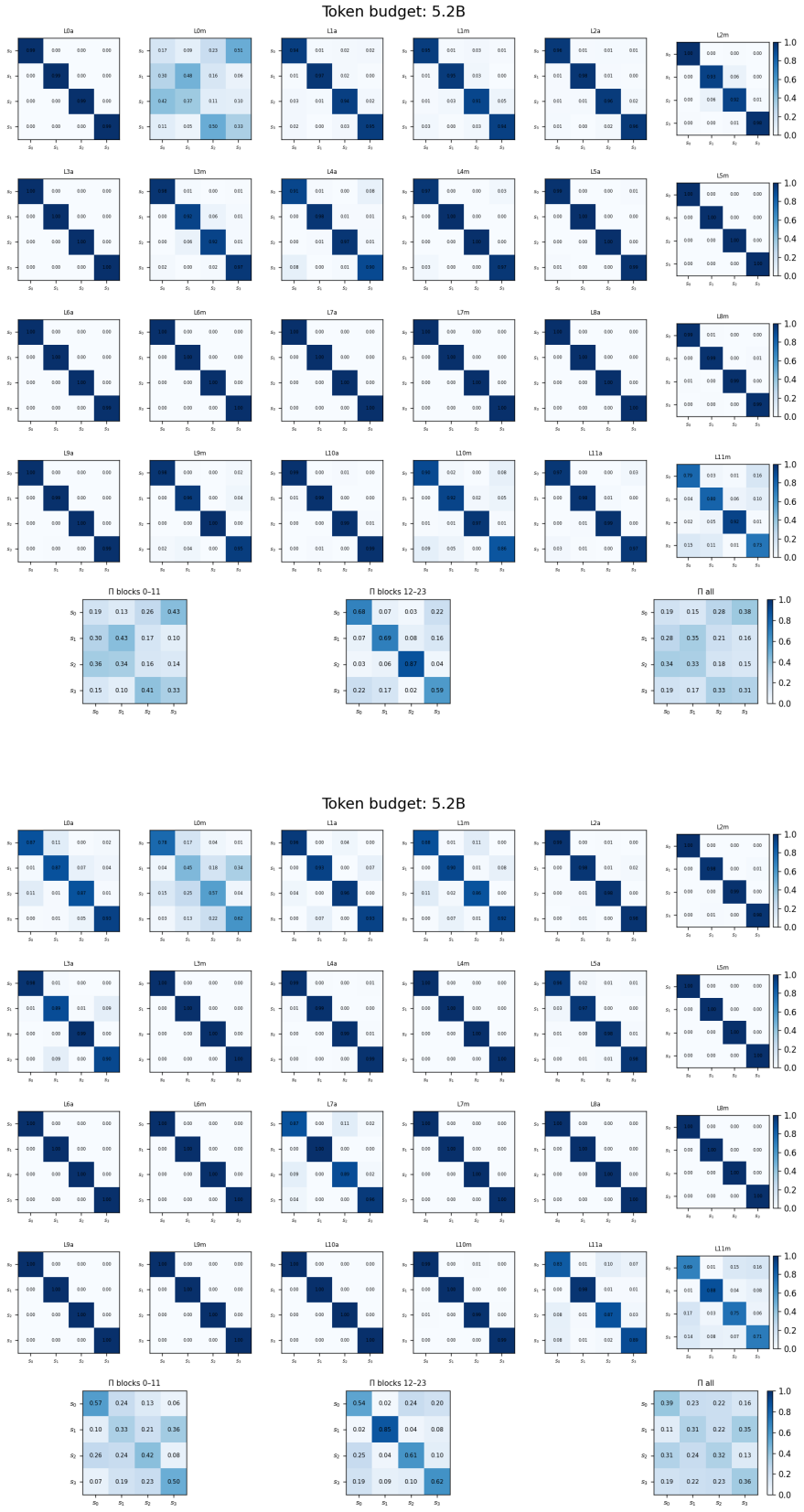

Figure 11. Token budget 5.2B: *m*HC-lite (top), LSS (bottom).