RETHINKING MEMORY-EFFICIENT OPTIMIZERS FOR LLM PRE-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

A growing family of memory-efficient optimizers reduces state storage through gradient projection, moment factorization, and shared statistics. Yet their practical value depends on when optimizer states constrain training and whether structural compression offers an advantage over simple state quantization. We examine these questions through a broad benchmark of memory-efficient methods and strong conventional optimizers under a common language-model pretraining setup. Our evaluation covers 257M and 500M dense models and extends to mixture-of-experts architectures, varying compute and state precision independently. We investigate how compression strength affects training quality and compare the approaches in terms of peak memory, validation loss at fixed token budgets, and measured training-step time. In the main 500M comparison, Muon with FP8 states achieves lower validation loss and lower peak training memory than the evaluated structural methods with FP32 states under both BF16 and FP8 computation. Quantizing the states of structural methods remains feasible, but yields limited further memory savings over quantized-state Muon without closing the quality gap. These findings support quantizing a strong optimizer as a practical starting point for memory-constrained pretraining and make competitive quantized baselines essential to assessing the benefits of structural compression. Code is available at anonymous.4open.science.

1 INTRODUCTION

Modern optimizers maintain auxiliary states such as moments, matrix statistics, and momentum buffers (Kingma & Ba, 2015; Jordan et al., 2024; Vyas et al., 2025; Pagliardini et al., 2024). These states scale with the number of model parameters and can occupy several times the parameter storage. Recent memory-efficient optimizers reduce this cost through projected gradients, factorized moments, shared statistics, or blockwise updates (Zhao et al., 2024; Chen et al., 2025; Zhu et al., 2025; Zmushko et al., 2025; Zhang et al., 2024; Kalra et al., 2025).

Smaller state tensors do not imply a proportional reduction in peak training memory, which also includes parameters, gradients, activations, and temporary workspaces. State storage is independent of microbatch size, while activation storage grows with it. Compression can therefore determine whether a small-microbatch configuration fits yet save only a few percent once activations dominate. In our 500M measurements, quantizing AdamW states from FP32 to FP8 reduces peak memory by 31.3% and 4.2% at per-device microbatch sizes of 1 and 32 sequences, respectively. Nominal state compression can thus substantially overstate the end-to-end memory benefit.

Distributed pretraining provides a separate motivation. ZeRO reduces state replication by sharding training tensors across data-parallel ranks (Rajbhandari et al., 2020). Matrix-aware optimizers require complete parameter matrices, so several distributed implementations gather sharded momentum before the update (Liu et al., 2025a; Li et al., 2026a; Wang et al., 2026b). In such implementations the communicated payload is the optimizer state itself, and its precision sets the transfer volume. In our eight-GPU NVLink/NVSwitch setup with a 4.9B model, FP8 states reduce Muon state-communication time from 39.4 to 22.1 ms ($1.78\times$); when the same all-gather crosses a slow inter-node link, the reduction becomes a measurable share of step time (Section 3).

These memory and communication costs motivate our central comparison: should optimizer states be compressed structurally or quantized? Structural methods reduce the number or dimensionality

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

of retained statistics, whereas quantization reduces their precision while preserving the state structure (Dettmers et al., 2022; Xi et al., 2025; Gupta et al., 2025). Because the two approaches discard different information and require different temporary allocations, we compare them under matched compute precision and token budgets using measured peak memory.

We first screen fourteen optimizers on a 257M Llama-style model, then select nine spanning full-state baselines and structural compression methods for a more detailed 500M comparison. On FineWeb-Edu, we vary compute and optimizer-state precision at matched Chinchilla-scaled token budgets. We evaluate validation loss and downstream metrics alongside measured peak memory across per-device microbatch sizes and step time, relating changes in quality to realized resource savings. Longer runs, up to $8\times$ Chinchilla, and a 1B mixture-of-experts model test whether the quality comparison depends on training horizon or architecture.

Our main contributions are:

- We use peak-memory measurements across microbatch sizes and optimizer-state communication timings to identify two regimes in which state size matters: capacity-limited training and distributed updates that communicate optimizer states.
- We compare structural compression with state quantization under matched conditions across compute and state precisions and model architectures, finding that quantizing a strong optimizer gives better loss–memory and loss–time tradeoffs.
- We test whether this conclusion holds with longer training and at matched training time, and find that Muon with FP8 states remains the stronger practical choice among the methods evaluated.

2 Related Work

2.1 Structured Optimizer-State Compression

Gradient-projection methods form adaptive states in a reduced subspace. GaLore and Fira use SVD-derived projections (Zhao et al., 2024; Chen et al., 2025), while APOLLO and FRUGAL use random projections to avoid repeated decompositions (Zhu et al., 2025; Zmushko et al., 2025). COAP, COSMOS, and SUMO retain the projection template but modify subspace selection or updates in the complementary directions (Xiao et al., 2025; Liu et al., 2025b; Refael et al., 2025). GaLore 2 scales the SVD approach to a 7B model on 500B tokens with a 500-step subspace refresh and reports parity with 8-bit Adam (Su et al., 2025);

Other methods compress the statistics themselves. LDAdam transports low-dimensional moments between changing subspaces with error feedback (Robert et al., 2025), and LoRA-Pre factorizes momentum through a LoRA-inspired representation (Wang et al., 2026c). Adam-mini and SlimAdam retain dense first moments but share second-moment statistics (Zhang et al., 2024; Kalra et al., 2025). BAdam follows a separate block-coordinate strategy, retaining optimizer state for only the active parameter block (Luo et al., 2024). SCALE normalizes gradients by column and uses momentum only for the output layer (Glentis et al., 2026). Appendix C describes the evaluated methods.

2.2 Optimizer-State and Activation Quantization

Quantization lowers the precision of optimizer states while retaining their structure and update rule (Li et al., 2023). Blockwise 8-bit Adam established this approach for adaptive optimizers (Dettmers et al., 2022). COAT extends FP8 storage to AdamW states and saved activations (Xi et al., 2025), while FlashOptim uses companding for INT8 states and also compresses master weights (Ortiz et al., 2026). Muon momentum has likewise been quantized (Gupta et al., 2025).

Reduced-precision states are already used in production. DeepSeek-V3 stores AdamW moments in BF16 while retaining FP32 master weights and gradients, and uses FP8 for MoE dispatch activations (DeepSeek-AI, 2024). Lower precision is not always loss-neutral, however. Repeated rounding can leave moving averages unchanged (Topollai & Choromanska, 2026), and some 8-bit Adam runs exhibit loss spikes or require a lower learning rate (Wang et al., 2026b).

Quantization can also be applied to projected-gradient methods, as in 8-bit GaLore (Zhao et al., 2024) and quantized APOLLO (Wang, 2026). These combinations leave open whether structural compression improves the trade-off over quantizing a strong full-state optimizer. Even the largest projected-gradient study does not include a quantized matrix-based baseline (Su et al., 2025). We therefore compare full-state and structural methods under the same COAT-derived FP8 state format.

2.3 Optimizer Benchmarks and Evaluation Gaps

Existing benchmarks vary model scale and Chinchilla budget (Wen et al., 2025), standardize pre-training settings (Semenov et al., 2025), or test memory-efficient configurations with full-rank corrections (Glentis et al., 2025). They compare complete optimizers rather than isolate state representation.

OmniOpt is closest in scope: it evaluates 24 optimizers from 60M to 1B parameters and reports quality, stability, runtime, and optimizer-state memory (Li et al., 2026b). Because it compares complete configurations, the update rule, state structure, and state representation change together. Its memory metric excludes parameters, gradients, and activations, and it does not study FP8 model computation. We compare FP32 and FP8 states at matched token budgets, measure peak training memory, and test structural methods against quantized full-state baselines including Muon, SOAP, and AdEMAMix (Jordan et al., 2024; Vyas et al., 2025; Pagliardini et al., 2024).

2.4 Optimizer States in Distributed Training

Sharding and offloading reduce the device memory occupied by optimizer states (Rajbhandari et al., 2020; Ren et al., 2021). Low-precision states can be combined with these placements, although the placement of quantization scales and group boundaries affects whether sharding adds communication (Peng et al., 2023; Wang et al., 2026b). State transfers remain relevant when a distributed update needs the complete state.

Gather-based Muon implementations reconstruct sharded momentum before the matrix update (Liu et al., 2025a; Li et al., 2026a; Wang et al., 2026b). Reducing the frequency of these gathers is one response (Li et al., 2026a). Owner-based designs instead communicate gradients and updated parameters while keeping momentum local (Wang et al., 2026a; Chen et al., 2026). Our measurements examine the gather-based case, where state precision directly changes the transferred payload.

Dynamic MoE placement provides another example. FlexMoE transfers optimizer states when relocating expert replicas, and Piper includes these transfers in its migration model (Nie et al., 2023; Dash & Wang, 2026). SYMI avoids moving states during rebalancing by keeping them sharded in host memory (Skiadopoulos et al., 2026). Smaller states could reduce migration traffic in the former design or host-memory demand in the latter.

3 When Does Optimizer-State Compression Matter?

Optimizer-state compression serves two training-time systems goals. Under tight device-memory budgets, it can make small-microbatch training feasible. In distributed matrix-aware optimizers that exchange state shards, it can reduce the volume of data transferred between devices.

Capacity-limited training. Persistent optimizer-state storage scales with model size but not with the per-device microbatch. The same states are saved in resumable checkpoints, so storing them in compressed form also reduces the space needed to retain multiple checkpoints. Saved activations grow with the microbatch, while increasing the effective batch through gradient accumulation does not produce the same simultaneous activation footprint. State compression should therefore have its largest effect on device capacity when memory limits force small microbatches. Once activations dominate, even a large reduction in state bytes may change peak memory only slightly.

Figure 1 shows this transition directly. At a microbatch of one sequence per device, estimated AdamW state storage accounts for 34.3% of measured peak memory, and FP8 states reduce peak memory by 31.3%. At a microbatch of 32 sequences, the estimated state share falls to 5.8%, and the reduction from state quantization shrinks to 4.2%. Compute quantization follows the opposite trend. With FP32 states, moving from `W16A16G32` to `W8A8G8` (notation defined in Section 4.1)

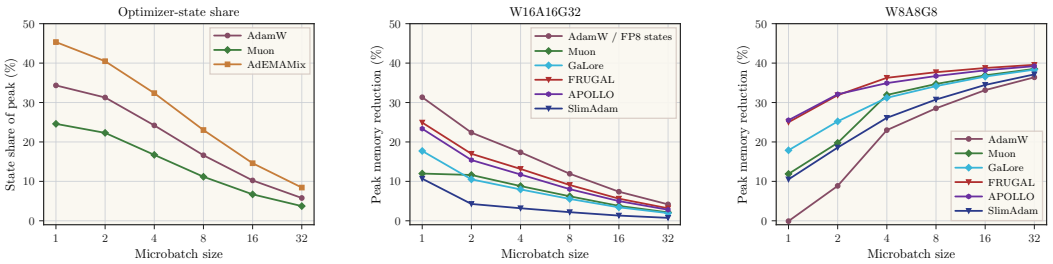

Figure 1: Memory versus per-device microbatch size in sequences for the 500M model at sequence length 1024. Left: estimated FP32 state bytes as a fraction of measured peak memory under `W16A16G32`. Middle and right: peak-memory savings under `W16A16G32` and `W8A8G8`, respectively. Both use AdamW under `W16A16G32` with FP32 states as the reference. Section 4.1 defines the `WxxAxxGxx` notation.

changes AdamW peak memory by less than 0.1% at a one-sequence microbatch but reduces it by 36.4% at a 32-sequence microbatch. In the practical regime where a moderate microbatch fits in device memory, compressing optimizer states therefore offers little additional capacity. Quantizing weights and activations saves substantially more memory in this regime, with only a small change in final validation loss for the 500M model (Table 14 and Figure 4).

State sharding further reduces the local capacity benefit. If optimizer states are balanced across D data-parallel ranks, an aggregate reduction of ΔS saves approximately $\Delta S/D$ bytes per rank (Rajbhandari et al., 2020). Sharding and compression remain compatible, but the incremental per-rank memory saving becomes smaller as other tensors dominate.

Communication-limited training. Some distributed implementations of matrix-aware optimizers reconstruct complete matrices before applying the update. In the Muon implementation we profile, each data-parallel rank holds a row shard of the momentum and participates in an all-gather before the matrix update (Jordan et al., 2024; Li et al., 2026a). Quantizing the momentum reduces both its local storage and the communicated payload. This benefit is implementation-specific. Methods that communicate gradients or parameters instead must compress those tensors to reduce communication (Chen et al., 2026).

Communication grows with model size (Figure 2). From 1B to 5B parameters, total Muon state communication increases from 5.2 to 33.4 ms with BF16 states and from 2.8 to 22.4 ms with FP8 states. In an eight-GPU node-local configuration, it reduces communication for the 5B model from 39.4 to 22.1 ms, a 1.78× reduction. Cross-node communication makes this reduction more consequential. In a cross-node configuration unoptimized for distributed training, communication decreases from 3.46 to 1.96 s, reducing complete step time from 20.81 to 19.45 s, or 6.6%. Appendix E reports the complete timing breakdowns.

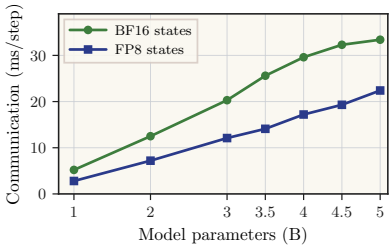

Figure 2: Muon optimizer-state communication versus model size on four GPUs (PP1/DP4). Total communication includes FP8 encoding and momentum all-gather.

These results identify two practical motivations for optimizer-state compression: increasing usable device capacity when microbatches are small and reducing state communication in distributed training.

4 How Should Optimizer States Be Compressed?

4.1 Notation and Experimental Setup

We write compute precision as `WxxAxxGxx`, where W and A denote the weight and activation operands of linear layers. `W16A16G32` denotes the BF16 reference with FP32 persistent parameter gradients. `W8A8G8` uses FP8 weight, activation, and backward activation-gradient operands,

while persistent parameter gradients still accumulate in FP32. The suffix `/ fp8 states` or `/ fp32 states` specifies the precision of persistent optimizer statistics. Appendix A.1 gives the implementation details.

We train 257M and 500M Llama-style models on FineWeb-Edu (Penedo et al., 2024), matching the model, data, and token budget within each comparison. Microbatch sizes count 1024-token sequences per device. Following the Chinchilla reference of about 20 tokens per parameter (Hoffmann et al., 2022), $k \times C$ denotes $20kN$ training tokens for a model with N parameters. A warmup-stable-decay (WSD) schedule extends a shared stable phase before a separate cooldown at each budget (Hu et al., 2024; Hägele et al., 2024). Appendix D gives model configurations and hyperparameters.

Repeating every configuration across seeds is costly. We test variability with three 257M $1 \times C$ runs per optimizer for each of two factors: PyTorch seed and data order. The sample standard deviation of final loss across PyTorch seeds was 0.0006 for AdamW and 0.001 for Muon. Varying data order gave 0.0016 and 0.00017, respectively (Tables 11 and 12).

Appendix C describes the evaluated optimizers and gives their pseudocode. We include FRUGAL MM to test whether FRUGAL's inner optimizer accounts for its quality gap. This variant uses Muon with momentum in the retained subspace and momentum-free Muon in its complement. The remaining steps follow FRUGAL (Zmushko et al., 2025; Jordan et al., 2024).

4.2 Selecting Optimizers for Detailed Comparison

Setup. We first compare tuned optimizers under BF16 computation and FP32 optimizer states to select methods for the detailed evaluations that follow. On the 257M and 500M models, we measure validation loss at matched token budgets (Section 4.1). Figure 3 shows the trajectories. Appendix B.2 reports final losses, and Appendix A.2 gives a broader 257M screening of memory-efficient optimizers (Figure 10).

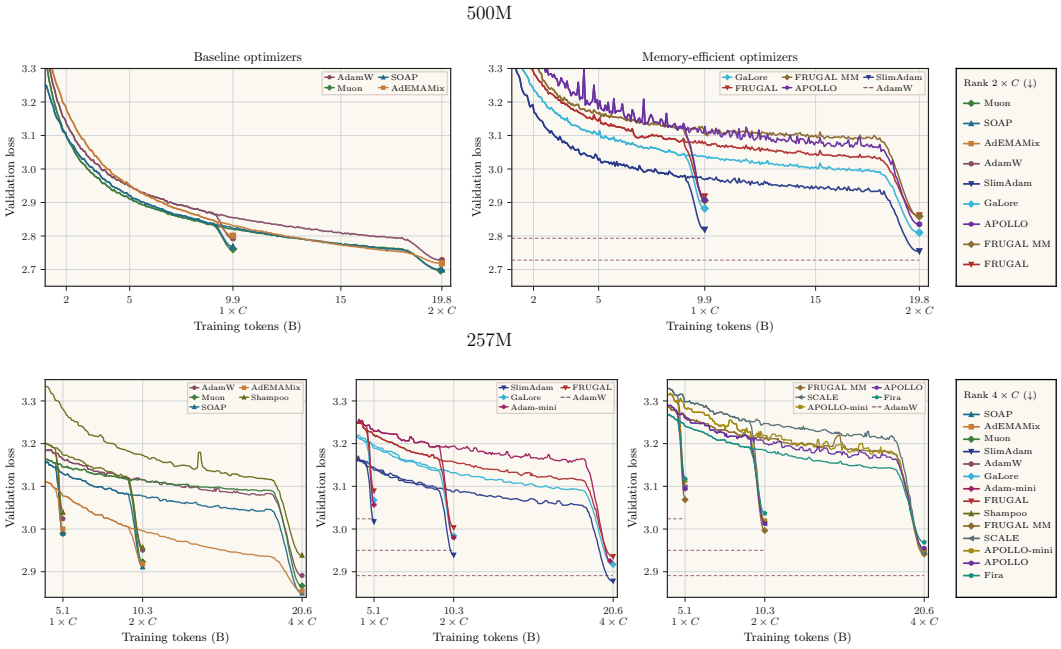

Figure 3: BF16 validation-loss trajectories with native optimizer states for the 500M (top) and 257M (bottom) models. The right-hand columns rank optimizers by final loss at $2 \times C$ and $4 \times C$, respectively. The 257M curves use a centered five-point rolling median. Dashed lines mark AdamW's final loss at the matched budget in the memory-efficient panels.

Discussion. On 257M, AdEMAMix stays below the other full-state baselines during WSD's stable phase, yet Muon and SOAP finish ahead at $1 \times C$. A similar stable-phase shape appears in the 500M

baseline runs, which use weight decay 10^{-4}. AdamW, Muon, SOAP, and AdEMAMix all use 0.1 on 257M, so different decay settings cannot explain AdEMAMix's separation within those runs. At $2 \times C$ and $4 \times C$, SOAP and AdEMAMix overtake Muon. Muon and SOAP remain close on 500M, while AdamW trails them at both model sizes. AdamW alone is therefore an incomplete quality reference for state compression. Shampoo's selected learning rate of 10^{-2} exceeds that of every other optimizer in Figure 3, and its raw trajectories contain loss spikes (Appendix A.3).

Overall, the memory-efficient methods perform markedly worse than the strongest full-state baselines. SlimAdam is the closest competitor and beats AdamW at every 257M budget, although this advantage does not carry over to 500M. Within the projected-gradient methods in Figure 3, GaLore has the lowest final loss for both models at every budget, despite being the earliest method in this group. To test whether FRUGAL's gap to Muon comes from its inner optimizer, we replace FRUGAL's inner updates with Muon variants. FRUGAL MM stays close to FRUGAL and behind standalone Muon at every budget. COSMOS also combines a projected SOAP variant with Muon in the residual subspace, yet trails the full-state baselines in our 257M $1 \times C$ screening (Appendix A.2 and C.4.8). Together, these results suggest that the quality gap may arise from the projection-based construction rather than only from the optimizer used within it.

We carry AdamW, Muon, SOAP, and AdEMAMix forward as full-state baselines, and GaLore, FRUGAL, FRUGAL MM, APOLLO, and SlimAdam as memory-efficient methods. The memory-efficient choices combine relatively strong BF16 results within this group with coverage of SVD-based projection, random projection, and shared second-moment statistics (Section 2.1).

4.3 Quantization versus Structural Compression

Setup. We compare the selected optimizers on the 500M model under `W16A16G32` and `W8A8G8`, with FP32 or FP8 optimizer states, measuring final validation loss and peak memory. For projected-gradient methods, we use the standard retained fraction $\rho = 0.25$ and include an ablation on the 257M model.

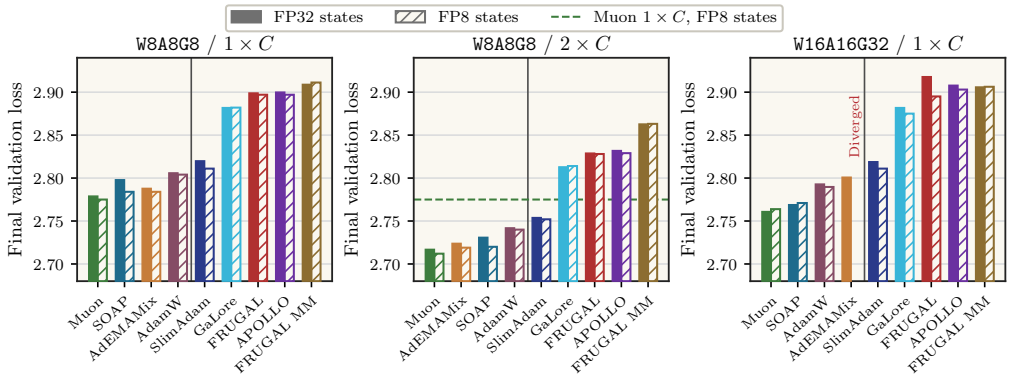

Figure 4: Final validation loss for the 500M model under `W8A8G8` (left and center) and `W16A16G32` (right). The vertical line in each panel separates the baselines from the memory-efficient optimizers. "Diverged" marks AdEMAMix with FP8 states under `W16A16G32`.

Discussion. At matched training budgets, FP8-state AdamW, Muon, and SOAP reach lower validation loss than the evaluated structured methods under both `W16A16G32` and `W8A8G8` (Figure 4). This holds whether the structured methods store their states in FP32 or FP8. Loss changes little with state precision for these baselines or the structured methods, leaving the quality gap intact. Appendix B.5 reports the exact endpoints.

Under `W8A8G8`, Muon with FP8 states at $1 \times C$ even outperforms four of the five structured methods trained to $2 \times C$. AdEMAMix limits the scope of the BF16 comparison: it finishes under `W8A8G8`, but all `W16A16G32 / fp8 states` repeats diverge. Its BF16 runs with FP32 states also require stronger gradient clipping. Appendix A.5 examines this instability.

	W16A16G32						W8A8G8					
Microbatch	1	2	4	8	16	32	1	2	4	8	16	32
AdamW	−1.85	+0.06	+0.42	+0.38	+0.35	+0.36	−2.58	−2.23	0.00	+0.39	+0.37	+0.38
Muon	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
SOAP	+9.09	+9.07	+8.66	+8.65	+8.58	+8.59	+9.07	+9.08	+9.05	+8.62	+8.60	+8.58
AdEMAMix	+0.93	+0.93	+0.98	+0.99	+0.98	+0.97	+0.93	+0.93	+0.91	+0.98	+0.98	+0.98
GaLore	−0.28	+1.56	+1.96	+1.90	+1.88	+1.88	−0.29	−0.29	+1.51	+1.90	+1.89	+1.90
FRUGAL	−1.11	+0.74	+1.11	+1.06	+1.03	+1.04	−1.11	−1.13	+0.68	+1.06	+1.05	+1.06
FRUGAL MM	−0.72	+0.36	+0.73	+0.69	+0.67	+0.68	−0.73	−0.74	+0.32	+0.69	+0.68	+0.69
APOLLO	−0.93	+0.94	+1.34	+1.31	+1.27	+1.29	−1.17	−1.16	+0.90	+1.29	+1.28	+1.29
SlimAdam	+0.53	+2.35	+2.74	+2.70	+2.69	+2.70	+0.56	+0.55	+2.34	+2.71	+2.71	+2.72
Muon / FP8 (GB)	9.76	9.76	13.09	20.58	35.45	65.21	9.75	9.75	9.74	13.77	22.62	40.29

Table 1: Extra peak memory (GB) relative to Muon with FP8 states for the 500M model at sequence length 1024. Blue indicates lower memory and red higher memory. AdamW, Muon, SOAP, and AdEMAMix use FP8 states, while structural methods use FP32 states. The last row gives absolute reference memory.

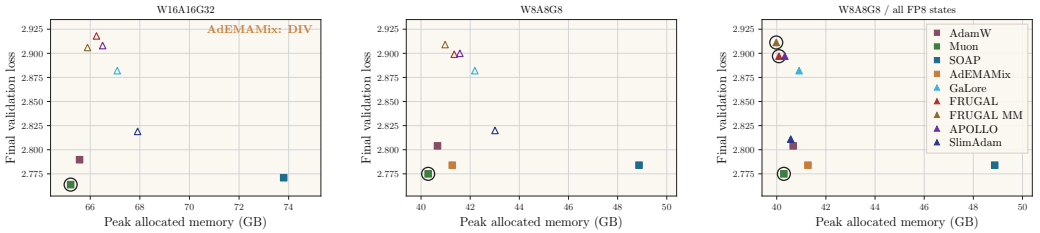

Figure 5: Final validation loss at $1 \times C$ versus peak allocated memory at a microbatch of 32 sequences for the 500M model. The first two panels compare FP8-state baselines with FP32-state structured methods. The third uses FP8 states for all methods. Squares denote baselines and triangles memory-efficient methods. Filled and open markers denote FP8 and FP32 states, respectively. Black rings mark Pareto-optimal configurations. AdEMAMix diverges under `W16A16G32 / fp8 states`.

For projected parameter matrices in GaLore, FRUGAL, and APOLLO, two FP32 Adam moments at $\rho = 0.25$ have the same nominal storage as two dense FP8 moments, at two bytes per parameter (Zhao et al., 2024; Zmushko et al., 2025; Zhu et al., 2025). Projection bases and temporary workspaces add to peak allocation, however, so equal moment storage need not yield equal measured memory. Muon with FP8 states uses less peak memory than every structured method at microbatch sizes of at least 2 sequences under `W16A16G32` and at least 4 sequences under `W8A8G8` (Table 1). At the smaller micro-batches where some structured methods use less memory than Muon, FP8-state AdamW uses less still and reaches lower loss. Appendix B.6 reports the absolute measurements.

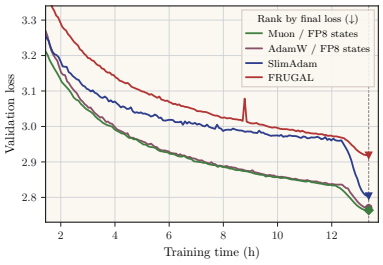

Figure 6: 500M BF16 validation loss versus training time. The dashed line marks the matched time budget.

The 32-sequence microbatch in the Pareto comparison represents a practical setting without severe device-memory constraints. At this size, FP8-state AdamW and Muon each have lower peak memory and lower loss than every memory-efficient optimizer with FP32 states under both compute precisions (Figure 5). When all methods use FP8 states, only FRUGAL and FRUGAL MM use less memory than Muon, and both reach substantially higher loss. In this regime, quantizing a strong full-state optimizer offers the better memory-quality tradeoff.

Time to quality. We compare validation loss at matched training time under BF16 computation on the 500M model. We extend the stable portion of the WSD schedule for FP8-state AdamW, SlimAdam, and FRUGAL so that each trains for as long as Muon with FP8 states takes to complete

$1 \times C$. At matched time, Muon with FP8 states reaches the lowest loss, followed closely by FP8-state AdamW (Figure 6). SlimAdam and FRUGAL take more steps in the same time but still finish at higher loss than either full-state baseline. Their shorter steps relative to Muon do not close the quality gap within this budget.

Retained-fraction ablation. A comparison at one retained fraction could understate the quality achievable with projected methods. In a 257M BF16 sweep, FRUGAL improves as more directions are retained, whereas GaLore changes little across most of the range (Figure 7). Neither method reaches the matched BF16 AdamW or Muon endpoints reported in Appendix B.2, even at its best retained fraction. GaLore shows a sharp loss increase at $\rho = 1$ in this run. Full rank does not make GaLore mathematically equivalent to AdamW because its coordinate-wise Adam update operates in the SVD basis before being mapped back to parameter space. The dashed FP8-state references use a different compute configuration and serve only as context.

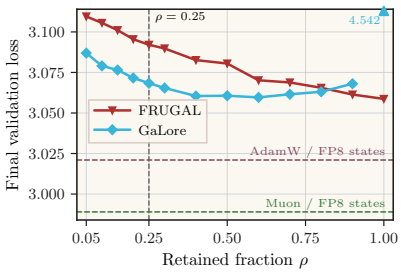

Figure 7: Final validation loss across retained fractions for the 257M model at $1 \times C$ in BF16.

Combining quantization and structural compression. FP8 state quantization is also compatible with the five memory-efficient methods. All complete 500M training under both `W16A16G32` and `W8A8G8`, with changes in final loss comparable to those of the full-state baselines (Figure 4). The additional memory benefit depends on the method and microbatch size. At a microbatch of one sequence, FRUGAL uses 5.1% to 5.3% less peak memory than Muon with FP8 states, whereas GaLore uses more even at this size, consistent with temporary allocations from its periodic SVD. FRUGAL's advantage falls below 1% at a microbatch of 32 sequences (Table 3 in Appendix A.6). Where combining the methods saves memory, it leaves the validation-loss gap to full-state baselines largely intact (Figure 4). Under a tight device-memory budget, optimizer-state sharding or offloading may therefore be preferable, subject to their runtime costs (Rajbhandari et al., 2020; Ren et al., 2021).

Takeaway. FP8-state AdamW and Muon retain nearly the same loss as with FP32 states and, at a microbatch of 32 sequences, outperform the evaluated memory-efficient methods with FP32 states in both loss and peak memory (Figure 5). SOAP remains competitive in loss but uses substantially more peak memory (Table 1). We therefore recommend FP8-state AdamW or Muon as the starting point for state compression in the regimes studied here.

4.4 Scaling with Training Data

Longer training could change the ranking of structural methods and full-state baselines. APOLLO reports improving relative performance with more tokens (Zhu et al., 2025, §5.4, A4), and GaLore 2 reports a similar trend (Su et al., 2025, §6.1). Moonlight reports smaller Muon gains over AdamW when scaling both model size and training tokens (Liu et al., 2025a, §2.2). To test these trends at fixed model size and schedule, we compare final validation loss from $1 \times C$ to $8 \times C$ on 257M and at $1 \times C$ and $2 \times C$ on 500M under a common WSD protocol. Tables 13 and 14 give the BF16 endpoints.

Discussion. The BF16 loss trajectories (Figure 3) show that on 257M, GaLore, FRUGAL, APOLLO-mini, and Fira move closer to AdamW by $4 \times C$ but remain behind Muon and SOAP. APOLLO and FRUGAL MM show little catch-up. SOAP and AdEMAMix improve relative to AdamW, while Muon's lead narrows. On 500M, the projected methods start farther behind AdamW, and their gap narrows little through $2 \times C$.

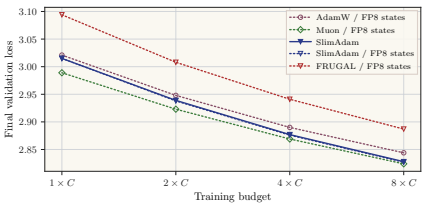

Figure 8: Final validation loss on the 257M model with BF16 computation from $1 \times C$ to $8 \times C$.

As 257M training extends to $8 \times C$, the loss gap between FP8-state SlimAdam and FP8-state Muon narrows, and SlimAdam finishes ahead of FP8-state AdamW (Figure 8, Table 15). SlimAdam's FP32-state and FP8-state trajectories stay close, so the narrowing is not

specific to state quantization. The quality gain has limited memory payoff. In the 500M memory measurements, FP8-state SlimAdam uses less peak memory than FP8-state Muon only at microbatches of one or two sequences under BF16 computation and up to four under FP8 computation (Table 3). SlimAdam keeps the full first moment and compresses only the second, leaving it among the less memory-efficient structural methods. Its wider loss gap to Muon on 500M also leaves open whether the 257M trend persists at larger model size.

4.5 Mixture-of-Experts Models

Setup. We train a MoE model with 1B total and 280M active parameters on FineWeb-Edu for 7.34B tokens, with the same recipe for every optimizer. Appendix D.3 gives the model configuration and training hyperparameters. Appendix A.8 reports the exact losses and peak-memory comparisons.

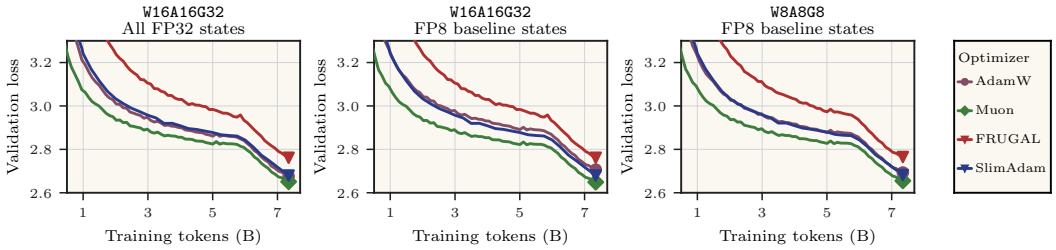

Figure 9: MoE validation loss over 7.34B training tokens. Left: `W16A16G32` with FP32 states for all methods. Center: `W16A16G32` with FP8 states for AdamW and Muon and FP32 states for SlimAdam and FRUGAL. Right: the same state assignment under `W8A8G8`. All panels use the same axes. Markers show the fitted final validation losses from Table 8.

Discussion. Muon reaches the lowest validation loss in all three MoE comparisons (Figure 9). SlimAdam remains competitive with AdamW: it narrowly trails when both use FP32 states, but surpasses FP8-state AdamW under both `W16A16G32` and `W8A8G8`. In those two settings, Muon with FP8 states also finishes ahead of the structurally compressed FRUGAL and SlimAdam while using less peak memory at every measured microbatch (Appendix A.8). The MoE results therefore support the conclusion from the dense models: quantizing Muon's states gives a better measured balance of loss and memory than structural state compression.

5 Conclusion

Optimizer-state compression matters when states limit device capacity at small microbatches or must be communicated between devices. As activation memory grows, reducing state bytes brings much less relief in measured peak memory. Our controlled comparisons show that quantizing AdamW, Muon, and SOAP states largely preserves quality, while structural methods retain a loss gap that their memory savings rarely offset. In the main 500M experiment, FP8-state Muon gives the best memory versus loss result at a 32-sequence microbatch and reaches the lowest loss at matched training time, despite slower individual steps than some structural methods. Longer 257M training narrows several gaps, but none of the tested structural methods surpasses FP8-state Muon within the measured budgets. The 1B MoE experiment supports the same choice: FP8-state Muon reaches lower loss and peak memory than FP32-state FRUGAL and SlimAdam under both compute precisions. We therefore recommend quantizing a strong optimizer before replacing its update with a structurally compressed one.

Limitations. Our dense-model quality comparison stops at 500M parameters, a modest scale for current language models. Most optimizer hyperparameters were selected on 257M rather than systematically retuned for 500M (Appendix D).

USE OF GENERATIVE AI

We used generative AI tools to identify potentially relevant related work and to polish the manuscript's wording and structure. We did not use them to generate research ideas. We reviewed the AI-assisted material and take responsibility for the paper's final text, citations, and claims.

REPRODUCIBILITY STATEMENT

Appendix D reports model configurations, training schedules, and optimizer hyperparameters. Appendices A.1 and C describe the precision formats, implementation, and optimizer algorithms. The anonymous code repository provides training and evaluation scripts, with instructions for downloading the public FineWeb-Edu data and recreating the data preparation and training setup.

REFERENCES

Jeremy Bernstein and Laker Newhouse. Old optimizer, new norm: An anthology. *arXiv preprint arXiv:2409.20325*, 2024.

Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signSGD: Compressed optimisation for non-convex problems. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 560–569. PMLR, 2018. URL https://proceedings.mlr.press/v80/bernstein18a.html.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Vincent Chen, Starrick Liu, Regis Cheng, Dance Yang, Shalfun Li, Ryan Yu, Lucy Liang, Hang Su, Roy Gan, Hao Wang, and Qian Wang. Dmuon: Efficient distributed muon training with near-adam overhead, 2026. URL https://arxiv.org/abs/2606.27153.

Xi Chen, Kaituo Feng, Changsheng Li, Xunhao Lai, Xiangyu Yue, Ye Yuan, and Guoren Wang. Fira: Can we achieve full-rank training of llms under low-rank constraint? *arXiv preprint arXiv:2410.01623*, 2025. URL https://arxiv.org/abs/2410.01623.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? Try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

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

Sajal Dash and Feiyi Wang. Piper: Efficient large-scale moe training via resource modeling and pipelined hybrid parallelism, 2026. URL https://arxiv.org/abs/2605.05049.

DeepSeek-AI. DeepSeek-V3 Technical Report. *arXiv preprint arXiv:2412.19437*, 2024. URL https://arxiv.org/abs/2412.19437.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization, 2022. URL https://arxiv.org/abs/2110.02861.

Athanasios Glentis, Jiaxiang Li, Qiulin Shang, Andi Han, Ioannis Tsaknakis, Quan Wei, and Mingyi Hong. Scalable parameter and memory efficient pretraining for llm: Recent algorithmic advances and benchmarking. *arXiv preprint arXiv:2505.22922*, 2025. URL https://arxiv.org/abs/2505.22922.

Athanasios Glentis, Jiaxiang Li, Andi Han, and Mingyi Hong. Memory-efficient llm pretraining via minimalist optimizer design, 2026. URL https://arxiv.org/abs/2506.16659.

Aman Gupta, Rafael Celente, Abhishek Shivanna, DT Braithwaite, Gregory Dexter, Shao Tang, Hiroto Udagawa, Daniel Silva, Rohan Ramanath, and S Sathiya Keerthi. Effective quantization of muon optimizer states. *arXiv preprint arXiv:2509.23106*, 2025.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1842–1850. PMLR, 2018. URL https://proceedings.mlr.press/v80/gupta18a.html.

Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro Von Werra, and Martin Jaggi. Scaling laws and compute-optimal training beyond fixed training durations. In *Advances in Neural Information Processing Systems*, 2024.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

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

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

Dayal Singh Kalra, John Kirchenbauer, Maissam Barkeshli, and Tom Goldstein. When can you get away with low memory adam? *arXiv preprint arXiv:2503.01843*, 2025.

Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian U. Stich, and Martin Jaggi. Error feedback fixes SignSGD and other gradient compression schemes. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. URL https://arxiv.org/abs/1412.6980.

Bingrui Li, Jianfei Chen, and Jun Zhu. Memory efficient optimizers with 4-bit states. In *Advances in Neural Information Processing Systems*, volume 36, pp. 15136–15171. Curran Associates, Inc., 2023. doi: 10.52202/075280-0666. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/3122aaa22b2fe83f9cead1a696f65ceb-Paper-Conference.pdf.

Chenghao Li, Xiao Han, Xinxin Huang, Wei Liu, Boyang Li, Bing Xiao, Heran Zhang, Juanma Perez Rua, Ke Xu, Kangning Liu, Linjun Kuang, Na Li, Tan Wang, Tian Xie, Wei Peng, Yang Pei, Yifan Xu, Yuanhao Zhai, Yuwei Lin, Zhe Wang, Zihao He, Daniel Li, Junbiao Tang, Ziyang Jiang, and Dake Chen. Scaling muon for diffusion transformers, 2026a. URL https://arxiv.org/abs/2608.20818.

Siyuan Li, Jiabao Pan, Yumou Liu, Zhuoli Ouyang, Xin Jin, Xinglong Xu, Jingxuan Wei, Shengye Pang, Jintao Che, Xuanhe Zhou, Conghui He, and Cheng Tan. Omniopt: Taxonomy, geometry, and benchmarking of modern optimizers, 2026b. URL https://arxiv.org/abs/2607.04033.

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, 2025a. URL https://arxiv.org/abs/2502.16982.

Liming Liu, Zhenghao Xu, Zixuan Zhang, Hao Kang, Zichong Li, Chen Liang, Weizhu Chen, and Tuo Zhao. Cosmos: A hybrid adaptive optimizer for memory-efficient training of llms. *arXiv preprint arXiv:2502.17410*, 2025b. URL https://arxiv.org/abs/2502.17410.

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

Qijun Luo, Hengxu Yu, and Xiao Li. Badam: A memory efficient full parameter optimization method for large language models. *Advances in Neural Information Processing Systems*, 37:24926–24958, 2024.

Chao Ma, Wenbo Gong, Meyer Scetbon, and Edward Meeds. Swan: Sgd with normalization and whitening enables stateless llm training. *arXiv preprint arXiv:2412.13148*, 2025. URL https://arxiv.org/abs/2412.13148.

James Martens and Roger B. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. *CoRR*, abs/1503.05671, 2015. URL http://arxiv.org/abs/1503.05671.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. Fp8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 1–15, 2021.

Yurii Nesterov. A method for solving the convex programming problem with convergence rate $O(1/k^2)$. *Proceedings of the USSR Academy of Sciences*, 269:543–547, 1983.

Xiaonan Nie, Xupeng Miao, Zilong Wang, Zichao Yang, Jilong Xue, Lingxiao Ma, Gang Cao, and Bin Cui. Flexmoe: Scaling large-scale sparse pre-trained model training via dynamic device placement. *Proceedings of the ACM on Management of Data*, 1(1):1–19, May 2023. ISSN 2836-6573. doi: 10.1145/3588964. URL http://dx.doi.org/10.1145/3588964.

Jose Javier Gonzalez Ortiz, Abhay Gupta, Christopher Rinard, and Davis Blalock. Flashoptim: Optimizers for memory-efficient training, 2026. URL https://arxiv.org/abs/2602.23349.

Matteo Pagliardini, Pierre Ablin, and David Grangier. The ademamix optimizer: Better, faster, older. *arXiv preprint arXiv:2409.03137*, 2024. URL https://arxiv.org/abs/2409.03137.

Andrei Panferov, Erik Schultheis, Soroush Tabesh, and Dan Alistarh. Quartet ii: Accurate llm pre-training in nvfp4 by improved unbiased gradient estimation. *arXiv preprint arXiv:2601.22813*, 2026.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

Houwen Peng, Kan Wu, Yixuan Wei, Guoshuai Zhao, Yuxiang Yang, Ze Liu, Yifan Xiong, Ziyue Yang, Bolin Ni, Jingcheng Hu, et al. Fp8-lm: Training fp8 large language models. *arXiv preprint arXiv:2310.18313*, 2023.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: international conference for high performance computing, networking, storage and analysis*, pp. 1–16. IEEE, 2020.

Yehonathan Refael, Guy Smorodinsky, Tom Tirer, and Ofir Lindenbaum. Sumo: Subspace-aware moment-orthogonalization for accelerating memory-efficient llm training. *arXiv preprint arXiv:2505.24749*, 2025. URL https://arxiv.org/abs/2505.24749.

Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. ZeRO-Offload: Democratizing Billion-Scale model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 551–564. USENIX Association, July 2021. ISBN 978-1-939133-23-6. URL https://www.usenix.org/conference/atc21/presentation/ren-jie.

Thomas Robert, Mher Safaryan, Ionut-Vlad Modoranu, and Dan Alistarh. Ldadam: Adaptive optimization from low-dimensional gradient statistics. *arXiv preprint arXiv:2410.16103*, 2025. URL https://arxiv.org/abs/2410.16103.

Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs. In *Interspeech*, 2014.

Andrei Semenov, Matteo Pagliardini, and Martin Jaggi. Benchmarking optimizers for large language model pretraining. *arXiv preprint arXiv:2509.01440*, 2025.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. *arXiv preprint arXiv:1804.04235*, 2018. URL https://arxiv.org/abs/1804.04235.

Athinagoras Skiadopoulos, Mark Zhao, Swapnil Gandhi, Thomas Norrie, Shrijeet Mukherjee, and Christos Kozyrakis. SYMI: Efficient Mixture-of-Experts training via model and optimizer state decoupling. In *23rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 26)*, pp. 75–92, Renton, WA, May 2026. USENIX Association. ISBN 978-1-939133-54-0. URL https://www.usenix.org/conference/nsdi26/presentation/skiadopoulos.

DiJia Su, Andrew Gu, Jane Xu, Yuandong Tian, and Jiawei Zhao. Galore 2: Large-scale llm pre-training by gradient low-rank projection, 2025. URL https://arxiv.org/abs/2504.20437.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

Changxin Tian, Kunlong Chen, Jia Liu, Ziqi Liu, Zhiqiang Zhang, and Jun Zhou. Towards greater leverage: Scaling laws for efficient mixture-of-experts language models. *arXiv preprint arXiv:2507.17702*, 2025.

Kristi Topollai and Anna Choromanska. Understanding quantization of optimizer states in llm pre-training: Dynamics of state staleness and effectiveness of state resets, 2026. URL https://arxiv.org/abs/2603.16731.

Nikhil Vyas, Depen Morwani, Rosie Zhao, Mujin Kwun, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. Soap: Improving and stabilizing shampoo using adam. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2409.11321. ICLR 2025.

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.

Yan Wang. Beyond dense adam states: Adaptive log-space quantization for memory-efficient optimizers, 2026. URL https://arxiv.org/abs/2608.22322.

Zezhou Wang, Youjie Li, Zhiqi Lin, Jiacheng Yang, Cong Xie, Guanyu Feng, Zheng Zhong, Ziyue Huang, Hongyu Zhu, Zhi Zhang, Yanghua Peng, and Xin Liu. vescale-fsdp: Flexible and high-performance fsdp at scale, 2026b. URL https://arxiv.org/abs/2602.22437.

Zhengbo Wang, Jian Liang, Ran He, Zilei Wang, and Tieniu Tan. Taming momentum: Rethinking optimizer states through low-rank approximation. In *The Fourteenth International Conference on Learning Representations*, 2026c.

Kaiyue Wen, David Hall, Tengyu Ma, and Percy Liang. Fantastic pretraining optimizers and where to find them. *arXiv preprint arXiv:2509.02046*, 2025.

Haocheng Xi, Han Cai, Ligeng Zhu, Yao Lu, Kurt Keutzer, Jianfei Chen, and Song Han. COAT: compressing optimizer states and activations for memory-efficient FP8 training. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=XfKSDgqIRj.

Jinqi Xiao, Shen Sang, Tiancheng Zhi, Jing Liu, Qing Yan, Yuqian Zhang, Linjie Luo, and Bo Yuan. Coap: Memory-efficient training with correlation-aware gradient projection. *arXiv preprint arXiv:2412.00071*, 2025. URL https://arxiv.org/abs/2412.00071.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019.

Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Diederik P Kingma, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. Adam-mini: Use fewer learning rates to gain more. *arXiv preprint arXiv:2406.16793*, 2024.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*, 2024.

Hanqing Zhu, Zhenyu Zhang, Wenyan Cong, Xi Liu, Sem Park, Vikas Chandra, Bo Long, David Z. Pan, Zhangyang Wang, and Jinwon Lee. Apollo: Sgd-like memory, adamw-level performance. *arXiv preprint arXiv:2412.05270*, 2025. URL https://arxiv.org/abs/2412.05270.

Philip Zmushko, Aleksandr Beznosikov, Martin Takáč, and Samuel Horváth. Frugal: Memory-efficient optimization by reducing state overhead for scalable training. *arXiv preprint arXiv:2411.07837*, 2025. URL https://arxiv.org/abs/2411.07837.

A Experimental Details and Additional Results

A.1 Training Implementation and Precision

In `W8A8G8`, weights and forward activations use E4M3, and activation gradients are quantized to E5M2 before backward matrix multiplications (Micikevicius et al., 2022; Xi et al., 2025). The G8 label refers to these backward operands, not to the accumulator: FP8 products accumulate in FP32, immediate matrix-multiplication outputs are BF16, and persistent parameter gradients accumulate in FP32 across microbatches. Both compute paths retain FP32 master parameters. The FP8 path derives its compute weights from them. FP8 optimizer states use COAT's dynamic range expansion and E4M3 storage in groups of 128 values (Xi et al., 2025). Each optimizer update reconstructs these states in FP32 and quantizes them again afterward. Validation uses the non-FP8 forward path.

The dense experiments use Quartet-II-based training code (Panferov et al., 2026) with COAT FP8 integration, including quantized Llama operators and optimizer-state kernels (Xi et al., 2025). We implement optimizers from released code when available and otherwise follow the algorithms in their papers. Distributed runtime measurements use Megatron-LM/Core (Narayanan et al., 2021).[1] Our WSD runs continue the stable phase before applying a separate cooldown at each budget, a protocol also used in a broader optimizer benchmark (Wen et al., 2025).

A.2 BF16 Optimizer Screening

We screened eleven memory-efficient optimizers in BF16 on the 257M model at $1 \times C$ (Figure 10). GaLore, SlimAdam, FRUGAL, and COSMOS finish within 0.023 validation loss of one another, whereas COAP, SUMO, and BAdam trail this group substantially.

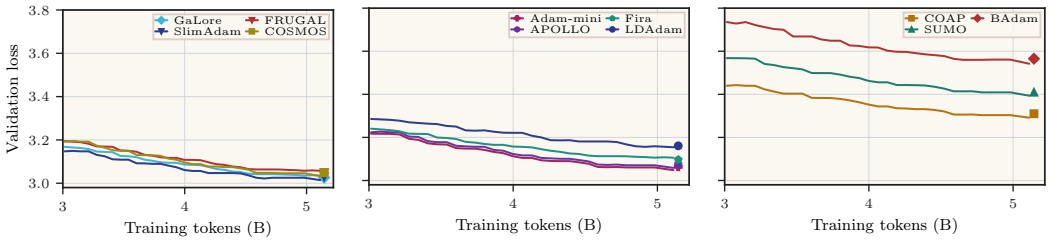

Figure 10: BF16 validation loss for eleven memory-efficient optimizers on the 257M model at $1 \times C$. Left to right, panels show ranks 1 to 4, 5 to 8, and 9 to 11 by final loss. Curves use a centered five-point rolling median. Markers show the unsmoothed final evaluation. All panels share the same vertical scale.

The endpoint tables show that SlimAdam beats AdamW at every 257M budget but not on 500M. Although GaLore is the earliest projected-gradient method we evaluate, it has the lowest loss in this family at every reported BF16 budget. Its periodic SVD incurs a substantial runtime cost: at 3B parameters and an effective batch of one sequence, GaLore takes 1.61 s per step versus 0.18 s for FRUGAL (Table 6). This makes GaLore impractical in the larger-model configurations we profile. FRUGAL MM remains close to FRUGAL despite its Muon-based inner updates.

One method was excluded before the screening figure. SWAN (Ma et al., 2025), a stateless optimizer that replaces Adam's moments with gradient normalization and whitening, did not train to a competitive loss in our implementation: across 27 learning-rate and weight-decay configurations on the 257M model at $1 \times C$ (learning rates 10^{-5} to 2×10^{-3}, weight decay 0.01 to 0.2), the best final validation loss was 5.43 against 2.99 for AdamW, and most runs plateaued between 5.6 and 6.0 or diverged. We draw no conclusion about the method from this, since the gap is far larger than any tuning effect we observed for other optimizers and most likely reflects our implementation; we report it so that the screening set is complete.

A full sweep across model sizes, token budgets, compute and state precisions, and microbatch sizes is too costly for every screened method. We therefore carry AdamW, Muon, SOAP, and AdEMAMix

[1] Upstream codebases: `https://github.com/IST-DASLab/Quartet-II`, `https://github.com/NVlabs/COAT`, and `https://github.com/NVIDIA/Megatron-LM`.

forward as full-state baselines, and GaLore, FRUGAL, FRUGAL MM, APOLLO, and SlimAdam as memory-efficient methods. GaLore provides a quality reference for SVD-based projection, while FRUGAL covers split updates, APOLLO random-projection scaling, and SlimAdam shared second-moment statistics. FRUGAL MM tests whether changing the inner optimizers narrows FRUGAL's loss gap.

A.3 Shampoo Training Dynamics

Figure 3 applies a centered five-point rolling median to every trajectory. Figure 11 reports the raw Shampoo trajectories and the hyperparameter selection runs. Among the three tested configurations, learning rate 10^{-2} and weight decay 10^{-2} give the lowest final validation loss. We retain this pair for the 257M and 500M comparisons. Longer token budgets improve the final loss, although the trajectories contain isolated spikes. The 500M $1 \times C$ and $2 \times C$ curves share the same training prefix and separate when the shorter run enters learning-rate decay.

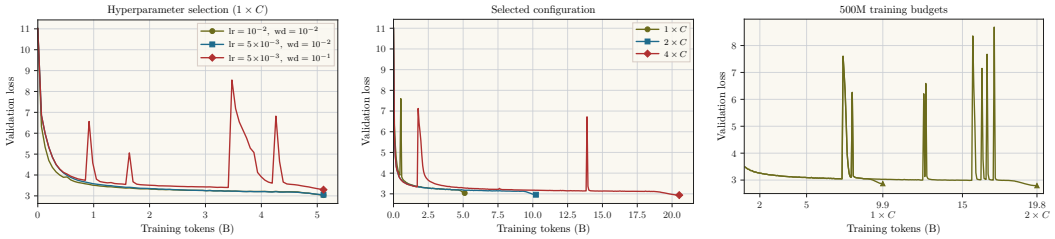

Figure 11: Shampoo training dynamics in BF16. Left: 257M hyperparameter selection at $1 \times C$. Center: the selected 257M configuration at $1 \times C$, $2 \times C$, and $4 \times C$. Right: 500M training at $1 \times C$ and $2 \times C$. Curves show individual training runs.

A.4 Downstream Evaluation

We also evaluate checkpoints beyond FineWeb-Edu validation loss. PIQA tests physical commonsense through two-choice questions and is scored by length-normalized accuracy (Bisk et al., 2020). HellaSwag tests commonsense event completion with the same metric (Zellers et al., 2019). ARC-Easy measures accuracy on multiple-choice grade-school science questions (Clark et al., 2018).

For held-out language modeling, we report validation loss on WikiText-103 (Merity et al., 2016). On GSM8K, we report five-shot bits per byte for the reference chain-of-thought solutions (Cobbe et al., 2021). This measures how well the model assigns probability to the supplied solutions, not the accuracy of answers it generates. Higher values are better for the three accuracy metrics, whereas lower values are better for WikiText-103 loss and GSM8K bits per byte.

Figures 12–16 show the downstream evaluations of the BF16 runs in Figure 3. Lines connect evaluations at the reported token budgets rather than a continuously measured trajectory. The 500M FRUGAL MM run has no downstream evaluation and is omitted from these figures.

The downstream metrics largely restate the validation-loss ranking. Across the 55 BF16 endpoints, the Spearman correlation between validation loss and HellaSwag accuracy is -0.99 on both model sizes, with WikiText-103 loss $+0.99$, with GSM8K bits per byte $+0.96$ to $+0.97$, and with PIQA -0.93 to -0.94. ARC-Easy is less stable, with weaker correlations of -0.65 on 257M, -0.84 on 500M, and -0.52 within a single budget.

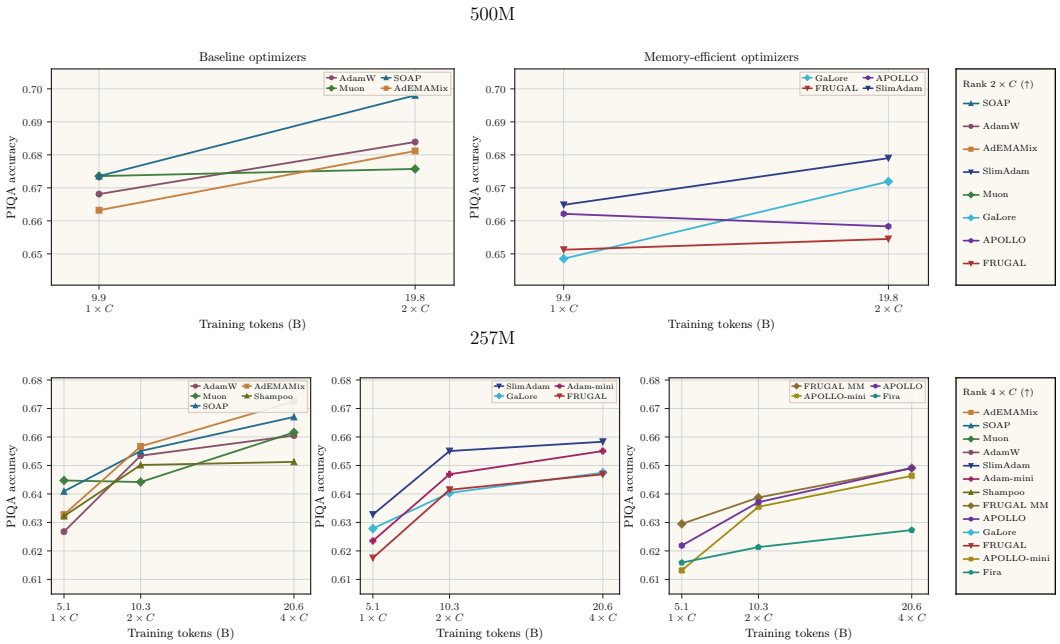

Figure 12: PIQA length-normalized accuracy for the 500M (top) and 257M (bottom) BF16 runs. Lines connect budget endpoints. The right columns rank optimizers at $2 \times C$ and $4 \times C$, respectively. Higher is better.

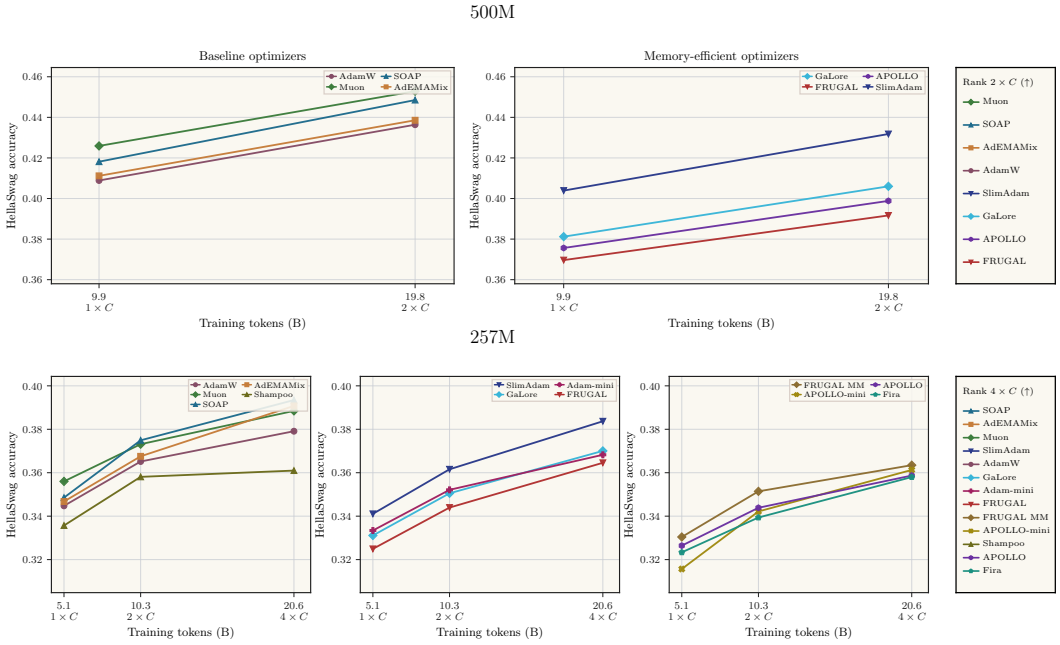

Figure 13: HellaSwag length-normalized accuracy for the 500M (top) and 257M (bottom) BF16 runs. Lines connect budget endpoints. The right columns rank optimizers at $2 \times C$ and $4 \times C$, respectively. Higher is better.

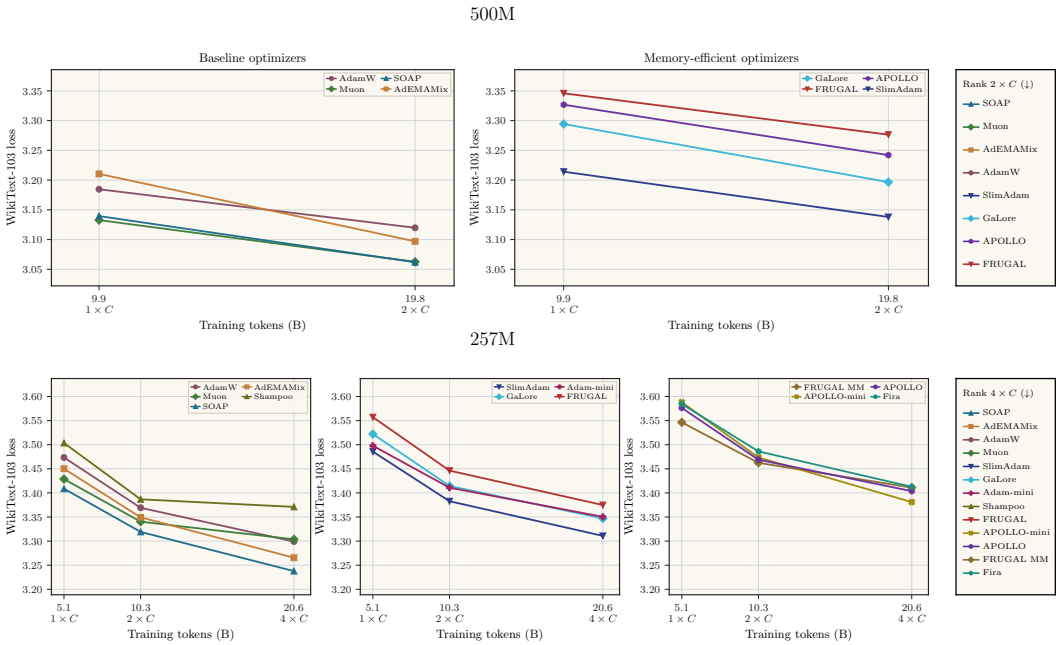

Figure 14: WikiText-103 validation loss for the 500M (top) and 257M (bottom) BF16 runs. Lines connect budget endpoints. The right columns rank optimizers at $2 \times C$ and $4 \times C$, respectively. Lower is better.

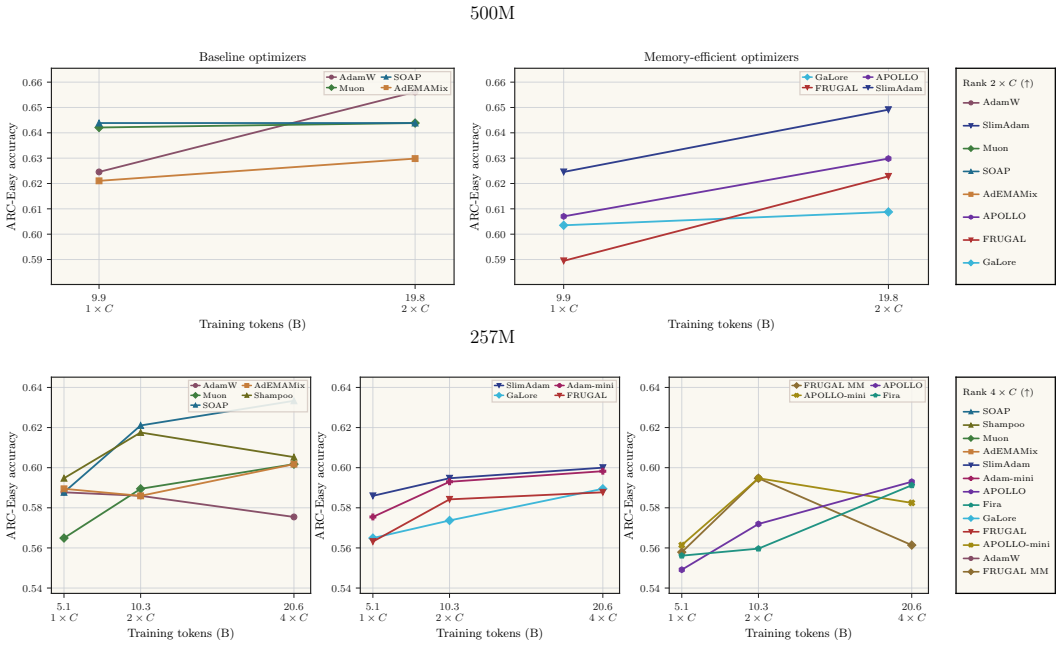

Figure 15: ARC-Easy accuracy for the 500M (top) and 257M (bottom) BF16 runs. Lines connect budget endpoints. The right columns rank optimizers at $2 \times C$ and $4 \times C$, respectively. Higher is better.

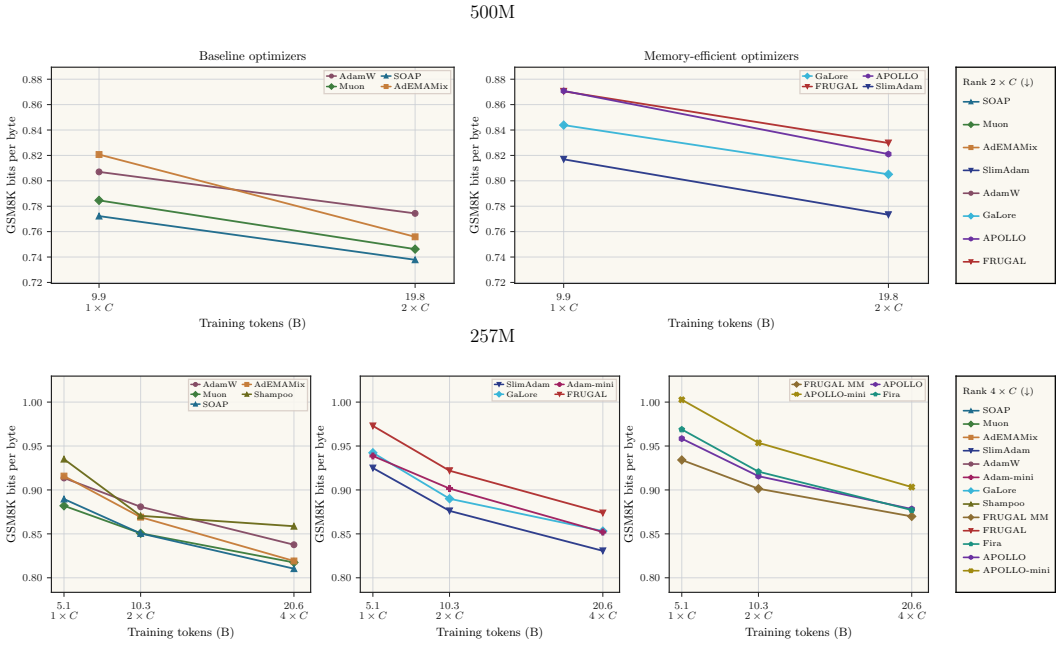

Figure 16: Five-shot bits per byte for GSM8K reference solutions on the 500M (top) and 257M (bottom) BF16 runs. Lines connect budget endpoints. The right columns rank optimizers at $2 \times C$ and $4 \times C$, respectively. Lower is better.

A.5 AdEMAMix under FP8 State Quantization

AdEMAMix is competitive when its states remain in FP32, but its behavior changes when all three persistent statistics are quantized. In the `W16A16G32 / fp8 states` setting, each of the three 500M repeats develops abrupt validation-loss spikes and terminates between steps 37,800 and 43,000. The same failure occurs on the 257M model. The instability is therefore reproducible across runs and is not confined to one model size. By contrast, quantizing weights and activations while retaining FP32 states does not produce this failure.

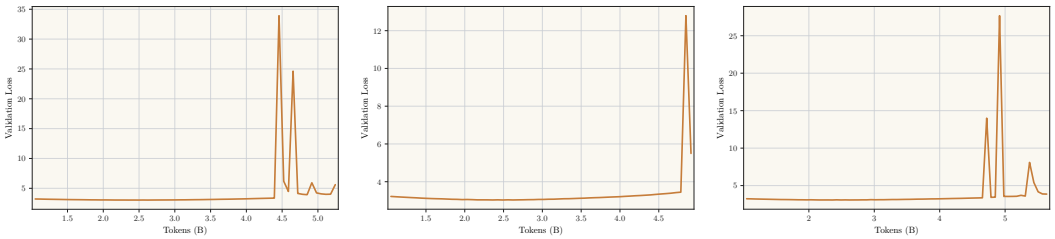

Figure 17: Validation loss in three AdEMAMix runs with `W16A16G32 / fp8 states` on the 500M model. All three runs become unstable before completing the $1 \times C$ budget.

The delayed failure is consistent with error accumulating in the slow first-moment state. AdEMAMix maintains this state with β_3 approaching 0.9999 and increases its contribution to the update through a mixing coefficient that approaches 8. Repeated low-precision storage can perturb this long-lived statistic, after which the growing mixing coefficient can amplify the error. Quantization of small second-moment entries may also reduce the adaptive denominator. These mechanisms are plausible explanations for the observed drift, rather than an identified causal decomposition.

We tested several modifications from a configuration using E4M3 moments, a per-group power-law transform, and FP32 quantization metadata. Table 2 summarizes the results. Update clipping limits individual spikes but does not remove the preceding drift. The other interventions likewise fail to recover the FP32-state trajectory at the same state-storage budget.

Table 2: Stabilization attempts for AdEMAMix with FP8 states. Each change is evaluated separately from the same reference configuration.

Modification	Intended effect	Observed outcome
Variance floor (10^{-12}, 10^{-10})	protect the adaptive denominator	loss drift persists
RMS update clipping ($\tau = 1$)	limit large updates	spikes are capped, drift persists
Stochastic rounding of slow momentum	reduce rounding bias	loss drift persists
Reduced mixing coefficient ($8 \to 4$)	reduce slow-momentum amplification	loss drift persists
Shorter second-moment decay ($0.999 \to 0.95$)	reduce stale statistics	loss drift persists

These results qualify the otherwise favorable behavior of state quantization. FP8 storage is stable for the other optimizers in the main comparison, but it cannot be treated as an optimizer-independent transformation. AdEMAMix requires a different quantization recipe or a larger state budget.

Two features of the failure constrain its explanation. It appears only when all three states are quantized under BF16 computation and not under `W8A8G8` with the same FP8 states, so it is not a property of the state codec alone; the FP8 compute path changes the gradient statistics that feed the slow moment, and this is enough to keep the run inside the stable region. And it is delayed by roughly 38,000 to 43,000 steps in every repeat, a few multiples of the 10,000-step time constant of the slow moment ($\beta_3 = 0.9999$), rather than the time scale of any single rounding error. The behavior is consistent with the staleness mechanism described for quantized moving averages (Topollai & Choromanska, 2026) and with the loss spikes reported for 8-bit Adam at scale (Wang et al., 2026b).

A.6 Combining State Quantization and Structural Compression

Table 3 compares peak memory with FP8-state Muon. Four methods save memory at a one-sequence microbatch, but at 32 sequences only FRUGAL and FRUGAL MM do, each by less than 1%. GaLore uses more memory at every measured microbatch under both compute precisions. Its projected FP8 states therefore provide no peak-memory advantage over quantized Muon in this setup.

	W16A16G32 / fp8 states					W8A8G8 / fp8 states				
Microbatch	GaLore	FRUGAL	FRUGAL MM	APOLLO	SlimAdam	GaLore	FRUGAL	FRUGAL MM	APOLLO	SlimAdam
1	−3.18	+5.33	+3.89	+2.66	+2.97	−3.18	+5.13	+3.79	+2.77	+2.97
2	−3.28	+5.33	+4.00	+2.66	+0.82	−3.28	+5.13	+3.90	+2.87	+2.97
4	−5.04	+1.60	+2.44	−0.46	−2.14	−3.29	+5.13	+3.90	+2.67	+1.13
8	−2.77	+1.26	+1.75	−0.05	−1.46	−4.50	+1.45	+2.25	−0.44	−1.96
16	−1.69	+0.65	+0.96	−0.06	−0.79	−2.70	+0.93	+1.46	−0.18	−1.19
32	−0.90	+0.35	+0.52	−0.03	−0.41	−1.54	+0.50	+0.77	−0.10	−0.69

Table 3: Peak-memory reduction (%) relative to Muon with FP8 states for the 500M model at sequence length 1024. Microbatch sizes count sequences per device. All methods use FP8 states. Positive (blue) values indicate lower peak memory and negative (red) values higher peak memory at the same microbatch and compute precision.

A.7 Runtime with Adaptive Microbatches

We measure mean training-step time after 10 warmup steps, using 50 timed steps for each configuration. For SOAP, GaLore, FRUGAL, FRUGAL MM, and APOLLO, the timed window contains one projection or preconditioner refresh. At each effective batch size, every optimizer uses the largest fitting microbatch that divides the batch. The reported step time includes any gradient accumulation needed to complete the effective batch.

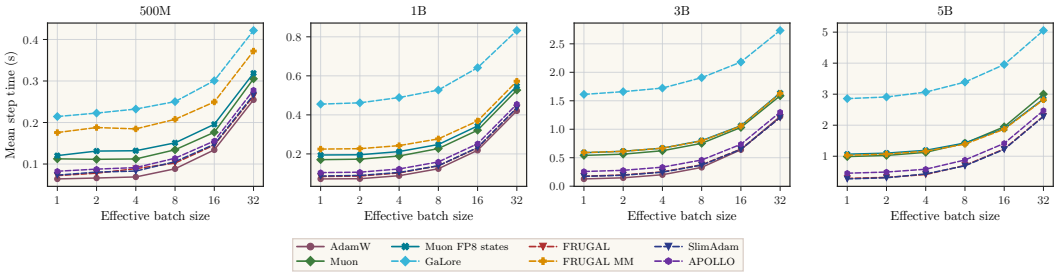

Figure 18: Mean step time under adaptive microbatch selection. Each optimizer uses the largest fitting microbatch at every effective batch size. Step time includes gradient accumulation and amortizes periodic optimizer work over its refresh interval.

Tables 4–7 report the exact step times and the largest fitting microbatch. The AdamW / FP8 states rows use a fused update that stores FP8 moments and performs the optimizer arithmetic in FP32. The tables also include SOAP and AdEMAMix, including configurations that do not fit at a microbatch of one sequence.

At small effective batches, matrix operations dominate step time. Relative to AdamW, Muon takes $1.8\times$ as long on 500M and $4.2\times$ on 3B at one sequence, but only $1.2\times$ and $1.3\times$ at 32 sequences. The increase with model size is consistent with the cost of matrix orthogonalization, while larger batches amortize that cost over more tokens. This batch effect matters for the quality runs, which use 128 sequences per effective step. SOAP's preconditioner refresh and GaLore's periodic SVD show a stronger batch effect: on 500M, their ratios to AdamW fall from $11\times$ and $3.3\times$ at one sequence to $3.6\times$ and $1.7\times$ at 32. Across 500M, 1B, and 3B, FRUGAL, SlimAdam, and APOLLO stay within about 9% of AdamW at 32 sequences. The shorter steps of FRUGAL and SlimAdam nevertheless leave them behind Muon in the matched-time comparison (Figure 6). FP8 states barely

Optimizer	Max MB	1	2	4	8	16	32
AdamW	32	64.1	66.3	68.9	88.5	134.1	254.7
AdamW / FP8 states	32	65.2	67.1	70.0	90.7	134.2	259.2
Muon	32	112.5	111.3	112.2	134.3	176.5	305.3
Muon / FP8 states	32	120.3	131.2	132.0	151.0	195.7	318.8
SOAP	32	716.5	722.1	725.9	749.2	795.6	912.1
AdEMAMix	32	79.0	85.0	87.5	108.9	150.3	274.0
GaLore	32	214.4	222.7	232.4	250.3	300.8	421.7
FRUGAL	32	72.1	77.9	88.9	102.3	145.7	263.6
FRUGAL MM	32	175.8	187.9	184.6	207.7	249.5	372.1
SlimAdam	32	73.2	80.6	83.0	105.6	147.4	266.3
APOLLO	32	82.6	87.7	91.4	113.9	155.5	278.0

Table 4: Adaptive-microbatch step times for the 500M model. Values are milliseconds per effective training step and include gradient accumulation. Column headers give effective batch sizes in sequences. Max MB is the largest fitting per-device microbatch in sequences.

Optimizer	Max MB	1	2	4	8	16	32
AdamW	32	71.8	72.9	88.3	123.9	218.4	420.6
AdamW / FP8 states	32	69.7	71.7	87.9	125.1	218.8	416.5
Muon	32	170.5	173.0	189.2	227.3	321.4	526.8
Muon / FP8 states	32	194.4	195.6	212.0	248.2	343.2	547.8
SOAP	16	1074.5	1071.1	1093.2	1131.7	1226.2	1428.9
AdEMAMix	32	98.5	101.5	118.0	151.3	244.1	446.7
GaLore	32	455.1	461.8	488.8	527.3	641.8	832.5
FRUGAL	32	84.1	86.5	102.5	139.2	231.1	435.0
FRUGAL MM	32	224.6	226.8	242.5	277.0	368.9	572.5
SlimAdam	32	86.7	89.7	104.9	139.1	232.6	436.0
APOLLO	32	104.2	106.4	122.8	158.9	252.0	455.1

Table 5: Adaptive-microbatch step times for the 1B model. Values are milliseconds per effective training step and include gradient accumulation. Column headers give effective batch sizes in sequences. Max MB is the largest fitting per-device microbatch in sequences.

Optimizer	Max MB	1	2	4	8	16	32
AdamW	8	127.6	147.1	201.3	330.1	641.3	1224.7
AdamW / FP8 states	8	126.2	144.7	198.1	325.4	596.4	1130.0
Muon	8	541.0	562.8	622.6	752.6	1031.8	1588.2
Muon / FP8 states	16	587.6	613.1	667.3	800.2	1061.9	1636.3
SOAP	OOM	OOM	OOM	OOM	OOM	OOM	OOM
AdEMAMix	4	209.3	229.2	284.2	469.5	790.0	1446.6
GaLore	8	1612.1	1660.2	1723.0	1905.7	2180.3	2734.6
FRUGAL	8	176.5	196.9	252.1	376.5	651.3	1206.3
FRUGAL MM	16	590.7	612.5	667.6	795.0	1055.0	1629.8
SlimAdam	8	172.1	191.6	245.5	369.2	647.7	1205.3
APOLLO	8	257.9	278.6	334.1	459.9	735.8	1299.8

Table 6: Adaptive-microbatch step times for the 3B model. Values are milliseconds per effective training step and include gradient accumulation. Column headers give effective batch sizes in sequences. Max MB is the largest fitting per-device microbatch in sequences.

Optimizer	Max MB	1	2	4	8	16	32
AdamW	OOM	OOM	OOM	OOM	OOM	OOM	OOM
AdamW / FP8 states	4	232.8	265.4	342.5	586.4	1075.0	2045.1
Muon	4	988.6	1025.4	1125.6	1419.8	1949.4	3001.9
Muon / FP8 states	4	1065.3	1099.3	1190.4	1425.6	1889.4	2842.2
SOAP	OOM	OOM	OOM	OOM	OOM	OOM	OOM
AdEMAMix	OOM	OOM	OOM	OOM	OOM	OOM	OOM
GaLore	2	2857.3	2909.4	3066.4	3389.7	3956.0	5053.6
FRUGAL	4	287.1	318.8	409.6	704.7	1234.4	2291.6
FRUGAL MM	4	1031.3	1069.7	1161.9	1390.8	1871.0	2820.1
SlimAdam	2	268.3	305.5	430.4	690.5	1221.0	2280.4
APOLLO	4	452.6	490.3	581.3	878.7	1414.1	2472.6

Table 7: Adaptive-microbatch step times for the 5B model. Values are milliseconds per effective training step and include gradient accumulation. Column headers give effective batch sizes in sequences. Max MB is the largest fitting per-device microbatch in sequences.

change AdamW's step time at 500M and 1B and make its 3B run 8% faster at 32 sequences. Muon's FP8-state path is up to 18% slower through 3B, although it is faster at larger batches on 5B.

A.8 MoE Results

Table 8 gives the final test and validation losses across compute and state precisions. Table 9 compares peak memory when AdamW and Muon use FP8 states and the structural methods use FP32 states. Table 10 gives the corresponding comparison when FRUGAL and SlimAdam also use FP8 states.

	Test loss				Validation loss			
	W16A16G32		W8A8G8		W16A16G32		W8A8G8	
Optimizer	FP32	FP8	FP32	FP8	FP32	FP8	FP32	FP8
AdamW	2.659	2.689	2.688	2.676	2.676	2.706	2.705	2.693
Muon	**2.635**	**2.633**	**2.639**	**2.639**	**2.651**	**2.649**	**2.655**	**2.655**
FRUGAL	2.747	2.755	2.746	2.765	2.761	2.774	2.765	2.784
SlimAdam	2.666	2.666	2.669	2.669	2.679	2.679	2.682	2.682

Table 8: Final loss (↓) of the MoE model after 17,242 steps (7.34B tokens) with FP32 or FP8 optimizer states. Validation loss is a quadratic fit to the validation points of the last 2,000 steps, evaluated at the final step. The best value in each column is bold.

	W16A16G32					W8A8G8				
Microbatch	1	2	4	8	16	1	2	4	8	16
AdamW	+4.46	+2.64	+1.07	+1.08	+1.08	+4.46	+3.26	+1.07	+1.07	+1.08
Muon	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
FRUGAL	+1.00	+1.51	+1.50	+1.51	+1.51	+0.69	+1.51	+1.50	+1.51	+1.50
SlimAdam	+2.54	+3.05	+3.05	+3.05	+3.05	+2.25	+3.06	+3.06	+3.06	+3.05
Muon / FP8 (GB)	14.58	16.41	21.11	30.48	49.25	16.45	17.66	21.72	29.80	45.99

Table 9: Extra peak memory (GB) relative to Muon with FP8 states for the MoE model at sequence length 2048. Blue indicates lower memory and red higher memory. AdamW and Muon use FP8 states, while structural methods use FP32 states. The last row gives absolute reference memory.

	W16A16G32 / fp8 states		W8A8G8 / fp8 states	
Microbatch	FRUGAL	SlimAdam	FRUGAL	SlimAdam
1	+6.55	−3.64	+7.59	−3.16
2	+2.71	+0.10	+2.50	+0.14
4	+2.09	+0.09	+2.06	+0.13
8	+1.46	+0.07	+1.49	+0.08
16	+0.90	+0.02	+0.98	+0.06

Table 10: Peak allocated memory reduction (%) relative to Muon with FP8 states for the MoE model. FRUGAL and SlimAdam also use FP8 states. Entries are $100(M_{\text{Muon/FP8}} - M_{\text{method/FP8}})/M_{\text{Muon/FP8}}$ at the same microbatch and compute precision. Blue cells (positive values) indicate lower memory and red cells (negative values) higher memory. Measurements use sequence length 2048.

With FP8 states for the baselines and FP32 states for the structural methods, Muon has the lowest peak memory at every measured microbatch (Table 9). Quantizing the structural methods' states gives FRUGAL the only material advantage over Muon: 6.55% to 7.59% at a microbatch of one, falling below 1% at 16. SlimAdam is within 0.14% of Muon beyond a microbatch of one (Table 10). Yet FP8-state FRUGAL's validation loss remains 0.125 to 0.129 above FP8-state Muon's (Table 8), so the memory saving does not offset its quality gap. Together with the dense-model comparison in Figure 5, these results favor Muon with FP8 states for the measured tradeoff between memory and quality.

B Exact Loss and Memory Results

B.1 Seed Variability

Tables 11 and 12 give the final validation losses underlying the seed-variability estimates in Section 4.1. The two experiments vary the PyTorch seed and data order separately.

Optimizer	Seed 1	Seed 2	Seed 3
AdamW	3.019	3.019	3.020
Muon	2.987	2.989	2.988

Table 11: Final validation loss for the 257M model at $1 \times C$ across PyTorch random seeds, rounded to three decimal places.

Optimizer	Seed 1	Seed 2	Seed 3
AdamW	3.0119	3.0150	3.0128
Muon	2.9823	2.9822	2.9825

Table 12: Final validation loss for the 257M model at $1 \times C$ across data-order seeds.

B.2 BF16 Endpoint Losses

Tables 13 and 14 give the final losses for the 257M and 500M BF16 trajectories in Figure 3.

	Baselines					Memory-efficient								
Budget	AdamW	Muon	SOAP	AdEMAMix	Shampoo	GaLore	Fira	FRUGAL	FRUGAL MM	SCALE	APOLLO	APOLLO-mini	SlimAdam	Adam-mini
$1 \times C$	3.024	**2.989**	2.990	3.000	3.039	3.068	3.117	3.089	3.069	3.098	3.093	3.112	3.017	3.056
$2 \times C$	2.950	2.923	**2.911**	2.918	2.957	2.984	3.036	3.002	2.997	3.012	3.012	3.019	2.938	2.979
$4 \times C$	2.891	2.867	**2.850**	2.855	2.939	2.917	2.969	2.935	2.942	2.946	2.955	2.951	2.878	2.920

Table 13: Final validation loss (↓) for the 257M model under `W16A16G32` with native optimizer states.

	Baselines					Memory-efficient				
Budget	AdamW	Muon	SOAP	AdEMAMix	Shampoo	GaLore	FRUGAL	FRUGAL MM	APOLLO	SlimAdam
$1 \times C$	2.793	**2.761**	2.769	2.801	2.868	2.882	2.918	2.906	2.908	2.819
$2 \times C$	2.728	**2.699**	2.700	2.719	2.794	2.810	2.861	2.859	2.835	2.754

Table 14: Final validation loss (↓) for the 500M model under `W16A16G32` with FP32 optimizer states.

B.3 FP8-State Losses across Training Budgets

Table 15 reports the exact 257M endpoints plotted in Figure 8.

Optimizer	$1 \times C$	$2 \times C$	$4 \times C$	$8 \times C$
AdamW	3.021	2.948	2.890	2.844
Muon	2.989	2.923	2.869	2.824
FRUGAL	3.094	3.008	2.941	2.887
SlimAdam	3.015	2.938	2.876	2.827

Table 15: Final validation loss (↓) for the 257M model under `W16A16G32` with FP8 optimizer states across training budgets.

B.4 Retained-Fraction Ablation Endpoints

Increasing ρ could recover baseline quality before combining projection with FP8 state quantization. FRUGAL improves from 3.110 at $\rho = 0.05$ to 3.059 at $\rho = 1$, yet still trails AdamW by 0.035 and Muon by 0.070 (Tables 16 and 13). Moving from $\rho = 0.25$ to 1 quadruples nominal moment storage while reducing loss by only 0.033. GaLore stays near 3.06 from $\rho = 0.4$ to 0.9, but the single run at $\rho = 1$ has anomalously high loss. Even at full rank, GaLore's update is not mathematically equivalent to AdamW. The weak response to higher ρ over the rest of the sweep suggests that discarded directions alone do not explain the gap. Periodic subspace refresh could also contribute, although this ablation does not isolate its effect. Projected FP32 moments at $\rho = 0.25$ already match the nominal storage of dense FP8 moments, so retaining more directions raises state storage without recovering baseline quality.

Table 16: Final validation loss (↓) by retained fraction ρ for the 257M model at $1 \times C$ in BF16.

ρ	1.00	0.90	0.80	0.70	0.60	0.50	0.40	0.30	0.25	0.20	0.15	0.10	0.05
FRUGAL	3.059	3.061	3.066	3.069	3.070	3.081	3.083	3.090	3.092	3.096	3.101	3.106	3.110
GaLore	4.542	3.068	3.063	3.062	3.060	3.061	3.060	3.065	3.068	3.072	3.077	3.079	3.087

B.5 Losses across Precision Configurations

Tables 17, 18, 19, and 20 report final validation losses across model sizes and token budgets, grouped by computation and optimizer-state precision.

	257M			500M	
Optimizer	$1 \times C$	$2 \times C$	$4 \times C$	$1 \times C$	$2 \times C$
AdamW	3.024	2.950	2.891	2.793	2.728
Muon	2.989	2.923	2.867	2.761	2.699
SOAP	2.990	2.911	2.850	2.769	2.700
AdEMAMix	3.000	2.918	2.855	2.801	2.719
Shampoo	3.039	2.957	2.939	2.868	2.794
GaLore	3.068	2.984	2.917	2.882	2.810
FRUGAL	3.089	3.002	2.935	2.918	2.861
FRUGAL MM	3.069	2.997	2.942	2.906	2.859
APOLLO	3.093	3.012	2.955	2.908	2.835
SlimAdam	3.017	2.938	2.878	2.819	2.754

Table 17: Final validation loss for `W16A16G32`. NR denotes a missing result and DIV divergence.

	257M			500M	
Optimizer	$1 \times C$	$2 \times C$	$4 \times C$	$1 \times C$	$2 \times C$
AdamW	3.028	2.956	2.898	2.806	2.742
Muon	2.995	2.929	2.874	2.779	2.717
SOAP	3.004	2.929	2.873	2.798	2.731
AdEMAMix	3.007	2.926	2.863	2.788	2.724
GaLore	3.080	2.998	2.930	2.882	2.813
FRUGAL	3.096	3.013	2.948	2.899	2.829
FRUGAL MM	NR	NR	NR	2.909	2.863
APOLLO	3.105	3.023	2.964	2.900	2.832
SlimAdam	3.027	2.951	2.889	2.820	2.754

Table 18: Final validation loss for `W8A8G8`. NR denotes a missing result and DIV divergence.

B.6 Loss and Peak Memory at Microbatch 32

Table 21 gives final loss and measured peak memory for nine optimizers under `W16A16G32` and `W8A8G8`, with FP32 or FP8 states. Memory is profiled at a microbatch of 32 sequences. Quantizing

Optimizer	500M $1 \times C$
AdamW	2.790
Muon	2.764
SOAP	2.771
AdEMAMix	DIV
GaLore	2.875
FRUGAL	2.895
FRUGAL MM	2.906
APOLLO	2.903
SlimAdam	2.811

Table 19: Final validation loss for the 500M model under `W16A16G32 / fp8 states`. DIV denotes divergence.

	257M			500M	
Optimizer	$1 \times C$	$2 \times C$	$4 \times C$	$1 \times C$	$2 \times C$
AdamW	3.029	2.957	2.898	2.804	2.740
Muon	2.996	2.930	2.876	2.775	2.712
SOAP	3.002	2.929	2.872	2.784	2.720
AdEMAMix	DIV	DIV	DIV	2.784	2.719
GaLore	3.079	2.998	2.928	2.882	2.814
FRUGAL	3.097	3.008	2.937	2.897	2.828
FRUGAL MM	NR	NR	NR	2.911	2.863
APOLLO	3.069	2.979	2.913	2.897	2.829
SlimAdam	3.044	2.970	2.907	2.811	2.752

Table 20: Final validation loss for `W8A8G8 / fp8 states`. NR denotes a missing result and DIV divergence.

optimizer states reduces peak memory by roughly the bytes removed from the moments. In the 500M-parameter model, AdamW's two FP32 moments occupy 3.96 GB. Grouped FP8 storage is expected to save about 2.9 GB. The measured peak falls by 2.85 GB under either compute precision. AdEMAMix's three moments save 4.2 GB, while Muon's single momentum on matrix parameters saves about 1.75 GB. SOAP saves 2.83 GB, similar to AdamW, because its Kronecker preconditioners and eigenbases remain in FP32. Those tensors keep SOAP's peak 8.6 to 9.7 GB above Muon's.

Compute precision has a larger effect at this microbatch. Switching to `W8A8G8` reduces peak memory by about 24.9 GB for every optimizer, consistent with a largely optimizer-independent activation saving. The five structured methods with FP32 states nevertheless use 0.7 to 2.7 GB more peak memory than FP8-state Muon. Their smaller moment buffers coexist with projection bases, gradients retained for complementary updates where needed, and refresh workspaces. GaLore's periodic SVD requires additional workspace, consistent with its FP8-state peak exceeding Muon's at every measured microbatch (Table 3).

C Advanced Optimizers: Algorithmic Details

We first define the Adam direction used by several optimizers below. It forms a coordinate-wise adaptive update from bias-corrected first and second gradient moments (Kingma & Ba, 2015). AdamW applies decoupled weight decay outside this routine (Loshchilov & Hutter, 2019), while projected methods call it on reduced-dimensional gradients.

	W16A16G32				W8A8G8			
	FP32 states		FP8 states		FP32 states		FP8 states	
Optimizer	Val. loss	Peak (GB)	Val. loss	Peak (GB)	Val. loss	Peak (GB)	Val. loss	Peak (GB)
AdamW	2.793	68.42	2.790	65.57	2.806	43.52	2.804	40.67
Muon	2.761	66.96	2.764	65.21	2.779	42.06	2.775	40.29
SOAP	2.769	76.63	2.771	73.80	2.798	51.72	2.784	48.87
AdEMAMix	2.801	70.40	DIV	66.18	2.788	45.50	2.784	41.27
GaLore	2.882	67.09	2.875	65.80	2.882	42.19	2.882	40.91
FRUGAL	2.918	66.25	2.895	64.98	2.899	41.35	2.897	40.09
FRUGAL MM	2.906	65.89	2.906	64.87	2.909	40.98	2.911	39.98
APOLLO	2.908	66.50	2.903	65.23	2.900	41.58	2.897	40.33
SlimAdam	2.819	67.91	2.811	65.48	2.820	43.01	2.811	40.57

Table 21: Final validation loss (↓) at $1 \times C$ and peak allocated memory (↓) for the 500M model under two compute precisions and two optimizer-state formats. Memory is profiled separately at a microbatch of 32 sequences, each of length 1024. NR denotes a missing training result and DIV training divergence.

Algorithm 1 AdamUpdate

Parameters: β_1, β_2 – moment decays; ε – stabilizer.

function ADAMUPDATE(G_t, M_{t-1}, V_{t-1})
  $M_t \leftarrow \beta_1 M_{t-1} + (1-\beta_1) G_t$
  $V_t \leftarrow \beta_2 V_{t-1} + (1-\beta_2)(G_t \odot G_t)$
  $\hat{M}_t \leftarrow M_t / (1-\beta_1^t)$
  $\hat{V}_t \leftarrow V_t / (1-\beta_2^t)$
  $N_t \leftarrow \hat{M}_t / (\sqrt{\hat{V}_t} + \varepsilon)$
  return $N_t,\ M_t,\ V_t$
end function

C.1 ADEMAMIX: BETTER GRADIENT HISTORY

AdEMAMix (Pagliardini et al., 2024) extends Adam's gradient history by maintaining two separate EMA buffers alongside the standard second moment. The fast accumulator tracks the recent gradient direction with $\beta_1 \approx 0.9$, as in Adam (Kingma & Ba, 2015). A second, slow accumulator uses a much larger decay coefficient (β_3 up to 0.9999) and integrates gradient signal over thousands of preceding steps. At each iteration the update direction is a weighted combination of the two accumulators, blending short-horizon reactivity with long-horizon consistency. To prevent the slow accumulator from interfering with early optimization dynamics, β_3 is warmed up from a value close to β_1 to its target over an initial training phase.

Algorithm 2 AdEMAMixUpdate

function ADEMAMIXUPDATE($G_t, M_{t-1}, S_{t-1}, V_{t-1}$)
  $M_t \leftarrow \beta_1 M_{t-1} + (1-\beta_1) G_t$ ▷ fast EMA
  $S_t \leftarrow \beta_3 S_{t-1} + (1-\beta_3) G_t$ ▷ slow EMA (uncorrected)
  $V_t \leftarrow \beta_2 V_{t-1} + (1-\beta_2)(G_t \odot G_t)$
  $\hat{M}_t \leftarrow M_t / (1-\beta_1^t)$
  $\hat{V}_t \leftarrow V_t / (1-\beta_2^t)$
  $N_t \leftarrow \dfrac{\hat{M}_t + \alpha\, S_t}{\sqrt{\hat{V}_t} + \varepsilon}$
  return $N_t,\ M_t,\ S_t,\ V_t$
end function

AdEMAMix introduces no matrix operations and no additional preconditioner, adding only a single gradient-shaped buffer to Adam's memory footprint.

C.2 SHAMPOO AND SOAP: FULL-MATRIX PRECONDITIONING

Algorithm 3 ShampooUpdate

$G_t, M_t \in \mathbb{R}^{m\times n}$
function SHAMPOOUPDATE($G_t,\ M_{t-1},\ L_{t-1},\ R_{t-1},\ \eta$)
  $M_t \leftarrow \beta_1 M_{t-1} + (1-\beta_1)\, G_t$ ▷ (torch implementation)
  $L_t \leftarrow L_{t-1} + G_t G_t^\top$
  $R_t \leftarrow R_{t-1} + G_t^\top G_t$
  $U_t \leftarrow L_t^{-1/4}\, M_t\, R_t^{-1/4}$ ▷ preconditioned update
  $W_{t+1} \leftarrow W_t - \eta\, U_t$
  return $W_{t+1},\ M_t,\ L_t,\ R_t$
end function

Adam's (Kingma & Ba, 2015) second-moment accumulator is a diagonal preconditioner: it rescales each parameter independently, ignoring all correlations between entries of the same weight matrix. A more principled approach, motivated by K–FAC (Martens & Grosse, 2015), is to approximate the full curvature matrix using its Kronecker product structure. For a linear layer $W \in \mathbb{R}^{m\times n}$, the Fisher information matrix admits the approximation $F \approx A \otimes B$ with $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{m\times m}$, so the natural gradient update $F^{-1}\mathrm{vec}(G)$ can be computed as $B^{-1}GA^{-1}$ at cost $O(m^3+n^3)$ rather than $O(m^3n^3)$.

Shampoo (Gupta et al., 2018) operationalizes this idea by maintaining running averages of the left and right gradient covariance matrices,

$$L_t = \sum_{s\le t} G_s G_s^\top \in \mathbb{R}^{m\times m},$$

$$R_t = \sum_{s\le t} G_s^\top G_s \in \mathbb{R}^{n\times n},$$

and preconditions the update as $\widetilde{G}_t = L_t^{-1/4} G_t R_t^{-1/4}$. This captures the Kronecker structure of the curvature that Adam's diagonal estimate cannot represent. The main cost is the periodic computation of the matrix inverse fourth-roots $L_t^{-1/4}$ and $R_t^{-1/4}$, typically done via coupled Newton iteration, which can be amortized by updating the preconditioner every T steps. However, reducing this frequency too aggressively degrades performance, since the preconditioner becomes stale.

Algorithm 4 SOAPUpdate (QR + Adam in rotated basis)

$G_t, M_t, V_t \in \mathbb{R}^{m\times n}$
function SOAPUPDATE($G_t,\ L_{t-1},\ R_{t-1},\ M_{t-1},\ V_{t-1}$)
  $L_t \leftarrow \beta_2 L_{t-1} + (1-\beta_2)\, G_t G_t^\top$
  $R_t \leftarrow \beta_2 R_{t-1} + (1-\beta_2)\, G_t^\top G_t$
  $Q_{L,t}, R_{L,t} \leftarrow \mathrm{QR}(L_t)$ ▷ left QR
  $Q_{R,t}, R_{R,t} \leftarrow \mathrm{QR}(R_t)$ ▷ right QR
  $\tilde{G}_t \leftarrow Q_{L,t}^\top\, G_t\, Q_{R,t}$ ▷ project gradient
  $N_t, M_t, V_t \leftarrow \mathrm{AdamUpdate}(\tilde{G}_t, M_{t-1}, V_{t-1}, \beta_1, \beta_2)$
  return $Q_{L,t}\, N_t\, Q_{R,t}^\top$ ▷ project back and return
end function

SOAP (Vyas et al., 2025) resolves this tension by establishing a formal equivalence: Shampoo with the $1/2$ exponent is equivalent to running Adafactor (Shazeer & Stern, 2018) in the eigenbasis of Shampoo's preconditioner. The key geometric insight is that rotating into the eigenbasis of L_t and R_t diagonalizes the Kronecker-factored preconditioner, so Adam's diagonal second-moment accumulator

is no longer an approximation in this basis, it is exact. This motivates running Adam's moment updates directly in the rotated coordinate system defined by the current eigenvectors of L_t and R_t, rather than in the original parameter space, where the diagonal assumption is a crude approximation of the true curvature.

C.3 MUON: SPECTRAL PRECONDITIONING VIA ORTHOGONALISATION

Muon (Jordan et al., 2024) takes a different approach to addressing Adam's (Kingma & Ba, 2015) diagonal preconditioning limitation. Muon is motivated by the observation that the natural cost of an update ΔW to a weight matrix is its spectral norm $\|\Delta W\|_2$, the largest singular value, which measures how much the update changes the operator's action on inputs. Steepest descent under the spectral norm yields an update equal to the nearest orthogonal matrix (in Frobenius distance) to the gradient (Bernstein & Newhouse, 2024), i.e., the polar factor $UV^\top$ where $G = U\Sigma V^\top$ is the SVD of the gradient. Muon approximates this polar factor efficiently using Newton–Schulz iteration.

Algorithm 5 MuonUpdate

$G_t, M_t \in \mathbb{R}^{m\times n}$
function MUONUPDATE($G_t,\ M_{t-1}$)
  $M_t \leftarrow \beta M_{t-1} + (1-\beta)\, G_t$ ▷ momentum (often Nesterov-style)
  $U_t \leftarrow \mathrm{NewtonSchulz}(M_t)$ ▷ 4–5 iters
  return $U_t,\ M_t$
end function
function NEWTONSCHULZ(G, steps$=5$)
  $X \leftarrow G/(\|G\|_F + \epsilon)$
  for $i = 1$ to steps **do**
    $A \leftarrow X^\top X$
    $X \leftarrow aX + (bA + cA^2)X$ ▷ e.g. $a = 3.4445$, $b = -4.7750$, $c = 2.0315$
  end for
  return X
end function

Concretely, Muon applies standard SGD with Nesterov momentum (Nesterov, 1983; Sutskever et al., 2013) to accumulate M_t, then replaces the update with its orthogonalized version:

$$W_t \leftarrow W_{t-1} - \eta\, \mathrm{NS}(M_t),$$

where $\mathrm{NS}(\cdot)$ denotes the Newton–Schulz approximation to the polar factor. No second-moment accumulator is maintained, reducing optimizer state to a single gradient-shaped buffer per parameter matrix. Muon is applied only to weight matrices and is paired with AdamW (Loshchilov & Hutter, 2019) for embeddings, the classifier head, and all scalar or vector parameters. This hybrid mode is the recommended default and is used in all experiments reported in Section 4.2.

C.4 GALORE-LIKE METHODS

C.4.1 GALORE (ZHAO ET AL., 2024)

Algorithm 6 GaLoreUpdate

Parameters: T – SVD period; r – target rank;
$G_t \in \mathbb{R}^{m\times n}, \quad P_t \in \mathbb{R}^{m\times r}$
function GALOREUPDATE(G_t, P_{t-1}, t)
  if $t \bmod T = 0$ **then**
    $U, \Sigma, V^\top \leftarrow \mathrm{SVD}(G_t)$
    return $U[:, 1:r]$ ▷ Update projector
  else
    return P_{t-1} ▷ Reuse the previous projector
  end if
end function

Algorithm 7 Adam with GaLore

Parameters: η – lr; α – projection scaling.

$M_0 \in \mathbb{R}^{r\times n} \leftarrow 0, \quad V_0 \in \mathbb{R}^{r\times n} \leftarrow 0, \quad t \leftarrow 0$
for $t = 1, 2, \ldots$ **do**
  $G_t \leftarrow \nabla_W f(W_t)$
  $P_t \leftarrow \text{GaLoreUpdate}(G_t, P_{t-1}, t, T, r)$
  $R_t \in \mathbb{R}^{r\times n} \leftarrow P_t^\top G_t$ ▷ Project gradient
  $N_t, M_t, V_t \leftarrow \text{AdamUpdate}(R_t, M_{t-1}, V_{t-1})$
  $\widetilde{G}_t \in \mathbb{R}^{m\times n} \leftarrow \alpha\, P_t N_t$ ▷ Project back
  $W_t \leftarrow W_{t-1} - \eta\, \widetilde{G}_t$
  $t \leftarrow t + 1$
end for

The core idea is to maintain full-rank weights while projecting gradients into a low-rank subspace, so that optimizer states are stored in this compact space rather than in the full parameter space $\mathbb{R}^{m\times n}$. The projection matrix $P_t \in \mathbb{R}^{m\times r}$ is obtained by taking the top-r left singular vectors of the gradient matrix G_t, and is recomputed every T steps to track the slowly evolving gradient subspace.

Unlike LoRA (Hu et al., 2021), which restricts optimization to a fixed low-rank adaptor and thereby limits the parameter search to a low-rank subspace throughout training, GaLore enables the model to traverse the entire parameter space by rotating the subspace over time. This approach exploits the empirical observation that weight gradients naturally become low-rank during training.

In terms of memory, the first- and second-moment accumulators have shape $\mathbb{R}^{r\times n}$ rather than $\mathbb{R}^{m\times n}$, yielding a saving proportional to $r/\min(m, n)$. The SVD recomputation every T steps introduces a periodic overhead which the other gradient-projection methods surveyed here aim to reduce.

GaLore can be viewed as the root method from which the other gradient-projection approaches surveyed here develop their ideas, each addressing some of its practical limitations: the cost of SVD, the treatment of the residual gradient, or the frequency of subspace updates.

C.4.2 FIRA (CHEN ET AL., 2025)

Algorithm 8 Fira with Adam

Parameters: η – lr; α – proj. scaling; γ – norm-growth cap.

$M_0 \in \mathbb{R}^{r\times n} \leftarrow 0, \quad V_0 \in \mathbb{R}^{r\times n} \leftarrow 0, \quad t \leftarrow 0$
for $t = 1, 2, \ldots$ **do**
  $G_t \in \mathbb{R}^{m\times n} \leftarrow \nabla_W f(W_t)$
  $P_t \leftarrow \text{GaLoreUpdate}(G_t, P_{t-1}, t)$
  $R_t \in \mathbb{R}^{r\times n} \leftarrow P_t^\top G_t$ ▷ Project gradient
  $S_t \in \mathbb{R}^{m\times n} \leftarrow (I - P_t P_t^\top) G_t$
  $N_t, M_t, V_t \leftarrow \text{AdamUpdate}(R_t, M_{t-1}, V_{t-1})$
  $K \leftarrow \left[\frac{\|N_t[:,1]\|_2}{\|R_t[:,1]\|_2+\epsilon}, \ldots, \frac{\|N_t[:,n]\|_2}{\|R_t[:,n]\|_2+\epsilon}\right]$ ▷ Norm-based scaling
  $S_t \leftarrow [k_1 S_t[:,1], \ldots, k_n S_t[:,n]]$
  $S_t \leftarrow S_t \cdot \gamma \,/\, \max\left\{\frac{\|S_t\|_2}{\|S_{t-1}\|_2+\epsilon}, \gamma\right\}$ ▷ Norm growth limiter
  $\tilde{G}_t \in \mathbb{R}^{m\times n} \leftarrow \alpha\,(P_t N_t + S_t)$ ▷ Project back
  $W_t \leftarrow W_{t-1} - \eta\, \tilde{G}_t$
  $t \leftarrow t + 1$
end for

The key departure from GaLore is that Fira does not discard the residual component of the gradient. After projecting G_t onto the low-rank subspace, Fira explicitly retains the orthogonal complement

$$S_t = (I - P_t P_t^\top)\, G_t \in \mathbb{R}^{m\times n},$$

which captures the part of the gradient lying outside the subspace.

The key empirical observation is that the column-wise ratio between Adam's (Kingma & Ba, 2015) adapted gradient N_t and the raw projected gradient R_t is nearly the same in low-rank training as in full-rank training. Following this observation, Fira introduces Norm-based scaling. The scaling that Adam applies column-by-column in the low-rank space is a good proxy for the scaling it would apply in full rank. Fira therefore reuses this low-rank scaling to rescale each column of S_t:

$$k_j = \frac{\|N_t[:, j]\|_2}{\|R_t[:, j]\|_2 + \epsilon}, \qquad S_t[:, j] \leftarrow k_j \, S_t[:, j].$$

A practical issue with norm-based scaling is that it can amplify sudden rises in the gradient norm, leading to loss spikes. To address this, Fira introduces a Norm-growth limiter, which caps the relative increase of $\|S_t\|$ between consecutive steps by a factor γ:

$$S_t \leftarrow S_t \cdot \gamma \Big/ \max\left\{ \frac{\|S_t\|_2}{\|S_{t-1}\|_2 + \epsilon}, \, \gamma \right\}.$$

If the norm grows by less than γ, the limiter is inactive and S_t is unchanged. If it grows faster, the update is rescaled so that the increase is exactly γ, converting abrupt spikes into gradual changes and stabilizing training.

C.4.3 APOLLO (Zhu et al., 2025)

APOLLO shares the same norm-based scaling principle as Fira (Chen et al., 2025), but applies it in a simpler setting. Rather than tracking full-size Adam moments, APOLLO stores first and second moments in the projected space and uses them to compute a coarse, channel-wise (or scalar) scaling factor. For channel j, this factor is computed by exactly the same ratio used in Fira:

$$s_j^R = \frac{\|N_t[:, j]\|_2}{\|R_t[:, j]\|_2 + \epsilon},$$

where $R_t = P_t G_t$ is the low-rank projected gradient and N_t is the Adam-adapted version of R_t. The intuition is identical: the column-wise ratio between the adapted and raw projected gradients approximates the scaling Adam would apply in full rank.

The main difference from Fira lies in the projection itself. Instead of maintaining a subspace via SVD, APOLLO re-samples a random Gaussian projector $P_t \in \mathbb{R}^{r \times m}$ every T steps, which avoids the cost of any SVD entirely. The weight update then applies the recovered scaling directly to the full gradient:

$$W_t \leftarrow W_{t-1} - \eta \, \alpha \, G_t \cdot S,$$

where $S = \mathrm{diag}(s_1^R, \ldots, s_n^R)$.

Algorithm 9 Adam with APOLLO

Parameters: η – lr; α – proj. scaling; T – proj. period; r – rank.

$M_0 \in \mathbb{R}^{r \times n} \leftarrow 0, \quad V_0 \in \mathbb{R}^{r \times n} \leftarrow 0, \quad t \leftarrow 0$
for $t = 1, 2, \ldots$ **do**
  $G_t \in \mathbb{R}^{m \times n} \leftarrow \nabla_W f(W_t)$
  if $t \bmod T = 0$ **then**
    $P_t \in \mathbb{R}^{r \times m} \leftarrow \mathcal{N}(0, 1/r)$
  else
    $P_t \leftarrow P_{t-1}$
  end if
  $R_t \in \mathbb{R}^{r \times n} \leftarrow P_t \, G_t$ ▷ Project gradient
  $N_t, M_t, V_t \leftarrow \mathrm{AdamUpdate}(R_t, M_{t-1}, V_{t-1})$
  $S \in \mathbb{R}^{n \times n} \leftarrow \mathrm{diag}\big(s_1^R, \ldots, s_n^R\big)$ ▷ Channel-wise scaling
  $W_t \leftarrow W_{t-1} - \eta \, \alpha \, G_t \cdot S$
end for

The same paper also proposes APOLLO-Mini, which pushes this idea to its logical extreme by collapsing the channel-wise vector of scaling factors to a single tensor-wise scalar:

$$s^R = \frac{\|N_t\|_F}{\|R_t\|_F},$$

trading some accuracy for a further reduction in memory footprint.

C.4.4 COAP (XIAO ET AL., 2025)

The core idea of COAP is to choose the projection matrix P_t so that it captures not only the current gradient direction but also the direction of the accumulated momentum. Rather than computing P_t via SVD of the gradient alone, COAP obtains it by minimizing a composite objective that jointly encourages low-rank reconstruction fidelity and alignment with the previous first moment:

$$\min_{P_t} \underbrace{\mathrm{MSE}\left(R_t,\, G_t\right)}_{\text{reconstruction}} \cdot \underbrace{\left(1 - \mathrm{CosSim}(M_{t-1},\, R_t)\right)}_{\text{direction}},$$

where $R_t = G_t P_t$ denotes the projected gradient. The first term penalizes information loss from projecting G_t into the low-rank subspace, while the second term penalizes directional deviation between the current gradient and the previous momentum, discouraging abrupt shifts in the optimization trajectory.

Algorithm 10 COAP

Parameters: η – lr; r – rank.
$M_0 \in \mathbb{R}^{m\times r} \leftarrow 0, \quad V_0 \in \mathbb{R}^{m\times r} \leftarrow 0$
$G_0 \in \mathbb{R}^{m\times n} \leftarrow \nabla_W f(W_0)$
Randomly initialize: $P_0 \in \mathbb{R}^{n\times r}$
$P_0 \leftarrow \text{LowRankSVD}(P_0, G_0)$
for $t = 1, 2, \ldots$ **do**
  $G_t \in \mathbb{R}^{m\times n} \leftarrow \nabla_W f(W_t)$
  $P_t \leftarrow \text{COAPUpdate}(P_{t-1}, G_t, M_{t-1}, t)$
  $R_t \in \mathbb{R}^{m\times r} \leftarrow G_t P_t$ ▷ Project gradient
  $N_t, M_t, V_t \leftarrow \text{AdamUpdate}(R_t, M_{t-1}, V_{t-1})$
  $W_t \leftarrow W_{t-1} - \eta\,(N_t P_t^\top)$ ▷ Project back and step
end for

Algorithm 11 COAPUpdate

Parameters: T – update period; λ – SVD refresh factor.
$G_t \in \mathbb{R}^{m\times n}, \quad P_t \in \mathbb{R}^{n\times r}, \quad M_t \in \mathbb{R}^{m\times r}$
function COAPUPDATE(P_{t-1}, G_t, M_t, t)
  if $t \bmod T = 0$ **then**
    if $t \bmod (\lambda T) = 0$ **then**
    Low-rank SVD
      $Q_{\text{red}}, _ \leftarrow \text{QR}_{\text{red}}(G_t P_{t-1})$
      $U, \Sigma, P_t \leftarrow \text{SVD}(Q_{\text{red}}^T G_t)$
    else
      $P_t \leftarrow (P_{t-1}, G_t, M_{t-1})$ ▷ 1 GD step w.r.t. reconstruction-direction loss
    end if
  else
    $P_t \leftarrow P_{t-1}$
  end if
  return P_t
end function

In practice, this objective is minimized with a single gradient descent step from the previous P_{t-1}, making each update cheap. To prevent the procedure from getting stuck in a poor local solution, the projection matrix is periodically refreshed every λT steps using a low-rank SVD, which serves as an intermediate between a random projector and a full SVD: it inherits the low computational cost of randomized methods while producing a more principled subspace than a purely random one. Between SVD refreshes, the single-step gradient update keeps P_t continuously adapted to the evolving gradient and momentum directions.

C.4.5 LDADAM (ROBERT ET AL., 2025)

Robert et al. (2025) note a theoretical issue with GaLore-like methods: most modern optimizers, including Adam (Kingma & Ba, 2015), maintain coordinate-wise statistics, and these statistics become inconsistent when the coordinate system changes every T steps as in GaLore (Zhao et al., 2024). The proposed remedy is to reproject the accumulated optimizer states onto the new subspace whenever the projection matrix is updated.

Unlike GaLore, LDAdam updates the subspace at every step rather than periodically. To make this tractable, block power iteration with warm initialization replaces the full SVD, keeping per-step adaptation computationally cheap. The subspace is fitted not to the raw gradient alone but to an interpolation of the gradient and the previous momentum, with the interpolation factor set to $\rho = \beta_1$, balancing gradient approximation quality against continuity of the stored optimizer state.

LDAdam also incorporates a generalized error feedback mechanism (Seide et al., 2014; Karimireddy et al., 2019), which accumulates both the gradient compression error and the state transition error incurred when switching subspaces. The error buffer is stored in-place in the gradient accumulation variable, adding no extra memory.

Algorithm 12 LDAdam

$M_0 \in \mathbb{R}^{r\times n} \leftarrow 0; \quad V_0 \in \mathbb{R}^{r\times n} \leftarrow 0; \quad A_0 \in \mathbb{R}^{m\times n} \leftarrow 0$
$P_0 \in \mathbb{R}^{m\times r} \leftarrow \mathrm{SVD}(\nabla_W f(W_0))$
for $t = 1, 2, \ldots, T$ **do**
  Gradient accumulation and error buffer unloading
    $A_t \leftarrow A_t + \nabla_W f(W_t)$
  Learning subspace adaptation
    $B_t \leftarrow \rho U_{t-1}\hat{M}_{t-1} + (1-\rho)A_t$
    $P_t \leftarrow \text{Gram-Schmidt}(B_t B_t^\top \cdot P_{t-1})$
  $M_{t-1/2}, V_{t-1/2} \leftarrow \text{LDUpd}(G_t, P_t, P_{t-1}, M_{t-1}, V_{t-1}, t)$
  Optimizer states Adam-type update
    $a_t \in \mathbb{R}^{r\times n} \leftarrow U_t^\top A_t$
    $N_t, M_t, V_t \leftarrow \text{AdamUpdate}(a_t, M_{t-1/2}, V_{t-1/2})$
  Model update
    $\theta_{t+1} \leftarrow \theta_t - \eta_t P_t \cdot N_t$
  Error buffer loading
    $A_{t+1} \leftarrow A_t - P_t a_t + \dfrac{\beta_1}{1-\beta_1}(P_{t-1}M_{t-1} - P_t M_{t-1/2})$
end for

Algorithm 13 LDUpd (Reprojection)

$G_t \in \mathbb{R}^{m\times n},\ P_t \in \mathbb{R}^{m\times r},\ M_t, V_t \in \mathbb{R}^{r\times n}$
function LDUPD($G_t, P_t, P_{t-1}, M_{t-1}, V_{t-1}, t$)
  $M_{t-1/2} \leftarrow P_t^\top P_{t-1} M_{t-1}$
  $V_{t-1/2} \leftarrow (1-\beta_2^{t-1})\big[(P_t^\top P_{t-1})^2 \cdot (V_{t-1} - M_{t-1}^2) + (P_t^\top P_{t-1} \cdot M_{t-1})^2\big]$
  return $M_{t-1/2}, V_{t-1/2}$
end function

C.4.6 FRUGAL (ZMUSHKO ET AL., 2025)

FRUGAL combines random gradient projection with a space-splitting update rule. On the projection side, it follows APOLLO (Zhu et al., 2025) in using a random projector rather than SVD, avoiding the periodic SVD overhead entirely. It was also observed that when the projection matrix is resampled, simply resetting the optimizer states performs comparably to explicitly reprojecting them onto the new subspace, further simplifying the implementation.

On the update side, FRUGAL splits the gradient into two components. The low-rank projected component is updated with Adam (Kingma & Ba, 2015), preserving adaptive optimization on the

most informative gradient directions. The residual component, lying in the orthogonal complement, is updated with signSGD (Bernstein et al., 2018), which requires no additional optimizer state. This decomposition allows the method to retain the benefits of adaptivity where it matters most while keeping the memory cost of the residual update negligible.

Algorithm 14 FRUGAL

Parameters: η – learning rate;
$M_0 \in \mathbb{R}^{r\times n} \leftarrow 0$; $V_0 \in \mathbb{R}^{r\times n} \leftarrow 0$;
for $t = 1, 2, \ldots, K$ **do**
  $G_t \in \mathbb{R}^{m\times n} \leftarrow \nabla_W f(W_t)$
  if $t \bmod T = 0$ **then**
    $P_t \in \mathbb{R}^{r\times m} \leftarrow \mathcal{N}(0, 1/r)$
  else
    $P_t \leftarrow P_{t-1}$
  end if
  $R_t \in \mathbb{R}^{r\times n} \leftarrow P_t G_t$ ▷ Project gradient
  $S_t \in \mathbb{R}^{m\times n} \leftarrow G_t - P_t^{-1} R_t$ ▷ Residual gradient
  $M_{t-1}, V_{t-1} \leftarrow \text{LDUpd}(G_t, P_t, P_{t-1}, M_{t-1}, V_{t-1})$
  $N_t, M_t, V_t \leftarrow \text{AdamUpdate}(R_t, M_{t-1}, V_{t-1})$
  $\widetilde{S}_t \leftarrow \text{Sign}(S_t)$ ▷ Coordinate-wise
  $W_{t+1} \leftarrow W_t + \eta P_t^{-1} N_t + \eta \widetilde{S}_t$ ▷ Project back and step
end for

C.4.7 FRUGAL MM

FRUGAL MM keeps the projection and residual decomposition of FRUGAL, replacing Adam on the projected gradient with momentum Muon and signSGD on the residual with momentum-free Muon. The projected momentum is reset when the projector changes. The operator MuonNS_5 denotes five Newton–Schulz steps with Muon's matrix-shape scaling.

Algorithm 15 FRUGAL MM

Parameters: η (learning rate), μ (momentum coefficient)
$M_0 \in \mathbb{R}^{r\times n} \leftarrow 0$
for $t = 1, 2, \ldots, K$ **do**
  $G_t \in \mathbb{R}^{m\times n} \leftarrow \nabla_W f(W_t)$
  if $t = 1$ or $t \bmod T = 0$ **then**
    $P_t \in \mathbb{R}^{r\times m} \leftarrow \mathcal{N}(0, 1/r)$
    $M_{t-1} \leftarrow 0$ ▷ Reset projected momentum
  else
    $P_t \leftarrow P_{t-1}$
  end if
  $R_t \in \mathbb{R}^{r\times n} \leftarrow P_t G_t$ ▷ Project gradient
  $S_t \in \mathbb{R}^{m\times n} \leftarrow G_t - P_t^{-1} R_t$ ▷ Residual gradient
  $M_t \leftarrow \mu M_{t-1} + (1-\mu) R_t$
  $N_t \leftarrow -\mathrm{MuonNS}_5(R_t + \mu M_t)$ ▷ Stateful Muon
  $\widetilde{S}_t \leftarrow -\mathrm{MuonNS}_5(S_t)$ ▷ Stateless Muon
  $W_{t+1} \leftarrow W_t + \eta P_t^{-1} N_t + \eta \widetilde{S}_t$ ▷ Project back and step
end for

C.4.8 COSMOS (LIU ET AL., 2025B)

Algorithm 16 COSMOS

Parameters: M, V – moments; P, S – basis and spectrum.
η – learning rate; β_1, β_2 – EMA decays; γ – residual scaling.

$M_0 \in \mathbb{R}^{m\times n} \leftarrow 0, \quad V_0 \in \mathbb{R}^{m\times r} \leftarrow 0$
for $t = 0, 1, 2, \dots$ **do**
  $G_t \in \mathbb{R}^{m\times n} \leftarrow \nabla_W f(W_t)$
  SOAP in the leading eigenspace
  $M_t \leftarrow \beta_1 M_{t-1} + (1-\beta_1) G_t$
  $L_t \in \mathbb{R}^{n\times n} \leftarrow \beta_2 P_{t-1} S_{t-1} P_{t-1}^\top + (1-\beta_2) G_t^\top G_t$ ▷ Left preconditioner EMA
  $P_t \in \mathbb{R}^{n\times r} \leftarrow \mathrm{QR}(L_t P_{t-1})$
  $S_t \in \mathbb{R}^{r\times r} \leftarrow P_t^\top L_t P_t$ ▷ Project preconditioner
  $V_t \leftarrow \beta_2 V_{t-1} + (1-\beta_2)(G_t P_t) \odot (G_t P_t)$ ▷ Projected second moment
  $A_t \leftarrow \left(\frac{M_t P_t / (1-\beta_1^t)}{\sqrt{V_t/(1-\beta_2^t)}} \right) P_t^\top$
  Muon in the remaining subspace
  $R_t \leftarrow M_t - M_t P_t P_t^\top$
  $B_t \leftarrow \mathrm{NORM}(\mathrm{NS5}\,(R_t/\|R_t\|_F))$
  $\widetilde{G}_t \leftarrow A_t + \gamma B_t \sqrt{m}$
  $W_{t+1} \leftarrow W_t - \eta\, \mathrm{NORM}(\widetilde{G}_t)\sqrt{m}$
end for

COSMOS further develops the space-splitting idea of Fira (Chen et al., 2025) and FRUGAL (Zmushko et al., 2025), but substitutes more expressive optimizers for both components. The leading low-rank subspace is updated with a one-sided variant of SOAP (Vyas et al., 2025), while the residual is handled by Muon (Jordan et al., 2024), replacing the simpler signSGD (Bernstein et al., 2018) used in FRUGAL.

To combine the two components, each is normalized as $\mathrm{NORM}(X) = X/\|X\|_F$ before summation. This serves two purposes: it ensures numerical stability of the Newton–Schulz iteration used inside Muon, which converges only when the spectral norm of the input is below a certain threshold (typically $\sqrt{3}$), a condition guaranteed since $\|A\|_2 \le \|A\|_F$, and it prevents scale mismatch between the two updates, which may otherwise operate on incompatible scales and cause one to dominate the other.

C.4.9 SUMO (REFAEL ET AL., 2025)

Algorithm 17 SUMO

Parameters: M – first moment; r – rank; η – learning rate; β – EMA decay.
γ – residual scaling; λ – weight decay; τ – clipping ratio.

$M_0 \in \mathbb{R}^{m\times n} \leftarrow 0$
for $t = 0, 1, 2, \ldots$ **do**
  $G_t \in \mathbb{R}^{m\times n} \leftarrow \nabla_W f(W_t)$
  if $t \bmod T = 0$ **then**
    $P_t \in \mathbb{R}^{m\times r} \leftarrow \text{TR_SVD}(G_t, r)$
  else
    $P_t \leftarrow P_{t-1}$
  end if
  $M_t \leftarrow \beta M_{t-1} + (1-\beta)G_t$
  $M_t^{\text{proj}} \in \mathbb{R}^{r\times n} \leftarrow Q_t^\top M_t$
  Exact orthogonalization in subspace
  $O_t \in \mathbb{R}^{r\times n} \leftarrow \text{SVD_Ortho}(M_t^{\text{proj}})$
  $\hat{G}_t \in \mathbb{R}^{m\times n} \leftarrow P_t\, O_t$
  Handle residual
  $R_t \in \mathbb{R}^{m\times n} \leftarrow M_t - Q_t Q_t^\top M_t$
  $B_t \leftarrow \text{NS5}\big(R_t/\|R_t\|_F\big)$
  $B_t \leftarrow \text{NORM}(B_t)$
  Update weights in original space
  $\widetilde{G}_t \leftarrow \hat{G}_t + \gamma\, B_t\sqrt{m}$
  $U_t \leftarrow \text{NORM}(\widetilde{G}_t)$
  $W_{t+1} \leftarrow W_t - \eta\, U_t\sqrt{m} - \eta\,\alpha\, W_t$
end for

SUMO is built around the following principle: perform exact orthogonalization of the first-order moment inside a low-dimensional subspace, while handling the orthogonal complement with a lightweight update rule. Projecting into a low-dimensional subspace is not merely a memory-saving device, it also makes operations that would be prohibitively expensive in full dimension computationally feasible. SUMO exploits this by computing an exact SVD of the projected first moment, rather than relying on Newton–Schulz iterations, whose approximation error degrades with ill-conditioning of the moment matrix. The orthogonalized result is then reconstructed back in the original weight space with controlled norm. The leading subspace itself is estimated via truncated randomized SVD, positioning SUMO between the full SVD of GaLore (Zhao et al., 2024) and the purely random projectors of APOLLO (Zhu et al., 2025) and FRUGAL (Zmushko et al., 2025). The residual outside the subspace is processed separately, and a norm-growth limiter is applied to maintain numerical stability and prevent scale mismatch between the subspace and residual updates, similarly to Fira (Chen et al., 2025).

The paper provides theoretical motivation for this design: the first-order moment tends to become low-rank during LLM training, and the Newton–Schulz orthogonalization error is shown to grow with the condition number of the moment matrix.

C.5 Other Methods

C.5.1 BAdam (Luo et al., 2024)

Algorithm 18 BAdam (Block-wise Adam)

Parameters: β_1, β_2 – EMA coeffs; α – lr; K – inner steps; D – # blocks.

$W^0 \in \mathbb{R}^d$, $t \leftarrow 0$ ▷ $d = \sum_{i=1}^{D} d_i$, $d_i = |\pi_i|$
for $t = 1, 2, \ldots$ **do**
  Generate block partition $\pi = \{\pi_1, \ldots, \pi_D\}$
  for $i = 1$ to D **do** ▷ One block-epoch
    $M_{\pi_i}^{t,0} \leftarrow 0,\ V_{\pi_i}^{t,0} \leftarrow 0,\ W_{\pi_i}^{t,0} \leftarrow W_{\pi_i}^{t}$
    for $k = 1$ to K **do** ▷ Inner Adam steps
      $G^{t,k} \leftarrow \nabla_{W_{\pi_i}} \mathcal{L}(W_{<i}^{t+1}, W_{\pi_i}^{t,k-1}, W_{>i}^{t})$
      $N^{t,k}, M^{t,k}, V^{t,k} \leftarrow \text{AdamUpdate}(G^{t,k}, M^{t-1,k}, V^{t-1,k})$
      $W_{\pi_i}^{t,k} \leftarrow W_{\pi_i}^{t,k-1} - \alpha\, N^{t,k}$
    end for
    $W_{\pi_i}^{t+1} \leftarrow W_{\pi_i}^{t,K}$
    `clear` G, M, V ▷ Free memory
  end for
  $t \leftarrow t + 1$
end for

BAdam applies block coordinate descent to parameter optimization. The parameter space is partitioned into D blocks $\{W_{\pi_i}\}_{i=1}^{D}$, and at each outer iteration the blocks are updated sequentially: block i is optimized for K inner Adam (Kingma & Ba, 2015) steps while all other blocks are held fixed, approximately solving

$$W_{\pi_i}^{t+1} = \arg\min_{W_{\pi_i}} \frac{1}{n} \sum_{j=1}^{n} \mathcal{L}_j\big(W_{<i}^{t+1},\, W_{\pi_i},\, W_{>i}^{t}\big)\,,$$

where $\mathcal{L}_j(\cdot)$ is the loss on the j-th training example, $W_{<i}^{t+1}$ denotes the blocks already updated in the current round, and $W_{>i}^{t}$ denotes the blocks not yet updated. Optimizer states are instantiated only for the currently active block and freed immediately after its update, so the peak live state at any step is a small fraction of that required by full Adam.

C.5.2 Adam-mini (Zhang et al., 2024)

Algorithm 19 Adam-mini

Parameters: η – lr; β_1, β_2 – EMA coeffs; Π – parameter partition.

$M_0 \in \mathbb{R}^{m \times n} \leftarrow 0$, $V_0^{(b)} \in \mathbb{R} \leftarrow 0$ for each $b \in \Pi$
for $t = 1, 2, \ldots$ **do**
  $G_t \leftarrow \nabla_W f(W_t)$
  $M_t \leftarrow \beta_1 M_{t-1} + (1 - \beta_1) G_t$ ▷ First moment
  for each block $b \in \Pi$ **do**
    $g_t^{(b)} \leftarrow \text{mean}((G_t^{(b)})^2)$ ▷ Block-wise second moment
    $V_t^{(b)} \leftarrow \beta_2 V_{t-1}^{(b)} + (1 - \beta_2)\, g_t^{(b)}$
    $\hat{V}_t^{(b)} \leftarrow V_t^{(b)} / (1 - \beta_2^t)$
  end for
  $\hat{M}_t \leftarrow M_t / (1 - \beta_1^t)$
  for each block $b \in \Pi$ **do**
    $W_t^{(b)} \leftarrow W_t^{(b)} - \eta\, \dfrac{\hat{M}_t^{(b)}}{\sqrt{\hat{V}_t^{(b)}} + \epsilon}$
  end for
end for

Adam-mini reduces optimizer memory by replacing Adam's(Kingma & Ba, 2015) element-wise second-moment accumulator with a single scalar per parameter block. The motivation comes from an analysis of the Hessian structure of Transformer models: parameters whose Hessian rows are similar can share the same adaptive learning rate without a loss in optimization quality. For Transformers, this structure aligns naturally with the architecture: each attention head and each FFN neuron forms a block with homogeneous curvature characteristics. Concretely, Adam-mini uses one block per embedding matrix, separate blocks for Q, K, and V projections within each attention head, and one block per neuron in the FFN up-projection.

The first moment is maintained at full resolution for all parameters, so the expressiveness of the momentum estimate is unchanged. Only the second moment is compressed, collapsing from mn scalars to one scalar per block. This yields a 45–50% reduction in optimizer memory compared to Adam.

C.5.3 SLIMADAM (KALRA ET AL., 2025)

Algorithm 20 SlimAdam

Parameters: η – lr; β_1, β_2 – EMA decays; Π – partition; $K^{(b)}$ – compression axis.

$M_0 \in \mathbb{R}^{m\times n} \leftarrow 0, \quad V_0^{(b)} \leftarrow 0$ for each $b \in \Pi$
for $t = 1, 2, \ldots$ **do**
  $G_t \leftarrow \nabla_W f(W_t)$
  $M_t \leftarrow \beta_1 M_{t-1} + (1-\beta_1) G_t$ ▷ Full first moment
  for each block $b \in \Pi$ **do**
    if $K^{(b)} = \emptyset$ **then** ▷ No compression
      $V_t^{(b)} \leftarrow \beta_2 V_{t-1}^{(b)} + (1-\beta_2)\, G_t^{(b)} \odot G_t^{(b)}$
    else ▷ Compress along $K^{(b)}$
      $\bar{G}_t^{(b)} \leftarrow \mathbb{E}_{K^{(b)}}[G_t^{(b)}]$
      $V_t^{(b)} \leftarrow \beta_2 V_{t-1}^{(b)} + (1-\beta_2)\, (\bar{G}_t^{(b)})^2$
    end if
    $\hat{V}_t^{(b)} \leftarrow V_t^{(b)} / (1-\beta_2^t)$
  end for
  $\hat{M}_t \leftarrow M_t / (1-\beta_1^t)$
  for each block $b \in \Pi$ **do**
    $W_t^{(b)} \leftarrow W_t^{(b)} - \eta \dfrac{\hat{M}_t^{(b)}}{\sqrt{\widehat{V}_t^{(b)}} + \epsilon}$ ▷ Broadcast $\widehat{V}_t^{(b)}$ if compressed
  end for
end for

SlimAdam asks when Adam's per-parameter second moments can be safely replaced by their mean along a given dimension. Rather than applying a uniform compression strategy across all layers, it uses a layer-specific Signal-to-Noise Ratio (SNR) analysis to determine which dimensions are compressible. For a second-moment matrix $V_t \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$, the SNR along dimension K is defined as

$$\mathrm{SNR}_K(V_t) = \mathbb{E}_{K'}\left[\frac{\mathbb{E}_K[V_t]^2}{\mathrm{Var}_K[V_t]}\right].$$

When $\mathrm{SNR}_K \gtrsim 1$, entries along K are tightly clustered around their mean and can be replaced by a single value per slice without significant information loss. When $\mathrm{SNR}_K \lesssim 1$, individual entries carry significant curvature information and should be retained. Based on this criterion, SlimAdam compresses each layer's second moment along the dimension with the highest SNR exceeding a threshold, or leaves it uncompressed if no dimension qualifies.

The first moment is maintained at full resolution throughout, keeping momentum estimates unaffected and yielding memory consumption comparable to Adam-mini (Zhang et al., 2024).

D Model Configurations and Training Hyperparameters

D.1 Model Configurations

Table 22 specifies the dense decoder-only models used in training and distributed communication measurements. All use RoPE, RMSNorm with $\epsilon = 10^{-5}$, SwiGLU, multi-head attention without grouped queries, and untied input and output embeddings. Linear layers have no bias and dropout is zero.

Table 22: Dense model configurations. L is the number of transformer blocks, d the hidden size, d_{ff} the SwiGLU intermediate size, H the number of attention heads, and PP the pipeline-parallel degree used in the distributed communication measurements.

Model	L	d	d_{ff}	H	Head dim.	PP
257M	12	1024	2816	8	128	1
500M	18	1280	3584	20	64	1
1B	5	3456	9216	27	128	1
2B	12	3456	9216	27	128	1
3B	19	3456	9216	27	128	1
3.5B	22	3456	9216	27	128	1
4B	26	3456	9216	27	128	1
4.5B	29	3456	9216	27	128	1
5B	32	3456	9216	27	128	1
5B-wide	14	5120	13824	40	128	1
9.9B	64	3456	9216	27	128	2

The 1B to 5B communication models form a depth-scaling series with fixed width. The 5B-wide model changes width at approximately the same parameter count as 5B. The 9.9B label denotes twice the depth of 5B, with 32 blocks per pipeline stage.

D.2 Dense Pretraining Protocol

The 257M and 500M training runs use FineWeb-Edu with the GPT-2 tokenizer, sequence length 1024, an effective batch of 128 sequences, model seed 0, and data seed 1337. Microbatch size, gradient accumulation, and data-parallel degree vary while preserving this effective batch. All reported token-budget comparisons use WSD with a cosine cooldown over the final 10% of steps to zero learning rate. The $1 \times C$, $2 \times C$, and $4 \times C$ budgets for 257M correspond to 39,250, 78,500, and 157,000 optimizer steps. The 500M $1 \times C$ and $2 \times C$ budgets use 75,457 and 150,914 steps. Selected 257M FP8-state runs also extend to $8 \times C$ at 314,000 steps. Warmup lasts 2,000 steps for 257M and 7,000 for 500M.

Tables 23 and 24 give the BF16-compute settings. All rows use $\beta_1 = 0.9$ and $\epsilon = 10^{-7}$ where applicable. The β_2 column describes Adam-family moments or the AdamW fallback for parameters outside Muon's matrix updates. Muon's matrix momentum is listed separately below.

For GaLore and FRUGAL, the retained fraction is $\rho = 0.25$ and the subspace is refreshed every 50 steps. GaLore uses SVD projection, while FRUGAL and FRUGAL MM select columns. APOLLO uses random projection, channel-wise scaling, $\rho = 0.25$, and a 50-step refresh interval. APOLLO-mini instead uses tensor-wise scaling. Muon uses Nesterov momentum 0.95 and five Newton-Schulz steps. SOAP refreshes its preconditioner every 50 steps with preconditioner decay 0.99. Shampoo uses a 100-step refresh interval and a maximum preconditioner dimension of 1024. AdEMAMix uses $\beta_3 = 0.9999$ and mixing coefficient 8, with both warmup controls spanning 157,000 steps on 257M and 150,914 steps on 500M. Fira uses $\alpha = 1$. SlimAdam uses the bundled layer map without a custom rules file.

Figure 19 shows the 257M learning-rate sweeps used to select peak rates before the longer training runs.

Table 23: Optimizer hyperparameters for 257M BF16 training. LR is the peak learning rate, WD weight decay, and Clip the gradient-norm threshold. Values are from the $4 \times C$ training runs.

Optimizer	LR	WD	β_2	Clip
AdamW	10^{-3}	0.1	0.99	1
Muon	10^{-3}	0.1	0.99	1
SOAP	10^{-3}	0.1	0.99	1
AdEMAMix	10^{-3}	0.1	0.999	0.5
Shampoo	10^{-2}	0.01	0.99	1
GaLore	10^{-3}	0.1	0.999	1
Fira	10^{-3}	0.1	0.999	1
FRUGAL	10^{-3}	0.1	0.999	1
FRUGAL MM	2×10^{-3}	0.1	0.99	1
APOLLO	2×10^{-3}	0.1	0.999	1
APOLLO-mini	2×10^{-3}	0.1	0.999	1
SlimAdam	5×10^{-4}	0.1	0.999	1
Adam-mini	10^{-3}	0.1	0.999	1

Table 24: Optimizer hyperparameters for 500M BF16 training. The FRUGAL MM settings follow its local launch configuration.

Optimizer	LR	WD	β_2	Clip
AdamW	10^{-3}	10^{-4}	0.99	1
Muon	10^{-3}	10^{-4}	0.99	1
SOAP	10^{-3}	10^{-4}	0.99	1
AdEMAMix	10^{-4}	10^{-4}	0.999	0.5
Shampoo	10^{-2}	0.01	0.99	1
GaLore	10^{-3}	0.1	0.999	1
FRUGAL	10^{-3}	0.1	0.999	1
FRUGAL MM	2×10^{-3}	0.1	0.99	1
APOLLO	2×10^{-3}	0.1	0.999	1
SlimAdam	5×10^{-4}	0.1	0.999	1

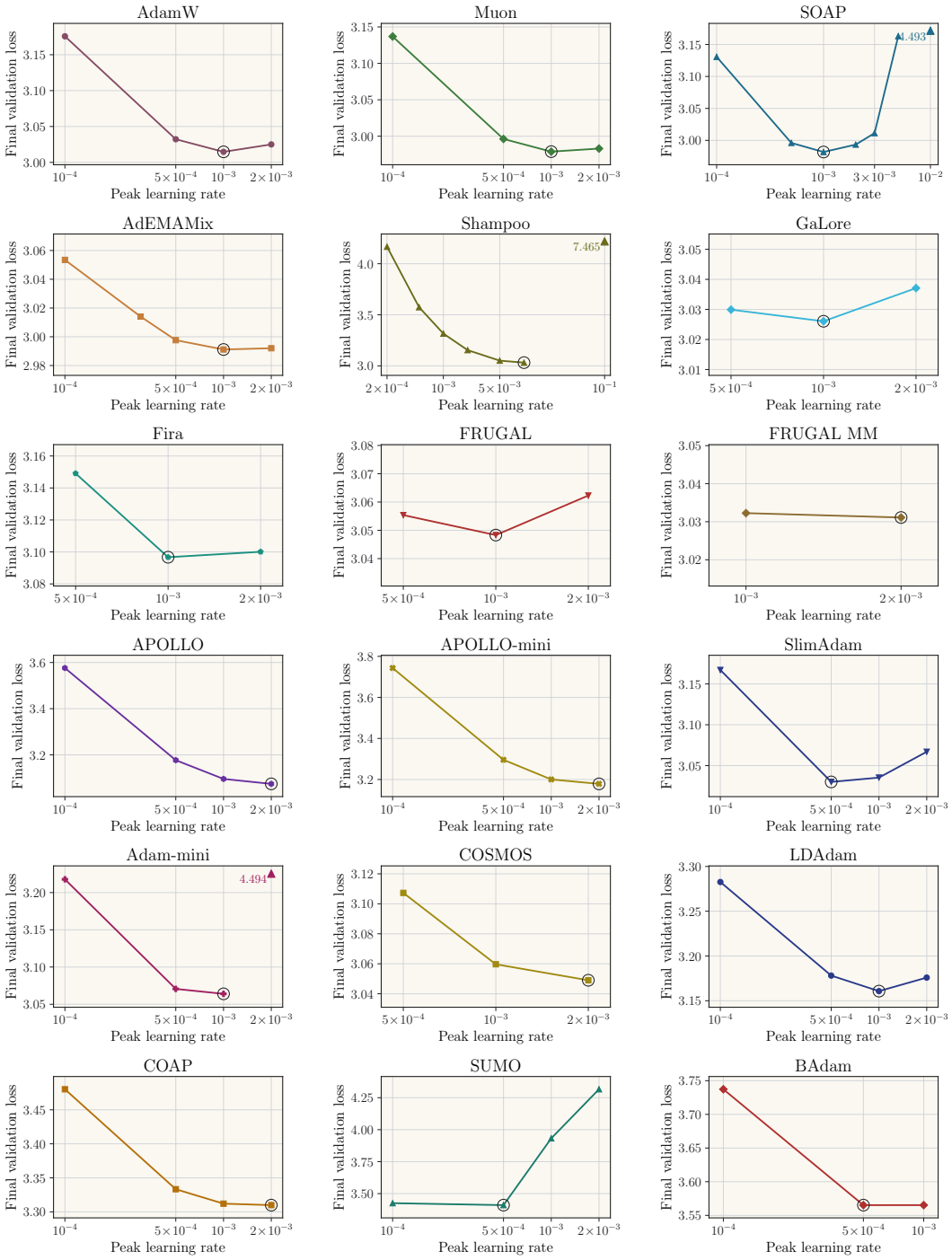

Figure 19: Learning-rate sensitivity on the 257M FineWeb-Edu model after $1 \times C$ tokens in BF16. Each panel shows final validation loss for one optimizer. Circles mark the lowest observed loss. Upward triangles with numeric labels denote values above the displayed vertical range.

D.3 MoE Pretraining Protocol

The MoE model of Section 4.5 is trained with Megatron-Core. It has 18 layers; the first is dense and each of the others holds 64 routed experts with top-8 sigmoid routing and one shared expert, for 1.03B total and 280M active parameters. The expert granularity, the single shared expert, and the dense first layer follow the MoE scaling study of Tian et al. (2025). Sigmoid top-8 routing and auxiliary-loss-free expert biases follow DeepSeek-AI (2024). Training uses FineWeb-Edu with the GPT-2 tokenizer, sequence length 2048, a global batch of 208 sequences, and 17,242 steps (7.34B tokens) under WSD with 173 warmup and 3,448 decay steps. All optimizers use peak learning rate 1.63×10^{-3}, $\beta_2 = 0.95$, and weight decay 0.1. FRUGAL selects columns with $\rho = 0.25$, and SlimAdam keeps separate second moments for the gate and up halves of each SwiGLU input projection. FP8 states follow the dense setup. `W8A8G8` uses Transformer Engine's block-scaled E4M3 recipe for all GEMM operands. Peak memory (Table 9) is measured separately on one 80 GB GPU with a freshly initialized model and 16 sequences per optimizer step, as the maximum allocated memory over 12 steps after 5 warmup steps.

E Distributed Optimizer-State Communication Measurements

We profile Muon and FRUGAL MM with BF16 and FP8 optimizer states on four GPUs (PP1/DP4), eight GPUs (PP1/DP8 and PP2/DP4), and two nodes (PP1/DP8). All runs use sequence length 1024, a per-device microbatch of one sequence, and one microbatch per data-parallel rank. Where uncertainty is shown, entries report the mean and sample standard deviation over three repeats of 50 timed steps after 10 warmup steps.

Optimizer time includes Newton-Schulz with input preparation, wire encoding, state all-gather, and residual optimizer work. Total state communication is encoding plus all-gather. The Non-opt. column is step time minus optimizer time. The Other column is optimizer time minus Newton-Schulz with preparation, encoding, and all-gather. Input preparation includes FP8 decoding when applicable.

E.1 Four GPUs: PP1/DP4

From 1B to 4.5B parameters at fixed width, Muon's total state-communication time grows from 5.2 to 32.3 ms with BF16 states and from 2.8 to 19.3 ms with FP8 states. This is a $1.7\times$ to $1.9\times$ reduction. FP8 encoding accounts for 0.4 to 2.6 ms of each total and offsets part of the payload saving. The reduction does not shorten the complete step in this sweep. At 4.5B, communication falls by 13.0 ms, but Newton-Schulz with preparation grows by 26.2 ms, and step time rises from 1069.1 to 1077.3 ms. Tables 25 and 26 give the full breakdown. The 5B results come from an earlier configuration and are separate from the fixed-width sweep.

Table 25: 4 GPUs, PP1/DP4: step-time breakdown (ms per step).

Model	Optimizer	States	Step	Non-opt.	Optim.	NS+prep.	Other
1B	Muon	BF16	202.8 ± 1.0	62.5 ± 0.9	140.3 ± 0.1	116.1 ± 0.1	19.1 ± 0.0
1B	Muon	FP8	219.0 ± 7.6	71.2 ± 7.7	147.9 ± 0.1	119.6 ± 0.1	25.5 ± 0.1
1B	FRUGAL MM	BF16	263.0 ± 0.3	62.3 ± 0.4	200.8 ± 0.2	131.5 ± 0.3	61.9 ± 0.1
1B	FRUGAL MM	FP8	272.2 ± 4.1	62.4 ± 0.5	209.8 ± 3.6	138.0 ± 4.1	61.1 ± 0.2
2B	Muon	BF16	455.5 ± 0.7	123.0 ± 0.6	332.5 ± 0.4	280.9 ± 0.4	39.1 ± 0.1
2B	Muon	FP8	463.4 ± 0.5	123.3 ± 0.5	340.1 ± 0.3	290.7 ± 0.1	42.1 ± 0.1
2B	FRUGAL MM	BF16	523.9 ± 2.1	124.1 ± 1.0	399.8 ± 1.4	266.1 ± 0.9	117.8 ± 0.3
2B	FRUGAL MM	FP8	522.6 ± 1.7	123.1 ± 0.8	399.5 ± 1.0	271.1 ± 0.9	115.6 ± 0.4
3B	Muon	BF16	708.7 ± 1.7	184.1 ± 1.2	524.6 ± 0.5	445.1 ± 0.2	59.1 ± 0.1
3B	Muon	FP8	716.7 ± 0.5	184.3 ± 0.1	532.4 ± 0.4	461.5 ± 0.5	58.8 ± 0.1
3B	FRUGAL MM	BF16	781.5 ± 1.3	183.8 ± 0.5	597.7 ± 1.0	401.6 ± 0.7	173.4 ± 0.1
3B	FRUGAL MM	FP8	779.9 ± 0.4	182.9 ± 0.1	597.0 ± 0.6	406.6 ± 0.6	170.4 ± 0.1
3.5B	Muon	BF16	817.2 ± 0.9	208.8 ± 0.5	608.4 ± 0.4	515.3 ± 0.4	67.6 ± 0.1
3.5B	Muon	FP8	823.0 ± 1.5	208.6 ± 1.0	614.4 ± 0.7	534.6 ± 1.4	65.7 ± 0.3
3.5B	FRUGAL MM	BF16	891.4 ± 1.2	208.5 ± 0.3	683.0 ± 1.3	459.4 ± 1.2	197.3 ± 0.2
3.5B	FRUGAL MM	FP8	891.8 ± 0.7	207.8 ± 0.4	683.9 ± 0.4	466.0 ± 0.3	194.4 ± 0.2
4B	Muon	BF16	958.2 ± 0.7	242.3 ± 0.7	716.0 ± 0.1	607.5 ± 0.2	78.9 ± 0.0
4B	Muon	FP8	965.8 ± 0.4	242.9 ± 0.2	723.0 ± 0.3	630.6 ± 1.1	75.1 ± 0.1
4B	FRUGAL MM	BF16	1041.4 ± 2.7	242.6 ± 0.8	798.8 ± 2.0	539.1 ± 0.9	228.8 ± 0.1
4B	FRUGAL MM	FP8	1036.9 ± 1.4	242.4 ± 0.9	794.6 ± 0.5	544.1 ± 1.0	225.1 ± 0.2
4.5B	Muon	BF16	1069.1 ± 0.4	268.9 ± 0.2	800.2 ± 0.5	680.1 ± 2.0	87.8 ± 0.3
4.5B	Muon	FP8	1077.3 ± 0.9	269.1 ± 0.3	808.2 ± 0.7	706.3 ± 1.3	82.5 ± 0.2
4.5B	FRUGAL MM	BF16	1148.8 ± 2.6	268.5 ± 1.1	880.2 ± 1.7	592.8 ± 0.6	252.5 ± 0.5
4.5B	FRUGAL MM	FP8	1150.7 ± 0.2	268.9 ± 0.5	881.9 ± 0.4	603.2 ± 2.1	248.9 ± 0.4
5B	Muon	BF16	1145.5 ± 1.7	264.3 ± 0.8	881.1 ± 1.2	746.0 ± 1.0	101.7 ± 0.1
5B	Muon	FP8	1140.2 ± 0.3	264.2 ± 0.6	876.0 ± 0.7	769.7 ± 0.5	83.8 ± 0.0
5B	FRUGAL MM	BF16	1210.5 ± 0.7	261.6 ± 0.4	948.9 ± 0.4	645.4 ± 0.6	268.4 ± 0.2
5B	FRUGAL MM	FP8	1208.3 ± 1.8	261.6 ± 0.2	946.7 ± 1.7	654.0 ± 1.6	263.8 ± 0.2

Table 26: 4 GPUs, PP1/DP4: optimizer-state communication (ms per step).

Model	Optimizer	States	Encode	All-gather	Total comm.
1B	Muon	BF16	0	5.2 ± 0.1	5.2 ± 0.1
1B	Muon	FP8	0.4 ± 0.0	2.4 ± 0.1	2.8 ± 0.1
1B	FRUGAL MM	BF16	0	7.3 ± 0.3	7.3 ± 0.3
1B	FRUGAL MM	FP8	0.4 ± 0.0	10.3 ± 3.0	10.6 ± 3.0
2B	Muon	BF16	0	12.5 ± 0.3	12.5 ± 0.3
2B	Muon	FP8	1.1 ± 0.0	6.2 ± 0.2	7.2 ± 0.2
2B	FRUGAL MM	BF16	0	15.9 ± 0.7	15.9 ± 0.7
2B	FRUGAL MM	FP8	0.8 ± 0.0	11.9 ± 0.3	12.7 ± 0.3
3B	Muon	BF16	0	20.3 ± 0.6	20.3 ± 0.6
3B	Muon	FP8	1.7 ± 0.0	10.4 ± 0.4	12.1 ± 0.4
3B	FRUGAL MM	BF16	0	22.8 ± 0.4	22.8 ± 0.4
3B	FRUGAL MM	FP8	1.3 ± 0.0	18.7 ± 0.1	20.0 ± 0.1
3.5B	Muon	BF16	0	25.6 ± 0.2	25.6 ± 0.2
3.5B	Muon	FP8	2.0 ± 0.0	12.2 ± 1.0	14.1 ± 1.0
3.5B	FRUGAL MM	BF16	0	26.3 ± 0.3	26.3 ± 0.3
3.5B	FRUGAL MM	FP8	1.6 ± 0.0	22.0 ± 1.0	23.6 ± 1.0
4B	Muon	BF16	0	29.6 ± 0.3	29.6 ± 0.3
4B	Muon	FP8	2.3 ± 0.0	14.9 ± 0.8	17.2 ± 0.8
4B	FRUGAL MM	BF16	0	30.9 ± 1.5	30.9 ± 1.5
4B	FRUGAL MM	FP8	1.7 ± 0.0	23.6 ± 0.6	25.4 ± 0.5
4.5B	Muon	BF16	0	32.3 ± 1.9	32.3 ± 1.9
4.5B	Muon	FP8	2.6 ± 0.0	16.8 ± 1.1	19.3 ± 1.0
4.5B	FRUGAL MM	BF16	0	34.9 ± 0.7	34.9 ± 0.7
4.5B	FRUGAL MM	FP8	2.0 ± 0.2	27.7 ± 2.1	29.8 ± 2.0
5B	Muon	BF16	0	33.4 ± 0.5	33.4 ± 0.5
5B	Muon	FP8	2.8 ± 0.0	19.6 ± 0.2	22.4 ± 0.2
5B	FRUGAL MM	BF16	0	35.1 ± 0.8	35.1 ± 0.8
5B	FRUGAL MM	FP8	2.0 ± 0.1	26.9 ± 0.4	28.9 ± 0.4

E.2 EIGHT GPUS: PP1/DP8 AND PP2/DP4

For Muon, FP8 states reduce total communication from 39.6 to 21.6 ms in PP1/DP8 and from 35.7 to 23.0 ms in PP2/DP4, while complete-step times change by less than 1%. The 5.0B-wide model takes 56.8 ms to communicate its BF16 states, compared with 39.6 ms for the similarly sized deep model. Parameter count alone therefore does not determine communication time, although these measurements do not separate the effects of matrix shape and bucketing. FRUGAL MM benefits less from FP8 states because it retains momentum for only a quarter of the columns and communicates BF16 gradients for the remainder. Its total communication falls from 38.5 to 32.7 ms in PP1/DP8 and from 42.6 to 35.2 ms in PP2/DP4 (Tables 27 and 28).

Table 27: 8 GPUs, PP1/DP8 and PP2/DP4: step-time breakdown (ms per step).

PP/DP	Model	Optimizer	States	Step	Non-opt.	Optim.	NS+prep.	Other
1/8	5B	Muon	BF16	1145.0 ± 2.6	266.4 ± 3.2	878.7 ± 0.6	748.5 ± 0.5	90.6 ± 0.2
1/8	5B	Muon	FP8	1137.0 ± 1.2	264.7 ± 0.9	872.3 ± 0.3	776.0 ± 0.7	74.7 ± 0.0
1/8	5B	FRUGAL MM	BF16	1226.4 ± 1.3	270.1 ± 1.2	956.3 ± 1.7	649.5 ± 1.7	268.3 ± 0.3
1/8	5B	FRUGAL MM	FP8	1222.3 ± 4.0	271.0 ± 2.5	951.4 ± 1.7	655.8 ± 1.2	262.9 ± 0.5
1/8	5.0B-wide	Muon	BF16	1331.0 ± 0.4	221.7 ± 0.3	1109.4 ± 0.1	960.9 ± 1.5	91.6 ± 0.1
1/8	5.0B-wide	Muon	FP8	1324.5 ± 0.7	222.3 ± 0.6	1102.2 ± 0.2	995.2 ± 0.6	75.5 ± 0.0
2/4	9.9B	Muon	BF16	1280.5 ± 2.2	401.8 ± 2.1	878.7 ± 0.5	747.0 ± 1.5	96.0 ± 0.2
2/4	9.9B	Muon	FP8	1274.3 ± 1.4	401.6 ± 1.5	872.7 ± 0.7	771.4 ± 1.5	78.3 ± 0.1
2/4	9.9B	FRUGAL MM	BF16	1326.4 ± 2.9	401.8 ± 1.5	924.6 ± 1.5	622.2 ± 1.1	259.8 ± 0.4
2/4	9.9B	FRUGAL MM	FP8	1325.4 ± 2.5	402.6 ± 0.5	922.8 ± 2.1	631.8 ± 1.6	255.9 ± 0.3

Table 28: 8 GPUs, PP1/DP8 and PP2/DP4: optimizer-state communication (ms per step).

PP/DP	Model	Optimizer	States	Encode	All-gather	Total comm.
1/8	5B	Muon	BF16	0	39.6 ± 0.0	39.6 ± 0.0
1/8	5B	Muon	FP8	2.4 ± 0.0	19.2 ± 0.4	21.6 ± 0.4
1/8	5B	FRUGAL MM	BF16	0	38.5 ± 2.1	38.5 ± 2.1
1/8	5B	FRUGAL MM	FP8	1.9 ± 0.0	30.8 ± 1.5	32.7 ± 1.5
1/8	5.0B-wide	Muon	BF16	0	56.8 ± 1.5	56.8 ± 1.5
1/8	5.0B-wide	Muon	FP8	1.3 ± 0.0	30.3 ± 0.8	31.6 ± 0.8
2/4	9.9B	Muon	BF16	0	35.7 ± 1.8	35.7 ± 1.8
2/4	9.9B	Muon	FP8	2.8 ± 0.0	20.2 ± 1.1	23.0 ± 1.1
2/4	9.9B	FRUGAL MM	BF16	0	42.6 ± 2.3	42.6 ± 2.3
2/4	9.9B	FRUGAL MM	FP8	2.0 ± 0.0	33.2 ± 2.5	35.2 ± 2.5

E.3 TWO NODES: PP1/DP8

The node-local and cross-node measurements come from the same two-node job. With node-local placement, Muon's total state communication falls from 39.4 to 22.1 ms with FP8 states, but Newton-Schulz with preparation rises from 748.6 to 774.0 ms. The complete step remains near 1.14 s. Across nodes, state communication falls from 3460.8 to 1963.3 ms, reducing optimizer time from 4263.2 to 2783.3 ms and complete-step time from 20813.8 to 19450.0 ms, or 6.6%. The cross-node step also spends about 16.6 s outside the optimizer, including unoptimized data-parallel gradient traffic. Tables 29 and 30 give the step-time and communication breakdowns. This cross-node configuration was not optimized for distributed training.

Table 29: Node-local and cross-node PP1/DP8: step-time breakdown (ms per step).

Placement	Optimizer	States	Step	Non-opt.	Optim.	NS+prep.	Other
Node-local	Muon	BF16	1137.2	265.3	871.9	748.6	83.9
Node-local	Muon	FP8	1141.6	264.2	877.5	774.0	81.3
Node-local	FRUGAL MM	BF16	1218.9	263.5	955.5	650.7	268.5
Node-local	FRUGAL MM	FP8	1213.1	263.3	949.8	655.8	262.8
Cross-node	Muon	BF16	20813.8	16550.6	4263.2	719.6	82.8
Cross-node	Muon	FP8	19450.0	16666.6	2783.3	741.9	78.2
Cross-node	FRUGAL MM	BF16	21677.6	16649.9	5027.7	649.0	271.1
Cross-node	FRUGAL MM	FP8	20906.1	16631.3	4274.8	657.0	264.4

Table 30: Node-local and cross-node PP1/DP8: optimizer-state communication (ms per step).

Placement	Optimizer	States	Encode	All-gather	Total comm.
Node-local	Muon	BF16	0	39.4	39.4
Node-local	Muon	FP8	2.4	19.7	22.1
Node-local	FRUGAL MM	BF16	0	36.2	36.2
Node-local	FRUGAL MM	FP8	1.8	29.4	31.2
Cross-node	Muon	BF16	0	3460.8	3460.8
Cross-node	Muon	FP8	2.3	1960.9	1963.3
Cross-node	FRUGAL MM	BF16	0	4107.6	4107.6
Cross-node	FRUGAL MM	FP8	1.8	3351.5	3353.3