RESIDUAL CONNECTIONS WITH EXPLICIT EMBEDDING REWEIGHTING

Anonymous authors
Paper under double-blind review

ABSTRACT

In a pre-norm Transformer, each token's input embedding enters the residual stream with a fixed coefficient of one. We find that pretrained language models nevertheless reweight the embedding implicitly: across six models from four families, the residual state's projection onto each token's own embedding grows substantially with depth and far exceeds its projection onto other tokens' embeddings. This reweighting is produced indirectly, by attention and MLP layers that, after the first, see the embedding only mixed with earlier updates. To provide a direct path for it, we introduce *Explicit Embedding Reweighting* (EER), a lightweight module that adds the original embedding with a positive coefficient predicted from the current state. EER keeps the identity shortcut and stores no layer outputs or extra embedding tables. At three scales (356M–900M parameters), EER lowers validation loss by 0.022–0.025 nats with about 0.007% additional parameters. At 900M parameters, it raises average zero-shot accuracy on seven tasks by 1.39 points, is competitive with Attention Residuals, and further improves SATFormer and identity Hyper-Connections. Ablations show that its state-dependent coefficient outperforms fixed, per-layer, and token-only coefficients.

1 INTRODUCTION

Transformer language models carry information through their attention and MLP layers with residual connections (Vaswani et al., 2017; He et al., 2016a;b). At each position, the residual stream starts from the token's input embedding and accumulates the outputs of these layers. The embedding depends only on the token, whereas the accumulated outputs also depend on the preceding context. In a standard pre-norm Transformer (Xiong et al., 2020), the residual state is the sum of the two, and the embedding keeps a fixed coefficient of one.

We find that pretrained LLMs nevertheless reweight their input embeddings. Across six models from four families, the state's projection onto each token's own embedding grows substantially with depth and is much larger than its projection onto the embeddings of other tokens (Figure 1; Section 2).

This reweighting is indirect. After the first sublayer, every sublayer sees the embedding only through a normalized mixture with earlier updates, so any reweighting must be produced by these sublayers themselves, within the same state that carries contextual features. Because each sublayer reads the state through RMSNorm, the embedding component sets the scale of all other components relative to the embedding in the next sublayer's input (Section 3.1). We therefore ask whether giving the model a direct path for this reweighting improves prediction. Since the implicit reweighting is itself produced by context-dependent sublayers, we let the coefficient of this direct path depend on the current state as well.

We introduce *Explicit Embedding Reweighting* (EER), a lightweight module inserted after ordinary residual updates. EER predicts a positive injection coefficient from the current state and adds that multiple of the original embedding to the stream. It keeps the identity shortcut, reuses the input embedding, and stores no layer outputs or extra embedding tables. Depth-aggregation methods (Pagliardini et al., 2024; Xiao et al., 2025; Kimi Team et al., 2026) and Hyper-Connections (Zhu et al., 2025) broaden access to earlier representations or streams, and token-memory methods (Jaiswal et al., 2026; Cheng et al., 2026) supply additional knowledge through large token-indexed tables; when the input embedding takes part in these designs, its weight is one

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

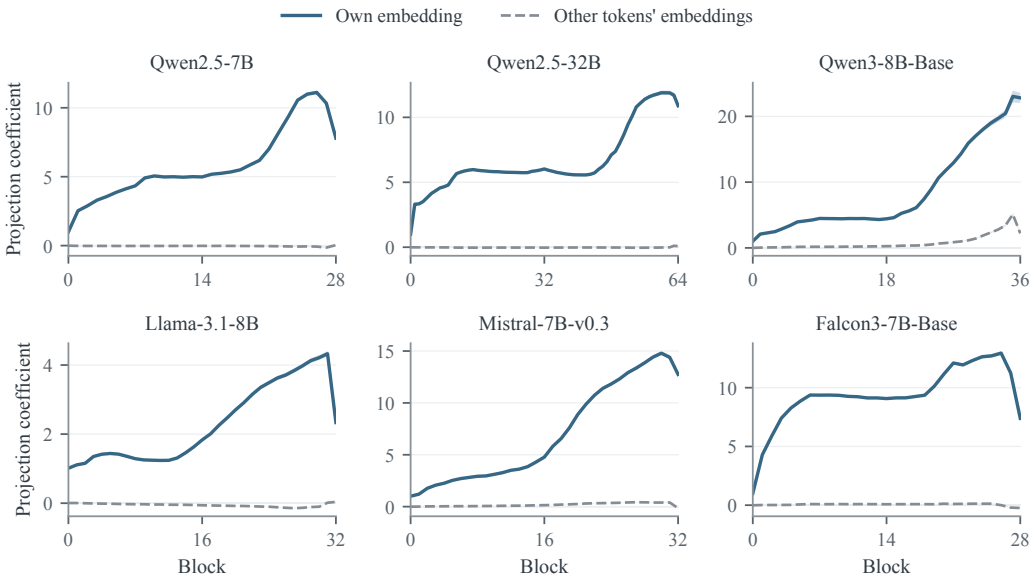

Figure 1: Pretrained language models implicitly reweight their input embeddings. Projection coefficients onto each token's own embedding (solid) grow substantially over most of the depth and remain much larger than those onto other tokens' embeddings, sampled according to their frequency in the measurement sample (dashed). Both curves use reference embeddings centered by their frequency-weighted sample mean; only the reference vectors are centered, and the hidden states are unchanged. Shading: pointwise 95% document-bootstrap intervals. Measurement details are in Appendix C.

term in a larger mixture. EER gives this weight a dedicated control, motivated by the reweighting measured in pretrained models (Section 5 gives a fuller comparison).

We train Small, Medium, and Large models from random initialization, with 356M–900M total parameters and up to 40.90B tokens. EER lowers validation loss by 0.022–0.025 nats at all three scales, adding about 0.007% of total parameters. At the largest scale, it raises average zero-shot accuracy on seven tasks by 1.39 points and is competitive with Attention Residuals (AttnRes) (Kimi Team et al., 2026); added to SATFormer (Gunasekaran et al., 2026) or an identity-residual variant of Hyper-Connections (iHC) (Zhu et al., 2025; Tencent Hy Team, 2026), it further lowers perplexity. In ablations on the Small model, the state-dependent coefficient outperforms fixed, per-layer, and token-only coefficients at the same insertion sites. The learned coefficients follow a pronounced depth profile and vary widely within each block (Section 4.5).

Our contributions are as follows:

- We find that, although the input embedding enters the residual stream with a fixed coefficient, pretrained LLMs implicitly reweight it: in six models from four families, sublayer outputs increase the component along each token's own embedding far more than along other tokens' embeddings (Section 2).
- We propose EER, a lightweight module that adds the original embedding with a coefficient predicted from the current state, giving this reweighting an explicit path (Section 3).
- We show that EER improves language modeling at three scales and downstream accuracy at the largest, is competitive with AttnRes, complements SATFormer and iHC, and outperforms static and token-only coefficients at the same sites (Section 4).

2 Implicit Embedding Reweighting

We consider decoder-only Transformers with standard residual connections and pre-normalization, as used in many recent LLMs (Touvron et al., 2023; Yang et al., 2024a; 2025). Token x_t starts from

its embedding $e_t = E[x_t] \in \mathbb{R}^d$, where E is the learned embedding table and d is the model width; e_t denotes the lookup vector before any normalization. A model with L blocks applies $2L$ residual updates, one per attention or MLP *sublayer*. Let h_t^k be token t's residual state (henceforth *state*) before the kth sublayer update and H^k the states of all tokens. Then

$$h_t^{k+1} = h_t^k + F_k(H^k)_t, \qquad h_t^k = e_t + \sum_{j<k} F_j(H^j)_t, \tag{1}$$

where F_k includes the sublayer's input RMSNorm (Zhang & Sennrich, 2019). The embedding and every sublayer output enter with coefficient one, although the outputs can themselves contain components along e_t.

To measure the state's component along the embedding, we divide the signed projection length by the embedding norm:

$$\alpha_t^k = \frac{\langle h_t^k, e_t \rangle}{\|e_t\|^2} = 1 + \sum_{j<k} \frac{\langle F_j(H^j)_t, e_t \rangle}{\|e_t\|^2}. \tag{2}$$

We call α_t^k the *projection coefficient*. Equivalently, the state decomposes as $h_t^k = \alpha_t^k e_t + h_{t,\perp}^k$, where $h_{t,\perp}^k$ is orthogonal to e_t. Because RMSNorm is invariant to positive rescaling of its input (Zhang & Sennrich, 2019), for $\alpha_t^k > 0$ sublayer k reads $\mathrm{RMSNorm}(h_t^k) = \mathrm{RMSNorm}(e_t + h_{t,\perp}^k/\alpha_t^k)$, ignoring the stabilizer. Compared with a fixed coefficient of one and the same $h_{t,\perp}^k$, the embedding's weight in this input is therefore scaled by α_t^k, which places α_t^k in the same units as EER's direct coefficient (Section 3.1). Sublayers need not write along e_t: the identity path already carries it, and the target is the next token, not the current one. If their outputs were unrelated to e_t, α_t^k would stay near one at every depth. As a control, we compute the same coefficient with the embeddings of other tokens, sampled according to their frequency in the measurement sample; this tests whether the accumulated component is specific to the token's own embedding. Figure 1 centers both references by the frequency-weighted mean embedding without modifying the hidden states; raw references show the same qualitative pattern (Appendix C).

Pretrained models reweight the embedding. We measure own-embedding and other-token coefficients after every block of six pretrained models from four families on 256 FineWeb-Edu documents (Penedo et al., 2024). The results depart sharply from this expectation (Figure 1). In all six models, the mean-removed own-embedding coefficient, about one at the input, grows substantially over most of the depth, then falls in the final block, ending at 2.3–22.8. The coefficient on other tokens' embeddings remains much smaller throughout, so the growth is specific to each token's own embedding rather than a drift shared by all embeddings. Although the embedding's coefficient is fixed at one, the models thus reweight it implicitly: sublayer outputs accumulate a large component along a direction the identity path already carries. Because α_t^k combines the state's norm with its alignment to e_t, it measures a component in embedding units, not relative alignment.

3 Explicit Embedding Reweighting

EER adds a state-dependent multiple of the original embedding after an ordinary residual update. By default, one EER module follows the MLP update of every block (Figure 2); Section 4.4 evaluates other sites, and Appendix A.4 extends EER to identity Hyper-Connections (iHC) (Zhu et al., 2025; Tencent Hy Team, 2026).

3.1 The EER module

Let r_t^ℓ be token t's state after the MLP update of block ℓ. The module's *controller* normalizes this state with its own RMSNorm N_ℓ, takes a dot product with a learned vector w_ℓ, and maps the score to a positive *injection coefficient*:

$$g_t^\ell = \mathrm{softplus}\big(w_\ell^\top N_\ell(r_t^\ell) + b_0\big)\,, \tag{3}$$

where b_0 is a fixed offset. The module then adds g_t^ℓ times the original embedding,

$$\tilde{r}_t^\ell = r_t^\ell + g_t^\ell e_t, \tag{4}$$

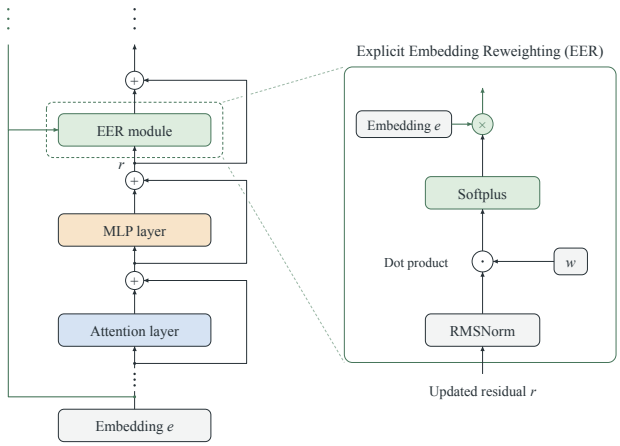

Figure 2: EER after an MLP residual update. Left: the module reads the updated state r and adds a weighted copy of the original embedding e. Right: the controller applies its own RMSNorm, a learned score vector w, and Softplus to predict the positive coefficient g.

and $\widetilde{r}_t^{\ell}$ becomes the input to block $\ell+1$. Because every module has its own controller, the injection coefficient varies with both the current state and depth, while the same unnormalized lookup vector e_t is reused at every site.

Although the addition is linear, its effect on the next sublayer is mediated by RMSNorm, which is invariant to positive rescaling of its input (Zhang & Sennrich, 2019). Dropping indices, write the state as $r = \alpha e + r_\perp$ with $r_\perp \perp e$, where α is the projection coefficient of Equation (2). If $\alpha + g > 0$ and the stabilizer ϵ is ignored,

$$\mathrm{RMSNorm}(r + ge) = \mathrm{RMSNorm}\big((\alpha+g)e + r_\perp\big) = \mathrm{RMSNorm}\big(e + r_\perp/(\alpha+g)\big), \quad (5)$$

because the coordinatewise gain acts after normalization. Thus $r_\perp$, the accumulated updates outside the embedding direction, enters the normalization as $r_\perp/(\alpha+g)$ relative to e. When $\alpha > 0$, injection reduces this relative scale by a factor of $\alpha/(\alpha+g)$. Pre-norm residual connections accumulate layer outputs with unit weights, diluting individual contributions as the state grows (Kimi Team et al., 2026). Equation (5) describes this mechanism from the embedding's viewpoint.

Since the identity shortcut fixes the embedding's direct coefficient and RMSNorm cancels any positive rescaling of the whole state, moving the state along e_t, a direction that depends only on the token, is a natural way to adjust this relative scale. We therefore view the embedding as a token-specific reference that is fixed across depth and the accumulated updates as context-dependent refinements: at each site and position, the controller sets how strongly the state is pulled back toward this reference. The effect depends on α in Equation (5) and the relative injection $g\|e_t\|/\|r\|$ (Appendix A.2), so the same g can act differently across layers.

Each module raises the projection coefficient by exactly g_t^{ℓ} and leaves $r_\perp$ unchanged, reweighting the existing embedding component without introducing new content. After all L blocks, the embedding's *direct coefficient* is $\lambda_t = 1 + \sum_\ell g_t^{\ell}$. Each g_t^{ℓ} is a local increment at one site, and λ_t accumulates the explicit path; the projection coefficient α_t additionally includes the components contributed by sublayer outputs (Appendix A.1).

EER follows the completed residual update, so the previous state and sublayer output retain their unit coefficients (Equation (1)), preserving the identity shortcut (He et al., 2016b). Softplus keeps each injection coefficient positive; allowing the direct coefficient to decrease yields similar losses in our ablations (Section 4.4).

3.2 Implementation and Cost

We initialize each score vector w_ℓ to zero and the RMSNorm scales to one; the fixed offset $b_0 = -3$ gives an initial injection coefficient of about 0.05. Controllers and backbone are trained jointly with

the next-token prediction loss, and gradients through the injections also update the embedding table. Each module adds $2d$ parameters (d RMSNorm scales and a d-dimensional score vector) and $O(d)$ operations per token. The embedding must stay available during the forward pass, and the KV cache is unchanged.

4 EXPERIMENTS

Our experiments address four questions. (1) Does EER improve language modeling across scales, and at what cost (Section 4.2)? (2) Do the gains carry over to downstream tasks, and how does EER compare with and combine with other residual designs (Section 4.3)? (3) Which design choices matter, in particular whether the coefficient must depend on the current state (Section 4.4)? (4) What does the controller learn (Section 4.5)?

4.1 EXPERIMENTAL SETUP

Models. We train Small, Medium, and Large pre-norm, decoder-only Transformers from random initialization, with 356M–900M total parameters (122.7M–511.2M excluding input embeddings and output heads). All use RMSNorm (Zhang & Sennrich, 2019), rotary position embeddings (RoPE) (Su et al., 2021), grouped-query attention (Ainslie et al., 2023), query–key RMS normalization as in Qwen3 (Yang et al., 2025), and gated attention (Qiu et al., 2025).

Compared methods. We compare with three residual designs that add few parameters, each a recent improvement in one of the lines of work reviewed in Section 5: SATFormer (SAT) (Gunasekaran et al., 2026) for reusing early attention values, Attention Residuals (AttnRes) (Kimi Team et al., 2026) for aggregation across depth, and identity Hyper-Connections (iHC) (Tencent Hy Team, 2026) for widening the stream. Each gives later layers some access to early information, so together they test whether EER's gain is already covered by existing mechanisms. For AttnRes, we use Block AttnRes with eight depth blocks (three Transformer blocks each), the default setting of Kimi Team et al. (2026), and keep the baseline's gated attention; iHC uses four streams (Appendix A.4). When adding EER to SAT or iHC, we keep the design unchanged and insert EER after the MLP updates. All methods, including Baseline, are implemented in PyTorch and compiled with `torch.compile` (Ansel et al., 2024), without custom or fused kernels; such kernels could further reduce the wall-clock overhead of EER and the other residual designs.

Training. All models are trained on the FineWeb-Edu corpus (Penedo et al., 2024) with about 80 tokens per baseline non-embedding parameter, 2,048-token sequences, and 524,288 tokens per update. At each scale, we select Baseline's peak learning rate by grid search and use it, together with all other settings, for every method; no method is tuned separately. Values with $\pm$ are means and standard deviations over three seeds (0, 1, and 42). Table 1 lists model sizes and token budgets, and Appendix B gives the full protocol.

Evaluation. We report final FineWeb-Edu validation cross-entropy (nats) and zero-shot accuracy on seven benchmarks: HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2020), ARC-Easy and ARC-Challenge (Clark et al., 2018), SciQ (Welbl et al., 2017), and LAMBADA (Paperno et al., 2016) (Appendix B.2).

4.2 LANGUAGE MODELING ACROSS SCALES

Table 1 compares Baseline and EER under matched token budgets. EER lowers validation loss by 0.022–0.025 nats at all three scales, or 2.1–2.5% in perplexity. Model size and token budget grow together across the three settings, so we treat each pair as a separate result rather than as a point on a scaling curve (Section 6). The gap appears early and persists: EER has lower validation loss at every intermediate and final evaluation across the three scales (Appendix B.1).

These gains come at almost no cost in parameters or FLOPs. EER adds about 0.007% of total parameters (0.012–0.020% of non-embedding parameters) and 0.015–0.018% forward FLOPs per 2,048-token sequence. The wall-clock cost is higher: training the Large model takes 3.8% longer (Table 2), because FLOP counts miss the memory-bound normalization and elementwise operations, which we run without custom kernels.

Table 1: EER lowers validation loss at all three scales. Each pair shares its token budget; EER follows every MLP update.

Scale	Params.†	Tokens	L	d	Forward FLOPs (TFLOPs)		Val. loss		
					Baseline	EER	Baseline	EER	Reduction↑
Small	122.7M	9.82B	16	768	1.18963	1.18984	2.789	**2.764**	0.0250
Medium	272.7M	21.81B	20	1024	2.10258	2.10291	2.581	**2.559**	0.0217
Large	511.2M	40.90B	24	1280	3.41331	3.41381	2.440	**2.415**	0.0249

†Baseline parameters excluding input embedding and output head. L: blocks; d: model width. FLOPs are per forward pass on one 2,048-token sequence (batch size 1).

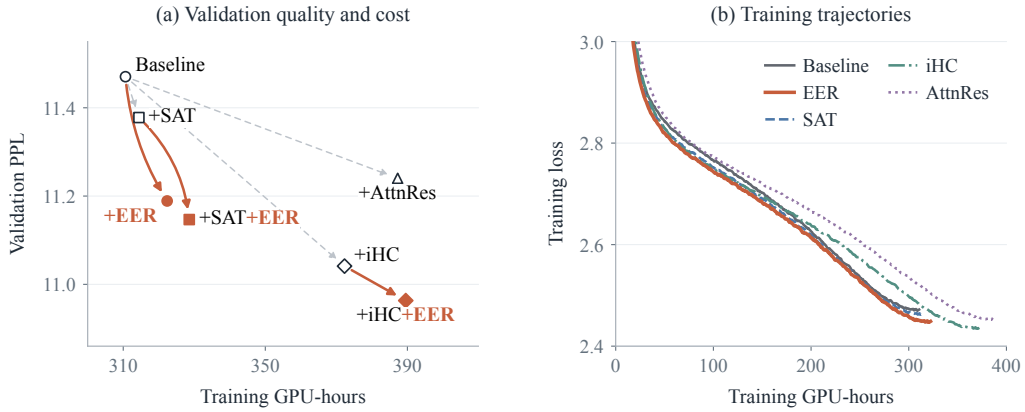

Figure 3: Quality and training cost at Large scale. (a) Validation perplexity versus training GPU-hours (Table 2). Solid orange arrows add EER; dashed gray arrows add SAT, iHC, or AttnRes to Baseline. (b) Training loss versus GPU-hours (500-update trailing mean). At a given GPU-hour budget, each curve is at a different stage of its own learning-rate schedule, so this panel reflects training speed; final quality is compared in panel (a).

4.3 Downstream accuracy and comparison with residual designs

Downstream accuracy. At Large scale, EER raises the seven-task zero-shot average from 59.36% to 60.75% (Table 2). The improvement is spread across tasks: five tasks gain 1.3–2.7 points, PIQA gains 0.6, and only ARC-Challenge is essentially flat (35.64% vs. 35.81% for Baseline).

Comparison with other residual designs. Added alone, EER and Block AttnRes perform similarly (perplexity 11.19 vs. 11.24; average accuracy 60.75% vs. 60.44%), although AttnRes attends over the embedding and depth-block summaries while EER reuses only the embedding. They differ mainly in cost: relative to Baseline, EER adds 3.8% training GPU-hours and AttnRes 25%. iHC, which widens the residual stream to four copies, reaches lower perplexity than either (11.04) at about 20% more GPU-hours than Baseline (Figure 3a).

The combinations test whether these designs already provide what EER adds. SATFormer lets later attention layers read first-layer values directly, yet EER still lowers its perplexity by 2% and improves the mean accuracy on all seven tasks (+0.92 points in average accuracy); at Medium scale, it lowers SATFormer's validation loss by 0.0116 nats (Appendix D.1). The remaining gain points to a benefit beyond supplying early information to attention: EER changes the embedding component of the state that both attention and MLP layers read.

On iHC the gain is smaller, 0.7% in perplexity and 0.24 points in accuracy, although the combination reaches the lowest perplexity in our study (10.96). With four streams, iHC may already retain more

Table 2: Large models. Validation perplexity (PPL) and task accuracies are means, with standard deviations in the row below; AVG is the mean of the seven task means. GPU-hours follow Appendix B.3. Horizontal rules separate groups; bold marks the best value within each group. LMB.: LAMBADA; HELLA.: HellaSwag; WINO.: WinoGrande; ARC-E/C: ARC-Easy/Challenge; AVG.: average.

Model	GPU-h ↓	PPL ↓	Downstream accuracy (%) ↑							
			LMB.	PIQA	HELLA.	WINO.	ARC-E	ARC-C	SCIQ	AVG.
Baseline	**310.63**	11.47 ±0.0184	43.31 ±0.32	71.33 ±0.45	53.32 ±0.22	56.77 ±1.09	66.88 ±0.94	**35.81** ±0.60	88.07 ±0.52	59.36
+AttnRes	387.26	11.24 ±0.0202	44.93 ±0.41	71.27 ±0.54	54.15 ±0.16	**58.80** ±0.65	68.60 ±0.87	35.67 ±0.62	89.63 ±0.46	60.44
+EER	322.36	**11.19** ±0.0134	**45.14** ±0.38	**71.89** ±0.27	**54.62** ±0.31	58.51 ±0.89	**69.61** ±0.93	35.64 ±0.38	**89.83** ±0.36	**60.75**
+SAT	**314.42**	11.38 ±0.0341	43.94 ±0.43	71.49 ±0.62	53.64 ±0.19	57.85 ±0.67	68.86 ±0.77	34.73 ±0.81	88.40 ±0.70	59.84
+SAT+EER	328.59	**11.15** ±0.0234	**45.64** ±1.21	**71.65** ±0.47	**54.91** ±0.17	**58.09** ±0.71	**69.15** ±0.87	**36.52** ±0.39	**89.33** ±0.30	**60.76**
+iHC	**372.30**	11.04 ±0.0210	44.67 ±0.32	72.09 ±0.46	54.76 ±0.22	57.70 ±0.83	**70.02** ±1.21	36.60 ±0.92	90.63 ±0.56	60.92
+iHC+EER	389.54	**10.96** ±0.0153	**44.71** ±0.28	**72.31** ±0.54	**55.42** ±0.15	**58.12** ±0.56	69.91 ±0.61	**36.80** ±0.32	**90.83** ±0.45	**61.16**

Table 3: Design ablations at Small scale (validation loss). (a) Coefficient rules with one injection after every MLP; means and standard deviations over three seeds. (b) Loss reduction relative to Baseline (L: number of blocks).

(a) Coefficient rule

Coefficient	Val. loss ↓
None (Baseline)	2.789 ±0.002
Fixed ($g = 1$)	2.782 ±0.004
Learned per layer	2.780 ±0.009
Token-only controller	2.775 ±0.003
State-dependent (EER)	**2.764** ±0.002

(b) Insertion sites

After	Sites	Reduction ↑
every MLP	L	0.0250
every 2nd MLP	$L/2$	0.0234
every 4th MLP	$L/4$	0.0190
every 8th MLP	$L/8$	0.0135
final MLP only	1	0.0073
every attention	L	0.0230
attention and MLP	$2L$	0.0239

of the earlier information. Adding EER to either SATFormer or iHC increases GPU-hours by under 5% relative to the corresponding model without EER.

4.4 Design ablations

Coefficient rule. Table 3a keeps one module after every MLP in the Small model and changes only how the injection coefficient is set. Re-adding the embedding with a fixed coefficient of one lowers the loss slightly (2.782 vs. 2.789), and learning a per-layer scalar lowers it only slightly further (2.780), within the variation across seeds. A token-only controller, which predicts the coefficient from the normalized embedding with the same parameter count as EER, reaches 2.775. Conditioning on the state instead brings the loss to 2.764. The 0.011-nat gap is larger than the standard deviations of both controllers across seeds (0.003 and 0.002) and supports conditioning the coefficient on the current state rather than token identity alone. Predicting the total direct coefficient allows it to decrease while retaining similar validation loss: a separate Small Add/Total comparison gives 2.766 for both rules, and the Large comparison gives 2.417 for Total versus 2.415 for EER (Appendices D.3 and D.4).

Insertion sites. Table 3b varies where the modules sit. Inserting after every second MLP keeps most of the full gain (0.0234 vs. 0.0250 nats) with half as many modules, but the gain falls steadily

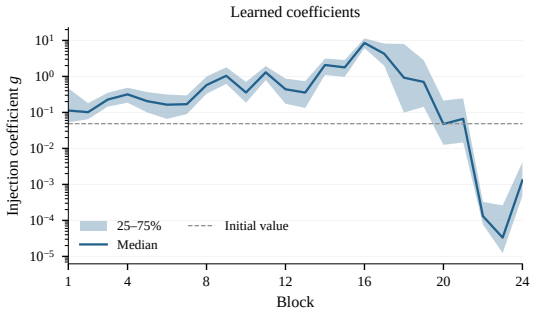

Figure 4: Learned EER coefficients at Large scale. Per-block distributions over validation tokens: the solid line shows the median, the shaded band spans the 25th–75th percentiles, and the dashed line marks the initial value.

with sparser insertion, down to 0.0073 for a single module after the final MLP. A single injection near the output does not replace repeated injection through the network. Inserting after attention updates works about as well as after MLP updates (0.0230), and inserting at both does not improve on MLP-only insertion (0.0239). At Large scale, inserting at both sites is also worse (0.0196 vs. 0.0249) and takes about 4% more GPU-hours than MLP-only insertion (Appendix D.4), so we insert after every MLP by default.

4.5 What the controller learns

All controllers start at about 0.05, but after training the injection coefficients of the Large model follow a clear depth profile (Figure 4). Early blocks inject little (medians 0.1–0.3), the medians rise through the middle of the network to 8.5 at block 16, and the last three blocks almost switch the injection off (both medians and means below 0.01). These are local injection coefficients, distinct from the accumulated projection coefficients in Figure 1. Within a block, the coefficient spans one to three orders of magnitude across tokens; in block 18, the 10th and 90th percentiles are 0.02 and 14.2. A fixed or per-layer coefficient cannot represent this spread, which is consistent with the ablation above. The relative strength of injection also depends on $\|e_t\|/\|r\|$ (Appendix A.2), so g alone does not measure injection strength across blocks.

5 Related Work

Reuse of initial and early representations. Value Residual Learning reuses first-layer attention values in later layers (Zhou et al., 2025), and SATFormer gates this reuse per token and KV head (Gunasekaran et al., 2026). Several methods instead reuse the initial embedding directly or through learned projections. ExoFormer learns embedding-derived projections for attention anchors (Su, 2026), Lexical Shortcuts mix embedding-derived signals into attention keys and values (Emelin et al., 2019), and SIBO mixes initial representations into adapter and LoRA inputs with a fixed coefficient (Wen et al., 2024). Input injection is also used in looped Transformers (Yang et al., 2024b) and deep equilibrium sequence models (Bai et al., 2019), where a weight-shared computation repeatedly accesses the input.

Aggregation across depth. DenseFormer forms the next block's input by a learned, static weighted sum of the input embedding and current and earlier block outputs (Pagliardini et al., 2024). DeepCrossAttention (Heddes et al., 2025), MUDDFormer (Xiao et al., 2025), and Attention Residuals (Kimi Team et al., 2026) combine earlier representations, including the input, with input-dependent weights. AttnRes computes these weights by attention over depth: Full AttnRes attends over earlier sublayer outputs, while Block AttnRes attends over block summaries and recovers most of the gain of the full version (Kimi Team et al., 2026). These designs retain depth history, and sparse or blockwise variants reduce how many representations are stored; EER needs only the embedding beside the current state.

Residual routing. Hyper-Connections learn read, residual-mixing, and write maps over expanded streams (Zhu et al., 2025), and mHC constrains the mixing map to be doubly stochastic (Xie et al., 2025). The identity-residual variant used in Hy4 preview (iHC) (Tencent Hy Team, 2026) fixes the mixing map to the identity, so each stream is retained through an identity shortcut. Qwen3.8-Flash-Next's Gated Residual follows a similar design with widened streams and identity residual paths, but uses elementwise read gates (Qwen Team, 2026). We use iHC as a simple and effective multi-stream baseline and find that adding EER further improves it (Section 4.3).

Additional token memory. Several designs add token-indexed parameters that supply token-level content beyond the input embedding. TIDE targets under-trained rare-token embeddings and contextual collapse by adding K independent vocabulary-sized token tables, whose mixture is injected into every layer through a softmax router over the state with a learnable null bank (Jaiswal et al., 2026). Engram retrieves memory vectors by hashed local N-grams and fuses them with context-aware gating (Cheng et al., 2026); Gemma 3n adds per-layer token embeddings (Gemma Team, 2025); and Bank of Values replaces value projections in deeper attention layers with token-indexed tables (He et al., 2026). These methods expand the token-level content available to the network. EER adds no token-indexed parameters and introduces no new content: it controls the coefficient of the existing embedding term, motivated by the reweighting measured in Section 2. Our controlled ablations therefore focus on rules that reuse this same embedding at the same insertion sites (Section 4.4).

6 Discussion and Limitations

Our experiments cover models with up to 900M total parameters (511.2M non-embedding), trained on up to 40.90B FineWeb-Edu tokens. Every run exceeds the compute-optimal budget of about 20 tokens per parameter (Hoffmann et al., 2022) under either parameter count: about 80 tokens per non-embedding parameter, or 28–45 per total parameter. The gain is similar across the three settings (0.022–0.025 nats), but we have not tested larger models or the much higher token-to-parameter ratios used to train current LLMs.

7 Conclusion

We find that pretrained language models implicitly reweight each token's input embedding across depth, although the residual stream fixes its direct coefficient and sublayers after the first see it only mixed with earlier updates. EER adds an explicit, state-dependent path for this reweighting: a small controller reads the current state and adds a positive multiple of the original embedding, with about 0.007% additional parameters. EER lowers validation loss at all three evaluated scales, improves downstream accuracy at 900M parameters, is competitive with Attention Residuals, and complements SATFormer and identity Hyper-Connections. A state-dependent coefficient outperforms static and token-only alternatives. These results suggest that the direct coefficient of the input embedding, fixed at one in standard residual streams, is worth exposing as an explicit, state-dependent control.

AI use statement

We used Codex to assist with manuscript revision and coding.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.298. URL https://aclanthology.org/2023.emnlp-main.298/.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elias Ellison, Will Feng, Jiong Gong, Michael Gschwind, Brian Hirsh, Sherlock Huang, Kshiteej Kalambarkar, Laurent Kirsch, Michael Lazos, Mario Lezcano, Yanbo Liang, Jason Liang, Yinghai Lu, CK Luk, Bert Maher, Yunjie Pan, Christian Puhrsch, Matthias Reso, Mark Saroufim, Marcos Yukio Siraichi, Helen Suk, Michael Suo, Phil Tillet, Eikan Wang, Xiaodong Wang, William Wen, Shunting Zhang, Xu Zhao, Keren Zhou, Richard Zou, Ajit Mathews, Gregory Chanan, Peng Wu, and Soumith Chintala. PyTorch 2: Faster machine learning through dynamic Python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. ACM, 2024. doi: 10.1145/3620665.3640366. URL https://docs.pytorch.org/assets/pytorch2-2.pdf.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/hash/01386bd6d8e091c2ab4c7c7de644d37b-Abstract.html.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020. doi: 10.1609/aaai.v34i05.6239. URL https://ojs.aaai.org/index.php/AAAI/article/view/6239.

Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Yukun Li, Han Zhang, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. Conditional memory via scalable lookup: A new axis of sparsity for large language models, 2026. URL https://arxiv.org/abs/2601.07372v1.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge, 2018. URL https://arxiv.org/abs/1803.05457.

Denis Emelin, Ivan Titov, and Rico Sennrich. Widening the representation bottleneck in neural machine translation with lexical shortcuts. In *Proceedings of the Fourth Conference on Machine Translation (Volume 1: Research Papers)*, pp. 102–115. Association for Computational Linguistics, 2019. doi: 10.18653/v1/W19-5211. URL https://aclanthology.org/W19-5211/.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

Gemma Team. Gemma 3n. Model documentation, 2025. URL https://ai.google.dev/gemma/docs/gemma-3n.

Skye Gunasekaran, Téa Wright, Rui-Jie Zhu, and Jason Eshraghian. Transformers with selective access to early representations, 2026. URL https://arxiv.org/abs/2605.03953.

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 (CVPR)*, pp. 770–778, 2016a. doi: 10.1109/CVPR.2016.90.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision – ECCV 2016*, pp. 630–645, Cham, 2016b. Springer International Publishing. URL https://arxiv.org/abs/1603.05027.

Muyu He, Yuchen Liu, Qingya Huang, and Li Zhang. Do value vectors in deep layers need context from the residual stream?, 2026. URL https://arxiv.org/abs/2606.02780.

Mike Heddes, Adel Javanmard, Kyriakos Axiotis, Gang Fu, Mohammadhossein Bateni, and Vahab Mirrokni. DeepCrossAttention: Supercharging transformer residual connections. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Pro-*

ceedings of Machine Learning Research, pp. 22881–22903. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/heddes25a.html.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

Ajay Jaiswal, Lauren Hannah, Han-Byul Kim, Duc Hoang, Mehrdad Farajtabar, and Minsik Cho. TIDE: Every layer knows the token beneath the context, 2026. URL https://arxiv.org/abs/2605.06216.

Keller Jordan, Jeremy Bernstein, Brendan Rappazzo, @fernbear.bsky.social, Boza Vlado, You Jiacheng, Franz Cesista, Braden Koszarsky, and @Grad62304977. modded-nanogpt: Speedrunning the NanoGPT baseline. Software, 2024. URL https://github.com/KellerJordan/modded-nanogpt/tree/master/records/track_1_short/2024-11-06_ShortcutsTweaks. Embedding-skip implementation dated November 6, 2024.

Kimi Team, Guangyu Chen, Yu Zhang, Jianlin Su, Weixin Xu, Siyuan Pan, Yaoyu Wang, Yucheng Wang, Guanduo Chen, Bohong Yin, Yutian Chen, Junjie Yan, Ming Wei, Y. Zhang, Fanqing Meng, Chao Hong, Xiaotong Xie, Shaowei Liu, Enzhe Lu, Yunpeng Tai, Yanru Chen, Xin Men, Haiqing Guo, Y. Charles, Haoyu Lu, Lin Sui, Jinguo Zhu, Zaida Zhou, Weiran He, Weixiao Huang, Xinran Xu, Yuzhi Wang, Guokun Lai, Yulun Du, Yuxin Wu, Zhilin Yang, and Xinyu Zhou. Attention residuals, 2026. URL https://arxiv.org/abs/2603.15031.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1711.05101.

Meta. Llama 3.1 model card. Model documentation, 2024. URL https://huggingface.co/meta-llama/Llama-3.1-8B.

Mistral AI. Model card for Mistral-7B-v0.3. Model documentation, 2024. URL https://huggingface.co/mistralai/Mistral-7B-v0.3.

Matteo Pagliardini, Amirkeivan Mohtashami, François Fleuret, and Martin Jaggi. DenseFormer: Enhancing information flow in transformers via depth weighted averaging. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/f67449c7ab72f441d3a713b046c6818c-Abstract-Conference.html.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534. Association for Computational Linguistics, 2016. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144/.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems*, volume 37, pp. 30811–30849. Curran Associates, Inc., 2024. doi: 10.52202/079017-0970. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/370df50ccfdf8bde18f8f9c2d9151bda-Paper-Datasets_and_Benchmarks_Track.pdf.

Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. In *Advances in Neural Information Processing Systems*, volume 38, pp. 100092–100118. Curran Associates, Inc., 2025. doi: 10.52202/085713-3345. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/904e89bb4e632e75fb47f093b620b257-Paper-Conference.pdf.

Qwen Team. Qwen2.5: A party of foundation models. Official model release, September 2024. URL https://qwenlm.github.io/blog/qwen2.5/.

Qwen Team. On the design of Qwen3.8-Next architecture: Evaluation, efficiency, and training stability, 2026. URL https://arxiv.org/abs/2608.30320.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020. doi: 10.1609/aaai.v34i05.6399. URL

https://ojs.aaai.org/index.php/AAAI/article/view/6399.

Noam Shazeer. GLU variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding, 2021. URL https://arxiv.org/abs/2104.09864.

Jonathan Su. Attention projection mixing with exogenous anchors, 2026. URL https://arxiv.org/abs/2601.08131.

Tencent Hy Team. Hy4 preview. Model release, 2026. URL https://github.com/Tencent-Hunyuan/Hy4-preview.

TII Team. Falcon 3 family of open foundation models. Model documentation, December 2024. URL https://huggingface.co/tiiuae/Falcon3-7B-Base.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://arxiv.org/abs/1706.03762.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106. Association for Computational Linguistics, 2017. doi: 10.18653/v1/W17-4413. URL https://aclanthology.org/W17-4413/.

Zhihao Wen, Jie Zhang, and Yuan Fang. SIBO: A simple booster for parameter-efficient fine-tuning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 1241–1257, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.72. URL https://aclanthology.org/2024.findings-acl.72/.

Da Xiao, Qingye Meng, Shengping Li, and Xingyuan Yuan. MUDDFormer: Breaking residual bottlenecks in transformers via multiway dynamic dense connections. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 68440–68458. PMLR, 2025. URL https://proceedings.mlr.press/v267/xiao25d.html.

Zhenda Xie, Yixuan Wei, Huanqi Cao, Chenggang Zhao, Chengqi Deng, Jiashi Li, Damai Dai, Huazuo Gao, Jiang Chang, Kuai Yu, Liang Zhao, Shangyan Zhou, Zhean Xu, Zhengyan Zhang, Wangding Zeng, Shengding Hu, Yuqing Wang, Jingyang Yuan, Lean Wang, and Wenfeng Liang. mHC: Manifold-constrained hyper-connections, 2025. URL https://arxiv.org/abs/2512.24880.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 10524–10533. PMLR, 2020. URL https://proceedings.mlr.press/v119/xiong20b.html.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu,

Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report, 2024a. URL https://arxiv.org/abs/2407.10671.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Liu Yang, Kangwook Lee, Robert D. Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/b8402301e7f06bdc97a31bfaa653dc32-Abstract-Conference.html.

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. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/1e8a19426224ca89e83cef47f1e7f53b-Paper.pdf.

Zhanchao Zhou, Tianyi Wu, Zhiyun Jiang, Fares Obeid, and Zhenzhong Lan. Value residual learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28341–28356, Vienna, Austria, July 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.1375. URL https://aclanthology.org/2025.acl-long.1375/.

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*, pp. 97183–97219, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/f1e8ff97057e0c3f6bcb7018c78d4df1-Paper-Conference.pdf.

A Mathematical and Implementation Details

A.1 Direct coefficient

With EER after the MLP update of every block in a set S, and $g_t^\ell = 0$ for $\ell \notin S$, unrolling Equations (1) and (4) gives the final state

$$h_t^{2L} = \lambda_t e_t + \sum_{k=0}^{2L-1} F_k(H^k)_t, \qquad \lambda_t = 1 + \sum_{\ell \in S} g_t^\ell, \tag{6}$$

where every sublayer output is computed from the EER model's own states. The projection coefficient is $\alpha_t = \lambda_t + \sum_k \langle F_k(H^k)_t, e_t\rangle / \|e_t\|^2$: the direct coefficient λ_t counts the explicit path, and the remainder is contributed by sublayer outputs.

A.2 Local direction and normalization

For fixed nonzero e and a state r not collinear with e, let $\widehat{e} = e/\|e\|$, $p = \widehat{e}^\top r$, and $\beta = \|r - p\widehat{e}\| > 0$. The angle θ between $r + ge$ and e satisfies

$$\cos\theta(g) = \frac{p + g\|e\|}{\sqrt{(p + g\|e\|)^2 + \beta^2}}, \qquad \frac{d\theta}{dg} = -\frac{\|e\|\beta}{(p + g\|e\|)^2 + \beta^2} < 0. \tag{7}$$

An injection therefore turns the state toward e; later sublayer updates need not preserve this alignment. RMSNorm divides by a scalar magnitude before applying learned coordinatewise gains (Zhang & Sennrich, 2019), so the direction change generally survives normalization. The size of the change is set by the relative injection $g\|e\|/\|r\|$, not by g alone.

A.3 Identity shortcut and controller derivative

With embeddings and parameters fixed, let $J_{ts}^F = \partial F_k(H^k)_t / \partial h_s^k$ for the MLP sublayer k of block ℓ, and let $v_t = \nabla_{r_t^\ell} g_t^\ell$. The Jacobian blocks of one MLP update followed by EER are

$$\frac{\partial \widetilde{r}_t^\ell}{\partial h_s^k} = (I + e_t v_t^\top)(\delta_{ts} I + J_{ts}^F), \tag{8}$$

so the same-token block keeps its identity term alongside the sublayer and controller derivatives. During training, the injections also pass gradients to the embedding table.

A.4 Identity Hyper-Connections and post-write injection

Our implementation of identity Hyper-Connections (iHC) maintains four residual streams for each token, matching the default iHC configuration. A sublayer reads a weighted combination of the streams, then distributes its output back to them while retaining each incoming stream with coefficient one. EER acts after this write, adding the original embedding to each updated stream with its own positive coefficient.

For one token, let $X \in \mathbb{R}^{d \times m}$ contain residual streams as columns, with $m = 4$. The iHC read, write, and EER addition are

$$z = Xa, \qquad \widetilde{X} = X + F(Z)_t b^\top, \qquad X^+ = \widetilde{X} + e_t g^\top. \tag{9}$$

Here $\widetilde{X}$ contains the updated streams and $g \in \mathbb{R}^m_{>0}$ gives one EER coefficient per stream. Z collects the sublayer inputs z for all token positions, so $F(Z)_t$ is the sublayer output for the current token. The vectors $a, b \in \mathbb{R}^m$ are read and write gates. General HC can replace the identity residual propagation with a learned stream-mixing matrix (Zhu et al., 2025); iHC fixes it to I_m. Each stream starts from e_t, and the EER coefficients g_j are not multiplied by the write gates b_j.

The gates read RMS-normalized flattened input streams: $a = \sigma(s_a u_a + c_a) + \epsilon_{\mathrm{HC}}$ and $b = 2\sigma(s_b u_b + c_b) + \epsilon_{\mathrm{HC}}$ elementwise, where u_a, u_b are learned linear projections. We use $m = 4$, $\epsilon_{\mathrm{HC}} = 10^{-6}$, initial $s_a = s_b = 0.01$, $c_a = -\log(m-1)\mathbf{1}_m$, and $c_b = 0$; projection weights use the backbone's standard initialization (standard deviation 0.02). Initial gates are near $1/m$ and one, respectively; read gates need not sum to one. Each attention and MLP site has its own gate module. A final learned sigmoid reducer uses the same normalized flattened input, scale 0.01, and read-base initialization to combine streams before final RMSNorm and the language-model head.

A.5 Controller for iHC

Let $\widetilde{x}_j$ be column j of $\widetilde{X}$ and N_A the controller's learned RMSNorm. The Large iHC combination uses shared per-stream scoring:

$$g_j = \mathrm{softplus}\big(w^\top N_A(\widetilde{x}_j) + b_0\big)\,, \tag{10}$$

with the same fixed offset $b_0 = -3$ as in Equation (3). The RMSNorm scales and the score vector are shared across streams, so each site adds $2d$ parameters. Controllers are independent across sites, with zero initial score weights and unit RMSNorm scales. The combination uses every-MLP insertion.

B Experimental and Measurement Protocols

B.1 Data, architecture, and optimization

We use the `sample-100BT` subset of FineWeb-Edu (Penedo et al., 2024) and the tokenizer from `Qwen/Qwen3-0.6B` (Yang et al., 2025). All token budgets count tokens after tokenization and packing. We reserve the first 1,000 documents for validation, exclude them from training, shuffle the remaining stream with seed 42 and a 10,000-document buffer, append EOS to each document, and pack 2,048-token blocks. The resulting corpus contains 48,659,206 training blocks (99,654,053,888 tokens) and 538 validation blocks (1,101,824 tokens).

All models are trained from random initialization on this data under the setup in Section 4.1.

Table 4: Backbone specifications and complete training budgets. N excludes both the input embedding and the untied output head; P includes them. All variants inherit the scale's backbone and baseline token budget.

	Small	Medium	Large
Blocks	16	20	24
Hidden width	768	1,024	1,280
MLP width	2,304	3,072	3,840
Query / KV heads	6 / 3	8 / 4	10 / 5
N (M)	122.713	272.677	511.250
P (M)	356.087	583.842	900.206
Tokens (B)	9.817293	21.814575	40.900231
Optimizer updates	18,725	41,608	78,011
Peak learning rate	0.006	0.005	0.002
GPU processes	8	8	32
Sequences / GPU	8	8	4
Gradient accumulation	4	4	2

The decoder uses grouped-query attention (Ainslie et al., 2023), with RMSNorm (Zhang & Sennrich, 2019) before each attention and MLP sublayer. Queries and keys are also RMS-normalized over the head dimension, following Qwen3 (Yang et al., 2025), before RoPE (Su et al., 2021). We use a RoPE base of 10^6 and RMSNorm stability constant $\epsilon = 10^{-6}$. The MLP uses SwiGLU (Shazeer, 2020). The decoder has no attention projection bias or dropout, and uses an untied output head. Vocabulary size is 151,936 and head dimension is 128. Elementwise sigmoid gating is applied to attention outputs before the output projection (Qiu et al., 2025). Every optimizer update uses 256 sequences, or 524,288 tokens. AdamW (Loshchilov & Hutter, 2019) uses $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-6}$, weight decay 0.1 (including the embedding and output head), and gradient clipping at norm 1. The schedule warms up for 2,000 updates and then uses cosine decay to 10^{-5}. Training uses BF16, SDPA with automatic backend choice, and `torch.compile`/Inductor in default mode, without gradient checkpointing.

Biases and the backbone's RMSNorm scales before attention and MLP, on queries and keys, and before the output head use zero weight decay. The EER controller's RMSNorm scales use weight decay 0.1.

At each scale, the peak learning rate is selected for Baseline by grid search with other settings fixed, and shared by all variants; no variant is tuned separately. Token budgets are set from Baseline's parameter count and rounded up to whole optimizer updates; variants use the same budget.

Validation runs every 1,907 optimizer updates and at the final update. Validation loss is the mean next-token cross-entropy over fixed-length packed sequences with the same label masking as in training. Labels copy the input IDs, with the usual causal shift and no extra EOS or document-boundary masking. Each block therefore supplies 2,047 prediction targets; the token counts above count 2,048 input tokens per block. We report the final evaluation rather than the checkpoint with the lowest validation loss. Across the three Baseline/EER pairs in Table 1, EER has lower validation loss at all 10, 22, and 41 intermediate and final evaluations for Small, Medium, and Large.

B.2 Downstream evaluation protocol

We evaluate with the Language Model Evaluation Harness (Gao et al., 2023). We evaluate the seven Large configurations in Table 2 at the final training step, 78,011, using the same tokenizer as in training, BF16 inference with SDPA, and a 2,048-token context. EER follows every MLP update; the Large iHC combination uses four streams and the shared per-stream controller in Equation (10). Evaluation is zero-shot, with batch size 16, no chat template, and the full evaluation set of each task. The seven tasks are LAMBADA (Paperno et al., 2016), PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2020), ARC-Easy and ARC-Challenge (Clark et al., 2018), and SciQ (Welbl et al., 2017). We use `acc_norm` for HellaSwag and ARC-Challenge, and `acc` for the other tasks, all as percentages. AVG in Table 2 is the unweighted mean of the seven task means. Validation perplexity is $\exp(L_{\text{val}})$. Table 2 reports the mean and standard deviation of PPL across seeds. GPU-hours follow the timing protocol in Appendix B.3.

B.3 Cost measurement

GPU-hours are the wall-clock training time in hours multiplied by the number of GPU processes. Timing starts after checkpoint restoration and includes in-loop compilation, evaluation, logging, and checkpointing; setup and the final evaluation, checkpoint saving, and upload are excluded.

All scales use H800 devices: Small and Medium use eight GPU processes, and Large uses 32. All methods at a given scale are trained and timed on the same hardware configuration and software stack, with the same PyTorch and `torch.compile` implementation and no custom kernels.

Figure 3b plots a 500-update trailing mean of training loss against GPU-hours, with each run on its own learning-rate schedule.

C Projection Measurements

Models and data. We evaluate six pretrained decoders (Table 5): Qwen2.5 (Qwen Team, 2024), Qwen3 (Yang et al., 2025), Llama 3.1 (Meta, 2024), Mistral-7B-v0.3 (Mistral AI, 2024), and Falcon3 (TII Team, 2024). They process the same 256 deduplicated documents, taken in source order from the `sample-10BT/train` split of FineWeb-Edu (Penedo et al., 2024). Each model tokenizes this sample and truncates each document to at most 512 tokens. Forward passes use BF16; projections and statistics use FP32. We exclude special tokens and the first position of each document; including the first position changes no coefficient by more than 0.2.

Measurement. For each valid token, we compute the projection coefficient α (Equation (2)) at the input of the first block and after every block, before the final normalization. The reference is the token's original lookup embedding; none of the six models rescales it before the first block. Pointwise 95% confidence intervals use 500 document-bootstrap resamples.

Mean removal and control. In our samples, embeddings share a common mean direction, which can inflate projections onto both the token's own embedding and other embeddings. Figure 1 subtracts the frequency-weighted sample mean embedding from each reference vector. Let μ be the frequency-weighted mean embedding of the sample and $\bar{e}_t = e_t - \mu$. The plotted coefficient is

$$\bar{\alpha}_t^k = \frac{\langle h_t^k, \bar{e}_t \rangle}{\|\bar{e}_t\|^2} = \frac{\|e_t\|^2 \alpha_t^k - \langle h_t^k, \mu \rangle}{\|e_t - \mu\|^2}. \tag{11}$$

Table 5: Pretrained checkpoints and evaluation sizes for the projection analysis. All models process the same 256 documents; valid-token counts depend on tokenization and exclude each document's first valid position.

Model identifier	Blocks	Valid tokens
`Qwen/Qwen2.5-7B`	28	104,863
`Qwen/Qwen2.5-32B`	64	104,863
`Qwen/Qwen3-8B-Base`	36	104,863
`meta-llama/Llama-3.1-8B`	32	103,858
`mistralai/Mistral-7B-v0.3`	32	108,485
`tiiuae/Falcon3-7B-Base`	28	105,157

Only the measurement reference changes; the residual state and model forward pass are unchanged. The raw and mean-removed coefficients are not numerically identical: the raw coefficient starts exactly at one, whereas the mean-removed coefficient averages 0.999–1.013 at the input across the six models. Equation (2) and all EER quantities in Section 3 use the raw, uncentered embedding.

The control draws $K = 8$ other token IDs per token from the empirical token distribution of the 256-document sample and averages their projection coefficients, using the same mean removal as for the own-embedding reference. Both raw and mean-removed measurements support the same qualitative finding: the own-minus-control difference is positive, and its paired 95% bootstrap interval excludes zero at every block of every model (Figure 5). This difference changes by less than 1% across four independent control draws. Without mean removal (Figure 6), the control stays near zero in five models; in Qwen3-8B-Base, the shared mean direction raises it to about 57% of the own-embedding coefficient at the final block. Mean removal reduces this ratio to about 10% (2.22 versus 22.82). Thus the shared mean affects the magnitude of the control but does not account for the stronger projection onto each token's own embedding.

D Additional Results

D.1 Combinations at Medium scale

Table 6 complements the Large results in Table 2: EER also lowers validation loss on Medium SATFormer.

Table 6: Medium backbone combinations at 21.81B tokens. Δ_{EER} is the reduction in validation loss from adding EER to the backbone above it.

Configuration	Val. loss ↓	Δ_{EER} ↑	GPU-h ↓
SATFormer	2.566	–	86.37
SATFormer + EER	2.554	0.0116	87.81

D.2 Insertion protocol

Inserting after every kth MLP places modules in blocks $k, 2k, \ldots, 16$, giving $16/k$ modules and $24{,}576/k$ added parameters; $k = 16$ is the final MLP only. Each placement is trained separately.

D.3 Coefficient rules at Small scale

The coefficient-rule study uses the Small setup (Appendix B) with one injection site after every MLP update. Let r_t^ℓ be the state after the MLP update of block ℓ and λ_t^ℓ the direct coefficient before it, with $\lambda_t^1 = 1$. We compare two update rules:

$$\begin{aligned} \text{Add:} \quad & \tilde{r}_t^\ell = r_t^\ell + g_t^\ell e_t, && \lambda_t^{\ell+1} = \lambda_t^\ell + g_t^\ell, \\ \text{Total:} \quad & \tilde{r}_t^\ell = r_t^\ell + (\tau_t^\ell - \lambda_t^\ell) e_t, && \lambda_t^{\ell+1} = \tau_t^\ell. \end{aligned} \tag{12}$$

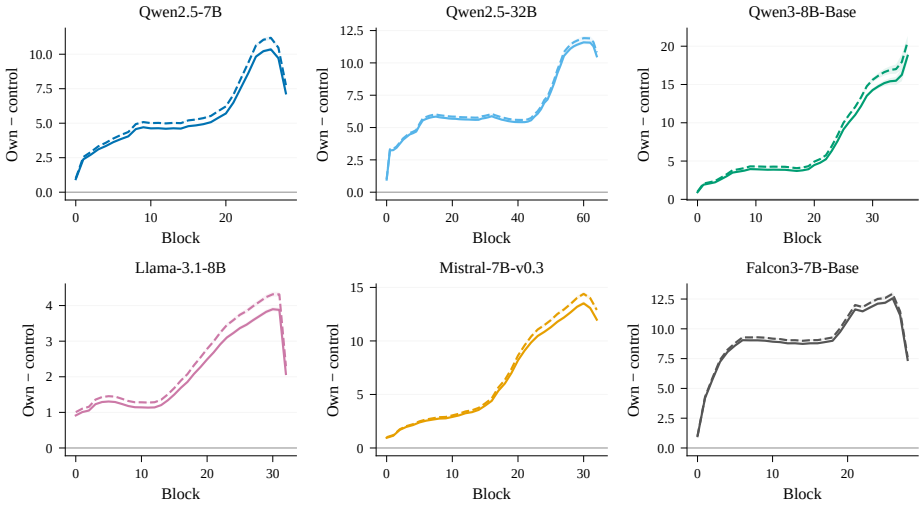

Figure 5: **Stronger projection onto each token's own embedding persists after mean removal.** Each curve shows the projection onto the token's own embedding minus the average projection onto other sampled token embeddings. Solid lines use the original embeddings; dashed lines subtract the shared mean from the reference embeddings. Shading indicates 95% confidence intervals.

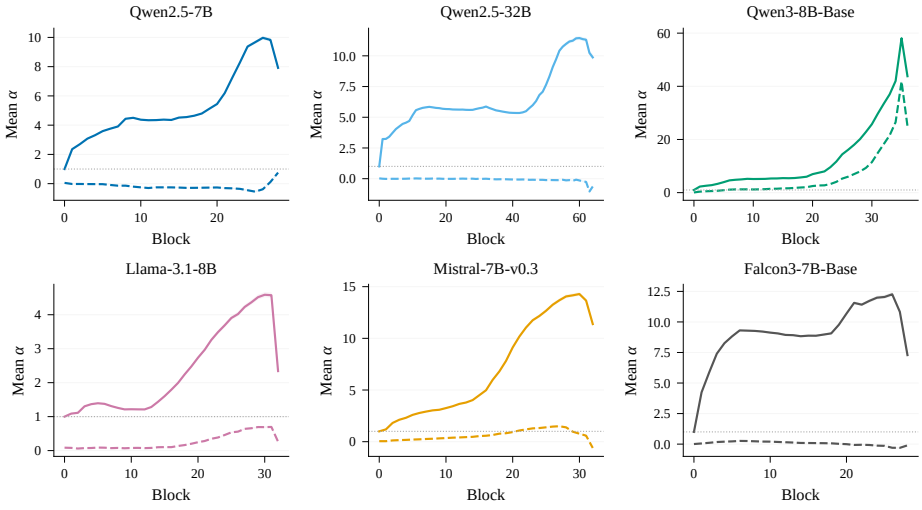

Figure 6: **Projections onto original embeddings, before mean removal.** Solid lines show projections onto each token's own embedding; dashed lines show the average projection onto other sampled token embeddings. The elevated control curve in Qwen3-8B-Base illustrates the contribution of the shared mean direction. Shading indicates 95% confidence intervals.

Add, the default, predicts an increment g_t^ℓ. Total predicts a target τ_t^ℓ for the direct coefficient and can therefore lower it below an earlier value. Total changes only the direct coefficient, not embedding components carried by sublayer outputs.

The increment or target is produced by one of the following:

- **Fixed ($g = 1$):** a constant coefficient of one.
- **Learned per layer:** one unconstrained scalar per site, initialized at EER's initial coefficient $\mathrm{softplus}(-3) \approx 0.0486$ and trained without weight decay (16 parameters). This is the additive form of the input skip in Jordan et al. (2024), without rescaling the residual stream (Appendix D.5).
- **Token-only controller:** Equation (3) applied to the embedding e_t instead of the state r_t^ℓ, with the same parameter count as EER.
- **State-dependent (EER):** Equation (3) applied to r_t^ℓ (24,576 parameters, weight decay 0.1).

Table 3a reports means and standard deviations for the main Add-rule comparison: the fixed coefficient one, the per-layer scalar, the token-only controller, and EER. In a separate Add/Total comparison, the state-dependent controller reaches 2.766 under either rule.

D.4 Large-model design variants

Table 7 compares default EER with two variants at Large scale. Inserting after both attention and MLP updates costs more without lowering loss further. Predicting the total direct coefficient also improves on Baseline (2.440), and its loss is close to the default.

Table 7: Large-model variants. Δ is the reduction in validation loss relative to Baseline (2.440).

Configuration	Val. loss ↓	Δ ↑	GPU-h
EER after MLP (default)	2.415	0.0249	322.36
EER after attention and MLP	2.420	0.0196	336.19
Total-coefficient controller after MLP	2.417	0.0231	323.45

D.5 Comparison with the modded-nanoGPT input skip

The modded-nanoGPT repository (Jordan et al., 2024) contains an input skip that was introduced in a community training speedrun, where architectural changes are kept when they reduce training time. It is similar to EER in one respect: every block re-adds the input embedding to the residual stream. Its coefficients, however, are static per block, and it also rescales the residual state. We test this form at Small scale (Appendix B), replacing Equation (4) at each site with

$$\tilde{r}_t^\ell = a_\ell \, r_t^\ell + b_\ell \, e_t, \tag{13}$$

where a_ℓ and b_ℓ are learned scalars shared by all tokens. Unlike the original implementation, which normalizes the embedding, we use the same e_t as EER. Modules follow every MLP update, as in Table 3a.

Table 8: Small-scale comparison with the modded-nanoGPT input skip. Baseline, the per-layer scalar, and EER are from Table 3a.

Configuration	Val. loss ↓
Baseline	2.789
Learned per layer ($a_\ell = 1$)	2.780
modded-nanoGPT style (a_ℓ, b_ℓ learned)	2.810
EER	2.764

The modded-nanoGPT-style variant raises validation loss above Baseline by 0.021 nats, whereas fixing $a_\ell = 1$ lowers it by 0.009 nats. We therefore attribute the degradation to rescaling the residual stream. With $a_\ell \neq 1$, the identity shortcut no longer carries earlier states with unit coefficient: every component written before site ℓ is multiplied by a_ℓ, which changes its scale relative to all later sublayer updates. EER leaves the updated state unchanged and adds only along e_t (Section 3.1).