On the Design of Qwen3.8-Next Architecture: Evaluation, Efficiency, and Training Stability

Qwen Team

Abstract

We describe the architecture and ablations of **Qwen3.8-Flash-Next**, a sparse mixture-of-experts model with 125B parameters, 6B activated per token, and additional 51B parameters of n-gram embedding tables held off the accelerator. On fourteen pre-training benchmarks the model leads the 397B-A17B predecessor on eight and trails it on the rest by at most 2.6 points, at **1/3** the activated parameters, **1/3** the training tokens, and roughly **1/9** the training FLOPs. Token mixing uses a layer-wise hybrid of **Gated DeltaNet** (**GDN**) and global attention, with one full-attention layer in every four; at continued-pretraining time those full-attention layers are replaced by **Qwen Sparse Attention** (**QSA**), which scores context at micro-block granularity with a compressed lightweight indexer. The residual stream is widened to four branches and read through an elementwise gate, a design we call the **Gated Residual** (**GR**). Capacity is added outside the backbone by a single **n-gram embedding layer** whose tables are prefetched from host memory. We evaluate every candidate change along three axes: loss together with downstream benchmarks; the cost of the change in training, prefill and decode; and its effect on the optimal hyperparameters and training stability. Loss and downstream accuracy do not always move together: enlarging the n-gram vocabulary lowers loss monotonically while downstream accuracy saturates. The architecture and the **Muon** optimizer together shift the optimal learning rate and batch size upwards, render batch-size warmup unnecessary, and substantially improve stability under stress tests. Loss, benchmarks, efficiency and stability form one design problem. Solved jointly, they yield a recipe that is simultaneously *more efficient, more capable and more stable*.

1 Introduction

Qwen3.8-Flash-Next is a sparse mixture-of-experts model with 125B total parameters, 6B activated per token, and additional 51B parameters of n-gram embedding tables held off the accelerator. The design goal is to retain the quality of the previous generation's 397B-A17B flagship (Qwen Team, 2026) at a fraction of its compute budget. On fourteen pre-training benchmarks spanning knowledge, STEM, reasoning, coding and multilingual ability, the resulting base model leads that predecessor on eight and trails it on the remaining six by at most 2.6 points (Tab. 11), while activating roughly a third as many parameters per token and training on roughly a third as many tokens, for about a ninth of the training FLOPs.

An architectural change touches three things at once: what the model can do on downstream tasks, what it costs to train and to serve, and whether the training run remains optimal and stable at scale. We therefore evaluate every candidate change along three axes: loss together with downstream benchmarks; the cost of the change in training, prefill and decode; and its effect on the optimal hyperparameters and on training stability. Loss, benchmarks, efficiency and stability form one design problem, and we report each on its own terms. Throughout this report, we highlight where the three axes disagree and the design choices those disagreements led to.

Four architectural components carry the design, each addressing a distinct bottleneck. Token mixing uses a layer-wise hybrid of Gated DeltaNet (GDN) (Yang et al., 2024) and global attention: the recurrent layers compress the prefix into a fixed-size state at linear cost, while one full-attention layer in every four retains the direct token-level retrieval that no finite-state memory reproduces exactly (§2.1.1). At continued-pretraining time those full-attention layers are replaced by Qwen Sparse Attention (QSA), which follows the sparse-attention route of Liu et al. (2025a) but scores context at micro-block granularity with a compressed lightweight indexer, so that the cost of indexing itself falls with sequence length. The residual stream is widened to four branches (Baykal et al., 2023; Zhu et al., 2024) and read through an elementwise gate, a design we call Gated Residual (GR) (§2.2): widening adds capacity to the residual path, and the gate decides how that capacity is spent, while also supplying the rescaling that keeps training stable. Capacity is further added outside the backbone by a single n-gram embedding layer (Google DeepMind, 2025; Cheng et al., 2026) whose tables are prefetched from host memory (§2.3), scaling parameter count with negligible additional per-token FLOPs and latency.

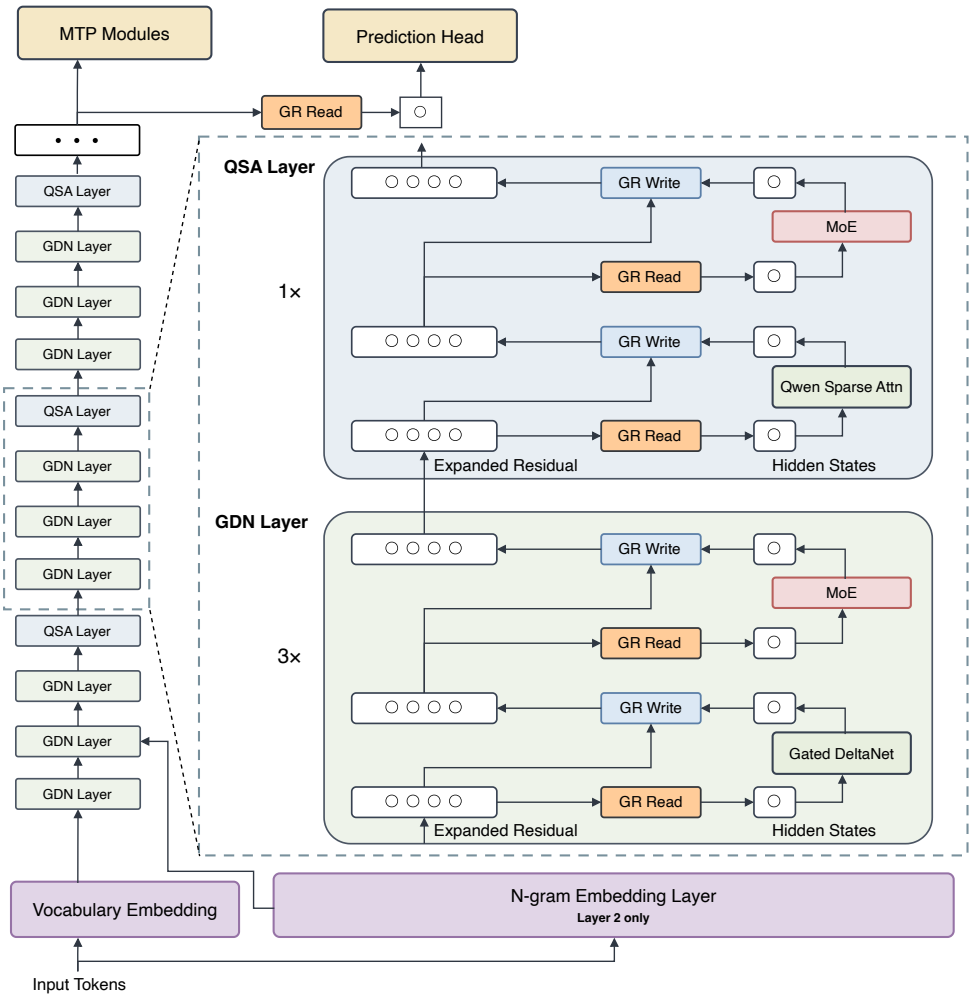


Figure 1: **Qwen3.8-Flash-Next architecture.** Token mixing alternates three GDN layers with one QSA layer per block of four. Every sublayer reads and writes through GR, which widens the residual stream and gates the read elementwise. An n-gram embedding layer at Layer 2 scales capacity off the accelerator via host-memory prefetching. The MTP module reuses QSA indices across speculative decoding steps.

Evaluation. Loss and downstream accuracy do not always move together, and we observe disagreements in both directions. Enlarging the n-gram vocabulary lowers loss monotonically while downstream accuracy saturates, and under a fixed parameter budget the loss optimum diverges from the accuracy optimum (Tab. 9, Tab. 8). Conversely, predicting the residual read and write weights from the residual state yields only a marginal loss reduction but a clear benchmark gain (§2.2). Other disagreements surface late: restricting each block to the two highest-gated residual branches is almost free in pre-training loss yet degrades with further training (§2.2), and removing positional encoding from the full-attention layers is indistinguishable during pre-training but affects generation quality at later stages (§2.1.1). In the sections that follow, where loss and benchmarks move together we report loss alone, for brevity.

Efficiency. We evaluate cost separately in training, prefill and decode. **In training**, FlashQLA achieves a 2–3$\times$ forward and roughly 2$\times$ backward speedup over the Triton baseline on GPUs (§2.1.1). Muon introduces its own engineering costs: its per-parameter FLOPs depend on matrix shape rather than parameter count, so the data-parallel gradient buffer is repartitioned by estimated orthogonalization cost; and its step fragments into many small kernels once fused parameters are split, so the step is captured in a CUDA graph (§3.1). We set the Newton–Schulz iteration to 8 steps, favouring the additional stability under stress. At inference, **prefill** is dominated by attention over the whole context, which QSA addresses by compressing the key sequence by a factor r, reducing indexer cost from $O(n^2)$ to $O(n^2/r)$. **Decode** is dominated by memory traffic, which is why the GDN layers keep a fixed-size recurrent state, the GR drops the branch-mixing operator H_{res}, and the residual state supports FP8 storage. At a context length of 1M, QSA is 7.6$\times$ faster than dense attention in prefill and 4.9$\times$ faster in decode at the kernel level.

Optimization. Muon (Jordan et al., 2024) is applied to the two-dimensional weights that act as linear maps; the input and n-gram embeddings, output head, MoE router and the low-rank projections of GR stay

on AdamW, where orthogonalization is either impractical or unhelpful (§3.1). Fused parameters are split before orthogonalization, since orthogonalizing a concatenated matrix mixes singular directions across unrelated sub-blocks (§3.1). The new architecture and optimizer also shift the optimal hyperparameters, so we refit the scaling law (Kaplan et al., 2020) used for the Qwen3.5 series (§3.2). The new scaling law predicts a larger batch size and learning rate; both predictions are verified separately and confirmed. The larger batch size improves parallel throughput at scale, and the larger learning rate improves convergence. Ramping the batch size over early training ends no better than starting at the target and costs 18.8% more optimizer steps, so we do not use it (§3.2).

Training Stability. We verify training stability under **stress tests** that reproduce large-scale instabilities at moderate model scale by raising the learning rate (Wortsman et al., 2023) and keeping a constant learning rate. *The criterion is that the new recipe must be at least as stable as the generation it replaces under equal stress.* At four times the optimal learning rate, the previous structure spikes frequently, whereas the new recipe remains stable throughout (§3.3). Isolating the gate in GR on a single-variable pair confirms it as a key contributor to the stability margin over the Qwen3.5 architecture. As a direct result of these stability enhancements, the full-scale training of Qwen3.8-Flash-Next proceeded smoothly without a single loss spike or anomalous fluctuation in gradient norms, without relying on explicit clipping methods such as qk-clip (Kimi Team, 2025) or SwiGLU-clip (Agarwal et al., 2025).

Organization. §2.1.1 and §2.1.2 describe token mixing. §2.2 covers the gated residual, §2.3 the n-gram embedding layer, §3.1 the optimizer, §3.2 the hyperparameter scaling and §3.3 the stability stress test. Evaluation of the resulting base and post-trained models follows.

2 Model Architecture

2.1 Attention

2.1.1 GDN Hybrid Architecture

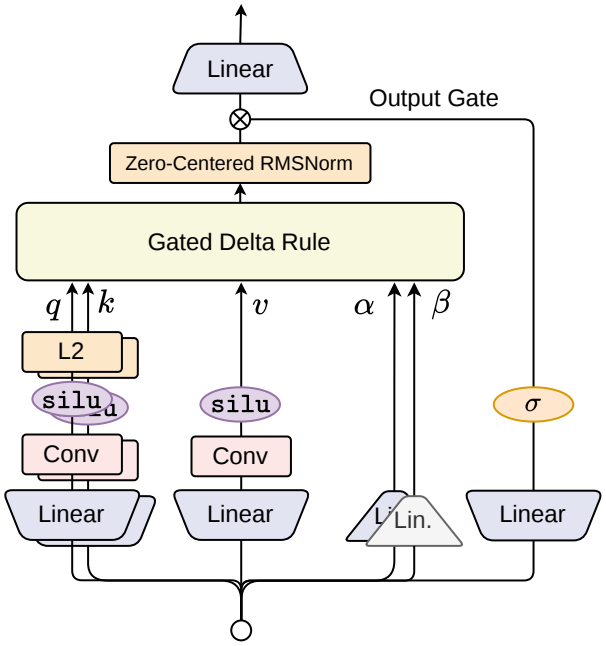


Figure 2: The Gated DeltaNet token mixer. The projected query, key, and value streams pass through short causal convolutions; queries and keys are L2-normalized before the gated delta recurrence. The decay gate α_t and write gate β_t control the recurrent update, while a sigmoid output gate modulates the zero-centered RMS-normalized output.

Motivation. Full self-attention provides direct content-based access to every preceding token, but its token-mixing cost grows quadratically with sequence length and its key–value (KV) cache grows linearly during autoregressive generation (Vaswani et al., 2017). Sliding-window attention (SWA) replaces global access with a bounded local receptive field, reducing both computation and cache consumption. However,

information outside the window can only propagate indirectly through depth. This creates a tension between efficient local processing and persistent content-dependent memory.

We address this tension with a layer-wise hybrid of Gated DeltaNet (GDN) (Yang et al., 2024) and global attention. GDN compresses the prefix into a fixed-size recurrent state and updates that state according to the current content, while the interleaved global-attention layers retain direct token-level retrieval that is difficult for any finite-state recurrent memory to reproduce exactly.

This design choice is consistent with the architecture ablation reported in Tab. 1. Relative to the full-attention Transformer baseline, the GDN-hybrid model improves 8 of the 9 selected benchmarks. Relative to the SWA-hybrid, it is stronger on seven of the nine benchmarks. These results motivate a hybrid design, but they do not by themselves isolate which architectural component causes each improvement.

Gated Delta Recurrence. Linear attention can be interpreted as a fast-weight memory that stores key–value associations in a matrix state (Schlag et al., 2021). For each head, let $\boldsymbol{q}_t, \boldsymbol{k}_t \in \mathbb{R}^{d_k}$ and $\boldsymbol{v}_t \in \mathbb{R}^{d_v}$. Following the implementation convention, GDN maintains a state $\boldsymbol{S}_t \in \mathbb{R}^{d_k \times d_v}$, which is the transpose of the state convention used in the original formulation, and applies the gated delta rule (Yang et al., 2024):

$$\widetilde{\boldsymbol{S}}_{t-1} = \alpha_t \boldsymbol{S}_{t-1}, \tag{1}$$

$$\boldsymbol{e}_t = \boldsymbol{v}_t - \widetilde{\boldsymbol{S}}_{t-1}^{\top} \boldsymbol{k}_t, \tag{2}$$

$$\boldsymbol{S}_t = \widetilde{\boldsymbol{S}}_{t-1} + \beta_t \boldsymbol{k}_t \boldsymbol{e}_t^{\top}, \tag{3}$$

$$\boldsymbol{y}_t = \boldsymbol{S}_t^{\top} \boldsymbol{q}_t, \tag{4}$$

where $\alpha_t \in (0, 1)$ is a data-dependent decay and $\beta_t \in (0, 1)$ controls the delta update. Equivalently,

$$\boldsymbol{S}_t = \alpha_t \left(\boldsymbol{I} - \beta_t \boldsymbol{k}_t \boldsymbol{k}_t^{\top} \right) \boldsymbol{S}_{t-1} + \beta_t \boldsymbol{k}_t \boldsymbol{v}_t^{\top}. \tag{5}$$

The two gates play complementary roles. The decay α_t globally controls the lifetime of the existing state, whereas the delta term first estimates the value already associated with $\boldsymbol{k}_t$ and writes only the residual error. Consequently, repeated or similar keys update an existing association instead of accumulating unbounded outer products. This targeted erase-and-write operation distinguishes GDN from purely additive linear attention.

GDN Parameterization. Given the normalized residual-stream input $\boldsymbol{x}_t \in \mathbb{R}^d$, GDN computes content features using learned projections followed by a short depthwise causal convolution:

$$\boldsymbol{q}_t = \mathrm{L2Norm}\left(\mathrm{SiLU}\left(\mathrm{ShortConv}(\boldsymbol{W}_q \boldsymbol{x}_t)\right)\right), \tag{6}$$

$$\boldsymbol{k}_t = \mathrm{L2Norm}\left(\mathrm{SiLU}\left(\mathrm{ShortConv}(\boldsymbol{W}_k \boldsymbol{x}_t)\right)\right), \tag{7}$$

$$\boldsymbol{v}_t = \mathrm{SiLU}\left(\mathrm{ShortConv}(\boldsymbol{W}_v \boldsymbol{x}_t)\right). \tag{8}$$

Short convolution supplies an explicit local inductive bias before information is compressed into the recurrent state. L2 normalization bounds the magnitudes of $\boldsymbol{q}/\boldsymbol{k}$ and stabilizes the rank-one delta transition.

The write strength and decay are parameterized as

$$\beta_t = \sigma(\boldsymbol{W}_\beta \boldsymbol{x}_t), \tag{9}$$

$$\alpha_t = \exp\left[-\exp(\boldsymbol{A})\,\mathrm{softplus}(\boldsymbol{W}_\alpha \boldsymbol{x}_t + \boldsymbol{b}_\alpha)\right]. \tag{10}$$

After applying the recurrence independently across heads, the head outputs are normalized and modulated by an input-dependent output gate:

$$\boldsymbol{o}_t = \boldsymbol{W}_o \left[\sigma(\boldsymbol{W}_z \boldsymbol{x}_t) \odot \mathrm{RMSNorm}(\boldsymbol{y}_t)\right]. \tag{11}$$

Unlike the original GDN, which uses a SiLU output gate, we use the bounded sigmoid gate in Equation (11) and observe consistent improvements across our experiments. Following Qwen3-Next, we adopt zero-centered RMSNorm to constrain the growth of RMSNorm weights. The same formulation is consistently applied to all other RMSNorm layers used throughout the model. Figure 2 summarizes the complete GDN token-mixing path.

At the model level, our hybrid architecture retains rotary position embeddings (RoPE) (Su et al., 2024) in its full-attention layers. RoPE and a NoPE variant without positional encoding show little difference during pretraining, but *the NoPE variant exhibits a substantially higher rate of endless generation after post-training and is therefore more likely to fail to terminate.* We place one such full-attention layer in every four layers, with the other three layers using GDN. This schedule strikes a favorable balance between efficiency and quality, while periodic full attention is particularly important for long-context performance.

Table 1: Architecture comparison. All values are percentages and higher is better. EvalPlus and MultiPL-E are reported with pass@1-style aggregate fields. Avg. is the unweighted arithmetic mean across the nine benchmarks. Bold denotes the best result in each column.

	Knowledge			STEM		Reasoning	Multilingual	Code		
Architecture	MMLU	MMLU-Pro	SuperGPQA	MATH	GSM8K	BBH	MMMLU	EvalPlus	MultiPL-E	Avg.
Full attention	62.65	37.59	21.76	49.40	75.13	63.78	47.74	51.01	39.73	49.87
SWA hybrid	**66.30**	40.67	22.45	45.48	74.22	65.88	51.33	**52.12**	41.93	51.15
GDN hybrid	66.26	**42.82**	**23.45**	**53.98**	**77.07**	**68.72**	**54.83**	49.71	**47.48**	**53.81**

Kernel Efficiency. For efficiency, we optimize the GDN kernel with FlashQLA, a TileLang-based fused linear-attention kernel library. Across multiple settings on NVIDIA GPUs, FlashQLA achieves a 2–3× forward speedup and an approximately 2× backward speedup over the FLA Triton kernel (Yang & Zhang, 2024). The implementation and benchmarks are available at `https://github.com/QwenLM/FlashQLA`.

Architecture Ablation. We compare three checkpoints evaluated by the same evaluation pipeline: a full-attention Transformer, an SWA hybrid, and the GDN hybrid. Both hybrid variants use one full-attention layer in every four layers; the remaining token-mixing layers use SWA or GDN, respectively, with a window size of 128 for the SWA layers. Each checkpoint corresponds to a 28-layer 25B-A3B MoE model based on the Qwen3.5 architecture, pretrained first on 400B tokens with a 4K context length, and subsequently on 80B tokens with a 32K context length. We report knowledge results on MMLU (Hendrycks et al., 2021a), MMLU-Pro (Wang et al., 2024), and SuperGPQA (Du et al., 2025); STEM results on MATH (Hendrycks et al., 2021b) and GSM8K (Cobbe et al., 2021); reasoning results on BBH (Suzgun et al., 2023); multilingual results on MMMLU (OpenAI, 2024); and code results on EvalPlus and on MultiPL-E (Cassano et al., 2023). The results are reported in Table 1.

The GDN hybrid improves over the Transformer on eight of the nine selected benchmarks and exceeds the SWA hybrid on seven. It achieves the best result on seven benchmarks and the highest overall average, while the SWA hybrid is marginally higher on MMLU and stronger on EvalPlus.

2.1.2 Qwen Sparse Attention

Motivation. By selectively attending to important context, sparse attention alleviates the quadratic computational bottleneck of softmax attention in long-context scenarios. Recently, DSA (Liu et al., 2025a) has achieved considerable inference speedups using a lightweight indexer to generate token-level sparse masks. However, as sequence length increases, the overhead of its $O(n^2)$ indexer remains non-negligible.

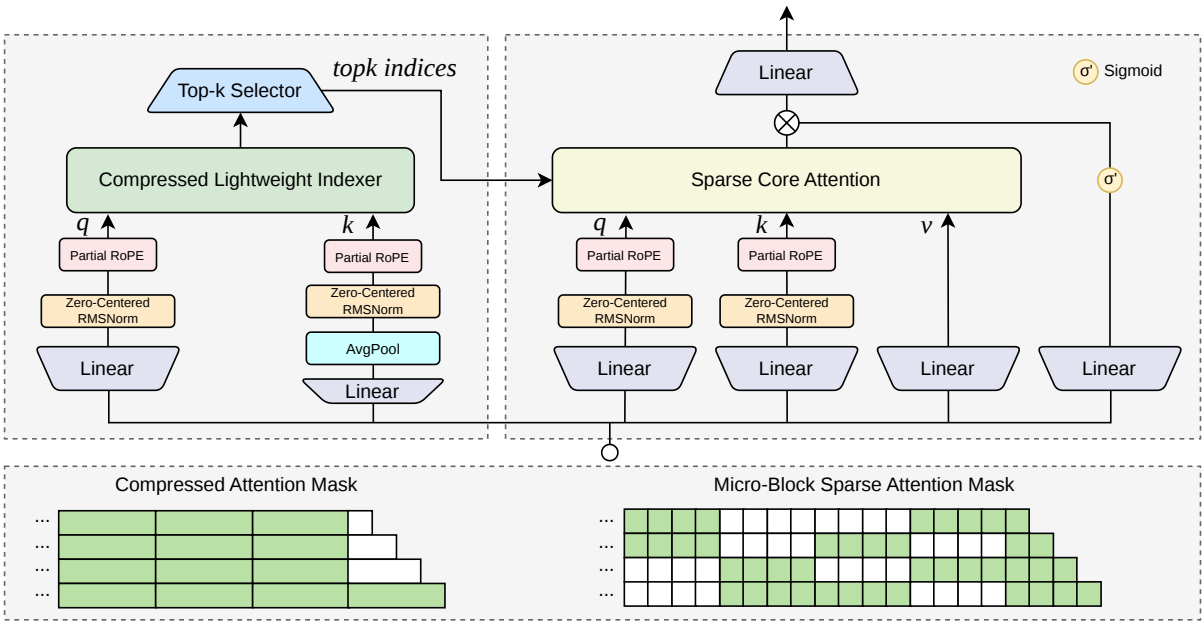


Figure 3: Overview of Qwen Sparse Attention (QSA). The QSA indexer (left) uses a compressed causal attention mask to score key blocks and select the top-k indices. These indices are expanded into a micro-block sparse attention mask for sparse core attention (right).

To reduce this indexing cost, Qwen3.8-Flash-Next adopts Qwen Sparse Attention (QSA). Specifically, QSA employs a lightweight indexer that compresses the sequence into micro-block representations, estimates their importance, and selects the most relevant context for attention computation. This design reduces indexing overhead on long inputs while balancing task performance and inference efficiency. Compared with methods that share indices across layers, the within-layer sequence compression in QSA relies less on cross-layer similarity, making it naturally suited to hybrid architectures.

Figure 3 illustrates the QSA architecture. A compressed lightweight indexer first estimates context importance at micro-block granularity and then guides sparse computation in the core attention module.

Compressed Lightweight Indexer. Given an input hidden state $\mathbf{x}_i$, the indexer adopts an MQA (Shazeer, 2019) with H query heads and one shared key head. It first applies independent lightweight projections:

$$\widehat{\mathbf{q}}_i^h = \mathrm{RMSNorm}(\mathbf{W}_Q^h \mathbf{x}_i), \qquad \mathbf{k}_i = \mathbf{W}_K \mathbf{x}_i. \tag{12}$$

To reduce the sequence length, keys are partitioned into non-overlapping blocks of r tokens and compressed by average pooling. Denoting the starting position of block b by $p_b = b \cdot r$, the corresponding compressed key is

$$\widehat{\mathbf{k}}_b = \mathrm{RMSNorm}\left(\mathrm{AvgPool}(\mathbf{k}_{p_b:p_b+r-1})\right), \qquad 0 \le b < \left\lfloor \frac{n}{r} \right\rfloor. \tag{13}$$

For positional encoding, the indexer applies partial RoPE. Specifically, partial RoPE is applied to 64 of the 128 dimensions in each indexer head, matching the rotary dimension used in the core attention module. As shown in Eq. (13), key compression is performed before positional encoding. Each block is therefore first summarized into a content representation and then assigned a single block-level position. This ordering avoids averaging token representations with different rotary phases. Each query retains its token position i, whereas each compressed key is assigned the starting position p_b of its block:

$$\mathbf{q}_i^h = \mathrm{PRoPE}(\widehat{\mathbf{q}}_i^h, i), \qquad \bar{\mathbf{k}}_b = \mathrm{PRoPE}(\widehat{\mathbf{k}}_b, p_b). \tag{14}$$

Block-level importance scores $\mathbf{I}$ are then obtained through block-causal scoring. For query token i and compressed block b, ReLU-activated query–key similarities are summed over all indexer heads:

$$I_{ib} = \begin{cases} \displaystyle\sum_{h=1}^{H} \mathrm{ReLU}\left(\left\langle \mathbf{q}_i^h, \bar{\mathbf{k}}_b \right\rangle\right), & p_b + r - 1 \le i, \\ -\infty, & \text{otherwise}, \end{cases} \tag{15}$$

This block-causal condition allows each query to score only blocks that have been fully observed. Given a token budget K, each query selects the highest-scoring compressed blocks. Since each block contains r tokens, the block budget is $K_B = \lceil K/r \rceil$:

$$\mathcal{B}_i = \mathrm{TopK}_{K_B}\left(\{I_{ib}\}_b\right), \qquad K_B = \left\lceil \frac{K}{r} \right\rceil. \tag{16}$$

Selected blocks are then expanded to their original token indices and truncated to the budget K. Together with the tokens in the final incomplete block, which are always included, they form the final set used for core attention computation.

Training Details. QSA is introduced during the continued pretraining (CPT) stage of Qwen3.8-Flash-Next with a sequence length of 256K tokens. The training procedure consists of two stages: dense distillation and sparse training.

- **Stage 1: Dense Distillation.** We first distill the full-sequence attention distribution of the backbone into the indexer. The token-level teacher distribution is obtained by summing the softmax attention distributions over all teacher heads and applying L_1 normalization. Denoting the resulting probability from query i to token j by a_{ij}, the full token-level distribution is $\mathbf{a}_i \in \mathbb{R}^n$. Following prior work (Gao et al., 2024; Wang et al., 2026b), we apply max pooling to align this distribution with the block-level indexer scores, thereby preserving salient token-level signals that could otherwise be diluted during aggregation:

$$\bar{a}_{ib} = \mathrm{MaxPool}(\mathbf{a}_{i,\, p_b:p_b+r-1}), \qquad \hat{\mathbf{a}}_i = \frac{\bar{\mathbf{a}}_i}{\|\bar{\mathbf{a}}_i\|_1}. \tag{17}$$

  Letting $B = \lfloor n/r \rfloor$, the pooled teacher distributions $\hat{\mathbf{a}}_i \in \mathbb{R}^B$ and the indexer scores $\mathbf{I}_i \in \mathbb{R}^B$ share the same block dimension. The normalized teacher scores are distilled into the indexer by minimizing

$$\mathcal{L}_{\mathrm{KL}} = \frac{1}{N} \sum_i D_{\mathrm{KL}}\left(\hat{\mathbf{a}}_{i,:} \,\middle\|\, \mathrm{Softmax}(\mathbf{I}_{i,:})\right), \tag{18}$$

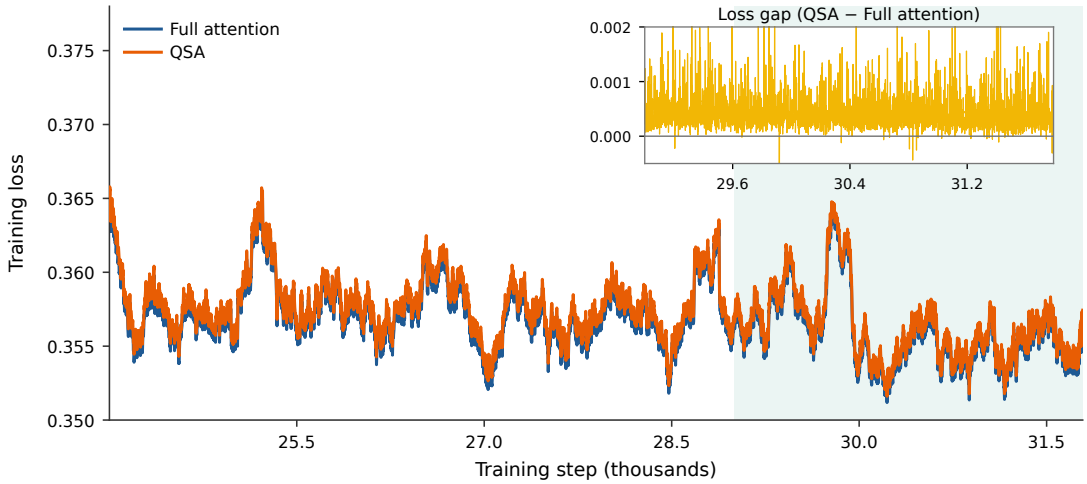


Figure 4: Training LM loss with and without QSA. Curves are smoothed with a 200-step moving average. The shaded region marks the final stage of continued pretraining, and the inset shows the per-step loss difference between QSA and the full-attention baseline in this region.

where N is the number of query tokens. Only complete key blocks are included in the KL loss for each query. During the warm-up stage, only the indexer is trained for 1,000 steps, using a learning rate of 1×10^{-3}. Each step comprises 8 sequences of 256K tokens, amounting to approximately 2B training tokens in total.

- **Stage 2: Sparse Training.** After indexer initialization, the entire backbone is trained under the guidance of the indexer to adapt to sparse attention patterns. Following Eq. (16), QSA first selects the top-K_B blocks. These blocks are expanded to token indices and combined with the tail tokens in the final incomplete block:

$$\mathcal{S}_i = \mathrm{Expand}(\mathcal{B}_i) \cup \left\{ r \left\lfloor \frac{i+1}{r} \right\rfloor, \ldots, i \right\}. \tag{19}$$

The Expand operator maps the selected blocks to their token indices. The resulting set $\mathcal{S}_i$ is used for core attention computation. In this stage, the indexer KL loss is computed only over the top-K_B blocks in $\mathcal{B}_i$. Before evaluating the KL divergence, the teacher probabilities within $\mathcal{B}_i$ are renormalized to sum to one:

$$\mathcal{L}_{\mathrm{KL}} = \frac{1}{N} \sum_i D_{\mathrm{KL}} \left(\hat{\mathbf{a}}_{i,\mathcal{B}_i} \,\|\, \mathrm{Softmax}(\mathbf{I}_{i,\mathcal{B}_i}) \right). \tag{20}$$

During the final stage of CPT, QSA is enabled and the backbone and indexer are jointly trained for 8,000 steps using a learning rate of 2.5×10^{-5}. Each step comprises 96 sequences of 256K tokens, totaling roughly 200B training tokens.

Implementation Details. For Qwen3.8-Flash-Next, all full-attention layers in the backbone and MTP module are replaced with QSA to improve inference efficiency on long sequences. The compressed lightweight indexer adopts an MQA structure with four query heads and one shared key head, using the partial-RoPE configuration described above. With a token budget of $K = 2048$ and a compression ratio of $r = 4$, QSA selects up to 512 complete blocks for each query and additionally includes the tail tokens in the final incomplete block. Accurate importance estimates from the indexer enable the model to closely match the LM-loss trajectory of full attention during Stage 2 sparse training. As shown in Fig. 4, the two loss curves remain highly consistent, with the overall loss difference on the order of 10^{-4}.

For efficient training, we implement a fused QSA kernel that jointly computes sparse attention outputs and the KL loss without materializing intermediate results, substantially reducing memory consumption. The compressed indexer scores the sequence after compression, reducing both indexer computation and top-k selection overhead. In addition, multi-step MTP reuses top-k indices across prediction steps to further reduce draft-model inference costs.

Evaluation of QSA. We evaluate QSA by comparing Qwen3.8-Flash-Next equipped with QSA against its full-attention baseline. Table 2 reports the results across knowledge, STEM, reasoning, multilingual, and coding benchmarks, providing a broad assessment of whether sparse attention affects the general

Table 2: Model performance comparison between Qwen3.8-Flash-Next with full attention and QSA across widely used knowledge, STEM, reasoning, multilingual, and coding benchmarks. Bold values indicate the best result for each benchmark.

	Knowledge		STEM		Reasoning	Multilingual	Code		
Method	MMLU-Pro	SuperGPQA	MATH	GSM8K	BBH	MMMLU	EvalPlus	MultiPL-E	Avg.
Full Attn	72.9	51.7	69.8	91.0	90.4	**81.8**	70.8	78.4	75.9
w/ QSA	**73.7**	**52.1**	**71.6**	**92.2**	**91.6**	81.1	**72.3**	**79.8**	**76.8**

Table 3: Long-context retrieval performance of Qwen3.8-Flash-Next on RULER and 8-needle MRCR. RULER scores are averaged over sequence-length ranges. Bold values indicate the best result in each setting; Avg. is the macro-average across the two benchmarks.

	RULER				MRCR				
Method	≤128K	128–256K	256–512K	512K–1M	128K	256K	512K	1M	Avg.
Full Attn	99.84	**99.81**	97.65	90.08	**97.14**	**94.20**	30.66	20.71	78.76
w/ QSA	**99.89**	99.62	**98.95**	**93.00**	95.98	93.00	**40.53**	**26.44**	**80.93**

capabilities of the model beyond long-context retrieval. QSA matches or outperforms the full-attention baseline on seven of the eight benchmarks and improves the average score from 75.9 to 76.8. The remaining differences are small and do not indicate systematic degradation in any task category. Overall, these results show that QSA preserves the general capabilities of the model on short-context tasks while enabling more efficient long-context inference.

Beyond short-context benchmarks, we evaluate the retrieval capability of QSA on two widely used long-context benchmarks, RULER (Hsieh et al., 2024) and MRCR (OpenAI, 2025). We evaluate RULER at sequence lengths from 4K to 1000K and use the 8-needle MRCR setting at lengths from 128K to 1M, as reported in Table 3. QSA remains comparable to full attention at shorter lengths and performs better as the context grows. In particular, QSA improves the RULER score from 90.08 to 93.00 beyond 512K. On MRCR, the score increases from 30.66 to 40.53 at 512K and from 20.71 to 26.44 at 1M. These results suggest that QSA improves long-context inference efficiency without sacrificing task performance, while delivering further gains at longer sequence lengths.

Table 4: Mean MTP accepted length with full attention and QSA under four-step speculative decoding.

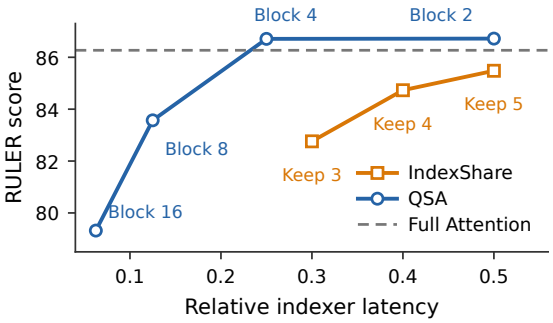
Method	MT-Bench	GSM8K	MATH	HumanEval	MBPP	Avg.
Full Attn	3.44	4.19	4.29	4.24	4.12	4.06
w/ QSA	3.47	4.20	4.30	4.26	4.13	4.07

We further examine the effect of QSA on the multi-token prediction (MTP) module. Beyond replacing the attention layers in MTP with QSA, we follow GLM (GLM-5-Team, 2026) and reuse the top-k indices across speculative decoding steps to further improve draft-model efficiency. To assess whether reusing QSA affects MTP performance, we conduct four-step speculative decoding experiments on benchmarks from different domains, as reported in Table 4. The results show no significant change in the mean accepted length after QSA reuse.

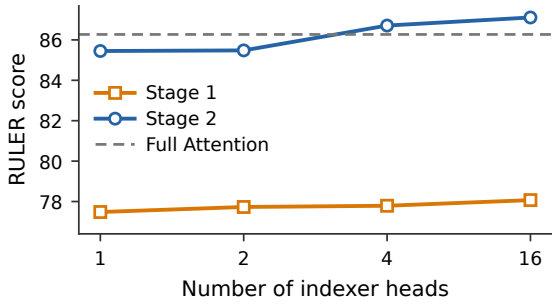
Architecture Ablation. We conduct architectural ablations of QSA at the 35B-A3B scale, focusing on the compression ratio and the number of indexer heads. As shown in Fig. 5, we use RULER (Hsieh et al., 2024) scores at sequence lengths up to 1M as the evaluation metric. All experiments are evaluated using the CPT models obtained after stage 2 sparse training.

For the compression-ratio ablation, we evaluate QSA with different micro-block sizes. We additionally include training-aware IndexShare (Bai et al., 2026) as a baseline, which reduces indexing overhead by uniformly sharing top-K indices across adjacent full-attention layers in the hybrid model. In Fig. 5(a), we report the RULER performance of both approaches against their estimated reductions in indexer latency. QSA matches the full-attention baseline at a relative indexer latency of 0.25, whereas IndexShare remains below the baseline at 0.5. For IndexShare, 0.5 denotes sharing a single index across two full-attention layers separated by three GDN layers. This result highlights the advantage of intra-layer compression in hybrid architectures, where cross-layer index sharing can be limited by low inter-layer similarity.

The number of query heads directly affects indexer efficiency, particularly during the prefill stage. We therefore evaluate MQA indexers with varying numbers of query heads on RULER, as shown in Fig. 5(b).



(a) Compression strategies.



(b) Indexer head configurations.

Figure 5: Architecture ablations of QSA on RULER. (a) QSA performance with different micro-block sizes; "Keep x" indicates the number of IndexShare indexer layers retained for computation. (b) Performance with different numbers of indexer query heads after dense distillation and sparse training.

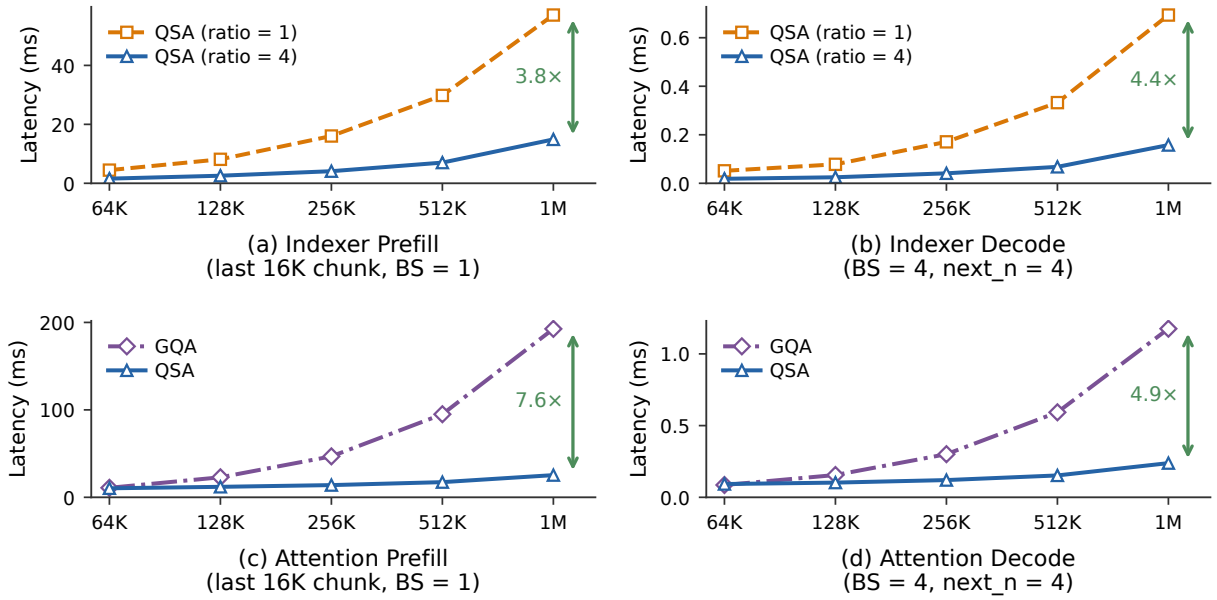


Figure 6: Kernel-level latency of QSA across context lengths during prefill and decode. Panels (a,b) compare indexer latency under different compression ratios, while panels (c,d) compare kernel-level attention latency between dense GQA and QSA, including both the indexer and sparse core attention. Chunked prefill uses a 16K-token chunk with batch size 1; decode uses batch size 4 and `next_n`$= 4$, corresponding to three MTP prediction steps. Arrows indicate speedups at a context length of 1M.

After dense initialization, directly applying the indexer for sparse attention leads to a clear performance drop. A brief period of joint training allows the backbone to adapt to the sparse attention pattern and recover to the full-attention level. The results show that QSA maintains performance with only a small number of indexer query heads, far fewer than used in the core attention module. To balance inference speed and accuracy, we ultimately adopt 4 query heads as a lightweight indexer configuration.

Efficiency Analysis. By compressing the key sequence with a ratio of r, QSA reduces indexer complexity from $O(n^2)$ to $O(n^2/r)$, which yields substantial efficiency gains on long sequences. For the indexer, sequence compression directly reduces the computation of MQA logits and top-k selection, yielding a speedup comparable to the compression ratio. As shown in Fig. 6(a,b), this substantially reduces inference costs on long sequences, where the indexer becomes a major bottleneck.

We further evaluate the kernel-level speedup of QSA, accounting for both the indexer cost and sparse core attention computation. For the dense-attention baseline, we use the paged GQA implementation provided by FlashInfer (Ye et al., 2025). Prefill is evaluated under a chunked-prefill setting, while decode includes three additional MTP steps. As shown in Fig. 6(c,d), QSA provides speedups from a context length of 64K, with increasingly larger gains as the sequence length grows. At a context length of 1M, QSA achieves $7.6\times$ and $4.9\times$ attention-module speedups for prefill and decode, respectively, demonstrating its scalability for long-context inference.

2.2 Residual

Motivation. Residual connections give every block a direct path to the network output (He et al., 2016). Pre-normalization keeps training stable at scale (Xiong et al., 2020), but it attenuates the signal each layer receives: every block reads the same stream, so a feature written early must compete with everything written after it. Several lines of work address this by adding paths that bypass the bottleneck, including dense inter-layer connectivity (Huang et al., 2017), an extra residual path for attention values (Zhou et al., 2024), and cross-layer reuse of cached states (Sun et al., 2024).

Work that modifies the residual path directly falls into two families. The first makes each layer's read and write more expressive, following highway networks (Srivastava et al., 2015). The second widens the stream itself: Alternating Updates (AltUp) (Baykal et al., 2023) and Hyper-Connections (HC) (Zhu et al., 2024) replace the single residual vector with several parallel branches. The two are complementary: widening adds capacity, and a richer read/write mechanism decides how that capacity is spent.

Widening the Residual Stream. We first study how much of the reported gain comes from width alone, using a simplified variant of AltUp (Baykal et al., 2023) that fits a pre-norm network. The residual state before block ℓ is a set of n_r branches $\boldsymbol{R}^{(\ell)} \in \mathbb{R}^{n_r \times d}$, where d is the hidden size and $\boldsymbol{R}_i^{(\ell)}$ denotes branch i. Each block holds n_r learnable scalars $\boldsymbol{h} \in \mathbb{R}^{n_r}$ and reads its input as a weighted sum of the branches,

$$\boldsymbol{x}^{(\ell)} = \sum_{i=1}^{n_r} h_i \, \boldsymbol{R}_i^{(\ell)}. \tag{21}$$

The block output $\boldsymbol{y}^{(\ell)}$ is then written back to a single branch, chosen round-robin by depth:

$$\boldsymbol{R}_i^{(\ell+1)} = \boldsymbol{R}_i^{(\ell)} + \mathbf{1}\left[i = \ell \bmod n_r\right] \boldsymbol{y}^{(\ell)}. \tag{22}$$

This adds n_r parameters per block and no matrix multiplication, so its compute cost is negligible; the extra cost is the memory traffic of carrying n_r branches instead of one. Even so, it lowers the training loss of a 25B-A3B MoE model trained on 400B tokens by roughly 0.01. Widening alone therefore yields a substantial loss reduction. The question that remains is how much read/write machinery is worth adding on top of the widened stream to reach a good balance of performance, efficiency, and stability.

Hyper-Connections (HC). HC generalizes Eq. (21) and Eq. (22) into three learnable operators. A read operator $\boldsymbol{H}_{\text{mix}}$ forms the block input, a write operator $\boldsymbol{H}_{\text{combine}}$ distributes the block output over branches, and a mixing operator $\boldsymbol{H}_{\text{res}}$ exchanges information between branches:

$$\boldsymbol{x}^{(\ell)} = \boldsymbol{H}_{\text{mix}}^{\top} \boldsymbol{R}^{(\ell)}, \tag{23}$$

$$\boldsymbol{y}^{(\ell)} = \mathcal{F}^{(\ell)}\left(\text{Norm}\left(\boldsymbol{x}^{(\ell)}\right)\right), \tag{24}$$

$$\boldsymbol{R}^{(\ell+1)} = \boldsymbol{H}_{\text{res}} \, \boldsymbol{R}^{(\ell)} + \boldsymbol{H}_{\text{combine}} \, \boldsymbol{y}^{(\ell)\top}, \tag{25}$$

with $\boldsymbol{H}_{\text{mix}}, \boldsymbol{H}_{\text{combine}} \in \mathbb{R}^{n_r}$ and $\boldsymbol{H}_{\text{res}} \in \mathbb{R}^{n_r \times n_r}$. In the notation of HC these are $\boldsymbol{A}_m$, $\boldsymbol{B}$ and $\boldsymbol{A}_r$. The three operators are predicted from the residual state. Let $\overline{\boldsymbol{R}} = \text{norm}(\boldsymbol{R}^{(\ell)})$ be a normalized view of the branches. Each operator is then the sum of a static term and a data-dependent term:

$$\boldsymbol{H}_{\text{mix}} = \boldsymbol{H}_{\text{mix}}^{\text{s}} + \boldsymbol{\lambda}_m \odot \phi\left(\overline{\boldsymbol{R}} \, \boldsymbol{W}_m\right), \tag{26}$$

$$\boldsymbol{H}_{\text{combine}} = \boldsymbol{H}_{\text{combine}}^{\text{s}} + \boldsymbol{\lambda}_c \odot \phi\left(\overline{\boldsymbol{R}} \, \boldsymbol{W}_c\right), \tag{27}$$

$$\boldsymbol{H}_{\text{res}} = \boldsymbol{H}_{\text{res}}^{\text{s}} + \boldsymbol{\lambda}_r \odot \phi\left(\overline{\boldsymbol{R}} \, \boldsymbol{W}_r\right), \tag{28}$$

where $\boldsymbol{H}_{\star}^{\text{s}}$ is a learnable static term, $\boldsymbol{W}_{\star}$ projects the normalized residual, ϕ is an activation function, and $\boldsymbol{\lambda}_{\star}$ is a learnable scale, with $\star$ standing for any of the three operators. HC uses $\phi = \tanh$ with $\boldsymbol{\lambda}_{\star}$ initialized to 0.01; mHC (Xie et al., 2025) uses a sigmoid and additionally constrains $\boldsymbol{H}_{\text{res}}$ to a manifold of doubly stochastic matrices. Setting the static terms to $\boldsymbol{e}_{\ell \bmod n_r}$, $\mathbf{1}$ and $\boldsymbol{I}$ with $\boldsymbol{W}_{\star} = \mathbf{0}$ starts the widened network exactly at the pre-norm one; with $\boldsymbol{\lambda}_{\star} = \mathbf{0}$ throughout, the operators stay static and the widened stream costs no extra computation, which recovers the simplified AltUp variant above.

Design Ablation. Starting from the static operators, we kept the added expressiveness only where it paid for itself. Tab. 5 reports the endpoints of that progression, evaluated with the benchmark suite and evaluation pipeline of §2.1.1. Five observations shaped the final design.

- **Bounded positive gates.** A sigmoid gate is better than $\tanh$ in both loss and training stability. This agrees with mHC (Xie et al., 2025) and is consistent with the observation in the GDN and attention components that sigmoid gates outperform SiLU or $\tanh$.

Table 5: Residual read/write ablation on 25B-A3B MoE models trained on 560B tokens. The benchmarks and the evaluation pipeline are those of §2.1.1. All widened variants use $n_r = 4$ branches; *static* is the case $\boldsymbol{\lambda}_\star = \mathbf{0}$ of Eq. (26)–(28), *dynamic* its data-dependent counterpart.

		Knowledge			STEM		Reasoning	Multilingual	Code		
Residual	Loss	MMLU	MMLU-Pro	SuperGPQA	MATH	GSM8K	BBH	MMMLU	EvalPlus	MultiPL-E	Avg.
Pre-norm	1.617	64.29	38.40	21.78	53.92	77.41	64.73	51.26	49.25	37.15	50.91
mHC (static)	1.596	64.62	43.69	22.20	55.08	78.05	65.42	52.78	49.59	40.94	52.49
mHC (dynamic)	1.594	66.11	45.84	**24.20**	59.54	**78.51**	66.01	**56.61**	**52.16**	41.30	54.47
GR	1.590	**66.69**	**46.02**	23.80	**61.18**	78.20	**66.54**	56.19	51.36	**42.00**	**54.66**

- **Data dependence.** Making $\boldsymbol{H}_{\text{mix}}$ and $\boldsymbol{H}_{\text{combine}}$ data-dependent reduces loss by 0.002 over the static variant, whereas the static variant reduced loss by 0.021 over the pre-norm baseline. The benchmark gain reverses this ratio: 1.98 points from static to dynamic against 1.58 from baseline to static (Tab. 5). This is one of several places in this report where loss and downstream accuracy do not move together.
- **Read granularity matters more than write granularity.** Refining $\boldsymbol{H}_{\text{mix}}$ in Eq. (23) from one scalar per branch to one weight per branch and channel helps. The same refinement of $\boldsymbol{H}_{\text{combine}}$ in Eq. (25) gives almost nothing, so the write stays a per-branch scalar.
- **Read all branches.** Predicting the operators from all branches is better than using only the last branch or pooling the branches first. Normalizing each branch separately, that is, a group RMSNorm over the widened stream, gives a further gain.
- $\boldsymbol{H}_{\text{res}}$ **adds little.** Once the read and the write are expressive enough, adding the $n_r \times n_r$ mixing operator brings no significant improvement.

Widening the stream with static operators is already worth 1.58 points of average accuracy over the pre-norm baseline, and making the read and the write data-dependent adds a further 1.98. The loss gap between mHC static and dynamic is only 0.002, yet the benchmark gap is substantial; this is a case where loss alone would have understated the value of the change, and it reinforced our practice of checking benchmarks alongside loss throughout the design process. We describe the concrete design of GR below.

Gated Residual. In separate work (Qiu et al., 2026), we had found that adding a lightweight elementwise self-gate after RMSNorm, which we call GatedNorm, markedly improves training stability:

$$\text{GatedNorm}(\boldsymbol{u}) = \text{RMSNorm}(\boldsymbol{u}) \odot \sigma\left(\boldsymbol{W}_2 \,\text{SiLU}\left(\boldsymbol{W}_1 \,\text{RMSNorm}(\boldsymbol{u})\right)\right), \tag{29}$$

where $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ form a low-rank bottleneck. The read the ablation arrives at, elementwise and data-dependent with a sigmoid gate, is exactly Eq. (29) applied to the widened stream. We therefore merge the two into a single operator and call the result Gated Residual (GR).

GR first normalizes each branch independently,

$$\widehat{\boldsymbol{R}}_i = \text{RMSNorm}\left(\boldsymbol{R}_i; \boldsymbol{\gamma}_i\right), \qquad i = 1, \ldots, n_r, \tag{30}$$

each with its own gain $\boldsymbol{\gamma}_i \in \mathbb{R}^d$. It then predicts elementwise gating scores per branch and channel from all branches, and averages the gated branches into the block input:

$$\boldsymbol{G} = \text{unvec}\ \sigma\left(\boldsymbol{W}_u \,\text{SiLU}\left(\tfrac{1}{n_r}\boldsymbol{W}_d \,\text{vec}(\widehat{\boldsymbol{R}})\right)\right) \in \mathbb{R}^{n_r \times d}, \tag{31}$$

$$\boldsymbol{x} = \frac{1}{n_r}\sum_{i=1}^{n_r} \boldsymbol{G}_i \odot \widehat{\boldsymbol{R}}_i, \tag{32}$$

where vec stacks the branches into one vector of length $n_r d$ and unvec is its inverse, with $\boldsymbol{W}_d \in \mathbb{R}^{r \times n_r d}$ and $\boldsymbol{W}_u \in \mathbb{R}^{n_r d \times r}$ for a bottleneck rank $r = d/8$. The block output $\boldsymbol{y} = \mathcal{F}(\boldsymbol{x})$ is written to every branch through one data-dependent scalar per branch:

$$\boldsymbol{s} = 2\,\sigma\left(\tfrac{1}{n_r}\boldsymbol{W}_w \,\text{vec}(\widehat{\boldsymbol{R}})\right) \in \mathbb{R}^{n_r}, \tag{33}$$

$$\boldsymbol{R}'_i = \boldsymbol{R}_i + s_i\,\boldsymbol{y}, \tag{34}$$

with $\boldsymbol{W}_w \in \mathbb{R}^{n_r \times n_r d}$. Eq. (31) and Eq. (33) are the read and write operators of Eq. (26) and Eq. (27) with $\phi = \sigma$, an elementwise $\boldsymbol{H}_{\text{mix}}$, a per-branch scalar $\boldsymbol{H}_{\text{combine}}$, and $\overline{\boldsymbol{R}}$ the group-RMSNorm of all branches. The static term $\boldsymbol{H}^{\text{s}}_\star$ brings no improvement for GR. With the current configuration, no special initialization of the learnable weights is needed, and the static term contributes negligibly; standard random initialization, as used in the backbone, suffices. We use $n_r = 4$ branches, with a separate GR module for the attention block and the MLP block of every layer.

In Tab. 5, GR and mHC (dynamic) differ mainly in the elementwise $\boldsymbol{H}_{\text{mix}}$ and the removal of $\boldsymbol{H}_{\text{res}}$; at this scale the two perform comparably. The efficiency advantage of GR is that removing $\boldsymbol{H}_{\text{res}}$ eliminates a full read of the residual state per block, reducing memory traffic. The stability advantage is twofold: GatedNorm itself improves training stability (analysed in §3.3), and dropping $\boldsymbol{H}_{\text{res}}$, which requires separate constraints, removes a potential source of instability.

GR belongs to the same family as HC, mHC and VWN (Seed, 2025); what differs is where the extra expressiveness is spent. HC and mHC keep the read and the write as per-branch scalars and put capacity into $\boldsymbol{H}_{\text{res}}$, which mHC further constrains to be doubly stochastic. VWN keeps those scalars as well and instead widens the token embedding, splitting it into many narrow segments; this splitting likewise pursues finer-grained read and write operations. GR spends its expressiveness on the read: the gate is elementwise, and $\boldsymbol{H}_{\text{res}}$ is dropped altogether, which the ablation shows costs nothing and which removes a read of the whole residual state per block, the dominant inference cost of a widened stream.

Because the read in Eq. (32) already normalizes and gates, GR also replaces the block's pre-normalization rather than sitting in front of it: Eq. (24) loses its Norm, and widening adds no normalization layer. And with no mixing operator the branches stay separate, since a branch is only ever written by blocks and read through Eq. (32). This makes the information flow easy to follow, which we use in the analysis in §2.2.

Comparison with Attention Residual. Attention Residual (AttnRes) (Team et al., 2026) uses a softmax attention over earlier layers' outputs to determine each sublayer's read. Full AttnRes attends over the output of every preceding sublayer. Block AttnRes partitions the L sublayers into blocks of S, sums each block into one representation, and attends over those.

Table 6: Residual designs at 28 layers, with and without GatedNorm (GN). Loss is the final training loss; S is the number of sublayers summed into one Block AttnRes representation. Subscripts give the change from the column on the left.

Residual design	Loss	Loss + GN
Pre-norm residual	1.789	$1.787_{-0.002}$
Block AttnRes, $S = 4$	1.773	$1.768_{-0.005}$
Block AttnRes, $S = 2$	1.770	$1.766_{-0.004}$
Full AttnRes	1.762	$\mathbf{1.758}_{-0.004}$
GR ($n_r = 4$)	—	1.762

Tab. 6 compares both variants against GR in our setting, a 28-layer model ($L = 56$ sublayers), each with and without GN. Full AttnRes is the strongest setting of that family and lands level with GR at 1.762, while summarizing blocks costs 0.008 at $S = 2$ and 0.011 at $S = 4$. The same ordering holds deeper: at 48 layers, Block AttnRes at $S = 4$ reaches 1.711 against 1.707 for GR. GN lowers the loss at every setting, by 0.004–0.005 on AttnRes and by 0.002 on the plain pre-norm baseline. The gate helps more when the input a sublayer reads is more complex, which is the same gate GR carries inside its read.

What the Branches Are Used For. The ablation shows that GR improves both loss and benchmarks, but what mechanism produces the gain? As GR has no residual branch mixing, each branch is a plain accumulator of past outputs, and we can decompose exactly what each block reads into contributions from every earlier block. We use this decomposition to compare a GR model against an otherwise identical reference without GR, and find that the extra width is spent on several specific paths: one branch preserves early attention outputs across many layers, while the other three stay local.

The statistic. Branch c before block v holds

$$\boldsymbol{R}_c^{(v)} = \boldsymbol{R}_c^{(0)} + \sum_{u<v} s_c^{(u)}\, \boldsymbol{y}^{(u)}, \tag{35}$$

where $\boldsymbol{R}_c^{(0)}$ is the initial value of branch c (the token embedding), $\boldsymbol{y}^{(u)}$ is the output of block u, and $s_c^{(u)}$ is the scalar write gate of Eq. (33) controlling how much of $\boldsymbol{y}^{(u)}$ enters branch c.

Block v reads this branch through the gated read of Eq. (32): it normalizes the branch, scales it by a learned per-channel gain $\boldsymbol{\gamma}_c$ (from Eq. (30)), gates the result elementwise with $\boldsymbol{G}_c^{(v)}$ (the data-dependent gate of Eq. (31)), and averages over branches. Since normalization divides by $\mathrm{rms}(\boldsymbol{R}_c^{(v)})$, the contribution of block u to block v's input is

$$\boldsymbol{a}_{u\to v} = \frac{1}{n_r}\sum_{c=1}^{n_r} \boldsymbol{G}_c^{(v)} \odot \boldsymbol{\gamma}_c \odot \frac{s_c^{(u)}\, \boldsymbol{y}^{(u)}}{\mathrm{rms}\left(\boldsymbol{R}_c^{(v)}\right)}, \tag{36}$$

evaluated at the gate values the forward pass actually took. Each factor has a direct reading: $s_c^{(u)}\, \boldsymbol{y}^{(u)}$ is what block u deposited on branch c; the division by $\mathrm{rms}(\boldsymbol{R}_c^{(v)})$ accounts for the normalization applied at read time; $\boldsymbol{\gamma}_c$ rescales each channel; and $\boldsymbol{G}_c^{(v)}$ decides how much of this branch block v actually uses.

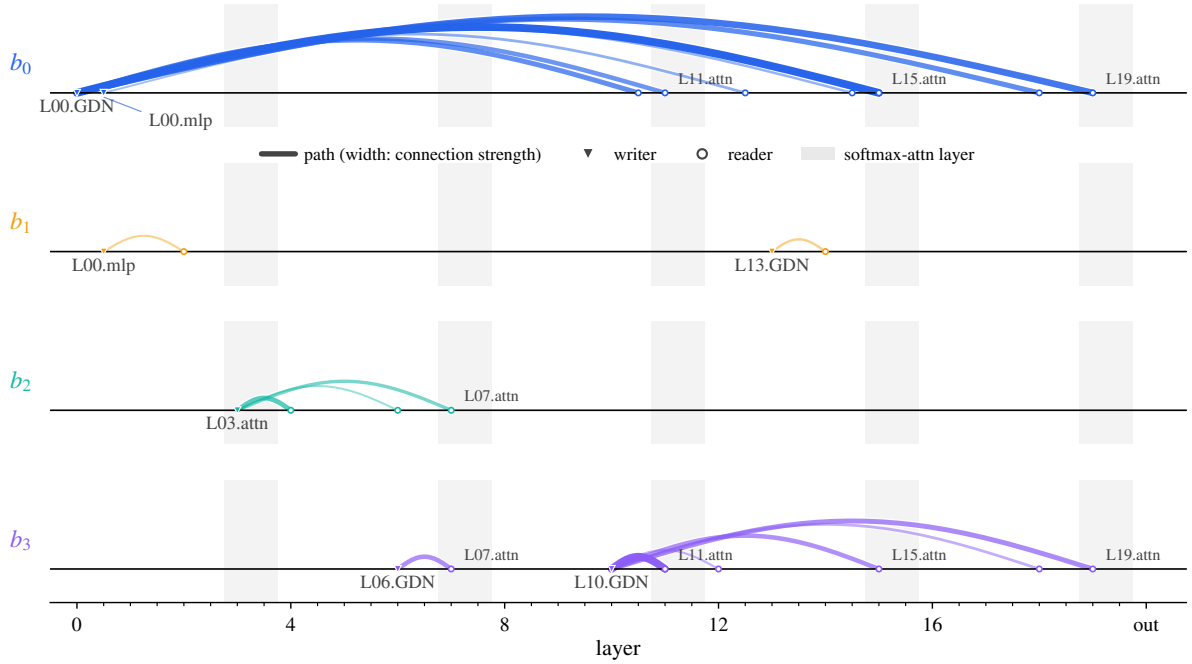


Figure 7: **Cross-layer paths added by GR.** Each row corresponds to one residual branch; a connection runs from the sublayer that wrote into that branch to a later sublayer that reads it back. Vertical position encodes the number of layers skipped; line width and opacity encode Δ_{uv} (Eq. (37)), the additional share of the reader's input supplied by this writer compared to a single residual stream. Shaded regions denote softmax-attention layers, every fourth in this hybrid; the rest are GDN, and sublayers are named accordingly (`L00.GDN`, `L03.attn`, `L00.mlp`). Readers are named where they are softmax-attention sublayers, which is where the long-range paths land. One branch carries long-range paths (all connections on b_0 originate at layer 0 and land past layer 10), while the other three stay local (median skips of 1.2–3.5 layers). The extra width is spent on a small number of specific paths, mostly preserving early GDN output across depth and delivering it to the softmax-attention layers. Thresholds and counts are given in the text.

We report the normalized magnitude of this contribution,

$$\pi_{uv} = \frac{\|\boldsymbol{a}_{u\to v}\|}{\sum_{u'<v}\|\boldsymbol{a}_{u'\to v}\|}, \tag{37}$$

the fraction of block v's input that came from block u. Because Eq. (36) splits each contribution by branch before summing, we can see which branch carries each path. The decomposition is exact: every reader's shares sum to one to within 3×10^{-8}.

We compare a 20-layer MoE trained with GR against an otherwise identical reference without GR, same recipe, data, optimizer and training step, probed on the same tokens. We report the difference $\Delta_{uv} = \pi^{\mathrm{GR}}_{uv} - \pi^{\mathrm{ref}}_{uv}$, since subtracting the reference removes the pattern common to all residual networks, namely that nearby writers dominate.

What the figure shows. Fig. 7 shows the 21 paths (out of 780 ordered pairs) with $\Delta_{uv} \geq 0.05$ over a skip of at least one layer. To calibrate: at layer 15 there are 30 writers, so an equal split gives each about 0.03; a share of 0.13 is four times that.

One branch carries long-range paths while the other three stay local. Which branch it is does not matter, since the branches are exchangeable at initialization, but the pattern is consistent: across five GR checkpoints, each has exactly one such branch, with a typical skip of 10.9 layers against 3.4–3.9 for the rest. Three examples illustrate the pattern; each gives the share under the reference model (a single residual stream without GR) and then under GR.

- Layer 0 GDN $\to$ layer 15 attention: the share rises from 0.020 in the reference to 0.138 in GR. This share holds at 0.072–0.138 across every reader from layer 10 to 19 with no downward trend. This confirms the outsized role of the first layer in preserving information across depth, consistent with prior observations (Elhage et al., 2021; Men et al., 2024).
- Layer 10 GDN $\to$ layer 11 attention: $\Delta_{uv} = 0.117$. The gain is as large as the path above, but over a single layer: GR strengthens short-range connections as well.

- Layer 0 MLP, on two branches at once: to layer 15 on the long-range branch, where its share rises from 0.008 in the reference no-GR model to 0.058 in GR; and to layer 2 on a local one, where it rises from 0.139 to 0.192. The same output reaches a nearby and a distant reader with different strengths, which a single stream cannot express since it has only one decay rate for every writer.

The typical long-range path in branch 0 follows from Eq. (35): the branch is written most heavily at layer 0 and barely updated thereafter, so that layer-0 information remain accessible to all subsequent layers. Furthermore, the sublayers that read most heavily from the GR branches are predominantly the softmax attention layers, indicating that global attention acts as a critical hub for integrating explicit long-range historical context that the GDN layers compress away.

To see the overall pattern, we group all 780 paths by how many layers they skip and sum Δ_{uv} within each group, where Δ_{uv} is the extra share that GR gives to path (u, v) over the reference. The result: adjacent-layer paths (skip 1) collectively gain 0.96 in share; long-range paths (skip > 12) collectively gain 0.91; and mid-range paths (skip 2–12) collectively lose 3.21. The weighted-average skip is almost unchanged (3.97 vs 3.91), so the total amount of cross-layer information is similar; what changes is its distribution. GR selects a few paths and amplifies them, at the expense of mid-range ones.

Inference Efficiency. The inference cost of GR is dominated by the memory traffic of the widened residual state. We therefore looked for ways to move fewer bytes.

The first attempt was to sparsify the read. We observed that in trained models, the write at each GR layer is typically dominated by two branches. We therefore tried introducing sparse writes, either from scratch or mid-training, where each block reads only the two branches with the highest gate values instead of all n_r. Pre-training loss and benchmarks were almost unaffected, but the quality degraded clearly after post-training, so we did not adopt it. More complex variants, such as varying the sparsity level across layers, did not resolve the issue. We note *this as a case where pre-training metrics alone would have led to the wrong decision*. Recent work xHC (Zhang et al., 2026) explores using a larger n_r to make sparse branch updates easier, but given the memory overhead of a larger n_r, we did not explore this direction further.

The second attempt was to keep the residual state in FP8. The gates in GR, gated attention, and GDN all bound the magnitude of what is written into the stream, so residual values stay in a narrow range and are well matched to a low-precision format. Storing the branches in FP8 halves the bytes moved for the residual state relative to BF16, with almost no loss in quality. Finally, the read of Eq. (30)–(32) and the write of Eq. (33)–(34) are each fused into a single kernel, with the group RMSNorm folded into the read, so the widened stream is traversed once per block in each direction.

2.3 N-gram Embedding

Motivation. Embedding-based memory offers a complementary dimension for scaling model capacity (Google DeepMind, 2025; RWKV Community, 2025; Gemma Team, 2026; Liu et al., 2026; Sadhukhan et al., 2026; Tseng & De Sa, 2026). N-gram embeddings further generalize unigram lookup by conditioning memory retrieval on local context rather than token identity alone (Huang et al., 2021; Roy et al., 2022; Huang et al., 2025; Yu et al., 2025; Cheng et al., 2026; Chen et al., 2026). Concretely, short n-grams ending at each token serve as keys into embedding tables, and the retrieved vectors augment the corresponding token representation. N-gram memory scales capacity with negligible additional per-token FLOPs, while deterministic addressing enables host-memory offloading and asynchronous prefetching (Google DeepMind, 2025; Cheng et al., 2026). In this section, we systematically study the key architectural choices for N-gram embedding. We use 300 tokens per active parameter (TPP) throughout experiments.

2.3.1 Placement of the N-gram Embedding Layers

Table 7: Effect of N-gram embedding layer placement. The total number of N-gram embedding parameters is fixed across all settings.

		Knowledge			STEM		Reasoning	Multilingual	Code		
Layer Index	Loss	MMLU	MMLU-Pro	SuperGPQA	MATH	GSM8K	BBH	MMMLU	EvalPlus	MultiPL-E	Avg.
w/o N-gram emb.	1.585	62.78	33.43	20.97	32.52	59.21	53.40	54.06	52.13	45.42	45.44
1st	1.541	64.19	35.25	21.30	36.20	**65.73**	56.00	56.16	50.95	**47.45**	47.30
2nd	1.541	64.71	**35.80**	21.49	37.32	64.00	57.56	**56.64**	**53.09**	45.56	47.94
3rd	1.543	63.20	34.93	20.67	35.74	63.15	56.71	55.02	50.15	44.90	46.76
4th	1.544	63.36	34.81	**22.33**	36.20	61.26	55.32	55.79	51.05	45.60	46.89
10th	1.544	64.22	34.99	20.81	33.80	62.17	55.54	54.81	52.69	43.61	46.62
15th	1.543	**65.07**	34.95	21.35	36.12	63.50	57.15	55.98	52.21	44.42	47.37
25th	1.541	64.70	35.15	22.13	36.26	63.31	55.73	55.99	52.66	44.33	47.40
2nd + 15th	1.541	63.82	35.63	21.25	**37.48**	63.23	56.52	55.68	50.44	43.85	47.01
2nd + 25th	**1.540**	64.94	35.40	21.80	37.40	64.33	**57.79**	56.45	51.69	44.73	47.75

We ablate the placement and number of N-gram embedding layers under a fixed parameter budget. Single-layer variants span shallow (Layers 1-4), intermediate (Layers 10 and 15), and deep (Layer 25) locations. For multi-layer configurations, we combine a shallow layer (Layer 2) with either an intermediate layer (Layer 15) or a deep layer (Layer 25). The results are shown in Table 7.

No single depth regime consistently dominates. The first two layers perform strongly, while intermediate and deep placements remain competitive. Distributing the same parameter budget across multiple layers yields no consistent benefit. The marginal loss reduction from combining Layers 2 and 25 does not translate into improved downstream performance. A single N-gram embedding layer is sufficient. Moreover, the relative performance of different placements is similar under full attention and GDN (§2.1.1), suggesting that placement choice is largely insensitive to the attention mechanism. We place it at Layer 2, *allowing host-memory prefetching to overlap with the computation of the first layer.*

2.3.2 N-gram Vocabulary Size

We further explore the effect of scaling the vocabulary size of N-gram embeddings.

Table 8: Effect of N-gram vocabulary scaling under a fixed total model parameter budget. Vocabulary scales are measured relative to the base tokenizer vocabulary size (250K). The number of MoE experts is adjusted to offset the additional N-gram embedding parameters.

Vocab. Scale	Loss	Uncheatable	Knowledge			STEM		Reasoning	Chinese		Multilingual
(Param. Ratio)		PPL	MMLU	MMLU-Pro	SuperGPQA	MATH	GSM8K	BBH	C-Eval	CMMLU	MMMLU
None (0%)	1.202	5.55	**68.25**	44.38	25.22	46.02	**74.79**	**65.71**	70.78	73.01	**58.32**
5× (10%)	1.200	**5.54**	68.15	44.49	24.25	45.64	72.86	65.39	70.93	**73.44**	57.49
10× (25%)	**1.197**	5.55	67.71	**44.66**	**25.64**	**46.56**	73.65	65.54	70.71	73.31	56.22
30× (50%)	1.201	5.59	67.75	42.61	24.18	44.62	74.45	65.55	**72.49**	73.28	56.66

Allocation under Fixed Parameter Budget. Following prior work, we scale the N-gram embedding slots while reducing the number of experts to maintain a fixed total parameter budget. The results are reported in Table 8. The loss varies non-monotonically with vocabulary size and is lowest at $10\times$ (25%), consistent with the allocation sweet spots reported in prior work (Liu et al., 2026; Cheng et al., 2026). The same optimum is not evident in other evaluations. The out-of-domain uncheatable PPL changes little across budgets, and downstream benchmarks show no clear improvement over the MoE-only baseline. These results suggest that *N-gram embeddings and MoE experts play distinct roles in scaling capacity. We therefore study N-gram vocabulary scaling while holding the MoE parameter budget fixed.*

Table 9: Effect of N-gram vocabulary scaling. Vocabulary scales are measured relative to the base tokenizer vocabulary size (250K). Larger vocabularies increase the total number of parameters.

		Knowledge			STEM		Reasoning	Chinese		Multilingual
Vocab. Scale	Loss	MMLU	MMLU-Pro	SuperGPQA	MATH	GSM8K	BBH	C-Eval	CMMLU	MMMLU
None	1.585	62.78	33.43	20.97	32.52	59.21	53.40	66.91	68.10	54.06
20×	1.553	64.14	34.46	21.83	**37.38**	**65.09**	57.13	71.75	72.29	55.94
50×	1.541	64.71	35.80	21.49	37.32	64.00	**57.56**	72.12	72.48	56.64
100×	1.534	64.70	**35.87**	**21.93**	36.98	63.08	56.03	73.75	72.73	**56.65**
200×	**1.526**	**64.85**	35.21	21.11	35.34	62.96	56.23	**74.94**	**73.24**	55.82

Scaling with Additional Parameters. Because embedding tables are sparsely accessed and deterministically addressed, they can be scaled with negligible additional per-token computation and stored in off-accelerator storage (Google DeepMind, 2025; Cheng et al., 2026). We therefore move beyond the fixed-size setting and scale the N-gram vocabulary from $20V$ to $200V$, where V denotes the base tokenizer vocabulary size of Qwen3.5 (Qwen Team, 2026). The results are shown in Table 9. Loss decreases monotonically as the N-gram vocabulary grows, while downstream performance does not follow the same trend. Vocabulary scaling yields broad gains over the baseline, but performance on some benchmarks saturates or fluctuates as the vocabulary grows. Furthermore, performance on Chinese benchmarks (e.g., C-Eval and CMMLU) improves consistently with N-gram vocabulary size.

Beyond vocabulary scaling, we explored a range of strategies for improving parameter efficiency of N-gram embeddings, including but not limited to token normalization for vocabulary compression (Cheng et al., 2026), non-uniform allocation across N-gram orders, and frequency-based partitioning of embedding slots. Despite these efforts, we observed no consistent performance gains in our training recipe.

3 Optimization

3.1 Optimizer

Motivation. The matrix-based optimizer Muon (Jordan et al., 2024) computes the update direction for a matrix parameter by orthogonalizing the momentum using Newton–Schulz (NS) iterations, and has been shown effective at scale (Liu et al., 2025b; Kimi Team, 2025). We use Muon as the main optimizer and find that several design choices influence its practical efficiency and stability, which we describe below.

Orthogonalization. The NS iteration is applied to a Nesterov-accelerated momentum (Jordan et al., 2024) with $\mu = 0.95$, and the orthogonalized result is scaled by $\gamma(A, B) = 0.2\sqrt{\max(A, B)}$ for a parameter of shape $A \times B$, making the RMS of the update independent of the matrix shape (Liu et al., 2025b). For the NS iteration, we adopt the per-step coefficient schedule of the Polar Express method (Amsel et al., 2025), which is minimax-optimal for a given step budget. We set the number of iteration steps to 8, which provides more accurate orthogonalization than fewer steps and reduces both the magnitude and frequency of gradient-norm spikes in our stress test. We set the numerical stability constant in the Frobenius normalization preceding the NS iteration to 10^{-14}.

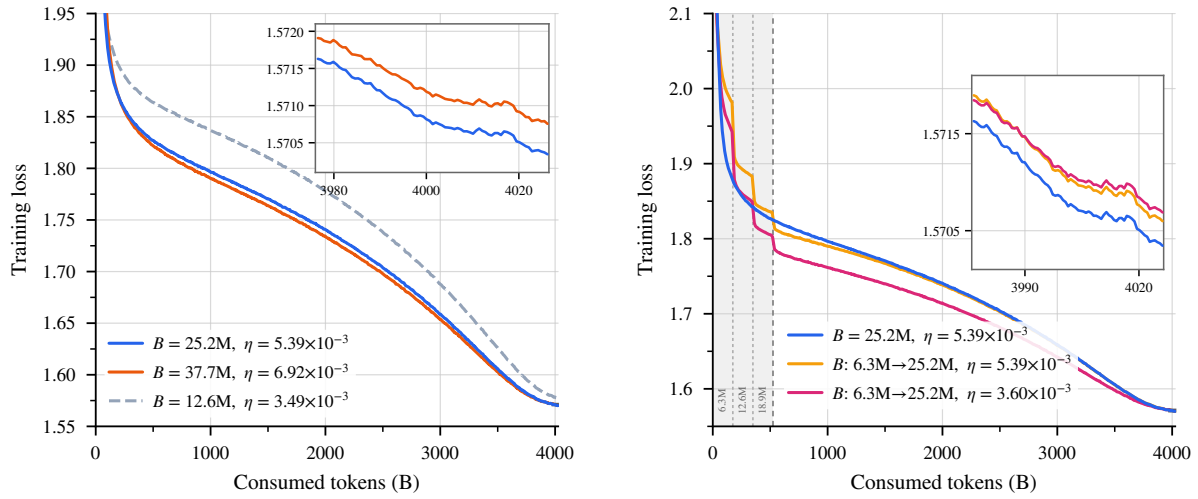
Which Parameters Use Muon. We apply Muon to the two-dimensional weights that genuinely act as linear maps: the attention q/k/v and output projections, the GDN (Yang et al., 2024) input and output projections, the fc1/fc2 of both routed and shared experts, and the key/value projection in N-gram embedding layers. The input embeddings and the output head stay on AdamW. For the MoE router, we observe that Muon exacerbates early-training fluctuations and destabilizes the router. Although applying Muon to the router during the mid-to-late training stages does not cause instability, we find no significant performance gains. Therefore, we use AdamW for the router. One possible explanation is that each output dimension of the router corresponds to the score of one expert, and the dimensions are largely independent, leaving no shared linear structure for orthogonalization to exploit. The two low-rank projections of GR (§2.2) likewise perform better with AdamW. We attribute this to their very elongated shape. Finally, the n-gram embedding table runs on Adam with weight decay disabled.

Splitting Fused Parameters. In Megatron-LM (Shoeybi et al., 2019), the attention qkv projection, the SwiGLU (Shazeer, 2020) fc1, and the GDN input projection are each stored as one fused matrix, but semantically they are concatenations of independent linear operators along the output dimension. Orthogonalizing the fused matrix is then wrong in two ways: the iteration mixes singular directions across unrelated sub-blocks, and $\gamma(A, B)$ is computed from the concatenated shape instead of the true operator shape. We therefore split the fused gradient before orthogonalization, run NS on each sub-matrix independently, and gather the results back into the original layout before applying the update. The qkv and GDN input projections are split at per-head granularity, which improves both loss and downstream benchmarks. The fc1 is split into its gate and up halves; the loss is essentially unchanged while benchmarks improve slightly. Splitting also provides a natural granularity for excluding individual sub-matrices from Muon. Specifically, the GDN decay and beta projections produce one scalar per head and are therefore vectors, on which orthogonalization is not meaningful. For the output gates (the attention output gate (Qiu et al., 2025b) and the GDN z projection), our ablations found AdamW on par with or slightly better than Muon.

Implementation. The NS iteration introduces two main implementation challenges for Muon: First, the iteration requires a holistic update over each full parameter matrix ($\approx 4K \max(A, B) \min(A, B)^2$ FLOPs for K iteration steps), which clashes with Megatron's sharding in two ways: under TP, no rank owns the full weight matrix; under DP, the cost is cubic in the shorter dimension, so Megatron's equal-element partition leaves severe stragglers. We developed **Canzona**, which decouples logical optimizer assignment from physical parameter layout (Wang et al., 2026a): an α-balanced static partitioner reassigns whole parameters (no cut inside a tensor) to equalize estimated NS FLOPs across DP ranks, and an asynchronous Micro-Group pipeline reconstructs each Muon-owned matrix via fused All-to-All across TP ranks. Each owner then runs a step mathematically equivalent to single-device Muon, ZeRO-1's bucket geometry is preserved so Megatron's Reduce-Scatter/backward overlap is retained, and the abstraction extends to other matrix-based optimizers. Second, after splitting, one layer contributes on the order of a hundred sub-matrices, and the optimizer step becomes a long sequence of very small kernels bounded by launch overhead rather than arithmetic. We capture the whole step in a CUDA graph to remove this overhead.

3.2 Hyperparameter Scaling

Motivation. Choosing near-optimal hyperparameters is critical for efficient and stable model training. In previous generations of Qwen models, we fitted scaling laws for key hyperparameters, such as the



(a) Constant batch size.

(b) Ramped batch size.

Figure 8: **Batch size on a 4T-token budget.** Training loss against consumed tokens for a 20-layer 10.8B-A0.89B MoE; each inset resolves the last 50B tokens. **(a)** Moving from the previous recipe ($B = 12.6$M) to the predicted optimum ($B = 25.2$M) is worth 7.2×10^{-3}, while a further step to $B = 37.7$M leads to a minor degradation. The loss rises steeply below the prediction and remains nearly flat above it. **(b)** Reaching $B = 25.2$M through a ramp instead performs no better than using this batch size from the start and takes 18.8% more optimizer steps.

learning rate (η) and batch size (B) (Qwen Team, 2024; Yang et al., 2025). The optimal learning rate and batch size depend on the model architecture and the optimizer, so a recipe that was optimal for the previous-generation model may become suboptimal once both change. Under our previous Qwen3.5 hyperparameter recipe (Qwen Team, 2026), we find that the new architecture and optimizer train noticeably more stably than before (§3.3), suggesting room for a more aggressive hyperparameter setting to further improve training performance.

Therefore, we develop an updated hyperparameter scaling law. The architecture and optimizer changes shift the predicted near-optimal hyperparameters toward substantially larger batch sizes and learning rates, with a slower decay in the learning rate as model size increases.

A scaling law fitted at small scales is only useful if it extrapolates reliably. We validated the learning-rate and batch-size predictions separately, evaluating each in a regime designed to make its effect more pronounced: the batch size prediction on a small model trained for many tokens (where an excessively large batch size may lead to suboptimal performance), and the learning rate prediction on a large model trained for a limited token budget (where training instability is the primary risk). The predicted hyperparameters lie in the near-optimal basin and yield clear improvements over the previous Qwen3.5 recipe in both pretraining loss and benchmark performance.

Batch Size at a Large Token Budget. The batch-size prediction is evaluated on a 20-layer 10.8B-A0.89B MoE model trained over 4T tokens. We compare the previous recipe ($B = 12.6$M), the optimum predicted by the new scaling law ($B = 25.2$M), and a setting $1.5\times$ larger ($B = 37.7$M), with each configuration using the learning rate prescribed by the scaling law for its respective batch size. All runs consume the same token budget, ensuring the comparison is based on equal compute.

Averaged over the final 20B tokens, the losses are 1.5702 at $B = 25.2$M , 1.5707 at $B = 37.7$M, and 1.5774 under the previous recipe (Fig. 8a). The new scaling-law fit therefore improves the loss by 7.2×10^{-3} over the previous recipe, while a further $1.5\times$ increase in batch size incurs a 4.3×10^{-4} penalty, which is not significant. The loss increases sharply below the predicted batch size and plateaus above it, indicating that the prediction is close to optimal and large enough to realize performance gains without being excessive.

Batch-Size Warmup Is No Longer Necessary. Large runs commonly ramp the batch size over early training rather than starting at its final value (Brown et al., 2020; Liu et al., 2024). The rationale is that the critical batch size is small early in training and grows over time (McCandlish et al., 2018; Zhang et al., 2024), making a large batch inefficient at the start. Furthermore, smaller batches paired with a scaled learning rate (Goyal et al., 2017) are generally easier to stabilize during the initial steps.

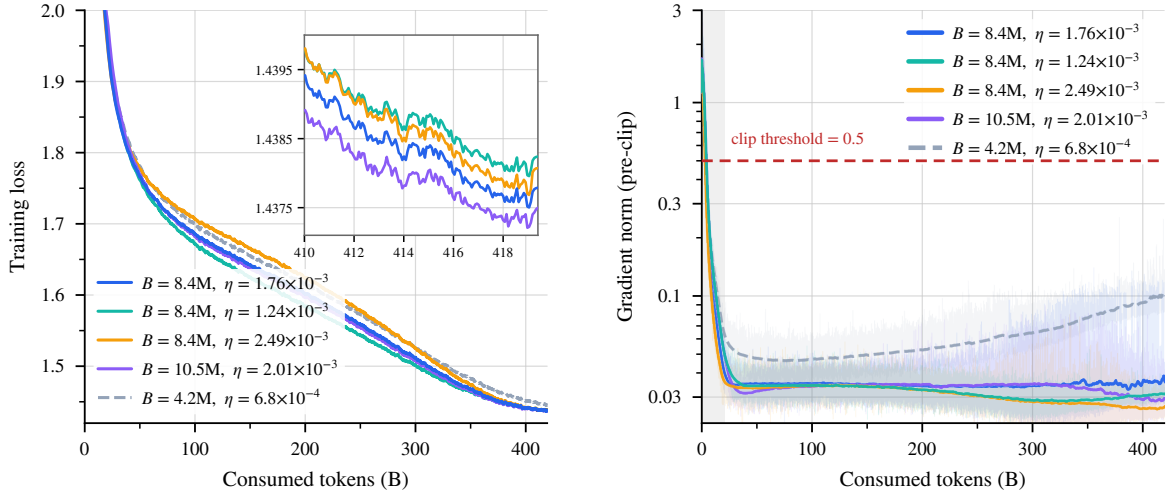
However, our previous hyperparameter sweep suggests a different behavior when using the Muon optimizer. We observe that decreasing the batch size below the predicted optimum incurs a more

significant performance penalty than increasing it, and that Muon preserves data efficiency at larger batch sizes where AdamW's performance degrades (Jordan et al., 2024; Essential AI, 2025). For sparse MoE models, a larger batch also ensures that every expert receives a sufficient and diverse token signal per step, which plausibly aids expert specialization (Qiu et al., 2025a). Together, these observations led us to hypothesize that batch-size warmup may be unnecessary.

We therefore re-tested the batch-size warmup. We increase the batch size from 6.3M in increments of 6.3M, reaching the target of 25.2M at 524B tokens. We evaluated two variants: the first maintains the peak learning rate of the constant-batch optimum, making the ramp its only difference; the second lowers the peak learning rate to account for the smaller batch sizes in the early stages.

Neither variant improves performance (Fig. 8b). Both ramps converge within the run-to-run variance but remain slightly worse than the constant-batch baseline, underperforming by 2.5×10^{-4} and 3.5×10^{-4}, respectively. Additionally, the warmup requires 18.8% more optimizer steps for the same token budget, resulting in greater wall-clock time overhead. Training stability is also unaffected: no run in this sweep exhibits a step where the loss exceeds its local median by more than 0.1, and the p99.9 pre-clip gradient norm ranges from 0.088 to 0.190, well below the clipping threshold of 0.5.

The loss trajectory of the warmup runs reveals the underlying mechanism. During the warmup phase, the smaller batch size introduces higher gradient noise under the same learning rate, resulting in a higher loss compared to the constant-batch baseline. Shortly after the batch size reaches its target, the warmup variant may exhibit a transient loss advantage due to the greater number of optimization steps accumulated during the early phase. However, as the learning rate decays and the model approaches convergence, this step-count advantage is neutralized. Ultimately, the constant-batch baseline surpasses the warmup runs, yielding a better final loss. Consequently, we do not employ batch-size warmup in our production runs.



(a) Training loss.

(b) Gradient norm.

Figure 9: **Learning rate at a larger model scale.** Five runs of a 48-layer MoE on a 419B-token budget: the predicted optimum ($B = 8.4$M, $\eta = 1.76 \times 10^{-3}$), η divided and multiplied by $\sqrt{2}$, a 25% larger batch with its matched η, and the previous recipe (dashed). **(a)** The inset covers the last 10B tokens. The loss differences among the predicted optimum and the three nearby settings are near the noise level, so the predicted optimum sits at the bottom of a flat bowl. **(b)** Pre-clip gradient norm on a log scale, with individual steps faint behind a moving average, the warmup shaded and the clip threshold dashed. After warmup, the pre-clip gradient norms of all runs near the predicted optimum remain below 50% of the clipping threshold, including the run at $\sqrt{2}$ times the predicted learning rate.

Learning Rate at a Larger Model Scale. The learning-rate prediction is tested on a much larger 48 layers 156B-A7B MoE, over a 419B-token budget that is again the same for every run. We evaluate the predicted optimum together with learning rates scaled by $1/\sqrt{2}$ and $\sqrt{2}$, a 25% larger batch with its matched learning rate, and the previous hyperparameter recipe.

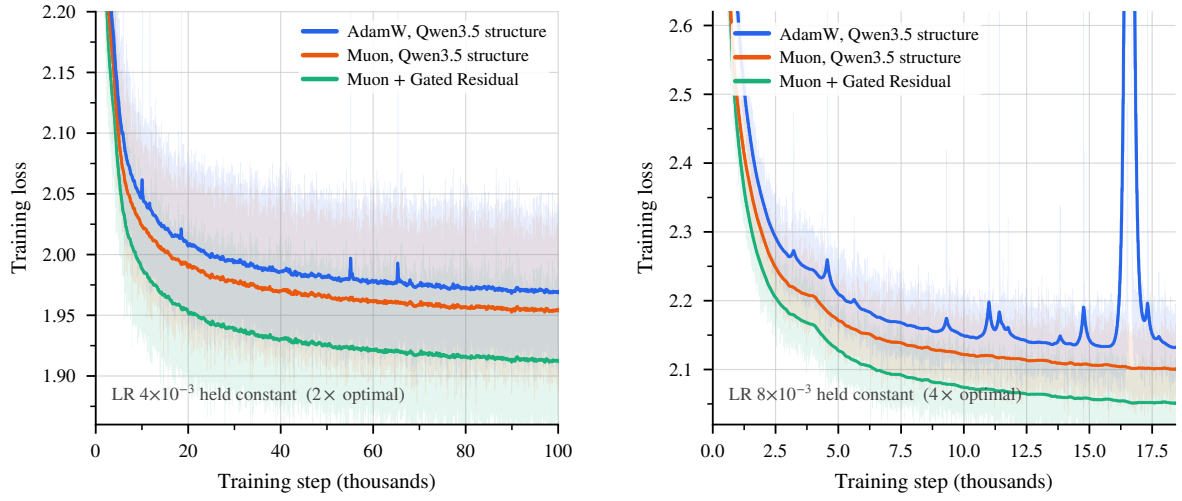
The previous recipe ends 7.8×10^{-3} above the predicted optimum, while the four settings near the predicted optimum end within 7×10^{-4} of each other, near the noise level (Fig. 9a). The optimum therefore sits at the bottom of a bowl that is flat over at least a factor of $\sqrt{2}$ in either direction in learning rate and +25% in batch size; the larger batch is nominally the best of the four, by 3×10^{-4}.

Table 10: Downstream accuracy of the five learning-rate runs of Fig. 9a, all at the same 419B-token budget on the 48-layer 156B-A7B MoE. All values are percentages and higher is better; the benchmarks and the evaluation pipeline are those of §2.1.1. Bold denotes the best result in each column.

			Knowledge			STEM		Reasoning	Multilingual	
Setting	B	η	MMLU	MMLU-Pro	SuperGPQA	MATH	GSM8K	BBH	MMMLU	Avg.
New fit, predicted optimum	8.4M	1.76×10^{-3}	**73.84**	48.35	**29.31**	**49.98**	**80.89**	73.25	68.23	**60.55**
New fit, $\eta \div \sqrt{2}$	8.4M	1.24×10^{-3}	73.59	47.00	28.10	48.92	77.48	72.00	66.88	59.14
New fit, $\eta \times \sqrt{2}$	8.4M	2.49×10^{-3}	**73.84**	46.92	28.04	49.58	80.06	**73.82**	**68.46**	60.10
New fit, $B \times 1.25$	10.5M	2.01×10^{-3}	73.73	**48.51**	28.52	49.32	80.06	72.58	67.72	60.06
Qwen3.5 recipe	4.2M	6.8×10^{-4}	71.23	45.35	25.67	45.54	74.32	69.54	63.19	56.41

Downstream benchmark results demonstrate the robust convergence achieved by our new scaling-law fit (Tab. 10). The predicted optimum yields the highest average accuracy, securing the best or tied-best scores across the majority of evaluated tasks, while the previous recipe falls noticeably behind. Crucially, increasing the batch size or learning rate beyond the predicted optimum results in only a marginal, statistically insignificant drop in benchmark performance. This indicates a highly stable optimization landscape where the model's generalization does not sharply degrade with slight hyperparameter deviations. We treat these specific rankings as observational; given the single evaluation per run and the narrow margins among the top settings, these minor variations likely fall within standard evaluation noise.

Beyond final performance, maintaining training stability is paramount at large model scales. The training dynamics remain exceptionally stable across all configurations. Gradient clipping never engages after the warmup phase in any of the five runs. At the predicted optimum, the maximum pre-clip gradient norm reaches only 28% of the clipping threshold, compared to 51% under the previous recipe, which suffers from noisier gradients due to its smaller batch size (Fig. 9b). Furthermore, the loss curves are remarkably smooth without any loss spike; no run exhibits a single step where the loss exceeds its local median by more than 0.1. Even further increasing the predicted learning rate at this scale remains entirely stable. This confirms that our new scaling law does not dangerously over-extrapolate, providing an efficient, safe and robust hyperparameter recipe for large-scale training.



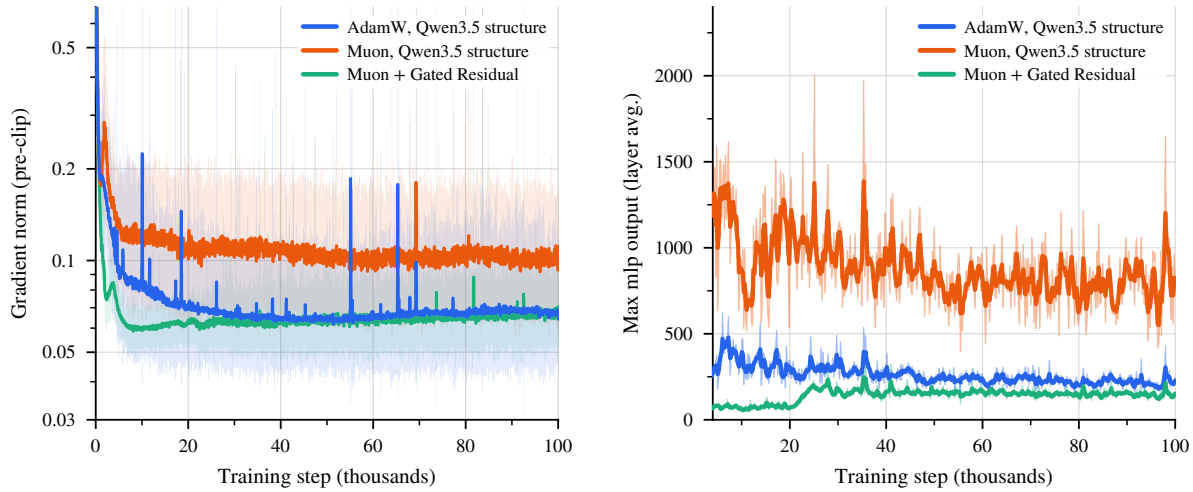
(a) 2× optimal learning rate.

(b) 4× optimal learning rate.

Figure 10: **Training loss under stress.** The 28-layer 25B-A3B MoE at a constant learning rate: the Qwen3.5 structure under AdamW, the same structure under Muon, and Muon with GR. Bold lines are a moving average over the faint per-step trace. GR enables more stable training.

3.3 Stability Stress Test

Motivation. When a model scales to trillions of parameters and is trained on tens of trillions of tokens, it enters a regime where stability challenges emerge that are entirely absent in smaller-scale experiments (Chowdhery et al., 2023; Zhang et al., 2022; Dehghani et al., 2023; Qwen Team, 2026). For example, long training runs can spend substantially more optimizer steps at peak learning rates, increasing the opportunity for instabilities to emerge. Such instabilities often manifest as loss spikes or divergence that require checkpoint restarts. To iterate efficiently at moderate scale while still surfacing the instabilities that would appear at production scale, we design a set of stress tests that amplify the relevant stress within a smaller budget. This verification becomes essential when the architecture and optimizer change



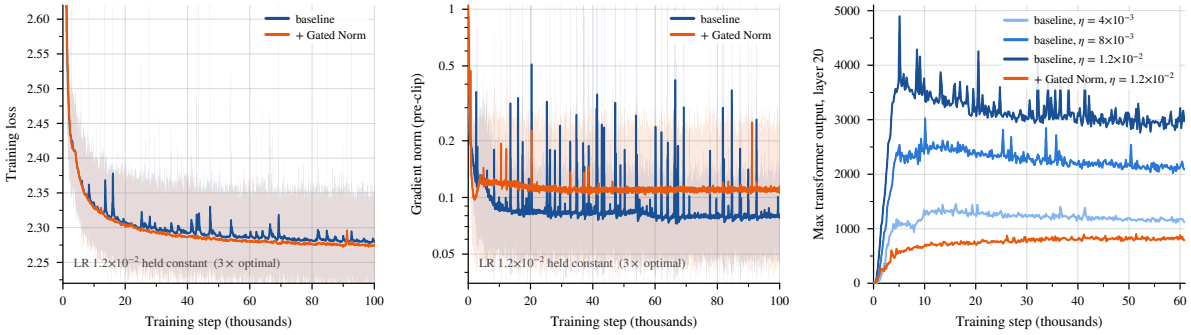
(a) Pre-clip gradient norm. (b) Maximum MLP output.

Figure 11: **Gradient norm and activations under stress.** The same three runs as Fig. 10 (a). The activation in (b) is averaged over layers. Gradient Residual reduces both the frequency and magnitude of gradient-norm spikes, as well as the magnitude of activation outliers (Fig. 11).

together, as they do in Qwen3.8-Flash-Next: the gated residual, the GDN hybrid, and Muon all alter how updates and activations are scaled, and confirming that these changes remain stable under prolonged training is a prerequisite for reliable scaling.

Stress Test Design. Following the observation that large-scale instabilities can be reproduced in small models by raising the learning rate (Wortsman et al., 2023), we hold the learning rate constant at a multiple of its optimal value, bypassing the standard decay schedule to simulate the prolonged peak learning rate of a production run. We apply this to a 28-layer MoE at $2\times$ and $4\times$ its optimal learning rate. The evaluation criterion is that the new recipe must be at least as stable as the previous Qwen3.5 structure with AdamW (Qwen Team, 2026), which has already been scaled successfully. All runs share the same batch size and a gradient-norm clipping threshold of 0.5 (Pascanu et al., 2013). We measure three quantities: loss spikes (steps exceeding a 201-step rolling median by more than 0.1), the $p_{99.9}$ of the pre-clip gradient norm and the number of threshold crossings, and the per-block maximum activation.

Stress Test Results. The stress tests reveal a clear stability margin for the new recipe. At $2\times$ the optimal learning rate, the AdamW baseline begins to spike (4.3 per 10k steps), while both Muon configurations remain highly stable (0.2 per 10k steps). At $4\times$ the optimal learning rate, the differences become categorical (Fig. 10). The AdamW run spikes on 183 per 10k steps and crosses the clipping threshold on 213 of $19{,}932$ steps, engaging the clipper continuously. In contrast, both Muon runs never cross the clipping threshold, and the configuration with the gated residual records zero loss spikes. Under equal stress, the new architecture and optimizer combination is substantially more stable.

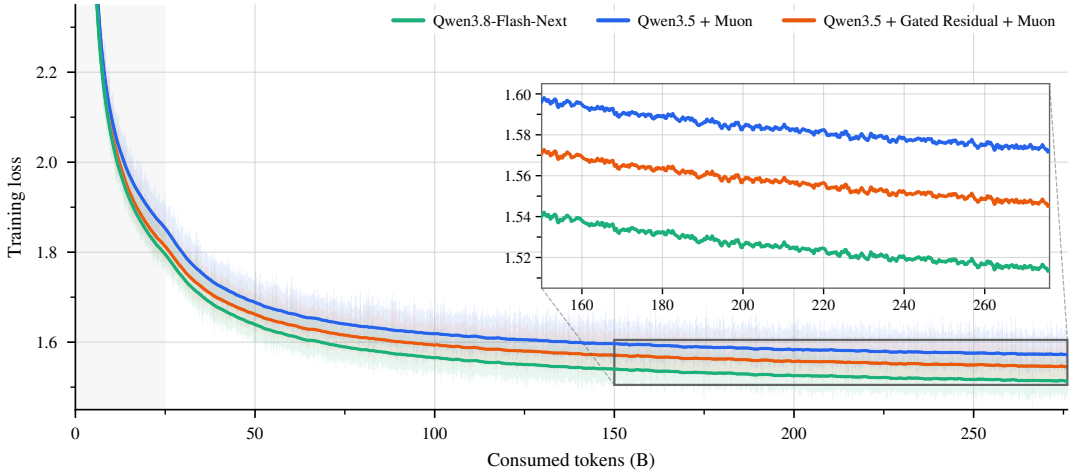


(a) Training loss. (b) Pre-clip gradient norm. (c) Outliers against learning rate.

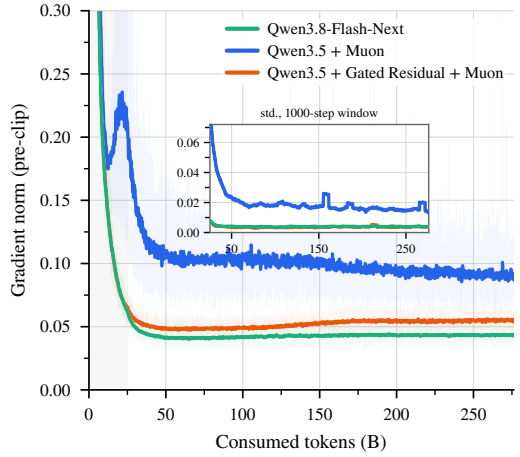
Figure 12: **Isolating the effect of the gate.** GatedNorm off and on, with the AdamW optimizer, structure and data order held fixed. (**a**) and (**b**) are the pair at $3\times$ the optimal learning rate; bold lines are a moving average over the faint per-step trace. (**c**) adds the ungated baseline at $1\times$ and $2\times$ the optimal rate.

Mechanism: The Role of the Gate. Analyzing the gradient norms and activations provides insight into this stability margin. At $2\times$ the optimal learning rate, Muon runs exhibit a higher median gradient norm and larger maximum activations than AdamW, yet they produce far fewer loss spikes. Adding GR reduces both the frequency and magnitude of gradient-norm spikes, as well as the magnitude of activation outliers (Fig. 11).

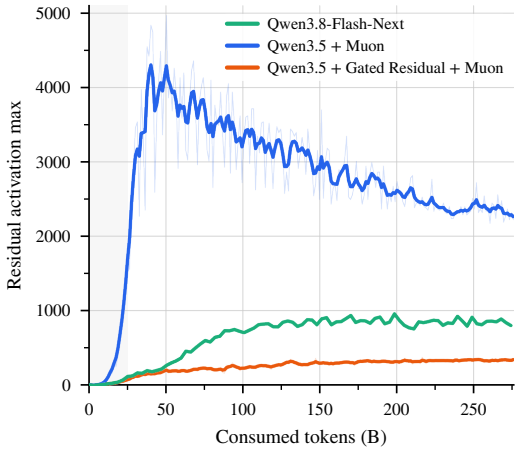
To isolate the effect of the gate, we evaluate a single-variable pair on the 28-layer model at $3\times$ its optimal learning rate, keeping the AdmaW optimizer and structure fixed while toggling GatedNorm (Fig. 12). Enabling the gate reduces the spike rate from 32.0 to 3.2 per 10k steps and cuts threshold crossings from 256 to 20. A learning-rate ladder on the ungated baseline shows that activation outliers grow almost proportionally with the learning rate, while the spike rate grows much faster (Fig. 12c (c)). With the gate enabled at the highest learning rate, the outlier level drops below the baseline at the lowest learning rate. This suggests that training at high learning rates requires a rescaling mechanism: without an explicit gate, the network achieves this by growing activation outliers, leaving it fragile; the multiplicative gate supplies the necessary rescaling directly, keeping the training stable (Qiu et al., 2025b; 2026).



(a) Training loss.



(b) Pre-clip gradient norm and sliding window std.



(c) Residual maximum over training.

Figure 13: **The early phase of Qwen3.8-Flash-Next, at the shipped learning rate.** The first 276B tokens of three runs that share data order, learning-rate schedule, and optimizer: Qwen3.5 with Muon, the same plus the gated residual, and Qwen3.8-Flash-Next. The shaded band corresponds to the learning-rate warmup. The inset in (a) resolves the last 126B tokens of the window; the inset in (b) is the standard deviation of the gradient norm inside a rolling 1000-step window.

Verification at the Production Run. While the stress test amplifies instabilities by leaving the shipped configuration, we also verify that the stability benefits persist under the actual production training configuration. We compare the first 276B tokens of three runs sharing the same data order, learning-rate schedule, and optimizer: the Qwen3.5 structure with Muon, the same structure plus GR, and the full Qwen3.8-Flash-Next recipe with the further refined GR and the n-gram embedding layer.

On loss (Fig. 13a), adding GR lowers the loss at 276B tokens by 0.026, and the full Flash-Next recipe lowers it by a further 0.032, for a total gain of 0.058 over the Muon baseline. This loss improvement translates into significant benchmark gains (§4), enabling Qwen3.8-Flash-Next to reach pre-training results comparable to Qwen3.7-Plus at roughly a ninth of the training cost.

On gradient norm (Fig. 13b), Muon on its own has roughly twice the median norm and $4.2\times$ the $p_{99.9}$ of either gated run ($0.097/0.298$ vs. $0.053/0.071$ and $0.043/0.066$), and is the only run to cross the clipping threshold. The gated runs are also steadier, with 4.3–$4.7\times$ lower standard deviation inside a 1000-step window, reproducing the stress-test finding at $8\times$ the model scale and at the production learning rate. Fusing the residual read and the final normalization before the LM head into one gated read operation further reduces the gradient norm, likely the main contributor to the gap between Flash-Next and the Muon + GR configuration. On activations (Fig. 13c), adding GR markedly reduces the residual maximum throughout the network, consistent at every probed depth. This allows stable training without explicit activation control such as qk-clip (Kimi Team, 2025) and SwiGLU-clip (Agarwal et al., 2025).

4 Evaluation

We evaluate Qwen3.8-Flash-Next-Base against a set of strong base models across a broad range of capabilities, including general knowledge, reasoning, mathematics, scientific knowledge, coding, and multilingual understanding. The evaluation covers 14 benchmarks:

- **General Tasks**: MMLU (Hendrycks et al., 2021a) (5-shot), MMLU-Pro (Wang et al., 2024) (5-shot, CoT), MMLU-Redux (Gema et al., 2024) (5-shot), BBH (Suzgun et al., 2023) (3-shot, CoT), SuperGPQA (Du et al., 2025) (5-shot, CoT).
- **Math & STEM Tasks**: GPQA (Rein et al., 2024) (5-shot, CoT), GSM8K (Cobbe et al., 2021) (4-shot, CoT), MATH (Hendrycks et al., 2021b) (4-shot, CoT).
- **Coding Tasks**: EvalPlus (Liu et al., 2023) (0-shot; the average over HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021), HumanEval+ and MBPP+), MultiPL-E (Cassano et al., 2023) (0-shot; Python, C++, Java, PHP, TypeScript, C#, Bash, JavaScript), SWEBench-Pretrain, a pre-training variant of SWE-bench (Jimenez et al., 2024).
- **Multilingual Tasks**: MGSM (Shi et al., 2022) (8-shot, CoT), MMMLU (OpenAI, 2024) (5-shot), INCLUDE (Romanou et al., 2024) (5-shot).

Table 11: Comparison among the base models of Qwen3.8-Flash-Next, Qwen3.8-27B and Qwen3.7-Plus. The highest and second-best scores are shown in **bold** and underlined, respectively.

	Qwen3.8-Flash-Next-Base	Qwen3.8-27B-Base	Qwen3.7-Plus-Base
# Params	125B	27B	397B
# Activated Params	6B	27B	17B
# N-gram Embedding Params	51B	–	–
General Tasks			
MMLU	90.36	87.51	**90.43**
MMLU-Redux	90.68	87.26	**91.47**
MMLU-Pro	**73.23**	68.60	70.90
SuperGPQA	**51.36**	44.86	48.42
BBH	**90.87**	89.56	89.41
Math & STEM Tasks			
GPQA	51.42	45.01	**51.52**
GSM8K	**93.29**	93.18	92.95
MATH	72.78	60.54	**74.38**
Coding Tasks			
EvalPlus	**78.76**	76.05	78.06
MultiPL-E	79.09	74.50	**81.68**
SWEBench-Pretrain	**50.99**	41.66	49.24
Multilingual Tasks			
MGSM	**89.33**	86.37	85.42
MMMLU	**84.86**	79.74	84.53
INCLUDE	78.40	74.37	**78.90**

Tab. 11 compares Qwen3.8-Flash-Next-Base with two strong baselines, Qwen3.8-27B-Base and Qwen3.7-Plus-Base. Qwen3.8-Flash-Next-Base consistently outperforms Qwen3.8-27B-Base across all 14 benchmarks, demonstrating gains across general knowledge, reasoning, mathematics, coding, and multilingual capabilities. More notably, it outperforms the much larger Qwen3.7-Plus-Base on 8 of 14 benchmarks while remaining competitive on the others. These results are achieved with only about 1/3 of the activated parameters and 1/3 of the training tokens, corresponding to roughly 1/9 of the training FLOPs.

Beyond training efficiency, the architectural improvements of Qwen3.8-Flash-Next, together with its substantially smaller number of activated parameters, also lead to significantly lower inference cost. Overall, Qwen3.8-Flash-Next-Base delivers a substantially better performance-efficiency trade-off in both training and inference.

5 Conclusion

We have described the architecture of Qwen3.8-Flash-Next and the ablations that selected it. The resulting model retains the quality of the previous generation's 397B-A17B flagship while activating a third of the parameters, training on a third of the tokens, and consuming roughly a ninth of the FLOPs.

The design reflects a conviction that architecture, efficiency, and optimization form one coupled system. GR supplies a rescaling that markedly improves training stability, and that stability margin shifts the optimal learning rate and batch size upward, improving both throughput and convergence. Phase-by-phase cost accounting directed the sparse-attention design into the indexer and concentrated the gated residual's expressiveness on the read. Removing any axis from the loop would have admitted seemingly harmless shortcuts: sparse writes that degrade after post-training, positional encoding that appears dispensable during pre-training, or a batch-size warmup that costs extra optimizer steps for no gain.

Equally important is how these decisions were validated. Every claim was tested at a scale where the full evaluation budget remains tractable, and the settings were designed to surface the failure modes of production training: the stress test holds the learning rate at multiples of its optimal value so that instabilities emerge within a moderate budget, and scaling-law predictions are verified at the regime where each is most sensitive. Where pre-training metrics agreed but late-stage evaluation diverged, the joint protocol caught the discrepancy before it reached production. Looking forward, the tightest bottleneck is evaluation throughput: *a cheaper mid-scale probe that reliably predicts post-training ordering would make the design space far more searchable.*

6 Authors

Core Contributors: Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bo Zheng✉, Dayiheng Liu✉

Contributors[1]: Bochao Mao, Chengruidong Zhang, Fan Zhou, Hao Luo, Haofeng Huang, Haoran Lian, Haoyan Huang, Hongqing Chen, Jianwei Zhang, Jing Xu, Junjie Wang, Langshi Chen, Liangyu Wang, Linlang Jiang, Man Yuan, Minmin Sun, Peng Jin, Siqi Zhang, Siyu Wang, Xingzhang Ren, Yakai Wang, Yi Zhang, Yiming Dong, Yizhong Cao, Yubo Ma, Yunfei Mao

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Noah Amsel, David Persson, Christopher Musco, and Robert M. Gower. The polar express: Optimal matrix sign methods and their application to the muon algorithm. *arXiv preprint arXiv:2505.16932*, 2025.

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

Yushi Bai, Qian Dong, Ting Jiang, Xin Lv, Zhengxiao Du, Aohan Zeng, Jie Tang, and Juanzi Li. Indexcache: Accelerating sparse attention via cross-layer index reuse. *arXiv preprint arXiv:2603.12201*, 2026.

Cenk Baykal, Dylan Cutler, Nishanth Dikkala, Nikhil Ghosh, Rina Panigrahy, and Xin Wang. Alternating updates for efficient transformers. *arXiv preprint arXiv:2301.13310*, 2023.

✉Corresponding authors.

[1]Alphabetical order.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems* (*NeurIPS*), 2020. arXiv:2005.14165.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, et al. Multipl-e: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*, 49(7):3675–3691, 2023.

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

Yilong Chen, Yanxi Xie, Zitian Gao, Xin He, Yihao Xiao, Jason Klein Liu, Haoming Luo, Yifan Luo, Zhengmao Ye, Tingwen Liu, Xin Zhao, Ran Tao, and Bryan Dai. Beyond N-gram: Data-aware X-GRAM extraction for efficient embedding parameter scaling. *arXiv preprint arXiv:2604.21724*, 2026.

Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Han Zhang, Yu-Kun Li, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. Conditional memory via scalable lookup: A new axis of sparsity for large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics* (*Volume 1: Long Papers*), 2026.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. PaLM: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24, 2023. arXiv:2204.02311.

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

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, et al. Scaling vision transformers to 22 billion parameters. In *Proceedings of the 40th International Conference on Machine Learning* (*ICML*), 2023. arXiv:2302.05442.

Xinrun Du, Yifan Yao, Kaijing Ma, Bingli Wang, Tianyu Zheng, King Zhu, Minghao Liu, Yiming Liang, Xiaolong Jin, Zhenlin Wei, et al. SuperGPQA: Scaling LLM evaluation across 285 graduate disciplines. *arXiv preprint arXiv:2502.14739*, 2025.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Essential AI. Practical efficiency of Muon for pretraining. *arXiv preprint arXiv:2505.02222*, 2025.

Yizhao Gao, Zhichen Zeng, Dayou Du, Shijie Cao, Peiyuan Zhou, Jiaxing Qi, Junjie Lai, Hayden Kwok-Hay So, Ting Cao, Fan Yang, et al. Seerattention: Learning intrinsic sparse attention in your llms. *arXiv preprint arXiv:2410.13276*, 2024.

Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, Alessio Devoto, Alberto Carlo Maria Mancino, Rohit Saxena, Xuanli He, Yu Zhao, Xiaotang Du, Mohammad Reza Ghasemi Madani, et al. Are we done with MMLU? *CoRR*, abs/2406.04127, 2024.

Gemma Team. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

GLM-5-Team. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Google DeepMind. Gemma 3n model overview, 2025.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

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)*, 2016. arXiv:1512.03385.

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

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

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. arXiv:1608.06993.

Hongzhi Huang, Defa Zhu, Banggu Wu, Yutao Zeng, Ya Wang, Qiyang Min, and Xun Zhou. Overtokenized transformer: Vocabulary is generally worth scaling. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

W. Ronny Huang, Tara N. Sainath, Cal Peyser, Shankar Kumar, David Rybach, and Trevor Strohman. Lookup-table recurrent language models for long tail speech recognition. In *Interspeech 2021*, 2021.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.06770.

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

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

Kimi Team. Kimi K2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

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

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025a.

Hong Liu, Jiaqi Zhang, Chao Wang, Xing Hu, Linkun Lyu, Jiaqi Sun, Xurui Yang, Bo Wang, Fengcun Li, Yulei Qian, Lingtong Si, Yerui Sun, Rumei Li, Peng Pei, Yuchen Xie, and Xunliang Cai. Scaling embeddings outperforms scaling experts in language models. *arXiv preprint arXiv:2601.21204*, 2026.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. *arXiv preprint arXiv:2305.01210*, 2023.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, Mengnan Dong, Zheng Zhang, Yongsheng Kang, Hao Zhang, Xinran Xu, Yutao Zhang, Yuxin Wu, Xinyu Zhou, and Zhilin Yang. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025b.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect, 2024. URL https://arxiv.org/abs/2403.03853.

OpenAI. Multilingual massive multitask language understanding (mmmlu), 2024. Dataset available at Hugging Face.

OpenAI. OpenAI MRCR: Long context multiple needle in a haystack benchmark. https://huggingface.co/datasets/openai/mrcr, 2025. Dataset, initially released April 12, 2025.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning* (*ICML*), 2013. arXiv:1211.5063.

Zihan Qiu, Zeyu Huang, Bo Zheng, Kaiyue Wen, Zekun Wang, Rui Men, Ivan Titov, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Demons in the detail: On implementing load balancing loss for training specialized mixture-of-expert models, 2025a. URL https://arxiv.org/abs/2501.11873.

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

Zihan Qiu, Zeyu Huang, Kaiyue Wen, Peng Jin, Bo Zheng, Yuxin Zhou, Haofeng Huang, Zekun Wang, Xiao Li, Huaqing Zhang, Yang Xu, Haoran Lian, Siqi Zhang, Rui Men, Jianwei Zhang, Ivan Titov, Dayiheng Liu, Jingren Zhou, and Junyang Lin. A unified view of attention and residual sinks: Outlier-driven rescaling is essential for transformer training, 2026. URL https://arxiv.org/abs/2601.22966.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

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

Angelika Romanou, Negar Foroutan, Anna Sotnikova, Zeming Chen, Sree Harsha Nelaturu, Shivalika Singh, Rishabh Maheshwary, Micol Altintas, Alham Fikri Aji, Fahim Faisal, et al. INCLUDE: Evaluating multilingual language understanding with regional knowledge. *arXiv preprint arXiv:2411.19799*, 2024.

Aurko Roy, Rohan Anil, Guangda Lai, Benjamin Lee, Jeffrey Zhao, Shuyuan Zhang, Shibo Wang, Ye Zhang, Shen Wu, Rigel Swavely, Tao Yu, Phuong Dao, Christopher Fifty, Zhifeng Chen, and Yonghui Wu. N-Grammer: Augmenting transformers with latent n-grams. *arXiv preprint arXiv:2207.06366*, 2022.

RWKV Community. RWKV-V8's DeepEmbed. https://wiki.rwkv.com/basic/architecture.html#rwkv-v8-s-deepembed, 2025. Accessed: 2026-08-20.

Ranajoy Sadhukhan, Sheng Cao, Harry Dong, Changsheng Zhao, Attiano Purpura-Pontoniere, Yuandong Tian, Zechun Liu, and Beidi Chen. STEM: Scaling transformers with embedding modules. *arXiv preprint arXiv:2601.10639*, 2026.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International conference on machine learning*, pp. 9355–9366. PMLR, 2021.

Seed. Virtual width networks. *arXiv preprint arXiv:2511.11238*, 2025.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, et al. Language models are multilingual chain-of-thought reasoners. *arXiv preprint arXiv:2210.03057*, 2022.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015. Presented at the ICML 2015 Deep Learning Workshop.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. *arXiv preprint arXiv:2405.05254*, 2024.

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

Kimi Team, Guangyu Chen, Yu Zhang, Jianlin Su, Weixin Xu, Siyuan Pan, Yaoyu Wang, Yucheng Wang, Guanduo Chen, Bohong Yin, 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.

Albert Tseng and Christopher De Sa. L^3: Large lookup layers. In *Forty-third International Conference on Machine Learning*, 2026.

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

Liangyu Wang, Siqi Zhang, Junjie Wang, Yiming Dong, Bo Zheng, Zihan Qiu, Shengkun Tang, Di Wang, Rui Men, and Dayiheng Liu. Canzona: A unified, asynchronous, and load-balanced framework for distributed matrix-based optimizers, 2026a. URL https://arxiv.org/abs/2602.06079.

Yixuan Wang, Huang He, Siqi Bao, Haifeng Wang, Qingfu Zhu, Wanxiang Che, et al. Proxyattn: Guided sparse attention via representative heads. In *International Conference on Learning Representations*, volume 2026, pp. 18603–18617, 2026b.

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

Mitchell Wortsman, Peter J. Liu, Lechao Xiao, Katie Everett, Alex Alemi, Ben Adlam, John D. Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, Jeffrey Pennington, Jascha Sohl-Dickstein, Kelvin Xu, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Small-scale proxies for large-scale transformer training instabilities. *arXiv preprint arXiv:2309.14322*, 2023.

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. *arXiv preprint arXiv:2512.24880*, 2025.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the transformer architecture. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, 2020. arXiv:2002.04745.

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

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

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *Proceedings of Machine Learning and Systems*, 7, 2025.

Da Yu, Edith Cohen, Badih Ghazi, Yangsibo Huang, Pritish Kamath, Ravi Kumar, Daogao Liu, and Chiyuan Zhang. Scaling embedding layers in language models. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Hanlin Zhang, Depen Morwani, Nikhil Vyas, Jingfeng Wu, Difan Zou, Udaya Ghai, Dean Foster, and Sham Kakade. How does critical batch size scale in pre-training? *arXiv preprint arXiv:2410.21676*, 2024.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Xiangdong Zhang, Xiaohan Qin, Sunan Zou, Tuo Dai, Xiaoming Shi, Huaijin Wu, Yebin Yang, Zhuo Xia, Shaofeng Zhang, Lin Yao, Yuliang Liu, Yu Cheng, and Junchi Yan. xhc: Expanded hyper-connections, 2026. URL https://arxiv.org/abs/2607.14530.

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

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-connections. *arXiv preprint arXiv:2409.19606*, 2024.