Where Should Optimizer State Live? Tiered State Allocation for Memory-Efficient Mixture-of-Experts Training

Nuemaan Malik
Independent Researcher
nuemaan.research@gmail.com

Abstract

Optimizer state is the largest single line item in the memory budget of mixture-of-experts (MoE) training: on a 6.78B-parameter MoE language model, AdamW keeps 50.6 GB of first and second moments to update 12.6 GB of bfloat16 weights. We study *SkewAdam*, an optimizer built on the observation that the three parameter populations of an MoE—the dense backbone, the experts, and the router—differ enough in size and gradient statistics that they should not receive the same state. SkewAdam keeps float32 momentum plus a factored second moment for the backbone (5% of parameters), a factored second moment alone for the experts (95%), and an exact second moment for the router (<0.01%). The resulting state occupies 1.29 GB, 2.6% of AdamW's, and peak training memory falls from 81.4 GB to 31.3 GB, within the budget of a 40 GB accelerator. In a controlled comparison from identical initializations over 82M tokens, SkewAdam reaches validation perplexity 108.4, ahead of AdamW (126.8), Muon (120.2), and Lion (393.7), and settles router load balance to within 1% of its uniform floor. The allocation is not what earns that perplexity: a tier ablation matches it with twenty times the state, and Adafactor, which shares the factored estimator but drops momentum, plateaus 40 points behind. The tiers buy memory at no cost to accuracy—the accuracy comes from keeping momentum, which a uniform optimizer shares too. Sweeping the baselines' learning rates narrows but does not close the gap: the best tuned AdamW reaches 118.5, tuned Adafactor 139.7. Where optimizer state lives, these results suggest, matters at least as much as how much of it there is.

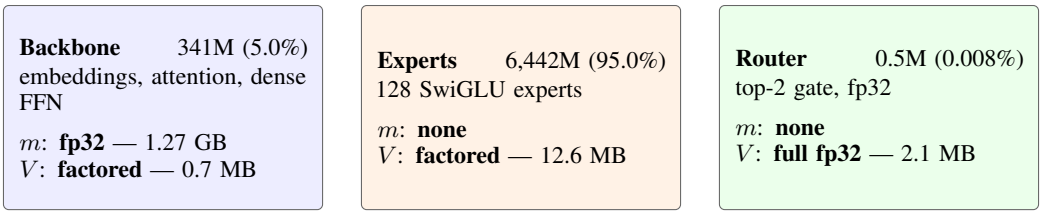
1 Introduction

Sparse mixture-of-experts (MoE) architectures decouple parameter count from per-token compute [Shazeer et al., 2017, Fedus et al., 2022, Jiang et al., 2024]: the 6.78B-parameter model we study activates roughly 440M parameters per token. Optimizer state enjoys no such discount. AdamW [Kingma and Ba, 2015, Loshchilov and Hutter, 2019] keeps two float32 moments for every parameter—8 bytes of state shadowing every 2-byte bfloat16 weight—so our model carries 50.6 GB of optimizer state on top of 12.6 GB of weights, and training peaks at 81.4 GB.[1] The component that owns most of those parameters, the expert bank, is also the one whose individual parameters are touched least often.

Memory-efficient optimizers exist, but they treat the network as a homogeneous block. Lion [Chen et al., 2023] keeps one momentum buffer and updates with its sign, halving state but discarding gradient magnitude everywhere, including in the router, where relative magnitudes carry the load-balancing signal. Muon [Jordan et al., 2024] also keeps a single buffer and orthogonalizes each

[1] Throughout, GB denotes GiB (2^{30} bytes), matching `torch.cuda.max_memory_allocated`.

Preprint.



Total optimizer state: **1.29 GB** — 2.6% of AdamW's 50.55 GB on the same model.

Figure 1: Tiered state allocation on our 6.78B-parameter MoE (m: momentum buffer, V: second-moment estimate). Each ingredient of Adam is kept only where it earns its memory: momentum where gradients are dense, factored variance where parameters are plentiful but sparsely excited, and exact variance where 2 MB buys per-logit adaptivity for the gate.

update by Newton–Schulz iteration, a structural prior designed for dense hidden layers. Adafactor [Shazeer and Stern, 2018] factors the second moment into row and column statistics, which suits a 16.8M-parameter expert matrix well but applies the same recipe to the 0.5M-parameter router that decides where every token goes. None of these methods asks whether different parts of an MoE deserve different state.

SkewAdam does. It allocates optimizer state by tier (Figure 1). The dense backbone—embeddings, attention, the dense feed-forward block—holds 5% of parameters and sees every token, so float32 momentum is cheap there and earns its keep; it also gets a factored second moment. The experts hold 95% of parameters, but under top-2 routing over 128 experts each one processes on average $1/64$ of the tokens; they keep only a factored second moment, which removes the single largest state cost in the model. The router keeps an exact, unfactored second moment: per-logit adaptivity costs 2 MB and steers all the traffic. The name is the design—the state budget is skewed toward where it pays.

We make three contributions. First, we describe the tiered allocation policy and its closed-form memory model: 1.29 GB of optimizer state on a 6.78B-parameter MoE, with peak training memory of 31.3 GB. Second, we run a controlled single-GPU comparison against AdamW, Lion, and Muon—identical initialization, identical data order, and an identical bfloat16 weight-update path with dithered stochastic rounding for all four—in which SkewAdam attains the best validation perplexity (108.4) at a throughput within 1.5% of the fastest baseline; same-protocol follow-ups on two further GPUs add Adafactor and a GaLore-style baseline, ablate the policy's tiers, and sweep the baselines' learning rates: the policy matches full-momentum perplexity at a twentieth of the state, and its lead over the baselines survives their tuning. Third, we analyze routing stability by measuring the load-balancing loss against its analytic floor, finding that Lion's magnitude-blind updates cost perplexity rather than balance, and that Muon drifts away from balance late in training.

2 Related work

Memory-efficient optimizers. Adafactor [Shazeer and Stern, 2018] is the closest relative of this work. The factored estimator inside SkewAdam is Adafactor's nonnegative rank-one estimator written with row and column means instead of sums, and the update-RMS clipping is Adafactor's as well; we claim no novelty for either. What we take up is the allocation question that uniform recipes leave open: Adafactor either drops momentum everywhere or keeps it everywhere, and factors every matrix regardless of its role. SM3 [Anil et al., 2019] and 8-bit optimizers [Dettmers et al., 2022] compress state by other means, and ZeRO [Rajbhandari et al., 2020] shards it across devices rather than shrinking it; all are compatible with, and orthogonal to, a tiered policy. GaLore [Zhao et al., 2024] projects gradients into a low-rank subspace before applying Adam, again uniformly across matrices.

Sign- and geometry-based updates. Lion [Chen et al., 2023] reduces state to a single buffer by updating with the sign of an interpolated momentum. Muon [Jordan et al., 2024] replaces adaptive scaling with Newton–Schulz orthogonalization of the momentum and has been scaled to large dense and MoE models [Liu et al., 2025]; its authors route embeddings and other non-matrix parameters

to an Adam-style rule, which our implementation follows. Both methods are uniform over hidden matrices, expert or not.

MoE training. Sparsely gated MoE layers were introduced by Shazeer et al. [2017] and scaled by GShard [Lepikhin et al., 2021] and Switch Transformers [Fedus et al., 2022], whose load-balancing auxiliary loss we use. The router z-loss follows ST-MoE [Zoph et al., 2022], which documents how fragile router training can be; Mixtral [Jiang et al., 2024] and DeepSeekMoE [Dai et al., 2024] are recent open systems. This literature concentrates on architecture and losses; optimizer state allocation for MoE has received little direct attention.

Low-precision training. Mixed-precision practice keeps float32 master weights [Micikevicius et al., 2018]; pure-bfloat16 training is attractive for memory but loses small updates to rounding [Kalamkar et al., 2019]. Stochastic rounding repairs much of the damage [Gupta et al., 2015, Zamirai et al., 2021]. We train with bfloat16 master weights and a dithered approximation to stochastic rounding (Section 3), applied identically under every optimizer we compare.

3 SkewAdam: tiered state allocation

Factored second moments. For a weight matrix $W \in \mathbb{R}^{n\times m}$ with gradient G_t, SkewAdam maintains exponential moving averages of the row-wise ($\mathrm{mean_c}$, over columns) and column-wise ($\mathrm{mean_r}$, over rows) mean squared gradient,

$$R_t = \beta_2 R_{t-1} + (1-\beta_2)\,\mathrm{mean_c}(G_t \odot G_t) \in \mathbb{R}^n, \quad C_t = \beta_2 C_{t-1} + (1-\beta_2)\,\mathrm{mean_r}(G_t \odot G_t) \in \mathbb{R}^m, \tag{1}$$

and reconstructs the second-moment matrix as the rank-one estimate $\widehat{V}_t = R_t C_t^\top / \bar{R}_t$, where $\bar{R}_t$ is the mean of R_t. This is exact whenever the true second-moment matrix has rank one and coincides with Adafactor's estimator [Shazeer and Stern, 2018]. Storage falls from nm to $n + m$ floats per matrix; for a 4096×4096 expert matrix, from 64 MB to 32 KB.[2]

The tiers. The policy assigns state by parameter role (Figure 1, Algorithm 1). *Backbone* tensors (embeddings, attention projections, the dense feed-forward block, norms) carry float32 momentum and a factored second moment. Their gradients are dense—every token contributes every step—so momentum smooths a signal that is actually there, and at 5% of parameters the buffer costs 1.27 GB. *Expert* tensors carry only the factored second moment. They are 95% of parameters, each expert sees roughly $1/64$ of tokens under top-2-of-128 routing, and a momentum buffer here would cost 24 GB to smooth gradients that arrive sparsely and with high variance; Adafactor's results suggest momentum can be dropped without losing adaptivity [Shazeer and Stern, 2018]. *Router* weights keep a full, unfactored second moment and no weight decay. Factoring a 128×4096 gate would pool scale statistics across the very logits whose relative magnitudes determine load balance; exactness here costs 2 MB. The policy is a judgment about where each ingredient of Adam earns its memory, not a theorem; Section 5 tests it.

Update clipping and low-precision updates. The preconditioned update is clipped to unit root-mean-square, $U \leftarrow U/\max(1, \mathrm{RMS}(U))$, Adafactor's update clipping with threshold 1. Master weights are bfloat16. Each step is computed in float32 and written back through a dithered rounding: uniform noise of one-ULP width, $\xi \sim \mathcal{U}(-\mathrm{ulp}(w)/2,\ \mathrm{ulp}(w)/2)$ with $\mathrm{ulp}(w) \approx 2^{-7}|w|$, is added before the cast, approximating unbiased stochastic rounding [Gupta et al., 2015, Zamirai et al., 2021]. Every optimizer in our comparison uses this same write-back path, so none is privileged by precision handling.[3]

Memory model. Let N_{bb}, N_{ex}, N_{rt} be the tier sizes. SkewAdam's state is

$$S = 4\big[N_{\mathrm{bb}} + \textstyle\sum_{W \in \mathrm{bb}\cup\mathrm{ex}}(n_W + m_W) + N_{\mathrm{rt}}\big] \text{ bytes} \approx 4\,N_{\mathrm{bb}}, \tag{2}$$

[2] Stacked expert tensors of shape (E, n, m) factor along the last two axes; in our model the experts are separate matrices, so the two-dimensional path is the one exercised.

[3] A side effect we accept deliberately: with $\eta = 3 \times 10^{-4}$ and $\lambda = 0.05$, the decoupled weight-decay step changes weights by 1.5×10^{-5} relative—more than two orders of magnitude below the bfloat16 ULP of 2^{-7}—so weight decay rounds to a no-op in *all* runs. The comparison is fair but effectively unregularized; see Section 6.

Algorithm 1 SkewAdam step for one tensor W in tier τ (β_1=0.9, β_2=0.999, ϵ=10^{-8})

$G \leftarrow \nabla_W \mathcal{L}$ in float32
if $\tau =$ ROUTER **or** W is a vector **then**
  $V \leftarrow \beta_2 V + (1-\beta_2)\, G \odot G; \quad D \leftarrow \sqrt{\max(V, \epsilon)}$ ▷ full second moment
else
  update R, C by Eq. (1); $\quad D \leftarrow \sqrt{R C^\top / \bar{R}}$ ▷ factored, with ϵ-clamps
end if
if $\tau =$ BACKBONE **then**
  $M \leftarrow \beta_1 M + (1-\beta_1)\, G; \quad U \leftarrow \dfrac{\sqrt{1-\beta_2^t}}{1-\beta_1^t}\, M \oslash D$ ▷ fp32 momentum
else
  $U \leftarrow \sqrt{1-\beta_2^t}\; G \oslash D$ ▷ no momentum buffer
end if
$U \leftarrow U / \max\big(1, \mathrm{RMS}(U)\big)$ ▷ update clipping
$W \leftarrow \mathrm{bf16}\big(W_{\mathrm{fp32}} - \eta_t\, U + \xi\big), \quad \xi \sim \mathcal{U}\big(\pm\, \mathrm{ulp}(W)/2\big)$ ▷ dithered rounding

since the factored vectors are negligible. With $N_{\mathrm{bb}} = 341.4$M, $N_{\mathrm{ex}} = 6{,}442.5$M, $N_{\mathrm{rt}} = 0.52$M this gives 1.29 GB (a component-level account is in Appendix B), against $8N_{\mathrm{total}} = 50.55$ GB for AdamW. The saving is structural: it does not depend on quantization and would compound with it.[4]

4 Experimental setup

Model. We train a decoder-only transformer with two blocks: the first uses a dense SwiGLU feed-forward layer [Shazeer, 2020], the second an MoE layer with 128 SwiGLU experts of hidden width 4096 and top-2 routing. Width is 4096 throughout, with grouped-query attention (32 query heads, 8 KV heads) [Ainslie et al., 2023], learned positions, a GPT-2 BPE vocabulary padded to 50,304 [Radford et al., 2019], and tied embeddings; 6,784M parameters in total, about 440M active per token. Two blocks is shallow by intent as well as by budget: it concentrates 95% of parameters in a single expert bank, the population whose optimizer state we want to stress, and it lets a 6.78B-parameter, four-optimizer comparison run on one GPU. The router operates in float32 with input noise $\sigma = 0.5$ at training time and a zero-initialized gate; we use the Switch-style balancing loss with $\alpha = 0.05$ [Fedus et al., 2022] and a z-loss of 10^{-4} [Zoph et al., 2022]. LayerNorms are kept in float32.

Data and protocol. We stream OpenWebText [Gokaslan and Cohen, 2019], hashing each document into a 95/5 train/validation split so the two sides share no documents. Each run takes 10,000 steps at batch size 64 $\times$ 128 tokens (81.9M tokens, single epoch; no batch repeats). Validation uses the same 64 held-out batches (≈0.5M tokens) for every optimizer. All four optimizers start from one shared initialization and consume identical batches in identical order, under bfloat16 autocast with per-block activation checkpointing [Chen et al., 2016] and gradient clipping at 1.0. Learning rates follow a cosine schedule with 3% warmup: 3×10^{-4} for AdamW and SkewAdam, 1×10^{-4} for Lion (the 3–10$\times$ reduction its authors recommend), and 0.02 for Muon's matrices with an internal Adam at 10^{-3} for embeddings, router, and vector parameters. Runs execute on a single NVIDIA H200; its 141 GB simply lets us include the AdamW baseline that a 40 GB device could not hold. Peak memory is read from CUDA's peak-allocation counter. Full hyperparameters are in Appendix A.

5 Results

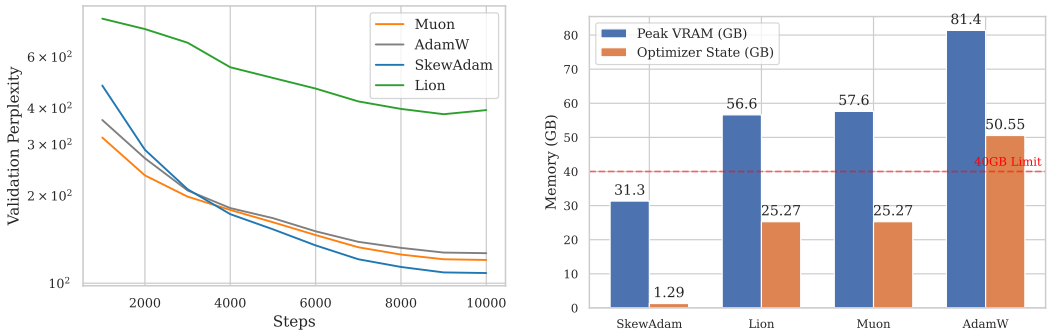
Memory and throughput. Table 1 and Figure 2b give the budget. AdamW peaks at 81.4 GB, of which 50.6 GB is optimizer state; the rest is bfloat16 weights (12.6 GB), gradients (12.6 GB), and

[4] Quantized state also carries a scale ceiling that factoring avoids. Current 8-bit optimizer kernels terminate the process—a hard exit from C++, not an exception—once a single parameter tensor reaches 2^{31} elements, a size that fused expert weights reach in larger MoE models (bitsandbytes issue 1785). We measured the boundary on an A100: the 8-bit step succeeds at $2^{31} - 1$ elements and kills the process at 2^{31}, while factored state, built from native PyTorch ops with 64-bit indexing, crosses it cleanly. Scripts and logs are in the repository's `experiments/` directory.

Table 1: 6.78B-parameter MoE, 10,000 steps from a shared initialization. State sizes are analytic (Appendix B); memory and throughput are measured at the final step. Balance loss has a floor of 0.05 under uniform routing.

Optimizer	State (GB)	Peak mem. (GB)	Tokens/s	Val. PPL ↓	Balance loss
SkewAdam	**1.29**	**31.3**	5,000	**108.4**	0.0505
AdamW	50.55	81.4	4,692	126.8	**0.0502**
Muon	25.27†	57.6	3,409	120.2	0.0608
Lion	25.27	56.6	**5,075**	393.7	0.0537

† One float32 buffer per parameter; the Adam-style state Muon keeps for embeddings, router, and vector parameters adds roughly 0.8 GB.



(a) Validation perplexity (log scale).

(b) Peak memory and optimizer state.

Figure 2: Convergence and memory. AdamW and Muon lead through step 3,000; SkewAdam overtakes both by step 4,000 and finishes 14.5% below AdamW. Only SkewAdam clears the 40 GB line.

activations. Lion and Muon halve the state and still peak near 57 GB, above the 40 GB class of accelerators. SkewAdam's 1.29 GB of state brings the peak to 31.3 GB, with headroom to spare on a 40 GB device. Throughput tracks state traffic: SkewAdam sustains 5,000 tokens/s, 6.6% above AdamW, and 1.5% below Lion, whose single buffer is the cheapest to maintain. Muon pays 32% relative to SkewAdam for running Newton–Schulz iterations over 6.4B expert parameters every step.

Convergence. AdamW and Muon converge faster for the first 3,000 steps—at step 1,000 SkewAdam trails AdamW by 114 points of perplexity—but SkewAdam passes both by step 4,000 and ends at 108.4 against 120.2 (Muon) and 126.8 (AdamW); per-step values are tabulated in Appendix C. Lion ends at 393.7, more than three times SkewAdam, having peaked at 381.1 at step 9,000 and worsened thereafter. Lion does not recover. We read the Lion result as consistent with the magnitude-blindness account—a fixed-magnitude step treats a heavily routed expert and a rarely routed one identically—while noting it reflects one learning rate and one seed. That SkewAdam ends *below* AdamW is the more surprising outcome, and we offer an interpretation rather than a claim: with ≈128 tokens reaching each expert per step, Adam's per-coordinate second moments are estimated from little data, while the factored estimator pools statistics over 4096 coordinates per row and column; the pooled preconditioner, plus RMS clipping of occasional large updates, may simply be better conditioned at this routing sparsity.

Adafactor and GaLore. The comparison this method most owes its readers is against Adafactor, which supplies the factored estimator SkewAdam builds on. We ran it when compute later allowed: same code, data, seed, and shared initialization, on an H100 MIG slice (Table 2). SkewAdam, re-run in the same batch as the anchor, lands at 109.0 against its 108.4 on the H200—0.5% apart, so the protocol transfers across hardware. Uniform Adafactor carries even less state than SkewAdam (12 MB; no momentum anywhere) but plateaus at 149.5, forty perplexity points behind, its final thousand steps improving by less than 0.1. Both optimizers share the rank-one second-moment estimator and update clipping; the difference is that SkewAdam keeps momentum while Adafactor

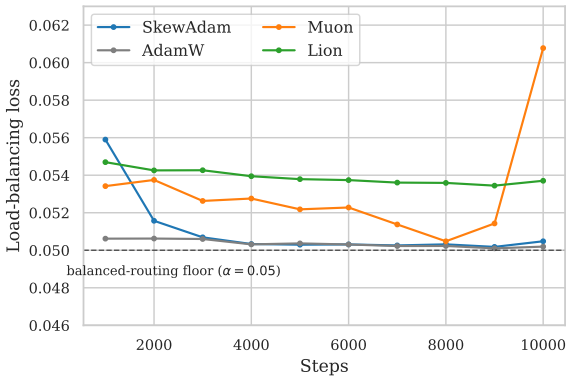


Figure 3: Load-balancing loss per evaluation step. The dashed line is the value attained under perfectly uniform routing ($\alpha = 0.05$). SkewAdam and AdamW sit within 1% of the floor from step 4,000; Muon jumps 22% above it in the final thousand steps.

Table 2: Factored and low-rank baselines under the identical protocol (same model, data, seed, and shared initialization within the batch), run on an NVIDIA H100 NVL 47 GB MIG slice. Throughput is not comparable with Table 1 (different hardware). Adafactor's state is analytic; the GaLore-style implementation's state was not instrumented.

Optimizer	State (GB)	Peak mem. (GB)	Tokens/s	Val. PPL ↓	Balance loss
SkewAdam	1.29	31.3	3,778	**109.0**	0.0505
Adafactor	**0.01**	**29.6**	4,032	149.5	**0.0502**
GaLore-style (rank 128)	—	31.7	**4,209**	1,839.9	0.0510

drops it entirely, alongside Adafactor's annealed decay.[5] The tier ablation below (Table 3) locates this gap in the momentum and its decay schedule, not the allocation: uniform allocation *with* momentum reaches the same perplexity as the tiered policy. The rank-128 GaLore-style baseline in our trainer fails outright at these settings (1,839.9), balancing router load, like Lion, while never learning the language model. We read this as a caution about projecting sparse expert gradients onto low-rank subspaces, not as a verdict on GaLore, whose reference implementation and tuning differ from ours.

Which tier does the work? Table 3 toggles the tiers one at a time, on the MI300X under the same protocol. Every variant reaches the same validation perplexity (108.2–108.9, within single-seed noise) and the same load balance (≈ 0.050); what moves twentyfold is optimizer state. Restoring momentum to the experts costs 24 GB and moves perplexity by 0.2—expert momentum is dead weight, which is exactly what the policy discards. Factoring the router changes neither perplexity nor balance, so the exact router (2 MB) is a harmless but not load-bearing choice. Uniform allocation, with momentum and factoring everywhere, matches the tiered policy at twenty times the state. The policy's contribution is therefore memory: it recovers full-momentum perplexity from momentum on the dense backbone alone, where it costs 1.27 GB rather than 25. SkewAdam itself reaches 108.9 here, matching its 108.4 (H200) and 109.0 (H100) numbers—a third platform, and a second vendor, within noise.

Tuning the baselines. The one quality claim left standing after the ablation—SkewAdam ahead of AdamW—rested on a single untuned learning rate, so we swept both strong baselines while leaving SkewAdam at its default (Table 4). Tuning is worth real perplexity to them: AdamW improves from 126.8 at 3×10^{-4} to 118.5 ± 0.5 at 10^{-4} (three seeds), Adafactor from 149.5 to 139.7 at the same rate (two seeds, 0.005 apart). Adafactor's minimum is bracketed on both sides—3×10^{-5} undertrains to 257.5 and every higher rate is worse—while AdamW's is bracketed only from above, though Adafactor's collapse at 3×10^{-5} suggests this step budget undertrains at lower rates generally. Neither tuned baseline reaches SkewAdam: untuned at 3×10^{-4}, it leads the best AdamW by ten perplexity points, twenty times the seed-level standard deviation, and the best Adafactor by thirty-

[5]HuggingFace Adafactor anneals its second-moment decay as $1 - t^{-0.8}$ rather than holding $\beta_2 = 0.999$.

Table 3: Tier ablation on an MI300X (192 GB), identical protocol, one seed per variant. Each row toggles one decision of the policy. Perplexity and load balance are flat across all four; optimizer state and peak memory are not.

Variant	State (GB)	Peak mem. (GB)	Val. PPL	Balance loss
SkewAdam (full policy)	**1.29**	**31.4**	108.9	0.0505
+ momentum on experts	25.29	55.4	108.7	0.0506
factored router	1.28	31.4	108.2	0.0505
uniform (momentum + factored)	25.29	55.4	108.3	0.0503

Table 4: Learning-rate sweeps for the two strongest baselines, run on the MI300X under the identical protocol (the Adafactor 3×10^{-4} entry is the H100 run of Table 2; the AdamW 3×10^{-4} entry reproduces the H200 baseline to within 0.5). SkewAdam is deliberately left untuned.

Optimizer	LRs swept	Best LR	Best Val. PPL
AdamW	10^{-4}, $3{\times}10^{-4}$, 10^{-3}	10^{-4}	118.5 ± 0.5 (3 seeds)
Adafactor	$3{\times}10^{-5}$, 10^{-4}, $3{\times}10^{-4}$, 10^{-3}, $3{\times}10^{-3}$	10^{-4}	139.7 (2 seeds)
SkewAdam (untuned)	—	$3{\times}10^{-4}$	108.4–109.0 (3 GPUs)

one. Tuning narrows the headline gaps; it does not close them. The persistent AdamW–Adafactor separation (118.5 vs 139.7, both tuned) is the momentum story of Table 3 again, now visible between independent optimizers.

Routing stability. The balancing loss $\alpha E \sum_i \bar{p}_i f_i$ equals $\alpha = 0.05$ exactly when both the router probabilities and the realized token assignment are uniform, which makes deviation from 0.05 a calibrated imbalance measure (Figure 3). SkewAdam and AdamW stay within 1% of the floor from step 4,000 onward (final values 0.0505 and 0.0502). Lion is stable but elevated 7–9% above the floor throughout—it balances load while failing to learn the language model. Muon improves steadily to 1–3% above floor and then, in the last thousand steps, jumps to 0.0608, 22% above. Whether this is the onset of an instability or a transient cannot be settled from one run, but no other optimizer moves by that much at any point in training, and the jump coincides with Muon's only flat segment in validation perplexity.

Zero-shot evaluation. For completeness we score the final checkpoints with the LM Evaluation Harness [Gao et al., 2023] on PIQA, WinoGrande, HellaSwag, and ARC-Challenge [Bisk et al., 2020, Sakaguchi et al., 2020, Zellers et al., 2019, Clark et al., 2018]. After 82M tokens—three to four orders of magnitude below modern budgets—all four models sit near chance on three of the four tasks and a few points above chance on PIQA (53.5–55.9%), with no separation between optimizers beyond one to two standard errors (full table in Appendix D). At this token budget the downstream numbers neither support nor undermine any optimizer; we report them so that the perplexity gains are not mistaken for more than they are.

6 Limitations

The model is two blocks deep. That choice concentrates 95% of parameters in one expert bank, which is the stress test we wanted, but it leaves untested how tiered allocation behaves when routing decisions compose across many MoE layers. Most numbers are one run per configuration. The sweeps of Table 4 addressed this where it mattered most—both strong baselines are tuned over bracketing grids with repeated seeds, and SkewAdam's own number is replicated on three GPUs—but SkewAdam was not itself tuned, AdamW was not probed below 10^{-4}, and Lion and Muon keep single untuned rates. The 82M-token horizon and 128-token contexts are small, and weight decay was inert in all runs (Section 3); anyone scaling this recipe to production horizons must reintroduce weight decay, fusing the decay term into the float32 update ahead of the stochastically rounded cast so that it survives the bfloat16 ULP, a configuration we have not yet validated at length. The Adafactor and GaLore runs of Table 2 were added when compute later became available; they share everything with the main protocol except the GPU; where configurations repeat across machines they agree closely (SkewAdam within 0.6 perplexity over three GPUs, AdamW at 3×10^{-4} within 0.5 over two). The

GaLore number in particular reflects a single untuned configuration of our own implementation and should not be read as more than that. The tier ablation (Table 3) likewise rests on one seed per variant, and its perplexity spread (0.6 across four variants) is within run-to-run noise; we read it as evidence of memory-at-parity, not of any perplexity ordering among the variants. It leaves open whether the tiers separate at all under a multi-seed protocol, or at a longer horizon where expert momentum might begin to earn its 24 GB. Finally, the downstream evaluations are near chance and should be read as a completeness check, not evidence of capability. We plan to validate the approach at larger scales as compute becomes available.

7 Conclusion

A mixture-of-experts model is not a homogeneous bag of parameters, and its optimizer need not pretend otherwise. Spending Adam's full state only on the dense backbone, factored variance on the expert bank, and an exact second moment on the router cuts optimizer state by 97.4% and peak training memory by 61% on a 6.78B-parameter MoE. A tier ablation shows this costs nothing: the policy reaches the same perplexity and routing balance as a uniform optimizer carrying twenty times the state. The contribution is memory, not a better optimizer—Adam-family quality at 2.6% of Adam's state, a comparison that survives sweeping the baselines' learning rates. The broader suggestion is a design principle rather than a single optimizer: decide where optimizer state lives tier by tier, with the gradient statistics of each tier in view.

Acknowledgments

This work was self-funded and run on rented GPU time; the compute budget, not the experimental design, set the scale of the study. The author welcomes collaboration or compute support to extend the comparison to deeper models, longer horizons, and multi-seed replication.

References

J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

R. Anil, V. Gupta, T. Koren, and Y. Singer. Memory efficient adaptive optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Y. Bisk, R. Zellers, R. Le Bras, J. Gao, and Y. Choi. PIQA: Reasoning about physical commonsense in natural language. In *AAAI Conference on Artificial Intelligence*, 2020.

T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

X. Chen, C. Liang, D. Huang, E. Real, K. Wang, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu, and Q. V. Le. Symbolic discovery of optimization algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

D. Dai, C. Deng, C. Zhao, R. X. Xu, H. Gao, D. Chen, J. Li, W. Zeng, X. Yu, Y. Wu, et al. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024.

T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer. 8-bit optimizers via block-wise quantization. In *International Conference on Learning Representations (ICLR)*, 2022.

W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, et al. A framework for few-shot language model evaluation. `https://github.com/EleutherAI/lm-evaluation-harness`, 2023.

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

S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning (ICML)*, pages 1737–1746, 2015.

A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

K. Jordan, Y. Jin, V. Boza, J. You, F. Cesista, L. Newhouse, and J. Bernstein. Muon: An optimizer for hidden layers in neural networks. `https://kellerjordan.github.io/posts/muon/`, 2024.

D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, et al. A study of BFLOAT16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations (ICLR)*, 2021.

J. Liu, J. Su, X. Yao, Z. Jiang, G. Lai, Y. Du, Y. Qin, W. Xu, E. Lu, J. Yan, et al. Muon is scalable for LLM training. *arXiv preprint arXiv:2502.16982*, 2025.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu. Mixed precision training. In *International Conference on Learning Representations (ICLR)*, 2018.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI Technical Report*, 2019.

S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He. ZeRO: Memory optimizations toward training trillion parameter models. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2020.

K. Sakaguchi, R. Le Bras, C. Bhagavatula, and Y. Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *AAAI Conference on Artificial Intelligence*, 2020.

N. Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

N. Shazeer and M. Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning (ICML)*, pages 4596–4604, 2018.

N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017.

P. Zamirai, J. Zhang, C. R. Aberger, and C. De Sa. Revisiting BFloat16 training. *arXiv preprint arXiv:2010.06192*, 2021.

R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. HellaSwag: Can a machine really finish your sentence? In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

J. Zhao, Z. Zhang, B. Chen, Z. Wang, A. Anandkumar, and Y. Tian. GaLore: Memory-efficient LLM training by gradient low-rank projection. In *International Conference on Machine Learning (ICML)*, 2024.

B. Zoph, I. Bello, S. Kumar, N. Du, Y. Huang, J. Dean, N. Shazeer, and W. Fedus. ST-MoE: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906*, 2022.

A Hyperparameters

Table 5: Complete configuration. All values are as used in the released training script.

Model		**Training**	
Width d_{model}	4096	Steps	10,000
Blocks	2 (dense FFN, MoE)	Tokens	81.9M (single epoch)
Attention heads / KV heads	32 / 8	Batch	64 seq $\times$ 128 tokens
Dense FFN hidden (SwiGLU)	4096	Microbatch	8 (8 accumulation steps)
Experts / hidden / top-k	128 / 4096 / 2	Gradient clip	1.0
Router noise (train)	$\mathcal{N}(0, 0.5^2)$	Schedule	cosine, 3% warmup
Balance coef. α / z-loss	0.05 / 10^{-4}	Seed	42
Vocabulary (GPT-2 BPE, padded)	50,304	Precision	bf16 master weights
Positions (learned, max)	256		fp32 norms and router
Dropout	0	Checkpointing	per block
Optimizers			
AdamW	$\eta = 3 \times 10^{-4}$, $\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$		
SkewAdam	$\eta = 3 \times 10^{-4}$, $\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$		
Lion	$\eta = 1 \times 10^{-4}$, $\beta = (0.9, 0.99)$		
Muon	$\eta = 0.02$, momentum 0.95, 3 NS steps; internal Adam $\eta = 10^{-3}$		
Adafactor	HF impl.; $\eta = 3 \times 10^{-4}$, no momentum, decay $1-t^{-0.8}$		
GaLore-style	rank 128, $\eta = 3 \times 10^{-4}$, $\beta = (0.9, 0.999)$; full Adam off-matrix		
Weight decay	0.05 (0 on router); inert under bf16 rounding, see Section 3		

B Memory accounting

Table 6: SkewAdam optimizer state by component. The momentum buffer of the dense backbone is, to first order, the entire footprint.

Component	Parameters	State kept	Size
Backbone momentum (fp32)	341.4M	m	1,302.2 MB
Backbone factored 2nd moment	—	R, C vectors	0.5 MB
Backbone vector params (norms)	0.04M	full v	0.2 MB
Expert factored 2nd moments	6,442.5M	R, C per matrix	12.0 MB
Router full 2nd moment	0.52M	v	2.0 MB
Total	6,784.3M		1.29 GB
AdamW on the same model	6,784.3M	m, v	50.55 GB

Parameter counts by tier: token embedding $50{,}304 \times 4096 = 206.0$M (tied with the output head), positions 1.0M, attention 83.9M, dense FFN 50.3M, norms 0.04M, for a backbone of 341.4M; experts $128 \times 3 \times 4096^2 = 6{,}442.5$M; router $128 \times 4096 = 0.52$M. Total $6{,}784.3$M, matching the trainer's logged count.

C Convergence detail

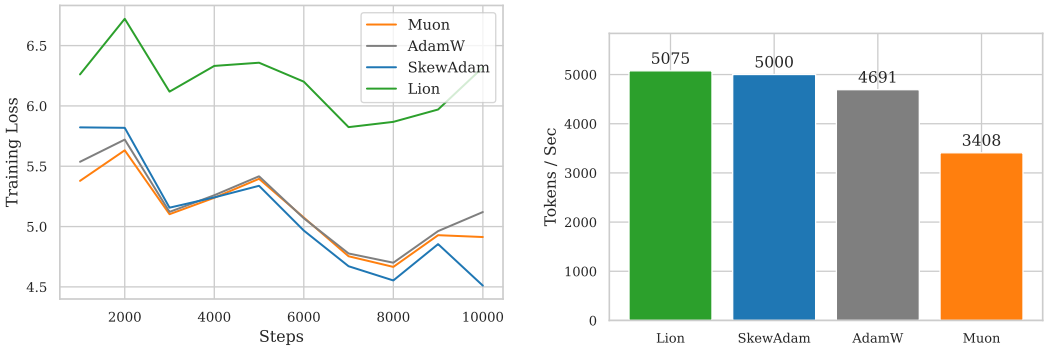
Table 7 tabulates validation perplexity at every evaluation step; Figure 4 shows the training loss traces and final sustained throughput. Muon leads for the first three thousand steps, SkewAdam from step four thousand on.

Table 7: Validation perplexity at each evaluation step. Bold marks the best optimizer at each step.

Step ($\times 10^3$)	1	2	3	4	5	6	7	8	9	10
SkewAdam	478.4	287.6	210.4	**172.8**	**153.2**	**134.9**	**120.9**	**113.7**	**108.9**	**108.4**
AdamW	364.1	268.9	208.4	181.3	167.4	150.8	138.8	132.3	127.5	126.8
Muon	**316.9**	**234.8**	**198.5**	178.6	162.0	146.4	132.9	125.5	120.9	120.2
Lion	812.0	747.8	671.2	552.7	507.9	466.9	421.7	397.4	381.1	393.7

Table 8: Validation perplexity at each evaluation step for the H100 follow-up (Table 2). Bold marks the best optimizer at each step; Adafactor leads for the first two thousand steps, SkewAdam thereafter.

Step ($\times 10^3$)	1	2	3	4	5	6	7	8	9	10
SkewAdam	477.3	293.8	**211.9**	**174.0**	**155.1**	**135.7**	**121.6**	**114.4**	**109.6**	**109.0**
Adafactor	**403.2**	**287.0**	242.7	207.3	187.1	169.2	157.8	152.4	149.6	149.5
GaLore	2703.1	2256.8	2070.4	1965.6	1901.7	1855.1	1829.0	1821.7	1826.0	1839.9



(a) Training language-model loss.

(b) Sustained throughput at step 10,000.

Figure 4: Training loss and throughput. The loss traces are single-batch measurements at evaluation steps, hence their visible variance.

D Zero-shot evaluation detail

Table 9 expands the summary in Section 5. No pairwise difference between optimizers exceeds two standard errors on any task.

Table 9: Zero-shot accuracy (%) with standard errors, LM Evaluation Harness, batch size 32. PIQA and WinoGrande report accuracy; HellaSwag and ARC-Challenge report length-normalized accuracy. Chance is 50% for the first two and 25% for the last two.

Optimizer	PIQA	WinoGrande	HellaSwag	ARC-Challenge
SkewAdam	54.7 ± 1.2	49.3 ± 1.4	25.4 ± 0.4	21.6 ± 1.2
AdamW	54.6 ± 1.2	49.4 ± 1.4	25.3 ± 0.4	22.9 ± 1.2
Muon	55.9 ± 1.2	50.4 ± 1.4	25.4 ± 0.4	22.0 ± 1.2
Lion	53.5 ± 1.2	51.1 ± 1.4	25.3 ± 0.4	22.2 ± 1.2

E Reproducibility

The main comparison ran on a single NVIDIA H200 (141 GB) using the released single-file trainer and evaluator. Code, per-step training logs, and figures are available at `https://github.com/nuemaan/skewadam`. The environment is installed with

```
pip install torch numpy transformers bitsandbytes datasets lm_eval
```

and the four reported runs come from one invocation that trains all four optimizers in sequence:

```
CUDA_VISIBLE_DEVICES=0 python train.py \
    --optimizers "skewadam,adam,lion,muon"
python evaluate.py runs/best_skewadam.pt   # likewise for the others
python plot_metrics.py
```

The trainer builds one initialization and one cached batch sequence, reseeds before each optimizer, and reuses both for all four, which is what licenses the matched-conditions claim of Section 4.

The follow-up of Table 2 ran on a different machine (NVIDIA H100 NVL, 47 GB MIG slice) under the same protocol:

```
python train.py --optimizers "adafactor,skewadam,galore" \
    --dataset-name Skylion007/openwebtext
```

The dataset flag points at the canonical parquet mirror of the same corpus (newer releases of the `datasets` library no longer accept the legacy `openwebtext` id); the document-hash split is unchanged, so the validation set is identical. Metrics and the training log for this run are kept under `runs/h100/`. The tier ablation and the learning-rate sweeps ran on an MI300X via `experiments/tier-ablation/run_ablation.py` and `experiments/lr-sweep/run_sweep.py` / `run_fix.py`; per-run metrics are under `runs/amd-ablation/` and `runs/lr-sweep/`. The trainer writes per-step metrics, analytic state sizes, and peak memory to JSON files under `runs/`; every number in this paper is read from those files or from the evaluation JSONs. The data split is deterministic (an MD5 hash of each document's first kilobyte selects train or validation), so the validation set is identical across runs and machines.