MoARa: Module-Aware Rank Allocation and Structure-Preserving Decomposition for Low-Rank LLM Pre-training

Keunyoung Kim
Seoul National University
keunyoung.kim@snu.ac.kr

Nojun Kwak
Seoul National University
nojunk@snu.ac.kr

Abstract

Low-rank gradient projection reduces the optimizer-state memory cost of large language model (LLM) pretraining, but the steps and wall-clock time needed to reach a target quality remain a meaningful axis for improvement. We attribute this to two design choices in existing methods: the projection-rank budget is allocated uniformly across Transformer modules with heterogeneous projection sensitivity, and projecting a raw gradient attenuates its magnitude and direction jointly. We propose **MoARa**, which combines a static profiling-based *module-aware projection-rank allocation* with a *block-wise magnitude–direction decomposition*; the default block size is set in the neighborhood of the attention head dimension. Across five Transformer architectures spanning Llama, Qwen, and DeepSeek at 300M to 7B scales, GaLore with MoARa reaches standard GaLore's final perplexity in 37% fewer steps and 34% less wall-clock time on Llama 2 7B, with only 0.2% peak reserved memory overhead under standard graph compilation. Across the six low-rank pretraining methods we evaluate, module-aware rank allocation alone delivers directionally consistent step reductions on all six. On compatible hosts, the two-component design reaches up to 41.7% step reduction and 37.1% wall-clock reduction.

1 Introduction

Recent advances in large language models (LLMs) have been driven by scaling model size, data, and training compute, with memory overhead emerging as a major systems bottleneck. Under Adam (Kingma and Ba, 2015)-style full-parameter training, optimizer states alone can consume memory comparable to or exceeding model weights, since maintaining 1st- and 2nd-moment statistics requires two additional tensors of the same size as each parameter. At the billion-parameter scale, this overhead substantially raises hardware requirements and limits practical scalability.

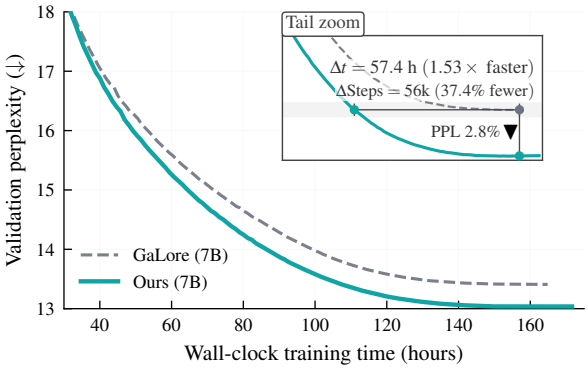

Figure 1: **Accelerated pre-training convergence at the 7B scale.** Validation PPL versus wall-clock time for GaLore with MoARa and standard GaLore on Llama 2 7B; the inset (*Tail zoom*) marks the convergence-time and final-perplexity gains highlighted in the figure.

A growing line of work addresses this bottleneck by projecting Adam's optimizer states into a low-dimensional subspace, retaining full-parameter learning while substantially reducing optimizer-state memory. Representative methods include GaLore (Zhao et al., 2024), which periodically recomputes the projection via SVD, and several subsequent extensions that refine how this subspace is maintained (Liang et al., 2024; Robert et al., 2025; Rajabi et al., 2025; Zhang et al., 2025b; Chen et al., 2025). These methods now approach the validation perplexity of full-rank training at substantially lower memory cost, but the number of optimization steps and the wall-clock time required to reach a given quality target remains a meaningful axis along which low-rank pretraining can be made more practical.

We identify two design limitations that constrain the convergence speed of existing methods. *First*, the projection-rank budget is allocated uniformly across Transformer modules, implicitly treating the seven attention and MLP projections (`attn.q/k/v/o`

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 26098–26127
October 24-29, 2026 ©2026 Association for Computational Linguistics

and `mlp.up/gate/down`) as equally sensitive to projection. As we show, the alignment between each module's full gradient and its low-rank reconstruction is sharply heterogeneous, so a uniform allocation over-provisions some modules while leaving others as bottlenecks. *Second*, projecting a gradient into a low-rank subspace attenuates both its local magnitude and direction. As a result, the per-block scale information that helps the optimizer adapt is lost together with the directional information.

To address these two limitations, we propose **Module-Aware Rank Allocation and Structure-Preserving Decomposition for Low-Rank LLM Pretraining (MoARa)**, a memory-efficient pretraining framework with two complementary components. The first, *module-aware projection-rank allocation*, redistributes a fixed total projection-rank budget across Transformer module types. Based on a profiling alignment diagnostic, we consolidate this redistribution into a static allocation rule (§3.2). The second, *block-wise magnitude–direction decomposition*, partitions each gradient row into contiguous blocks and separates per-block scale from direction before projection. The block-local scale is retained as a separate optimizer state outside the low-rank projection, which is the specific structure preserved by our method (§3.1). We set the default block size in the neighborhood of the attention head dimension, which our analysis identifies as a balanced regime and our empirical sweep supports as a robust plateau (§3.3).

We evaluate MoARa across five Transformer architectures (Llama 2 (Touvron et al., 2023), Llama 3.2 (Grattafiori et al., 2024), Qwen2.5 (Yang et al., 2024), Qwen3 (Yang et al., 2025), DeepSeek-V2 (DeepSeek-AI et al., 2024); 300M to 7B), at sequence lengths 256 and 2048, and across six low-rank pretraining methods. On Llama 2 7B (Figure 1), MoARa reaches standard GaLore's final perplexity 57.4 hours earlier — 37% fewer steps and 34% less wall-clock time — with only 0.2% additional peak reserved memory under standard graph compilation. Our main contributions are:

- **A module-aware low-rank pretraining framework.** We introduce MoARa, which couples module-aware projection-rank allocation with structure-preserving block-wise magnitude–direction decomposition. Both components are motivated by testable claims about the information geometry of low-rank gradient projection. The attention-head dimension provides an architecture-informed reference for the default block size.
- **Accelerated convergence across scale, architectures, and sequence length.** On Llama 2, MoARa applied to GaLore yields up to $1.64\times$ step speedup from 350M to 7B; at 7B this corresponds to 37% fewer steps and approximately 57.4 hours saved. Comparable reductions extend to sequence length 2048 and to four other recent Transformer architectures, with $+0.2\%$ peak reserved memory at 7B under standard graph compilation (Table 2).
- **Component-specific cross-method transferability.** Module-aware rank allocation transfers as a directionally consistent improvement across all six low-rank pretraining methods we evaluate (GaLore (Zhao et al., 2024), Fira (Chen et al., 2025), Q-GaLore (Zhang et al., 2025b), SubTrack++ (Rajabi et al., 2025), LDAdam (Robert et al., 2025), OSD (Liang et al., 2024)), with 8.3%–28.3% step reductions on the five non-GaLore methods. In contrast, structure-preserving decomposition is host-dependent and shows limited benefit under Q-GaLore. When combined with projection rank allocation on compatible hosts, the framework reaches up to 41.7% step and 37.1% wall-clock reduction.

2 Related Work

Memory-efficient pretraining via low-rank gradient projection. Reducing optimizer-state memory under full-parameter updates is a central challenge in LLM pretraining. GaLore (Zhao et al., 2024) introduces gradient low-rank projection, periodically recomputing the projection via SVD. Subsequent methods primarily refine how the subspace is maintained: OSD (Liang et al., 2024) replaces periodic SVD with online PCA and provides the first convergence guarantee for arbitrary projection update rules; LDAdam (Robert et al., 2025) introduces a projection-aware update rule for optimizer states across changing subspaces with a generalized error-feedback mechanism; SubTrack++ (Rajabi et al., 2025) tracks the gradient subspace on the Grassmannian manifold with projection-aware moments; Q-GaLore (Zhang et al., 2025b) combines INT4-quantized projection matrices with layer-wise adaptive SVD frequency; and Fira (Chen et al., 2025) restores full-rank gradient updates while keeping low-rank optimizer state through norm-based scaling. Recent work further refines sub-

space selection through importance sampling and moment orthogonalization (Zhang et al., 2025a; Refael et al., 2025). Across this family, however, the projection-rank budget is distributed *uniformly* across Transformer modules, implicitly assuming that the seven attention and MLP projections share equal optimization sensitivity. Our work explicitly challenges this assumption and reallocates the budget module-wise via a baseline profiling diagnostic.

Memory efficiency along orthogonal axes. Memory-efficient pretraining has also been pursued along axes orthogonal to optimizer-side SVD projection. CoLA (Liu et al., 2025) restructures attention and MLP projections as low-rank autoencoders, which alters the model architecture itself and places it outside our fixed-architecture, fixed-budget comparison frame. APOLLO (Zhu et al., 2025) uses random-projection-based structured AdamW (Loshchilov and Hutter, 2019) scaling whose auxiliary state carries no per-module r-dimensional subspace, so it lies in a different design space from the methods compared in §4.

Gradient and weight decomposition. Decoupling magnitude from direction has a long history of stabilizing high-dimensional optimization. Weight Normalization (Salimans and Kingma, 2016) reparameterizes weights as $w = g \cdot v/\|v\|$ for improved gradient conditioning, and DoRA (Liu et al., 2024a) extends this decoupling to parameter-efficient fine-tuning; both operate on weights rather than gradients. On the optimizer side, Adafactor (Shazeer and Stern, 2018) factorizes second-moment statistics row/column-wise, and VLoRP (Wang et al., 2025) varies the granularity of low-rank gradient projection. MoARa is, to our knowledge, the first method to apply magnitude–direction decomposition to gradients in low-rank pretraining, where the goal is preserving directional information through the projection bottleneck.

Module-aware rank allocation in Transformers. A growing literature establishes that Transformer modules are structurally heterogeneous and benefit from differentiated low-rank treatment. In post-hoc compression, LoRAP (Li et al., 2024) and A^3 (Wong et al., 2026) report distinct low-rank characteristics between attention sub-layers and feed-forward components; in parameter-efficient fine-tuning, AdaLoRA (Zhang et al., 2023), ALoRA (Liu et al., 2024b), and ARA (Xv et al., 2025) adaptively allocate the rank budget across weight matrices based on importance scores. All of these methods target compression or fine-tuning of already-trained models, where importance can be measured on stable representations. From-scratch pretraining is fundamentally different, as rank allocation must account for Adam's evolving moment states.

To our knowledge, MoARa is the first to bring budget-preserving, module-aware rank allocation to from-scratch LLM pretraining. This is done via a static profiling phase to maintain optimizer history without destabilizing training. The same module-wise sensitivity pattern persists across the five Transformer architectures we evaluate, suggesting the asymmetry is structural rather than incidental. A summary of how MoARa positions relative to these methods is provided in Appendix A (Table 4).

3 Methodology

We present **MoARa**, a memory-efficient pretraining framework with two components: Block-wise Magnitude–Direction Decomposition and Projection Rank Reallocation via Subspace Alignment. Algorithm 1 summarizes the overall procedure.

3.1 Block-wise Magnitude and Direction Decomposition

To mitigate the memory cost of optimizer states, GaLore (Zhao et al., 2024) projects each gradient $G \in \mathbb{R}^{m\times n}$ into a low-rank subspace as $G_{\text{proj}} = P^\top G$ via a projection matrix $P \in \mathbb{R}^{m\times r}$ ($r \ll m, n$), and maintains optimizer states only for the projected gradient. Projecting raw gradients into this fixed subspace, however, forces magnitude and direction to be compressed jointly, attenuating both factors together and potentially suppressing low-energy but still informative updates. Decomposing the gradient before projection reduces this coupled attenuation, so that the projected branch focuses on directional preservation while a separate state tracks magnitude. A naive row-wise decomposition is overly coarse for Transformer weight matrices: applying a single scalar magnitude to an entire row imposes an artificial dependency among heterogeneous features and can distort the projected subspace. Element-wise normalization, at the other extreme, is unnecessarily expensive. Thus block size is introduced to control the granularity of the decomposition. We adopt an intermediate granularity and decompose the gradi-

Algorithm 1 Overall procedure of MoARa

Require: Model $\mathcal{M}$ with module types $\mathcal{T}$; base rank r_{base}; total budget R_{total}; profiling step set $\mathcal{S}_p = \{x, 2x, \ldots, N_p\}$; donor set $\mathcal{D}$, receiver set $\mathcal{R}$, transfer amount Δr; SVD interval T_{SVD}; block size B; learning rate η.

Ensure: Trained weights W.

Setup phase (executed once before training).

Phase 1. Profiling.
Run uniform-rank GaLore for N_p steps; at each $t \in \mathcal{S}_p$ and each weight W of type τ, log $G_t(W)$ and $G_{\text{recon},t}(W)$.

Phase 2. Diagnostic verification.
Compute S_τ for all $\tau \in \mathcal{T}$; validate that $\mathcal{R}$ contains low-S_τ modules and $\mathcal{D}$ matches donor ablations.

Phase 3. Budget-preserving reallocation.
Set $r_\tau \leftarrow r_{\text{base}} - \Delta r$ for $\tau \in \mathcal{D}$; $r_\tau \leftarrow r_{\text{base}} + \Delta r \cdot |\mathcal{D}|/|\mathcal{R}|$ for $\tau \in \mathcal{R}$; $r_\tau \leftarrow r_{\text{base}}$ otherwise.

Training phase (with fixed ranks from Phase 3).

for each training step s **do**
  for each target weight W of type τ **do**
    $G \leftarrow \nabla_W \mathcal{L}$
    $M, V \leftarrow \text{Decompose}(G, B)$
    if $s \mod T_{\text{SVD}} = 0$ **then**
      $P_\tau \leftarrow \text{TruncatedSVD}(V, r_\tau)$
    end if
    $\tilde{V} \leftarrow P_\tau^\top V$
    $\Delta\tilde{V} \leftarrow \text{Optimizer}_V(\tilde{V})$
    $\Delta V \leftarrow P_\tau \Delta\tilde{V}$
    $\Delta M \leftarrow \text{Optimizer}_M(M)$
    $W \leftarrow W - \eta\big(\text{Rep}_B(\Delta M) \odot \Delta V\big)$
  end for
end for
return W

ent into block-wise magnitude and direction.

Let $G \in \mathbb{R}^{m\times n}$. When $B \nmid n$, we right-zero-pad each row to the smallest column dimension $n_{\text{pad}} \geq n$ such that $B \mid n_{\text{pad}}$. Let $k = n_{\text{pad}}/B$ and denote the padded gradient by $G_{\text{pad}} \in \mathbb{R}^{m\times n_{\text{pad}}}$. We partition each padded row into k contiguous $1 \times B$ blocks:

$$g_{i,j} = (G_{\text{pad}})_{i,\, jB:(j+1)B} \in \mathbb{R}^B, \quad \text{where} \quad j = 0, \ldots, k-1. \tag{1}$$

For each block, we compute a scalar magnitude $m_{i,j} = \|g_{i,j}\|_2$ and a normalized direction block

$$(V_{\text{pad}})_{i,\, jB:(j+1)B} = \frac{(G_{\text{pad}})_{i,\, jB:(j+1)B}}{m_{i,j} + \epsilon}$$

for a small $\epsilon > 0$. Collecting all magnitudes yields $M \in \mathbb{R}^{m\times k}$, and $V_{\text{pad}} \in \mathbb{R}^{m\times n_{\text{pad}}}$. With $\text{Rep}_B(M)$ denoting the matrix obtained by repeating each $m_{i,j}$ over its B columns, the padded representation satisfies

$$G_{\text{pad}} \approx \text{Rep}_B(M) \odot V_{\text{pad}},$$

with exact equality recovered as $\epsilon \to 0$. The padded columns are discarded after decomposition and recomposition, so the resulting direction and update tensors retain the original shape.

Choice of B: head-dimension-informed default. We adopt B in the neighborhood of the attention head dimension d_{head} as an architecture-informed scale reference, rather than requiring exact head alignment. We provide a theoretical analysis of the underlying trade-off in §3.3, empirical validation in Appendix B.4, and specific values per model scale in Appendix D.2.

3.2 Projection Rank Reallocation via Subspace Alignment

Architectural intuition and alignment diagnostic. Our second design is motivated by a mismatch between uniform projection-rank allocation and the heterogeneous sensitivity of Transformer modules. If projection distortion is structurally non-uniform, a uniform rank allocation over-provisions some modules while leaving others as bottlenecks. We form an a priori expectation about which modules tolerate projection-rank reduction by reading the decoder block: `attn.q` and `attn.k` parameterize token-to-token compatibility scores (routing), `attn.v` carries representational content and `attn.o` integrates the multi-head outputs, and the MLP projections transform and propagate representational content, with `mlp.down` acting as the bottleneck that compresses the expanded MLP representation back into the model dimension. We therefore expected the routing pair `attn.q`/`attn.k` to be more tolerant to rank reduction than the content-carrying modules, and `mlp.down` to be most sensitive. To test this, we measure how well the projected subspace preserves each module's gradient direction via cosine similarity. With $G_{\text{recon}} = \text{Rep}_B(M) \odot (PP^\top V)$, the alignment at step t is

$$S(W, t) = \cos\big(G_t(W),\, G_{\text{recon},t}(W)\big), \tag{2}$$

and the module-wise alignment score $S_\tau = \mathbb{E}_{t,\, W\in\tau}[S(W, t)]$ averages over profiling steps and matrix instances of module type τ.

Donor and receiver selection from joint evidence. The observed S_τ ordering places `mlp.down` lowest by a wide margin, the routing pair `attn.q`/`attn.k` at moderate values, and the content/integration modules `attn.v`/`attn.o` highest, with `mlp.up`

and `mlp.gate` in the middle (Table 6). The budget-receiver role of `mlp.down` is strongly supported, but the cosine ranking alone would suggest `attn.v`/`attn.o` as budget-donors — in conflict with our architectural intuition. A single-module rank-reduction ablation (Appendix C.6) resolves the conflict: reducing the rank of `attn.q` or `attn.k` causes only a small increase in final perplexity, while reducing `attn.v`, `attn.o`, `mlp.up`, or `mlp.gate` causes a substantially larger one. The high cosine alignment of `attn.v`/`attn.o` therefore reflects a structural redundancy the projection preserves well, rather than actual robustness to capacity loss. We accordingly select the donor set $\mathcal{D} = \{\texttt{attn.q}, \texttt{attn.k}\}$ and the receiver set $\mathcal{R} = \{\texttt{mlp.down}\}$.

Static budget-preserving reallocation. Given a total projection-rank budget R_{total}, we assign $r_\tau \leftarrow r_{\text{base}} - \Delta r$ for $\tau \in \mathcal{D}$, $r_\tau \leftarrow r_{\text{base}} + \Delta r \cdot |\mathcal{D}|/|\mathcal{R}|$ for $\tau \in \mathcal{R}$, and the uniform baseline rank otherwise; this preserves $\sum_\tau r_\tau = R_{\text{total}}$ by construction. We adopt $\Delta r = r_{\text{base}}/2$ as a balanced default that transfers a nontrivial fraction of donor capacity while preserving half of each donor's baseline rank, without claiming this value is optimal. The resulting integer ranks are fixed throughout training. This static allocation is a deliberate design choice. In Adam-style low-rank training, changing ranks online changes the dimensionality of the optimizer states — newly added directions require fresh first- and second-moment initialization, while removed directions discard accumulated optimizer history, which can destabilize the training trajectory. Static allocation captures persistent module-wise sensitivity while maintaining optimizer-state continuity. Profiling runs once before target training and is excluded from the target-training wall-clock trajectories; a 1k-step run ($\sim$11 min on 350M in our settings) suffices to recover the full assignment (Appendix C.3).

3.3 Theoretical Foundations

Two information-geometric properties, both built on the cosine perspective of S_τ, motivate MoARa's design choices.

(i) Carrying magnitude outside the projection isolates block-wise cosine distortion to direction. Let $\widetilde{V} = P_V P_V^\top V$ and let $\widetilde{v}_{i,b}$ denote the corresponding block of $\widetilde{V}$. For every block for which both vectors are nonzero,

$$\cos(G_{i,b}, G_{\text{recon},i,b}) = \cos(v_{i,b}, \widetilde{v}_{i,b}).$$

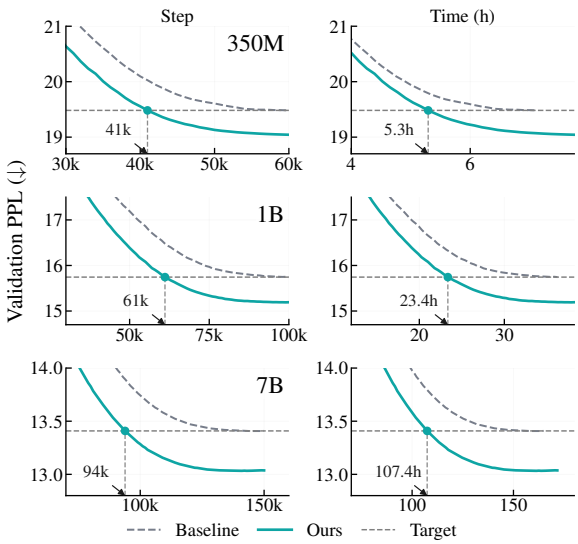

Figure 2: **Pre-training validation perplexity across model scales.** The left and right columns plot perplexity versus training steps and wall-clock time, respectively. Annotated points mark where our method reaches the baseline's final perplexity.

Thus, the scalar magnitude factor cancels from the block-wise cosine alignment, and any projection-induced alignment change is expressed through the direction branch.

(ii) Block size induces a statistical–coupling trade-off. Small B leaves block magnitudes dominated by per-coordinate noise. At the extreme $B = 1$, the direction signal becomes sign-valued and loses all coordinate-wise magnitude variation. Large B aggregates heterogeneous coordinates. In this regime, the unit-norm direction $v_{i,b} = G_{i,b}/m_{i,b}$ attenuates low-magnitude coordinates by the dominant ones within the same block. Setting B in the neighborhood of d_{head} provides an architecture-informed operating point in the balanced regime between the two extremes.

The two observations above provide an information-geometric rationale for MoARa's design, rather than a formal convergence guarantee for MoARa-augmented Adam. Full derivations are given in Appendix B.3.

4 Experiments

4.1 Experimental Setup

Models and Architectures. For the main scaling study, we adopt the Llama 2 (Touvron et al., 2023) architecture at 350M, 1B, and 7B scales. To evaluate architectural generalization, we further consider Llama 3.2 (300M) (Grattafiori et al., 2024),

Qwen2.5 (350M) (Yang et al., 2024), Qwen3 (350M) (Yang et al., 2025), and the dense variant of DeepSeek-V2 (350M) (DeepSeek-AI et al., 2024), scaled to be comparable to Llama 2 350M.

Baselines. Our primary comparisons are within the low-rank pretraining family, where GaLore serves as our main host. To assess transferability beyond GaLore, we additionally evaluate MoARa applied to five other low-rank methods in our cross-method ablation: Fira (Chen et al., 2025), Q-GaLore (Zhang et al., 2025b), SubTrack++ (Rajabi et al., 2025), LDAdam (Robert et al., 2025), and OSD (Liang et al., 2024).

Projection rank allocation protocol. For the Llama 2 scaling study, we profile the 350M configuration and reuse the resulting donor/receiver assignment at 1B and 7B, while the numerical ranks follow each scale's r_{base}. For Llama 3.2, Qwen2.5, and Qwen3, we fix Q and K as donors and Down as the receiver. For our custom dense DeepSeek-V2 variant, we use Q(b) and KV(b) as donors and Down as the receiver. These mappings are fixed before target training without target-architecture profiling or performance-based rank tuning.

Training details. All models are pretrained from scratch on the English C4 (Raffel et al., 2020) dataset. To ensure fair comparison, we control all training hyperparameters; detailed configurations are provided in Appendix D. Unless otherwise stated, all methods use the same total projection rank budget.

4.2 MoARa on GaLore across Scale, Architecture, and Sequence Length

We evaluate MoARa applied to GaLore along three axes: model scale, architecture family, and pretraining sequence length.

Pretraining efficiency at scale. Figure 2 reports validation perplexity trajectories with respect to both training steps and wall-clock time at 350M, 1B, and 7B scales on Llama 2. Across all three scales, GaLore with MoARa converges faster than standard GaLore in both axes. On Llama 2 7B (Figure 1), GaLore requires 150k steps to reach its final perplexity while GaLore with MoARa reaches the same target at 94k steps, reducing the required steps by 37% and the wall-clock time by 34%. These wall-clock values report target-training time only. Including the one-time 1k-step 350M profiling cost (∼11 min), the non-amortized time-to-target is 5.47 h at 350M versus 7.13 h for GaLore, while the wall-clock reduction at 7B remains approximately 34.7%. When the same assignment is reused across K target runs, the amortized time-to-target is T_{train} + $T_{profile}$ / K (Appendix C.3). As a secondary fixed-budget result, when both methods are trained for 150k steps, GaLore with MoARa continues to improve after the standard GaLore curve plateaus and achieves a further 2.8% gain in final perplexity. This acceleration is consistent across other scales and architectures, yielding up to $1.64\times$ step speedup from 350M to 7B in Llama 2 (Figure 7 in Appendix E). At 7B, the reduction in steps-to-target from 150k to 94k produces the net wall-clock saving despite a 4.19% per-step overhead.

Table 1: **Llama 2 1B benchmark results.** We report means over three seeds. Δ denotes relative change. "Win" counts the number of seeds for which our method outperforms GaLore. Full results are in Appendix F.

Task	Base	Ours	Δ	Win
Language modeling ↓				
C4 ppl	86.88	82.62	-4.90%	3/3
WikiText2 ppl	64.71	60.27	-6.86%	2/3
Commonsense / reasoning ↑				
HellaSwag	.386	.398	$+3.11\%$	3/3
WinoGrande	.513	.531	$+3.51\%$	3/3
PIQA	.677	.682	$+0.74\%$	2/3
BoolQ	.546	.558	$+2.20\%$	2/3
ARC-E	.425	.433	$+1.88\%$	2/3
ARC-C	.252	.248	-1.59%	1/3

Generalization across recent architectures. To verify that MoARa is not specific to one model family, we evaluate it on several recent Transformer architectures at the 300–350M scale, all of which depart from the standard multi-head attention used in Llama 2. Llama 3.2 and the Qwen family adopt Grouped-Query Attention, while DeepSeek-V2 uses Multi-head Latent Attention; both alter the dimensional ratios and parameter distributions of the `attn` matrices. Despite these differences, the Llama-2-derived allocation rule transfers through the predefined structural mapping described in §4.1, without target-architecture profiling or performance-based rank tuning, and achieves substantial reductions in steps-to-target across all tested architectures, including approximately 43% on Llama 3.2, 33% on the Qwen models, and 28% on DeepSeek-V2 (Appendix E.2), with proportional wall-clock reductions.

Table 2: **Component-wise memory cost (eager mode, MiB).** Treatment columns show relative change versus baseline. Under standard graph compilation at 7B, the max allocated/reserved overheads reduce to negligible levels. Full breakdown in Appendix G.

Scale	Metric	Base	+Decomp	+Rank	Full
350M	Max alloc.	47258	+0.05%	+0.00%	+0.12%
	Max reserv.	48154	+0.16%	-0.05%	+0.12%
	Opt. state	539	+13.4%	+7.5%	+20.9%
1B	Max alloc.	51611	+0.00%	+0.00%	+0.00%
	Max reserv.	54988	+0.29%	+0.26%	+0.57%
	Opt. state	1652	+8.7%	+9.7%	+18.4%
7B	Max alloc.	60899	+6.39%	+6.39%	+6.39%
	Max reserv.	62122	+7.43%	+7.22%	+7.43%
	Opt. state	7177	+5.4%	+12.0%	+17.4%

Robustness at sequence length 2048. The above scaling and architecture results follow the GaLore-standard sequence length of 256. To test whether the efficiency gains persist beyond this short-context setting, we re-evaluate GaLore with MoARa against standard GaLore at sequence length 2048 on Llama 2 350M. GaLore with MoARa reaches standard GaLore's final validation perplexity in 35.0% fewer steps and 33.3% less wall-clock time, and improves final perplexity by 2.65% (Appendix E.3).

Stability of the default configuration. The default configuration is supported by complementary diagnostics and sensitivity checks. The cosine-alignment analysis consistently identifies `mlp.down` as the receiver, while the single-module rank-reduction ablation identifies Q/K as the most robust donors. The 1k-step profile recovers the same donor/receiver assignment as the longer profiles, and this assignment remains unchanged across profiling strides of 50–400 steps. For decomposition, $B \in \{32, 64, 128\}$ forms a near-optimal plateau with little variation across the three settings. We use $\Delta r = r_{\text{base}}/2$ as a conservative budget-preserving default, while $T_{\text{SVD}} = 200$ follows standard GaLore and is not tuned for MoARa.

Robustness at the 1B scale and zero-shot downstream evaluation. As a limited sanity check, we assess whether the faster pretraining trajectory causes systematic downstream degradation. We train Llama 2 1B with three random seeds for both standard GaLore and GaLore with MoARa and evaluate the 100k-step checkpoints on a suite of zero-shot downstream benchmarks (Zellers et al., 2019; Sakaguchi et al., 2020; Bisk et al., 2020; Clark et al., 2019, 2018; Lin et al., 2022). Table 1 reports the results over three seeds. MoARa matches or improves upon standard GaLore on the majority of tasks across seeds on several primary metrics. The full benchmark set is reported in Appendix F.

Memory overhead in practice. These gains are achieved without materially compromising the memory advantage of low-rank pretraining. Table 2 summarizes the component-wise memory cost in eager mode, with the full eager- and compiled-mode breakdown reported in Appendix G. All wall-clock experiments use `torch.compile`. Under this compiled setting at the 7B scale, MoARa increases peak allocated memory by only $+0.04$ MiB and peak reserved memory by 0.2% relative to GaLore. Persistent optimizer-state memory is a separate, compilation-independent metric. At 7B, it increases by 1,250 MiB ($+17.4\%$), with the contributions from block-wise decomposition and module-aware rank allocation combining approximately additively. Peak allocated and reserved memory are dominated by model parameters and activations, so the eager-mode peak increase at 7B largely disappears under standard graph compilation.

4.3 Component-wise Cross-Method Transferability

To examine whether MoARa's two components are tied to GaLore's specific projection mechanism, we apply each component independently to five additional low-rank pretraining methods spanning distinct mechanism classes: Fira (Chen et al., 2025), Q-GaLore (Zhang et al., 2025b), SubTrack++ (Rajabi et al., 2025), LDAdam (Robert et al., 2025), and OSD (Liang et al., 2024). Combined with GaLore, this yields a 6×4 ablation at the 350M scale on C4. We report step reduction and wall-clock reduction to reach the host's vanilla final perplexity (Table 3). In summary, rank allocation improves all six hosts, whereas the added benefit of decomposition is host-dependent and is limited under Q-GaLore.

Module-aware rank allocation transfers as a directionally consistent improvement. Across all six host methods, rank allocation alone reaches the host's vanilla final PPL with strictly fewer steps and less wall-clock time. On the five non-GaLore hosts, step reduction ranges from 8.3% (Fira) to 28.3% (SubTrack++), and wall-clock reduction

Table 3: **Cross-method ablation at Llama 2 350M, 60k steps.** Each cell reports final PPL and step-to-/wall-clock-to-target. Target is each host's vanilla final perplexity; "—" indicates that the target was not reached within the 60k-step budget.

Condition	GaLore	Fira	Q-GaLore	SubTrack++	OSD	LDAdam
Vanilla	19.48 60k (−) 7.13h (−)	16.85 60k (−) 7.10h (−)	20.64 60k (−) 8.27h (−)	16.19 60k (−) 6.28h (−)	21.66 60k (−) 8.78h (−)	17.60 60k (−) 9.33h (−)
+ Rank	19.27 47k (↓22.0%) 5.58h (↓21.7%)	16.79 55k (↓8.3%) 6.60h (↓7.0%)	**20.45** **45k (↓25.0%)** **6.19h (↓25.2%)**	**15.57** **43k (↓28.3%)** **4.50h (↓28.4%)**	21.27 44k (↓26.7%) 6.97h (↓20.6%)	**17.19** **44k (↓26.7%)** **6.99h (↓25.1%)**
+ Decom	19.45 54k (↓9.6%) 7.06h (↓1.0%)	16.75 53k (↓11.7%) 6.79h (↓4.4%)	20.81 — —	28.29 — —	20.76 37k (↓38.3%) 5.41h (↓38.4%)	24.01 — —
+ Full (MoARa)	**19.04** **41k (↓31.7%)** **5.29h (↓25.8%)**	**16.70** **51k (↓15.0%)** **6.65h (↓6.3%)**	20.46 45k (↓25.0%) 6.26h (↓24.3%)	19.17 — —	**20.42** **35k (↓41.7%)** **5.52h (↓37.1%)**	20.24 — —

ranges from 7.0% to 28.4%; the two reductions remain closely matched in each case, since rank allocation only redistributes the projection budget rather than enlarging it. The directional consistency across six independent host mechanisms suggests that the module-wise differential in projection rank sensitivity is shared across the low-rank pretraining family rather than specific to GaLore's update rule.

The added benefit of block-wise decomposition is host-dependent. Three patterns emerge from Table 3: on hosts with slowly or discretely evolving projections (GaLore, Fira, OSD), the full configuration improves over vanilla on all three hosts and gives the best final PPL and steps-to-target. Its wall-clock gain additionally reflects the per-step cost of decomposition. On OSD, the full configuration achieves 41.7% step and 37.1% wall-clock reduction. On Q-GaLore, decomposition alone produces a small increase in final PPL and the full configuration recovers the rank-allocation-only result. On SubTrack++ and LDAdam, whose projection bases change continuously during training, decomposition does not combine stably with the evolving subspace despite multiple remedies, and the full configuration does not reach the host's vanilla target within the 60k-step budget. We analyze the host-specific mechanisms underlying all three patterns in §5.

5 Analysis and Discussion

The cross-method ablation of §4.3 revealed a sharp asymmetry between MoARa's two components: rank allocation transfers consistently across all six hosts, while decomposition transfers with strong host-dependence. We discuss the mechanisms responsible for this pattern. As an additional sanity check on the allocation itself, MoARa's sensitivity-guided allocation also outperforms 20 random budget-preserving allocations (Appendix E.1).

Projection rank allocation operates on an architectural property. The module-wise alignment scores S_τ — measured during a separate baseline profiling run as the cosine similarity between the full gradient and its low-rank reconstruction — characterize how much meaningful signal each module's gradient retains inside its projection subspace, independent of how the optimizer subsequently chooses to update along that subspace. The consistent benefit across the six evaluated hosts therefore suggests that this module-wise asymmetry is not specific to GaLore's update rule.

Decomposition operates on a host-mechanism property, splitting hosts into three groups. The behavior observed in §4.3 separates hosts by how their projection bases evolve over training and by the precision constraints of their numerical representations.

Group A — slowly or discretely evolving projections (GaLore (Zhao et al., 2024), Fira (Chen et al., 2025), OSD (Liang et al., 2024)). Fira maintains an r-dimensional optimizer state and applies a full-rank gradient update via norm-based scaling. This scaling partially restores the column-norm information attenuated by projection. In contrast, our decomposition addresses this information loss through a different path by separating the magnitude pre-projection. The Fira result in Table 3 suggests that its norm-based scaling and our de-

composition are compatible in the evaluated setting. OSD updates its projection basis online via PCA on accumulated statistics. A plausible interpretation is that its basis evolves gradually enough for decomposition to remain effective in our evaluation.

Group B — continuously evolving projections (SubTrack++ (Rajabi et al., 2025), LDAdam (Robert et al., 2025)). Both methods update the projection basis continuously during training. SubTrack++ operates on the Grassmannian manifold, while LDAdam applies a per-step subspace correction. Under this continuously evolving projection regime, a plausible explanation is that the separately optimized magnitude state becomes harder to coordinate with direction updates induced by the changing projection basis. Under the remedies we tested, the full configuration does not reach the host's vanilla final perplexity within the 60k-step budget.

Group C — quantized host (Q-GaLore (Zhang et al., 2025b)). Q-GaLore combines INT4 projection matrices and INT8 weights with an 8-bit Adam optimizer. In our precision ablation, increasing only the magnitude-branch optimizer state from 8-bit to FP32 does not recover the benefit of decomposition (Appendix E.5). This suggests that optimizer-state precision alone does not explain the degradation. The other quantized components of Q-GaLore are therefore plausible compatibility constraints for decomposition.

Two-axis summary. The cross-method pattern admits a clean two-axis reading. First, rank allocation is architecture-driven. Across the six evaluated hosts, it consistently improves time-to-target, suggesting that the module-wise asymmetry is not specific to GaLore. Second, decomposition is *host-mechanism-driven*. Across the evaluated hosts, decomposition is most effective when the projection basis evolves slowly or discretely and when the numerical representation is sufficiently precise. We treat these as observed compatibility factors rather than necessary or sufficient conditions. These conditions clarify when each component is likely to transfer within the broader design space of low-rank pretraining methods.

Practical trade-offs. Relative to the corresponding vanilla low-rank baselines, MoARa adds a one-time profiling stage and a static module-wise rank configuration. During target training, projection ranks remain fixed, so no online rank search or optimizer-state resizing is required. The remaining cost comes from per-step decomposition, recomposition, and the separate magnitude branch. In the 7B GaLore comparison, these additions result in a 4.19% per-step overhead. Our method also adds persistent optimizer state, while its compiled peak-memory overhead remains small.

6 Conclusion

We presented **MoARa**, a low-rank LLM pretraining framework. It combines profiling-based rank allocation with block-wise magnitude–direction decomposition. Together, these components improve the use of a fixed projection-rank budget and retain per-block magnitude information. Evaluated across five Transformer architectures from 300M to 7B, MoARa reaches the GaLore baseline's final perplexity in 37% fewer steps and 34% less wall-clock time on Llama 2 7B. Across all six evaluated hosts, module-aware rank allocation consistently reduces steps-to-target. On compatible hosts, the full design of MoARa achieves up to 41.7% step and 37.1% wall-clock reduction. Our analysis indicates that projection rank allocation captures a persistent architecture-related pattern within the evaluated settings, while decomposition shows host-dependent compatibility associated with projection-basis evolution and numerical precision.

Limitations

While MoARa effectively mitigates key bottlenecks in low-rank LLM pretraining, we discuss boundaries of our work and identify directions for future research.

Absence of a stable dynamic rank-allocation mechanism. MoARa allocates ranks statically through a single profiling phase prior to training, and the question of whether a stable *dynamic* rank-allocation mechanism is achievable remains open. A dynamic schedule that, for instance, expands the capacity of receiver modules after warmup is conceptually appealing, but our empirical investigations show that expanding a module's projection rank mid-training requires initializing the first- and second-moment statistics of Adam for newly spanned orthogonal directions, and we observe that this derails the training trajectory. We hypothesize that model parameters rapidly co-adapt to the initial low-rank bottleneck, and abruptly altering this representational capacity perturbs the accumulated moment states. Designing a stable dynamic mechanism that resolves this momentum–subspace mismatch is left to future work; the static allocation studied here should therefore be understood as the simplest stable instance of the broader module-aware allocation principle rather than its only realization.

Cross-method decomposition compatibility for continuously-evolving projections. Our cross-method ablation (§4.3) shows that block-wise magnitude–direction decomposition does not stably combine with hosts that update their projection basis continuously during training (SubTrack++, LDAdam) under the variants we tested. Future work could focus on two key questions. First, what conditions allow a magnitude state to be co-transported consistently with continuously-evolving projections? Second, does this co-transport rule lead to a stable optimization trajectory?

Formal convergence rate guarantees. Our theoretical analysis (§3.3, Appendix B.3) characterizes the information-geometric properties of MoARa's design but does not establish a formal convergence rate guarantee for MoARa-augmented Adam. Such a result would require tracking the moment dynamics of the magnitude and direction branches jointly and is a natural direction for follow-up theoretical work.

Scope of optimizer evaluation. Our experiments focus on low-rank optimizer-state projection methods built on AdamW (Loshchilov and Hutter, 2019) -style optimization, which provide a controlled setting for isolating the effects of projection-rank allocation and structure-preserving decomposition. Evaluating specialized optimizers such as 8-bit AdamW or Muon is important, but would introduce optimizer-specific numerical behavior and tuning choices that could confound the main algorithmic comparison. We therefore do not claim that MoARa outperforms all memory-efficient optimizers. A controlled cross-family comparison would require matched memory budgets and separate tuning protocols, which we leave to future work.

Scope of generalization. Our experiments focus on decoder-only Transformers, and whether the same allocation transfers to encoder–decoder architectures remains unverified. We also do not evaluate transfer across different pre-training datasets. Extending the 2048-token study across all model scales or to substantially longer contexts was computationally prohibitive within our available budget, so our long-context evidence is limited to the Llama 2 350M setting.

Acknowledgments

This work was supported by IITP grants funded by the Government of the Republic of Korea (Ministry of Science and ICT) under Grant Nos. RS-2021-II211343 and RS-2025-25442338, and by the "Advanced GPU Utilization Support Program".

References

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. PIQA: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7432–7439.

Xi Chen, Kaituo Feng, Changsheng Li, Xunhao Lai, Xiangyu Yue, Ye Yuan, and Guoren Wang. 2025. Fira: Can we achieve full-rank training of LLMs under low-rank constraint? In *Advances in Neural Information Processing Systems*, volume 38.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*

(Long and Short Papers), pages 2924–2936. Association for Computational Linguistics.

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

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, and 138 others. 2024. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *Preprint*, arXiv:2405.04434.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The Llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Guangyan Li, Yongqiang Tang, and Wensheng Zhang. 2024. LoRAP: Transformer sub-layers deserve differentiated structured compression for large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 28657–28672. PMLR.

Kaizhao Liang, Bo Liu, Lizhang Chen, and Qiang Liu. 2024. Memory-efficient LLM training with online subspace descent. In *Advances in Neural Information Processing Systems*, volume 37.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252. Association for Computational Linguistics.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024a. DoRA: Weight-decomposed low-rank adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 32100–32121. PMLR.

Zequan Liu, Jiawen Lyn, Wei Zhu, Xing Tian, and Yvette Graham. 2024b. ALoRA: Allocating low-rank adaptation for fine-tuning large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 622–641. Association for Computational Linguistics.

Ziyue Liu, Ruijie Zhang, Zhengyang Wang, Mingsong Yan, Zi Yang, Paul D. Hovland, Bogdan Nicolae, Franck Cappello, Sui Tang, and Zheng Zhang. 2025. CoLA: Compute-efficient pre-training of LLMs via low-rank activation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 4627–4645. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *International Conference on Learning Representations*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Sahar Rajabi, Nayeema Nonta, and Sirisha Rambhatla. 2025. SubTrack++: Gradient subspace tracking for scalable LLM training. In *Advances in Neural Information Processing Systems*, volume 38.

Yehonathan Refael, Guy Smorodinsky, Tom Tirer, and Ofir Lindenbaum. 2025. SUMO: Subspace-aware moment-orthogonalization for accelerating memory-efficient LLM training. In *Advances in Neural Information Processing Systems*, volume 38.

Thomas Robert, Mher Safaryan, Ionut-Vlad Modoranu, and Dan Alistarh. 2025. LDAdam: Adaptive optimization from low-dimensional gradient statistics. In *International Conference on Learning Representations*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. WinoGrande: An adversarial winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740.

Tim Salimans and Diederik P. Kingma. 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, volume 29.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4596–4604. PMLR.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu,

Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Yezhen Wang, Zhouhao Yang, Brian K Chen, Fanyi Pu, Bo Li, Tianyu Gao, and Kenji Kawaguchi. 2025. Memory-efficient LLM training by various-grained low-rank projection of gradients. *Preprint*, arXiv:2505.01744.

Jeffrey T. H. Wong, Cheng Zhang, Xinye Cao, Pedro Gimenes, Christos-Savvas Bouganis, George Anthony Constantinides, Wayne Luk, and Yiren Zhao. 2026. A3 : an analytical low-rank approximation framework for attention. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR.

Lin Xv, Jingsheng Gao, Xian Gao, Ting Liu, and Yuzhuo Fu. 2025. ARA: Adaptive rank allocation for efficient large language model SVD compression. *Preprint*, arXiv:2510.19389.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 23 others. 2024. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

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

Haochen Zhang, Junze Yin, Guanchu Wang, Zirui Liu, Lin F. Yang, Tianyi Zhang, Anshumali Shrivastava, and Vladimir Braverman. 2025a. Breaking the frozen subspace: Importance sampling for low-rank optimization in LLM pretraining. In *Advances in Neural Information Processing Systems*, volume 38.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. AdaLoRA: Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*.

Zhenyu Zhang, Ajay Kumar Jaiswal, Lu Yin, Shiwei Liu, Jiawei Zhao, Yuandong Tian, and Zhangyang Wang. 2025b. Q-GaLore: Quantized GaLore with INT4 projection and layer-adaptive low-rank gradients. In *Conference on Parsimony and Learning*, volume 280 of *Proceedings of Machine Learning Research*, pages 1035–1050. PMLR.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. 2024. GaLore: Memory-efficient LLM training by gradient low-rank projection. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 61121–61143. PMLR.

Hanqing Zhu, Zhenyu Zhang, Wenyan Cong, Xi Liu, Sem Park, Vikas Chandra, Bo Long, David Z. Pan, Zhangyang Wang, and Jinwon Lee. 2025. APOLLO: SGD-like memory, AdamW-level performance. In *Proceedings of Machine Learning and Systems*, volume 7.

A Positioning of MoARa

Table 4 summarizes how MoARa positions relative to the closest concurrent methods along five axes: architecture preservation, optimizer-state memory regime, module-aware rank allocation, magnitude–direction decomposition, and primary design objective. The memory regime is classified as *low-rank* (compression to an r-dimensional gradient subspace), *SGD-level* (per-channel/per-tensor scalars via random projection rather than SVD), or *indirect* (memory reduced primarily through architectural changes). For axis context we also include DoRA (Liu et al., 2024a) and AdaLoRA (Zhang et al., 2023), two fine-tuning methods.

B Block-wise Decomposition: Method Details and Analysis

This section provides the implementation details, justification, and theoretical analysis underlying the block-wise magnitude–direction decomposition introduced in §3.1 of the main text.

B.1 Implementation of Block-wise Decomposition on Flattened Tensors

For implementation, we apply block-wise decomposition by preserving the first axis and flattening all remaining axes. This gives a unified rule for all target linear layers. In other words, we keep the row axis unchanged and apply blocking only along the column side.

For $G \in \mathbb{R}^{s_0 \times s_1 \times \cdots \times s_{r-1}}$, we define

$$G_{\text{flat}} = \text{reshape}(G, m, n), \quad m = s_0, \quad n = \prod_{\ell=1}^{r-1} s_\ell.$$

If $B \nmid n$, we right-zero-pad the flattened column dimension to

$$n_{\text{pad}} = B \left\lceil \frac{n}{B} \right\rceil$$

and denote the padded tensor by

$$G_{\text{pad}} = \text{PadRight}\left(G_{\text{flat}}, n_{\text{pad}} - n\right) \in \mathbb{R}^{m \times n_{\text{pad}}}.$$

The padding is used only to complete the final block.

For block size B, each padded row is partitioned into $k = n_{\text{pad}}/B$ contiguous blocks:

$$g_{i,b} = (G_{\text{pad}})_{i,\, bB:(b+1)B} \in \mathbb{R}^B, \;\; b = 0, \ldots, k-1.$$

Block magnitude and direction are defined as

$$m_{i,b} = \|g_{i,b}\|_2, \qquad v_{i,b} = \frac{g_{i,b}}{\|g_{i,b}\|_2 + \epsilon},$$

where $\epsilon > 0$ is a small constant for numerical stability.

In our implementation, ϵ appears only in the normalization denominator. At recomposition, we multiply magnitude and direction directly:

$$\hat{g}_{i,b} = m_{i,b} v_{i,b} = \frac{\|g_{i,b}\|_2}{\|g_{i,b}\|_2 + \epsilon}\, g_{i,b}.$$

Hence $\hat{g}_{i,b} \neq g_{i,b}$ for finite ϵ, and exact equality is recovered as $\epsilon \to 0$. We use $\epsilon = 10^{-12}$ in all experiments. Because the padded entries are zero, padding does not change the L^2 magnitude of the final partial block. After forming the direction tensor, we discard the padded columns and reshape the remaining entries back to the original gradient shape. During recomposition, the direction update is temporarily padded, combined block-wise with the magnitude update, and cropped back to the original n columns.

B.2 Module-wise Interpretation Across Transformer Layers

The same block rule is used on all target linear layers:

$$\mathcal{W}_{\text{target}} = \{W_q, W_k, W_v, W_o, W_{\text{up}}, W_{\text{gate}}, W_{\text{down}}\}.$$

For each layer ℓ, with $G^{(\ell)} = \nabla W^{(\ell)}$, we preserve the row axis and block along columns:

$$g_{i,b}^{(\ell)} = G_{i,\, bB:(b+1)B}^{(\ell)}.$$

Although the blocking rule is the same, its interpretation depends on the module. In decoder-only transformers, different column-side channel groups can play different roles. Because of this, the same coarse normalization range may not be equally suitable for all modules.

Under this view, (i) for $q/k/v$, blocking acts along the representation axis within each row; (ii) for $W_o \in \mathbb{R}^{d_{\text{hidden}} \times (H d_{\text{head}})}$, blocking acts over concatenated head-channel columns.

This module-wise view helps explain why coarse normalization can be especially harmful in transformer blocks with heterogeneous column-side structure.

B.3 Detailed Analysis: Magnitude-Factor Preservation and the Block Size Trade-off

This appendix provides the formal magnitude-factor preservation result referenced in §3.3 and the detailed small-B/large-B derivation underlying the block-size trade-off discussed there.

Table 4: **Positioning of MoARa relative to closest concurrent works.**

Method	Arch. preserved?	Opt.-state regime	Module-aware allocation?	Mag.–dir. decomp.?	Primary objective
GaLore	✓	low-rank	×	×	memory reduction
LDAdam	✓	low-rank	×	×	adaptive low-D stats
OSD	✓	low-rank	×	×	online subspace update
SubTrack++	✓	low-rank	×	×	Grassmannian tracking
Q-GaLore	✓	low-rank	×	×	quant. + memory
Fira	✓	low-rank	×	×	full-rank update under low-rank state
APOLLO	✓	SGD-level	×	×	extreme memory reduction
CoLA	×	indirect	×	×	architectural compression
DoRA	✓	(fine-tune)	×	✓(weight-side)	fine-tuning capacity
AdaLoRA	✓	(fine-tune)	✓(LoRA budget)	×	fine-tuning budget
MoARa (ours)	✓	low-rank	✓(rank budget)	✓(gradient-side)	convergence speed under a fixed projection-rank budget

B.3.1 Setup

For notational simplicity, the derivation below assumes $B \mid n$; the same definitions apply to the shorter final block when $B \nmid n$. As in §3.1, we partition the column index set $\{0, \ldots, n-1\}$ into $K = n/B$ contiguous blocks of size B, indexed by $b \in \{0, \ldots, K-1\}$. For each row i and block b, let $G_{i,b} \in \mathbb{R}^B$ denote the corresponding sub-vector of G. For the exact identities used in the analysis below, we consider the $\epsilon \to 0$ limit of the implementation described in Appendix B.1. For each nonzero block, we define

$$m_{i,b} = \|G_{i,b}\|_2, \qquad v_{i,b} = \frac{G_{i,b}}{m_{i,b}},$$

so that

$$G_{i,b} = m_{i,b} v_{i,b}, \qquad \|v_{i,b}\|_2 = 1$$

exactly. For a zero block, we define $v_{i,b} = 0$. The implementation uses $\epsilon = 10^{-12}$ in the normalization denominator only for numerical stability. MoARa applies low-rank projection only to the direction component $V \in \mathbb{R}^{m \times n}$ (where each row block $V_{i,b} = v_{i,b}$) and treats the magnitude component $M \in \mathbb{R}^{m \times K}$ (where $M_{i,b} = m_{i,b}$) as a separate optimizer state. The projected direction is $\tilde{V} = P_V P_V^\top V$, where $P_V \in \mathbb{R}^{m \times r}$ is obtained from the SVD of V. The reconstructed gradient is $\tilde{G}_{i,b} = m_{i,b} \cdot \tilde{v}_{i,b}$, where $\tilde{v}_{i,b} \in \mathbb{R}^B$ denotes the corresponding row block of $\tilde{V}$.

B.3.2 Magnitude-Factor Preservation

Recall that pure low-rank projection acts on the full gradient matrix as $G_{\text{proj}} = PP^\top G$. We formalize a property specific to MoARa's decomposition: the pre-projection block-magnitude factor is carried outside the direction projection, so it does not contribute to block-wise cosine misalignment.

Proposition 1. For the alignment diagnostic, let

$$\widetilde{V} = P_V P_V^\top V,$$

and let $\widetilde{v}_{i,b}$ denote the row block of $\widetilde{V}$ corresponding to row i and block b. MoARa reconstructs each gradient block as

$$\widetilde{G}_{i,b} = m_{i,b} \widetilde{v}_{i,b}.$$

Thus, the pre-projection magnitude factor $m_{i,b}$ is carried outside the direction projection and reintroduced unchanged. Whenever $G_{i,b}$ and $\widetilde{G}_{i,b}$ are both nonzero,

$$\cos(G_{i,b}, \widetilde{G}_{i,b}) = \cos(v_{i,b}, \widetilde{v}_{i,b}).$$

Proof. For a nonzero block, $G_{i,b} = m_{i,b} v_{i,b}$ and $\widetilde{G}_{i,b} = m_{i,b} \widetilde{v}_{i,b}$. Therefore,

$$\begin{aligned} \cos(G_{i,b}, \widetilde{G}_{i,b}) &= \frac{m_{i,b}^2 \langle v_{i,b}, \widetilde{v}_{i,b} \rangle}{m_{i,b}^2 \|v_{i,b}\|_2 \|\widetilde{v}_{i,b}\|_2} \\ &= \cos(v_{i,b}, \widetilde{v}_{i,b}). \end{aligned}$$

Hence, the scalar magnitude factor does not contribute to the block-wise cosine misalignment; any such misalignment is determined by the projected direction branch. □

Connection to weight normalization. Proposition 1 is the gradient-side analogue of the conditioning argument made by Salimans and Kingma (Salimans and Kingma, 2016) for weight normalization, who showed that reparameterizing a parameter w

as $w = g \cdot v/\|v\|$ improves the conditioning of stochastic gradient descent by decoupling scale and direction. DoRA (Liu et al., 2024a) extends this principle to parameter-efficient fine-tuning by decomposing pretrained weights into magnitude and direction. MoARa adapts this principle to the gradient side in the context of low-rank pretraining: the motivation is not optimization conditioning in itself, but separating block-scale information from projection-induced cosine distortion, complementing the rank allocation strategy of §3.2.

B.3.3 Block Size Trade-off: Detailed Derivation

Small-B regime (statistical noise). When B is small, each magnitude $m_{i,b}$ is computed over only a few coordinates. In the extreme $B = 1$, we have $m_{i,j} = |G_{i,j}|$ and $v_{i,j} = \mathrm{sign}(G_{i,j})$, with $v_{i,j} = 0$ when $G_{i,j} = 0$. Thus, $V \in \{-1, 0, +1\}^{m\times n}$, and the relative magnitude structure of G is absorbed into M. The directional branch V then operates on a sign-valued matrix that no longer contains the original coordinate-wise magnitude variation. This extreme normalization can reduce the information available to the directional branch. More generally, for small B, $m_{i,b}$ inherits the noise of small-sample L^2 norm estimates, becoming progressively less reliable as a summary of the block's energy. Under approximate within-row stationarity and weak dependence, let

$$X_{i,j} = G_{i,j}^2, \qquad \mu_2 = \mathbb{E}[X_{i,j}] > 0,$$

and assume $\mathrm{Var}(X_{i,j}) < \infty$. For the block energy

$$S_{i,b} = \|G_{i,b}\|_2^2 = \sum_{j\in b} X_{i,j},$$

we then have

$$\mathbb{E}[S_{i,b}] \approx B\mu_2, \qquad \mathrm{Var}(S_{i,b}) = O(B).$$

Applying a first-order delta-method approximation to $m_{i,b} = \sqrt{S_{i,b}}$ gives

$$\mathbb{E}[m_{i,b}] = \Theta(\sqrt{B}), \qquad \mathrm{Std}[m_{i,b}] = O(1),$$

and therefore

$$\frac{\mathrm{Std}[m_{i,b}]}{\mathbb{E}[m_{i,b}]} = O(B^{-1/2}).$$

Thus, the relative variability of the block-magnitude estimate decreases as B grows.

Optimizer-dynamics view at $B = 1$. At $B = 1$, all coordinate-wise magnitude variation is carried by M, while the direction optimizer receives only a projection of the sign-valued direction signal. This extreme separation between the two branches provides one possible explanation for the degraded $B = 1$ result in Figure 3.

Large-B regime (column-axis subspace coupling). When B is large, $m_{i,b}$ aggregates over heterogeneous coordinates that may span multiple attention heads. The unit direction $v_{i,b} = G_{i,b}/m_{i,b}$ then exhibits an attenuation phenomenon we call *column-axis subspace coupling*: coordinates with relatively small magnitude within $G_{i,b}$ are scaled down by a factor proportional to the dominant coordinates' magnitude. Formally, partitioning $G_{i,b}$ into a high-magnitude subset $\mathcal{A}$ and a low-magnitude subset $\mathcal{C}$ (with $|G_{i,a}| \gg |G_{i,c}|$ coordinate-wise for $a \in \mathcal{A}$, $c \in \mathcal{C}$), the corresponding directional sub-norm satisfies

$$\|v_{i,\mathcal{C}}\|_2 = \frac{\|G_{i,\mathcal{C}}\|_2}{\sqrt{\|G_{i,\mathcal{A}}\|_2^2 + \|G_{i,\mathcal{C}}\|_2^2}} \le \frac{\|G_{i,\mathcal{C}}\|_2}{\|G_{i,\mathcal{A}}\|_2},$$

which can become arbitrarily small as the magnitude ratio grows. In the limit $B = n$ (rowwise normalization), each $v_{i,1}$ spans an entire row and the dominant coordinate of $G_{i,:}$ suppresses information from all subdominant directions, explaining why an overly coarse decomposition suppresses weaker local signals. This effect is amplified when a block contains a few unusually large coordinates, because the shared magnitude scale is then dominated by those entries.

Balanced regime. The two effects intersect in an intermediate regime: small-B noise diminishes as B increases, while large-B coupling grows with B. For Transformer gradients, the attention-head dimension provides a natural architecture-informed reference scale for choosing the block granularity. This suggests selecting B in the neighborhood of d_{head}, without requiring exact head alignment. Appendix B.4 reports empirical evidence consistent with this choice, observing a plateau of near-optimal performance across $B \in [d_{\text{head}}/2,\ 2d_{\text{head}}]$ on Llama 2 350M ($d_{\text{head}} = 64$).

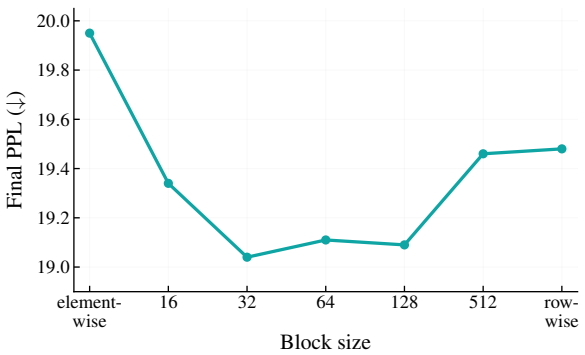

Figure 3: **Block size sensitivity sweep on Llama 2 350M 60k steps.** Final validation PPL as a function of $B \in \{1\,(\text{element}), 16, 32, 64, 128, 512, 1024\,(\text{row})\}$. The U-shape is consistent with the statistical–coupling trade-off analyzed in Appendix B.3.3. The plateau region $B \in \{32, 64, 128\}$ covers the head dimension of the Llama 2 scales we evaluate.

B.4 Block Size Sensitivity Sweep

We characterize the sensitivity of block-wise decomposition to the block size parameter B by sweeping $B \in \{1, 16, 32, 64, 128, 512, 1024\}$ on Llama 2 (Touvron et al., 2023) 350M while keeping all other settings identical to the full MoARa configuration. Figure 3 reports final validation perplexity at 60k steps as a function of B. The curve is U-shaped: both endpoints — elementwise ($B = 1$, PPL 19.95) and rowwise ($B = 1024$, PPL 19.48) — give clearly higher perplexity than the interior, while a region of small variation spans $B = 32$ through $B = 128$ (PPL 19.04, 19.11, 19.09 respectively; all within 0.07 PPL of the best).

The U-shaped trend is consistent with the statistical–coupling trade-off analyzed in Appendix B.3.3. At $B = 1$, the direction signal becomes sign-valued and loses coordinate-wise magnitude variation, which provides a possible qualitative explanation for the degraded trajectory. At the largest tested block size, the coarse block-wise magnitude statistics are less able to track heterogeneous local scale structure. The head dimension of Llama 2 350M sits inside the plateau region $B \in \{32, 64, 128\}$, providing the architecture-aware default used throughout our experiments.

C Module-aware Rank Allocation: Procedure and Analysis

This section provides the algorithmic procedure and supporting evidence for the module-aware projection-rank allocation introduced in §3.2.

Table 5: **Short profiling runs match the full-trajectory assignment.** Module-wise ranks of S_τ are identical across 1k, 10k, and reference 60k profiling runs on Llama 2 350M. Rank 1 denotes the highest cosine alignment and 7 the lowest. All module-wise ranks are identical across the three profiling settings.

Module	**1k** ∼11 min	**10k** ∼72 min	**60k (full)** at 6k
`attn.q`	5 (.899)	5 (.892)	5 (.892)
`attn.k`	6 (.885)	6 (.847)	6 (.851)
`attn.v`	1 (.959)	1 (.944)	1 (.971)
`attn.o`	2 (.932)	2 (.934)	2 (.960)
`mlp.up`	3 (.922)	3 (.900)	3 (.905)
`mlp.gate`	4 (.921)	4 (.898)	4 (.900)
`mlp.down`	7 (.819)	7 (.721)	7 (.734)

C.1 Profiling-Based Rank Allocation

Our rank reallocation is based on a profiling step conducted before the target MoARa training run. We first run the standard GaLore (Zhao et al., 2024) configuration for N_p steps with uniform rank r_{base} across all modules, logging the full gradient $G_t(W)$ and the reconstructed gradient $G_{\text{recon},t}(W) = \text{Rep}_B(M) \odot (P_\tau P_\tau^\top V)$ at sampled time points $t \in \mathcal{S}_p$. The module-wise alignment score S_τ is then computed as in Eq. (2), and the resulting integer rank allocation is fixed for the entire training run, consistent with our static-allocation design.

C.2 Profiling protocol and intervals

Three distinct intervals appear in our procedure and serve different purposes. The *profiling stride* x controls how often $S(W, t)$ is recorded during the baseline profiling run. The *profiling window* N_p is the total length of the baseline profiling run. The *SVD update interval* T_{SVD} is how often the projection matrix P is recomputed via TruncatedSVD during MoARa training. The profiling stride and window control the diagnostic only; only the SVD update interval is part of the MoARa training loop itself.

C.3 Profiling cost and short-run sufficiency

Profiling cost. The profiling phase reuses the standard GaLore training setup and adds negligible per-step overhead. The sampling interval x in $\mathcal{S}_p = \{x, 2x, \ldots, N_p\}$ controls only the temporal resolution of the logged diagnostic and does not affect the resulting rank allocation. In our profiling runs we use $x = 50$; this choice is made for monitoring efficiency rather than for the allocation

procedure itself.

The profiling phase is off the wall-clock-to-target path. The profiling run is conducted once before target training. All raw wall-clock trajectories and training-only time-to-target measurements exclude this setup phase. In §4.2, we additionally report non-amortized and amortized time-to-target by accounting for the profiling cost separately. The resulting donor/receiver assignment can be reused across subsequent target runs.

A short profiling run already suffices. We further find that this one-time cost itself can be made very small. On Llama 2 (Touvron et al., 2023) 350M, a 1k-step profiling run completes in approximately 11 minutes and a 10k-step run completes in approximately 72 minutes in our hardware settings. The budget-donor/receiver assignment obtained from either short run matches the assignment obtained from the full 60k-step profiling trajectory at its 6k checkpoint (Table 5). This indicates that, across short and long profiles, the donor/receiver assignment is established early in the training trajectory. The observation is consistent with our claim in §5 that rank allocation operates on an architectural property of the Transformer rather than on optimizer-specific dynamics.

Amortization across scales. A 1k-step profiling run on Llama 2 350M corresponds to roughly 2.6% of a single 350M target run (7.13 h, see Table 3). The same donor/receiver assignment is reused at the 1B and 7B scales, while the numerical ranks follow each scale's r_{base}. The relative profiling overhead therefore drops further at those scales. In practice, the profiling cost is small in absolute terms at 350M and effectively amortized when applied to larger target runs. Accordingly, we distinguish the target-training time T_{train} from the non-amortized time T_{train} + $T_{profile}$. When the same assignment is reused across K target runs, the corresponding amortized time-to-target is T_{train} + $T_{profile}$ / K; §4.2 reports this accounting alongside the headline results.

C.4 Profiling-stride and profiling-window sensitivity

We test the dependence of the budget-donor/receiver assignment on both the profiling stride and the profiling window. We re-sample the recorded baseline GaLore trajectory at strides of 100, 200, and 400 steps in addition to the default 50 on Llama 2 (Touvron et al., 2023) 350M. We then examine the resulting EMA-smoothed cosine similarities at four checkpoints (6k, 15k, 30k, 60k). Figure 4 reports the per-module trajectories. Table 6 reports the corresponding numerical rankings.

Donor and receiver positions are stable. `mlp.down` is the lowest-scoring module in all sixteen (window, stride) combinations. In every case, it occupies rank 7 by a substantial margin. `attn.k` occupies rank 6 in every case. `attn.q` occupies rank 4 or 5 in every case. The gap $S_K - S_{\text{Down}}$ stays in the range $[0.10, 0.15]$ across all combinations. `mlp.down` is therefore clearly separated as the unique receiver.

Adjacent swaps occur only within the upper tier and stay tightly bounded. Some swaps appear in Table 6 among modules whose values lie within 0.01 of one another. These all occur strictly within the upper-tier cluster ($\{$`attn.v`, `attn.o`, `mlp.up`, `mlp.gate`, `attn.q`$\}$). The largest observed within-tier difference is 0.011 (between `attn.v` and `attn.o` at 6k, stride 50) and the typical difference is below 0.008. These are about two orders of magnitude smaller than the donor–receiver boundary $S_K - S_{\text{Down}} \in [0.10, 0.15]$. The upper-tier ordering is therefore sensitive to small perturbations in stride or window length, but the donor/receiver assignment used by Algorithm 1 is unaffected.

Alignment ordering is established within warmup. The inset in Figure 4 zooms into the warmup region (0–6k). The donor/receiver ordering (`attn.q`, `attn.k` as donors; `mlp.down` as receiver) is already present within this region and is preserved across the remainder of the 60k-step run. This is consistent with the short-run sufficiency observed in Appendix C.3 (a 1k-step profiling run, which lies entirely inside the warmup region, already recovers the full assignment). The pattern remains unchanged across the evaluated profiling strides of 50, 100, 200, and 400 steps.

C.5 Final rank allocation

The integer rank allocation $\{r_\tau\}_{\tau\in\mathcal{T}}$ used in our main experiments follows directly from our default configuration: $\mathcal{D} = \{$`attn.q`, `attn.k`$\}$, $\mathcal{R} = \{$`mlp.down`$\}$, $\Delta r = r_{\text{base}}/2$. Per-module ranks then reduce to $r_Q = r_K = r_{\text{base}}/2$ (donors), $r_V = r_O = r_{\text{up}} = r_{\text{gate}} = r_{\text{base}}$ (uniform base-

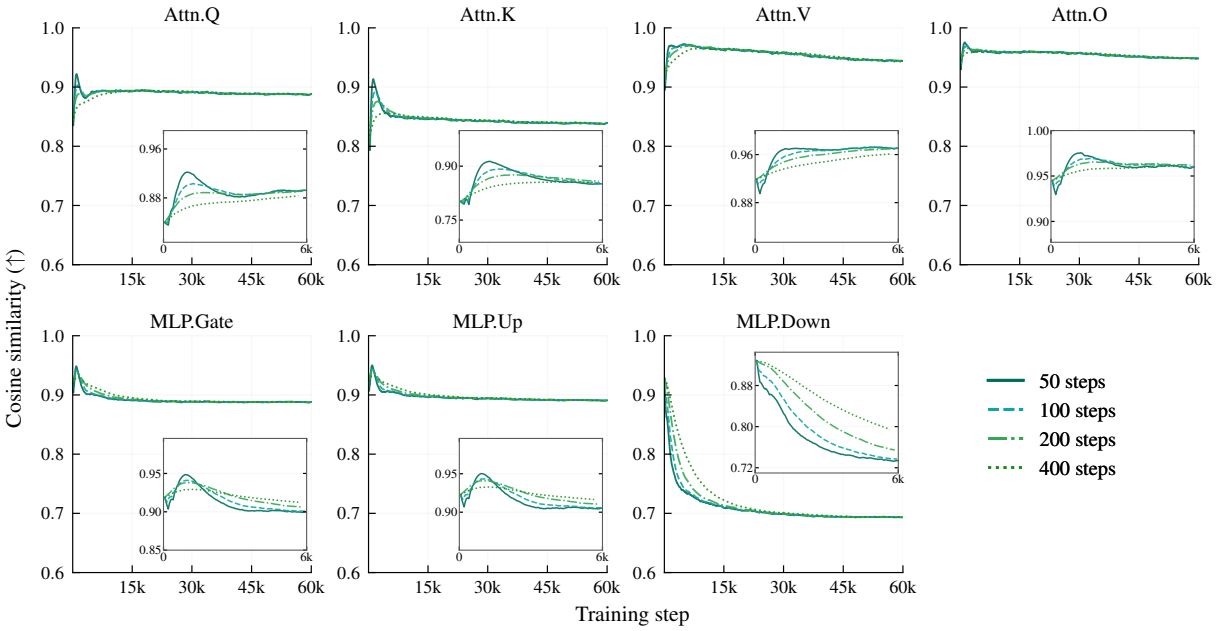

Figure 4: **Module-wise cosine similarity under varying profiling strides (Llama 2 350M, 60k-step standard GaLore).** Each panel plots the EMA of cosine similarity between full and reconstructed gradients for one projection module, sampled at strides of 50, 100, 200, and 400 steps. The inset zooms into the warmup region (0–6k). The donor/receiver assignment is insensitive to the choice of profiling stride.

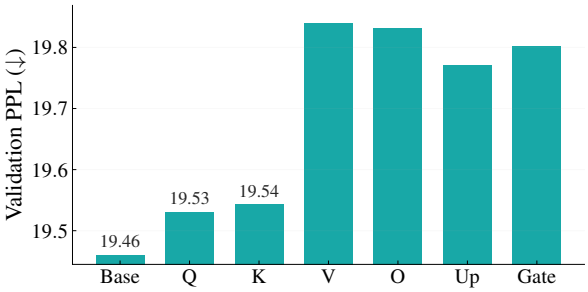

Figure 5: **Module-wise projection-rank sensitivity on Llama 2 350M.** We reduce the rank of one target module to $r = 128$ and keep the others at $r = 256$. `attn.q/k` are the most robust to rank reduction.

line), and $r_{\text{down}} = 2\, r_{\text{base}}$ (receiver). With $|\mathcal{D}| = 2$ and $|\mathcal{R}| = 1$, the total budget $\sum_\tau r_\tau = 7\, r_{\text{base}}$ is preserved by construction, with no residual adjustment required for this choice of Δr, $|\mathcal{D}|$, and $|\mathcal{R}|$. Per-scale baseline ranks r_{base} follow the GaLore (Zhao et al., 2024) convention at each model size and are reported in Appendix D.2.

C.6 Donor Suitability and Module-wise Sensitivity

Projection sensitivity and donor suitability are related but not identical. A module with high alignment robustness can be a donor candidate, but donor selection also depends on whether reducing its rank is less harmful than reducing the rank of other modules. To identify safe donors, we conduct a controlled module-wise projection-rank sensitivity ablation on Llama 2 (Touvron et al., 2023) 350M: we reduce one target module's projection rank from r_{base} to $r_{\text{base}}/2$ at a time while keeping all others at the uniform baseline rank. In our experiments at Llama 2 350M, this corresponds to reducing one target module to $r = 128$ while keeping the others at $r = 256$ (six configurations in total, one per non-receiver module).

Figure 5 reports the per-module final perplexity under this ablation. The donor/receiver implications and the underlying architectural reasoning that supports $\mathcal{D} = \{\texttt{attn.q}, \texttt{attn.k}\}$ are discussed in §3.2.

D Experimental Setup

D.1 Model Architecture

We follow the Llama configurations used in GaLore (Zhao et al., 2024) across model scales. For Llama 3.2 (Grattafiori et al., 2024), Qwen2.5 (Yang et al., 2024), Qwen3 (Yang et al., 2025), and DeepSeek-V2 (DeepSeek-AI et al., 2024), we derive compatible 350M-scale configurations from their official implementations, starting from our Llama 2 (Touvron et al., 2023) 350M setup. Specifically for DeepSeek-V2, which employs a distinct Multi-head Latent Attention (MLA) archi-

Table 6: **Module-wise cosine similarity rankings under different profiling intervals.** Values are EMA-smoothed cosine similarities. Modules are ordered from highest to lowest cosine similarity.

Step	Interval	Ranking by cosine similarity						
		1st	2nd	3rd	4th	5th	6th	7th
6k	50	V (0.9709)	O (0.9600)	Up (0.9052)	Gate (0.8997)	Q (0.8924)	K (0.8505)	Down (0.7339)
	100	V (0.9713)	O (0.9610)	Up (0.9061)	Gate (0.9010)	Q (0.8924)	K (0.8530)	Down (0.7371)
	200	V (0.9699)	O (0.9624)	Up (0.9103)	Gate (0.9059)	Q (0.8914)	K (0.8578)	Down (0.7533)
	400	V (0.9623)	O (0.9614)	Up (0.9158)	Gate (0.9117)	Q (0.8844)	K (0.8550)	Down (0.7897)
15k	50	V (0.9617)	O (0.9578)	Up (0.8959)	Q (0.8926)	Gate (0.8905)	K (0.8455)	Down (0.7091)
	100	V (0.9627)	O (0.9583)	Up (0.8968)	Q (0.8928)	Gate (0.8912)	K (0.8459)	Down (0.7103)
	200	V (0.9643)	O (0.9594)	Up (0.8986)	Q (0.8939)	Gate (0.8929)	K (0.8475)	Down (0.7131)
	400	V (0.9643)	O (0.9591)	Up (0.9008)	Gate (0.8953)	Q (0.8935)	K (0.8489)	Down (0.7219)
30k	50	V (0.9579)	O (0.9574)	Up (0.8942)	Q (0.8915)	Gate (0.8884)	K (0.8424)	Down (0.6992)
	100	V (0.9590)	O (0.9581)	Up (0.8947)	Q (0.8918)	Gate (0.8890)	K (0.8428)	Down (0.6997)
	200	V (0.9590)	O (0.9581)	Up (0.8949)	Q (0.8920)	Gate (0.8893)	K (0.8431)	Down (0.7002)
	400	V (0.9592)	O (0.9579)	Up (0.8952)	Q (0.8926)	Gate (0.8895)	K (0.8440)	Down (0.7016)
60k	50	O (0.9484)	V (0.9442)	Up (0.8909)	Q (0.8884)	Gate (0.8881)	K (0.8395)	Down (0.6936)
	100	O (0.9487)	V (0.9444)	Up (0.8908)	Q (0.8880)	Gate (0.8879)	K (0.8392)	Down (0.6937)
	200	O (0.9488)	V (0.9447)	Up (0.8908)	Q (0.8879)	Gate (0.8879)	K (0.8387)	Down (0.6936)
	400	O (0.9484)	V (0.9439)	Up (0.8904)	Gate (0.8876)	Q (0.8874)	K (0.8380)	Down (0.6936)

tecture, we carefully scale the latent projection dimensions to align with the 350M parameter budget (`q_lora_rank`=768, `kv_lora_rank`=256, `qk_nope_head_dim`=64, `qk_rope_head_dim`=32, and `v_head_dim`=64). Table 7 summarizes the key architectural and training hyperparameters across all evaluated models.

D.2 Training Parameters

We use the C4 English dataset with a T5 tokenizer (Raffel et al., 2020). We use AdamW (Loshchilov and Hutter, 2019) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-6}$, zero weight decay, and bias correction enabled. The peak learning rate is 0.01 (0.005 for 7B), with 10% linear warmup and cosine decay to a minimum ratio of 0.1. Unless otherwise noted, we use a global batch size of 512 and a maximum sequence length of 256.

For GaLore (Zhao et al., 2024), we use a scale of 0.25 and an interval $T = 200$. The magnitude branch M and its optimizer states are maintained in fp32, and the recomposed update is cast to bf16 before application. All runs use `torch.compile` in default mode. Gradient clipping of 1.0 is applied only to the 7B model.

Block size and rank budget per model scale. The block size B used in the main experiments follows the head-dimension-informed default discussed in §3.1: we choose B in the neighborhood of d_{head}, using $B = 32$ for Llama 2 350M ($d_{\text{head}} = 64$), $B = 64$ for Llama 2 1B ($d_{\text{head}} = 64$), and $B = 128$ for Llama 2 7B ($d_{\text{head}} = 128$). Per-scale baseline projection ranks r_{base} follow the GaLore convention: $r_{\text{base}} = 256$ at 350M, $r_{\text{base}} = 512$ at 1B, and $r_{\text{base}} = 1024$ at 7B; module-wise allocation $\{r_\tau\}$ follows the static reallocation rule of Algorithm 1 with $\mathcal{D} = \{Q, K\}$, $\mathcal{R} = \{\text{MLP.down}\}$, and $\Delta r = r_{\text{base}}/2$ (Appendix C.5).

Table 7: **Model architectures and pre-training hyperparameters.** We report architecture details and the pre-training budget for each evaluated model.

Model	Params	Architecture					Pre-training	
		Hidden	Interm.	Heads	Layers	KV	Steps	Tokens
Scaling study								
Llama 2	350M	1024	2736	16	24	16	60K	7.8B
Llama 2	1B	2048	5461	32	24	16	100K	13.1B
Llama 2	7B	4096	11008	32	32	16	150K	19.7B
Model-family comparison								
Llama 3.2	300M	1024	2736	16	24	4	60K	7.8B
Qwen2.5	350M	1024	2816	16	24	4	60K	7.8B
Qwen3	350M	1024	2816	16	24	4	60K	7.8B
DeepSeek-V2	350M	1024	2736	16	24	16	60K	7.8B

D.3 Hardware and Profiling Details

For all step-to-target and time-to-target experiments, GaLore (Zhao et al., 2024) and MoARa are compared under identical hardware conditions, using NVIDIA RTX A6000, H100, and RTX PRO 6000 GPUs depending on model scale.

For memory profiling, we measure memory usage after the first 50 training steps under the same sequence-length and batch-size settings as in Appendix D.2.

E Additional Experimental Validation

E.1 Random Rank Allocation

We performed a validation experiment to verify that our projection-rank redistribution outperforms random budget-preserving allocations. Under identical projection rank budget constraints, we pretrained 20 Llama 2 350M models with randomly assigned ranks for 10k steps and compared them with our method. The projection ranks and resulting evaluation PPL for each of the 20 runs are summarized in Table 8. Our module-aware allocation outperforms all 20 random variants, and none approach its convergence trajectory.

Table 8: **Random rank allocations and final evaluation perplexity.** Each run preserves the same total projection-rank budget. G, U, and D denote the gate, up, and down MLP projections. Lower PPL is better.

Run	Q	K	V	O	G	U	D	PPL
R01	128	240	560	184	256	80	344	24.72
R02	184	320	280	320	352	272	64	24.85
R03	216	208	248	328	376	304	112	24.53
R04	160	472	72	360	256	80	392	25.00
R05	352	192	88	448	408	128	176	24.88
R06	96	136	112	608	456	232	152	24.53
R07	352	256	192	208	120	536	128	24.83
R08	552	104	192	128	64	440	312	24.92
R09	200	360	104	344	376	80	328	24.73
R10	128	312	416	200	328	72	336	24.60
R11	72	568	376	80	72	152	472	25.29
R12	160	208	64	680	128	360	192	24.78
R13	144	208	344	248	328	312	208	24.33
R14	256	480	80	232	472	120	152	25.17
R15	184	224	256	312	560	136	120	24.63
R16	144	392	360	192	248	336	120	24.72
R17	232	304	64	424	376	304	88	24.88
R18	184	64	112	592	176	504	160	24.56
R19	136	72	720	80	224	288	272	24.73
R20	152	64	256	536	496	80	208	24.58
Ours	**128**	**128**	**256**	**256**	**256**	**256**	**512**	**24.07**

E.2 Cross-Architecture Generalization

Figure 6 reports the full validation PPL curves for the cross-architecture study referenced in §4.2. Across Llama 3.2 (Grattafiori et al., 2024), Qwen2.5 (Yang et al., 2024), Qwen3 (Yang et al., 2025), and DeepSeek-V2 (DeepSeek-AI et al., 2024), MoARa reaches the standard GaLore (Zhao et al., 2024) target earlier in all cases despite the architectural differences (GQA in Llama 3.2 and the Qwen family, MLA in DeepSeek-V2). The cross-scale speedup figure (Figure 7) is included alongside for visual reference.

E.3 Robustness at Sequence Length 2048

The main scaling and architecture results in §4.2 use the GaLore (Zhao et al., 2024)-standard sequence length of 256, which matches the GaLore baseline's setup and isolates the optimizer-side

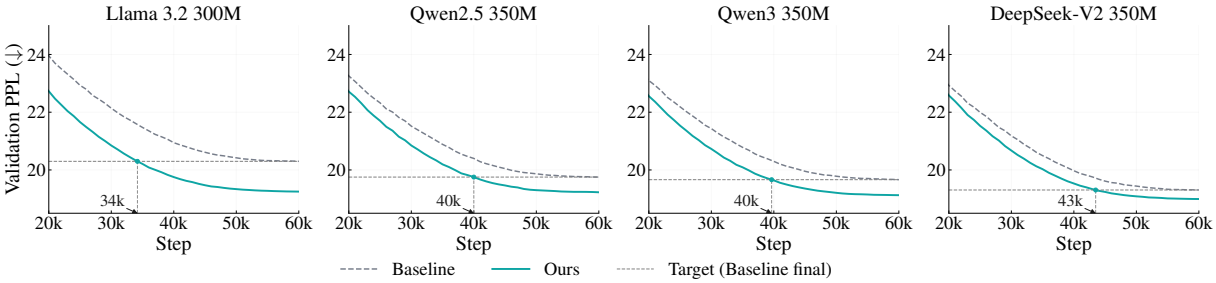

Figure 6: **Validation perplexity curves across recent Transformer architectures at the 350M scale.** We compare our method with GaLore on Llama 3.2, Qwen2.5, Qwen3, and DeepSeek-V2. The dashed horizontal line marks the final validation perplexity of the baseline. In all cases, our method reaches this target earlier than the baseline.

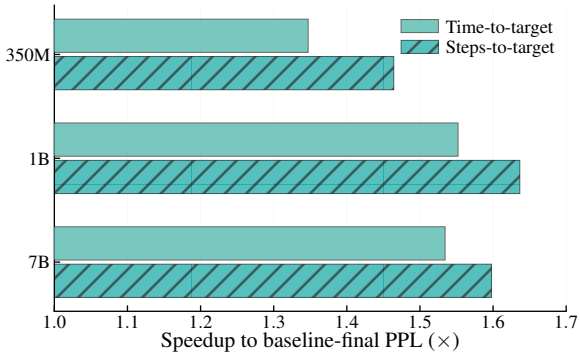

Figure 7: **Speedup across model scales.** MoARa's speedup over standard GaLore in reaching the baseline's final perplexity, measured by both steps-to-target and time-to-target. GaLore with MoARa achieves up to $1.64\times$ step speedup across all three configurations.

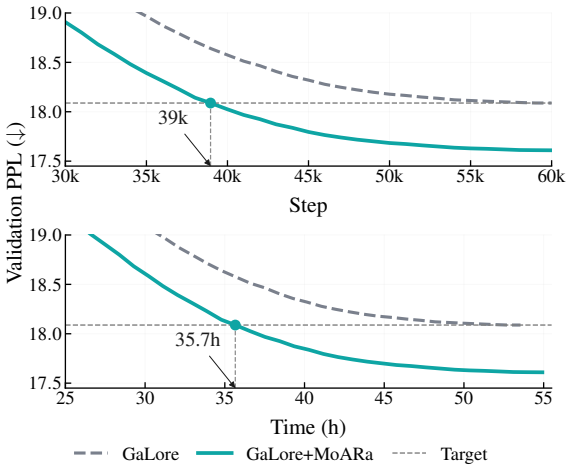

Figure 8: **Sequence-length 2048 validation perplexity on Llama 2 350M.** Validation perplexity versus training steps (top) and wall-clock time (bottom); annotated points mark where GaLore with MoARa reaches the standard GaLore target (dashed line).

comparison from sequence-length-dependent variables. To test whether the efficiency gains persist beyond this short-context setting, we re-evaluate GaLore with MoARa against standard GaLore at sequence length 2048 on Llama 2 (Touvron et al., 2023) 350M. All other training hyperparameters are kept identical to the sequence-length-256 setup of Appendix D.2.

As shown in Figure 8, GaLore with MoARa reaches standard GaLore's final validation perplexity in 35.0% fewer steps and 33.3% less wall-clock time, and improves final perplexity by 2.65% (PPL 18.09 $\rightarrow$ 17.61). The reduction profile in both steps and wall-clock time is consistent with the sequence-length-256 result at the same model scale. This single-scale experiment supports an optimizer-side interpretation but does not establish sequence-length invariance at larger scales or substantially longer contexts.

E.4 Multi-seed Robustness

Figure 9 shows the full 1B pre-training trajectories over three random seeds. Our method consistently outperforms the baseline in all runs, which supports the robustness claim in §4.2.

E.5 Q-GaLore Precision Analysis

We isolate whether the magnitude-branch optimizer-state precision explains the Group C behavior observed in §5. The official Q-GaLore configuration uses INT4 projection matrices, INT8 weights, and an 8-bit Adam optimizer. We therefore evaluate Q-GaLore + decomposition while changing only the magnitude-branch optimizer state from 8-bit to FP32; the INT4 projection matrices and INT8 weights remain unchanged. This change does not recover the benefit of decomposition. Thus, the magnitude-branch optimizer-state precision is unlikely to be the

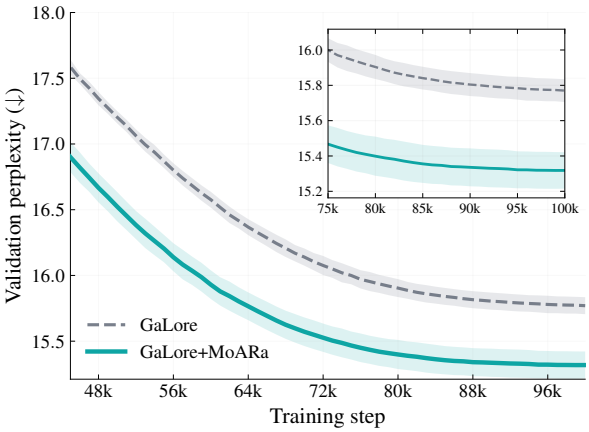

Figure 9: **Validation perplexity across three random seeds on Llama 2 1B.** Our method consistently outperforms the baseline. The inset zooms into the final phase.

primary bottleneck. Because projection and weight quantization remain active together in this ablation, we do not attribute the limitation to either component individually. Rather, the result supports a broader quantization-compatibility limitation for decomposition. A direct comparison with GaLore + decomposition is consistent with this interpretation: decomposition alone yields a final perplexity comparable to vanilla GaLore, whereas the same decomposition under Q-GaLore produces a small but consistent increase in final perplexity.

F Detailed Benchmark Results

This section reports the full benchmark tables that support the summary results in §4.2. Table 9 shows the three-seed evaluation at the final checkpoint on seven downstream benchmarks (HellaSwag (Zellers et al., 2019), Wino-Grande (Sakaguchi et al., 2020), PIQA (Bisk et al., 2020), BoolQ (Clark et al., 2019), ARC-Challenge and ARC-Easy (Clark et al., 2018), and TruthfulQA (Lin et al., 2022)) together with C4 (Raffel et al., 2020) and WikiText-2 (Merity et al., 2017) perplexity, reporting mean and standard deviation over three random seeds. Table 10 compares our 60k-step intermediate checkpoint with the 100k-step GaLore (Zhao et al., 2024) baseline to demonstrate token efficiency rather than final-checkpoint parity.

Multi-seed robustness at the final checkpoint. The three-seed evaluation in Table 9 shows that GaLore (Zhao et al., 2024) with MoARa is comparable to or outperforms GaLore on the vast majority of metrics. MoARa has lower standard deviation on all six perplexity metrics and on several downstream metrics. The consistent mean trends across seeds indicate that the reported gains are not driven by a single favorable seed.

ARC-Challenge under short-context pre-training. Among the summary metrics reported in Table 1, ARC-Challenge (Clark et al., 2018) normalized accuracy is the only metric that decreases. We interpret this gap with caution. ARC-Challenge is a four-way multiple-choice benchmark in which chance-level accuracy is 0.25, and both models attain mean normalized accuracy within 0.003 of chance level. The task which requires multi-step science reasoning and remains challenging for our 1B model pre-trained at sequence length 256 sits very near the chance-level floor for both checkpoints, so the per-seed differences reflect noise around that floor rather than a substantive capability gap. This interpretation is consistent with the modest seed-to-seed variation reported in Table 9.

60k MoARa versus 100k GaLore: downstream behavior at an earlier checkpoint. Table 10 compares the 60k-step intermediate checkpoint from our 100k-step GaLore (Zhao et al., 2024) with MoARa run against the final 100k-step GaLore checkpoint. This checkpoint is selected because its pretraining validation perplexity is close to GaLore's final training-run value. On the separate C4 and WikiText evaluation metrics in Table 10, however, GaLore retains a clear perplexity advantage, as MoARa has seen approximately 40% fewer training tokens (7.86B vs. 13.1B at a context length of 256).

Despite this perplexity gap, the downstream picture is different. Ten of the eleven reported downstream comparisons fall within two standard errors of the GaLore baseline, and on BoolQ the 60k GaLore with MoARa checkpoint actually exceeds the 100k GaLore baseline. The reductions in downstream accuracy that do appear are uniformly small in absolute terms. ARC-Challenge remains near the chance-level floor in normalized accuracy, whereas the ARC-Easy and PIQA differences are smaller than the corresponding standard errors reported by the evaluation harness.

Why does the downstream gap not track the perplexity gap? Two complementary explanations are consistent with this pattern. First, one possible explanation is that the additional 40% of

Table 9: **Detailed zero-shot benchmark results on Llama 2 1B.** Both standard GaLore and GaLore with MoARa are evaluated at their final 100k-step checkpoints. We report per-seed scores, mean μ, and standard deviation σ over three seeds. We abbreviate bits-per-byte, byte-perplexity, word-perplexity, and normalized accuracy as bpb, bppl, wppl, and acc_n.

Task	Metric	Dir.	GaLore					GaLore with MoARa				
			1	**2**	**3**	μ	σ	**1**	**2**	**3**	μ	σ
C4	bpb	↓	1.077	1.091	1.060	1.076	0.016	1.056	1.063	1.073	**1.064**	0.008
	bppl	↓	2.110	2.130	2.084	2.108	0.023	2.079	2.089	2.103	**2.091**	0.012
	wppl	↓	87.18	92.34	81.13	86.88	5.612	79.96	82.29	85.61	**82.62**	2.837
WikiText	bpb	↓	1.135	1.136	1.103	1.125	0.019	1.108	1.098	1.111	**1.106**	0.007
	bppl	↓	2.197	2.197	2.148	2.181	0.029	2.156	2.141	2.159	**2.152**	0.010
	wppl	↓	67.218	67.332	59.588	64.712	4.439	60.856	58.616	61.334	**60.269**	1.451
ARC-C	acc	↑	0.212	0.215	0.212	**0.213**	0.002	0.206	0.202	0.197	0.202	0.004
	acc_n	↑	0.255	0.248	0.252	**0.252**	0.004	0.244	0.257	0.244	0.248	0.007
ARC-E	acc	↑	0.477	0.469	0.484	0.477	0.008	0.487	0.476	0.481	**0.482**	0.006
	acc_n	↑	0.423	0.421	0.430	0.425	0.005	0.436	0.434	0.430	**0.433**	0.003
BoolQ	acc	↑	0.586	0.556	0.495	0.546	0.046	0.565	0.557	0.552	**0.558**	0.007
HellaSwag	acc	↑	0.326	0.325	0.326	0.326	0.001	0.329	0.331	0.332	**0.331**	0.001
	acc_n	↑	0.386	0.389	0.384	0.386	0.002	0.402	0.394	0.399	**0.398**	0.004
PIQA	acc	↑	0.679	0.681	0.680	**0.680**	0.001	0.675	0.677	0.682	0.678	0.004
	acc_n	↑	0.674	0.677	0.678	0.677	0.002	0.674	0.684	0.687	**0.682**	0.007
WinoGrande	acc	↑	0.504	0.526	0.508	0.513	0.012	0.527	0.539	0.526	**0.531**	0.007
TruthfulQA	mc2	↑	0.398	0.421	0.407	**0.409**	0.011	0.417	0.393	0.408	0.406	0.012

training tokens primarily refine the next-token distribution along directions that lower per-token surprisal on general-domain text without unlocking qualitatively new downstream behaviors. This is consistent with the broader observation that perplexity continues to decrease smoothly with more tokens, while downstream capability often follows a more saturating curve in this token range. Second, our main step-to-target results (§4.2) already demonstrate that GaLore with MoARa reaches GaLore's perplexity target with substantially fewer tokens; Table 10 extends this token-efficiency observation by showing that downstream behavior is largely retained at the point where GaLore with MoARa is still trailing GaLore in raw perplexity. We do not interpret the BoolQ improvement at 60k as evidence of a separate generalization gain.

G Detailed Memory Analysis

This section provides the comprehensive memory measurements used in §4.2.

For a target matrix W, let r_W denote its projection rank and d_W the state dimension not compressed by the projection. Let b_V and b_M denote the bytes per entry in each direction-branch and magnitude-branch moment tensor, respectively. The first- and second-moment states of the projected direction require approximately

$$\mathcal{M}_{\text{dir}}(W) = 2b_V r_W d_W.$$

Relative to the uniform baseline rank r_{base}, projection rank reallocation changes the persistent optimizer-state cost by

$$\Delta\mathcal{M}_{\text{rank}} = 2b_V \sum_W (r_W - r_{\text{base}})\, d_W.$$

For $W \in \mathbb{R}^{m_W \times n_W}$, the block-magnitude tensor has shape

$$m_W \times \left\lceil \frac{n_W}{B_W} \right\rceil,$$

so its Adam states add approximately

$$\Delta\mathcal{M}_{\text{mag}} = 2b_M \sum_W m_W \left\lceil \frac{n_W}{B_W} \right\rceil.$$

The sums include all target matrices across all layers, and the factor of 2 accounts for Adam's first and second moments. Because donor and receiver matrices can have different d_W, preserving the total projection-rank budget does not imply identical optimizer-state bytes. At 7B, the measured optimizer-state increases are 864 MiB from projection-rank allocation and 386 MiB from block-wise decomposition; together they match the 1,250 MiB increase of full MoARa.

Table 11(a) reports scale-wise memory profiling under both eager and compiled settings; notably, under the compiled setting, most intermediate overhead is absorbed, leaving the remaining

Table 10: **Zero-shot benchmarks on Llama 2 1B: 60k vs. 100k.** We compare standard GaLore at 100k steps with GaLore with MoARa at 60k steps. Δ denotes the difference between GaLore with MoARa and GaLore. $\pm$ denotes the standard error reported by the evaluation harness. Results are from a single seed. We abbreviate bits-per-byte, byte-perplexity, word-perplexity, and normalized accuracy as bpb, bppl, wppl, and acc_n.

Task	Metric	Dir.	GaLore	GaLore+MoARa	Δ
C4	bpb	$\downarrow$	1.060	1.085	+0.025
	bppl	$\downarrow$	2.084	2.121	+0.036
	wppl	$\downarrow$	81.130	89.981	+8.851
WikiText	bpb	$\downarrow$	1.103	1.125	+0.022
	bppl	$\downarrow$	2.148	2.180	+0.033
	wppl	$\downarrow$	59.588	64.603	+5.016
ARC-C	acc	$\uparrow$	0.212 ± 0.012	0.192 ± 0.012	-0.020
	acc_n	$\uparrow$	0.252 ± 0.013	0.247 ± 0.013	-0.005
ARC-E	acc	$\uparrow$	0.484 ± 0.010	0.475 ± 0.010	-0.009
	acc_n	$\uparrow$	0.430 ± 0.010	0.422 ± 0.010	-0.008
BoolQ	acc	$\uparrow$	0.495 ± 0.009	0.561 ± 0.009	+0.066
HellaSwag	acc	$\uparrow$	0.326 ± 0.005	0.329 ± 0.005	+0.003
	acc_n	$\uparrow$	0.384 ± 0.005	0.393 ± 0.005	+0.008
PIQA	acc	$\uparrow$	0.680 ± 0.011	0.676 ± 0.011	-0.004
	acc_n	$\uparrow$	0.678 ± 0.011	0.675 ± 0.011	-0.003
WinoGrande	acc	$\uparrow$	0.508 ± 0.014	0.511 ± 0.014	+0.004
TruthfulQA	mc2	$\uparrow$	0.407 ± 0.015	0.408 ± 0.015	+0.001

cost dominated by the added optimizer states. Furthermore, Table 11(b) breaks down the memory cost of each component, with all measurements taken in eager mode to effectively isolate the algorithmic overhead. This granular breakdown allows us to separate the specific memory requirements of block-wise decomposition from those of projection rank allocation, showing that the two components combine approximately additively across model scales.

H Cosine similarity of G and G_{recon}

This section reports module-wise cosine similarity trajectories between the full-rank gradient G and the reconstructed gradient G_{recon}. These plots support the alignment analysis in §3.2 and show how our projection rank reallocation improves low-alignment bottleneck modules across models.

H.1 Llama 2 350M

We first show the full module-wise cosine similarity trajectories on Llama 2 (Touvron et al., 2023) 350M in Figure 10. The baseline shows the lowest alignment on `mlp.down`, while our method improves this bottleneck by reallocating projection rank from robust donor modules.

H.2 Llama 2 1B

Figure 11 shows the same analysis on Llama 2 (Touvron et al., 2023) 1B. The overall pattern is consistent with the 350M case: the main improvement appears on the most projection-sensitive modules.

H.3 Llama 2 7B

Figure 12 reports the trajectories on Llama 2 (Touvron et al., 2023) 7B. The receiver-side alignment bottleneck and its improvement remain consistent at the larger scale.

H.4 Llama 3.2 300M

Figure 13 shows the trajectories on Llama 3.2 (Grattafiori et al., 2024) 300M. The same trend appears in this architecture as well, supporting the cross-architecture consistency of our alignment-based analysis.

H.5 Qwen2.5 350M

Figure 14 shows the trajectories on Qwen2.5 (Yang et al., 2024) 350M. Again, our method improves the lowest-alignment receiver while exhibiting the expected donor-side alignment reductions.

H.6 Qwen3 350M

Figure 15 shows the trajectories on Qwen3 (Yang et al., 2025) 350M. The result is consistent with the earlier models and supports the same bottleneck-resolution pattern.

H.7 DeepSeek-V2 350M

Figure 16 reports the trajectories on the dense variant of DeepSeek-V2 (DeepSeek-AI et al., 2024) 350M. Although the attention module structure differs from standard MHA-based models, the same qualitative pattern remains: our method improves the most sensitive modules while keeping the donor-side degradation small.

I Licenses and Intended Use

All artifacts used in this work are public research artifacts used in an academic research context consistent with their intended use.

Dataset and tokenizer. The C4 English corpus (Raffel et al., 2020) is released by AllenAI under ODC-BY 1.0, with use also subject to the Common Crawl terms of use. The T5 tokenizer is released by Google under Apache-2.0.

Model architectures. Llama 2 (Touvron et al., 2023) and Llama 3.2 (Grattafiori et al., 2024) reference implementations are released by Meta under the Llama 2 Community License and the Llama 3.2 Community License, respectively. Qwen2.5 (Yang et al., 2024) and Qwen3 (Yang et al., 2025) reference implementations are released by Alibaba under Apache-2.0. The DeepSeek-V2 (DeepSeek-AI et al., 2024) reference implementation is released under MIT (code license); the separately released pretrained weights, which we do not use, are under the DeepSeek Model License. For all five architectures, we use only the reference implementations to derive 350M-scale configurations for from-scratch pre-training and do not redistribute any original pretrained checkpoints.

Baseline optimizer implementations. The reference implementations of GaLore (Zhao et al., 2024), Fira (Chen et al., 2025), Q-GaLore (Zhang et al., 2025b), OSD (Liang et al., 2024), and LDAdam (Robert et al., 2025) are released under Apache-2.0. The SubTrack++ (Rajabi et al., 2025) reference implementation did not include an explicit license file at the time of access. We use it only as an academic evaluation baseline and do not redistribute its code.

Released artifacts. We release no new pretrained model checkpoints or datasets in this submission.

Table 11: **Memory profiling and component-wise memory ablations.** Values are reported in MiB. Panel (a) compares eager (OFF) and compiled (ON) settings across model scales; Δ denotes the increase of our method over the baseline. Panel (b) reports component-wise memory ablations; Δ rows show the relative change from the baseline.

(a) Memory profiling under eager and compiled settings

Setting	Metric	350M		1B		7B	
		Base	**Ours**	**Base**	**Ours**	**Base**	**Ours**
OFF	Max alloc.	47258.61	47316.03	51611.42	51611.42	60898.64	64790.63
	Δ	+57.42 (+0.12%)		+0.00 (+0.00%)		+3891.99 (+6.39%)	
	Max reserved	48154.00	48210.00	54988.00	55304.00	62122.00	66738.00
	Δ	+56.00 (+0.12%)		+316.00 (+0.57%)		+4616.00 (+7.43%)	
ON	Max alloc.	33740.11	33804.05	38208.39	38208.77	61777.64	61777.68
	Δ	**+63.94 (+0.19%)**		**+0.38 (+0.00%)**		**+0.04 (+0.00%)**	
	Max reserved	34632.00	34654.00	40850.00	41166.00	63486.00	63614.00
	Δ	**+22.00 (+0.06%)**		**+316.00 (+0.77%)**		**+128.00 (+0.20%)**	
OFF/ON	Opt. state	538.57	650.88	1652.34	1956.57	7177.02	8427.02
	Δ	+112.31 (+20.85%)		+304.23 (+18.41%)		+1250.00 (+17.42%)	

(b) Component-wise memory ablation

Model	Metric	Baseline	Decomp.	Rank Alloc.	Ours
350M	Max alloc.	47258.61	47280.14	47258.61	47316.03
	Δ	–	+0.05%	+0.00%	+0.12%
	Max reserved	48154.00	48232.00	48130.00	48210.00
	Δ	–	+0.16%	-0.05%	+0.12%
	Opt. state	538.57	610.75	578.69	650.88
	Δ	–	+13.40%	+7.45%	+20.85%
1B	Max alloc.	51611.42	51611.42	51611.42	51611.42
	Δ	–	+0.00%	+0.00%	+0.00%
	Max reserved	54988.00	55150.00	55132.00	55304.00
	Δ	–	+0.29%	+0.26%	+0.57%
	Opt. state	1652.34	1796.58	1812.32	1956.57
	Δ	–	+8.73%	+9.68%	+18.41%
7B	Max alloc.	60898.64	64790.63	64790.63	64790.63
	Δ	–	+6.39%	+6.39%	+6.39%
	Max reserved	62122.00	66738.00	66610.00	66738.00
	Δ	–	+7.43%	+7.22%	+7.43%
	Opt. state	7177.02	7563.02	8041.02	8427.02
	Δ	–	+5.38%	+12.04%	+17.42%

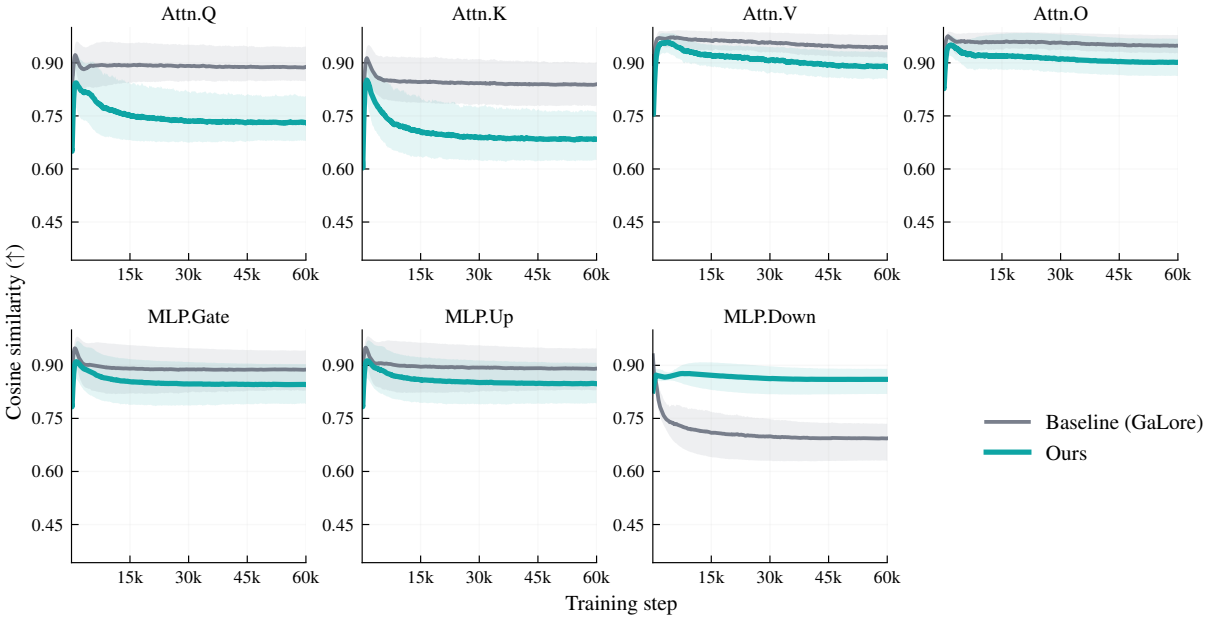

Figure 10: **Cosine similarity trajectories on Llama 2 350M.** We track the cosine similarity between G and G_{recon} for each target module during pre-training.

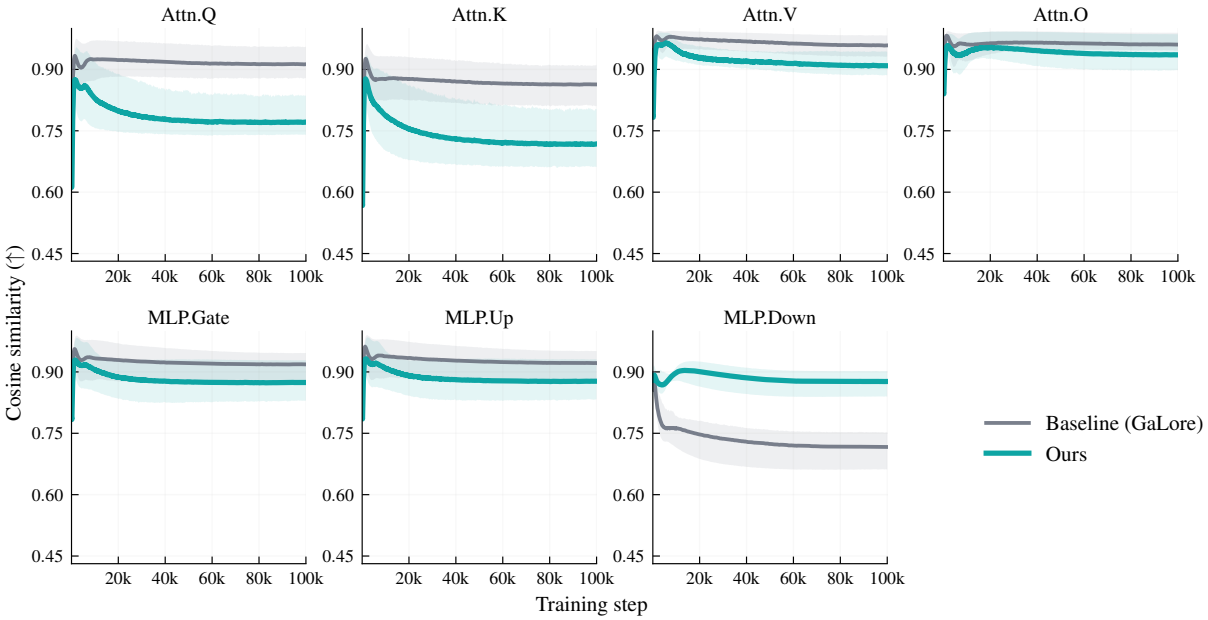

Figure 11: **Cosine similarity trajectories on Llama 2 1B.** The same module-wise alignment analysis is repeated at the 1B scale.

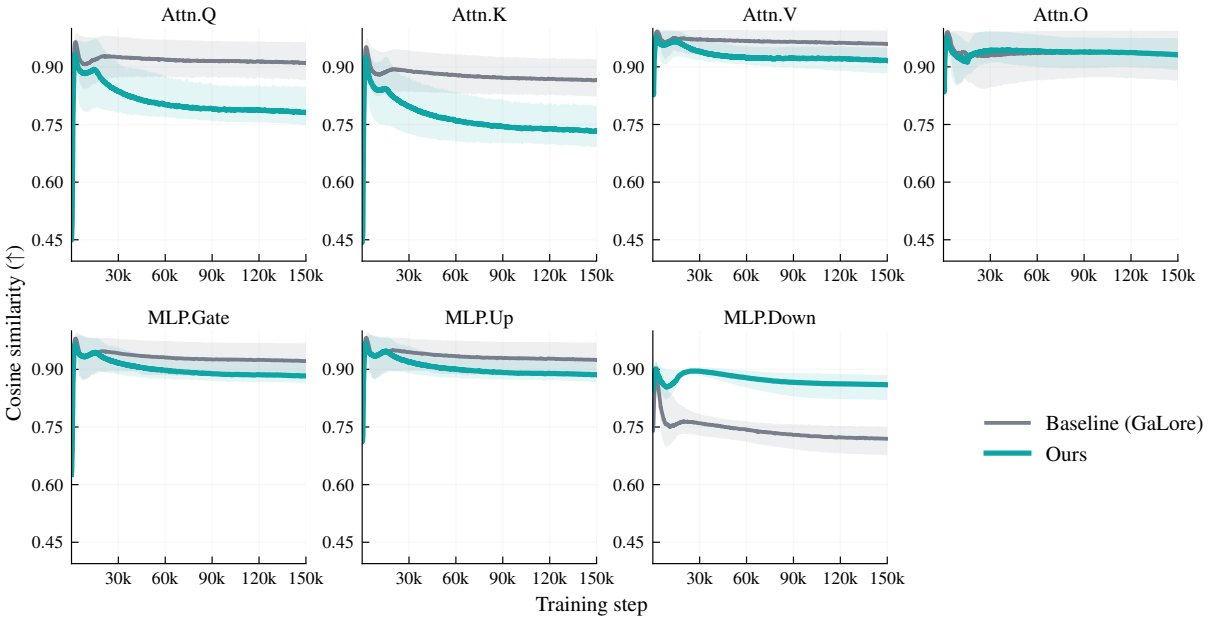

Figure 12: **Cosine similarity trajectories on Llama 2 7B.** The same module-wise alignment analysis is repeated at the 7B scale.

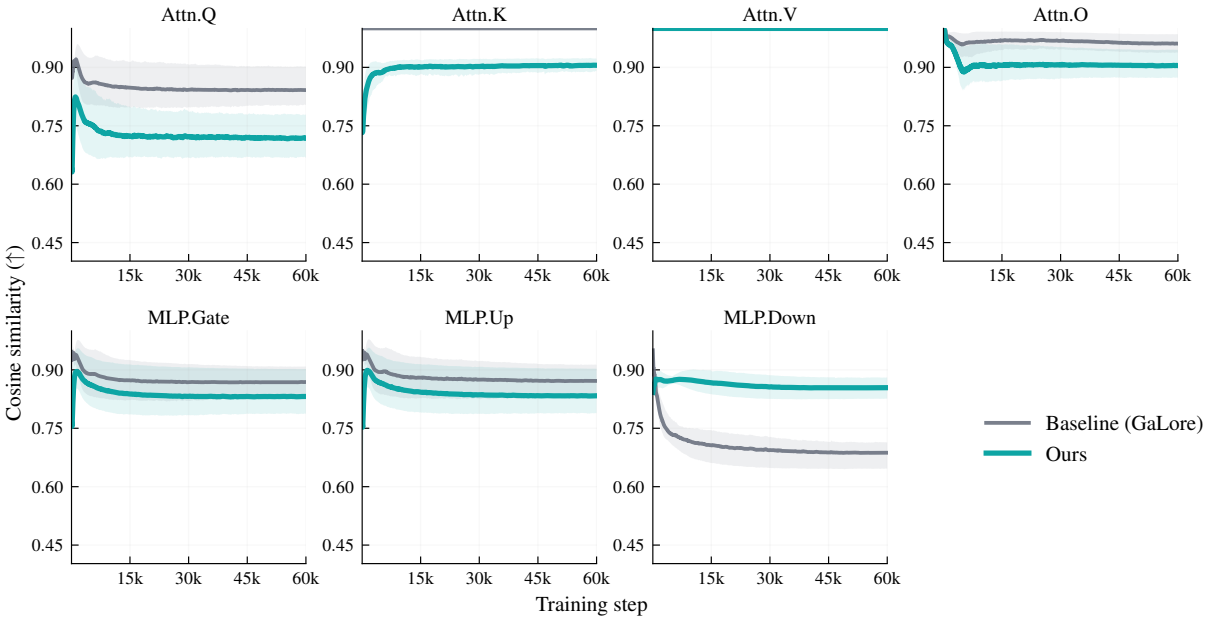

Figure 13: **Cosine similarity trajectories on Llama 3.2 300M.** The same analysis is applied to Llama 3.2.

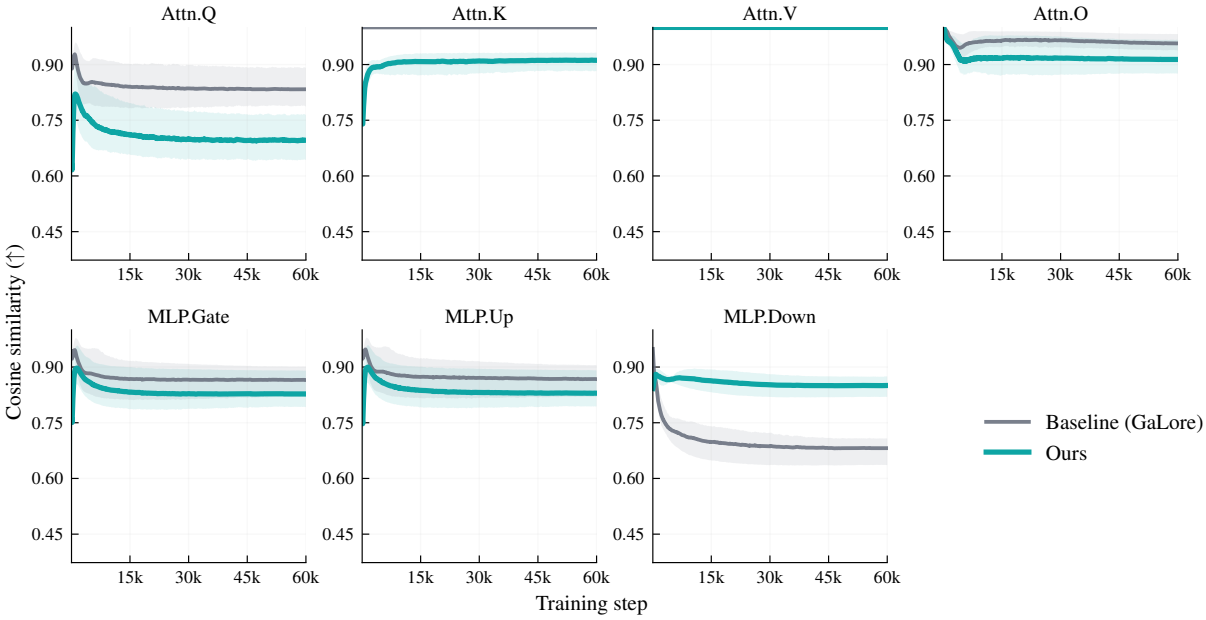

Figure 14: **Cosine similarity trajectories on Qwen2.5 350M.** The same analysis is applied to Qwen2.5.

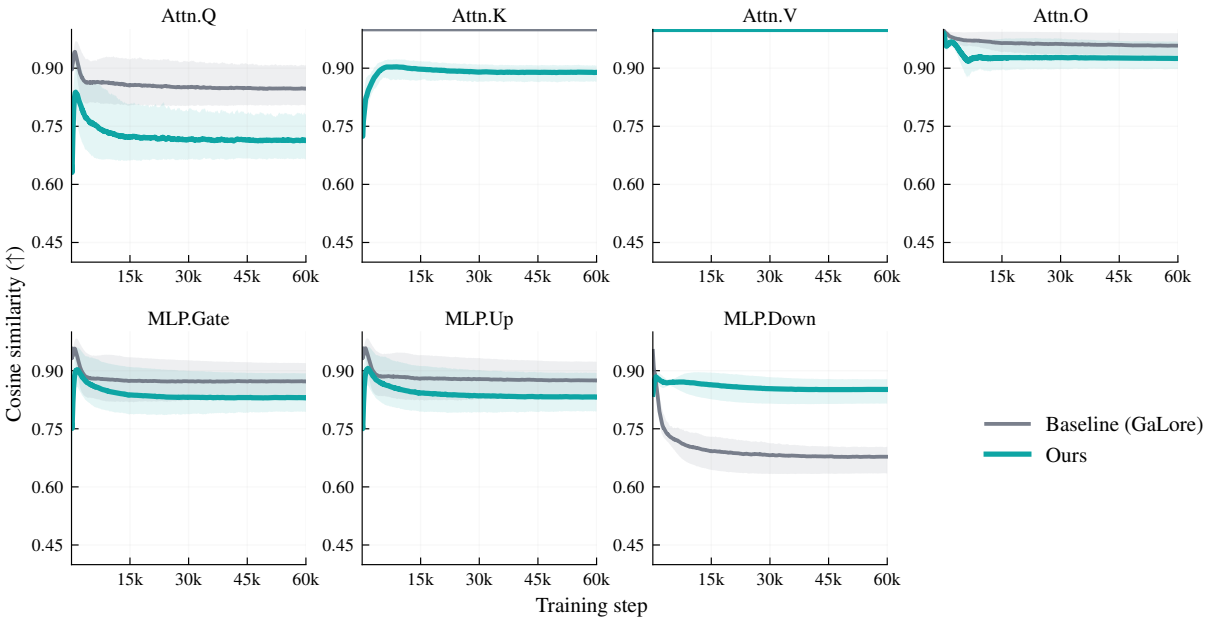

Figure 15: **Cosine similarity trajectories on Qwen3 350M.** The same analysis is applied to Qwen3.

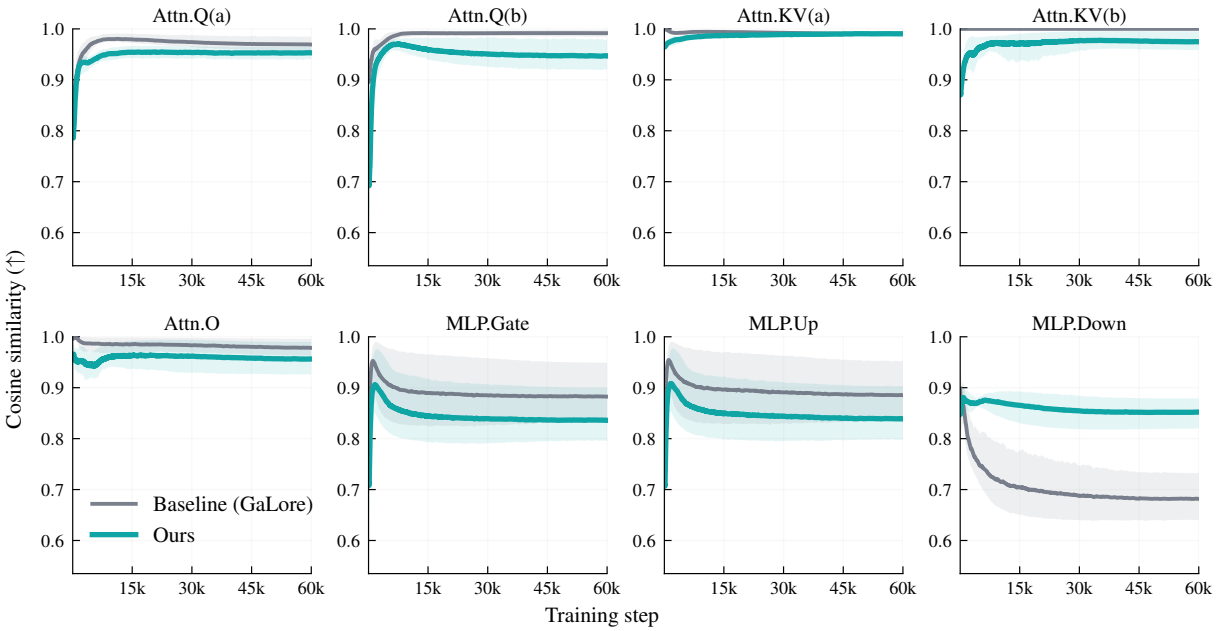

Figure 16: **Cosine similarity trajectories on DeepSeek-V2 350M.** The same analysis is applied to DeepSeek-V2.